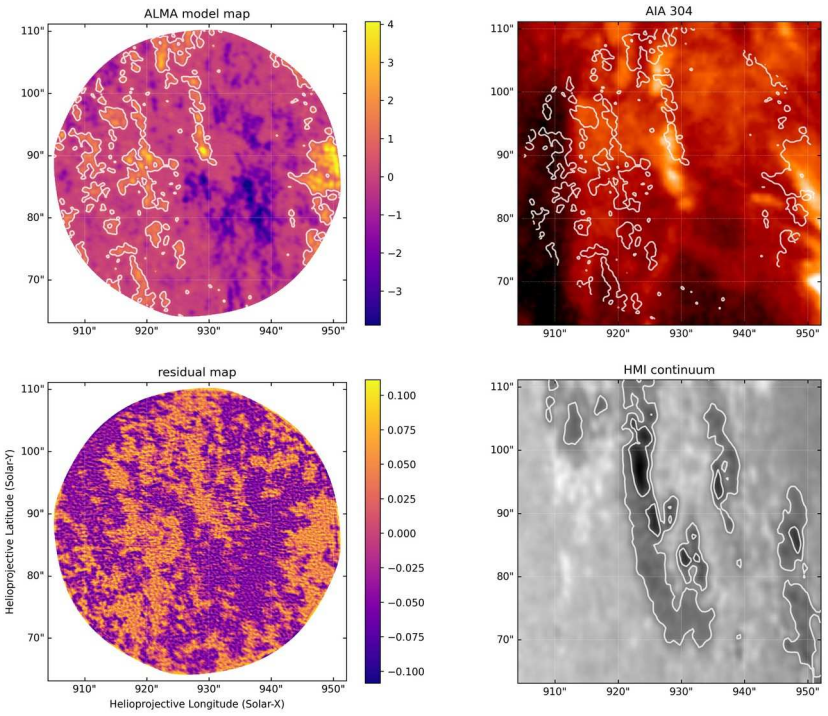
ESO ALMA Development Study

HIGH-RESOLUTION SOLAR OBSERVATIONS WITH ALMA

Final report



Miroslav Bárta & Yi Chai

EU ARC – Czech node

Astronomical Institute of the Academy of Sciences

EUROPEAN ARC

ALMA Regional Centre || Czech

Draft, October 1st, 2026

Table of Contents

1. Introduction....4
   1.1 Conceptual foreword to the *Final report*....5
   1.2 Related documents....6
      Annexes....6
      Other related documents....7
   1.3 List of abbreviations....7
2. Results....9
   2.1 Improved simulations and analysis of the phase-errors introduced by the atmospheric turbulence 9
      2.1.1 New model for the turbulent atmospheric refraction....9
      2.1.2 Additional metrics for assessment of the self-calibration performance....15
      2.1.3 Results of the 2023 test data processing....20
   2.2 Water-vapor signal in the science SPWs in Band 5 applied to phase-delays corrections....23
      2.2.1 Theoretical background....24
      2.2.2 From brightness temperatures to measured powers....26
      2.2.3 Implementation....31
      2.2.4 Correlations between dPWV proxy and the actual phase delays in BP calibrator....32
      2.2.5 Time-detrended correlation between the phase delays and dPWV....34
      2.2.6 Using the dPWV proxy to fixing the phase errors....38
      2.2.7 Comparison of dPWV proxy with WVR measurements....39
      2.2.8 Comparison of different source SPWs....45
      2.2.9 Testing the approach with different dataset with low median PWV....46
      2.2.10 Concluding remarks to this topic....48
   2.3 Science SPW-based PWV corrections for the solar data....49
      2.3.1 Theory....50
      2.3.2 Early results....52
         2.3.2.1 Nature of the source brightness variations....57
         2.3.2.2 Discussion to separation of pressure/humidity or dry/wet components in dPWV variations....59
      2.3.3 From dPWV fluctuations to full-PWV time series: Rigorous approach....62
         2.3.3.1 Starting point....62
         2.3.3.2 Polarization coupling in PWV....63
         2.3.3.3 Making use of the scan-averaged signal....67
            Mathematical background....67
         2.3.3.4 Early results....70
         2.3.3.5 Improving the solar spectrum....74
         2.3.3.6 Validating the linearization approach....75
         2.3.3.7 Introducing the antenna efficiencies....77
         2.3.3.8 Improving antenna gains....78
         2.3.3.9 High- vs. low-attenuation gain scaling: Issue with time dependence....81
         2.3.3.10 Instrumental parameters as a part of solution....85
         2.3.3.11 Adding the ACD calibration equations into the system....90
         2.3.3.12 Finding the optimal *calWeight*....94
         2.3.3.13 Implementation and results....96
            Summary of this part....102

2.4 Science SPW-based PWV(t) vs. direct WVR measurements....102
2.4.1 WVR data in the solar scan of 2022.1.00031.CSV – detailed analysis and comparison with the dPWV proxy....103
2.4.1.1 Theory 1: Differential approach....104
2.4.1.2 Results 1: Differential approach....106
2.4.1.3 Theory 2: Non-linear RT equations for full PWV(t) and L(t)....108
2.4.1.4 Results 2: Full PWV(t) and L(t) series....113
2.4.1.5 Testing/Verification....117
2.4.1.6 Our vs. SYSCAL PWV values....120
2.4.1.7 Further development – methodological convergence to the PWV-proxy calculations....123
2.4.2. WVR vs. sci SPW-based PWV: Specifically designed test observation....123
2.4.2.1 Setup of the observation....124
2.4.2.2 The observations....127
2.4.2.3 Results of the July 2026 observations....128
Comparison of direct and indirect PWV measurements using two concurrent subarrays....128
2.5 Application....134
2.5.1 Calculation of the optical-path dependence on PWV for different frequencies (dispersion)....134
2.5.2 sol_pwvGcal(): The solar counterpart to wvrgcal()....135
2.5.3 Final improvements....140
2.5.3.1 Improved antenna efficiencies....140
2.5.3.2 Making use of array-averaged T_B....141
2.5.3.3 Correct weights for the grand-system of equations....141
2.5.3.4 Combined instrumental and atmospheric delays....144
2.5.4 Results of applied PWV-based gain corrections....146
3. Conclusions, suggestions for implementation, and outlook....147
3.1 A brief recap and summary of obtained results....147
3.2 Suggestions for implementation of high-resolution solar ALMA Observing mode....149
3.3. Future directions of further development....151
References....152

1. Introduction

This report summarizes mostly the advances achieved during the 2nd half of the study (see the next Section 1.1), while the motivation, goals, suggested methodological approaches, and the early results have been already published in the *Mid-term-report* of the project. For the sake of just quick introduction, let us thus start with brief summary of the project aims and suggested ways how to reach them.

The main goal of the study is to investigate possibilities of extending ALMA array configurations for solar observations in order to achieve the higher spatial resolution, what is highly demanded by the community. The main obstacle to utilization of more extended configurations in solar mode has been identified in uncompensated fast phase errors during the target on-source observation between two phase-calibrator scans (there is no WVR pre-correction possible in solar mode). In order to overcome this issue, we study two approaches suggested in the project proposal:

1. Time-scale separation between the actual dynamics in the solar atmosphere and apparent time variations in the image caused by fast phase errors related to the turbulent terrestrial atmosphere. The goal is reached by self-calibration of the high-cadence series of solar interferometric images with hierarchical consecutive diminishing of the gain-cal solution interval. The high cadence (extra short integrations) is essential (i) for keeping coherence, and (ii), for sampling the phase variations on the time-scale much shorter than the actual dynamic timescale of studied phenomena at the Sun, so that we can separate both time-scales. In further text it is referred as *Approach 1*.
2. Using water-vapor signatures and their fast time variations in the spectra of science SPWs close to the water line in the Band 5 – this represents an alternative to the WVR measurement (which are impossible in solar observations) and subsequent wvrgcal() calibration. We derived a proxy to the variations of the water-vapor column based on science SPW data (auto-correlations and SQLDs), which is subsequently recalculated to the phase delays, and eventually applied for fast phase-error corrections (*Approach 2* in the following text)

The ultimate goal of the study is to provide a new suggested table of allowed array configurations for solar observations, an update to that, which is currently in charge in the ALMA technical handbook for Cycle 12, at page 123. This final result is presented in the concluding section Error: Reference source not found

The new array-extension limits for each ALMA observing band are being found by:

- Simulations. We create simulated visibilities for various array configurations from the high-resolution and large field-of-view image (broad scale coverage), perturb these ideal visibilities by instrumental and – above all – atmosphere-related fast-varying errors, apply the self-cal procedure as like for actual observations according to *Approach 1*, and eventually compare the re-constructed images with the original model.

- In Band 5, the pre-correction similar to **`wvrgcal()`** according to *Approach 2* is applied before the calibration and self-calibration.
- In Bands 3 and 5 we have even test-observation data, whose analysis has verified outcomes of the simulation/self-calibration in those Bands and can be considered as a gauge to the entire procedure.
- For the ALMA observing bands, which are not yet commissioned for the solar observations (e.g., B1, B2, or B10), we provide at least estimation based on the scaling (the maximum baseline expressed in units of wavelengths) for the maximum acceptable extension of possible array configuration for solar observations.

We refer kind reader to find the details of scientific motivation and methodological approaches in the project *Mid-term report* (including its attachments), appended to this document as *Annex 1*.

1.1 Conceptual foreword to the *Final report*

In line with the contract and its *Statement of Work* (SoW) for this project, the *Final report* represents a key deliverable for the 2nd stage of the development study (DS2). Having this in mind, we have concentrated namely on the results achieved during the second half of the project when writing this document, avoiding as much as possible repeating presentation of the results reached in its 1st half. These are summarized in the *Mid-term report* (MTR), which has already passed its review in July 2024. The MTR is attached to this document and we frequently refer to it. We believe that this arrangement is more practical for reviewing the work done during the project and for clear assessment, whether the goals of the study has been achieved and results delivered. For the same purpose, we clearly refer to identificators (DS*x.x*) of the deliverables as they are defined in the SoW, and action items (AIs) in *Recommendations for the 2nd half of the study* as they resulted form the mid-term review in the text of the current document.

On the other hand, we are highly internally motivated to see the results of our study being actually applied for enhancement of ALMA capabilities in the direction of improved resolution of solar ALMA imaging. For that sake, we are strongly committed to prepare a merged document, containing all important results in a single, more smoothly flowing text (e.g., with a bit different organization, no references to internal milestones of the study, etc.). This is going to be presented, after incorporation of recommendations from the Final review, as the deliverable DS3. Our clear intention is to formulate it as a reference material for the *ALMA ObsMode Committee* and the basis for its future decision about implementation of upgraded solar ALMA observing mode enhanced by high-resolution interferometry. A concise version of it will be highly likely also submitted for publication, as the results of this study may be interesting for the (not only solar) ALMA community, whether the enhanced solar ObsMode is eventually commissioned and implemented or not.

As already explained, we concentrate namely at topics not covered by the first half of the study and resulting MTR which are namely:

- Water-vapor line in Band 5, its emission and absorption features as captured in the *science* SPWs, and their application to the fast phase-error corrections as an alternative to WVRs and wvrgcal(), viable also for the solar observations. Insufficient elaboration of this topic has been actually the most serious concern of the expert panel during the mid-term review, so we have paid specific attention to it.
- Improved simulations of the atmospheric turbulence by the new *phase-screen* model.
- Presentation of the fully (self)calibrated test observations in Bands 3 and 5. As this is CPU-time demanding task, it was not possible to show the full-scale results already in the MTR.
- Improved quantitative metrics for the reconstructed-image quality plus new qualitative criteria for the credibility of the self-calibration loop.
- Summary of all the results, conclusion, and implementation recommendations.

In addition to the new tasks and deliverables specific for the 2nd stage of the study, there are also recommendations of the panel from the mid-term review for improvement of the results of its 1st half, formulated as a set of Action items (AI), appended as *Annex 2*. We wish to keep the flow of the text as smooth as possible and for that sake the special section (Section 3) is appended, which briefly describes resolution of each AI alongside with reference to the previous text, where it is handled in more detail.

Many results of the study, namely the full data products of the test observations in the form of self-calibrated high-cadence / high-resolution movies, are inherently of digital nature (and large in data extent) and in this report they will be presented by a link (URL) to the data repository directly in the text. For the limited space of the document, we also did not plot, e.g.,, the analytical results for all antennas and have selected just illustrative samples to be presented here. The rest is available on-line in the project repository http://wave.asu.cas.cz/shared/ESO/final_report.

The remaining relevant documents like Mid-term report (including its attachments), documentation to the SW developed in frame of the study, code listings, etc., are attached as annexes.

In spite the study should be concluded, and we believe that its goals have been achieved and clearly presented as deliverable documents, answers to the original questions – as usually in research – rise a bunch of the new ones. The very last Section 4 is therefore devoted to a brief indication of the future development in this direction. We are motivated and committed to carry out this research program even after the formal conclusion of the study, supported by our internal grants and institutional resources in order to improve the solar observing mode yet further.

1.2 Related documents

Annexes

- Complete *Mid-term report* after the last (accepted) revision – attached as *Annex 1*.
- Panel *Recommendations for the 2nd half of the study* – attached as *Annex 2*.

- Documentation for `sol_wvrgcal()` Python module – *Annex 3*.
- Listings of the scripts for simulation, calibration, imaging, and data analysis – *Annex 4*.
- On-line data & SW repository with results of the Study: Movies, datasets, XLS tables, etc.: http://wave.asu.cas.cz/shared/ESO/final_report/ – *online Annex 5*.
- *Directional phasor/Circular statistics* method for self-calibration quality assessment – *Annex 6*.

Other related documents

- General parent JIRA ticket for the 2023 test: https://jira.alma.cl/browse/CSV-3831
- Child-thread for the 2023 CSV observations: https://jira.alma.cl/browse/CSV-3832
- General parent JIRA ticket for the 2026 test: https://jira.alma.cl/browse/CSV-4017
- Child-thread for the 2026 CSV observations: https://jira.alma.cl/browse/CSV-4018

Further resources are available through the references/citations or as the functional hypertext links inserted directly in the text.

1.3 List of abbreviations

The standard and notorious abbreviations, used in the ALMA documents, are exploited also in this report without specific definition. Here we define just these less usual or ambiguous. A few others, which are not used too much repeatedly in the report, are defined in the appropriate place in the text.

AC	Auto-Correlations.
AI	Action Item, exceptionally also Artificial Intelligence
ASU	Astronomical Institute (of the Czech Academy of Sciences)
BP	Band-pass (calibrator, scan)
Cy *N*	(ALMA observing) Cycle no. *N*
EB	Execution block
INT	Interferometry/interferometric
FoV	Field of view
FR	Final report / Final review

MTR	Mid-term report / Mid-term review
PB	Primary beam
PM/PR	Progress meeting / progress report
PWV	Precipitable water vapor (a measure of, in mm)
QA2	Quality Assurance Level 2
(R)MHD	(Radiative) Magneto-Hydro Dynamics
SOT	Solar Optical Telescope (onboard Hinode satellite)
SoW	Statement of Work
SP	Single-pointing
SQLD	Square-Law Detector
SSG	Solar Script Generator
WV/WVR	(Earth atmospheric) water vapor / Water-vapor radiometer.

2. Results

We follow the chronological line of the project development and start with the incremental improvement to the results reached during the 1st part of the project and described in the MTR (*Annex 1*) in the Section 2.1. The next Section 2.2 handles the project conclusions, made based on those results of the particular tasks, generalization, and recommendation for the implementations of the enhanced Solar ALMA ObsMode.

2.1 Improved simulations and analysis of the phase-errors introduced by the atmospheric turbulence

The results presented here are related to Sections 4.1 – 4.3 of the MTR (deliverables DS1.1.1, DS1.1.2, and DS1.1.3) and represent significant improvement and extension of the preliminary outcomes reached in the 1st half of study. Most notably, the new, realistic model for the inclusion of the atmospheric turbulence increases credibility of our simulation results, what is essential for achieving the ultimate goal of the study – the table of maximum allowed configurations for solar observations presented in the Section Error: Reference source not found. In addition to improved simulations, also the analysis of the images re-constructed from the simulated visibilities is enhanced in order to quantitatively asses the ability of the self-calibration loop in removing the fast phase errors via fidelity between input-model and reconstructed images.

2.1.1 New model for the turbulent atmospheric refraction

Already the 1st half of the study, summarized in the MTR, has brought significant improvements in this direction to the earlier attempts in this topic, namely when compared to the previous solar development study (Wedemeyer et al., 2020). The features advancing the older approaches have been namely:

1. Using of flexible simulator tool instead of **`simalma()`** task for generating visibilities.
2. Having full ALMA heterogeneous array, including 7m-12m mixed baselines to capture the actual setup for solar observations,
3. Using high-resolution and, at the same moment, broad field-of-view image as an input model in order to reach broad span of the Fourier components in the input-model data.
4. Modeling the atmospheric turbulence via **`ATMOS.py`** (Ed Fomalont 2021) instead of CASA simulator tool and its methods **`sm.settrop()`** and **`sm.setnoise()`**, which are considered inappropriate for this task (L. Maud, personal communication).

Details of the simulation of visibilities, their perturbation by the atmosphere-generated errors via ATMOS, image reconstruction by means of the self-calibration loop suggested as the *Approach 1* for fast phase error corrections, and measuring the performance of the self-cal correction via calculating imaging fidelity between the input ideal model and resulting reconstructed image can be found in the MTR Section 4.1.

Nevertheless, in order to improve credibility of our simulations and asses the influence of the specific model of the turbulent atmospheric refraction to the entire simulation results, we have switched to the new, more realistic model for varying phase delays caused by the terrestrial atmosphere. The new model commonly known as *phase-screen* approach, is advanced namely in expressing influence of the

larger-scale disturbations. It has been kindly provided by Luke Maud and Yoshi Asaki (Maud et al., 2019) together with introduction, how to use it. The correct simulation of large-scale disturbances is an important feature of the model, as in the Band 3 we aim at observations in the array configuration C-7 with baselines longer than 3 km and the prospective estimation based on scaling of the density of visibilities in the *uv* space shows (see concluding Section 4) that it should be possible to use yet longer baselines in the (not yet commissioned for solar observations) Bands 1 and 2.

We use the same input model (high-resolution solar image from Hinode/SOT), simulation of the ideal (or just instrumentally-corrupted) visibilities using the **`sm`** CASA tool for heterogeneous 7m+12m array, and the same procedure (self-calibration loop) for subsequent interferometric image reconstruction as in Section 4.1 of the MTR. Only difference is in application of the atmospheric turbulence, now we use the phase-screen model by Maud et al. (2019).

First, because in our simulations we focus on the self-calibration, there is no phase-reference source (calibrator) in the model. In order to reflect it, the code for the phase-screen model has been modified in the following way:

```
#use assignPhase to generate phase screen
#parameters
res = 1
integration = 2 #sec
wspeed = 10
scaleh = 1000
maxBl = 4000
targetid = '0'
#phaseid = '0'
targetrms = 30
assignOutput = assignPhase(caltab=cal_table,caltabout=cal_table_out,
                           res=res,integration=integration,
                           wspeed=wspeed,scaleh=scaleh,maxBl=maxBl,
                           targetid=targetid,phref=False,
                           targetrms=targetrms,applyit=True)
```

We present here results for simulated observation in Band 5 in the array configuration C-6. The observing conditions has been simulated at two levels, characterized by the phase RMS measured over the reference time interval of 1 minute. Figure 1 shows the results for phase RMS 15 deg (over the 1 minute reference interval), Figure 2 for 30 deg. The first value corresponds to rather good observing conditions, the latter represents a typical or rather worse weather (Maud et al., 2017a). The upper panel in both Figs shows the map of ‘clouds’ – i.e., the distribution of the optical path over the antenna array. This distribution is advected over the antennas in the horizontal direction (x-coordinate in the map) with a wind speed that can be selected in the model; here we use 10 m/s. Time resolution is set to 2 secs, which corresponds to ~10 integrations in the actual solar observations (later we switched to the actual duration of solar-data integration of 0.192 s, the results are presented below). Middle left panel shows the phase errors as a function of time expressed as the index of 2-secs integration for each of the 53 antennas in the combined array (12m + 7m). Middle right shows the phase RMS as measured over entire simulated duration of observation, which is a 10 minute science-target scan. It is plotted for each antenna (53 points), the antenna is characterized by its radial distance form the reference antenna selected in the middle of the array. Bottom two panels display simulated atmospheric structure function (SSF) for the given parameters of the phase-screen model. Left panel is for the actual duration of the scan, the right one displays also the parameters for the reference time of 1 minute.

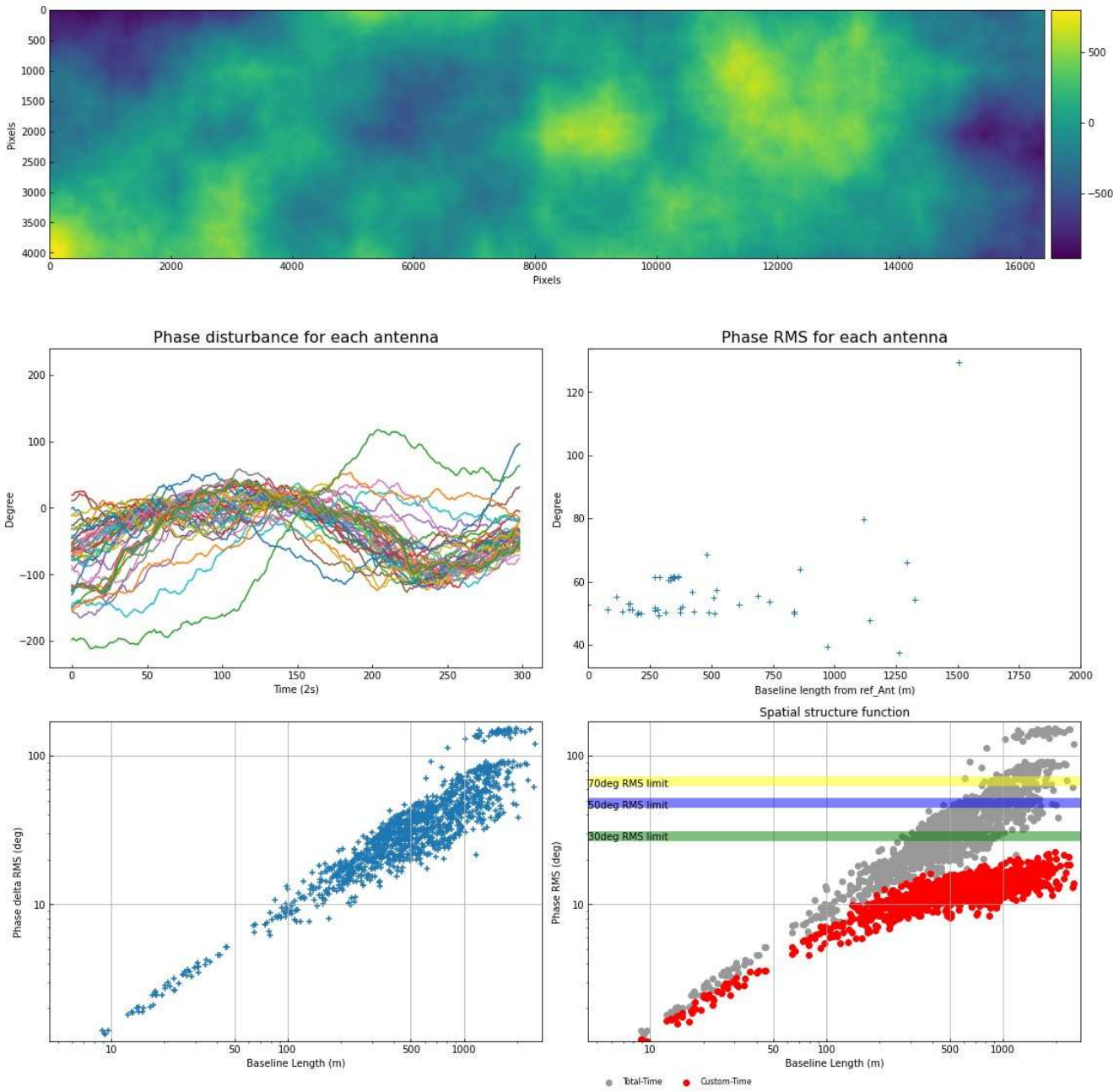


Figure 1*: The phase-screen model for phase errors introduced by a plain advection of the turbulent structures in the terrestrial atmosphere over ALMA array calculated by modified code of Maud et al. (2019) for ALMA observations in Band 5 for the array configuration C-6. The reference phase RMS over the 1-minute interval is 15 deg, the actual duration of the simulated scan is 10 minutes. See text for explanation of the displayed panels.*

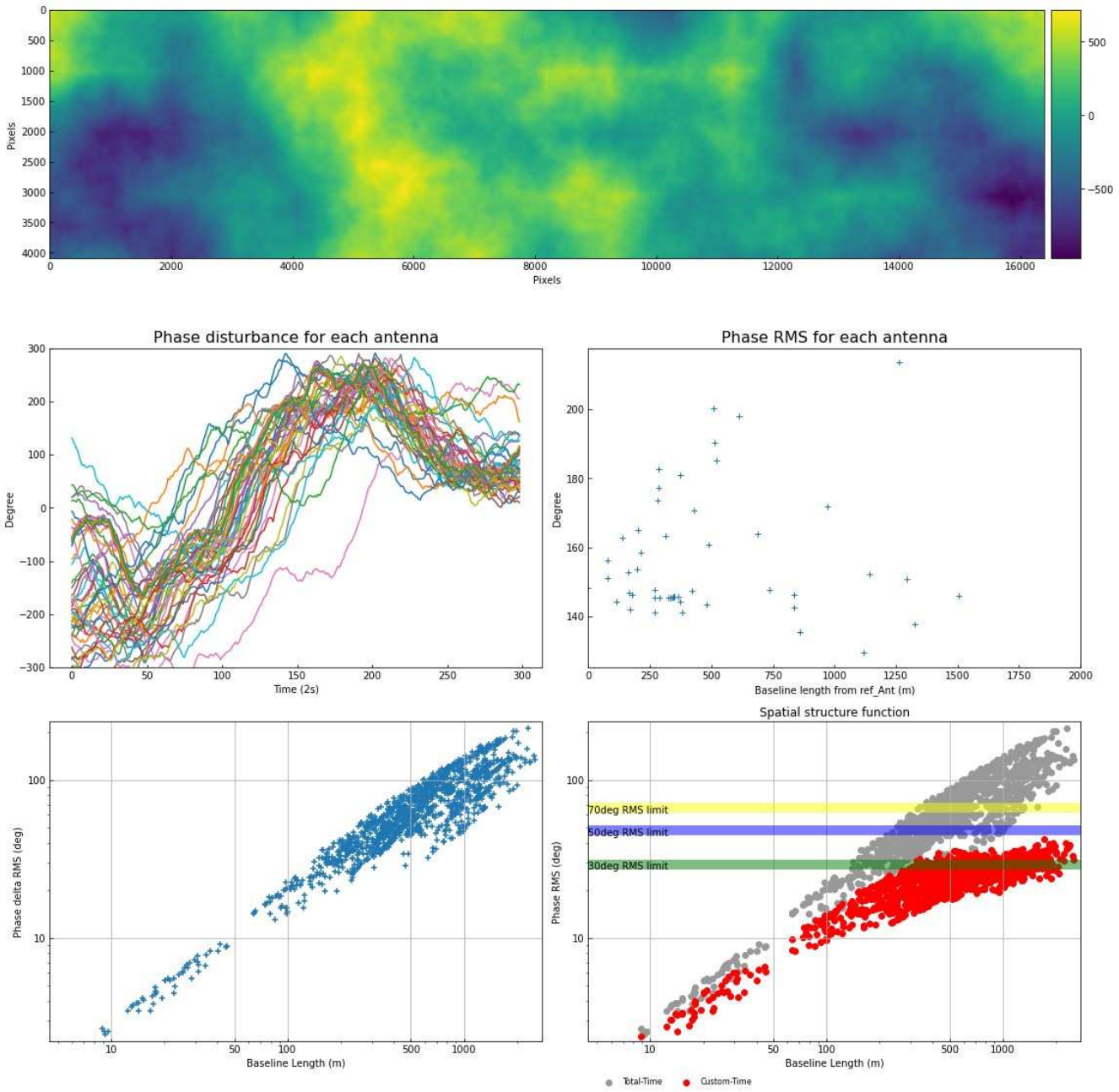


Figure 2*: Like in Figure 1, now for the higher value of the phase RMS over the reference time interval of 1 minute. This correspond to typical / rather worse weather conditions. See details in the text.*

In order to assess the impact of using different, more accurate model for phase errors, rooted in the atmospheric turbulence, to the validity of results found in the 1st half of the study with the ATMOS.py code, we applied the output of the new phase-screen model to perturbation of our simulated visibilities. These have been subsequently used for making the reconstructed interferometric image via the self-calibration procedure with consecutively hierarchically diminishing *solint* interval, described by the algorithm in Section 4.1.3 of the MTR (the full code listing is available in the MTR, too, as its *Annex 4*). Results of this comparison studies are presented in Figs. 3 and 4.

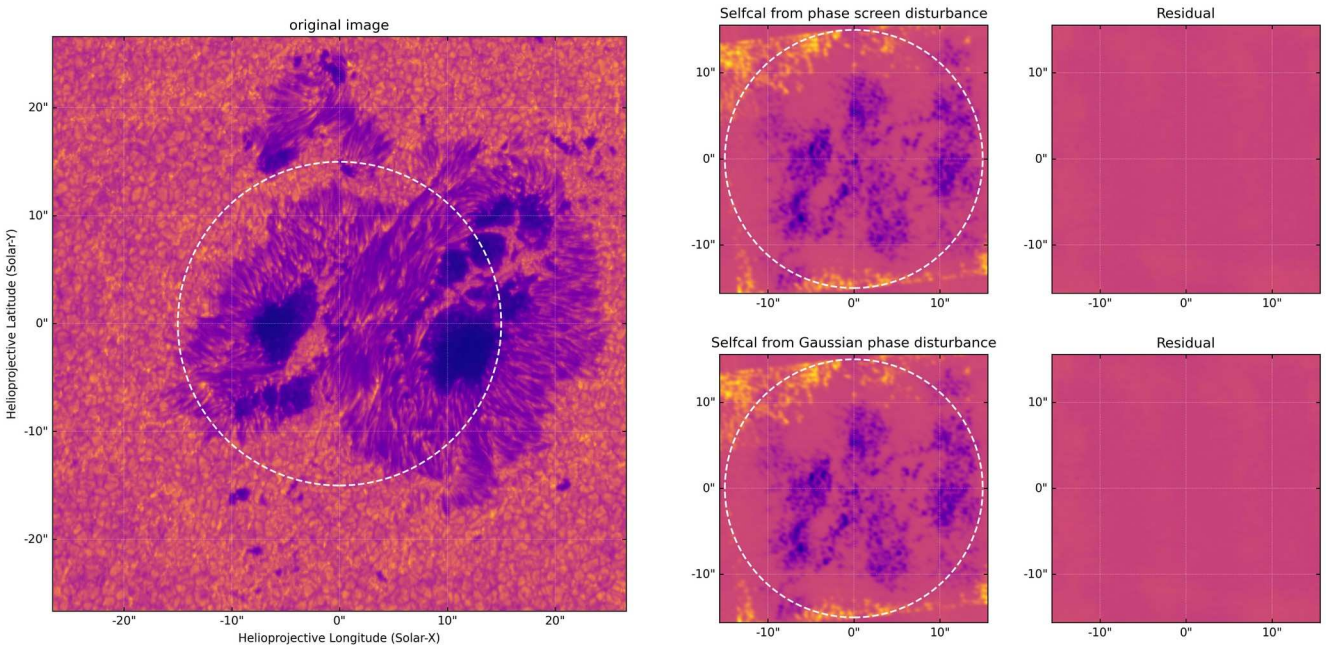


Figure 3: Comparison of images reconstructed by our hierarchical self-cal procedure (Approach 1) from simulated visbilities perturbed by (i) phase-screen model – two upper-right panels, and (ii) Gaussian-distribution scattering of the idealized atmospheric structure function (lower right panels). For both approaches the phase RMS over 1 minute reference interval was 30 deg. The input model image (Hinode/SOT) at the left panel is common for both methods. Simulated observation in Band 5, configuration C-6.

Figure 3 shows simulated observation in Band 5, array configuration C-6 (incl. 7m antennas) based on the same input model (optical solar image from Hinode/SOT) as described in the MTR Section 4.1.1 (the left panel). We then used the ATMOS.py and phase-screen models with the comparable structure function represented by phase RMS of 30 deg over 1 minute interval for perturbing the visibilities obtained by simulation. Both measurement sets has than been processed by the hierarchical self-cal procedure and resulting reconstructed images (and their residuals) compared. Results for the new phase-screen model are shown in upper-right panels, their counterparts for the phase errors with the means corresponding to the idealized atmospheric structure function with Gaussian scattering are shon below them. While the reconstructed images show differences up to 5%, the difference in residuals is negligible.

Figure 4 presents comparison between the two approaches to modeling of the atmospheric perturbations in its time evolution over a simulated scan (10 minutes). We show, how the qualitative performance metrics for the self-calibration procedure, defined in the MTR Section 4.2.4, vary with time. Apparently, for the more realistic model of the phase perturbations based on the phase-screen approach, our procedure reaches even better results, with much less frame-to-frame jumps in the image, that are present because of not being captured and fixed by the hierarchical self-calibration. Most likely the reason for that result is in the inappropriateness of the Gaussian perturbation method for the super-fast (sub-second) variations of the phase errors: The random choice of the phase error in the next integration is not in any way correlated to the phase delay (or optical path) just a little moment ago. In the real atmospheric flow, which is continuous, this is actually not possible. Hence, the ATMOS.py

model represents rather upper limit of the negative impact of the atmospheric turbulence to the phase delays.

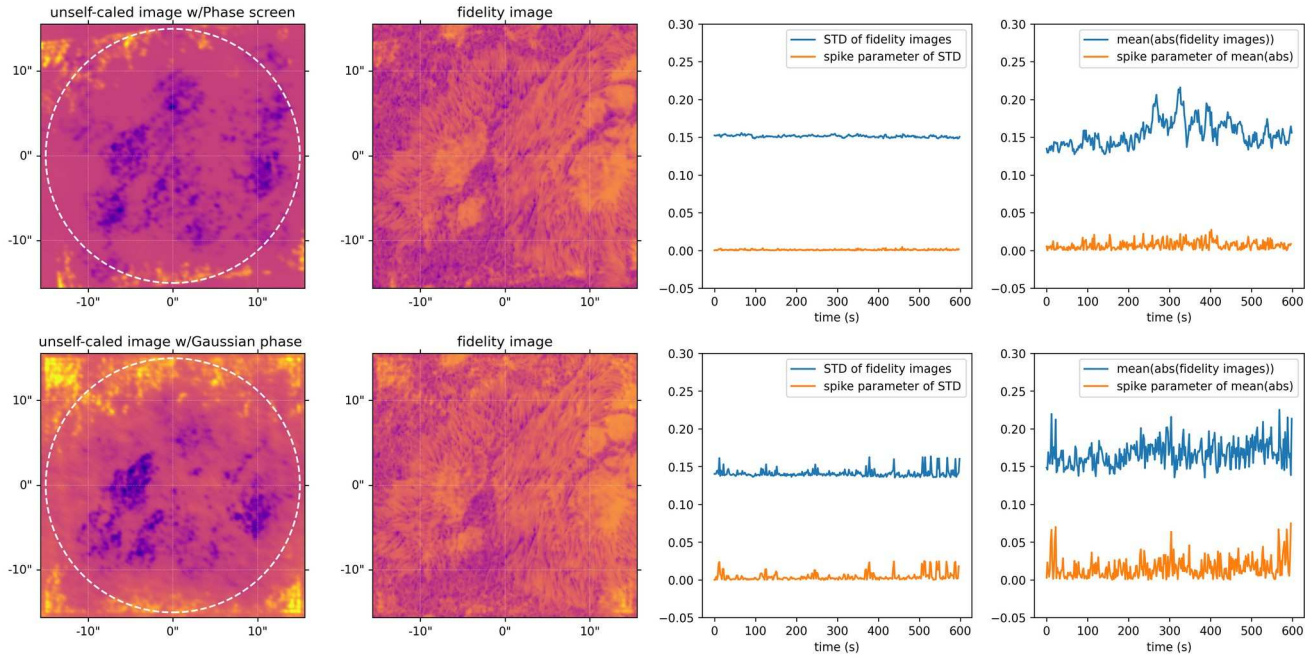


Figure 4: Comparison of quantitative metrics for the performance of the hierarchical self-calibration loop for the phase variations based on (i) the phase-screen model – upper row, and (ii) Gaussian perturbations (lower row). Presented are standard deviation of the fidelity images and the frame-to frame sudden changes expressed as the spike-parameter (as defined in MTR Section 4.4.2).

The results in Figs. 3 and 4 are available also as the full-time cadence movies in the project data repository http://wave.asu.cas.cz/shared/ESO/final_report/phase_screen/. There are two files, one for the ATMOS-based and one for the phase-screen simulations. In this case, the time-cadence used for the modeling of the atmosphere has been set in line with our observation-data integration time to 0.192 s. In addition to the performance metrics displayed in the figures above, the comparison between series of the raw frames (without self-calibration) and self-calibrated is presented. Once can clearly see that the phase-screen provides more realistic model for the image distortions: The raw (non-selfcal) movie exhibits clear drifts in position (shaking) as there is a tilt in the planar wavefront as the first term of the phase error. Similar features can be seen in non-corrected time series of images obtained form the real observational data. Contrary, the Gaussian perturbation model shows rather local deformations in the image accompanied by (apparent) localized changes in brightness.

Let us conclude this topic with the following main findings:

- The new phase-screen model for atmosphere-generated variations of the phase delay represents definitely more realistic approach to the ATMOS.py code (which is based on Gaussian perturbations with the means corresponding to the ideal structure function) that has been used in the 1st part of the study. The non-corrected (without the self-cal being applied) simulated interferometric movies exhibit similar features like these produced from the actual observations – namely wave-front tilt, and, vice versa, less artifacts like sudden jumps in phases, which we

can see in the Gaussian-perturbation model, which has essentially zero coherence time between two consecutive modeled phase perturbations.

- Albeit the phase-screen model is built on different backgrounds, the validity of the results found in the 1st half of the study using the ATMOS.py code remains untouched – difference in the resulting reconstructed interferometric images between both models is less than 5%. The array configuration limits found for each ALMA observing band via earlier ATMOS-based simulations are preserved in the sims involving the phase-screen method, too.
- With either method it was found that for non-extreme weather conditions (phase RMS 30deg over 1 min reference interval) we are able to fix the phase errors by hierarchical self-calibration loop up to the very limit of the maximum baseline length: If we use yet more extended configuration, we are not capable to produce image of reasonable quality even in the case when atmosphere is 'switched off'. In other words, the actual limits for the array extension are given by the too much diluted *uv*-space coverage, which current CLEAN-based imaging algorithms are not capable to handle (see Fig. 6 in the MTR).
- As it can be seen in Section 2.4.2.1 Setup of the observation, rather that the regular atmosphere-generated phase perturbations or their specific model there can be another limiting factor for the maximum usable array extension – the number of available antennas. This is apparently related to the issue described right above – the low density of points in the *uv* space.

2.1.2 Additional metrics for assessment of the self-calibration performance

A question arises up, how credible is the procedure of hierarchical self-calibration for image reconstruction, i.e., how far is the high-cadence time sequence of the reconstructed interferometric images from its idealized counterpart, which we would have seen if we had the perfectly resolving telescope and no interleaving medium (namely the terrestrial atmosphere). In case of simulated images, the natural choice is the image fidelity *IMF*, defined as (Pety et al, 2001, ALMA Memo Series 398):

$$IMF = \frac{\left|\left(M_{ideal}\right)\right|}{\left|\left(M_{ideal} - I_{rec}\right)\right|},$$

where M_{ideal} is idealized input model (Hinode/SOT high-resolution solar image in our case, with possible re-gridding applied – see Section 4.1.1 in the MTR), and I_{rec} is one reconstructed image in the time series obtained by our self-calibration procedure. Hence, for the time sequence of images we obtain the IMF(t) also as the time series (see, e.g., Fig. 4, the right panels).

Another useful metric is the power spectrum of the Fourier components contained in the reconstructed image. Again, this could be compared directly with the same quantity for the idealized model and see, whether all scales have been actually reproduced or not, i.e., whether the procedure does not act as undesirable filter introducing the frequency-domain distortions.

We have used these metrics in the 1st part of the study quite intensively (e.g., Figs. 14 – 18, Fig. 22 in the MTR), displaying usually the inverse value of the fidelity 1/IMF, as we are interested rather in searching for the weak points of the procedure manifested by the discrepancy between the original model and the reconstructed image.

In the 3rd revision of the MTR, accepted in July 2024 we have introduced another new metric for the self-cal performance quality, as per one of the Action items in *Recommendation for the 2nd half of the study* (Annex 2). It is the *spike parameter SPP(t)*, again being defined with the time dependence as

$$SPP(t)=\frac{\left|\sum_{ij}\left[I_{rec}(i,j;t+1)-I_{rec}(i,j;t)\right]\right|}{\sum_{ij} I_{rec}(i,j;t)}$$

The spike parameter is measured in high-cadence series of the self-calibrated images only and represents a measure of normalized difference between two neighboring frames *t+1* and *t*, where *i* and *j* have meaning of the pixel coordinates in the image. Ideally, its value should be zero because – in line with the basic idea of *Approach 1* – the frame-to-frame differences at the shortest time-scale should be caused solely by the phase errors and these are supposedly removed by the self-calibration procedure. Hence, any jump between two frames is detected by increased value of SPP. This metric is used, e.g., in Fig. 4 (right panel). It is not as robust as the first two criteria (IMF and Fourier scale analysis, which were actually used already in the previous solar study by Wedemeyer et al., 2020), on the other hand, it is not limited to the case, where we have apriori knowledge about the ideal model and can be thus utilized also in analysis of the image-series reconstructed from the observed data, when this procedure is going to be applied in practice. It can be complemented by the time series of the residual phase-error, the auxiliary metric of the self-calibration quality displayed, e.g., in Figs. 8 – 13 of the MTR.

During the 2nd half of the study we have been working mostly with the test-observation datasets and therefore we have focused on devising such additional metrics of the self-calibration/imaging loop performance that are independent of the apriori ideal model, as we have none in this case. One possibility is a stability of the phase solution over frequency channels. In the correct gain caltable one expects the phase solution to be only weak and smooth function of frequency – c.f., for example, the *phase vs. frequency* diagnostic plots created during manual QA2 for the band-pass calibrator – if the complex gain solution is not a smooth and weak function of the frequency channel, the band-pass calibration is deemed as failed. In the solar self-calibration procedure, which works in multiple – typically five – hierarchical steps with consecutively diminished *solint* down to the level of integration time, we have more ways how to asses the total phase solution over all steps

- Using **`gaincal()`** solution between the MODEL column of the very last self-calibrated MS and the DATA of the very initial, just standard-way calibrated (no self-cal) MS.
- Re-gridding the incremental phase solutions in gain caltables for each hierarchical step of the self-cal loop to the finest time resolution and accumulating the partial results to the total phase solution, and
- The phase delays may be simply assessed as a phase differences between the MODEL column of the final data and DATA column of the initial MS, keeping just the baselines where the first antenna in the pair is the reference antenna.

The last method provides the approximate single-baseline solution for each frequency channel separately and thus can be used for extraction of the *phase vs. channel* dependency. From the per-channel phase solutions we can make an average (here we neglect the small frequency dependence of the phase delay for the same optical path as the SPWs are as narrow as 0.5% of their nominal frequency), having thus just one representative point per integration and SPW (like in normal gain table) and, moreover, the standard deviation – measure of how the phase solutions in the specific time instant (integration) are spread over the frequencies. The per-channel phases has to be unwrap (by

+/-180 deg) before they are subtracted because the wrapping to the standard angle interval occurs in different moments of time for different channels. Results of this procedure are shown in Figs. 5 and 6. Figure 5 shows comparison between the standard gaincal() solution for accumulated (over the five steps of our hierarchical self-cal loop) phases and the phase difference (final MODEL – initial DATA) averaged per channel for antenna DA42 and our Band 5 test observation data (detailed description of the MS is in the MTR). The 1st panel (in the order left to right, top to bottom) represents the initial DATA column, the 2nd final MODEL, the 3rd is their difference averaged over channels. The 4th panel is the standard gaincal() solution over final MODEL and initial DATA columns and the 5th represents the difference between the approaches in panel 3 and panel 4. The last panel is the standard deviation corresponding to averages displayed in panel 3.

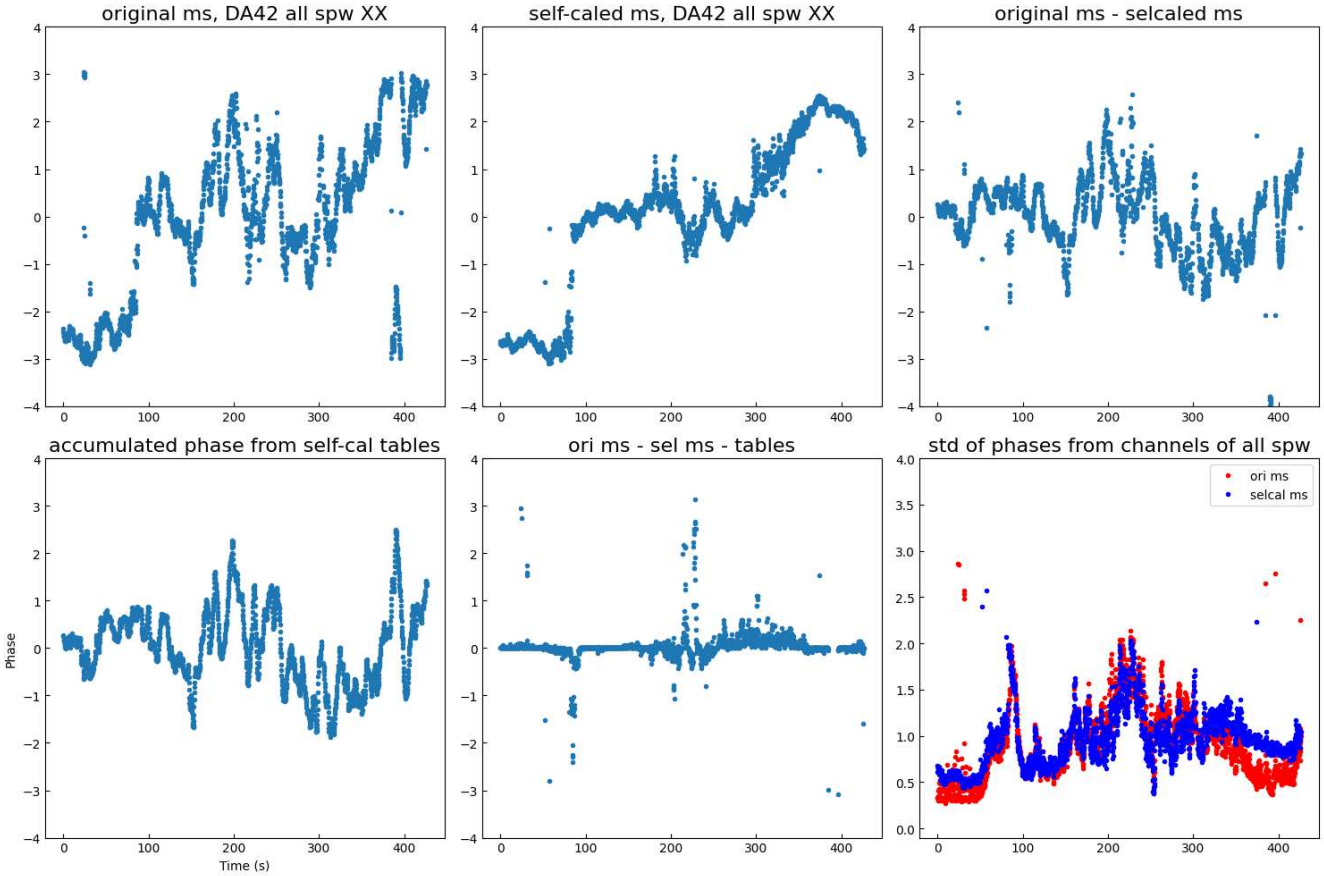


Figure 5: Comparison between phases obtained: (i) by the standard gaincal() solutions accumulated over all five steps of the self-calibration loop, (ii) as averaged per-channel phase differences between original and final self-calibrated MS. See detailed panel description in text.

As it can be seen in panel 5, at most time instants the phases inferred by both methods correspond each other. The little difference reflects more accurate *gaincal()* approach as it takes into account multiple baselines and minimizes the phase-closure error, while the per-channel approach uses just a single-baseline solution (between DA42 and the reference antenna, which is DV11 in our case). The few 'singular' time instants, where the difference is significantly higher, correspond to the integrations, for which the standard deviation displayed in panel 6 has a spike.

High standard deviation of per-channel phases may indicate failed self-calibration loop solution for the given frame, so we propose watching this quantity as one of the QA criteria of the entire hierarchical self-calibration procedure. The method has been further developed into *Directional phasor formulation/Circular statistics* and it is presented as *Annex 6* to this report.

Figure 6 presents closer look at the *phase vs channel* solution – its panels display per-channel phase solutions at selected time samples for different color-coded SPW.

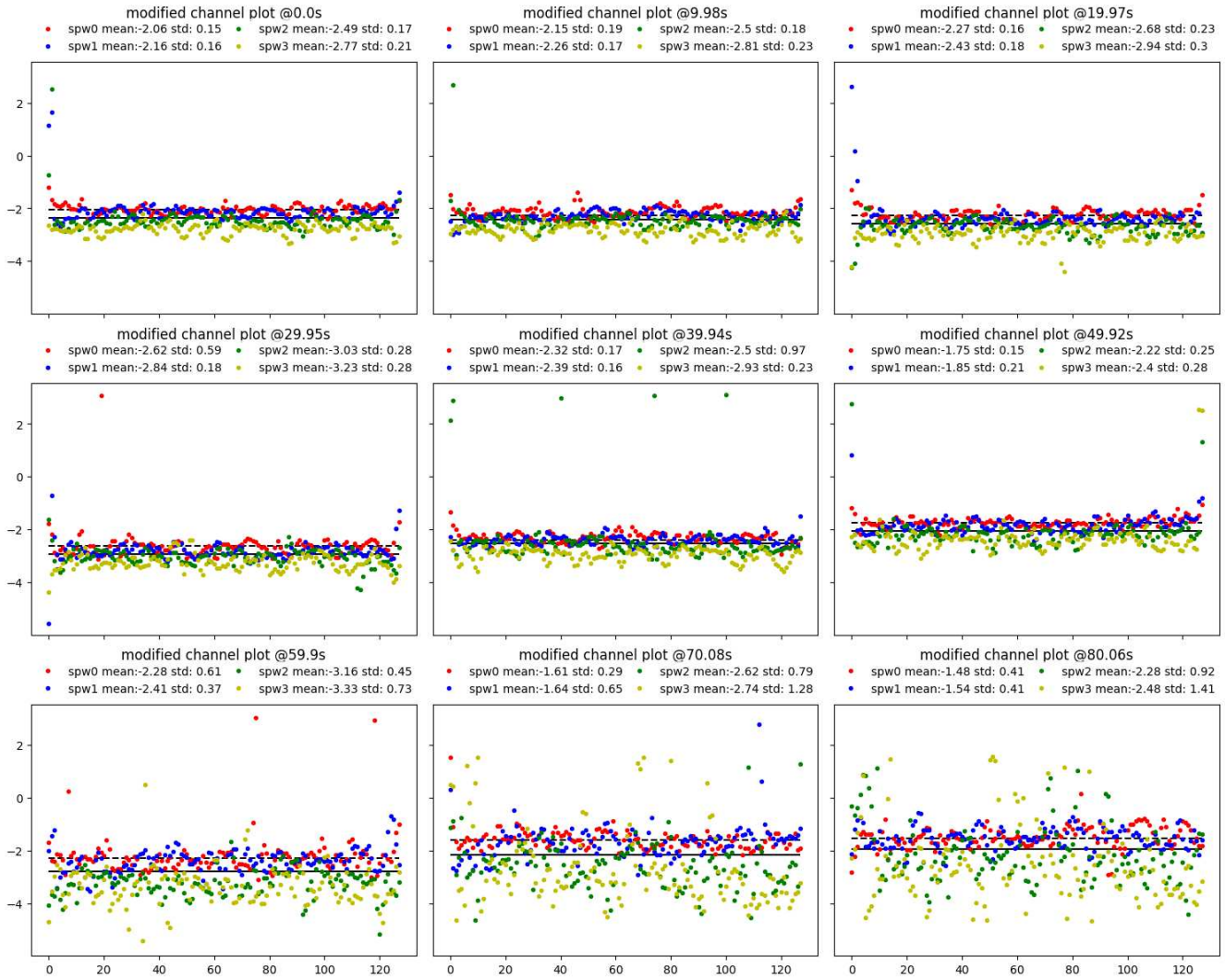


Figure 6: Variations of phase delays vs. frequency channel, the SPW number is color-coded in the legend. The data are for antenna DA42, each frame represents different time sample described in the respective panel title. The standard deviations computed for each time instant are plotted as panel 6 in the previous figure.

Identifying possible issues with self-calibration loop solution for specific time instants, we have turned our attention to another check of the procedure – we wish to make sure that imaging for the particular time instant actually converges. As it has been explained in the MTR, for the high resolution imaging

the scale separation between the *cellsize* and *imgsize* is large even in the case of single-pointing observations (typical image has 2kpx squared). Such a large span of scales in the image requires multi-

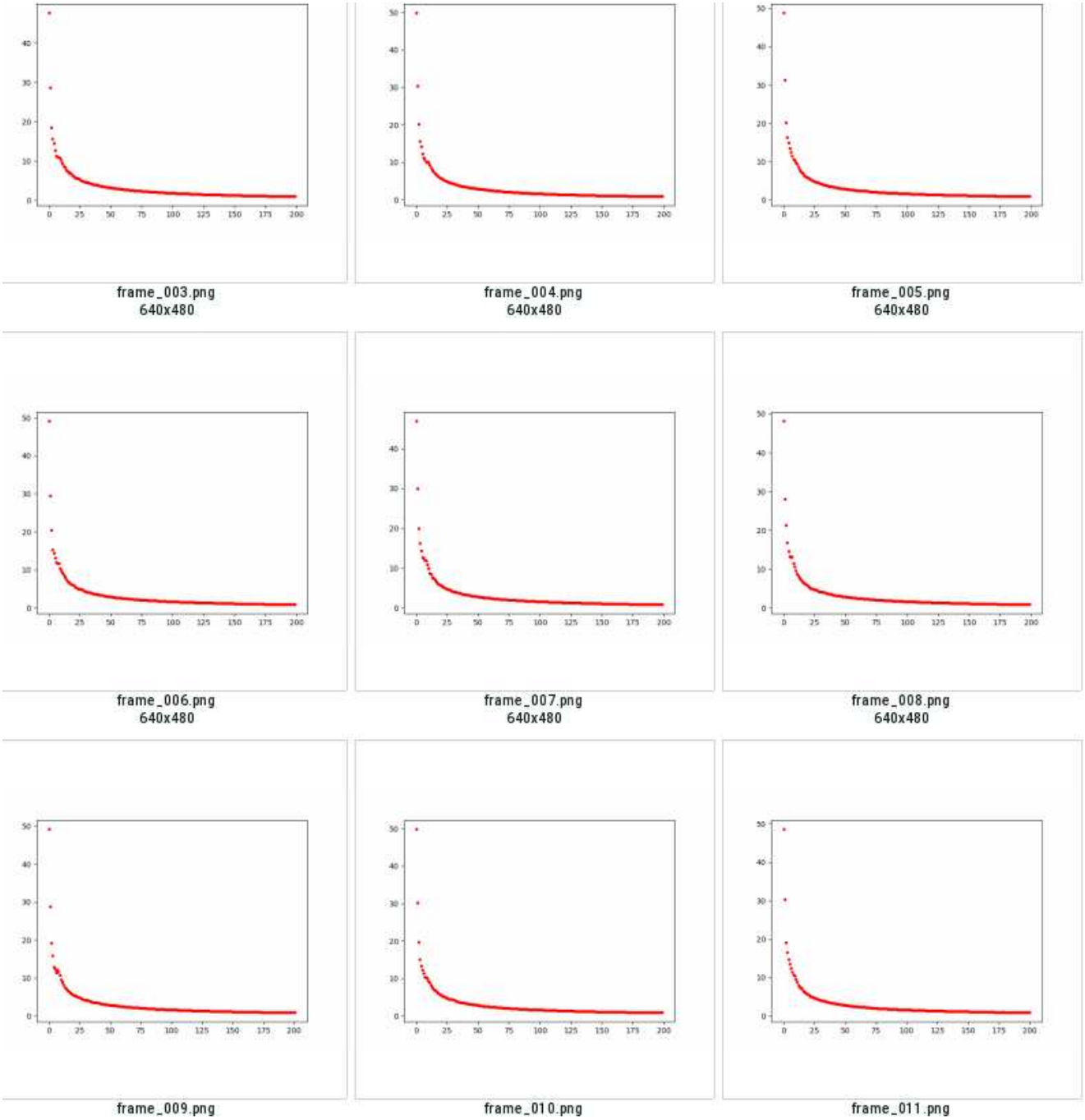


Figure 7: Convergence plots for the multi-scale tclean() used in our self-calibration/imaging loop. For each movie frame, that has been formed from the time-range corresponding to solint interval in gaincal(), the peak residual flux is displayed vs. the major cycle count.

scale cleaning. The multi-scale *tclean()* has, nevertheless, sometimes issues with convergence. Indeed, practical experimentation with the *tclean()* settings has shown, that the maximum range of the scale

vector in *tclean()*, which yields stable solution is **`scales = [0,7,15]`**. Moreover, we have to limit the number of iterations in the minor cycle to 100, and set the gain limited to 0.07. The *timerange* parameters selects the integration(s) used for the particular *solint* time interval in corresponding preceding *gaincal()* call in the self-cal loop:

```
converg = tclean(vis = msc, imagename=imgName, timerange=tRange,
          scales = [0,7,15],cycleniter=100, gain=0.07,…, fullsummary=True)
```

We use *fullsummary* parameter set to 'true' and save the convergence-tracing dictionary in the output variable *converg*. We extract peak-residuals for each major cycle

```
res=converg['summaryminor'][0][0][0]['peakRes']
```

and plot them for each frame of the self-calibrated movie, i.e. for the time of the self-cal solution interval defined by the *timerange* parameter. Small sample of the grid of such convergence plots is shown in Fig. 7. This is another QA diagnostic for the self-calibration/imaging procedure that should be observed during the processing of high-resolution solar data.

To sum up this section – we propose the following QA metrics for measuring performance of processing of the high-resolution solar datasets via *Approach 1*:

- The spike parameter and residual phase errors.
- Standard deviation of the distribution of self-calibration phase solutions over frequency channels.
- Convergence of the multi-scale *tclean()* measured by residuum peak flux.
- Span of the scales in the image assessed by Fourier power spectra.

In case of the simulated MSs, where the 'ground truth', i.e. the ideal model is known, these metrics can be supplemented by the image fidelity and frequency-domain distortion expressed as normalized difference between Fourier power spectra of the model and reconstructed image.

2.1.3 Results of the 2023 test data processing

Because the hierarchical, multi-step self-calibration/imaging loop for the high-resolution (each image is 2kpx square) and high-cadence (~5 frames per second) solar data is very CPU-time demanding, the test data form the end of 2023 has been only partly processed by the time of the Mid-term review. The MTR report thus contains just preliminary results for the Band 3 MS, while the Band 5 data had passed just the standard QA2-like processing by the time of the MTR.

During the 2nd half of the project we have managed to apply our Approach 1 (hierarchical self-cal/imaging loop described in the MTR Section 4) to both datasets. The Band 3 data has been processed up to the level of one image per integration, with the Band 5 MS we have finished with one image per 10 integrations, albeit the *solint* parameter in self-calibration gaincal() has been consecutively diminished down to the integration level, in order to remove the fast phase variations induced by the atmospheric turbulence.

The fully processed datasets are inherently digital and therefore we provide here URLs to accessing the movies in the project data on-line repository

- Band 3 dataset: http://wave.asu.cas.cz/shared/ESO/final_report/test_data/Band3/

- Band 5 dataset: http://wave.asu.cas.cz/sharedESO/final_report/test_data/Band5/

Along with the set of quite large movies (the integration-cadence based movie has 300GB) there are descriptions of each file directly in the same directory of the data repository.

We have performed analysis of the typical dynamic scale in the Band 3 movie by measuring the light-curve at the specific location in the image, followed by frequency decomposition using periodogram and Welch's methods. Results are shown in Fig. 8.

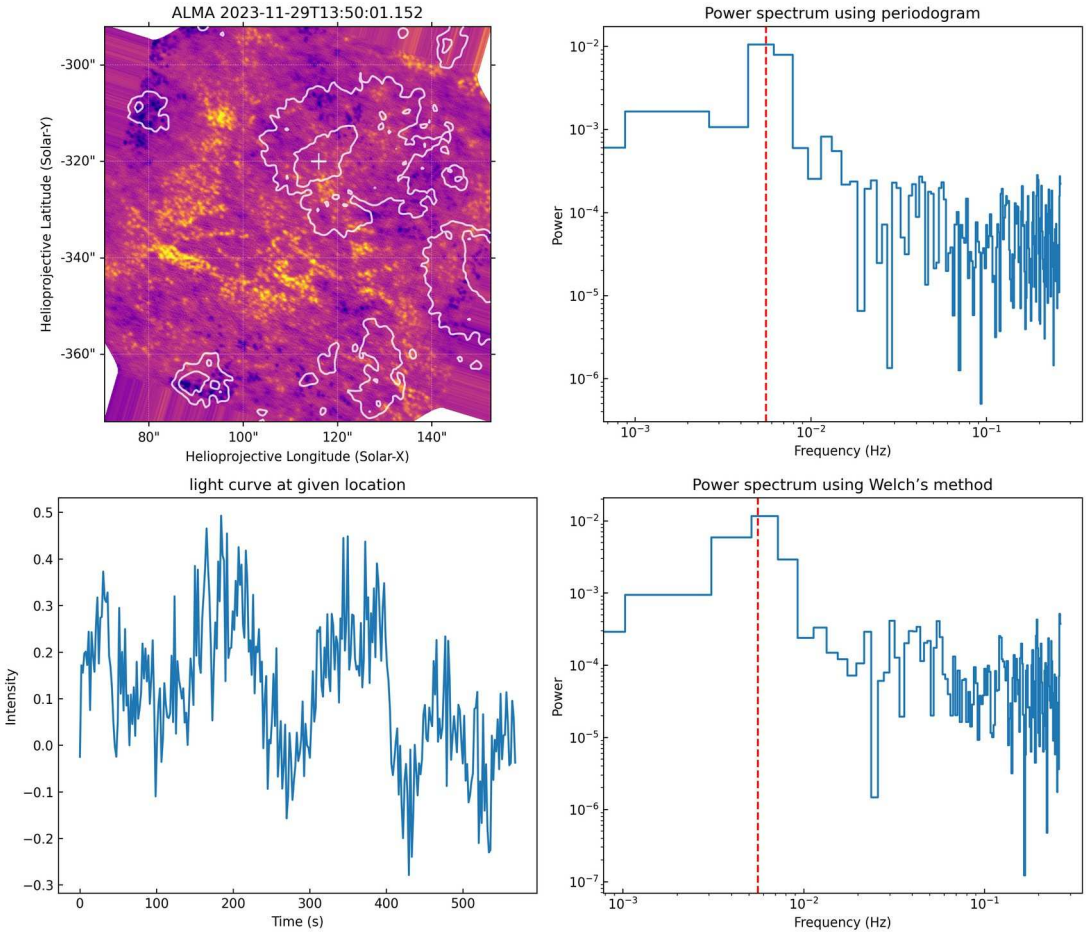


Figure 8: ALMA Band 3 map with indicated point (the white cross) selected for dynamical analysis (top left), the light-curve determined in that point (bottom left), and frequency analysis of the LC constructed via two different methods (right).

Basically, there is no real motion visible on the time-scales less than 2 secs for this specific dataset displaying essentially quiet part of the solar chromophere. This is in line with our estimation based on the signal propagation with the typical sound speed (0.2 arsecs, which is our current resolution, corresponds to 140 km at the Sun, having the sound speed in the 10000 K warm chromosphere about 10 km/s and the Mach number in the convective-generated shocks ~3, the typical transit time is ~5 secs – see detail in the MTR). However, we would like to point out that this need not to be true in general – the active regions at the Sun, most notably the emission kernels in the solar flares, may show faster changes as their propagation – in case of the flare kernels – is determined by the apparent motion of the

anchor of the field line currently reconnecting much higher up in the solar corona. Nevertheless, for the quiet chromosphere it justifies decreased cadence of imaging, still keeping the full time-resolution for *gaincal()/applycal()*, which we have utilized in the Band 5 dataset.

As a sample of the fully-processed datasets we present here just two snapshots in Figs. 9 and 10.

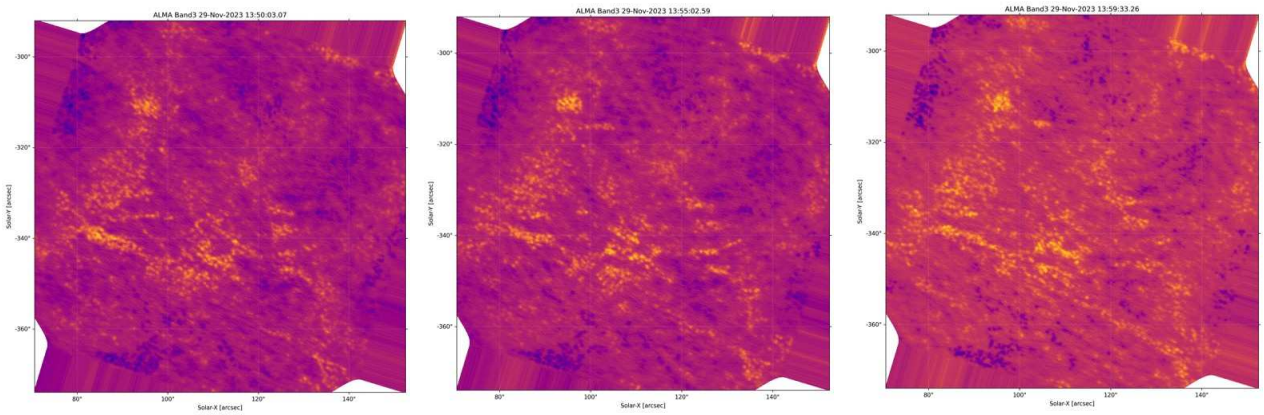


Figure 9: Three frames of the full movie of the Band 3 test data, taken at 13:50, 13:55, and 13:59 UT, consecutively from left to right. Time evolution at the very small, in ALMA images never before resolved scales, is obvious.

Figure 9 shows the ALMA Band 3 interferometric maps in three consecutive times. The presence of very small scale structures, never seen in the ALMA images before, is apparent, together with their dynamics.

Fig. 10 displays one snapshot of the Band 5 ALMA interefometric map (and its residuum) accompanied by the AIA 304Å image and HMI continuum image. The AIA wavelength is sensitive to the solar chromosphere, in the Band 5 we are lookin a bit deeper. The HMI optical continuum shows the underlying photospheric structure and serves as the context image. The AIA image is overlaid with the ALMA brightness contours in order to see correspondence between the bright emission features in both images.

As a brief summary of this part we can say

- Our previous simulations identified the configuration C-7 and C-6 as upper limits for solar observations in Band 3 and Band 5, respectively. The test observations confirmed this statement as we are able to produce high-resolution solar images, which pass the QA criteria defined in Section 2.1.2 Additional metrics for assessment of the self-calibration performance in both observing bands. To that extent, the test observations provide us with a ‘gauge’ to the more general simulation approach.
- The Band 5 data has been actually observed in quite worse weather conditions (see the MTR for detailed observation description and the phase plots in the next section) than we used for simulations, yet we were able to reconstruct the images solely using *Approach 1* (the hierarchical self-calibration loop).
- For quiet solar chromosphere the decreased-cadence imaging is acceptable, yet using the full cadence data for self-calibration. This may not be true for the active regions, specifically flares.

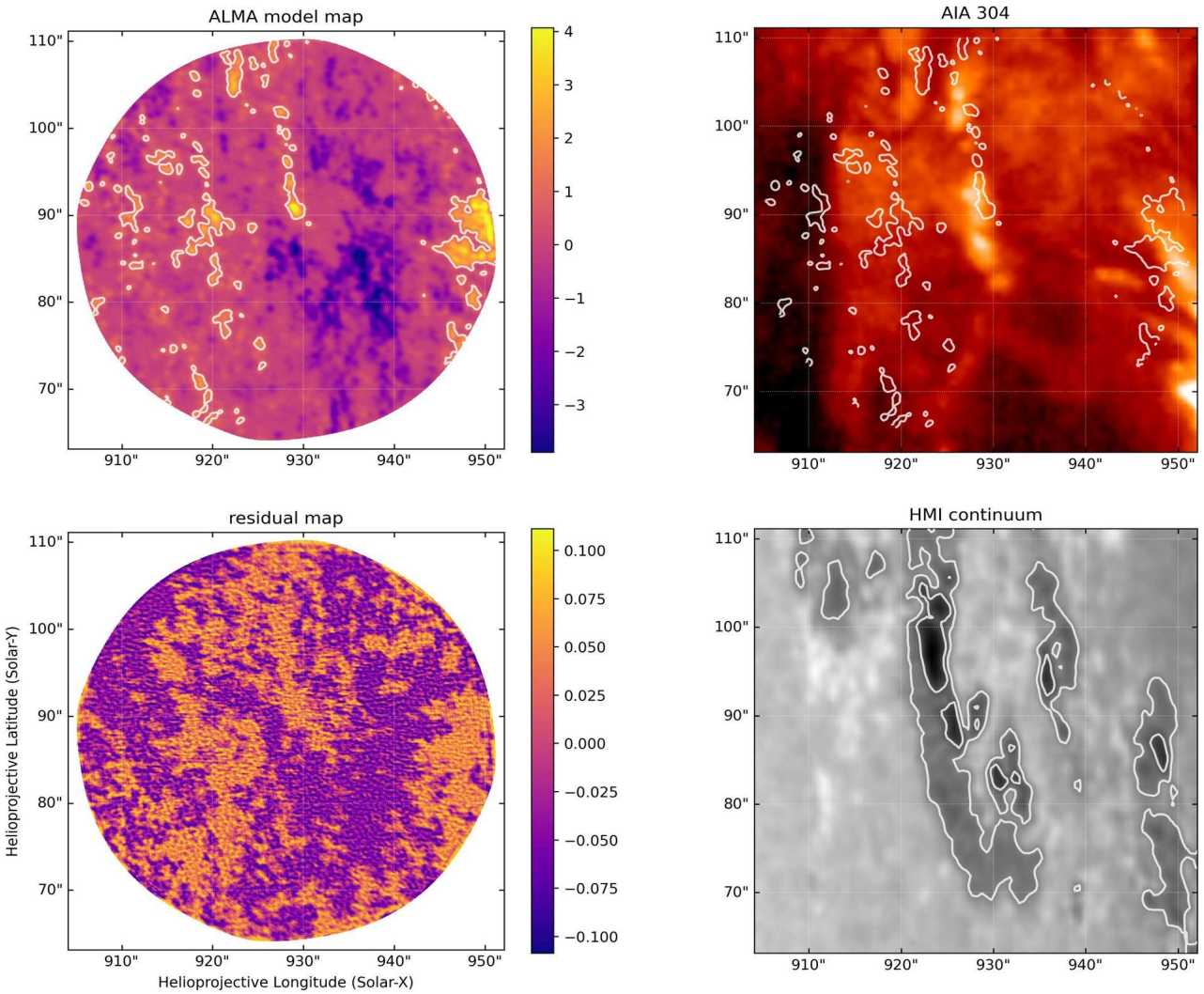


Figure 10: ALMA Band 5 interferometry map (top left), its corresponding residuum (bottom left - note different color scale in this case), and comparison of structures seen in co-observation by SDO/AIA (top right) and HMI. ALMA brightness contours are overlaid over AIA raster map for comparison of locations of the bright features in both images. The small displacement can be explained by a parallax (different points of observations for ALMA and SDO).

2.2 Water-vapor signal in the science SPWs in Band 5 applied to phase-delays corrections

This topic has remained under-developed during the 1st half of the study and thus we focused on it during the 2nd part of the project. The goal of this *Approach 2* (DS.1.2) is to provide an alternative to WVR/wvrgcal() that would be viable also for the solar observations, where WVRs are saturated in the on-source scans. We have essentially developed, extended, and tested the legacy idea of Richard Hills (2019), which is presented as the *Annex 3* to the MTR.

Here, after developing basic theoretical concept on how to calculate PWV proxy from the raw data (measured powers) in the science SPWs and its implementation as a collection of Python routines, we apply the methodology first to the band-pass calibrator scan. This is for three sakes: (i) Clear and easy comparison between PWV-based correction of phases with the exact *gaincal()* phase solution for the calibrator, (ii) clear and easy comparison between our *PWV proxy* and directly measured (by WVRs) PWV, and (iii), because we need not care about how much the variations of the signal in the science SPWs is influenced by variability of the target source – the calibrator is so weak that in auto-correlation (AC) signal we essentially see just the intrinsic emission of the terrestrial atmosphere (+ obvious instrumental noise). The more complicated case of the solar data deserves an extra chapter and it is handled in detail in the next part, in Chapter 2.3.

2.2.1 Theoretical background

We are going to use the *science* SPWs for extracting emission (in case of the weak source, typically band-pass or phase calibrator) or absorption (in case of the solar on-source scans) features connected with the atmospheric WV line at 183 GHz in Band 5. Fast variations of those features are supposedly connected with varying PWV and used to calculate a proxy quantity to the differential (i.e., the time-dependent differences to the time-averaged PWV) water-vapor column measure. We call this "*proxy dPWV",* in the following analytical derivation it is usually denoted just as *dw.* Eventually, this *dPWV proxy* is transformed to the phase delays, similarly as in the *wvrgcal(),* and used for corrections of the fast-varying phase errors.

We have multiple possibilities for choosing the science SPWs for analysis – it can be auto-correlations (ACs) either in the channel-resolved SPWs – i.e. SPWs 5,7,9, and 11 in our Band 5 MS data, or their channel-averaged counterparts (SPWs 6,8,10, and 12). Alternatively, the SQLDs in SPWs 0,1,2,3 may be utilized for the goal. Anyway, even in the case of channel-resolved SPWs, we have to perform integration at least over part of the SPW channels, in order to reach sufficient signal to noise ratio. In the following we start with general radiative transfer over the terrestrial atmosphere with wet and dry components and arrive to a general expression for the dPWV proxy expressed in terms of variations of the brightness temperature (as a function of both time and frequency). Because we are in fact measuring voltages/powers, we have to transform the brightness temperature to the directly measured quantities using calibration equations. After doing that, we provide specific expressions for the dPWV proxy in the case of channel-resolved AC SPWs, and channel-averaged ACs/SQLDs.

Brightness temperature T_{sky}, which represents a signal contributed by both the terrestrial atmosphere and the celestial source, can expressed as (see Richard Hills original analysis):

$$T_{sky} = \frac{1}{BW} \int_{spw} [\eta T_{src}(f) \cdot tr(f) + T_{atm}(1 - tr(f))] df \qquad (2.2.1)$$

where η is the effective filling factor for the antenna primary beam (for the Sun observed on the disc it is 1.0), *BW* is the bandwidth over which we collect the signal (~2 GHz for channel-integrated ACs and SQLDs), and *tr(f)* is the transmisivity of the terrestrial atmosphere

$$tr = \exp[-(w \cdot \tau_1 + \tau_0)] \qquad (2.2.2)$$

In (2.2.2), w is the water-vapour column (PWV), $\tau_1(f)$ is the (frequency-dependent) opacity for PVW=1.0, and $\tau_0(f)$ is the 'dry' component of the opacity.

Here we ignored the forward and back-ward antenna efficiencies described in White et al. (2017) because, if needed, these essentially instrument-determined parameters (T_{spill} in Eq. (5) in White (2017) is approx. the ground temperature in the location of antenna, perhaps not changing during the scan) can be incorporated into factor inverse gain and the additive constant in calibration equations below, which are instrument-based parameters, too.

Now, we assume that the water-vapor content w as a function of time during the scan can be decomposed into the constant (during the scan) mean part $<w>$ and a small fluctuation dw (c.f. Richard's analysis; the dw is a mathematical symbol for the quantity that we refer as dPWV in the text)

$$w(t)=\langle w\rangle+dw(t) \qquad (2.2.3)$$

where quantities in < >'s mean time-averaging over the scan duration. With this, one can express the transmissivity (2.2.2) as

$$tr(f,t)=\exp[-(\langle w\rangle\tau_1(f)+\tau_0(f))]\cdot\exp(-dw(t)\tau_1(f))\approx\langle tr(f)\rangle\cdot(1-dw(t)\cdot\tau_1(f)) \qquad (2.2.4)$$

Now, the source itself may have its own brightness variability, say its brightness temperature can be split into time-averaged (over the scan) part and a small fluctuation as

$$T_{src}(f,t)=\langle T_{src}(f)\rangle+\delta T(f,t) \qquad (2.2.5)$$

After inserting the fluctuating transmissivity (2.2.4) and source temperature (2.2.5) into Eq. (2.2.1), we get

$$T_{sky}(t)=\frac{1}{BW}\int_{spw}\left(\langle T_{src}(f)\rangle+\delta T_{src}(f,t)\right)\cdot\langle tr\rangle\cdot(1-\tau_1 dw)+T_{atm}(1-\langle tr\rangle)+T_{atm}\langle tr\rangle\tau_1 dw(t)\,df \qquad (2.2.6)$$

The integrated expression can be re-arranged as

$$\delta T_{src}\langle tr\rangle+\langle T_{src}(f)\rangle\cdot\langle tr\rangle+T_{atm}(1-\langle tr\rangle)-dw(t)\tau_1(\langle tr\rangle\cdot\langle T_{src}\rangle-\langle tr\rangle T_{atm}+T_{atm}-T_{atm}), \qquad (2.2.7)$$

where the last two terms $+T_{atm} - T_{atm}$ have been added in order to combine the expression in the parentheses into more meaningful form (see Eq. (2.2.9) below). We have also neglected the term $\delta T_{src}.dw$, which is second order in small fluctuations. Because the fluctuations of the intrinsic source brightness and fluctuations in the terrestrial-atmosphere transmissivity are uncorrelated, we can rewrite their time-average product as

$$\langle T_{src}\rangle\cdot\langle tr\rangle\approx\langle T_{src}\cdot tr\rangle$$

When we use the expression for T_{sky} in Eq. (2.2.1) to find its time-average (over a scan)

$$\langle T_{sky}\rangle = \frac{1}{BW}\int_{spw}\left[\langle T_{src}\cdot tr\rangle + T_{atm}\left(1-\langle tr\rangle\right)\right]df\ , \qquad (2.2.8)$$

the Eq. (2.2.6) can be rewritten as

$$T_{sky} = \langle T_{sky}\rangle - dw(t)\left(\frac{1}{BW}\int_{spw}\tau_1(f)\langle T_{sky}(f)\rangle df - \frac{T_{atm}}{BW}\int_{spw}\tau_1(f)df\right) + \frac{1}{BW}\int_{spw}\delta T(f,t)\langle tr(f)\rangle df \qquad (2.2.9)$$

The intrinsic source-brightness fluctuations (the last integral) are unknown, so we neglect them for now (and append the corresponding term to the final result for completeness). We, however, return back to this term in the following Chapter 2.3, which handles the case of solar data. In order to explicitly distinguish between the quantities that are averaged over the frequency channels with total bandwidth *BW*, we add the mean index to them, while the time-averaging over the scan is designated by the angular brackets. With this, we eventually arrive to expression

$$T_{sky}^{mean}(t) = \langle T_{sky}^{mean}\rangle - dw(t)\frac{1}{BW}\int_{spw}\tau_1(f)\left(\langle T_{sky}(f)\rangle - T_{atm}\right)df \qquad (2.2.10)$$

which relates the variations of the brightness temperature in the pass-band BW with the variations of the water-vapor column *dw(t)* – i.e. our dPWV proxy.

2.2.2 From brightness temperatures to measured powers

We follow here the general solar ALMA calibration procedure as it is described in White et al. (2017) and Shimojo et al. (2017).

The sky brightness temperature is related to the received power via Eq. (5) in White et al. (2017):

$$P_{sky} = G\cdot\left(T_{RX} + T_{sky}\right) + P_0 \qquad (2.2.11)$$

where G is the (generally channel-dependent) gain, T_{RX} is the instrumental receiver temperature and P_0 (in alternative notation P_{zero}) is the instrumental offset, which is zero in the case of auto-correlations and may have non-zero value for SQLDs.

In order to find the unknowns G and T_{RX}, ALMA antennas are equipped with the front-end ACD calibration device. This blocks (for a while, during the atmosphere / ATM scans) the sky view by the black-body obstacles at two different temperature levels – so called ambient and hot loads. The relation between temperature and corresponding power in the case of ambient, respective hot load thus reads

$$P_{\rm amb} = G\{T_{\rm rec} + T_{\rm amb}\} + P_{\rm zero}$$

$$P_{\rm hot} = G\{T_{\rm rec} + T_{\rm hot}\} + P_{\rm zero}.$$

From those, the gain *G* the effective receiver temperature (T_{rx} / T_{rec}) can be expressed as

$$G = \frac{P_{\rm hot} - P_{\rm amb}}{T_{\rm hot} - T_{\rm amb}} \qquad (2.2.12a)$$

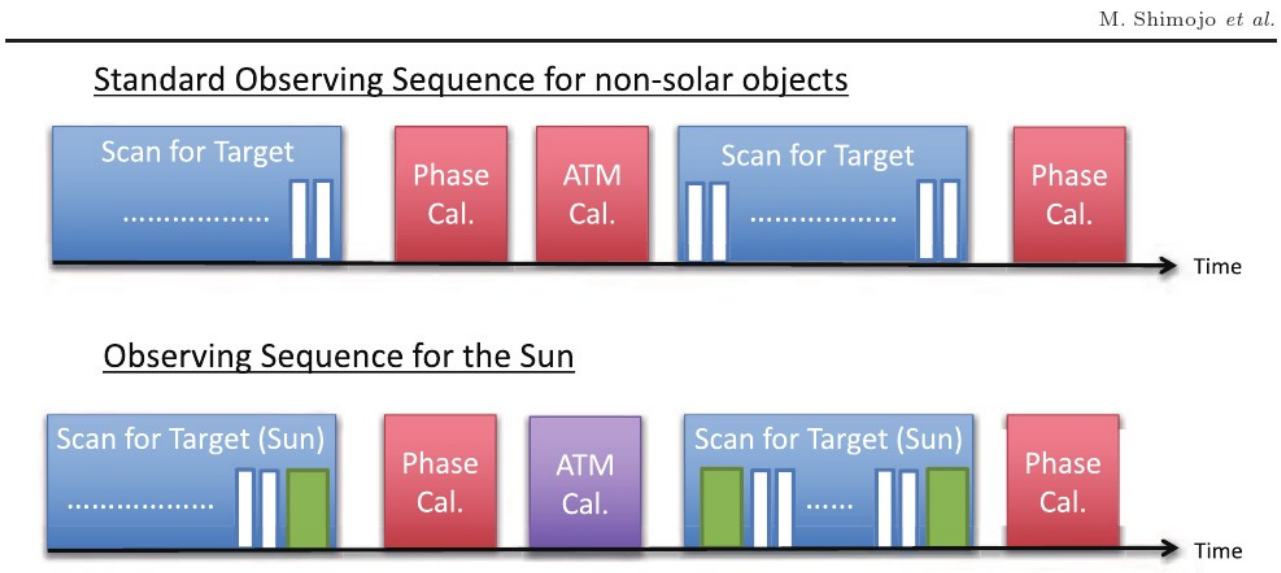
$$T_{\rm rec} = \frac{T_{\rm hot}\,(P_{\rm amb} - P_{\rm zero}) - T_{\rm amb}\,(P_{\rm hot} - P_{\rm zero})}{P_{\rm hot} - P_{\rm amb}} \qquad (2.2.12b)$$

Once *G* and T_{rx} are established for a given antenna, SPW, and ATM scan, they may be used for calibration of the AC power for the target – the BP calibrator or the Sun. With the BP calibrator the situation is simpler because it is observed with standard attenuation settings. So its corrected signal is

$$T_{\rm sky} = (1 - {\rm e}^{-\tau})\eta_{\rm l}\,T_{\rm atm} + {\rm e}^{-\tau}\eta_{\rm l}T_{\rm CMB} + (1 - \eta_{\rm l})\,T_{\rm spill} = \frac{P_{\rm sky} - P_{\rm zero}}{G} - T_{\rm rec}$$

where P_{sky} is the measured (raw) AC or SQLD signal, and T_{sky} is the sum of contributions of the terrestrial atmosphere (the vastly dominant one) and the emission of the BP calibrator. This is what we are interested in – its variations follow the WV contents and can be re-calculated into phase shifts.

For the Sun, the situation is more complex: It is observed with enhanced attenuation and therefore yet another ATM sub-scans (off-source) are performed – these are the very first and very last sub-scans of the science-target scan – see the following scheme (Shimojo et al., 2017). These ATM sky-like sub-scans are performed with the same attenuation level as the Sun is observed.



Therefore (because of higher attenuation), we have a different gain factor G_{src} for the 'true'-solar and solar-off scans:

$$P_{\rm src} \;=\; G_{\rm src}\{T_{\rm rec} \,+\, {\rm e}^{-\tau}\eta_{\rm l}\,T^{*}_{\rm src} \,+\, (1-{\rm e}^{-\tau})\eta_{\rm l}T_{\rm atm} \,+\, (1\,-\,\eta_{\rm l})\,T_{\rm spill}\} \,+\, P_{\rm zero}$$

$$P_{\rm off} \;=\; G_{\rm src}\{T_{\rm rec} \,+\, (1-{\rm e}^{-\tau})\eta_{\rm l}T_{\rm atm} \,+\, (1\,-\,\eta_{\rm l})\,T_{\rm spill} \,+\, {\rm e}^{-\tau}\eta_{\rm l}T_{\rm CMB}\} \,+\, P_{\rm zero}$$

In combination with the expressions above, this modified gain can be expressed by means of that 'standard' gain as

$$G_{\rm src} \;=\; G\,\frac{P_{\rm off}\;-\;P_{\rm zero}}{P_{\rm sky}\;-\;P_{\rm zero}} \qquad (2.2.13)$$

Example of the dual-load calibration results, showing the standard and reduced (solar) gains according to Eqs. (2.2.12a) and (2.2.13) are displayed in Fig. 11, full set being available online in the project repository. In spite some peculiarities, like that the attenuated solar gain is sometimes higher than the non-attenuated one and for some antennas both gain and T_{rx} somewhat oscillate with frequency, there is no visible excess in this procedure. However, as the final result *dw* is quite sensitive to errors in *G* and T_{inst} – see Eq. (2.2.14) below – even smaller inconsistencies in the ACD calibration may impact the outcomes.

From Eq. (2.2.11) the sky brightness temperature can be expressed as

$$T_{sky}(f)=\frac{1}{G(f)}\cdot P(f)+\left(-\frac{P_0}{G(f)}-T_{RX}(f)\right)$$

and after inserting it into the Eq. (2.2.10), relating the brightness-changes to water-vapor column variations, one can express our searched dPWV proxy as

$$dw(t)=-\frac{\int_{BW}\frac{1}{G(f)}\cdot\left(P(t,f)-\langle P(f)\rangle\right)df}{\int_{BW}\left(\frac{\langle P(f)\rangle}{G(f)}-T_{inst}(f)-T_{atm}\right)\tau_1(f)\,df} \qquad (2.2.14)$$

In this universal relation all quantities at the RHS are in principle known – the power *P(t,f)* and its averaged (either in over pass-band channels or time of the scan) values are found in the science SPWs of the MS, gains and the total 'instrumental' temperature

$$T_{inst}(f)\equiv\frac{P_0}{G(f)}+T_{RX}(f)$$

can be calculated from the Eqs. (2.2.12b) and (2.2.13) above, where P_o required for SQLDs is found in the 'zero' ATM scan of the MS. The unit opacity, temperature of the atmosphere and scaling air-mass (secant of the zenith distance) can be found using the CASA **`atmosphere`** tool.

In case of the channel resolved windows, the integrals in the expression (2.2.14) can be calculated directly. For the averaged ACs (SPW 6,8,10, and 12) and SQLDs we have just the SPW-averaged values for *P*, *G*, and T_{inst}, which cannot be directly inserted into (2.2.14). Yet we can perform the integration over entire frequency range of those single-channel SPWs at least approximately: We start with the Eq. (2.2.10) and consider the functions $\tau_1(f)$ and $\langle T_{sky}(f) \rangle$ as linear over the integrated interval; for the SPW bandwidth of 2 GHz this is quite well satisfied (verified using the ATM model in the *atmosphere* CASA tool):

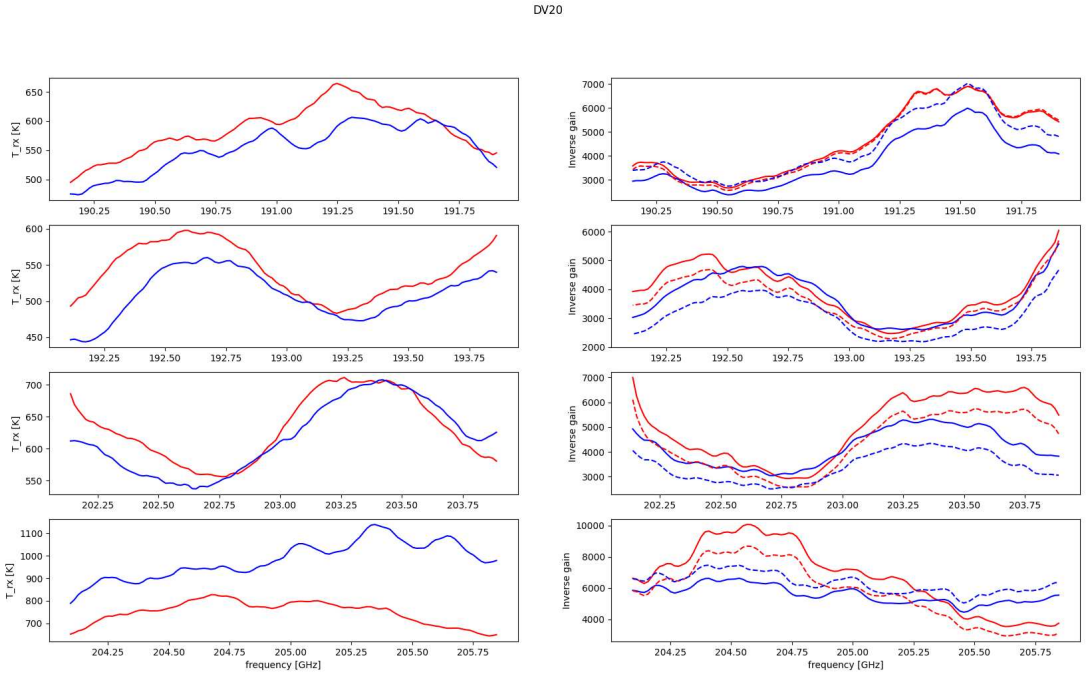


Figure 11: Receiver temperature T_{rx} (left part) and the inverse gain 1/G (right part) of antenna DV20 for SPWs 5,7,9, and 11 (from top to bottom) as a function of frequency. Red: XX, blue YY. Calculated from the ACD dual-load ATM scan preceding the Sun in-source target. The Sun is being observed with increased attenuation, so we need to calculate also the modified gain G (referred as G_{src} in Shimojo et al., 2017, see Eq. 2.2.13), which reflects the increased attenuation and is relevant for the Sun. It is calculated using the very first and very last sub-scans of the solar scan, which are done off-source – see acd_caltabs() in Annex 4. Here, the non-attenuated inverse gain 1/G is marked by dashed lines, the actual effective inverse gain 1/G*, is plotted solid by line.*

$$\left(f(x)\cdot g(x)\right)_{mean} \equiv \frac{1}{b-a}\int_a^b f(x)g(x)dx \approx f_{mean}(x)g_{mean}(x) + \frac{1}{12}\left(f(b)-f(a)\right)\cdot\left(g(b)-g(a)\right)$$

In this approximation, Eq. (2.2.10) can be rewritten as

$$T_{sky}(t)=\langle T_{sky}\rangle-dw(t)\left(\tau_1^{mean}\langle T_{sky}\rangle-T_{atm}\tau_1^{mean}\right)+\frac{1}{12}\left(\tau_1(f_{hi})-\tau_1(f_{lo})\right)\cdot BW\cdot\frac{d\langle T_{sky}\rangle}{df} \qquad (2.2.15)$$

Here, the index *'mean'* means averaging in frequencies over the entire spectral window and we do not put it explicitly to T_{sky}, as it is clear that this quantity is both time and frequency averaged. Frequencies f_{lo} and f_{hi} are the edge frequencies of the SPW, and the (linear) spectral slope of the $<T_{sky}>$ is determined by linear regression of the corresponding channel-resolved SPW (e.g., spw 5 in case of extracting the powers for brightness-temperature calculations from the spw 6), example of which is displayed for the Sun in Fig. 12 below.

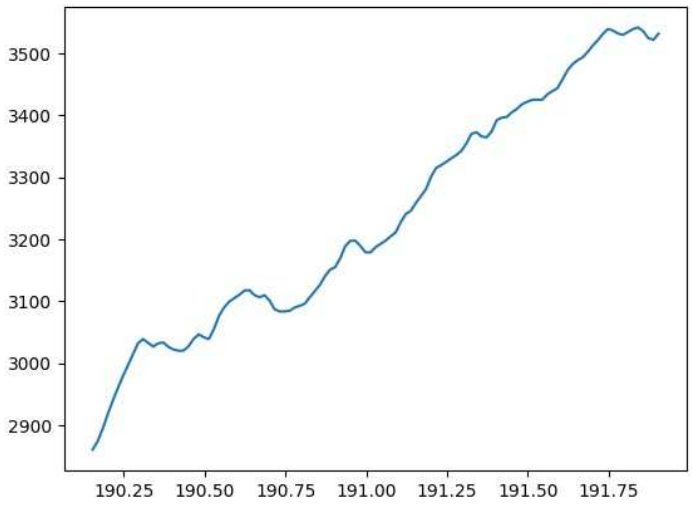


Figure 12: Time-averaged (over a scan) spectrum (frequency in GHz) of the sky brightness temperature when pointing to the Sun in the SPW 5 (in Kelvins). The T_{sky} has been calculated by Eq. (2.2.15) using the ACD dual-load calibration.

Eventually, if we express $T_{sky}(t)$ and its time-averaged value $<T_{sky}>$ from the dual-load calibration Eq. (2.2.11) by means of directly measured power $P(t)$ (and its time-average over the scan), and insert it into Eq. (2.2.15), we get

$$P(t)-\langle P\rangle=S_{ant}^{spw}\cdot dw(t)+\delta P_{src}^{tr}(t) \qquad (2.2.16)$$

Here, the last term corresponds to (unknown) intrinsic source variability as transmitted through the terrestrial atmosphere, and the *sensitivity factor* S_{ant}^{spw} ,which represents how much are the changes in PWV sensitive to the variations of the power measured by given antenna in a given SPW, is expressed as

$$S_{ant}^{spw}=-G\tau_1^{mean}\cdot\left(\frac{\langle P\rangle}{G}-T_{inst}-T_{atm}\right)-\frac{G\cdot BW}{12}\cdot\left(\tau_1(f_{hi})-\tau_1(f_{lo})\right)\cdot\frac{d\langle T_{sky}\rangle}{df} \qquad (2.2.17)$$

To sum up, we use directly Eq. (2.2.14), when expressing our dPWV proxy from the channel-resolved AC SPWs, while Eqs. (2.2.16) and (2.2.17) are suitable for the single-channel SPWs (averaged ACs or SQLDs).

2.2.3 Implementation

The above formulae and evaluation of all the necessary quantities are implemented – in addition to further methods for recalculation of the dPWV to phase delays, as a set of routines collected in the Python module *sol_wvrgcal.py*. Taking advantage of stability of the dual-load calibration in the channel-integrated SPWs, we use the ACs in single-channel spectral windows 6,8,10, and 12, by default, albeit alternatives using the channel-resolved auto-correlations and SQLDs have been tried, too, and comparison of all SPW choices is presented below. Namely we use the SPW 6, which is closest to the WV line center and the sensitivity factor (2.2.17) is thus the hightes. In order to get water-vapor variations dPWV (represented by the symbol *dw(t)* in Eq. 2.2.17) we need to:

1. Extract the raw measured power for the source as a function of time (per integration) from the MS for selected scan and spw – this is achieved by means of `tgt_pow()`. We then calculate also its time average <P> and subtract it from the raw data (see Eq. 13).
2. In order to calculate the sensitivity factor *S*, we need:
   - G (or G* for the Sun) – obtained via `acd_caltabs()`
   - T_{inst} (which needs P_o and T_{RX} in addition to G) – achieved via `acd_caltabs()`
   - τ_1 – the wet opacity for PWV=1.0 – its mean value over the SPW and values for the edge channels of the SPW. This is achieved via ATM model – we set the atmosphere tool in `setAtmTool()` similar way like `aU.plotAtmosphere()` does and then extract the necessary ATM characteristics (spectral dependence of τ_1 and the mean atmospheric temperature T_{atm}) using `wopacity()`. The spectral slope d<T*sky*>/df is obtained from time-averaged spectrum of the brightness temperature (source + atmosphere) – the power data in 128-channel spw (spw 5 in our case) are calibrated by the dual-load (ACD) calibration as implemented in `tgt_Tb()`; standard `np.polyfit()` is used for linear regression then.
   - Eventually, we divide the difference *(P(t) – <P>)* by the calculated factor *S* and get the *dw(t)*, i.e., dPWW as a function of time per antenna and SPW (neglecting the intrinsic source variability, of which we have no clue). Entire combined procedure is assembled in the routine `dPWV()` at the end of the file.
3. In interferometry, we are not measuring the phase delays w.r.t. the source in vacuum, but always w.r.t. the reference antenna. The last step therefore is that we calculate dPWV for a given antenna, the same quantity for the reference antenna (DV10 = id 37 in our specific dataset) and subtract the latter from the former. This quantity should be already proportional to the phase delays (related in any gain calibration also w.r.t the reference antenna).
4. In order to make actual comparison, an auxiliary method `getPhase()` is contained in our collection, which extracts the fast phase variations from the calibration table found as a fast (integration-based) `gaincal(type='p')` solution for the band-pass (BP) calibrator (this calibration is always done before `bandpass()` is called).

2.2.4 Correlations between dPWV proxy and the actual phase delays in BP calibrator

We try to compare the data for the BP calibrator first, where we can trust the phase solution (it is a self-cal for the point source, i.e., straightforward). The case of the solar scan is handled in the separate Chapter 2.3. Figs. 13 and 14 show comparison between our dPWV proxy obtained by the procedure described above (Fig. 13), and the phase delays as per BP-calibrator phase gaintable (Fig. 14), for the antenna DA45 (one of the antennas far from the array center). Actually we are plotting –dPWV, because the coefficient between dPVW and phase delay is negative and the obvious similarity is better visible in the positive correlation.

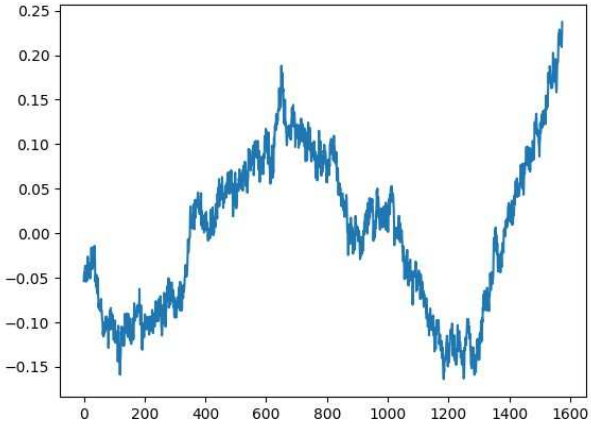


Figure 13: Negative (i.e. multiplied by factor -1.0) dPWV proxy calculated by Eq. (17) from the channel-integrated AC (SPW 6) in the BP-calibrator scan. Time is expressed as an integration number (integ. duration is 0.192s), dPWV value is in mm (of PWV).

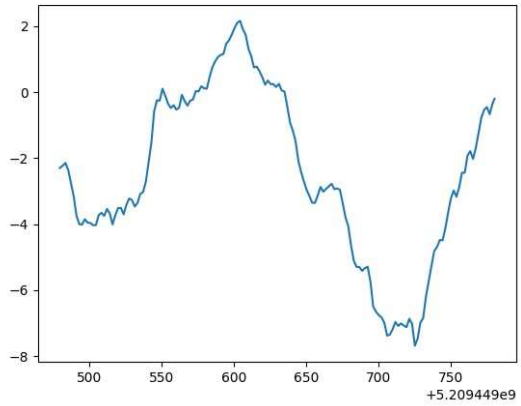


Figure 14: Integration-based phase solution for the BP calibrator in spw 5, the same scan as in left figure. Time is expressed in secs as the MJD, phase variations have been un-wraped and are displayed in radians, i.e. the displayed phase range is over 540 degs!

Similarities in both curves are apparent. We can go further and make cross-correlation plot between both quantities (after rebinning the dPWV to the same cadence of 1.92 sec). This is shown in Fig. 15.
We have deliberately used the line connecting the individual scatter-plot points for the same time instants, the line thus basically represents the time path or a 'history'. From that point of view, there is a strong impression that the slope (the coefficient that relates changes in dPWV and corresponding changes in phase) is more or less constant but the phases (their PWV-independent part) are systematically decreasing with time. We return to that effect later, now let us find the coefficient of proportionality between dPWV proxy and corresponding phase error.
We can start with the theoretical relation (R. Hills, 2019 – see MTR)

$$d\varphi = 2\pi/\lambda \,.\, r_{eff} \,.\, dw \qquad (2.2.18)$$

If we assume average wavelength 1.6mm for the solar Band 5 observations in the low-frequency SPW 5 (the left spw in the LSB; as much close to the WV line as possible) and the effective refractive index (R. Hills, 2019) of value of r_{eff}=6.3, the coefficient of proportionality is going to be -25.0 rad/mm.

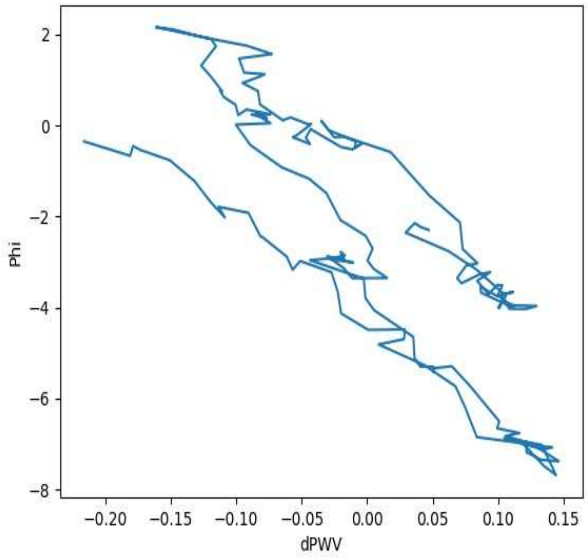


Figure 15: Cross-correlation between dPWV . Intentionally we use continuous line in the plot, which somehow helps to track also the time dimension. Correlation is apparent but there is a strong impression that the absolute value of the phase is decreasing also with time.

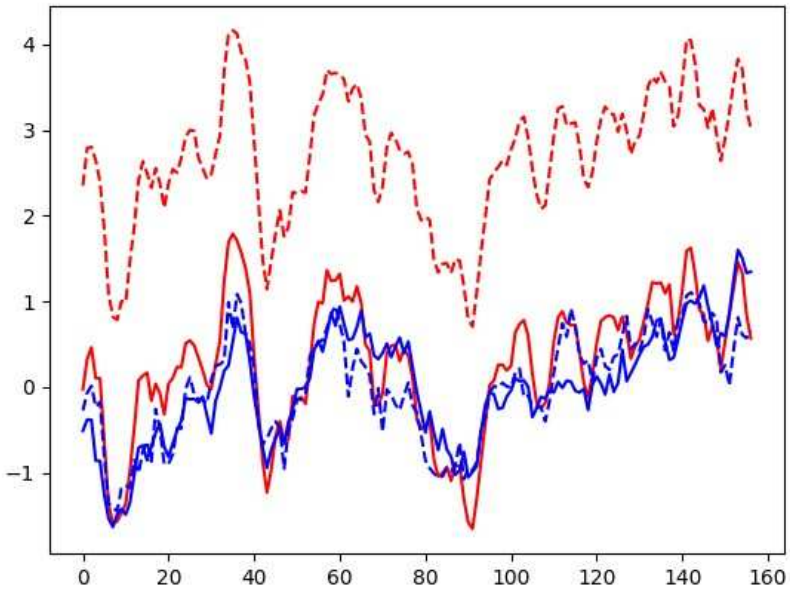


Figure 16: Actual phase delays as found from gaincal() for the BP calibrator (red curves; solid represents XX and dashed YY pol, respectively) in comparison with our prediction from the dPWV proxy, scaled by the theoretical factor -25.0 (blue) for antena DV14. X-axis shows time expressed as integration number, integration duration is 1.92 sec. Y-axis is phase in radians.

Using this value, we can overplot the expected (by measuring the dPWV and using the above scaling) and actually measured (determined by integration-based *gaincal()* solution for the BP calibrator) phase delays. This is implemented in *dwXphi_plots()* and a sample result for the antenna DV14 is shown in Fig. 16. Again, the similarity of predicted to actual phase delays is obvious.

2.2.5 Time-detrended correlation between the phase delays and dPWV

In spite the 'by eye' comparison between the predicted and actually determined phase delays for the BP calibrator shows high level of similarity, the scatter plot shown in Fig. 15 reveals that there may be some slow dynamics (a time trend) in phases, not correlated to the changes in PWV. In order to study this phenomenon systematically, we have adopted the following approach: We are trying to do linear regression between dPWV and phase-delay, but we, moreover, allow the phase variation to be depending also on time, which for the equidistantly-spaced time series can be expressed as an index *i*. Formally speaking, we are trying to minimize the functional (basically χ^2)

$$F(a,b,t)=\sum_{i=1}^{N}\left(y_i-t\cdot i-a\cdot x_i-b\right)^2, \qquad (2.2.19)$$

where x_i values represent our independent variable dPWV, y_i are corresponding actual phase delays, *a* is the coefficient of linear regression between those two (we expect it ideally at its theoretical predicted value of -25.0), and *t* is the linear time trend, as the index *i* in the series represents homogeneously growing time. The absolute term *b* represents the constant offset between the actual phases and those predicted by the dPWV. This is not relevant, as we have no possibility to detect anything above the PWV variations over its scan-averaged value (which can be found in the MS using the *atmosphere* tool).

We minimize the functional via standard procedure, i.e., finding the stagnation point for its all three parameters:

$$\frac{\partial F}{\partial a}=0,\frac{\partial F}{\partial b}=0,\frac{\partial F}{\partial t}=0 \qquad (2.2.20)$$

which leads to the system of linear equations for unknown minimizing parameters *a, b*, and *t:*

$$\begin{aligned} a\sum x_i^2+b\sum x_i+t\sum i\cdot x_i&=\sum x_i\cdot y_i \\ a\sum x_i+b\cdot N+t\cdot N\frac{(N+1)}{2}&=\sum y_i \\ a\sum i\cdot x_i+b\,N\frac{(N+1)}{2}+t\cdot\frac{2N^3+3N^2+N}{6}&=\sum i\cdot y_i \end{aligned}$$

Here we have omitted the obvious sum-limits (*i=1..N*) and replaced sums of *i* and i^2 by their analytical expressions. The system may be further easily reduced to system of two equations by expressing *b* from the 2nd equation and inserting it into 1st and 3rd.

$$a\cdot\left(\sum x_i^2 - N\cdot\bar{x}^2\right)+t\cdot\left(\sum i\cdot x_i - N\frac{(N+1)}{2}\bar{x}\right)=\sum x_i\cdot y_i - N\cdot\bar{x}\cdot\bar{y}$$
$$a\cdot\left(\sum i\cdot x_i - N\frac{(N+1)}{2}\bar{x}\right)+t\cdot\left(\frac{N(N-1)(N+1)}{12}\right)=\sum i\cdot y_i - N\frac{(N+1)}{2}\cdot\bar{y} \qquad (2.2.21)$$

Here, the bar over a quantity means its regular averaging (i.e., the sum divided by *N*). This linear 2x2 system is then easily solved by Cramer's rule. Not showing the algebra details here, it is straightforwardly implemented in helper routine *paregresion()* in the attached collection `sol_wvgcal.py`. We have tested this routine on an artificial signal before using it for our data. Complete results of this analysis are summarized in the XLS table available online at http://wave.asu.cas.cz/shared/ESO/final_report/solar_wvr/, the meaning of the table columns is following:

Col A: Antenna id_name.

Cols B – L: Individual terms for dPWV calculation according Eq. (14) in *Progress report no. 11*.

Cols M – R: Coefficients of parameterized linear regression (routine *paregression()* in Annex 1) – linear time-trend, correlation coefficient, and offset, respectively – all extra for XX and YY pols.

Cols S – V: Positions of the antenna in the array and its distance from the reference antenna DV10.

Cols W – AA: Measures of the correlation quality – Xi-square and 'by-eye' quality mark for XX and YY separately, and better value of them. We have kept only antennas for whose the value in column AA is at least 6.

Cols AB and AC: Time-trend and correlation coefficient for the better polarization (XX or YY) – basically the copied value from the column M or N depending whether XX or YY pol has better qualitative mark for t-trend and, analogically, O or P-column value for the correlation coefficient.

Col AD: Position of the antenna (cols S and T) projected to the unit vector with azimuth defined by value (in degs) in the cell V41.

For some antennas, like DV17, whose case is displayed in Fig 17, the time-detrending helps a lot to make the scatter plot very narrow, having perfect correlation between the dPWV and actual phase delays. However, there are also cases, where no significant improvement is reached – see Fig. 18 displaying antenna DA41. Some antennas exhibit yet weaker correlations.

We first focus on the coefficients *a* of the linear correlation between the dPWV proxy and the phase delay (see Eqs. (2.2.19) – (2.2.21)) – we are namely interested, how close they are to the theoretical prediction based on Eq. (2.2.18), which foresees the value close to -25.0 rad/mm.

First, we would like to compare the regression coefficient only for the data with good Tsys, and consequently, good cross-correlation. To that end, two parameters are measured: The Xi-square –

exactly the value of F from the Eq. (2.2.19) above, and a subjective 'by-eye' quality mark which evaluates the Figs. like 17 and 18 for each antenna and polarization and attributes each result by value from 0 (no correlation) to 10 (perfect match). As some antennas have good T_{sys} in one polarization and bad in another one, it projects to the results here similarly and there are antennas with good quality mark in one polarization and bad in the other. Eventually, we have kept that polarization, which shows better quality to have a single number per antenna. Why we do not use just Xi-square? Because it should be most likely properly normalized: A good cross-correlation of the phase variations (those actual vs. those predicted by dPWV) may have rather large Xi-square just because both quantities exhibit large fluctuations (typically for distant antennas). So the value of chi-square itself is not really addressing the quality of correlation. Hence, we kept in our ensemble only such antennas, which have at least in one polarization the by-eye quality mark above 5 and reasonably low Xi-square at the same moment. This keeps 37 of total 52 (and of 50 unflagged in further calibration) antennas in the processing. As can be seen from the column AC (with title "K better") in the on-line XLS table, the correlation coefficient $d\varphi/dPWV$ between dPWV and measured phase variation, indeed, varies around its average value -25 rad/mm with its standard deviation ~8 rad/mm.

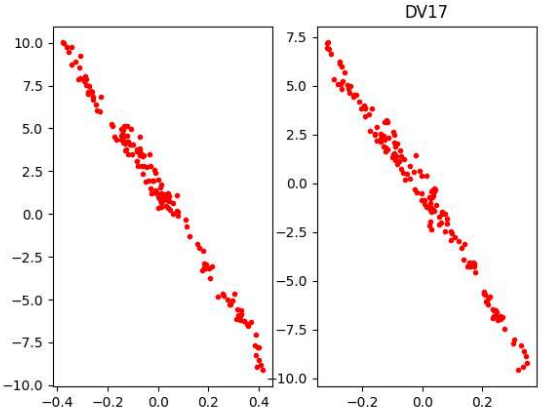


Figure 17: Phase vs dPWV for antenna DV17 – like in Fig. 15, but now after time detrending.

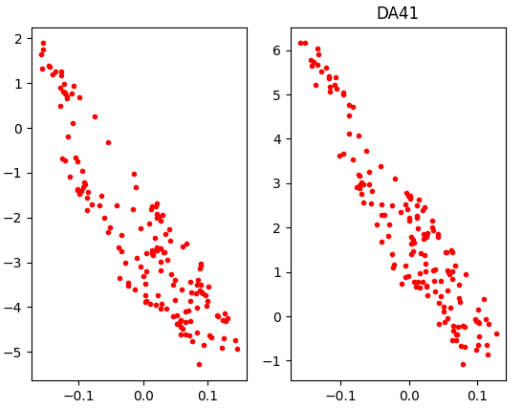


Figure 18: The same like in left figure, now for the antenna DA41.

In the next analysis we have been investigating the nature the slow phase-delay dynamics that is independent of the dPWV variations. Namely, we are interested in (i) whether and how the linear trend/drift in the residual phases (i.e., after the dPWV-correction is applied) is related to antenna position in the array (indication of whether the residual phase is related rather to instrumental or astronomical influences) and, (ii) whether the residual phase variations actually can be approximated as linear trend/drift or this slow dynamics is more complex.

Actually, easier answer is to the item (ii): No, as it can be seen from the overplots between the actual phases and their prediction made by dPWV scaled properly by the correlation coefficient (approx. -25.0 rad/mm) the difference between the two phases (i.e. phase residua) only sometimes can be approximated as a linear drift. In many other cases they show slow-time-scale variations that, however, cannot be interpreted as linear trend.

In this view, it is understandable that also answer to question (i) will be negative – we did not found any significant relation between the linear time-trend/drift and position of the antenna in the array. In many cases it is just because the linear trend is not appropriate description of the slow phase variations, as explained above. Anyway, for addressing this question we have characterized position of the antenna in the array by two quantities: (1) Distance from the reference antenna, (2) Position in the antenna in the array projected to the unit vector of selected direction (azimuth). The latter has been investigated for the sake to find possible correlation either with the source hour-angle or with the direction of the prevailing wind (that transports the PWV across the array). To that end, antenna positions in the array has been added into XLS table (referred by the URL above) and two columns of derived quantities: Distance from ref. antenna (column V), and antenna position projected to the unit direction vector at azimuth defined in the cell V41 (column AD). These two columns were compared with the linear trend/drift expressed for each antenna in the column AB ("T-trend better").

Results are shown in the on-line available XLS table and for convenience partly also here as Fig. 19. apparently in none case the relation between the coefficient of linear frequency drift and measure of the antenna position has been found. The current result shown in the on-line available material are displayed for the projected antenna position corresponding to projection direction (azimuth) of 75 deg, the other attempted azimuths give even more chaotic (i.e. with no systematic dependence) results.

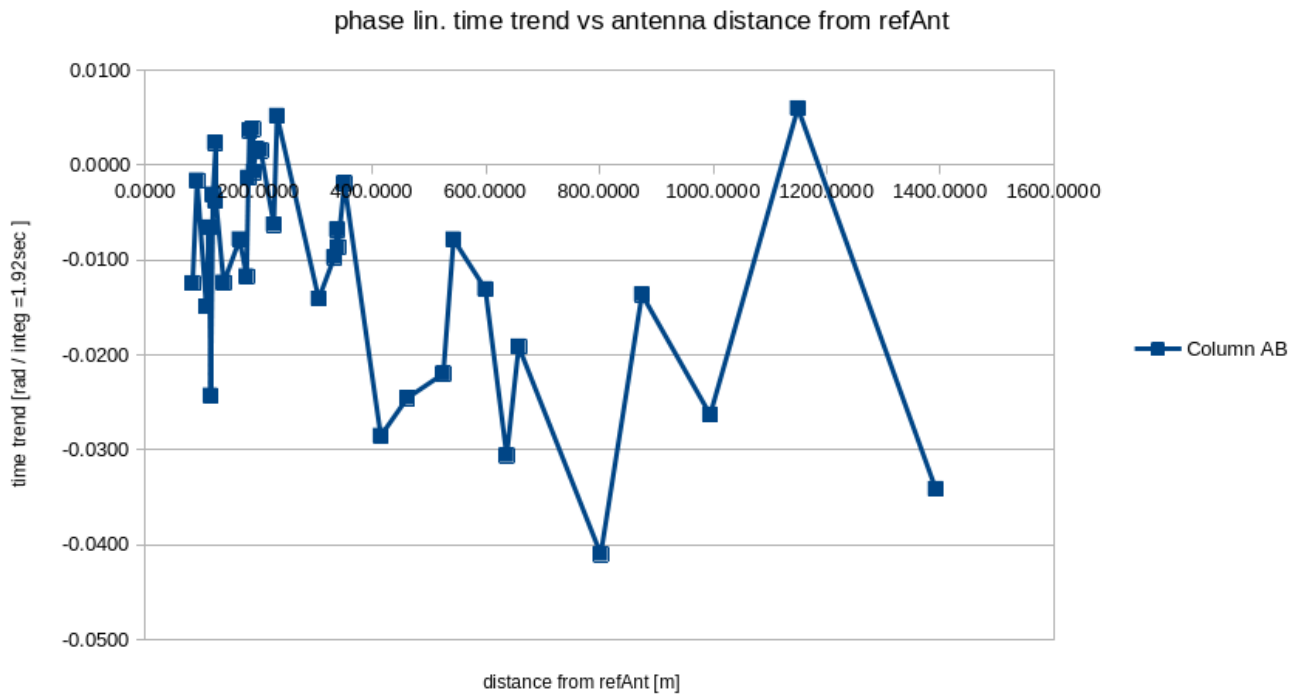


Figure 19: Linear time-trends/drifts of the frequency residuals in dependence on the antenna position in the array expressed by distance from the reference antenna. In the on-line XLS table there is moreover similar plot, where the slow phase dynamics expressed as the time trend is checked vs the antenna position projected to the specific direction vector. The results, however, show the same little or no systematic dependency.

2.2.6 Using the dPWV proxy to fixing the phase errors

From the plots like that in Fig. 16 we are already close to direct application of the dPWV proxy to apriori correction of the phase errors. Since it looks that for the good antennas (with good Tsys) the procedure works quite fine, we decided to come to the correction of the BP phases by dPWV. Two approaches were compared: (i) Blind correction, with the theoretical (and – basically – also experimentally confirmed by the analysis summarized in the on-line XLS table) correlation factor of -25rad/mm, and (ii) with the correlation factor experimentally found by solving Eqs. (2.2.19) – (2.2.21) – i.e. optimized for each antenna and with at least linear time trend in the slow dynamics of phases removed. The latter is studied just for comparison – in the future real applications to the solar data we will not have the actual phases to do the (even de-trended) regression with, only possibility is to study the stability of the correlation coefficient not just for the BP calibrator but also over the subsequent phase-calibrator scans, and use the same value also for the unknown target, the Sun. This approach mimics that used in wvrgal() calculations of dT/dL coefficients between the variations of WVR brightness temperature and corresponding fluctuations in optical path.

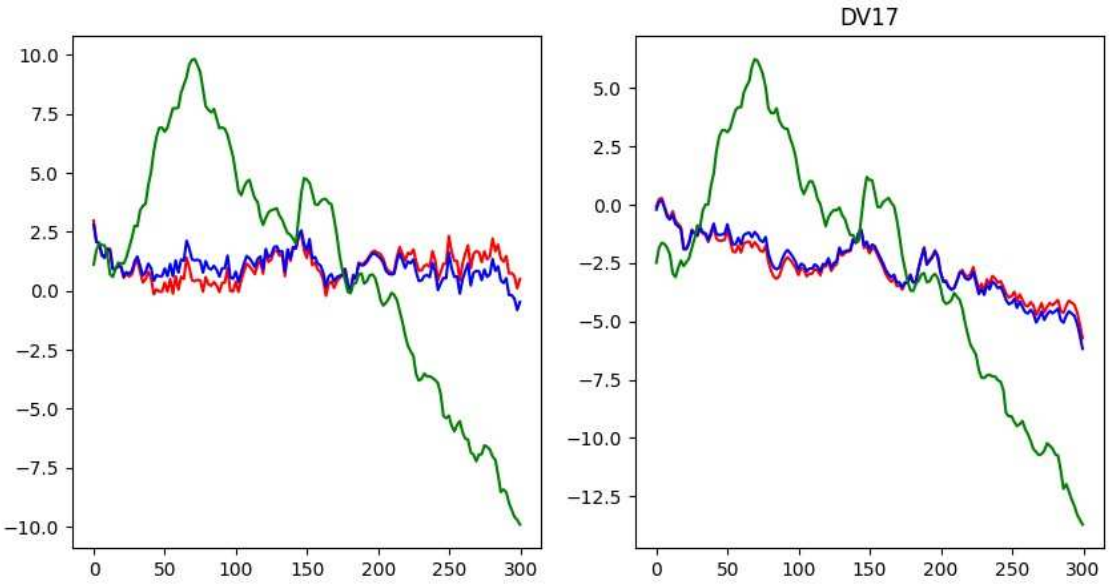


Figure 20: Antenna DV17, the BP-calibrator scan (no. 5) in SPW 5. Original phase variations (green) and after correction by dPWV-based calibration – a blind correction with the factor of -25rad/mm (red), correction by the optimised correlation factor found by linear regression with time de-trend (blue). Left and right panels are for XX and YY polarizations, respectively, the x-axis denotes time in seconds since the beginning of the scan. The y-axis is in radians (as all similar figures here).

The procedure is implemented in the routine *blindPhiFix()*. We, again, did the study systematically for all antennas and both polarizations, just two illustrative examples are shown in Figures 20 and 21, where we present the original phase variations as found by *gaincal()* in green, the phase-variation residuum after subtracting the blind correction from the original variations according to relation $\varphi_{residual}=\varphi_{orig}$ *+25.0*dPWV* (in red), and optimized correction with cross-correlation coefficient and time-detrend found by *paregression()* routine (blue curve). The first figure for antenna DV17 shows

really good reduction of the phase variations, with a small linear trend in the residuum of pol YY (right panel), the latter figure (antenna DA41) presents still good reduction of the original phase variations, but it is clear, that the residual phase error can not be interpreted as a linear trend (this is related to our investigation above). Generally speaking, even the blind correction reduces the RMS of phases by factor at least 2.

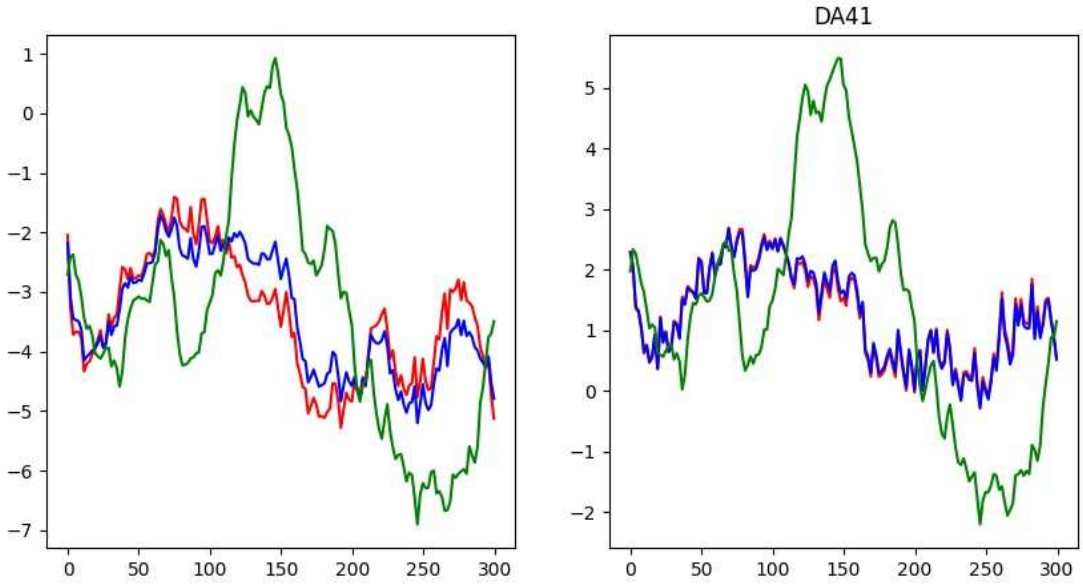


Figure 21: Like the figure above, now for the antenna DA41. Apparently, the residual (phase after correction) phase variations for DA41 exhibits the slow-time-scale variations which, however, cannot be approximated as a linear trend.

2.2.7 Comparison of dPWV proxy with WVR measurements

Clearly, if our approach to fixing the fast phase variations based on the varying WV signal in the science SPWs (the methodological *Approach 2* of the project) should represent an alternative to the standard WVR and *wvrgcal()*, both methods should give comparable results for the cases, where we can compare them. In our solar dataset this can be done for calibrators, namely for the BP/flux calibrator, whose scan no. 5 is long enough (~ 5 min). In this paragraph we aim at such a comparison.

After inspection of our MS, we have found four resources related to the direct WV measurements:

- ASDM_CALWVR in MS
- ASDM_CALATMOSPHERE in MS
- in original ASDM dir: CalWvr.xml and CalAtmosphere.xml
- SPW 4 in the MS.

AnalysisUtils offer a couple of related utilities for accessing the data (actually those in the items 1 – 3 above), most relevant for us being *readPWVFromMS()* and its 'graphical representation' *plotPWV()*. Quick inspection of the data in ASDM_CALWVR shows that we have actually quite sparse time coverage: This table contains 468 points for 52 antennas, giving 9 time points for entire measurement set. The ASDM_CALATMOSPHERE table is only slightly better with 12 time points over the entire MS and the XML files in the original ASDM dir seem to be just formatted representations of those above mentioned tables. The situation is illustrated by the output of plotPWV(), which is shown in Fig. 22: We have (spline-interpolated) variations of the PWV for all antennas only on the slow temporal scales, based on the ASDM_CALWVR table data.

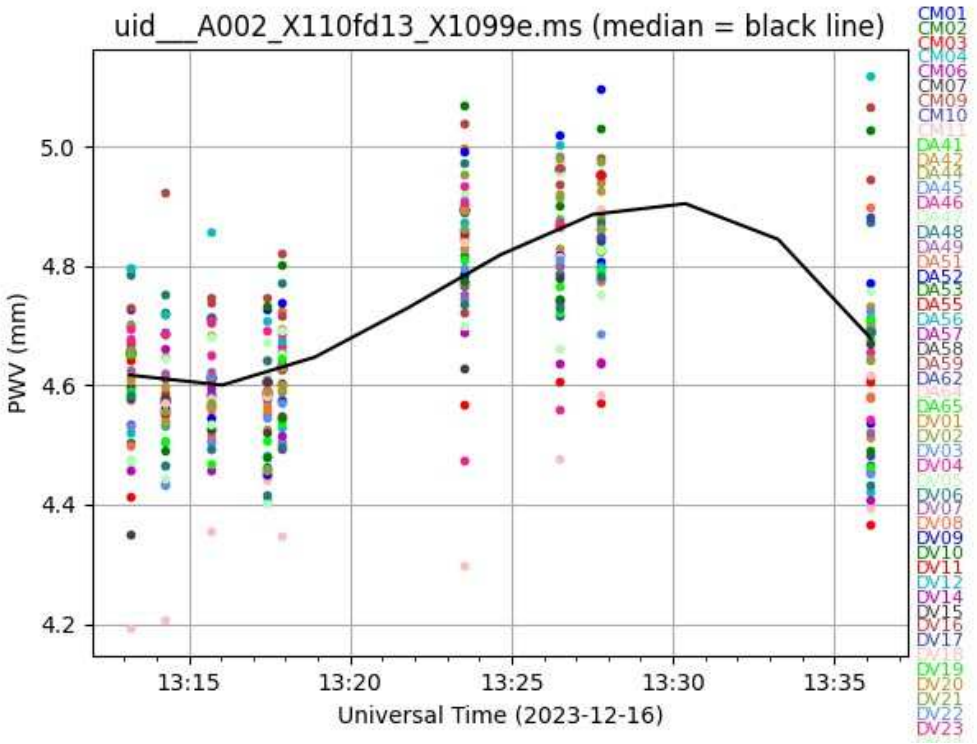


Figure 22: Output of plotPWV() for our MS.

Consequently, we have just a single point per scan and we thus cannot use the data for our purpose. The only usable high-cadence WV data are thus the direct temperatures in the four WVR channels in the SPW 4. These values are, on the other hand, far from the PWV derived from them. For this reason, we decided to make not the direct comparison between our dPWV-proxy and the directly measured PWV, but – also in view of the ultimate goal – compare the *impacts* of both quantities to the phase delays. Namely, we compare our dPWV proxy, scaled by the already several times mentioned factor $d\varphi/dPWV = -25.0$rad/mm, with the phases found by call to *wvrgcal()* routine. In order to prevent damage of the WVR data by the saturated solar scan, we have created the WVR gain table by a call

```
wvrgcal(vis=vis,caltable='wvr_gcal.W',segsource=True,
sourceflag=['Sun'],statsource='J1256-0547,refant='DV10',maxdistm=1.0,
minnumants=3)
```

which flags the solar data and calculates the stats from the BP calibrator. We have used the same reference antenna as for *gaincal()* in the calibration script. The last two parameters prevent making the wvrgcal() solution by interpolation for those antennas, which have no or corrupted WVR data. Namely, we prevent generating a 'fake' wvr gain caltable for 7m antennas, which are not equipped with WVRs.

The `wvr_gcal.W` table by the call above is accessed via *wvrPhases()* routine in our collection sol_wvrgcal.py and extracts the phases from the WVR-gaintable by applying the data selectors similarly as we do in get*Phases()* for the phases found by *gaincal()*. All antennas has been analyzed, here we present two randomly selected cases as an illustration – antenna DV10 (the reference antenna) and antenna DV17 – see Figs. 23 and 24.

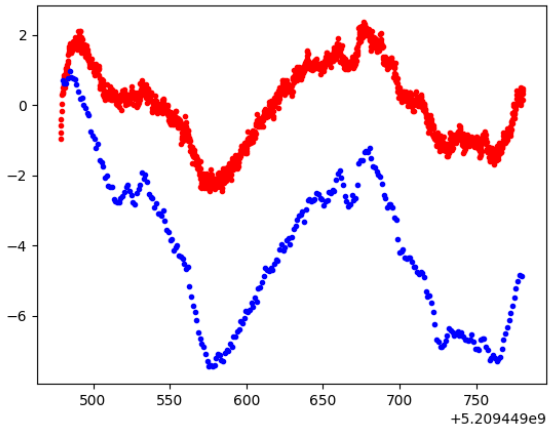


Figure 23: Comparison of phase delays derived from our dPWV proxy, as calculated from the differential absorption in the science SPWs (red), and those obtained via wvrgcal() from the original WVR data in SPW 4 for antenna DV10 (the reference antenna). X-axis is time in MJD seconds, phases at Y-axis in rads.

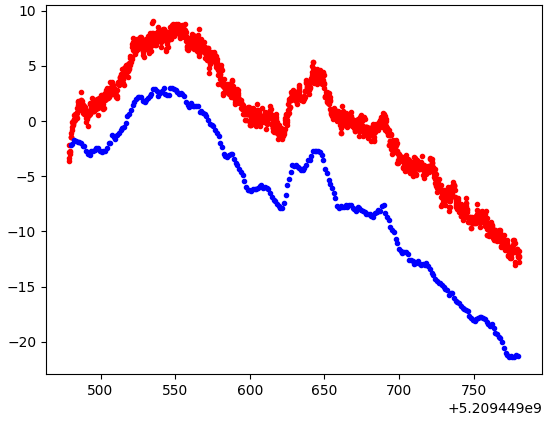


Figure 24: Like the figure on the left, now for antenna DV17.

In addition to comparing merely phase solutions as produced by our dPWV and *wvrgcal()* (the figures above) we would like to compare efficiency of both approaches in the phase correction. To that end we plot (i) the uncorrected phases as found by gaincal(), (ii) phases corrected by the wvrgcal() gaintable, and (iii) phases corrected via our dPWV proxy in a single figure. We prepare the corrected datasets as follows: (a) WVR-based correction of the BP-calibrator data (via applycal() of the wvr-caltable) prior running gaincal() on the same source, and (b) our dPWV-based correction to the uncorrected phases (we have no gaintable so far) by the phase subtraction. The procedure is implemented by routine *phiFixCompare()*.

In order to perform the WVR correction to the raw MS (solar MSs are *not* WVR corrected in the standard calibration procedure), we have performed the following steps prior call to *phiFixCompare()*:

```
clearcal('uid___A002_X110fd13_X1099e.ms',field='1')

wvrgcal(vis=vis,caltable='wvr_gcal.W',      segsource=True,      sourceflag=['Sun'],
statsource='J1256-0547',refant='DV10',maxdistm=1.0,minnumants=3)

from casarecipes.almahelpers import tsysspwmap

tsysmap = tsysspwmap(vis = mso, tsystable = mso+'.tsys', tsysChanTol = 1)

gaincal(vis=mso,caltable=mso+'.ap_pre_bandpass/wvrFix',     field='1',     scan='5',
nsolint='1.92s',      refant='DV10',      calmode='p',      gaintable=[mso+'.antpos',
mso+'.tsys','wvr_gcal.W'],  gainfield=['','1','1'],  interp=['',  'linear,spline',
'nearest'], spwmap=[[],tsysmap,[]])
```

As usually, just a sample of the results is presented in Figs. 25 and 26 for two selected antennas, the full set is available on-line in the project repository at http://wave.asu.cas.cz/shared/ESO/final_report/.

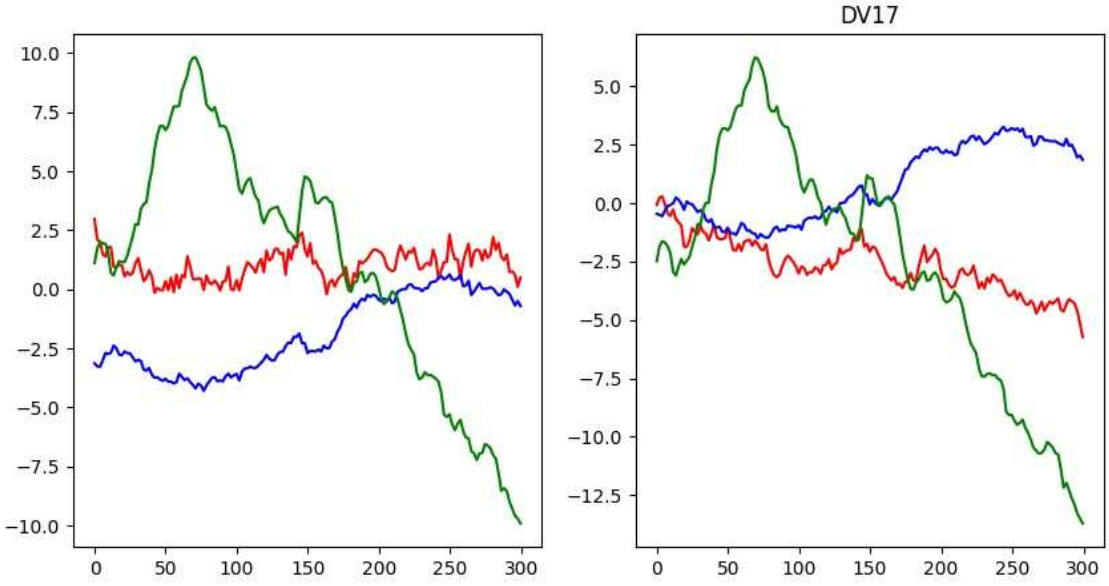


Figure 25: Comparison among (i) Raw phases for the band-pass calibrator – green, (ii) the same phase after being corrected by our proxy PWV method – red, and (iii) the phases after being corrected by applying wvrgcal() table – blue. This is example of well-working antenna DV17. It can be seen that the WVR-corrected phases may also exhibit long-term trends. Left panel: XX, right: YY. Time in secs form the BP-cal scan beginning.

What is interesting, and in line with the findings from our later analysis of another, low-PWV dataset (see below), is the relation between the raw phases (green) and their corrected residuals via PWV (red) and WVR (blue) methods in Fig. 26, left panel. Clearly, the PWV-corrected curve still exhibits features visible in the original (green) phases, so it means that we have subtracted a bit less than desirable. By

other words, our theoretical sensitivity coefficient, which relates dPWV proxy with the phase error (~25rad/mm for SPW 6) should be actually higher in order to do better job. And vice versa – the WVR corrected phase (the blue curve), which is in apparent anti-correlation with the dPWV-based correction exhibits 'over-correction' - i.e., the *dT/dL* coefficient for a given antenna, estimated in wvrgcal(), has been apparently a bit overestimated. We return back to this in the discussion of the low-PWV set in further paragraph.

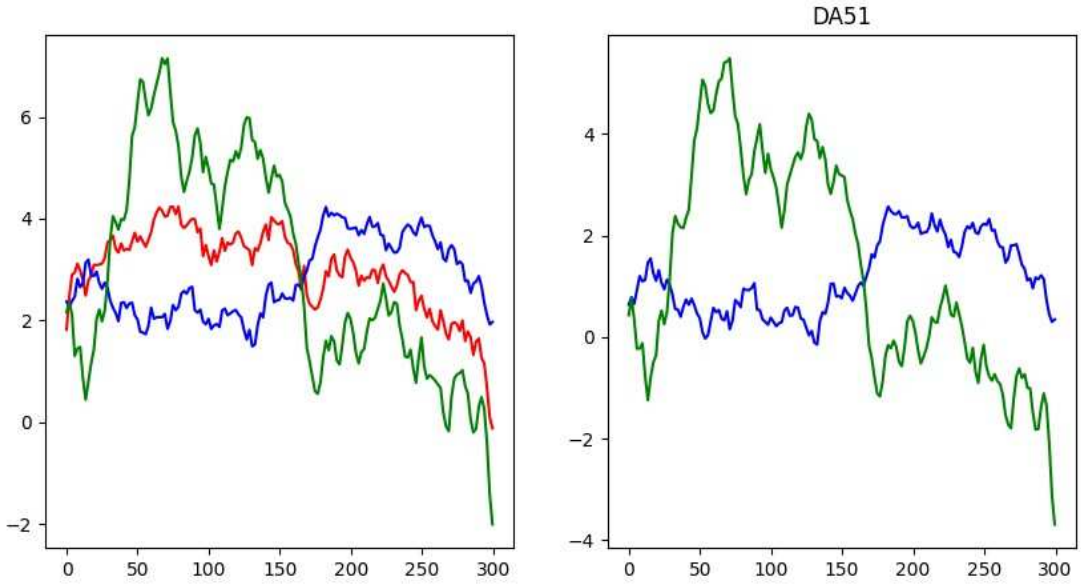


Figure 26: Like the previous figure, but now for antenna DA51. For YY pol. our PWV-based approach failed. Note the anti-correlation between PWV- and WVR-corrected phases for pol. XX (more in the text).

Ideally, the WVR-based phases and their dPWV-derived counterparts should be the same up to a constant offset (dPWV method corrects just the phase variations, not the time-averaged value over a scan). In fact, from the so far data it seems that difference between these two quantities exhibits systematic increase with time – see, e.g. Fig. 23. In order to study this systematically, we have performed similar analysis as we did above for time trends between dPWV and the phase delay, but now we have done mutual correlation between the phases corrected by two different methods: by our dPWV proxy and applycal() of the wvrgcal() table. Again, as we are allowing for the time-trend as an independent variable, we use our routine *paregression()* based on Eqs. (2.2.19) – (2.2.21). This time-detrending cross-correlation is implemented in the routine *dpwvXwvr_corrs()* in the sol_wvgcal.py collection.

Illustrative sample results are shown in Figs. 27 and 28, a table of the cross-correlation coefficients and time-trends (export form XLS) as well as the full antenna-set plots are available online in the project repository.

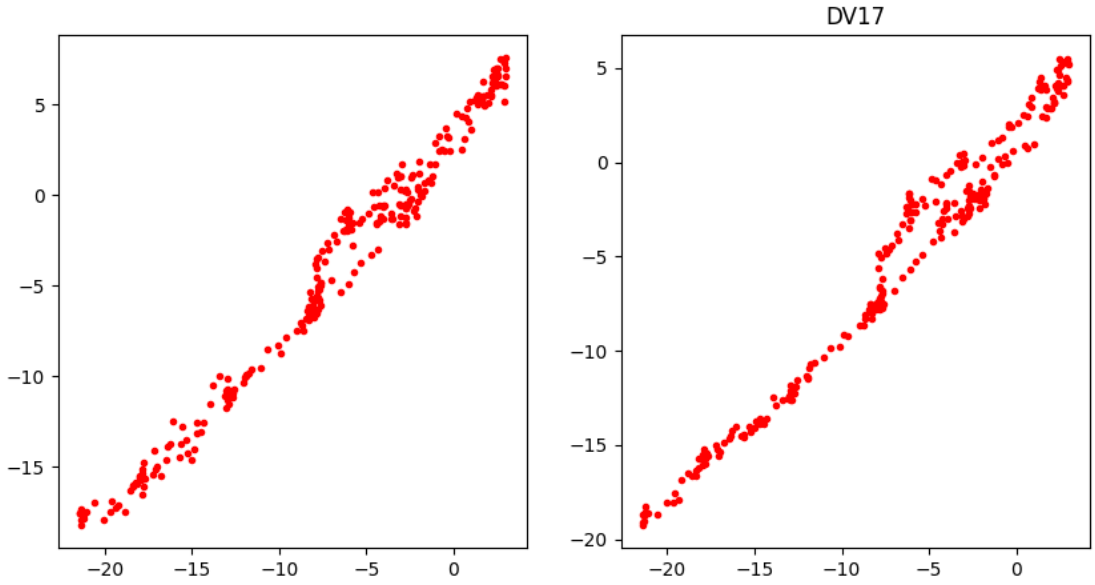


Figure 27: Cross-correlation plots between the dPWV- and WVR-corrected phases – the red and blue curves in Fig. 25 for the well-working example of antenna DV17. PWV has been calculated from channel-integrated SPW 6. Correlation coefficients for both XX (left) and YY (right) pols are very close to 1 (see the DV17 line in the XLS table online), time trend is small positive – meaning that the WVR-corrected phases grow a bit with time. w.r.t. the PWV-corrected ones (as is actually visible also in Fig. 24).

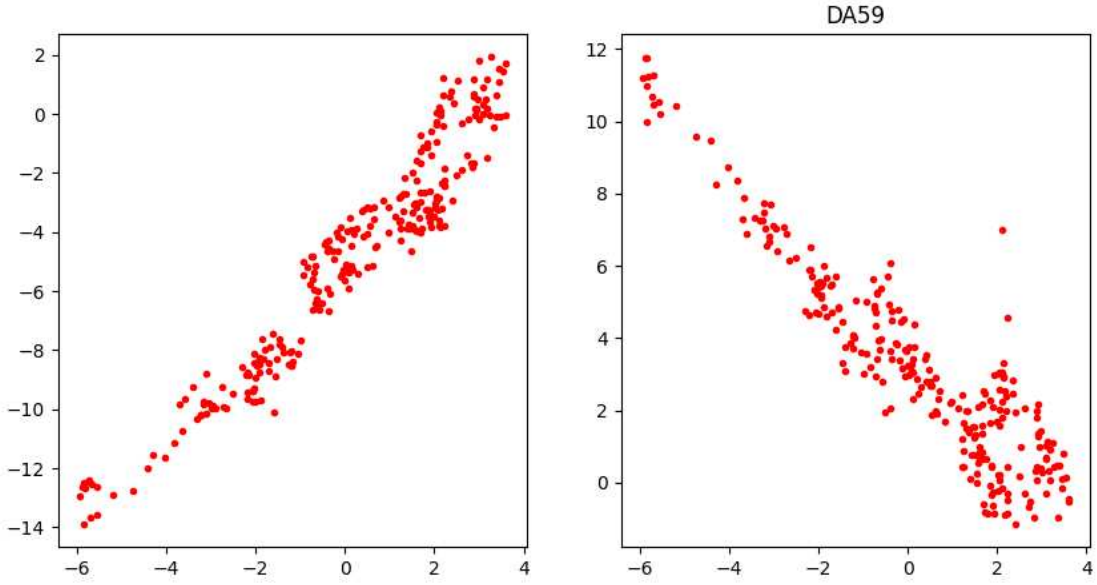


Figure 28: Like Fig. 27 above, now for antenna DA59. Clear anti-correlation is visible for the YY pol (right panel). Moreover, for the anti-correlated phases we have also negative time-trend.

From the results (see the XLS table online) it seems that

- WVR- and PWV-corrected phases are mostly (but not always) well correlated, either positively or negatively (anti-correlation).
- We have systematically small positive time-trend for phases corrected by WVR w.r.t. those corrected by dPWV (or, if we take the WVR as reference, then we can speak about systematic decrease with time for the dPWV-corrected phases) for the positive cross-correlation coefficient K (direct correlation), while negative time trend for negative K (anti-correlation).

We interpret this findings the following way: The factors, which are used for re-calculation of the water-vapor contents to the phase delays, i.e. our sensitivity factor *SF = -25rad/mm* for dPWV proxy and the dL/dT coefficients determined in WVRgcal() calibration, may be under- or over-estimated, which leads to under-/over-correction in the dPWV and WVR methods. This effect may lead to anti-correlation as already suggested above. For future development of our dPWV-based phase corrections we suggest to follow the WVRgcal approach, where dL/dT coefficients are also not based on the theoretical assumptions, but rather empirically found for each antenna separatelly. The same way we shall do it for our PWV-based method. We shall check the stability of the coefficient through the EB by establishing it not only for the band-pass scan but also for all phase-calibrator scans in the MS.

2.2.8 Comparison of different source SPWs

As we have mentioned in the beginning of this WV-based phase-fixing chapter, we have three options in selecting the primary data for the dPWV calculation: It can be either (i) auto-correlations in the channel-resolved or (ii) channel-averaged SPWs, or (iii) the SQLDs.
For the results shown in the previous paragraphs we have calculated dPWV from Eqs. (2.2.16) and (2.2.17) using the channel-averaged AC SPW 6. In order to test its robustness, we have repeated it with corresponding SQLD SPW 0, and also with channel-resolved SPW 5, but the results do not differ much.

For the SQLD we use the same equations as for the single-channel ACs, just we have to take into account the non-zero SQLD offset value P_o. For the channel-resolved SPW 5 we need to use directly the Eq. (2.2.14), which is implemented in method *dPWV_spwPart()* in our collection *sun_wvgcal.py*. For now, we can define the band-pass BW, over which the integration goes, just as a continuous band from *startChan* to *stopChan* in that method.

We applied it again to the BP calibrator of our solar Band 5 dataset. Somehow arbitrarily we have chosen integration in Eq. (2.2.14) over the lower half of the SPW5, i.e. next to the WV line core, where we can expect highest sensitivity of the power variation to fluctuating water-vapor contents. We also skip the very edge channels 0 – 7, as usually. Main motivation for using the channel-resolved SPW lies in the two facts: (i) we can select just the part of the SPW closer to the WV line and increase sensitivity, and, (ii) we can avoid the other weak atmospheric lines like O_3, present in the SPWs 5 and 6, which ‘contaminate’ the WV signal by the dry component.

The applicable result – comparison between the raw, dPWV-corrected (now dPWV being calculated by the alternative *dPWV_spwPart()* method), and WVR-corrected, data analogical to that shown in Fig. 26 (where we used the single-channel SPW 6), is presented in Fig. 29. As one can see, there is no significant difference to the 'traditional' method that uses channel-averaged SPW6. This holds for all antennas, as can be seen in the full set of figures available on-line – there are differences, but very little, at least for the given dataset. In any case, expected improvement due to better sensitivity implying from using just the highly-absorbing part of the SPW, did not come true. Comparison between single-channel ACs and SQLDs is available in the online repository, too. Also in this case the differences are negligible.

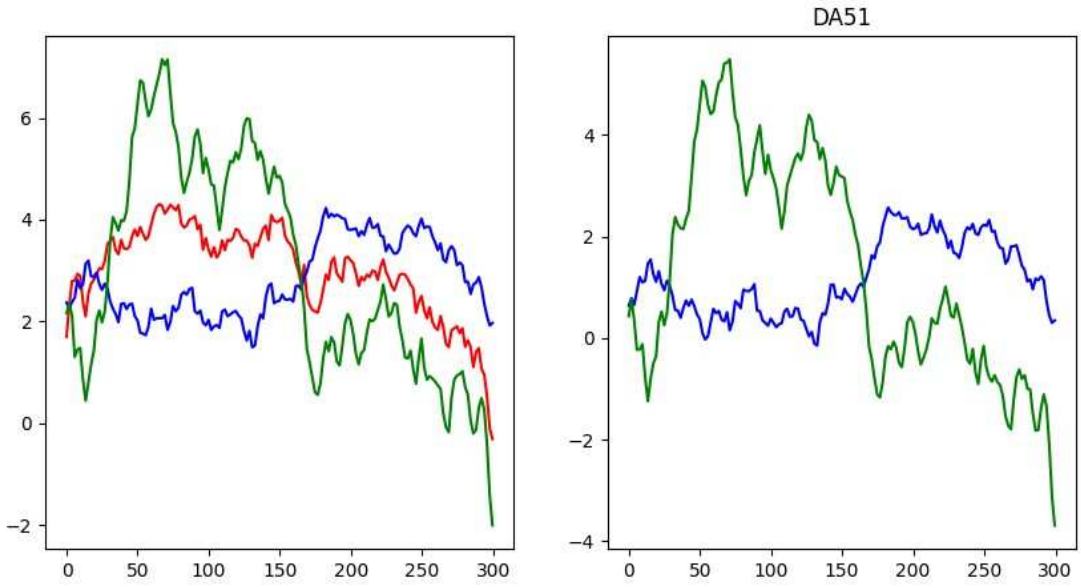


Figure 29: Like in Fig. 26: Green uncorrected phases, blue: corrected by WVR, and red: corrected by our dPWV proxy. Now the dPWV is calculated by Eq. (2.2.14), i.e. via integration over just a part of the spectrally-resolved SPW5. Displayed for the same antenna DA51 as in Fig. 26, for easy comparison.

2.2.9 Testing the approach with different dataset with low median PWV

Band-pass calibrators are present in basically all ALMA interferometry datasets and thus this analysis can be repeated for a MS with similar spectral setup and different, i.e. much lower, median PWV.

By the archive search we have found an observation, which fulfills the following conditions: (i) it has very similar frequency coverage for the SPW closest to the WV line – the lowest SPW being between 190 GHz and 192 GHz, like our solar dataset, (ii) observation is continuum, optimally in TDM regime (or FDM with low frequency resolution, covering the full base-band), and (iii) has low PWV, below

0.8mm. By this criteria we selected project 2021.1.00306.S, uid___A002_Xf8b429_X8ef3.asdm, science target IRAS_F14348-1447, observed on May 10, 2022 by 12m array.

We had to generalize our routines in sol_wvgcal.py in order to apply them to the band-pass calibrator for the above MS. Namely, the SPWs, for which the ATM scan has data for ACD calibration (hot, ambient, and sky sub-scans) are not the same as the science SPWs; we had to use the T_{sys} *spwmap* for their correct identification.

Before analysis, the *scriptForCalibration.py* has been generated via standard script generator

```
execfile('QA2/science/analysis_scripts/almaqa2csg.py')
generateReducScript(asdm_name)
```

just with *asis=CalDevice, CalAtmosphere* parameters set on top of the 'standard' settings (without that the ACD procedure fails on not having the ambient and hot-load temperatures). The calibration has been run up to its *Step 6* (antpos, Tsys and WVRgcal calibrations), skipping the AC-flagging step for obvious reason.

Then, we have run our *phiFixCompare()*, like in the step described above for the solar MS – as a result we got, again, the raw BP-calibrator phases (green curve in the following figs), and their reduced versions corrected by (i) WVRgcal (the blue curve), and (ii) our dPWV-proxy method (red curve), respectively. The straightforward application of this procedure led to a strange result. It seems that for the channel-integrated AC SPW 26, from which we have calculated the dPWV proxy, there is complication with the ACD calibration: The procedure reports T_{sky} temperature calculated by Eq. (11) from gain G and receiver temperature T_{rx} derived from our ACD calibration scan to be 3000K (!). This is far beyond our expectation – the T_{sky} for the ATM scan (i.e., OFF_SOURCE) shows temperature around 130K and the calibrator itself has its flux density only 1.6 Jy. When taking into account the typical Jy-per-Kelvin for the 12m antenna being ~42, the source should cause increase of the temperature by a small fraction of Kelvin (~40mK). Since the corresponding channel-resolved SPW 25 has 960 channels and different integration time than the SPW 17, which was identified as being adjunct to it by T_{sys} spwmap (has only 128 channels), one can understand that the gain should be perhaps re-scaled by the factor corresponding to ratio of integration times and channel widths.

In order to include this gain-scaling uncertainty, we tried the same procedure *phiFixCompare()* but now with the gain G arbitrarily down-scaled by a factor of 50, which gives us realistic T_{sky} for the BP-cal on-source scan. The result follows in Fig. 30.

The situation after gain-rescaling significantly improved, at least for some antennas – for the case of DA56 we can really speak about "correction" – the red curve (dPWV-corrected phases) has less variations than the uncorrected phases in green.

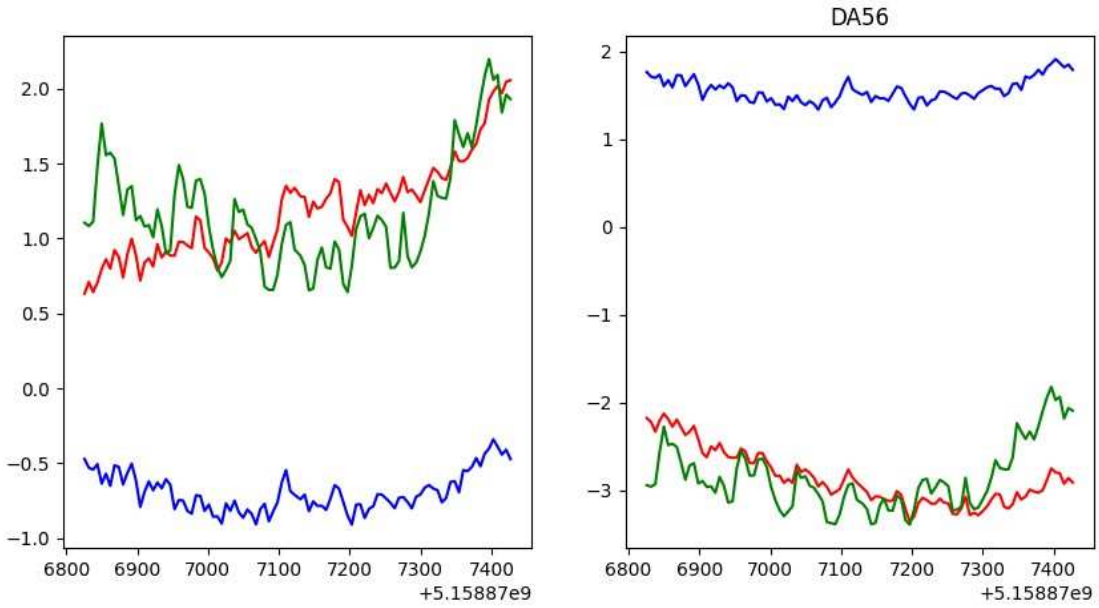


Figure 30: Like Fig. 25, now for the the low-PWV dataset (IRAS). The gain has been arbitrarily re-scaled by a factor 50, in order to fit the sky temperature for the source. Displayed antenna is DA56.

We have tried it yet for the SQLD windows, where the ATM scan and science scan SPWs have the same resolution 128 channels. However, the T_{sky} temperature after the dual-load calibration is 300K, so lower, but still unexpectedly high. The results of application of our procedure, without the gain scaling are similar as for AC SPWs. We have to admit that the transfer of gain and T_{rx} from the ATM scan in T_{sys} SPWs to the corresponding science SPWs (via *spwmap*) for this MS is unclear and needs further investigation.

Nevertheless, in spite the confusing results, still it looks that there is a correlation between dPWV and phase variations, just we need different factor for transforming it into the phase difference. For this dataset, the change of the correction factor may be related not only to the (unknown) gain re-scale between T_{sys} and science SPWs, but also it is likely to attribute at least part of it to influence of the dry component of the phase variations for this MS with ultra-low PWV.

2.2.10 Concluding remarks to this topic

We have developed the legacy idea of Richard Hills (2019) and derived a proxy measure dPWV (in mm) to the water-vapor contents in the atmosphere. This can be re-calculated to the phase delays; the scaling coefficient for the SPW 5 (the closest one to the WV line) in our solar MS is ~25rad/mm.

The original Richard's formula derived in the Annex 3 to the MTR can be in our notation written as

$$\frac{\delta T_{sky}(t)}{\langle T_{sky} \rangle} = -\tau_1^{mean} \cdot dw(t)$$

It has been derived for the limit, where T_{RX} and T_{atm} are negligible when compared to the brightness temperature of the Sun. This, however, is not fulfilled in our data as can be seen form Figs. 11 and 12 - the Trx can reach the values around 1000 K while the solar brightness is attenuated (by WV absorption) in the low-frequency part of the SPW 5 down to 3500K. Nevertheless, it can be easily shown that in the limit $T_{inst} \to 0$, $T_{atm} \to 0$ our Eq. (2.2.14) when applied to reasonably narrow bandwidth BW exactly matches the original formula above.

It has been found in the clear test case of the BP calibrator that the correction based on the dPWV works actually quite well and qualitatively it is comparable with that based on *wvrgcal()*. There are still some problematic antennas, for which the results are less convincing. Most likely reason is to be found in the errors of the ACD/dual-load calibration: The denominator of formula (14) contains a difference of two comparable and relatively large quantities, which both depend on gain or instrumental (noise) temperature. Even a small relative error in determination of both is thus strongly amplified, the resulting dPWV is quite sensitive to them.

In this respect, further development of the method is needed. We intent to

- Use also the phase calibrator scans and find the scaling factor between dPWV by regression over all scans of the MS, similarly as wvrgcal() does for its dT/dL coefficients.
- For the sources, where we have both *wvrgcal()* and our dPWV signal we may consider the parameters G and T_{rx} in our formula (2.2.14) as unknown and try to find them independently of the ACD/dual-load calibration by mutual fitting of the *dPWV* and *wvrgcal()* signals. These modified instrumental-calibration parameters will be then used also for improved T_{sys} calculation. This application might be of interest to non-solar community, too. Note, that for the case of solar data we have been able to find corrections to G and T_{rx} by solving an extended optimization task, which involves both the science and ACD scans data – see details below in Section 2.3.3.12.

2.3 Science SPW-based PWV corrections for the solar data

In the previous Chapter 2.2 we have described the procedure for differential (i.e., w.r.t. its mean value averaged in time over a scan) PWV changes, called *dPWV-proxy,* and successfully demonstrated its application on the band-pass (BP) calibrator. The method is based on measurement of signal variations in one science SPW, usually that closest to the WV line center, as it is most sensitive to the PWV variations. Depending on whether we are using (i) the channel-resolved auto-correlation (AC) SPWs (spws 5,7,…) or, (ii) the single-channel AC (spws 6,8,…) or SQLD SPWs (spws 0,1,...), we use expressions (2.2.14) for channel-resolved SPWs, and Eqs. (2.2.16) + (2.2.17) for channel-integrated SPWs (both AC and SQLDs). It has been shown that the results do not significantly differ – AC resolved/integrated and SQLD spws provide very similar dPWV(t) profiles which, moreover, correlate well (at least for part of antennas) with both raw BP-cal phases and with the *wvrgcal()* caltable.

For the Sun, however, such a simple procedure may be risky, because the signal variation can be caused by (i) varying absorption – i.e. truly PWV related, (ii) intrinsic source (the Sun) brightness variations or, (iii) apparent variations of the solar brightness caused by lensing effects of the atmospheric turbulence. For the BP calibrator we could neglect the last two contributions because the single-dish (i.e., AC or SQLD) sky brightness is far dominated by the terrestrial atmosphere emission. In the case of Sun this, however, need not to be true – the contribution (ii), the intrinsic variation of the solar brightness, would be for sure present in case of solar flares (and one such has been successfully observed in May 2024 by White et al.), and the contribution (iii), caused by focusing/defocusing effects of the bubbles in the terrestrial atmosphere to the incoming solar radiation, is, to some extent, present always – we devote specific section to it in further text below. In other words, for the Sun we cannot neglect the last term in Eq. (2.2.9) and we have to take it into account.

Well, do we have ever a chance to distinguish between those three contributions? Fortunately yes. The keys are following:

1. Frequency dependency: Variations of atmospheric absorption caused by varying PWV increase as we approach the WV line center, while the solar brightness variations – both intrinsic (flaring) and apparent (atmospheric lensing) – are more prominent with more transparent atmosphere, i.e. further from the WV line.
2. Antenna dependency: Intrinsic source variations (flaring at the Sun) are the same at all antennas, while the apparent brightening/darkening (atmospheric lensing) is randomized per antenna.

Based on that we have developed the improved procedure described in detail below.

2.3.1 Theory

We start with Eq. (2.2.9) in its full form, including the source-brightness variations, both intrinsic and apparent as these two are not distinguishable with just one antenna. Moreover, now we do not split the total opacity $\tau(f)$ into dry and wet parts $\tau(f)= \tau_d(f)+ \tau_1(f).w$, where just the wet part has been insofar considered as varying due to temporal dependence of $w(t)$. Instead of that, we consider general split of the opacity into scan-averaged and varying part $\tau(f,t)= <\tau(f)>+ d\tau(f,t)$. The equation (2.2.9) then reads

$$\overline{T_{sky}}(t)-\langle\overline{T_{sky}}\rangle=-\frac{1}{BW}\int_{spw} d\tau(f,t)\left[\langle T_{sky}(f)\rangle-T_{atm}\right]df+\frac{1}{BW}\int_{spw}\delta T_{src}(f,t)\,\mathrm{e}^{-\langle\tau(f)\rangle}df \qquad (3.1.1)$$

where, as in previous reports, the <>-brackets mean time average over a scan and over-bar denotes frequency averaging over the BW band pass (typically an SPW or its part).

Now, in order to move ahead, we have to assume something about the source variation $\delta T_{src}(f,t)$, namely we decompose its temporal ans spectral dependence as

$$\delta T_{src}(f,t)=F(f)\cdot\delta T_S(t) \qquad (3.1.2)$$

where for spectral profile we assume (for now) a power-law in the form

$$F(f)=\left(\frac{f}{f_o}\right)^s \tag{3.1.3}$$

The spectral index *s* shall be *s=2* for the Rayleigh-Jeans approximation of thermal radiation and f_o is the reference (central) frequency.

This is going to be a 'zero-order' approximation if the solar flaring activity at THz/sub-THz waves is of the thermal nature. However, in the following text (gradual improvements of the methodology) we have 'converged' to more realistic thermal-radiation spectrum, based on the paper by Alissandrakis et al. (2022). Discussions on thermal vs. gyro-synchrotron (GS) mechanism for solar bursts at mm wavelenghts are lively and ongoing in the solar community – some authors suggest strong GS component with completely different spectral index. Anyway, we have nothing better so let us the spectrum in the form above for now.

If we apply the standard ACD calibration technique (Eq. 2.2.11), we can express the sky temperatures in (1) in terms of the directly measured powers. Including source-variation assumption expressed here as (2) and (3), and assuming (as we did insofar) the opacity variations to be connected solely with the PWV dynamics, i.e. $d\tau(f,t)= \tau_1(f).dw(t)$ (this will be yet revisited in below) we can re-express Eq. (3.1.1) here as

$$\int_{BW}\frac{1}{G(f)}\cdot\left(P(t,f)-\langle P(f)\rangle\right)df=-dw(t)\cdot\int_{BW}\left(\frac{\langle P(f)\rangle}{G(f)}-T_{inst}(f)-T_{atm}\right)\tau_1(f)\,df + \delta T_S(t)\int_{BW}\left(\frac{f}{f_o}\right)^s\cdot e^{-\langle\tau(f)\rangle}\,df \tag{3.1.4}$$

Here, T_{inst} is the total 'instrumental temperature' – essentially T_{RX} generalized by the offset P_o for the SQLD SWPs:

$$T_{inst}(f)\equiv\frac{P_0}{G(f)}+T_{RX}(f)$$

Eq. (4) is the central formula for further calculations. We have two independent variables *dw(t)* and $dT_s(t)$ but since (3.1.4) can be expressed for all SPWs, we can form even over-determined set of linear equations for those two variables in every single time instant. In practice, in order to improve SNR, we use 12 SPWs: 4 channel-resolved ACs, 4 channel-integrated ACs, and 4 SQLDs. Eq. (3.1.4) for those 12 SPWs represents the linear system *A.x=C*, where row *i* of the matrix *A* and right-hand side vector *C*

$$A_{i1}\cdot x_1+A_{i2}\cdot x_2=C_i$$

represents the formula (3.1.4) for a given SPW *i; i* ranges from 1 to 12. The components of matrix *A* and the RHS-vector *C* are thus written as

$$C_i = \int_{BW} \frac{1}{G(f)} \cdot \left(P(t,f) - \langle P(f) \rangle \right) df ,$$
$$A_{i1} = -\int_{BW} \left(\frac{\langle P(f) \rangle}{G(f)} - T_{inst}(f) - T_{atm} \right) \tau_1(f)\, df , \qquad (3.1.5)$$
$$A_{i2} = \int_{BW} \left(\frac{f}{f_o} \right)^s \cdot \mathrm{e}^{-\langle \tau(f) \rangle}\, df$$

and the unknown 2-D vector is $x=(dw, dT_s)^T$; here the superscript '*T*' denotes matrix transposition (making *x* a column vector). The system (3.1.5) is then solved for every time sample in the on-source sub-scan of the solar scan. As already adduced above: The different frequency dependence of A_{i1} and A_{i2} [which represent the sensitivity factors for PWV variation *dw(t)* and total (intrinsic+apparent) source brightness dynamics *dTs(t)*] assures that the columns of the matrix *A* are independent vectors (for the SPWs occupying different frequency domains they are far from co-linearity), so the system (3.1.5) is well conditioned and can be solved.

Since the system (3.1.5) is over-determined (we have 12 equations for 2 variables), we use the least-square approach (minimizing the residual $(A.x-C)^2$), what leads to the system of so called 'normal equations':

$$A^T.A.x = A^T.C \qquad (3.1.6)$$

In our case it contains 2x2 matrix $A^T.A$ and 2-D vectors $A^T.C$ and *x*, so we have a unique solution for x.

To sum up: Solving the system (3.1.6) in each time instant *t*, we are able to decompose signal variations measured in science SPWs into: (i) variations of the PWV – the *dw(t)* time profile, and (ii) apparent+intrinsic brightness dynamics of the Sun (actually its part exposed in the antenna PB) – the *dTs(t)* time series.

Entire procedure is implemented in a new routine *dWdB()* in the *sol_wvrgcal.py* collection attached as Annex 4.

2.3.2 Early results

We have applied the method to our December 2023 test data – project 2022.1.00031.CSV, SB Sun_10_a_05_INT. First, we have been interested in quality assessment of the procedure – namely the residual error in solution of the system (3.1.5). Once we have the solution vector $x=(dw, dT_s)^T$, we can calculate the difference $R_i = A_{i1} \cdot x_1 + A_{i2} \cdot x_2 - C_i$ representing the residuum for SPW with index *i*. An example for antenna DV12 is shown in Fig. 31. Left and right sides represent results for XX and YY, respectively. The upper panels show the residua R_i for SPWs 5,7,9,11 (red, green, blue, black solid line), 6,8,10,12 (red, green, blue, black dashed), and 0,1,2,3 (red, green, blue, black dotted). As it can be seen, the residua are comparable for all SPWs in this case (in just a few antennas some SPWs are problematic, far most of antennas are good) and reach low units of Kelvins (except of a few peaks they lay within -3 K to +3 K limit). The residual error thus represents just a small fraction of the total sky brightness variations that are displayed in the middle row of panels and which oscillate between -100 K

and 100 K: We plot the right-hand-side vector C_i from Eq. (3.1.5) normalized to the band-width BW, over which we integrate, in order to get the SPW-averaged T_{sky}; the color-coding for SPWs is the same as in the residuals displayed in the top row.

The bottom row of panels shows, how much to the total sky-brightness variations is contributed by the apparent source brightness variations, i.e., the time series $dT_s(t)$ found as component x_2, modified by the sensitivity factor A_{i2} (which has a meaning of average transmissivity weighted slightly by the Rayleigh-Taylor frequency scaling) – we plot $A_{i2}.x_2$ for each SPW with index i, using the same color-coding as above. As it can be seen, the brightness dynamics of the source (apparent + intrinsic) represents smaller but already non-negligible fraction of the total signal variations, namely it contributes to its ultra-short

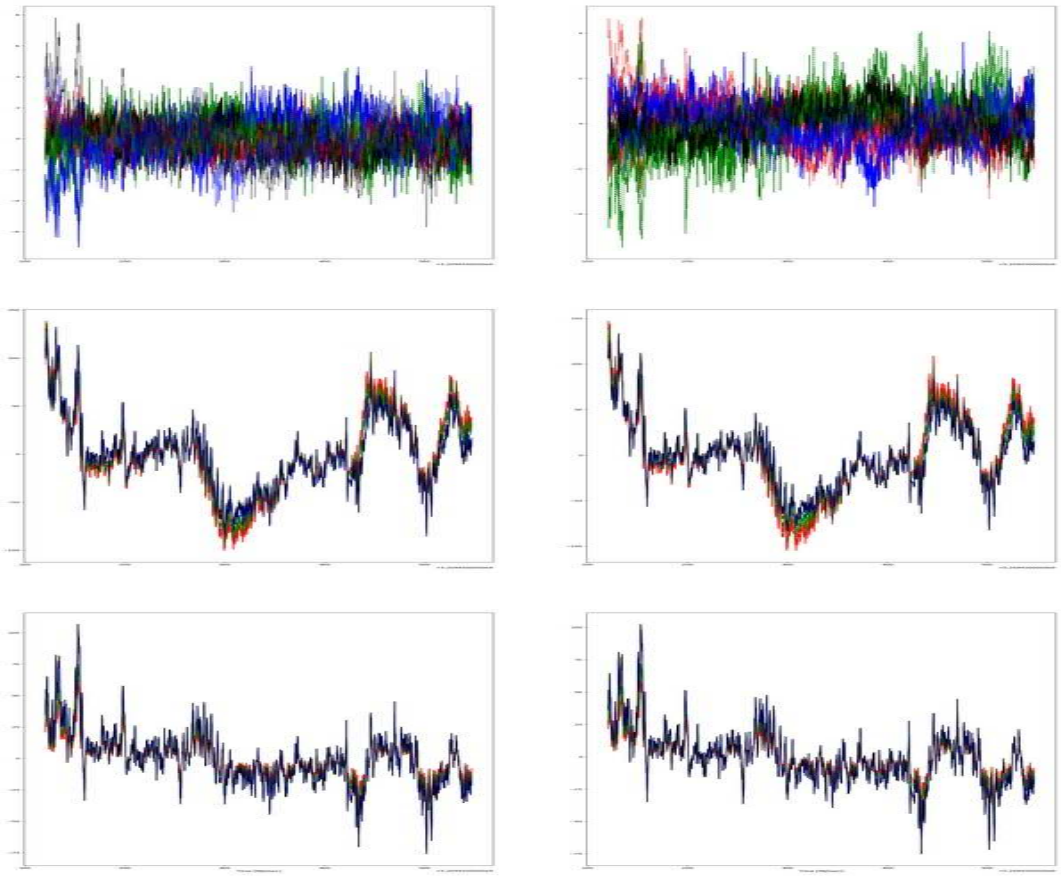


Figure 31 Residuals (top row), total sky brightness variations (middle) and source brightness contribution to the total sky brightness variations as extracted by solving the system in Eq. (5). Left and right columns were obtained from XX and YY pols in science SPWs.

and intense peaks, like that at around time 150 (+5.20945x10^9) MJD secs. Signal in all SPWs is 'synchronous'. Figures for all antennas are available on-line as *Annex 1*.

As the procedure seems to be fine – residuals are much smaller than any measured or derived quantity and all the SPWs show very similar results (the profiles are synchronous), we now check the solutions itself. Fig. 32 shows decomposition of the signal to the source-brightness variations $dT_s(t)$ in left panel and the PWV variations *dw(t)* in right panel. Blue and red colors correspond to XX and YY polarizations.

When the new PWV variation profile is compared with the same quantity but derived using the old, simple single-SPW approach according to Eq. (2.2.14), they are actually very similar – only significant difference is in the ultra-short high peaks (spikes) that are present in the single-SPW dPWV(t) curve, while not in the new one in Fig. 32 (right). Clearly, these spikes have moved into the left panel – they represent the source (apparent) brightness spikes, not a sudden, significant and at the same moment short-living (which is unlikely) PWV enhancements.

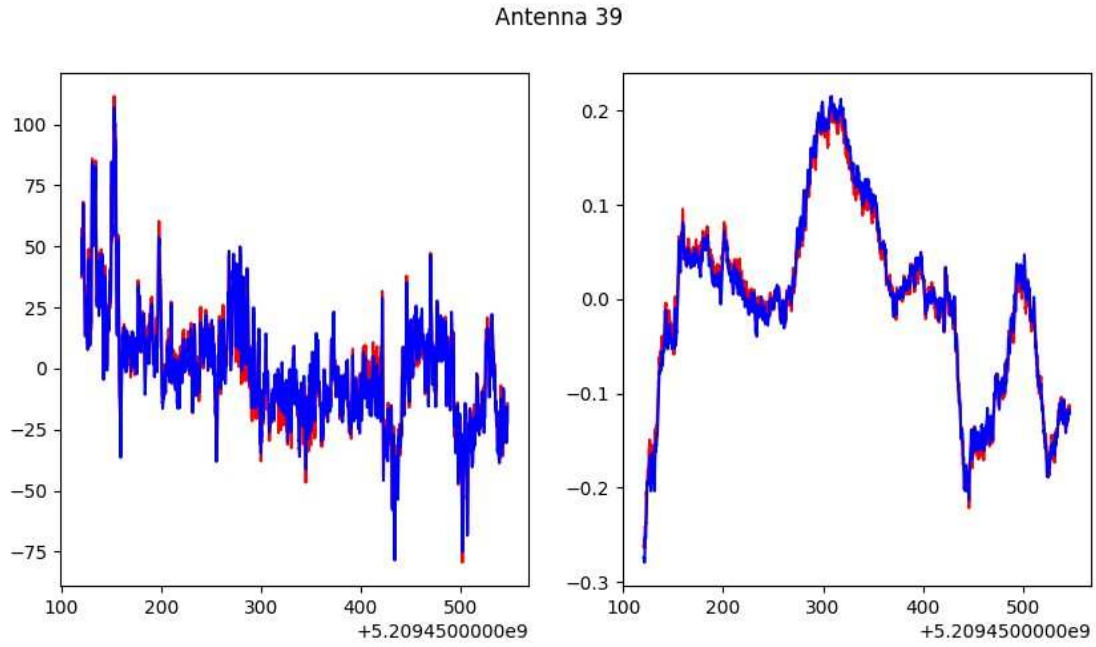


Figure 32: Source brightness dynamics (left, in Kelvins) and PWV variations (right, in mm) for antenna DV12. Blue and red curves are for XX and YY pols. Time at the x-axis is in MJDsecs.

Inclusion of the source-brightness fluctuations explains yet another observation that was a bit mysterious to us insofar: If we use the single-SPW calculations by Eq. (2.2.14), it is clear that the signal (T_{sky}) variations are linearly proportional to the PWV variations, in any considered SPW. The proportionality coefficient (which we call 'sensitivity factor' from obvious reason) clearly differs from SPW to SPW, the highest being in the SPW closest to the WV line. Hence, if we make a scatter plot between measured T_{sky} variations in, say SPW 5 and SPW 11, we would expect a straight line with the slope that corresponds to the ratio of those sensitivity factors. Such a scatter plot is displayed in Fig. 33 – x-axis shows T_{sky} variations (in K) in SPW 5 (closest to the WV line) while the y-axis T_{sky} variations in

SPW11 (farthest from the WV line). We have used continuous line (the red curve) instead of single dots in the scatter plot in order to trace the dynamics – the line connects the points that are consecutive in time.

What is eye-striking in the red curve that it looks like there are two effects working at two different timescales: The fast variations with the steeper slope, and the slower dynamics with the overall milder slope. Our first interpretation was in terms of the PWV dynamics – that it may have (i) the fast part related to fast compression/dilution of the atmospheric bubbles (as the total air pressure changes, it changes also its partial pressure of the water vapor), and, (ii) the 'slow mode' related to slow changes of water-vapor relative abundance (humidity). However, searching for decomposition into pressure and humidity variations failed (we returned to this topic yet in the discussion further). Nevertheless, inclusion of the *dTs(t)* dynamics changes the game – the fast variations may be interpreted as the apparent brightness dynamics of the source. And indeed: We have over-plotted the red scatter curve with two straight lines: The green one corresponds to ratio of matrix coefficients $A_{11,1}/A_{5,1}$ i.e. the sensitivity factors for the *dw(t)* variations, while the blue one corresponds to ratio of matrix coefficients $A_{11,2}/A_{5,2}$ i.e. the sensitivity factors for the $dT_s(t)$ variations. Clearly, the blue line follows the fast dynamics of the red curve while the green one corresponds to the slow 'global drift'.

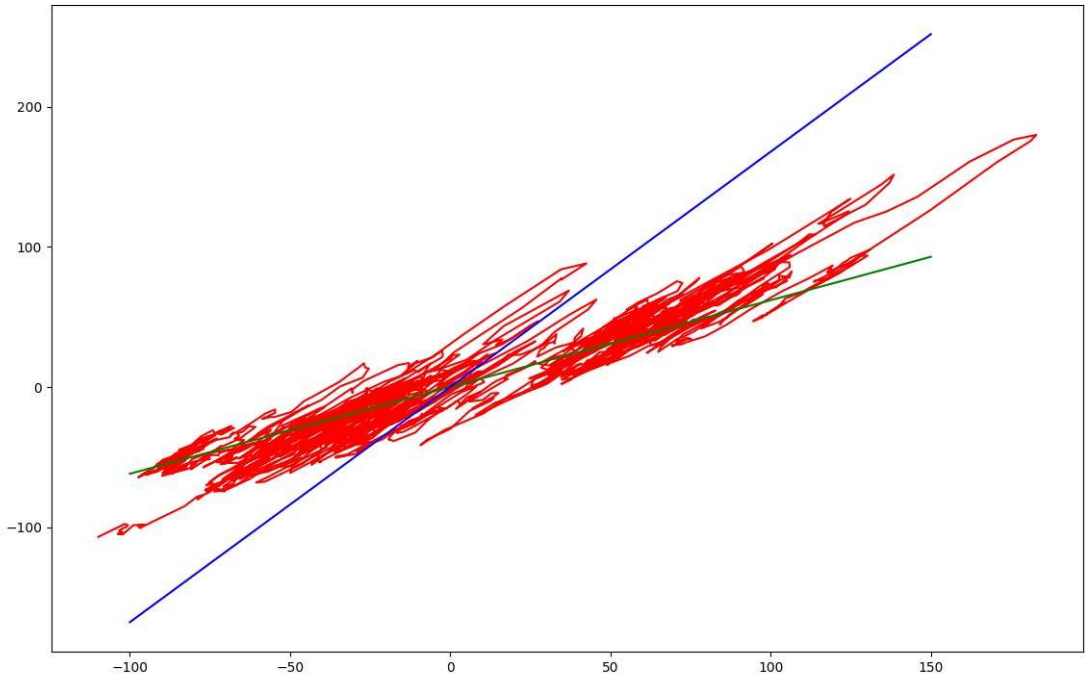


Figure 33: Red curve: Scatter plot (connected points with consecutive times) of Tsky variations in SPW 5 vs SPW 11. Green and blue lines: Ratio of matrix coefficients (sensitivity factors) for dW and dT, respectively (see details in the text).

Conclusion: The signal measured as T_{sky} variations in science SPWs for the Sun has two components: Slower one, which corresponds to PWV dynamics over a given antenna, and fast one related to the apparent brightness variations of the observed part of the Sun.

In order to increase our confidence in this interpretation, we can do a similar scatter plot for the BP calibrator. There is no source (neither intrinsic nor apparent due to atmospheric lensing) variation expected because the emission in ACs and SQLDs is far dominated by the terrestrial atmosphere. If, on the other hand, the slow/fast component splitting was related to humidity/pressure variations (i.e. purely atmospheric effect), we should see it in the BP cal, too.

As it can be seen, there is, indeed, no fast component present, except of the random noise, and the only linear structure in the figure is the 'global drift' that follows the green line corresponding to the ratio of sensitivity factors for the PWV variations.

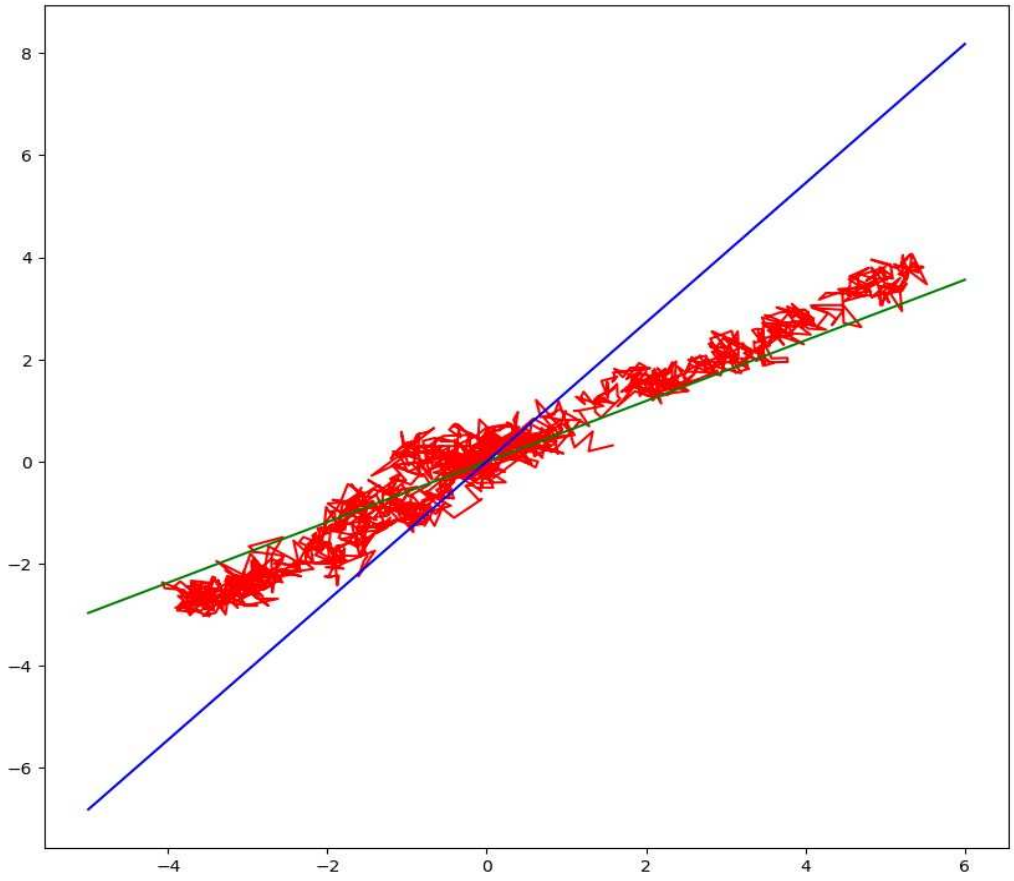


Figure 34: Like in Fig. 33, now for the BP calibrator (note the much smaller ranges of Tsky variations on both axes), variations in SPW 8 vs. SPW12

As already described above, the decomposition of the two components – PWV variations and the T_{src} dynamics – are possible only thanks to different frequency dependence of the sensitivity factors for *dw(t)* and *dTs(t)*. This different frequency functions lead to independence of columns of the matrix *A* in the system (5). Graphical illustration of that fact is visible in the Figs. 33 and 34 as the angle between the green and blue straight lines: The bigger is the angle, the more exact we have the decomposition (better conditioned matrix).

2.3.2.1 Nature of the source brightness variations

We have done the decomposition into *dw(t)* and *dTs(t)* for all antennas – see *Annex 1* online. Already at the first look it is clear that the *dTs(t)* is significantly different for different antennas. Hence, its main component hardly can be connected with the intrinsic brightness variation of (the observed part of) the Sun. Most likely interpretation lies in atmospheric lensing – the turbulence eddies have focusing and de-focusing effects on the transmitted solar radiation. If a focused-radiation 'hot spot' falls into an antenna, we record short and relatively intense brightening. Such an effect is well known in the long-wavelength radio observations, where the larger-scale traveling ionospheric disturbances (TIDs, basically 'bubbles' of enhanced electron density) may even cause 'fake' radio burst in ground-based receivers. These 'pseudo-bursts' draw characteristic features in their dynamic radio spectra and are known as *spectral caustics (*e.g., Koval et al. 2019, https://iopscience.iop.org/article/10.3847/1538-4357/ab1b52 and references therein). We do not have time for deep investigation into this topic but as an indication we present Figs. 35 and 36, which show correlation between the *dTs(t)* and the time derivative of the *dw(t)*. The idea (not really developed in depth) is the following: If *dw(t)* changes rapidly, it means that the phase delay changes rapidly, too. Rapidly changing phase may indicate focusing irregularity in the atmosphere.

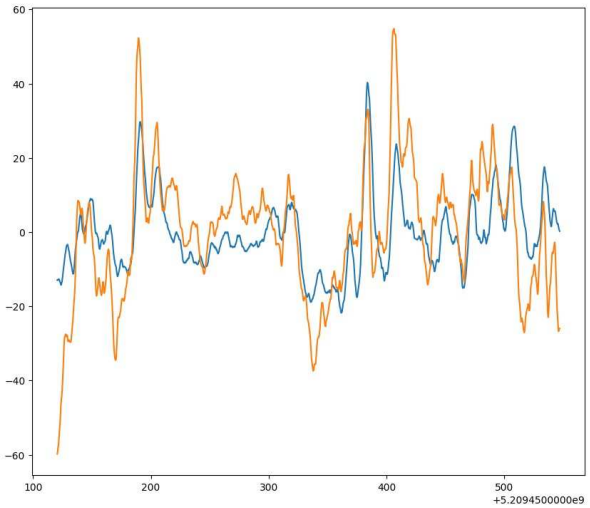


Figure 35: dw(t)/dt - smoothed and linearly scaled to the same range of values (orange) and dTs(t) (in blue) both as a function of time. There are apparent similarities in both curves.

Albeit, as written above, the idea is not really developed (and will not be, as it is off-topic for this study), the correlation between *dw(t)/dt* and *dTs(t)* indicates dominantly atmospheric origin of the T_{src} variations: If it was extra-terrestrial, there would be no reason why the two quantities should be related, because solution of the system (3.1.5). is performed in each time instant independently, so it 'knows' nothing about dynamics hidden in the *dw(t)/dt* derivative.

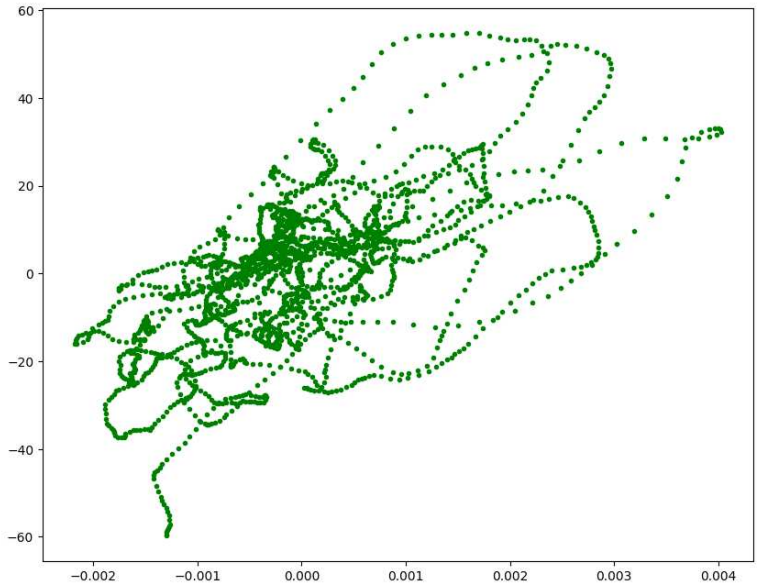


Figure 36: Cross correlation (scatter plot) between dw(t)/dt and dTs(t)

Anyway, we are interested in the question, whether there is also some intrinsic source brightness variation. The key, as it has been adduced above, is in the difference between *dTs(t)* time series for different antennas. If they contain just the random atmospheric component, the sum (or, better, the mean) of all *dTs(t)* profiles over entire array would tend to zero as the random variations tends to cancel each other. On the other hand, intrinsic solar brightness fluctuation should come out in the sum/mean *dTs(t)* curve.

We have tried it and the result is displayed in Fig. 37. Clearly, there is non-zero variation in both polarizations in units of Kelvins. Saying it is the intrinsic source T_B variation is too strong, the lensing component may have some amount of mutual correlation as the 'hot' and 'cold' spots on the Earth surface created by focusing/de-focusing effects are transported over the array by the mean wind. On the other hand, it is quite smooth and variations are not big, so its interpretation as the intrinsic source variability is plausible.

Anyway, whatever is the nature of the apparent source brightness variations, for this project it is important just one matter: We are able to separate them from the measured signal and thus produce more trustworthy the remaining cause of signal – the PWV variations.

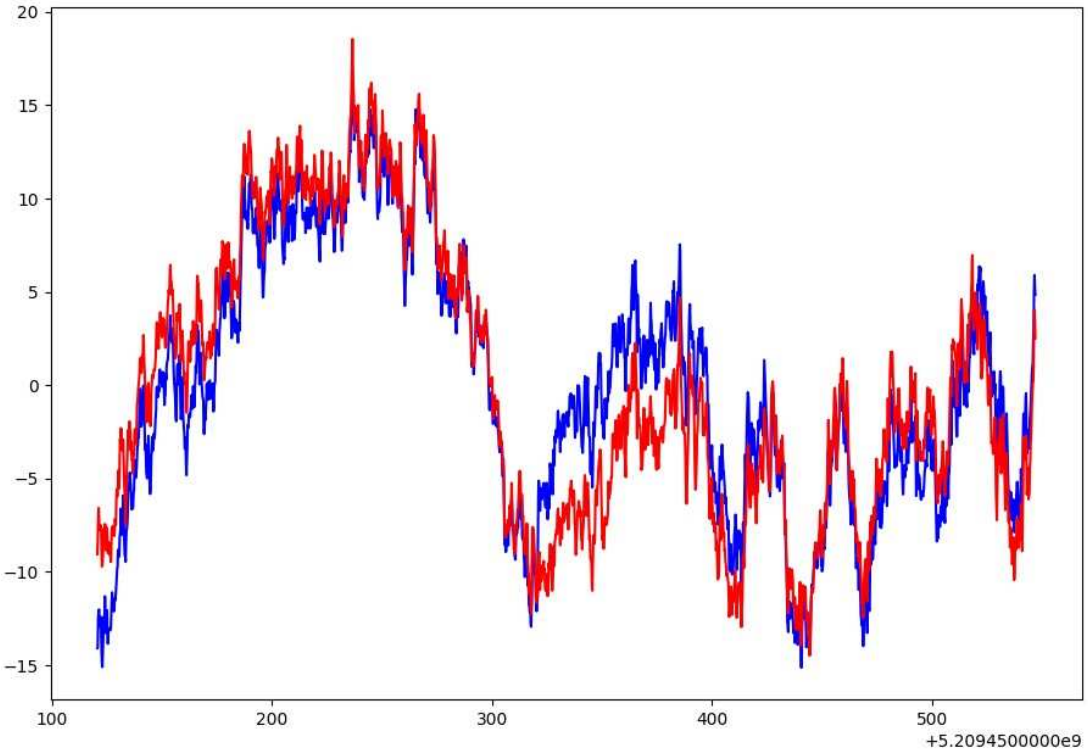


Figure 37: Sum of dTs(t) curves over all antennas (with exception of the two flagged): Red XX, blue YY, time in x-axis in MJDsec.

2.3.2.2 Discussion to separation of pressure/humidity or dry/wet components in dPWV variations

Before finding that the two components (slow and fast) with two different slopes manifested in the scatter plot in Fig. 33 are actually related to PWV variations (slow component) and dynamics of the (apparent/scintillating) source brightness (fast component), we have been thinking about interpretation in terms of varying relative humidity (slow) and total pressure/density of the air (fast). Alternatively, we may think about to making a decomposition into *dry* and *wet* component variations. Here we show that both decompositions (humidity/pressure and wet/dry) are actually equivalent. Nevertheless, as we will see, the results of an attempt to disentangle additional components in the signal are remaining negative.

Basically, the total PWV content *w(t)* should be proportional to integrated air mass (integral of air density) *R(t)=∫ρ(l,t) dl* and the (ray-path-averaged) relative abundance of the water vapor *a(t)*

$$w(t)=C\cdot R(t)a(t)$$

where *C* is appropriate constant coefficient (basically conversion between units). If we express each quantity as its time-averaged mean value and a small variation, e.g.

$$w(t)=\langle w\rangle+dw(t)$$

and likewise for *R(t)* and *a(t)*, we arrive (after some simple algebra) to expression

$$w(t)=\langle w\rangle\cdot\left(1+\frac{dR(t)}{\langle R\rangle}+\frac{da(t)}{\langle a\rangle}\right)=\langle w\rangle+\langle w\rangle\cdot\left(\frac{dR(t)}{\langle R\rangle}+\frac{da(t)}{\langle a\rangle}\right),$$

where we also neglected the second-order term *dR(t).da(t)*. Here, the latter term clearly represents the PWV variation *dw(t)*.

Now, we can express the wet opacity as

$$\tau_w(t)=\tau_1\cdot w(t)=\tau_1\langle w\rangle\cdot\left(1+\frac{dR(t)}{\langle R\rangle}+\frac{da(t)}{\langle a\rangle}\right)$$

Similarly, the dry opacity will reads

$$\tau_d(t)=\langle\tau_d\rangle\cdot\left(1+\frac{dR(t)}{\langle R\rangle}\right)$$

as it is related just to the total integrated air density variations and not to WV abundance. So, if we express the total opacity

$$\tau(t)=\tau_w(t)+\tau_d(t)=\tau_1\cdot\langle w\rangle\cdot\left(1+\frac{dR(t)}{\langle R\rangle}+\frac{da(t)}{\langle a\rangle}\right)+\langle\tau_d\rangle\cdot\left(1+\frac{dR(t)}{\langle R\rangle}\right) \qquad (3.2.1)$$

we can write it, again, as its mean (time-averaged) value and variation

$$\tau(f,t)=\langle\tau(f)\rangle+d\tau(f,t)$$

where the obvious frequency dependence is now explicitly indicated, and, the variation itself may be expressed in two alternative ways, depending on how we arrange and combine the terms in Eq. (3.2.1):

$$\begin{aligned} d\tau(f,t)&=\langle\tau(f)\rangle\cdot\frac{dR(t)}{\langle R\rangle}+\langle\tau_w(f)\rangle\cdot\frac{da(t)}{\langle a\rangle} \\ &=\tau_1(f)\cdot dw(t)+\langle\tau_d(f)\rangle\cdot\frac{dR(t)}{\langle R\rangle} \end{aligned} \qquad (3.2.2)$$

Now, this expression (3.2.2) may be inserted in the general Eq. (3.1.1) to get two alternative generalizations of the central formula (3.1.4):

$$\int_{BW}\frac{1}{G(f)}\cdot\left(P(t,f)-\langle P(f)\rangle\right)df=-dw(t)\cdot\int_{BW}\left(\frac{\langle P(f)\rangle}{G(f)}-T_{inst}(f)-T_{atm}\right)\tau_1(f)\,df$$
$$-\frac{dR(t)}{\langle R\rangle}\cdot\int_{BW}\left(\frac{\langle P(f)\rangle}{G(f)}-T_{inst}(f)-T_{atm}\right)\langle\tau_d(f)\rangle\,df \qquad (3.2.3a)$$
$$+\delta T_S(t)\int_{BW}\left(\frac{f}{f_o}\right)^s\cdot e^{-\langle\tau(f)\rangle}\,df$$

and

$$\int_{BW}\frac{1}{G(f)}\cdot\left(P(t,f)-\langle P(f)\rangle\right)df=-\frac{da(t)}{\langle a\rangle}\cdot\int_{BW}\left(\frac{\langle P(f)\rangle}{G(f)}-T_{inst}(f)-T_{atm}\right)\langle\tau_w(f)\rangle\,df$$
$$-\frac{dR(t)}{\langle R\rangle}\cdot\int_{BW}\left(\frac{\langle P(f)\rangle}{G(f)}-T_{inst}(f)-T_{atm}\right)\langle\tau(f)\rangle\,df \qquad (3.2.3b)$$
$$+\delta T_S(t)\int_{BW}\left(\frac{f}{f_o}\right)^s\cdot e^{-\langle\tau(f)\rangle}\,df$$

As a result, we get a linear system of 12 equations similar to Eq. (3.1.5) (if we use all AC and SQLD SPWs), now for three independent variables: In the approach (3.2.3a) the 3-vector of unknown variables is composed of the PWV variation, relative air-mass (integrated density) variations, and the (apparent) source-brightness dynamics; in case of (3.2.3b) the independent variables are relative 'humidity' variation, relative air density variation, and again, the source-brightness dynamics. The approaches 3.2.3a and 3.2.3b are equivalent, just the 3-vector of the unknown variations is expressed in another basis.

We first tried the approach (3.2.3b) but the results are absolutely disappointing – the relative variations in 'humidity' *a(t)* and air (averaged) density *R(t)* are both oscillating in interval much broader than -1 to +1 (which is non-sense – they represent the *relative* change of the respective quantities, so expected variations were at the level less than 1%), essentially in anti-phase, compensating each other in order to get the finite and correct right-hand-side vector of the system (the sky brightness variations). The reason is clear – their matrix coefficients differ only to that extent as the time-averaged wet opacity $<\tau_w(f)>$ differs from the total opacity $<\tau(f)>$. Since the difference is negligible, the system is very ill-conditioned (inverse value of the matrix determinant is of the order of 10^5). In this sense, the result is not surprising.

With the approach (3.2.3a) the result is somewhat better, but also not answering the question: Eq. (3.2.3a) essentially differs from Eq. (3.1.4) only by presence of the 2nd term at the RHS. Now we have, indeed, different frequency dependence of the matrix elements – the dry opacity $<\tau_d(f)>$, controlling the 2nd term at RHS, is very different function of frequency than $\tau_1(f)$, so the rows of the linear system formed by different SPWs are independent vectors. As a result, we reproduce the *dw(t)* variations with a curve that is indistinguishable from that produced by Eq. (3.1.4) (i.e., without dry/wet decomposition). On the other hand, the relative variations of the integrated air density are, again, very high. The reason is, that the matrix coefficient (sensitivity factor) for the air-density variations is very

low – it is proportional to the dry opacity – and, thus, its inverse value, used for the density variation calculation, is of the order of 10^4. So, in fact, we just effectively multiply the noise by this high factor.

Summary: We did not succeed with decomposition of the opacity into dry/wet (or humidity/density) sub-components: In a first case we reproduce the *dw(t)* curve correctly, but the resulting calculated density variations are essentially just an amplified noise, in the latter case, the humidity and total-density variations are both wrong as the system is ill-conditioned.

2.3.3 From dPWV fluctuations to full-PWV time series: Rigorous approach

In so far, when speaking about "*dPWV proxy*" calculated from science SPW signal, we have been always working with the PWV *fluctuations* – differentials to the mean PWV value, time-averaged over a scan. We have presented many various approaches to dPWV calculation, the most sophisticated one in the last Section 2.3.1. It decomposes the science-SPW signal fluctuations into (i) varying absorption (due to changing PWV), and, (ii) to the apparent brightness changes of the target, where 'apparent' stands here for (de)focusing effect of atmospheric 'bubbles' (convective turbulence vortices), i.e. the scintillation. Nevertheless, even this advanced method, implemented in *dWdB()* routine in our *sol_wvrgcal.py* collection, works also just with differences to the average PWVs and brightness temperatures.

In order to achieve the *full phase corrections*, not just removing (or significantly reducing) the fluctuating part, we, however, need the *complete* PWV time series for each antenna. To that end we have developed a new, rigorous methodology for calculation full PWV time series from the science data. This text represents a full-detail report on this topic, including all incremental steps that imporved the procedure. Impatient reader may skip directly to Section 2.3.3.13, where the final result implementing all accumulated gradual improvements is presented, and return back to the mathematical derivations, if interested.

2.3.3.1 Starting point

In the previous sections 2.3.1 and 2.3.2 we have presented a method, which decomposes fluctuations of signal recorded in the science SPWs around its mean (scan-averaged value) into two parts: (i) Variations due to changes of atmospheric transmissivity, which changes because of varying PWV, whose fluctuations we describe by a quantity *dw(t)*, and, (ii) variations of intrinsic or apparent (due to scintillation) brightness of the solar target, described as $dT_s(t)$. The decomposition is possible thanks to different frequency dependencies of transmissivity (determined by absorption profile of WV line @183GHz) and solar brightness (given by the solar thermal spectrum), and the fact that we observe over a relatively broad frequency domain of Band 5 in the range of 190 GHz – 206 GHz. We have eventually arrived to a linear system (3.1.4) for an unknown 2D vector $x \equiv (x_1, x_2) = (dw, dT_s)^T$ in the form of

$$A_{i1} \cdot x_1 + A_{i2} \cdot x_2 = C_i, \qquad (3.3.1)$$

where matrix coefficients A and the RHS-vector components C (recorded temperature fluctuations) read (see Eq. 3.1.5, we repeat it here for readers convenience):

$$C_i = \int_{BW} \frac{1}{G(f)} \cdot \left(P(t,f) - \langle P(f) \rangle \right) df \equiv \delta T_i^{rec},$$
$$A_{i1} = -\int_{BW} \left(\frac{\langle P(f) \rangle}{G(f)} - T_{inst}(f) - T_{atm} \right) \tau_1(f)\, df, \qquad (3.3.2)$$
$$A_{i2} = \int_{BW} \left(\frac{f}{f_o} \right)^s \cdot \mathrm{e}^{-\langle \tau(f) \rangle}\, df$$

Index *i* goes over our available SPWs (i.e., AC-resolved 5,7,9,11, AC-integrated 6,8,10,12, and SQLD 0,1,2,3) from 1 to 12. We solve this system independently for each polarization (XX/YY) and each time instant. The system is over-determined, because we have – for each time instant and each polarization – 12 equations and only 2 variables, so we solve it in the least square sense, i.e. the $(A.x - C)^2 \rightarrow min.$ The minimizer of this quadratic form leads to the system of so called normal equations $A^T.A.x = A^T.C$. The matrix coefficients A_{i1} and A_{i2} represent the *sensitivity factors* for *dw* and dT_s, respectively – the bigger they are, the larger are responses to *dw* and dT_s variations seen in data in form of the fluctuating received power. For each SPW id $S=i$ we write $SF_W^S = A_{s1}$ and $SF_B^S = A_{s2}$, where we have chosen index 'B' for brightness-temperature sensitivity in order to not mixing it with 't' for time.

Now we present the incremental steps in the development of our procedures towards mathematically rigorous handling of the problem of full PWV(t) time series.

2.3.3.2 Polarization coupling in PWV

Easiest incremental improvement is connected with improved handling of the differential approach governed by Eq. (3.3.1) and concerns 'double' solution for *dW(t)*. Insofar we have been handling the system (3.3.1) independently for both XX and YY pols. This way, we have got two time series for both *dw(t)* and $dT_s(t)$ – see, e.g., Fig. 32, which presents the time-series as red and blue curves per polarization. While the brightness variations $dT_s(t)$ can, in principle, change independently in each polarization, there is no logical reason for *dw(t)* should do so, too.

In addition of independent per-polarization solutions, we have also viewed Equations (3.3.1) as being independent for each time instant, i.e., we have been, in fact, handling *nPols x nTimeSamples* independent systems to be solved for each time (integration) and each polarization separately. It is, however, useful to conceptually change the view and look at it as to one large linear system like in Fig. 38: The large matrix of this system is composed from blocks of the matrix *A* of the basic system (3.3.1), and the new, large RHS vector is a formed as a stack of the basic RHS vectors *C*.

The vector of unknowns *dw* and dT_s is also formed as a stack of elementary (per time and polarization) solutions of system (3.3.1), the polarization and time (technically, index of correlator-data integration) now represent (multi-)index in this large vector of unknowns. Mathematically, both views are equivalent, because the blocks in the large matrix apply only on the relevant part of the composed unknown-variable vector and assure that the product will be equaled just with the appropriate (for a given time and polarization) segment of the RHS (data) vector. But this large-system view allows us to understand changes applied further in the following text.

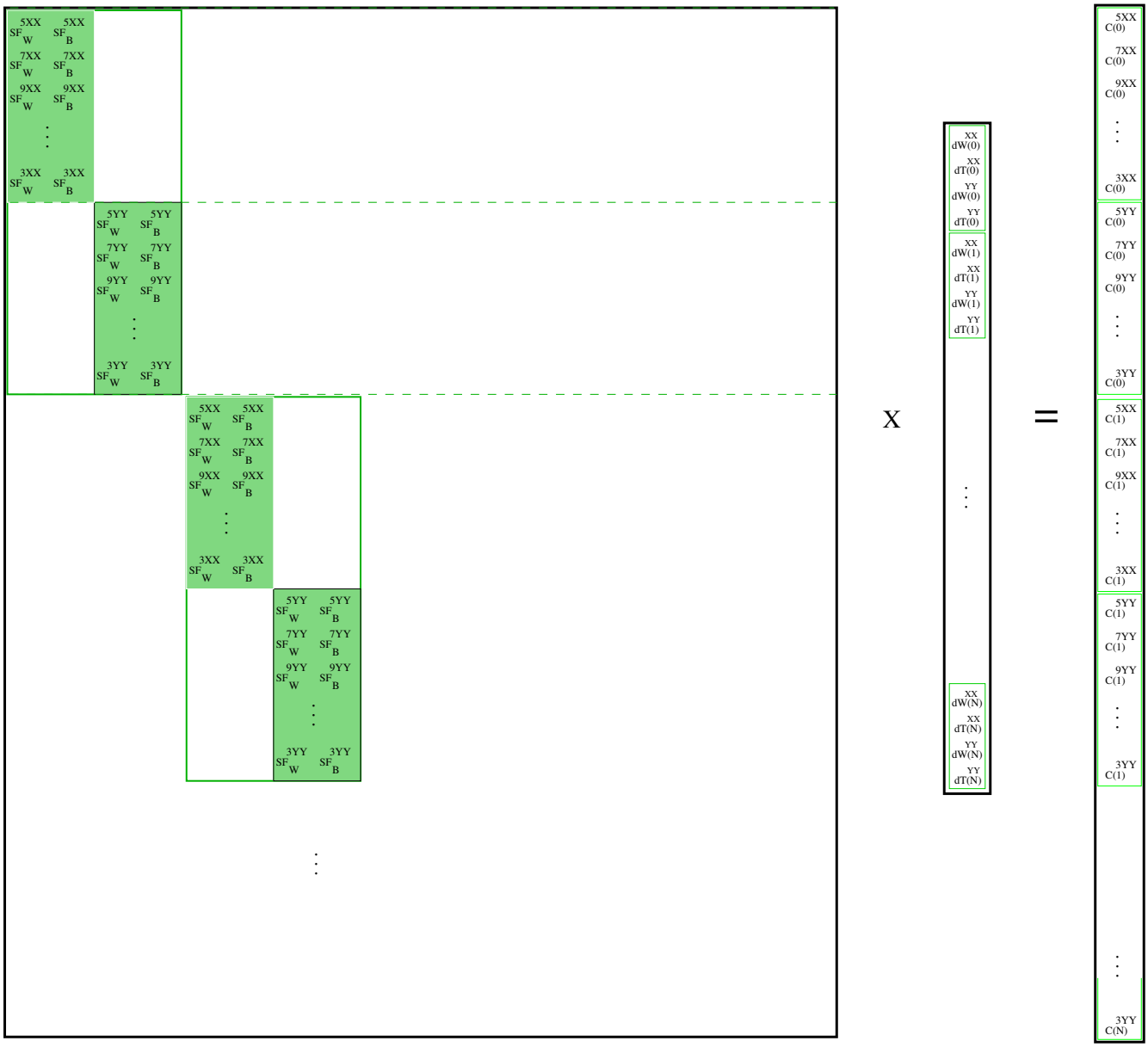


Figure 38: Linear system (3.3.1) when viewed not like nTimeSamples x nPols independent subsystems but as one equivalent large linear system of equations. The spw and polarization indices are written directly as super-scripts, the time index is traditionally written in parentheses (albeit it is discrete, too) and indexed from 1 to N, where N is a shortcut for nTimeSamples. Only green areas in the large matrix are occupied by non-zero numbers (the sensitivity factors for dW and dT), the white areas are all zeros.)

Our changed view on the system (3.3.1) gives a hint, how we can move forward:

- We re-arrange the unknown-variables vector (the order of its component) such a way, that variables of the same kind (*dw, dT*) are now next each other i.e., from the scheme in Fig. 38

$\{dW^{XX}(0), dT^{XX}(0), dW^{YY}(0), dT^{YY}(0), dW^{XX}(1), dT^{XX}(1), dW^{YY}(1), dT^{YY}(1),\ldots..,\ dW^{YY}(N), dT^{YY}(N)\}$
we come to $\{dW^{XX}(0), dW^{YY}(0), dT^{XX}(0), dT^{YY}(0), dW^{XX}(1), dW^{YY}(1), dT^{XX}(1),\ldots, dT^{XX}(N), dT^{YY}(N)\}$

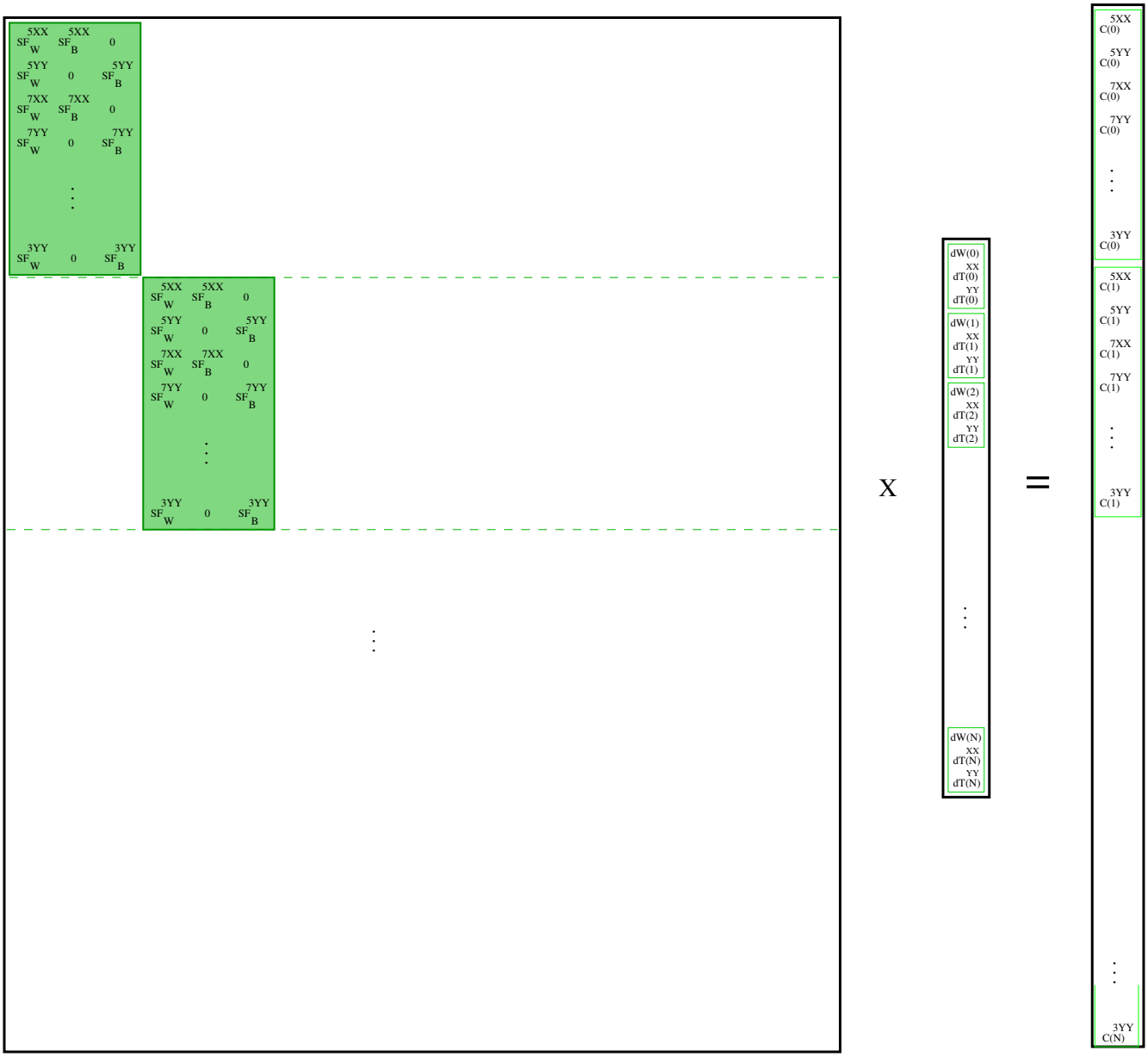


Figure 39: Re-arranged linear system from Fig. 38 with reduced set of unknowns: We have now single solution for dw(t) variations. The green blocks are of shape 24 x 3, rest white areas of the large-system matrix are zeroes.

- We further re-arrange the order of equations: So far, in Fig. 38, they were ordered by time, polarization, and SPW, so that the SPW-index changes fastest, time-index slowest. We make the

polarization to be the fastest-changing index, i.e. the RHS column components will take the form $\{C^{5XX}(0), C^{5YY}(0), C^{7XX}(0), C^{7YY},(0), \ldots, C^{3YY}(N)\}$.

- Now we can merge the variables $dw^{S,XX}(t)$ and $dw^{S,YY}(t)$ for SPW *S* and pols *XX/YY* together into a single one $dw^{S}(t)$ (note, still it is a vector, the SPW index *S* and the time index *t* remained, just the pol dimension contracted).

The above operations form a reduced system shown in Fig. 39, whose forming blocks are now matrices of the 24(rows) x 3(columns), where 3 comes from 3 sub-variables $\{dw^{S}(t), dT^{SXX}(t), T^{SYY}(t)\}$, still a vector with *S* and *t* indices, and 24 is a product of *nPols x nSPWs*.

The structure of the system keeps its block-diagonal form and the same hold for the corresponding normal equations $A^{T}.A.x = A^{T}.C$. The normal matrix $A^{T}.A$ is formed from 3x3 blocks at its diagonal with all other elements being zero. This has a practical consequence – we can still solve the problem for each time instant independently, having *nTimeSamples* small systems of size 3x3.

Results: See Fig. 40. The brightness variations have now more pronounced (scintillation) peaks, some 40 – 60 % higher, the *dw(t)* curve did not change much from its 'double', per-pol solution (it looks a bit smoother), because already when it was decomposed into two polarizations, they were almost identical – c.f. Fig. 32.

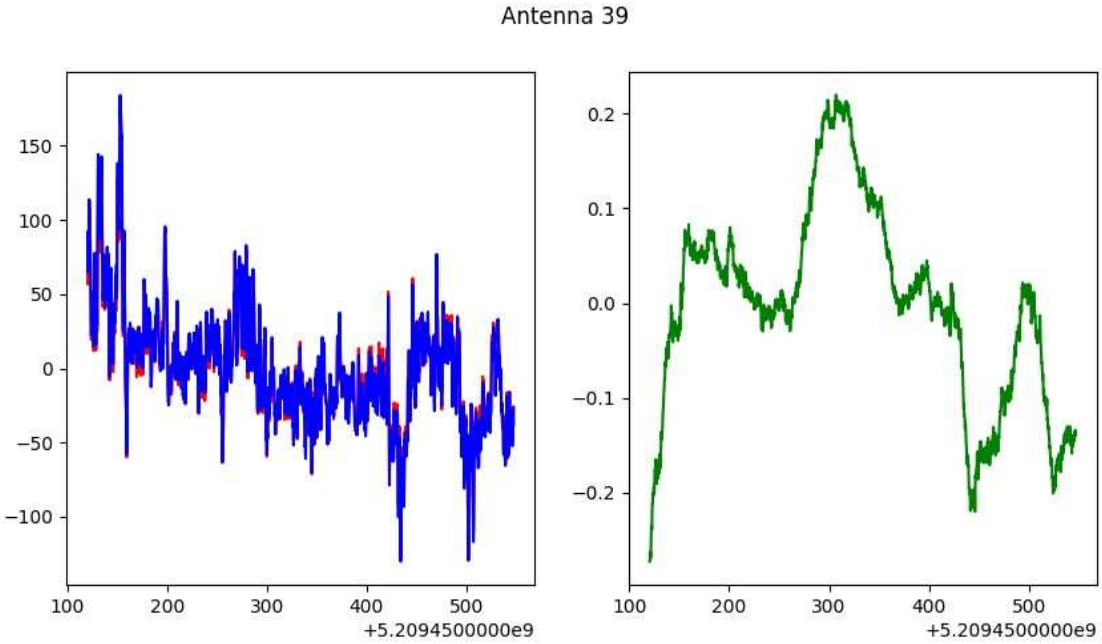


Figure 40: Left: Fluctuations of apparent (incl. scintillations) brightness temperature in XX (red) and YY (blue) pols as a function of time (MJDsec). Right: Fluctuations of PWV (mm) as a function of time. Both for antenna id 39 (DV11). We now have just a single dPWV(t) time series, not two per-polarization solutions.

2.3.3.3 Making use of the scan-averaged signal

Now we come to the main point – upgrade of our insofar differential (fluctuations-only) approach used for science SPW data in order to derive the dPWV proxy, towards the full-signal handling. The main change lies in our handling of atmospheric properties – its transmissivity and intrinsic emission. Insofar we have been using just differences/derivatives, now we go using their full 'expressions' as a functions of PWV. This change makes the set of our governing equations non-linear because the brightness temperature of the sky is non-linear function of PWV.

Mathematical background

We start with separating instrumental (internal) and external sources of the recorded signal

$$T_{rec}^{SP}(t) \stackrel{\text{def}}{=} \frac{P_{rec}^{SP}(t)}{G^{SP}} = T_{inst}^{SP} + T_{ext}(t) \qquad (3.3.3)$$

where the 'external' contribution to intensity of radiation, expressed by its equivalent temperature, is by the Sun, diminished by atmospheric transmissivity *tr()*, and intrinsic contribution of the terrestrial atmosphere (namely WV-line wing) and cosmic background (CMB), handled together as T_{sky}:

$$T_{ext}(t) = tr[w(t)] \cdot T_{Sun}^{B}(t) + T_{sky}[w(t)] \qquad (3.3.4)$$

The 'internal' T_{inst} contribution has been already defined in Section 2.2.2 as $T_{inst}=T_{RX}+ P_o/G$, in the AC SPWs it equals the receiver temperature T_{RX}, in SQLDs it accounts also for instrumental offset P_o.

We assume all quantities in Eq. (3.3.4) are integrated (averaged) over frequency channels of the SPW in a given polarization. The external brightness temperature can be thus written as as an integral over the bandwidth *BW* of the SPW *S* in polarization *P* (indicated by sub- and super-scripts):

$$T_{ext}^{SP}(t) = T_{Sun}^{B}(t) \cdot \int_S tr(w(t), f) \cdot S(f) \cdot g^{SP}(f) \frac{df}{BW} + \int_S T_{sky}^{S}(w(t), f) \cdot g^{SP}(f) \frac{df}{BW} \qquad (3.3.5)$$

Here we define the spectral gains $g^{SP}(f)$ normalized to unity as

$$G^{SP} \stackrel{\text{def}}{=} \int G^{SP}(f) \frac{df}{BW}, \quad g^{SP}(f) \stackrel{\text{def}}{=} \frac{G^{SP}(f)}{G^{SP}}$$

For the channel-resolved SPWs we calculate $g^{SP}(f)$ from the ACD calibrations, for the integrated AC SPWs and SQLDs they are set to unity. *S(f)* is the already earlier introduced solar spectrum normalized to 1.0 at the reference frequency f_o, where we evaluate the T^{B}_{Sun}, i.e. $T^{B}_{Sun}(f,t) = T_B^{Sun}(t).S(f)$. The source spectrum *S(f)* is expressed (for now – see improvements below) as a power-law $S(f)=(f/f_0)^s$, and we assume spectral index $s=2$, which corresponds to Rayleigh-Jeans law for black-body radiation.

In fact, we have no analytical formula expressing the transmissivity and brightness temperature of the sky (atmosphere + CMB) as a function of frequency and PWV, but we can obtain their numeric values from the ATM model (Pardo et al., 2001; implemented as *atmosphere/at* tool in CASA) for any frequency channel and any user-set PWV value (plus further parameters that are preset by

at.setAtmosphere() in order to match the atmosphere-model to actually measured meteo conditions). Specifically, we evaluate T_{sky} in SPW *S* as

$T_{sky}^{S}[w(t),f]=at.getTebbSkySpec(spw=S, userH2O=w(t))$ (3.3.6)

Inserting (5) into (3) we get

$$T_P^B(t)\cdot\int_S tr(w(t),f)\cdot S(f)\cdot g^{SP}(f)\frac{df}{BW}+\int_S T_{sky}^S(w(t),f)\cdot g^{SP}(f)\frac{df}{BW}=T_{rec}^{SP}(t)-T_{inst}^{SP} \qquad (3.3.7)$$

where we have dropped the subscript '*Sun*' for the brightness temperature as it cannot make confusion and we need the space for polarization index *P*: As can be seen from (3.3.7), we allow the (apparent) solar brightness temperature to be dependent on polarization *P,* while solar spectrum and atmospheric transmissivity are dependent only on frequency, hence they need to be calculated for each SPW separately (therefore subscript *S* at the integral, but no polarization dependence). As in the case of differential (fluctuation-only) approach, we can write equation (3.3.7) for all spw:pol combinations, which represents in total *numSPW x numPOL* = 12 x 2 = 24 equations for each time instant. At the same moment, it must be valid for all time instants (integrations), for which we have our measured data $T^{SP}_{rec}(t)$. So, in total, the Equation (3.3.7), in fact, represents a system of *numSPW x numPOL x nTimeSamples* equations. Basically, we are in similar situation as in the differential (fluctuation) approach (Fig. 39), but now we face a large set of *non-linear* equations. We have, again, three sub-vectors of unknowns: $w(t)$, $T_{XX}^B(t)$, and $T_{YY}^B(t)$, each of them having *nTimeSamples* components, so in total it represents a long vector of 3 x *nTimeSamples* unknown variables. Clearly, the system is over-determined as in case of fluctuation-only (linear) approach, because number of equations exceeds the number of unknown variables. It can therefore hardly be fulfilled exactly and we, again, search for a solution in the least-square sense. Our system of equations (3.3.7) takes the general form

$$f_k(x_j)-b_k=0 \qquad (3.3.8)$$

where $x_j=\{w(t),T_{XX}^B(t), T_{YY}^B(t)\}$ is our unknown vector, the vector function $f_k(x_j)$ represents the LHS of Eq. (3.3.7), and b_k stands for the vector of the RHS values ($T_{rec}(t) - T_{inst}$). Index *k* goes from 1 to *numSPW x numPOL x nTimeSamples.* In the least-square approach we search for such a vector x_j so that

$$\sum_k (f_k(x_j)-b_k)^2 \rightarrow min. \qquad (3.3.9)$$

For the minimum of Eq. (3.3.9), gradient in the x_j-space should be zero, which leads to equation

$$J_{ik}^T(x_j)[f_k(x_j)-b_k]=0 \qquad (3.3.10)$$

where J_{ik}^T is the transposed Jacobian matrix of the vector function $f_k(x_j)$, i.e.

$$J_{ik}^T=\left(\frac{\partial f_k(x_i)}{\partial x_i}\right)^T$$

Calculation of the Jacobian J_{ik} is performed by differentiation of the LHS of Eq. (3.3.7). For $T^S_{sky}[w(t),f]$, whose values are obtained numerically via interface-functions to the ATM model (CASA *at* tool), we calculate numerical derivative w.r.t. the unknown $w(t)$ also numerically, by evaluation the function in two close adjacent points for current solution of PWV +/- 0.05mm. Actually, the Jacobian

matrix J_{ik} is identical with our block-composed matrix in Fig. 39 – we can identify that the Jacobian elements represent exactly the expressions for sensitivity factors SF_W and $SF_B^{XX/YY}$ for the dW and dT^B variations, used in our older approaches derived without mathematical rigorozity, which backwards justifies their correctness.

Eq. (3.3.10) represents, in fact, a system of 3 x *nTimeSamples* non-linear (because J is evaluated at the exact solution x_j, as well as $f_k(x_j)$ is) equations, so we have now the same number of equations as variables. Still it is difficult to be solved, so we do one more step forward and perform linearization of the system around some operational point x^0_j. Using Taylor expansion to the first order we eventually get

$$J^T_{ik}(x^0_j)\cdot J_{kj}(x^0_j)\cdot(x_j - x^0_j) = J^T_{ik}(x^0_j)\cdot[b_k - f_k(x^0_j)] \qquad (3.3.11)$$

This is the "master equation" for our approach. It is a linear approximation of Eq. (3.3.10) for displacements of the solution from the operation point, but it can be used iterativelly such a way that the displacement

$$\Delta x^K_j \stackrel{\text{def}}{=} x^K_j - x^{0,K}_j$$

found as a solution of Eq. (11) in the iteration K, can be added to the original operation point x^0_j

$$x^{0,K+1}_j = x^{0,K}_j + \Delta x^K_j$$

for getting the new operation point in iteration $K+1$. This way the operation point converges to the exact solution of the non-linear system (3.3.10). When we are there, the RHS of Eq. (3.3.11) is exactly zero (because (3.3.10) is fulfilled), and so are then the displacements from the operation point $(x_j - x^0_j)$ on the LHS (because the LHS matrix $J^T.J$ is regular), so we stop moving in the x_j-space for further iterations (convergence completed). Essentially, this iterative approach is an implementation of general Newton-Raphson method for solution of non-linear systems specific for our problem.

In practice, Eq. (3.3.11) is a good approximation already if the operation point x^0_j is close to the exact solution. We conveniently chose x^0_j such a way that $\{w^0(t), T^0_{XX}{}^B(t), T^0_{YY}{}^B(t)\} = \{\langle w(t)\rangle, \langle T_{XX}{}^B(t)\rangle, \langle T_{YY}{}^B(t)\rangle\}$, i.e. replacing the actual values of the PWV and brightness temperatures for XX and YY pols in each specific time instant t by their scan-averaged means. Practically it means that we use iteration scheme

$$x^{0,K+1}_j = x^{0,K}_j + \langle \Delta x^K_j \rangle \ . \qquad (3.3.12)$$

The displacements $(x_j - x^0_j)$ at the LHS of Eq. (3.3.11) then represent fluctuations around means of our unknown quantities, exactly as we have had in our older approaches. Eq. (3.3.12) means that we iteratively correct the operation point by the averages of fluctuation, ensuring that the operation point at the end of iterations is really the time average, so that the sum of fluctuations around it is zero. When we work essentially with fluctuations again, what is then the practical difference to our older approach? Now we can always add the fluctuations (displacements found by Eq. 3.3.11) on top of the average (our operation point, which is found iteratively by Eq. 3.3.12) and get the full values of PWV and brightness temperature at each time instant!

Practically, this ‘uniform’ (time-independent) selection of the operation point assures that the matrix $J^T.J$ of the linearized system (3.3.11) is composed of *nTimeSamples* identical *3x3* blocks at the matrix diagonal, with all other LHS matrix elements being zero – it can be easily seen, when one makes right-sided matrix multiplication of the matrix in Fig. 39 with its own transposition (i.e. $J^T.J$). Such a simple structure of the linearized system facilitates the solution a lot. Because of the block-diagonal nature of the LHS matrix, it can be, in fact, solved for each time instant independently, resembling the system in Eq. (3.1.5) (the differential/fluctuation approach), but now using also information contained in time-averaged (over a scan) values of received powers (not just their fluctuations), which, in turn, allows for the full PWV and full T^B time-series solutions.

The above described procedure has been implemented in the new routine *dWdB_Tbs()*. Now we shall present its sample results for antenna 43 (DV17), the full dataset is available online.

2.3.3.4 Early results

Fig. 41 shows the time series of the PWV at antenna DV17 – we almost identical fluctuations as in our differential approach, varying around mean PWV~2.7mm, which is too low: Median PWV for the observation as reported by *aU.getMedianPWV()* (it reads it from the SYSCAL table) is 4.86mm. So, there is clearly some issue.

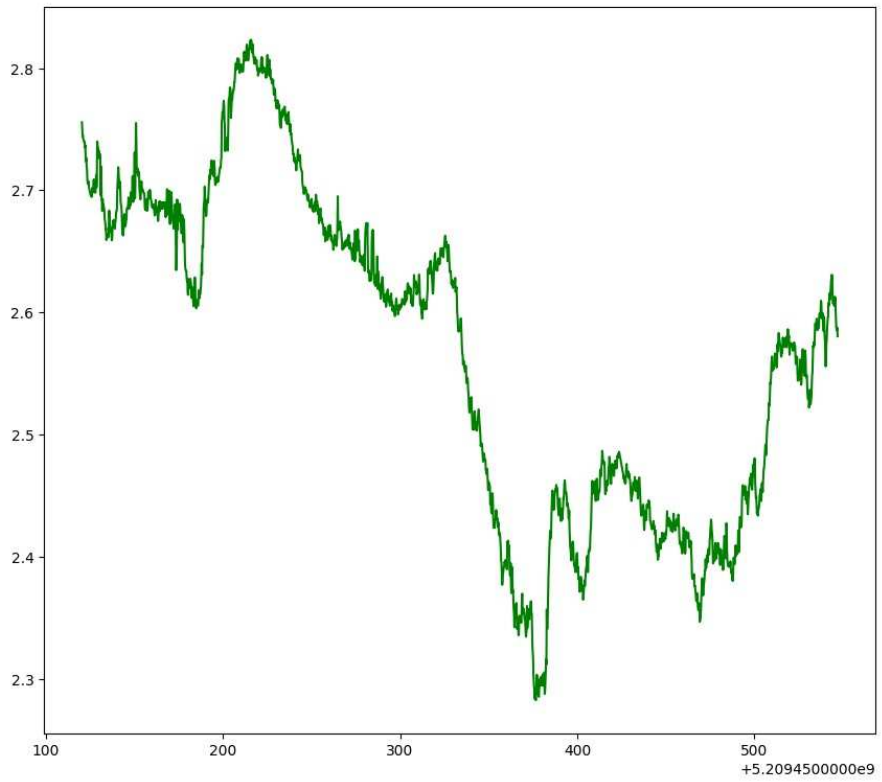


Figure 41: PWV in mm as a function of time (MJDsec) for Scan 9 at antenna id43 (DV17).

Complementary brightness temperatures for XX and YY pols are shown in Fig. 42. Again, the variations of the signal is almost identical to that found by our original differential approach, but the mean value ~4900 K lags far behind expectations: The nominal/reference brightness temperature for the solar disc center in Band 5 used in the TP imaging (White et al. 2017) is 6300 K. If the limb brightening (our target is very close to limb) is taken into account – see the next subsection “Improving the solar spectrum” – expected value is even ~7000 K. Hence, even in this respect the early result is not satisfactory.

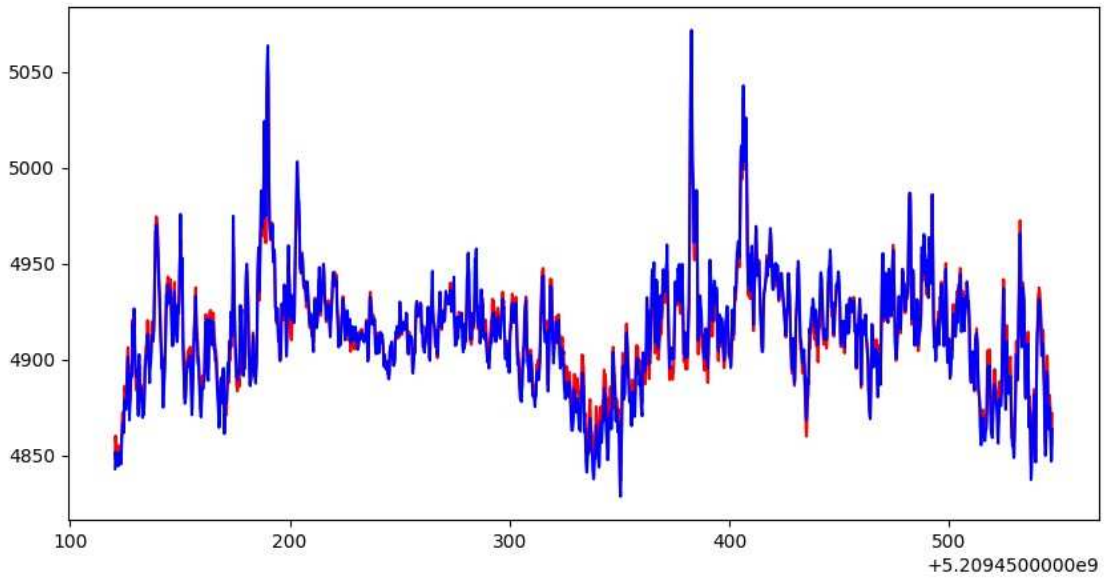


Figure 42: Solar-target brightness temperatures (in Kelvins) in XX (red) and YY (blue) polarizations as a function of time (as MJDSecs) in Scan no. 9.

In the previous Fig. 31 we have shown also analysis of the quality of the solution. We present similar analysis here, for our new approach. The results for antenna 43 (DV17) are displayed in Fig. 43. Again, like in Fig. 31we can calculate the residual value for each equation in the system (3.3.8) by inserting its solution found by the iterative usage of linearized Eq. (3.3.11). These residuals are shown in the top row of panels in Fig. 43, color- and line-style coded for specific SPW: The colors code the base-bands (BB1 – BB4), line-style within the same color shows the specific SPW covering the BB (AC resolved = solid, AC integrated = dashed, SQLD = dotted, see also description in Fig. 31), left column shows XX and the right the YY polarizations. The difference between the model of the sky+src brightness temperature (LHS of Eq. 3.3.7) and the scan-averaged received temperature corrected by instrumental offset (time-average of the RHS of Eq. 3.3.7) are presented in the middle row. Finally, the bottom row shows the ‘net’ source contribution to the recorded-signal variations (the fluctuation of T^B multiplied by its sensitivity factor). As it can be seen at the first look (comparing the top rows in Fig. 31 with Fig. 43 here), we have now much bigger residuals then in the purely differential approach. The main difference

is in the mean values of the residuals for *spw:pol* combination (all color- and line-style coded curves are far from zero), while the fluctuations are basically of the same magnitude as in Fig. 31. It is clear that we did not manage to fulfill the set of Eq. (3.3.7) simultaneously for all spw:pol data channels, because it is hard to fit the mean (scan-averaged) sky+src brightness temperatures for all spw:pols at once. Moreover, the line of the same color – i.e. the same BB – groups together, we see a clear trend of changing mean residual with frequency.

Summary: We have now, for he first time, the full PWV(t) time series and also the time series of apparent (including atmospheric scintillation effects) solar brightness temperature. On the other hand, the early results are quite disappointing:

- The PWV variations are basically the same as in the differential approach – this result is, indeed, robust. On the other hand, the mean value of ~2.7mm is far low from expected – the SYSCAL reports PWV median to be 4.8mm for this MS.
- The same holds for time-averaged solar brightness temperature – variations are OK, but the expected mean value in Band 5 for the target at the solar limb is ~7000 K, while we have got ~4900 K.
- The residual errors (specifically their time-averages) are now much higher than for our older treatment via signal-fluctuations only.
- Moreover, the residuals have been found to have a clear trend with frequency – see how the lines in Fig. 43. are ordered by colors: red, black, blue, green – the colors code the BBs. We address this finding immediately in the next subsection “Improving the solar spectrum”.

We spend a lot of effort in identification of the source of this discrepancy between model and data and our way towards improving the model/data fit is described in the following subsections.

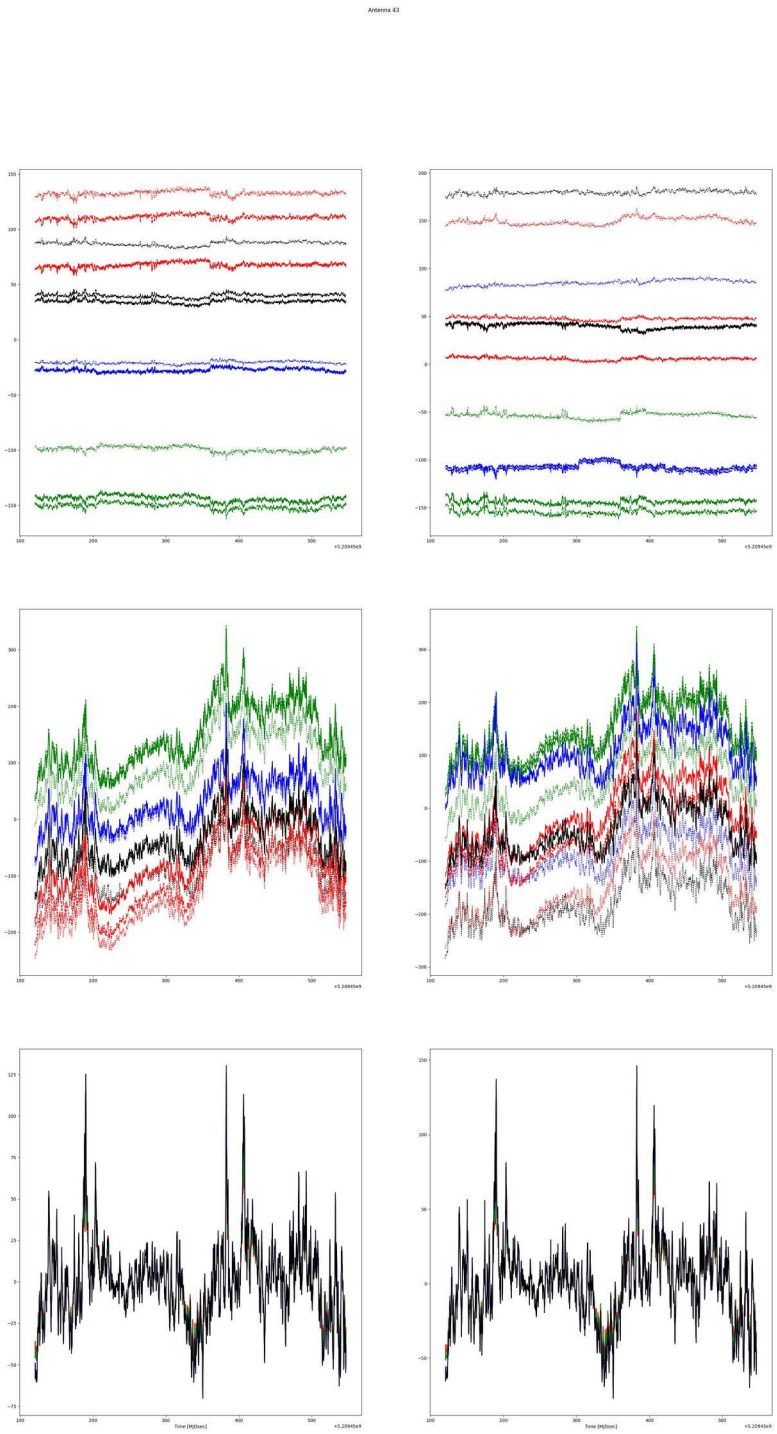


*Figure 43: Residuals of Eq. (*3.3.7*) as temperatures in K (top row), sky+src brightness model minus time-averaged observations (middle, in Kelvins) and source brightness contribution (K) to the total sky brightness variations as extracted by solving the system in Eq. (*3.3.7*). Left and right columns were obtained from XX and YY pols in science SPWs. Can be directly compared with Fig. 31*

2.3.3.5 Improving the solar spectrum

The solar spectrum (scaled by the atmospheric transmissivity) forms the sensitivity factor for brightness changes – see the 3rd equation in (3.3.2). Insofar we have adopted a naive 'zero-order' approach of the Sun as a black-body and thus the intensity increases with frequency-squared as prescribed by Rayleigh-Jeans law (see Eq. 3.1.3). However, the results of our fitting procedure show that it is most likely wrong, as we can see systematic trend in in residual errors for different frequency domains – while the low-frequency SPW 5 (and its integrated and SQLD companions 6 and 0) seem to have 'under-rated' temperatures, the opposite side of Band 5 frequencies (SPWs 11,12, and 3) are over-rated. We start playing with the spectral index *s* in the last Eq. of (3.3.2), which is nominally set to 2 for the R-J law, and succeeded to improve the results with negative *s* around *–0.5*. However, such an *ad hoc* change of the spectral law is highly unsatisfactory, so we started research in literature. We have realized that situation is, in fact, much more complicated than the black-body assumption and have eventually found semi-empiric formula for quiet-Sun brightness temperature, which fits observations, in Alissandrakis et al. (2020). The brightness temperature, in this approach, is a function of frequency and μ, which is the direction cosine of the position angle – the angle between the normal to the solar surface in the location of our target and the line of sight. Hence the object in the solar disc center has this angle zero and $\mu=1.0$, while close to the solar limb angle grows to 90 degs and its cosine approaches zero. The exact expression for $T^B(f, \mu)$ adapted from Alissandrakis et al. (2020) reads:

$$T^B(f,\mu)=T^B(f_o,1)+a_2\ln(\mu)+2a_2\ln\left(\frac{f}{f_o}\right) \qquad (3.3.13)$$

where the semi-empirical constant a_2 has a value –580 K and the reference brightness temperature $T^B(f_o,1)$ for f_o=198 GHz at the solar disc center (μ=1) is 6300 K (White et al. 2017). Dependence on the position angle means that we have to read ephemeris file in order to locate the target w.r.t. the solar disc center.

Results: Clear improvement in all respects

- Mean PWV for antenna DV17 is now 4.78 mm, i.e. much closer to the expected value, the mean solar brightness temperature increased to 5900 K, still being lower than expectations: Eq. (3.3.13) gives for out close-to-limb target with low μ estimated value 6990 K.
- The residual-means are lower – they diminished from amplitude of 150 K to 60 K, and rather random – the frequency trend disappeared – see Fig. 44.

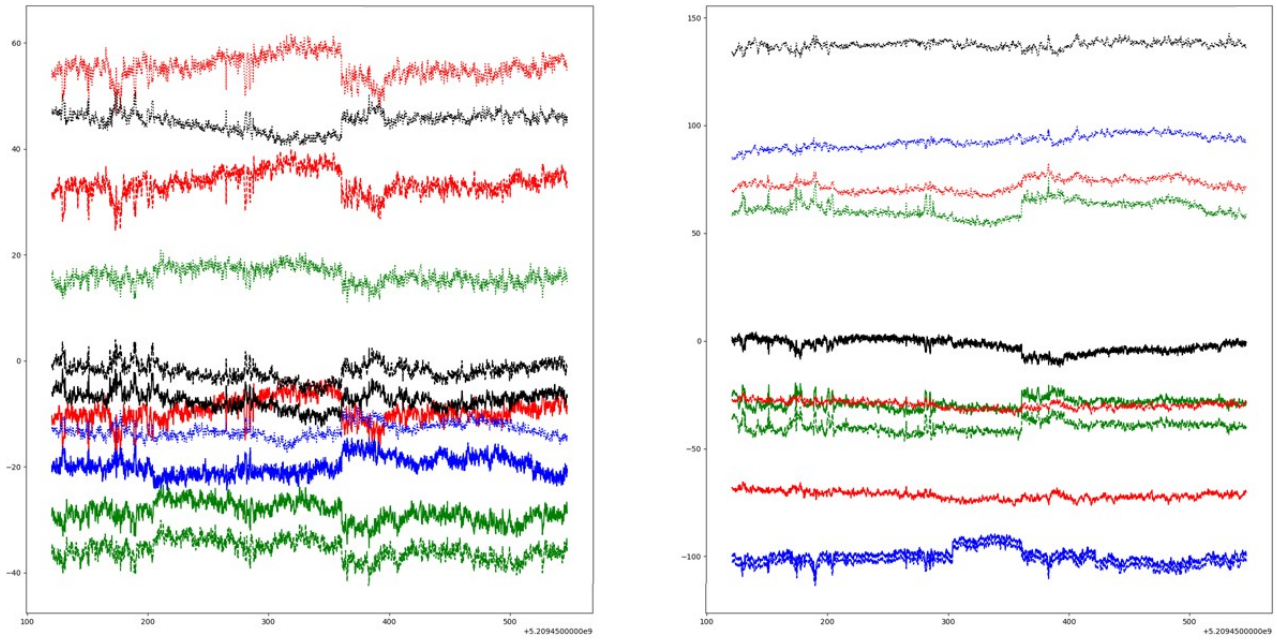


Figure 44: Residuals as in the top-row of Fig. 43. Clearly they diminished and do not show trend with the frequency.

2.3.3.6 Validating the linearization approach

In our routine *dWdB_Tbs()* we move the operation point, around which the linearization is performed, in each iteration not exactly to the improved solution but, for practical reasons, to its time averages (see Eq. 3.3.12). This allows us to keep the structure of linearized equations simple and solution cost-effective. Question arises, what price in terms of lost accuracy we pay for that. Can this simplification cause the increased residuals?

In order to answer the question, we have written another routine *dWdB_Tbs_NL()*, which uses 'brutal force' for solving the nonlinear (NL) system (3.3.10) directly and we compared the results with the linearized iterative version. The non-linear solution is CPU demanding so we can afford performing it only for few antennas, just as a verification. The difference between the linear (Eq. 3.3.11) and full NL (Eq. 3.3.7) solution are shown in Figs. 44 and 45 for PWV and brightness temperature, respectively.

As it can be seen, the error in PWV is of the order of micro-meters, so absolutely negligible, while in the case of brightness temperature it reaches low units of Kelvins, so, again, less than 0.1% of the nominal value. The magnitude of error correlates with the absolute value of PWV/T^B fluctuations as the exact solution in those points is than more distant form our operation point (in the linearized system) selected conventionally as the time-average of the exact solutions over a scan.

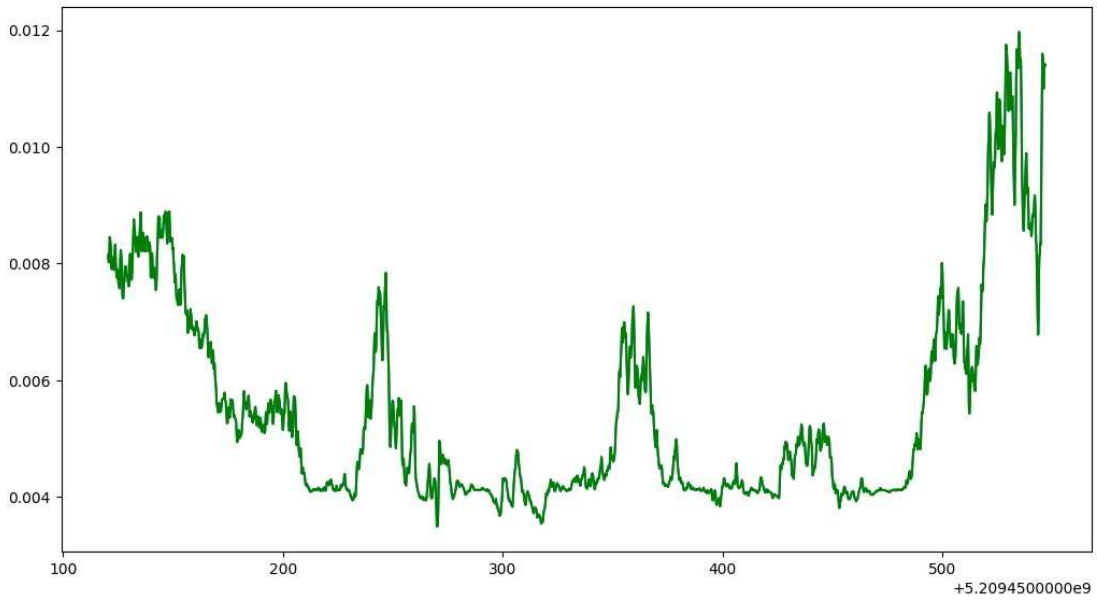


Figure 45: Difference of solutions for PWV (mm) as a function of time (MJDSec) between linearized equations (3.3.11) *and full non-linear solution* (3.3.10) at antenna id23.

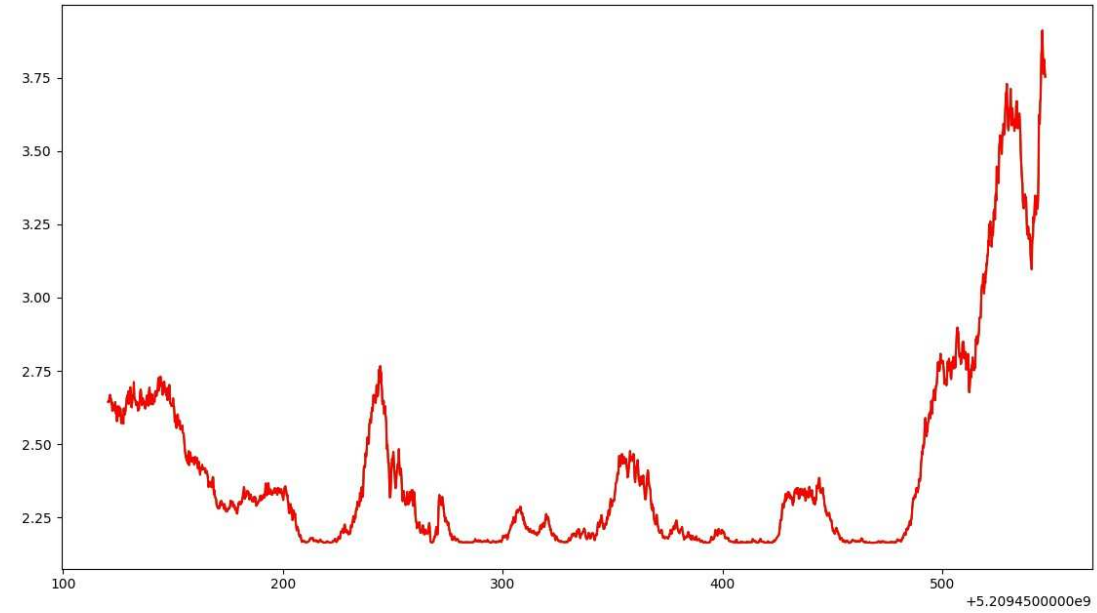


Figure 46: Like in Fig. 45 above, now for the brightness temperature in Kelvins.

2.3.3.7 Introducing the antenna efficiencies

The so far implemented model-upgrades improved the situation what concerns all respects – diminished residual errors of our over-determined system with no systematic trend (i.e. better fit), average PWV is closer to that reported by SYSCAL PWV median. Still the brightness temperatures seem to be underestimated w.r.t. the semi-empirical (and observation-validated) model in Alissandrakis et al. (2020).

We have realized that we have insofar ignored the antenna efficiencies – rear- and forward scattering and spill-over, traditionally designated as η_l and η_F. Earlier, we have somehow expected that they might be absorbed in the other instrumental parameters, namely into gain. But it is not true in full extent, because, in addition to diminishing the target and atmospheric signals (what lower gains does, too), they also bring the scattered radiation from the ground and skies to the antenna.

Inclusion of efficiencies changes the original expression for external temperature in Eq (3.3.5) to a new form:

$$\begin{aligned} T_{ext} = \eta_l \cdot \eta_F \cdot T_B^{Sun} \cdot \int tr(w(t), f) \cdot S(f) \cdot g(f) \frac{df}{BW} \\ + (1 - \eta_F + \eta_l) \cdot \int T_{sky}(w(t), f) \cdot g(f) \frac{df}{BW} \\ + (1 - \eta_l) \cdot T_{amb} \end{aligned} \qquad (3.3.14)$$

T_{amb} is the ambient temperature, here considered also as the ground temperature for the rear spillover contribution. The three lines in Eq. (3.3.14) mean, respectively, the solar contribution diminished by imperfect transmissivity through the atmosphere and by non-ideal antenna efficiencies, contribution from the skies, both arriving directly with the ray from the source and that from the forward spillover, and, eventually, the rear spillover contribution from the ground.

Eq. (3.3.14) can be inserted instead of (3.3.5) into the general relation (3.3.3) and we eventually get modified set (3.3.7). The ‘normal’ equations for iterative linearization (3.3.11) keep their form, just the Jacobian elements – the sensitivity factors, get modified by presence of the efficiencies. The solution of the system otherwise stays methodologically unchanged, the results are, of course different.

Results:

- Definitely improved resulting brightness temperature ~6900 K, which is now much closer to the expected value of 6990 K.
- PWV = 4.87 is now fully in line wit the SYSCAL estimation – se Fig. 47
- What still makes worries, are the residuals – they are significantly higher than those for the differential (fluctuation-only) approach. Namely, the mean value (over time) is high, at the level of 50 – 70 K, while the fluctuations are small (up to 10K) and comparable with the previous differential approach.

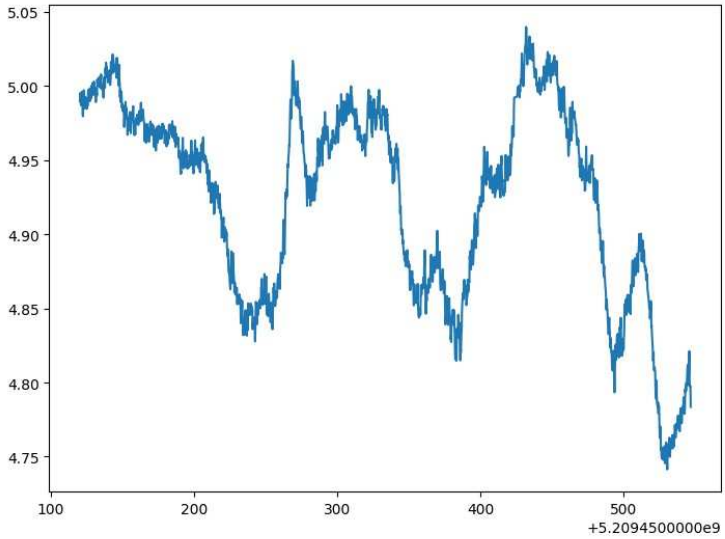


Figure 47: PWV in mm as a function of time (MJDsecs) for antenna id 21 (DA56). The model-upgrades including improved solar spectrum and antenna efficiencies have been applied, which has led to much better result than the 'naive' approach displayed in Fig. 41.

2.3.3.8 Improving antenna gains

Large residual mean-values seen in the top row of Fig. 43 and, somewhat improved but still quite high in Fig. 44, indicate that we have bad fit of our model on the LHS side to the measured temperatures at the RHS. Since we have focused on improving the model values in the previous paragraphs (and many improvements have been actually achieved) and we, perhaps, exhausted the possibilities for further upgrade of the model, we now focus on the measurement side. Moreover, the residuals now (after improving the model solar spectrum and including the antenna efficiencies) seem to be quite random with no clear trend related to the SPW frequency domain (the base-band BB), their values non-systematically differ from antenna to antenna – this also indicates problem rather in measurement than in the model.

Actually, the RHS temperatures still do not represent the pure measurement data because in Eq. (3.3.7) we are comparing the model and recorded temperatures, where the latter are only derived from the actual primary data (voltages/powers) via instrumental characteristics like gains G, receiver temperatures T_{RX}, and – in case of SQLD data – also the power offsets P_o. These instrumental parameters are pre-determined by the ACD (dual-load) calibration. The ACD calibration equations are solved by our routine *acdCaltabs()*.

In case of solar observations we have two levels of gain: One for the 'standard non-solar' (i.e. low) attenuation, which is derived from the ACD calibration solution, and second, with high attenuation, used during the solar scan. The scaling/attenuation factor *AF* between those two is derived by two

atmosphere scans: One with low (the ‘la’ index) and one with high (i.e., standard for the Sun) attenuation (see Shimojo et al. 2017)

$$AF \overset{\text{def}}{=} \frac{G}{G_{la}} = \frac{P_{sky} - P_o}{P_{off} - P_o}\,, \qquad (3.3.15)$$

where P_{off} data are taken in the solar off-source sub-scan with solar (increased attenuation). Technically, in general, this relation should be used for calculating the gain G solely in the SQLD spws, because the P_{off} signal is typically too low to be detected sufficiently accurately in the AC SPWs. In Band 5, however, the solar attenuation level is not so high and we are able to measure the P_{off} signals in AC SPWs, too.

Nevertheless, uncertainty around this method (using P_{off} scans in AC SPWs) has led us to searching for further improvement of determination of the scaling factor (3.3.15) for antenna gains between the low and high attenuation levels. The RHS values are typically ~3000K – 4500 K (indeed – the absorption in low-frequency SPW 5 at PWV~5mm – is more than 50%!). So, just 5% error in gain determination easily produces miss-fit of 150K – 200K, therefore it is important to estimate the gain as much accurately as possible.

The idea is following:

- We calculate the gain-to-gain scaling by Eq. (3.3.15) solely in the SQLD spws, in line with the approved procedure in solar ObsMode (see Shimojo et al. 2017).
- We shall transfer the scaling from the SQLD to the AC spws, that occupy the same frequency domain / base-band (e.g., from SPW 0 to SPWs 5 and 6).
- For the channel-resolved SPWs we assume that the scaling factor to gain is applied homogeneously to all frequency channel, i.e. that the spectral gain normalized to 1.0 does not change, when higher attenuation is switched on/off.

The transfer between SQLD and co-located AC spws is based on the idea that for the SPWs occupying the same frequency domain we have the same contribution in terms of radiation intensity (i.e. temperature) both from the Sun and from the atmosphere. Hence, the difference in received signal (powers) is just due to different instrumental characteristics (gain and generalized instrument-related power offset) in different SPWs. Hence, if we insert Eq. (3.3.14) for the external temperature together with definition of the instrumental temperature $T_{inst}=T_{RX}+ P_o/G$ into the general expression (3.3.3), after frequency averaging we get the following relation for power recorded in the SPW with index S

$$\overline{P_S} = \overline{G_S}\left[\overline{T^S_{RX}(f)\cdot g^S(f)} + \eta_l \eta_F \overline{tr^S}\cdot\overline{T^S_B}\cdot\overline{g^S(f)\,tn(f)\,S(f)} + (1-\eta_F+\eta_L)\cdot\overline{g^S(f)\,T_{sky}(f)} + (1-\eta_l)\,T_{amb}\right] + P^S_o \qquad (3.3.15a)$$

Here the over-bar means frequency averaging over the SPW and $g^S(f)$ is the spectral gain normalized to unity, defined already in Eq. (3.3.5), i.e. $g^S(f)=G^S(f)/\overline{G_S}$ and, similarly, $S^S(f)$ is the normalized solar spectrum in the SPW S, defined above (Eq. 3.3.5), too. $\overline{T}^S_B$ is the frequency-averaged solar brightness temperature in the SPW S and $tr^S(f)$ is atmospheric transmissivity in the same window. We define also

its normalized (to unity) counterpart as $tn(f)=tr(f)/\overline{tr}$, analogically to spectral normalized gain and normalized solar spectrum. In case of SPWs that are not channel-resolved, all these normalized quantities are 1.0 by definition, and the frequency-averaged quantities in the relation (3.3.15a) above (e.g., gain and receiver temperature) are, obviously, represented by their single-channel values as directly obtained from the MS data. The atmospheric + CMB brightness $T_{sky}(f)$ is defined by Eq. (3.3.6).

From Eq. (3.3.15a), applied onto two SPWs – the AC and SQLD in the same base-band (what assures the same source and atmospheric contributions) – one can express relation between the received powers in those two SPWs. The relation should be linear, in the form

$$P_{SQLD}=K\cdot P_{AC}+Q$$

where we have skipped the over-bars, as it is clear that we work with frequency-averaged (per-SPW) quantities, and the parameters K and Q can be expressed from Eq. (3.3.15a) as:

$$K=\frac{G_{SQLD}}{G_{AC}}\cdot SWF$$

$$Q=G_{SQLD}\cdot\left[T_{RX}^{SQLD}-SWF\cdot\overline{T_{RX}^{AC}(f)\,g^{AC}(f)}+(1-\eta_F+\eta_l)\cdot[\overline{T_{sky}}-SWF\cdot\overline{g^{AC}T_{sky}}]+(1-\eta_l)T_{amb}(1-SWF)\right]+P_o^{SQLD}$$

where the spectral-weight factor SWF is the abbreviation for

$$SWF=\frac{1}{\overline{g^{AC}(f)\,tn^{AC}(f)\,S^{AC}(f)}}$$

The parameters K and Q of the linear relation between P_{AC} and P_{SQLD} can be determined "experimentally", by means of linear regression (correlation) between actual recorded values of P_{AC} and P_{SQLD} in the target scan – see Fig. 48. This way we may eventually express the gain in the AC SPW in relation to the gain (with solar high-attenuation) in the corresponding SQLD spw as

$$G_{AC}=\frac{G^{SQLD}}{K}\cdot SWF \qquad (3.3.15b)$$

where the parameter K has been obtained as a regression coefficient as described above. For the AC channel-resolved SPWs we assume, as adduced above, that the low/high attenuation switch affects just the gain scaling, not its (normalized) spectral profile $g^{AC}(f)$. Hence, the per-channel high-attenuation gain in the AC-resolved SPWs, transferred from the same-baseband SQLD SPW reads:

$$G_{AC}(f)=\frac{G^{SQLD}}{K}\cdot SWF\cdot g^{AC}(f) \qquad (3.3.15c)$$

To sum up the methodology:

1. We calculate the high-attenuation (solar-scan) gains according to Eq. (3.3.15) solely for SQLD spws.
2. Using linear correlation between the powers P_{AC} and P_{SQLD}, recorded during the solar scan in the AC and SQLD spws in the same BB, we find regression coefficient K.

3. Using Eqs. (3.3.15b) or (3.3.15c) for AC-integrated or AC-resolved SPWs, respectively, we transfer the high-attenuation gains from SQLD to the respective AC spws.

The entire procedure is implemented in the improved ACD calibration routine *acdCaltabs2()*.

Results: The situation got further improved – PWV slightly raised to 4.83mm, solar brightness temperature remained 6900 K/6800 K for XX/YY polarizations, the residuals diminished by 30 – 70% depending on antenna. Still they remain high – their means are much higher than their fluctuation part, which signalizes imperfect fitting of the model and observed data – see Fig. 49.

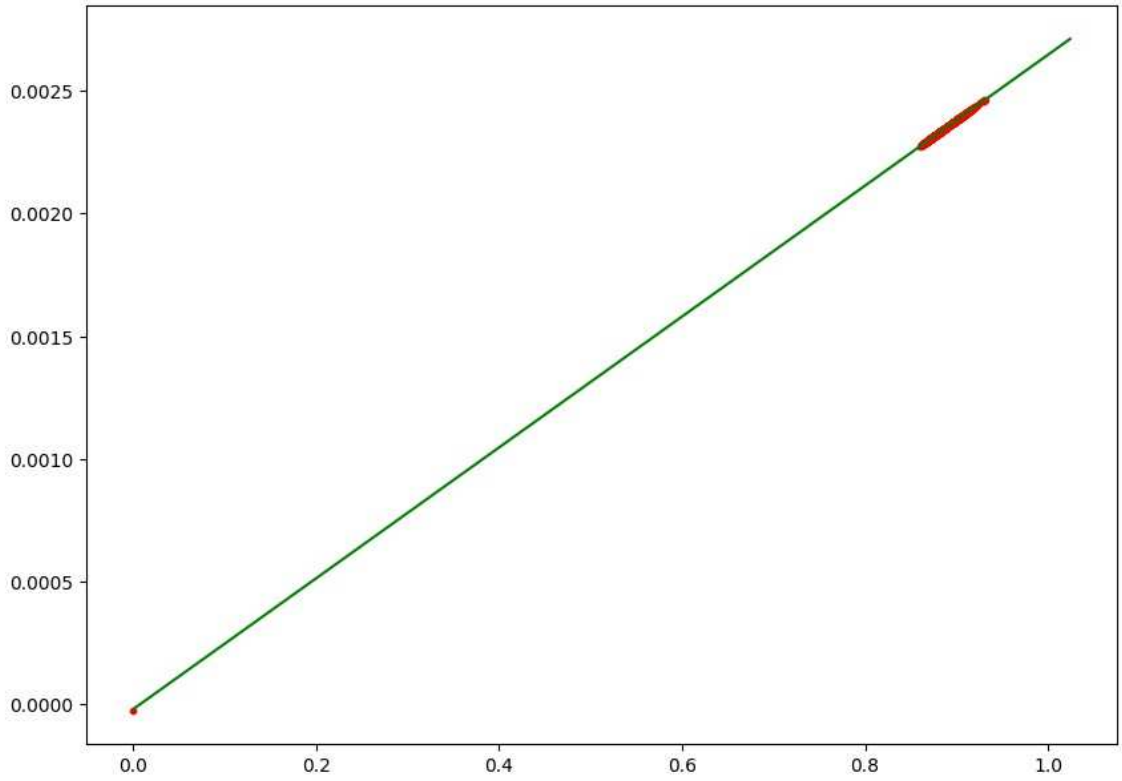


Figure 48: Cross-correlation between recorded powers in the AC SPW 5 (channel-averaged) and SQLD SPW 0 in the same BB. The red point close to zero corresponds to the point [0;Q], where Q is the regression offset, the higher values (around 0.9 for P^{AC}) represent the scatter-plot between P^{AC} and P^{SQLD} for the solar on-source sub-scans.

2.3.3.9 High- vs. low-attenuation gain scaling: Issue with time dependence

In the previous paragraph we have developed a method on how to scale the gain and the instrumental power-offset from SQLD spws to the AC spws, in line with the approved *Solar ObsMode* strategy.

Note, that the insofar used ‘official’ solar calibration implemented as *sol_ampcal2()* routine in *sun_reductionUtils.py* in fact works with antenna temperature T_{ant} (see Shimojo et al. 2017; White et al. 2017) and not directly with gains. Nevertheless, the principle and formulae used for calculations are the same, albeit in our method *acdCaltabs2()* we, moreover, in addition to the ACD ambient and hot loads (used in *sol_ampcal2()*), make also use of correlations for the time-varying solar signal between AC and SQLD spws, what improves statistics.

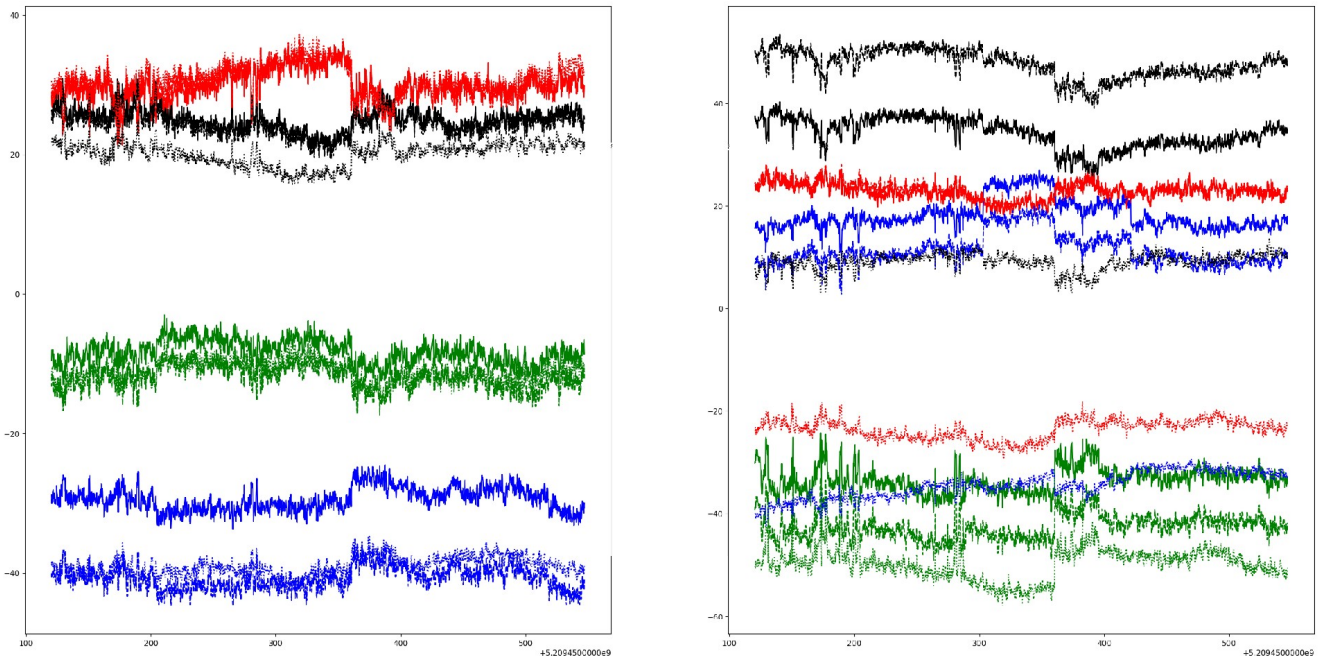


Figure 49: Residual errors like in Fig. 44 after the "solar" (i.e. high-attenuation state) gain has been transferred to the AC SPW from their respective SQLD counterparts in the same BB via Eqs. (3.3.15b) or (3.3.15c). The mean values of fluctuation residuals are further diminished. The same colors, i.e. the same Bbs, are now grouped together, but in this case the residual in each BB does not systematically increase with frequency, the order is random. Clearly, the error in the BB is determined by accuracy of the solar gain in the leading SQLD SPW.

Now, the question is, how good is the scaling in line with Eq. (3.3.15) for the SQLD spw itself, which is used as a gauge. The issue is related to time-dependence of the P_{off} values used in the relation (3.3.15). We have it determined from two (off-source) sub-scans of the target scan – one very first on the beginning of the target scan and another one at the very end. The value of P_{off} clearly drops in our MS between the leading and trailing sub-scan. There are basically three ‘attitudes’ to this fact:

- One interpretation may say that it is because the antenna-gain itself is time dependent. In such a case, it has sense to make linear time interpolation for the scaling factor in Eq. (3.3.15) between the leading and trailing sub-scans and apply the time-dependent high-attenuation (solar) gain to the solar data. This was our original (until April 2026) implementation in *acdCaltabs()*.

- Second approach, silently applied in the standard solar calibration in *sol_ampcal2()* assumes that the differences in P_{off} between the leading and trailing sub-scans are more-or-less random, so it takes time average from all integrations in both sub-scans together, creating a single value of P_{off}. So, in fact, we are taking the average of P_{off} values from leading and trailing sub-scans.
- Eventually, and we are convinced that it is the right way, we may interpret the decrease of P_{off} as a consequence of decreasing PWV (the decrease is clearly indicated by the PWV curve from SYSCAL table), and consequently decreasing emissivity of the atmosphere. With this interpretation, it makes sense to collect the P_{off} data from the first sub-scan only (ignoring the trailing one), because the leading sub-scan follows immediately after the ATM scan, so the PWV, hopefully, did not change too much.

We compared the approaches no. 1 (time-interpolation between leading/trailing sub-scan) and 3 (leading sub-scan only) and combined them with two methods of determining the scaling factor (3.3.15): (i) each SPW, including the ACs, collects P_{off} values directly from its own data (using the original *acdCaltabs()*) and, (ii) we transfer the scaling factor (3.3.15) to ACs from the SQLDs using Eqs. (3.3.15b) or (3.3.15c) (*acdCaltabs2()*). Results are shown in Fig. 50. We present the residual errors as we did in the top row of the Fig. 43 and in Figs. 44 and 49. Here, the top row shows the residuals for P_{off} collected in each SPW and time interpolation between leading/trailing sub-scans, the middle row displays approach with SQLD-only based (and transferred to AC spws) gains, still with the time-interpolation, and the bottom row displays the gains calculated from SQLD-only approach, with P_{off} collected in the leading sub-scan only. Namely the top row with high residuals shows sort of 'divergence' of the residuals, indicating that the time interpolation for the gain actually worsens the situation – so the P_{off} drop in time is not a consequence of the gain change. To some extent this effect is visible also in the middle row, which has, however, largely diminished residuals thanks to transfer of the high/low-attenuation gain-scaling (3.3.15) form SQLDs to ACs. Eventually, the bottom row has no time trend and as a whole the lowest residuals, indicating that the first sub-scan is the best choice for the P_{off} measurement and subsequent use in the gain scaling according to (3.3.15).

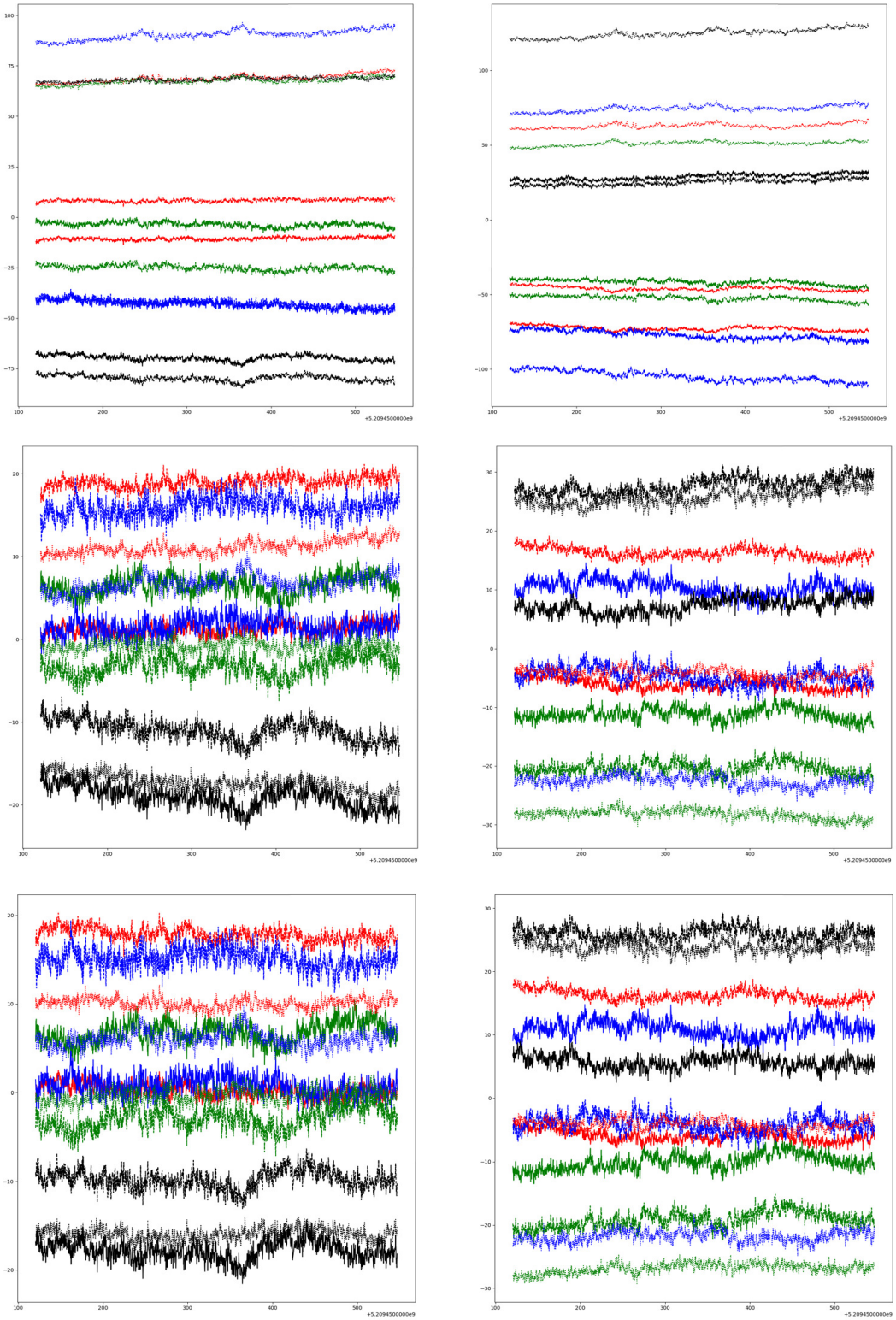


Figure 50: Comparison of residual errors (see Fig. 43 for description) for various approaches to solar (high-attenuation) gain. Top: P_{off} measured in all SPWs (incl. the ACs), leading and trailing off-subscans linearly interpolated. Middle: P_{off} only in SQLDs, transfer to ACs using Eq. GGG, linear interpolation of leading and trailing subscans. Bottom: With transfer to ACs, using leading sub-scan only for P_{off}.

Results: We see time variation (a drop) of the P_{off} between the leading and trailing off-source sub-scans of the target scan in our MS. This fact rises a question, which sub-scan should be used for P_{off} determination and subsequent calculation of the gain-scaling between the low- and high-attenuation regimes according to Eq. (3.3.15). We can use either the average of both (which the standard calibration procedure *sol_ampcal2()* does), or linear time interpolation between both sub-scans (what we have done until April 2026), or, take just the first sub-scan that is immediately following the ATM scan, from which the P_{sky} value in Eq. (3.3.15) is derived. Having the best results with the third option, we are using solely this setup in the following. Anyway, it looks like more general problem, at least for all the Band 5 solar data (in other ALMA Bands the emissivity is not so strongly dependent o the PWV). Also in view of the next paragraph, where we allow for limited corrections of gains, it would worth considering to measure the gain-scaling factor between the low (standard-sky) and high (solar) attenuation levels not between the two off-source sky observations (as the sky brightness may change due to changing PWV), but rather during the ACD calibration: Having the hot and ambient sub-scans once with low and once with increased attenuation levels. We discuss this further in the final Chapter 4, where recommendations for improvements of the *ALMA Solar ObsMode* are summarized.

2.3.3.10 Instrumental parameters as a part of solution

From the above results it is clear that we are making progress towards lower residuals (i.e. better fit) when trying to increase accuracy in gain estimation. Insofar we have considered the instrumental parameters: gain G^{SP} and generalized instrumental power-offset $P_{inst}^{SP} \equiv G^{SP}.T_{RX}^{SP}+P_o^{SP}$ for each *SPW:POL* (as indicated by the superscript SP) data channel as a set of constants, pre-determined apriori by the ACD calibration (implemented in *acdCaltabs()* routine or its improved version *acdCaltabs2()* in *sol_wvrgcal.py* collection). However, in fact, we may think about parameters G^{SP} and P_{inst}^{SP} to be the variables of the system as well and further improve the results. As argued above, we have *nTimeSamples x nPols x nSPWs* equations in the set (3.3.8), while, adding the instrumental powers and gains to the set of variables (i.e., appending them to the already calculated 'natural' variables PWV(t), $T^b_{xx}(t)$, and $T^b_{yy}(t)$), we got (3 x *nTimeSamples + 2 x nPols x nSPWs)* total variables, which is still much less than the number of equations (having the quite high number ~2300 of time samples/integrations). Hence, the system of equations is still (over)determined.

Formally, we extend our unknown vector of variables by the instrumental parameters, so it can be written as $x_j=\{w(t),T_{XX}^B(t),T_{YY}^B(t),P_{inst}^{SP},G^{SP}\}$, where the indices *t*, *S*, and *P* take ranges *t=1..nSamples*, *S={5,7,9,11,6,8,10,12,0,1,2,3}*, and *P={XX,YY}*. Inclusion of *G* and P_{inst} into the vector of unknown variables requires that we have to return back to measured powers, not transformed to temperatures by the apriori preset gain scaling. This makes situation cleaner – the RHS now contains just directly measured quantities, while instrumental parameters *G* and P_{inst} become part of the model.
The master equation used in our further calculations for power $P_{rec}^{SP}(t)$ directly recorded in SPW *S* and polarization *P* in time instant *t* reads as follows:

$$P_{rec}^{SP}(t)=P_{inst}^{SP}+G^{SP}\cdot T_{ext}(t) \qquad (3.3.16)$$

The instrumental power offset has been already defined above as $P_{inst}=G.T_{RX}+P_o$, where P_o is non-zero only for SQLD spws. T_{ext} characterizes radiation intensity from all external contributions outside of the receiver, i.e. the observed target (diminished by absorption and antenna inefficiencies), intrinsic radiation of the terrestrial atmosphere, and spillovers from ground (rear-spillover + antenna resistivity at ambient temperature) and skies (at lower T_{sky} temperature).

If we insert (3.3.14) into (3.3.16), we get, again, a very large set of equations, because the combined system must hold for every *spw:pol* data channel and for every time instant (integration):

$$\begin{aligned} P_{inst}^{SP} + G^{SP}\eta_l\cdot\eta_F\cdot T_B^P(t)\cdot\int tr^S(w(t),f)\cdot S^S(f)\cdot g^{SP}(f)\frac{df}{BW} \\ + G^{SP}(1-\eta_F+\eta_l)\cdot\int T_{sky}^S(w(t),f)\cdot g^{SP}(f)\frac{df}{BW} \\ + G^{SP}(1-\eta_l)\cdot T_{amb} \;=\; P_{rec}^{SP}(t) \end{aligned} \qquad (3.3.17)$$

Eq. (3.3.17) again takes the general form of Eq. (3.3.8) and it is solved by finding minimum of the respective quadratic form (the least-square of residuals in Eq. 3.3.9), which leads to the non-linear system (3.3.10). As a consequence of widening of the unknown-variables vector by the instrumental part, we got also a bit wider Jacobian matrix of the system (by *2 x nPols x nSPWs* = 48 new columns). The system (3.3.11) in its 'narrowed' form $J_{kj}.(x_j-x^o_j)=b_k-f_k(x^o_j)$, extended by the instrumental variables, is schematically shown in Fig. 51. The RHS vector $C_k\equiv b_k-f_k(x^o_j)$ represents the difference between measured quantities and the model value in its operation point. The next Fig. 52 shows the 'normal' equations (3.3.11), i.e. after multiplication by J^T, for our extended system. In order not to repeat all the matrix elements, we work with *N-blocks* and *G-blocks*, which are the sub-Jacobians for the 'natural' (i.e. $w(t)$ and $T_B^{XX/YY}(t)$) and the 'gain/instrumental' (i.e., G^{SP}, P^{SP}_{inst}) subset of unknown variables. If we define our operation point x^0_j, around which the linearization is performed, again by the scan-averaged 'natural' quantities complemented by the current-iteration 'gain/instrumental' parameters, i.e. $x^o=\{<w(t)>,<T_{XX}^B(t)>, <T_{YY}^B(t)>, P_{inst}^{SP}, G^{SP}\}$, the *N*- and *G*-blocks are all identical (the *G*-blocks do not depend on time, so in their case it is, actually, automatic), what simplifies situation a lot. The matrix $(J^T.J)$ for extended linearized system displayed in Fig. 52 has a bit more complicated structure: Its main (far biggest) part is composed of *nTimeSamples* identical *3x3* diagonal blocks of the form $(N^T.N)$, otherwise zeros – which is the same as we have had with the constant gains and instrumental power offsets – and, as a consequence of new ('gain/instrumental') columns in the Jacobian we have two 'belts' – one, the lower belt, as 48 appended rows and one, which is just the transposition of the former, is appended as 48 last columns at the right side of the matrix. The low and side belts are formed from blocks of mixed matrix products: $(N^T.G)$ and $(G\ .N^T)$, respectively. The entire square matrix $(J^T.J)$ is eventually completed by 48 x 48 block in the very lower-right corner, which takes form of $nTimeSamples.(G^T.G)$.

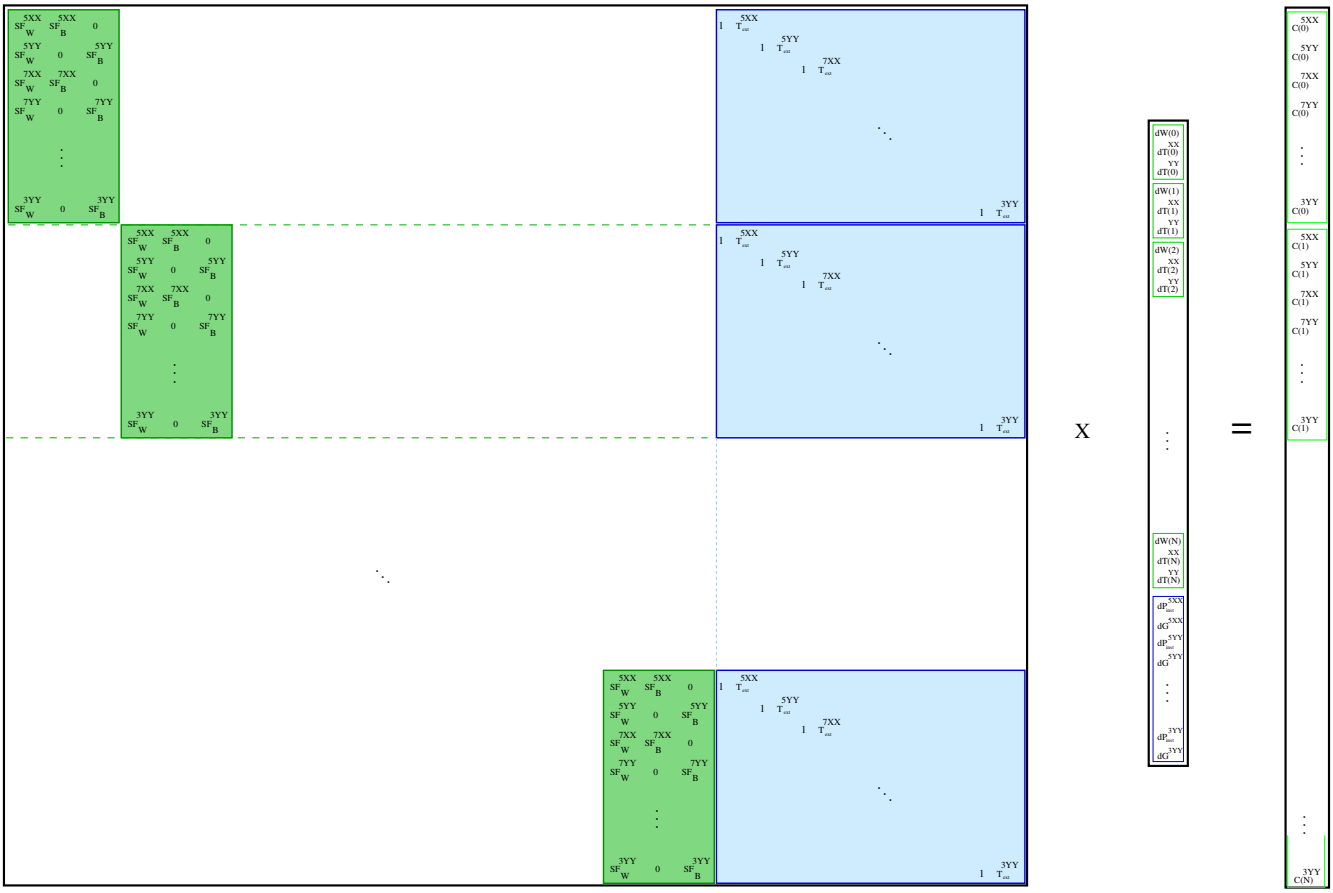


Figure 51: System for 'narrowed' Eq. (3.3.11) *with a vector function $f_k(x_j)$ prescribed by Eq. (3.3.17), extended by the instrumental variables (blue-framed block at the end of the unknown-variables vector), and corresponding identical blocks (the blue squares at the right) in the Jacobian matrix. The empty spaces in the matrix blocks mean zero elements. In further figures we will not explicitly show the elements and will refer to the green blocks as N-block ('natural' variables) and the blue one as G-block (for 'gains').*

Linearized system (3.3.11) with the extended Jacobian (Fig. 52) cannot be directly separated to *nTimeSamples* independent systems for each time extra. The instrumental part, which is common for all time instants, introduced sort of coupling among the single-time solutions: This is manifested by the matrix structure, whose additional side- and lower-belts destroy the simple block-diagonal form (which we have with constant instrumental parameters) and thus prevent the decomposition. In principle, it could be solved by *np.linalg* package, but it takes really long time as the matrix of the system is very large (~8000 squared). Nevertheless, the main part of the system matrix, composed of *nTimeSamples* identical ($N^T.N$) blocks (which are of *3x3* size each) is easily invertible (it is a block-diagonal matrix) and there is an expression related to the Woodbury formula (see Fig. 53), which allows for inversion of such a square matrix, when it is extended by a row, a column, and the lower-right corner element. The formula have been found with help of Gemini AI in Google with no reference to literature reported. But we proved that it is correct: While finding the inversion is difficult, its correctness can be shown easily by direct multiplication of inverted and original matrix. Moreover, we have tested it by a simple routine

extMatInv() in our collection *sol_wvgcal.py*. The formula works even if the added row/column are, in fact, belts, so it can be used for our purpose, too. Inverted large-system matrix (after additional inclusion of ACD calibration equations – see next paragraph below) is schematically shown in Fig. 56.

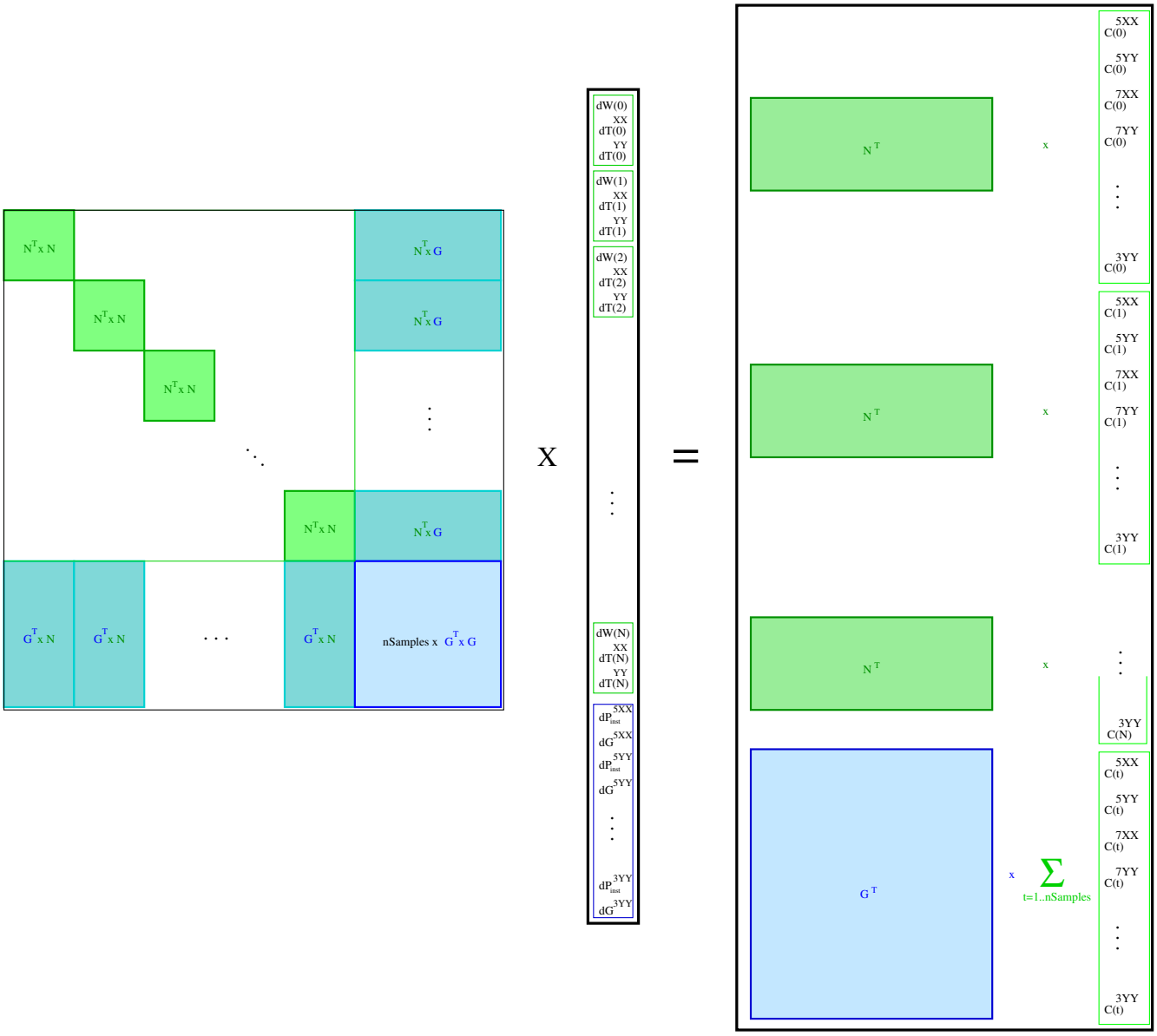


Figure 52: Normal equations (3.3.11) *for the system extended by instrumental variables – i.e. the system in Fig.* 51 *multiplied by J^T. The J^TxJ matrix at the LHS is composed from identical blocks, formed by products of the blocks (N and G) of the Jacobian J. The pre-extension original matrix is formed just by the green N^TxN blocks. The extension by instrumental part has led to a newly added left and bottom belts, composed of mixed instrumental (G) and natural (N) blocks (and their transpositions), and the 48x48 square block in the lower-right corner, composed solely from the G-blocks. The RHS vector represents a multiplication of the original RHS vector C from Fig.* 51 *by the transposed Jacobian J^T.*

AI Overview

If you have an invertible matrix A (size $n \times n$) and you know its inverse A^{-1}, the inverse of the matrix M extended by a row and a column can be computed efficiently.

If the extended matrix M is defined as:

$$M = \begin{pmatrix} A & b \\ c^T & d \end{pmatrix}$$

where b is a column vector, c^T is a row vector, and d is a scalar, and if the Schur complement $S = d - c^T A^{-1} b$ is non-zero, then the inverse M^{-1} is:

$$M^{-1} = \begin{pmatrix} A^{-1} + A^{-1} b S^{-1} c^T A^{-1} & -A^{-1} b S^{-1} \\ -S^{-1} c^T A^{-1} & S^{-1} \end{pmatrix}$$

Detailed Formula Components

1. **Inverse of the Schur Complement (S^{-1}):**
   Calculate $S = d - c^T A^{-1} b$. The scalar S is the Schur complement of A in M.
   $S^{-1} = \frac{1}{d - c^T A^{-1} b}$

2. **Extended Top-Left Block:**
   $A^{-1} + (A^{-1} b) S^{-1} (c^T A^{-1})$

3. **Extended Top-Right Block:**
   $-A^{-1} b S^{-1}$

4. **Extended Bottom-Left Block:**
   $-S^{-1} c^T A^{-1}$

5. **Extended Bottom-Right Block:**
   S^{-1}

Figure 53: Efficient method for calculation of inversion of the matrix that takes the form of easily invertible square matrix, complemented by lower and right belt and the lower-right square element as found by the Gemini AI. Matrix multiplication of the results M^{-1} with the original matrix M easily proves the correctness. Our matrices in Figs. 52 and 55 have exactly this form. We have implemented this method in solution of Eq. (3.3.11) in the routine dWdBdG().

Having the $J^T.J$ matrix inverted, we can directly write solution of the extended system (3.3.11). Then, using the iterations (3.3.12) for the operational point we converge to the approximate solution of the

full non-linear system (3.3.10), as usually. Entire this methodology has been implemented in the early version (later updated – see next paragraph) of *dWdBdG()* routine in our *sol_wvgcal.py* collection.

Results: Serious issue has been found with the super-high condition number of our extended ($J^T.J$) matrix – is almost singular. The reason is clear – we allowed the instrumental parameters (gains and P_{inst}) to take *any* value, absolutely independent of the pre-calculated ACD calibration solutions. As a result, the system converges to a solution with the least residuals, but it is apparently wrong (solar Tb ~ 2000 K, PWV ~7mm, gains shifted by ~100% of their nominal ACD calibration solution). Clearly, we need some anchor for the instrumental parameters.

2.3.3.11 Adding the ACD calibration equations into the system

The naive approach to instrumental parameters as new variables of the system failed as they are completely unbound in the new extended system of equations. We need some anchor for them and this can be achieved by adding also ACD calibration equations into our system. Actually, the scientific and calibration equations take the same form – we repeat here Eq. (3.3.16), which represent a framework for the science data-to model fit, for convenience:

$$P^{SP}_{rec}(t) = P^{SP}_{inst} + G^{SP} \cdot T_{ext}(t)$$

The ACD calibration equations are formally identical, just the external temperatures are replaced by ambient and hot loads:

$$\begin{aligned} P^{SP}_{amb} &= P^{SP}_{inst,la} + G^{SP}_{la} \cdot T^{SP}_{amb} = AF^{SP} \cdot \left(P^{SP}_{inst} + G^{SP} \cdot T^{SP}_{amb} \right) \\ P^{SP}_{hot} &= P^{SP}_{inst,la} + G^{SP}_{la} \cdot T^{SP}_{hot} = AF^{SP} \cdot \left(P^{SP}_{inst} + G^{SP} \cdot T^{SP}_{hot} \right) \end{aligned} \qquad (3.3.18)$$

Here, the '*la*' subscript in $G^{SP}{}_{la}$ and $P^{SP}{}_{inst,la}$ stands for low-attenuation gains in spw:pol data channel *SP,* while high-attenuation gain used in the solar scan has no distinguishing index. The scaling/attenuation factor AF^{SP} between the low and high attenuation gains for spectral window *S* and polarization *P,* is calculated as a ratio of signals between the atmospheric scan (low attenuation) and the solar off-source sub-scan (high attenuation) in the SQLD spws – see Eq. (3.3.15). We scale the instrumental power P_{inst} by the same factor, because it is essentially $G.T_{RX}$, (for P_o see Eq. 3.3.18a) and we assume (in line with solar calibration methodology) that T_{RX} remains constant during switching the attenuation from low to high.

The calibration equations (3.3.18) may be actually added with weight different from 1.0 – if we increase the weight to a high number, we force the instrumental parameters to obey the calibration equations more exactly. Indeed, in the limit of the weight going to infinity we arrive to the same solution as we have found with the fixed/constant parameters G^{SP} and $P_{inst}{}^{SP}$ and shown in Figs. 47 and 49. On the other hand, with the low weight we allow for lower residual errors of the combined system on account of large shifts of the gains from their nominal ACD solutions. So the weight factor has to be optimized. We leave this task to the next paragraph and finish here with construction of matrices J_{ik} and $(J^T.J)$ for the system extended by Eq. (3.3.18). We implement the increased weight by repeating the calibration Eq. (3.3.18) multiple times – let *calWeight* factor is the number of repetitions. In the regular

system of linear equations such a repetition does not add any new information into the system. But here, in the least-square solution of the over-determined system (3.3.8), we enforce that the calibration equations shall be better fitted. With low *calWeight* ~ 1 the dominance of the science-data equations (we have ~2300 time samples in sci-data vs. single calibration measurement) would tend to diminish the sum of residual-squares by fitting better the science data. If we repeat the calibration subset *calWeight*-times, we can balance the dominance of the science-data equations in the system.

After addition of the ACD calibration equations (3.3.18) into system with *calWeight* repetitions, the Jacobian matrix J_{ik} for $f_k(x_j)$ in Eq. (3.3.11) gets extended by *calWeight* x *2 x nPols x nSPWs* new rows, where the factor 2 comes from ambient and hot load measurements – see Fig. 54. We refer to those newly added block matrices (*48x48*) as to *A-blocks* and, specifically, their upper and lower halves for ambient and hot loads will be referred as *AA* and *AH* sub-blocks (24 rows x 48 cols each). At the right-hand side of equation (3.3.11) we now have appended expressions $b_k - f_k(x^o_j)$ for the ambient- and hot-load powers, i.e. similarly to science-SPW powers we have to make a difference between actually measured P_{amb} and P_{hot}, and the corresponding model-value at the operation point x^o_j. Direct calculations give for those 'differential' ambient and hot powers at the RHS of the system shown in Fig. 54 expressions

$$\begin{aligned} dP^{SP}_{amb} &= P^{SP}_{amb} - AF^{SP}\cdot\left(P^{SP}_{inst} - G^{SP}\cdot T_{amb}\right) + P^{SP}_{o}\cdot\left(AF^{SP} - 1\right) \\ dP^{SP}_{hot} &= P^{SP}_{hot} - AF^{SP}\cdot\left(P^{SP}_{inst} - G^{SP}\cdot T_{hot}\right) + P^{SP}_{o}\cdot\left(AF^{SP} - 1\right) \end{aligned} \qquad (3.3.18a)$$

where G^{SP} and $P^{SP}{}_{inst}$ represent our current-iteration solution for the instrumental variables.

Nevertheless, structure of the LHS matrix $(J^T.J)$ of the normal linearized system (3.3.11) is not changed – still we have *(3 x nTimeSamples) x (3 x nTimeSamples)* main (far biggest) part in upper-left corner, which itself is composed from *3x3* small identical blocks at its diagonal (all off-diagonal matrix elements being zero), the *(2 x nPols x nSPWs (= 48)) x (3 x nTimeSamples)* lower belt, which itself is composed of *nTimeSamples* identical blocks of sizes *48x3* each, further its transposition appended to the main part as the right belt, and, finally, *48x48* lower-right corner. The structure of the $(J^T.J)$ matrix and its inversion is shown in Figs. 55 and 56. One can see that addition of the ACD calibration equations (3.3.18) appended a set of new identical *A-blocks* into Jacobian *J* (Fig. 54), so the $(J^T.J)$ matrix (Fig. 55) is, of course, changed, but its structure remains the same as before the ACD equations have been included. Hence, the efficient-inverison formula from Fig. 53 can be used for the system as well. Namely, inclusion of the ACD equations proscribes into Schur complement *S* (in Fig. 53), which reads:

$$S = nS\cdot G^T\cdot G + cW\cdot A^T\cdot A - nS\cdot G^T\cdot N\cdot J_N^{-1}\cdot N^T\cdot G \ , \qquad (3.3.19)$$

where *nS* and *cW* are abbreviations for *nTimeSamples* and *calWeight*, respectively, and $J_N^{-1}=(N^T.\,N)^{-1}$.

It is already straightforward to multiply the RHS vector in Fig. 55 by the inverse matrix in Fig. 56 to find the unknown-variables vector.

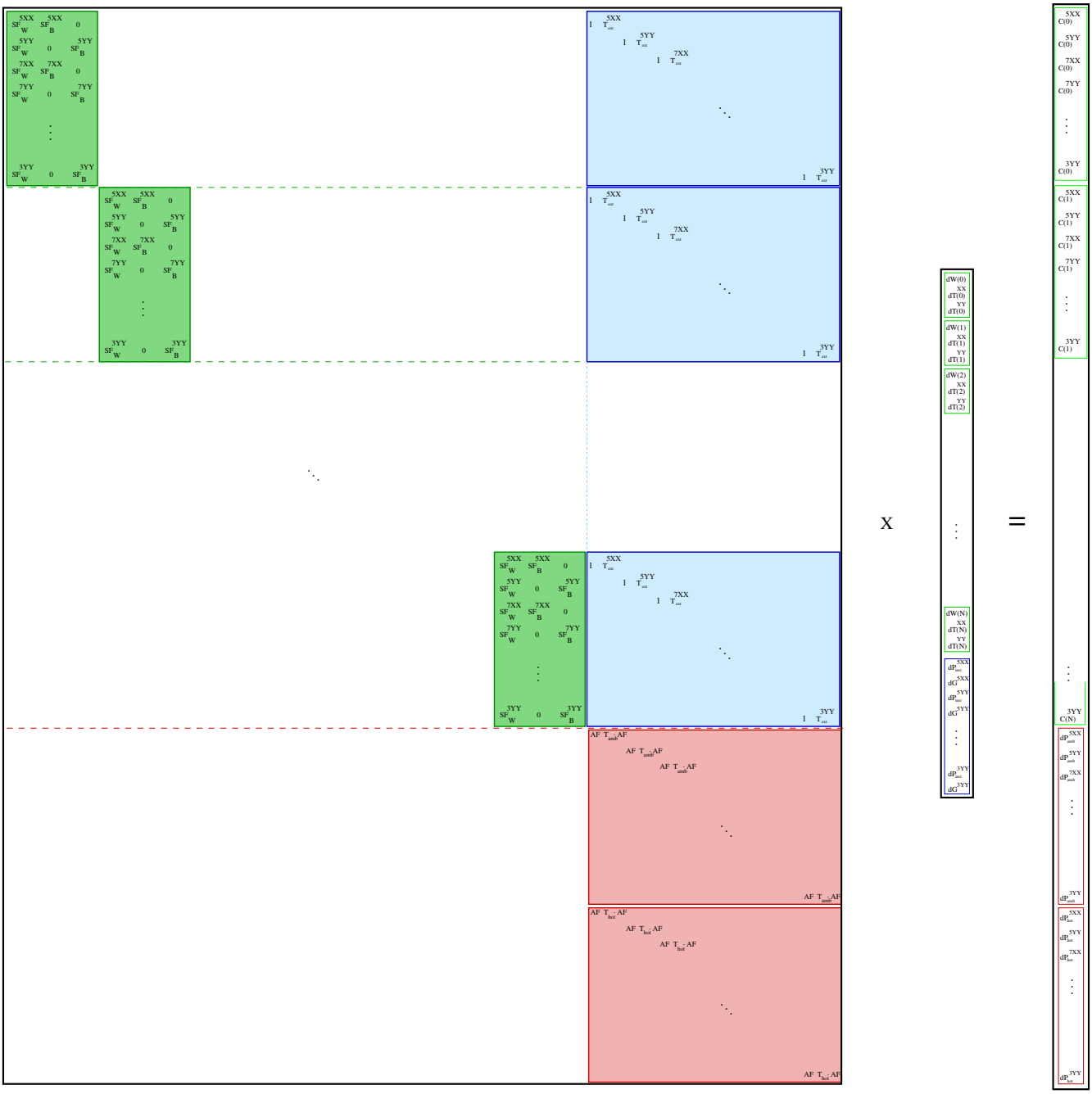


Figure 54: System from Fig. 51 extended by the set of the ACD calibration equations (3.3.18) . *The ACD calibration equations appear as two new red sub-blocks at the bottom of the matrix – the red rectangles with Jacobian elements for the ambient and hot loads. In the following they shall be referred as AA and AH sub-blocks, forming a single (AA | AH)^T block, referred jointly as the A-block. At the RHS, the added ACD equations form the red-framed vectors of the differential (see Eq. (3.3.18a) in text) ambient- and hot-load powers for each SPW S and each polarization P. The red A-block and correspondent RHS vectors may repeat* calWeight*-times, this way we increase relative weight of the ACD calibration equations in the system – see explanation in text.*

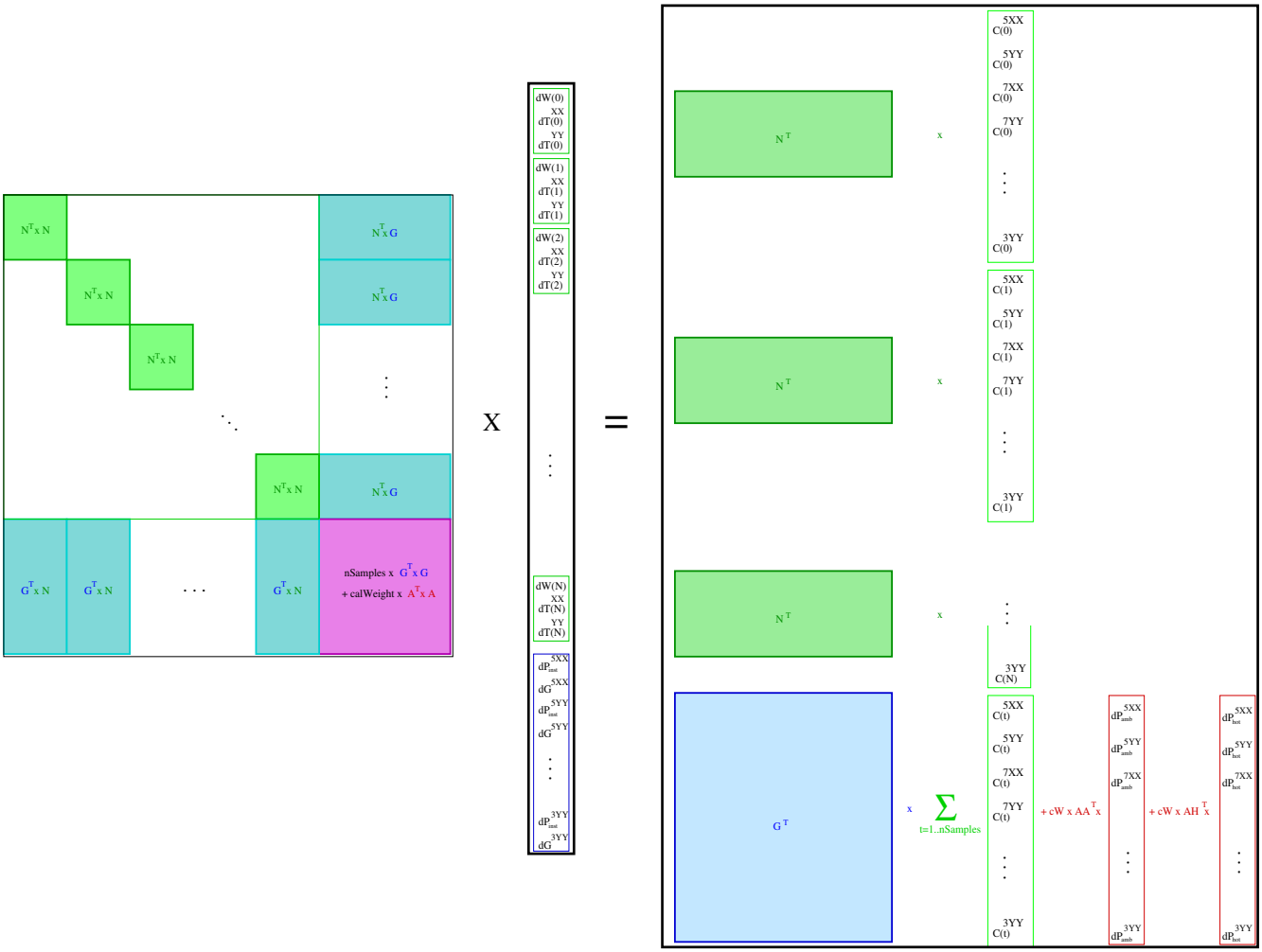


Figure 55: Linearized normal system formed from the system in Fig. 54 by multiplication by J^T. Symbol cW *at the RHS is abbreviation for calWeight, the factor controlling relative importance of the ACD calibration equations in the entire system.*

The matrix-vector multiplication can be, again, done block-wise, so the entire complex system with dimension (*3 x nTimesamples + 2 x nPols x numSPWs ~ 8000*) squared has been effectively decomposed to *nTimeSamples 3x3* systems plus inversion of the Schur complement, which is of size 48 squared. CPU time decreased, in comparison with the 'brutal-force' solution of the grand-system matrix (~8000 squared), enormously, the routine dWdBdG() now spends some 10% of time in the system solution, while its bottle-neck is in reading the MS data.

Just the final technical note: In order to improve numerical accuracy, we calculate the temperatures internally in kiloKelvins – this way the sensitivity factors $SF_W{}^S$ and $SF_B{}^S$ for variations of PWV and brightness temperature, which form the Jacobian elements, are numerically comparable. This diminished rounding errors in extensive matrix operations performed in the iterative solution of linearized Eq. (3.3.11).

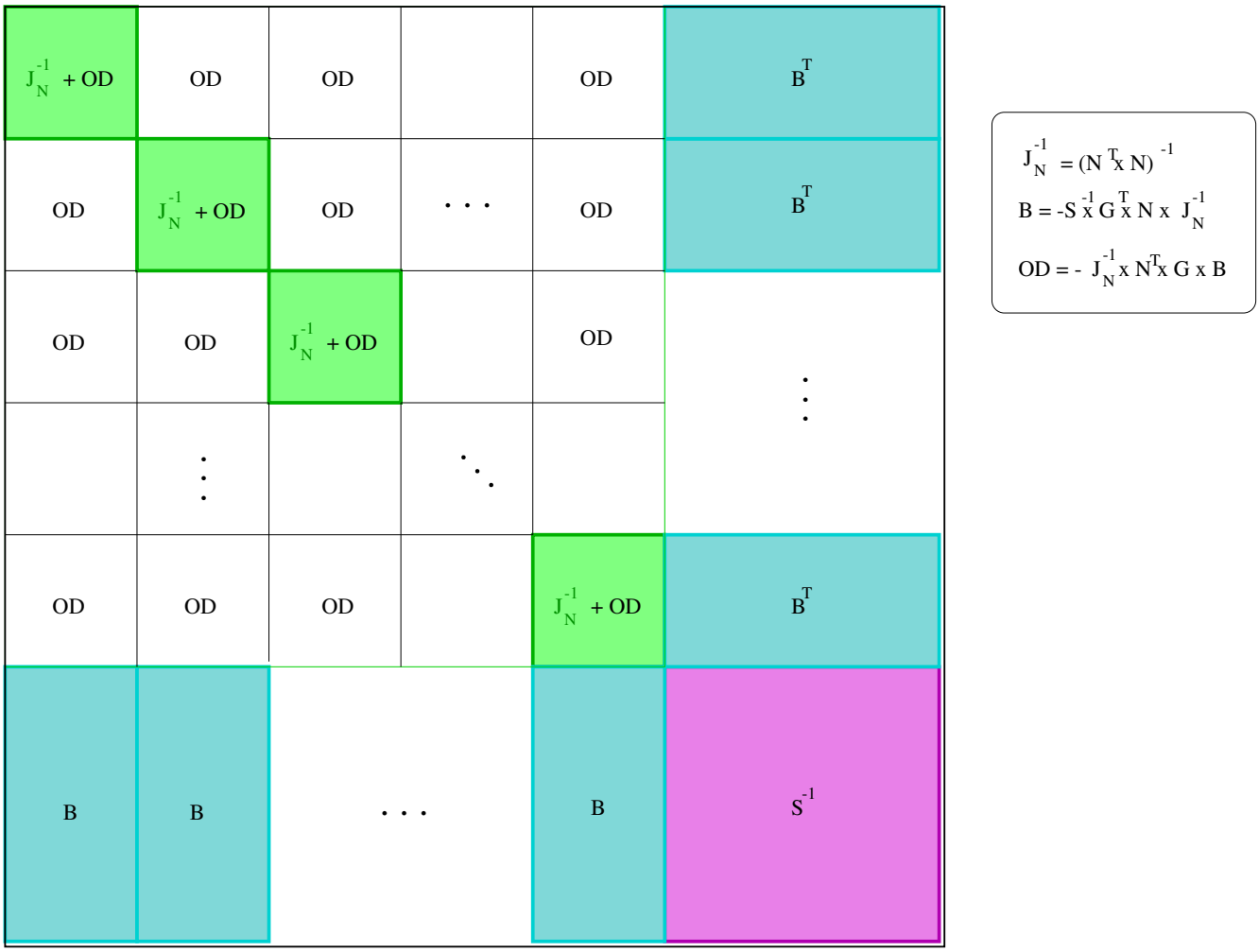


Figure 56: Inversion of ($J^T.J$) grand-system matrix from Fig. 55 using the per-block inversion displayed in Fig. 53. Expressions for the block matrices J_N^{-1}, OD (for Off-Diagonal), and B (Belt) are directly in the figure legend, the Schur complement S, whose inversion S^{-1} is used for calculation of the above mentioned blocks, is – specifically for our system – expressed in Eq. (3.3.19).

2.3.3.12 Finding the optimal calWeight

As it has been explained above, increasing *calWeight* parameter increases importance of the calibration equations (3.3.18) w.r.t. the equations for science data (3.3.16). We can study the effect by splitting the total sum of all squared residuals in Eq. (3.3.9) into two parts: The sum of squared residuals for equations for the science data χ^2_{sci} and the same for the calibration equations χ^2_{cal}. Because the sums grow with the equations count, we normalize them by the that number, i.e. χ^2_{sci} by *nTimeSamples* and χ^2_{cal} by *calWeight*. Now we can vary *calWeight* and study, how these normalized errors behave. The results are shown in Figs. 57 and 58. Fig. 57 presents the normalized sums χ^2_{sci} and χ^2_{cal} as a function of *calWeight* in a broad range of 1 – 10^4. Because of its extended displayed range, the *calWeight* parameter is expressed in decadic logarithmic scale. χ^2_{sci} and χ^2_{cal} are shown as a red and blue curves, respectively. It can be seen that at *calWeight* ~ 100 – 300 an error of the model-to-data fit for science

equations starts increasing rapidly, while we are not reaching further improvement in the fit of instrumental-variables to the dual-load ACD calibration measurements. More detailed view for antenna id 20 is in Fig. 58. We narrowed the range of displayed *calWeights* and show them now in the linear scale. The result is the same – *calWeight* slightly above 200 seems to be optimizing both science and calibration model-to-data fits together. This was demonstrated to be true (most likely) for all antennas – we have selected representatives from CM, DA, and DV groups with the same result. Apparently, the value most likely depends on the *nTimeSamples*, so for different MS it is advisable to re-run the optimization at least for one antenna.

As the last result in this section we present Figs. 59 and 60, which show, how much are our results for scan-averaged PWV and scan-averaged brightness temperatures dependent on the selected *calWeight*. We need not care about the varying parts, these have been shown to be very robust. Fig. 59 displays converged solution of iterations for Eq. (3.3.11) for $\langle PWV(t) \rangle$ as a function of *calWeight* in the range 100 – 1000. It can be seen that within the range 100 – 400, which we may consider as a broader optimum (based on Fig. 58), the resulting PWV varies in the range less than +/-0.01mm, which is acceptable – in terms of recalculated phase delay it represents less than 10 degs. Fig. 60, shows the same for $\langle T_B^{XX}(t) \rangle$, and $\langle T_B^{YY}(t) \rangle$. The displayed values are in kiloKelvins – they are used for technical reasons as internal unit (see the note at the end of previous paragraph). Again, variation within the displayed range is small, so the obtained results can be considered as quite robust and not dependent on the exact value of the *calWeight* parameter.

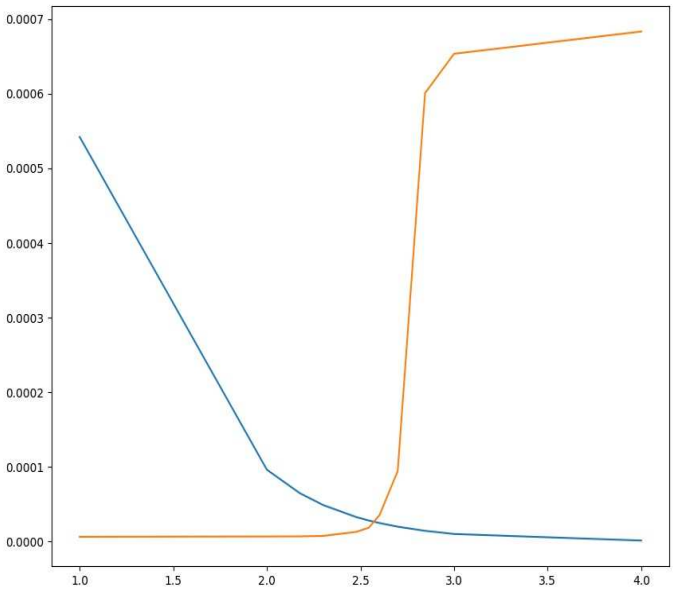


Figure 57: Sum of residuals separately for science-data equations subset scaled down by nTimeSamples (red) and ACD calibration equations sub-set scaled down by calWeight (blue) vs. decadic log of the calWeight parameter.

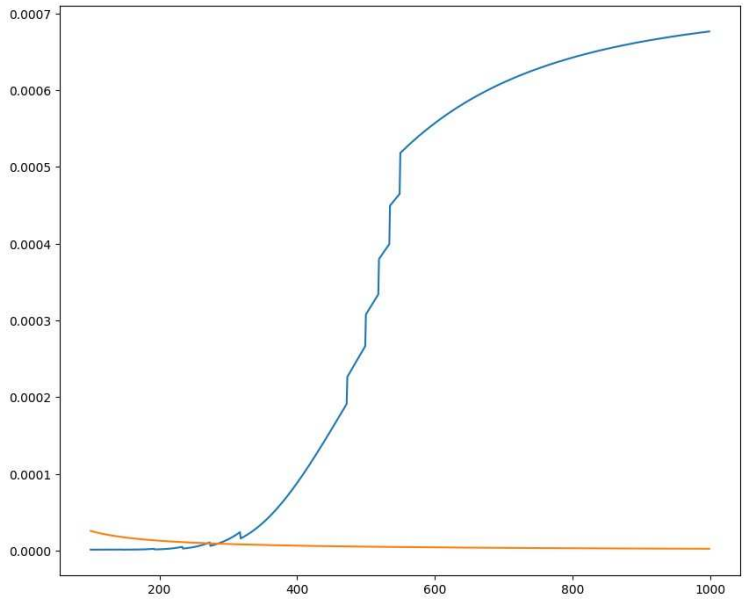


Figure 58: Essentially the same as in Fig. 57, now seen in more detail and in linear-scale for calWeight at the x-axis.

2.3.3.13 Implementation and results

The entire above described procedure, including all gradual improvements

- combination of XX/YY channels for *dw(t)* solution,
- non-linear solver for complete PWV(t) (and T^{B}_{Sun}(t) as a by-product),
- improved quiet-Sun spectrum,
- inclusion of antenna efficiencies,
- improved transfer of gain-attenuation factors from SQLDs to ACs, and
- making the ACD calibration equations to be a part of extended optimization task

is implemented in the routine *dWdBdG()*. We use the free parameter *calWeight*=200 in this routine, as found by the above described optimization. In order to save CPU time, we call the function *dWdBdG()* in a loop for all antennas in *saveSol_dWdB()* routine, saving the results as Python-native *.NPY* file, which is further re-used for analysis and applications.

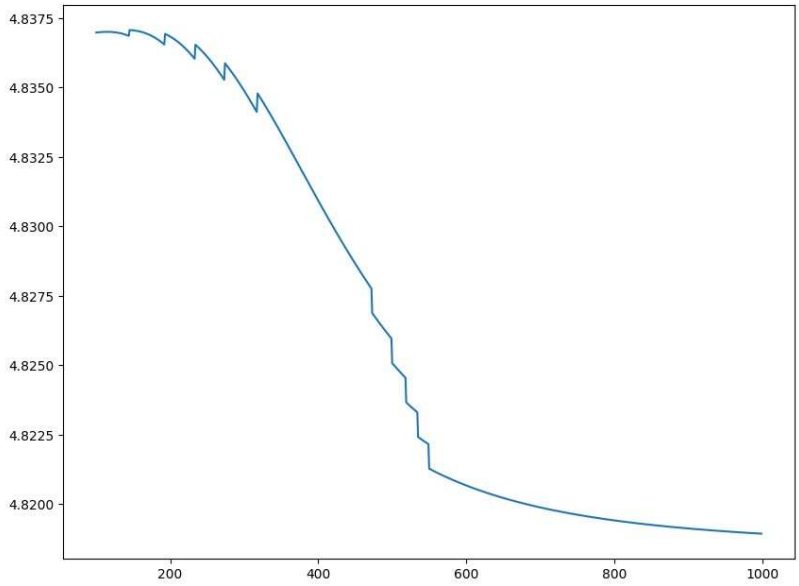


Figure 59: Scan-averaged value of PWV for antenna id20 vs. the calWeight parameter used for balancing weights of the ACD calibration equations with the equations for science data. The teeth visible on the curve are related to the convergence stopper (set to 0.001mm in our code) in the iterative solution of Eq. 3.3.11.

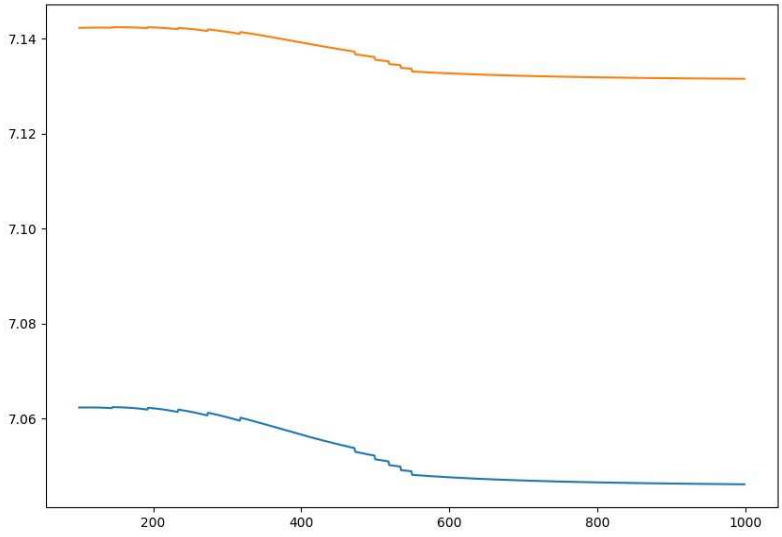


Figure 60: Like Fig. 59 above, now for scan-averaged brightness temperature of the Sun (red XX, blue YY polarizations), in kiloKelvins.

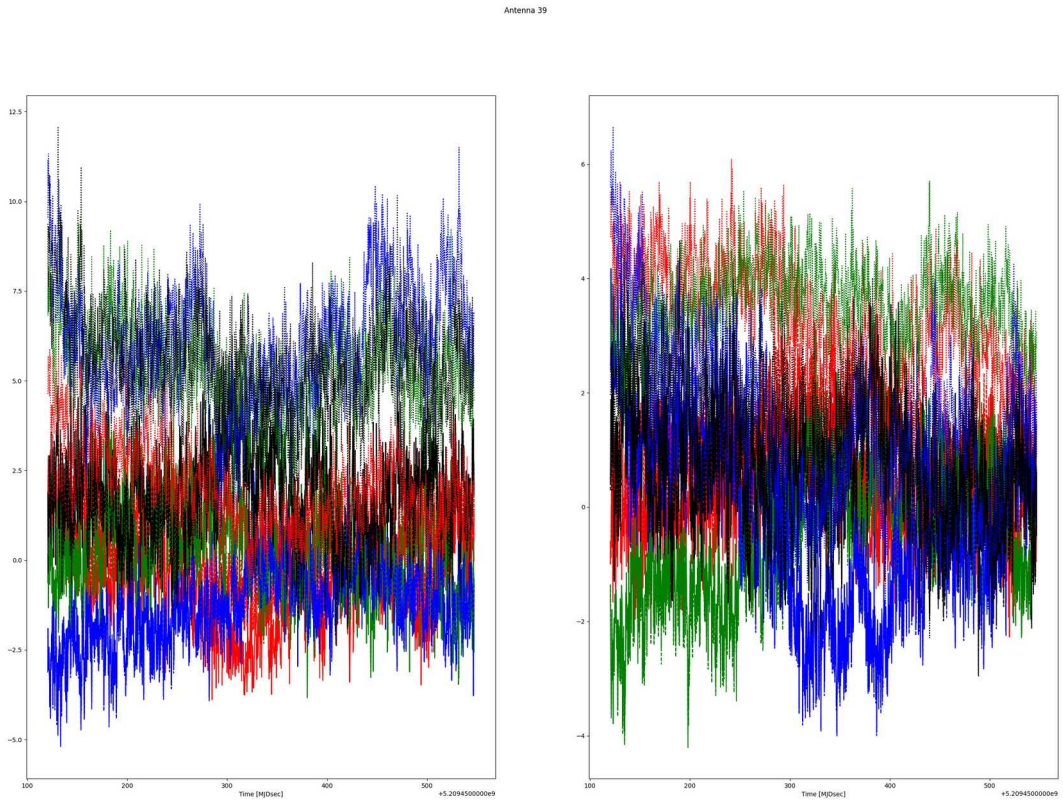


Figure 61: Least-square residuals for antenna id39 in our new procedure dWdBdG(), which allows for limited correction of gains and receiver temperatures in frame of the solution of the system. To be directly compared with the top row of panels in Fig. 43.

Results of this improved procedure are shown in Figs. 61 – 65. Fig. 61 shows the residual error in Kelvins for antenna 39 (DA56), so it can be directly compared with the top row in Fig. 43 Clearly, the problem of high (time-averaged) residuals is solved, we now have just fluctuations in order up to 10K. The small fix of gain/T_{RX} allowed for better fit of the scan-averaged brightness temperatures (sky+src) between model and observation. Figs. 98 – 64 show comparison between results of the older procedure *dWdB_Tbs()*, which uses constant pre-calculated gains and receiver temperatures (blue curves), and our new approach, implemented in *dWdBdG()*, which allows for limited (in order of low units of per-cent) corrections of the instrumental calibration (red curves). Fig. 98 presents comparison for the PWV solution, Figs. 63 and 64 the same for apparent solar brightness temperatures in XX and YY, respectively, all for antenna id38 (DV11). Clearly, the solar brightness temperatures are systematically higher for the gains that are allowed to be corrected as a part of the grand-system of equations (mimimizing the least-squares of residuals for the science + ACD calibration measurements together) and they are closer to the expected brightness temperature in Band 5, which is ~6990K (Alissandrakis et al., 2020). The constant-gain solution is, on the other hand, ~400 – 800 K below it. Again, the PWV variations proved to be really robust characteristics, while its absolute value is subject to some uncertainty. However, having better solution for the solar brightness temperatures provided by the

routine *dWdBdG()*, we trust its full-PWV solution, too. More figures for representatives of CM, DA, and DV antennas are available on-line.

As the last result of this comparative analysis we present Fig. 65. There, the scan- (over times/ integrations) and polarization-averaged ((XX+YY)/2) values of the solar brightness temperatures are shown for each antenna; the x-axis is discrete and represents the antenna IDs. Again, the red curve represents the correctable-gain solution, the blue one is for constant ACD-based instrumental parameters. Ideally we would expect a constant line around theoretical value of 6990 K. In fact, there are visible the following features:

- Some excesses – namely CM06. It should have been actually flagged (and it was flagged in the standard calibration procedure) but in the PWV analysis we retain all antennas.
- The blue curve is systematically low and – if the excesses in both curves are removed – it has a bit larger RMS.

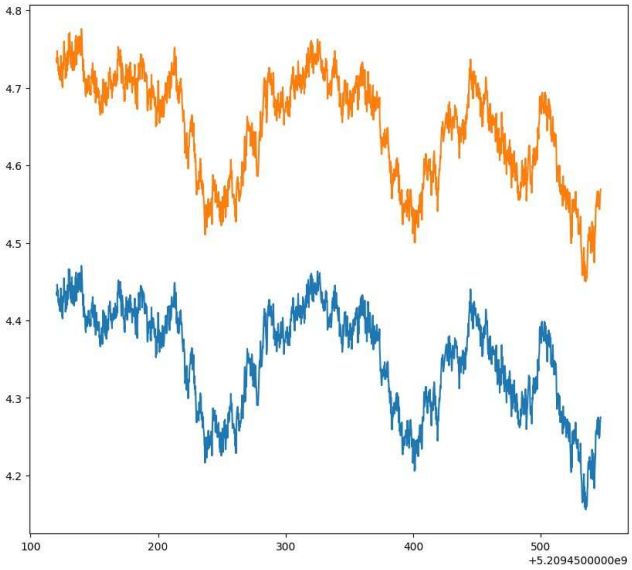


Figure 62: PWV (mm) as a function of time (MJDSec) for antenna DV11. Blue curve: Calculation with constant gains calculated by the ACD calibration equations (dWdB_Tbs() routine), red curve: Result of dWdBdG() routine, which allows for a limited correction of gains as a part of the grand-system solution.

It seems that the groups of antennas can be identified: CM with IDs 0 – 8, DA (9 – 27) and DV (28 – 51), with the CM antennas exhibiting (in average) lowest T^B, DA a bit higher, and the DV the highest. This is most likely connected with the rear- and forward-spillover efficiencies. We have taken 'universal' values of η_l = 0.95 and η_F =0.90 in line with Table 3 in White et al. (2017) for all antennas, but apparently, the DV antennas have a bit higher efficiencies than this values, while the 7m CM antennas a bit lower. We shall come back to this finding in coming Section 2.5.3, where we make use of measurement from an additional test dataset obtained in July 2026. Merely two datasets are not a big

ensemble, but we shall see that, indeed, there are some systematic trends in the scan-averaged solar T_B by antenna type. We have used the result to improved estimation of antenna efficiencies, which are, in turn, used in our improved calculations.

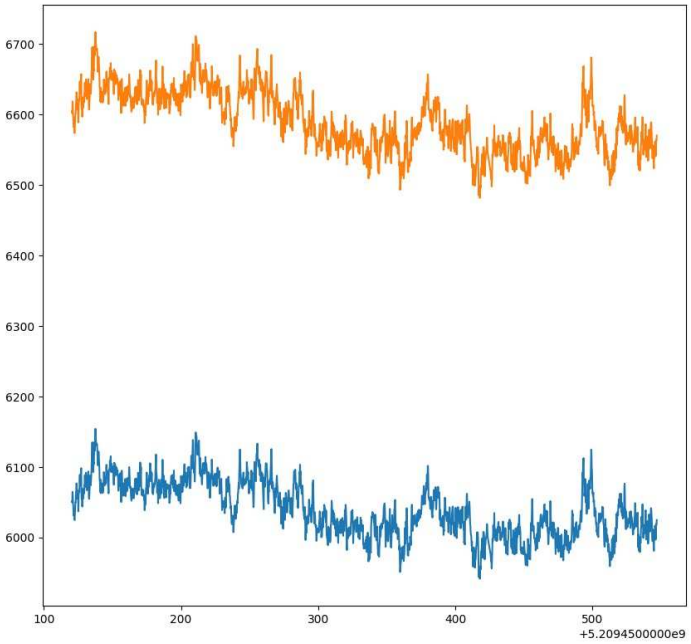


Figure 63: Like in Fig. 62, now we compare solutions for solar (apparent, incl. scintillation) brightness temperature in polarization XX (Kelvins).

Again, the average temperature closer to the expected theoretical value and slightly lower RMS speaks in favor of the method *dWdBdG()* with the flexible gains being part of the solution, albeit the result is not super-convincing.

Results: We are able – in addition to finding solution for PWV and the T^B time series – to make also small corrections (1 – 2 %) to the instrumental parameters G^{SP} and $P_{inst}{}^{SP}$, letting them to become variables of the system (3.3.10). At the same moment we prevent them from large departure from their nominal values found by the ACD calibration solution. This anchor is achieved by inclusion of the ACD calibration equations (3.3.18) into the system with appropriately selected weight. Thanks to clever solution for the system matrix inversion the procedure is, moreover, not so much time-demanding (the most time is spent by reading the received powers and ACD data from the MS, the iterative solution of extended system (3.3.11) takes just about 10% of total time).

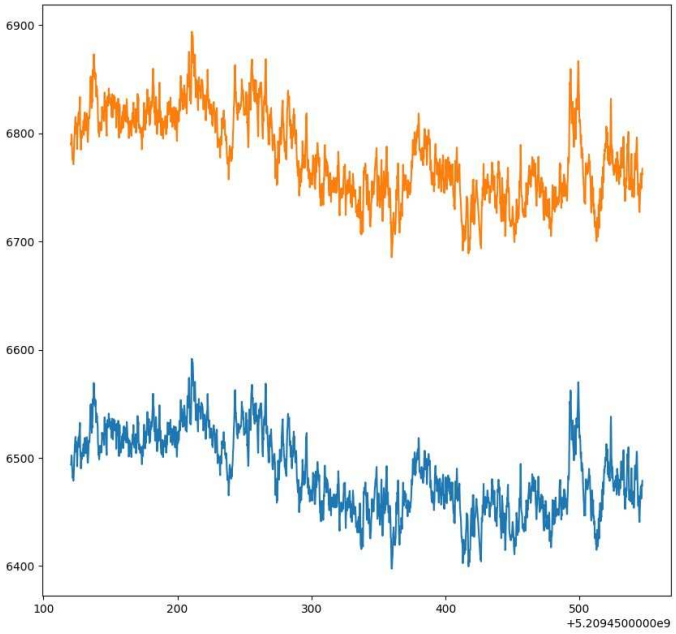


Figure 64: Like Fig. 63, now for YY polarization.

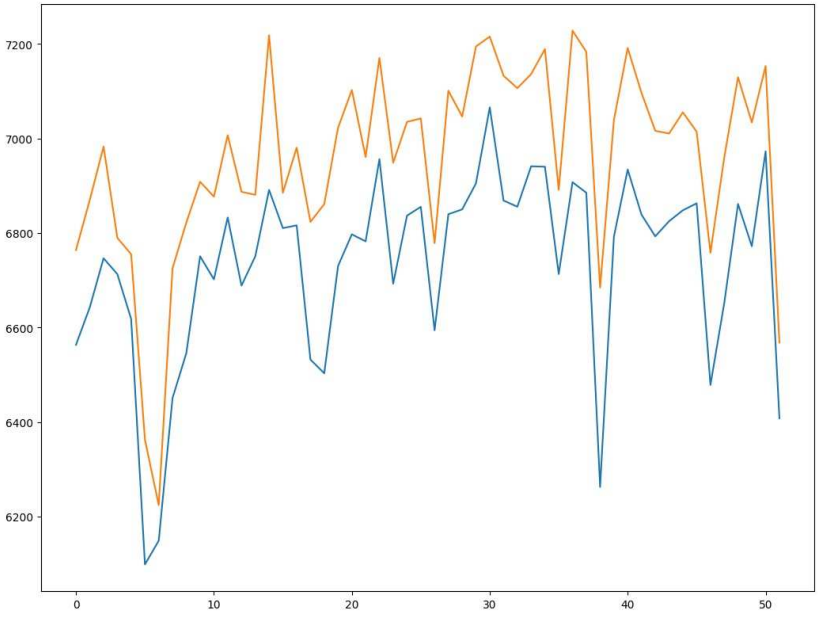


Figure 65: Scan- and polarization-averaged solar brightness temperatures for each antenna (sorted by antenna IDs). Red curve: Flexible-gain solution using dWdBdG(), blue: Constant-gain (from the ACD calibration solely) solution.

Summary of this part

By gradual improvements we have developed mathematically rigorous and plausible – as can be indicated by realistic values of scan-avareged solar brightness temperature (close to expected theoretical value) and PWVs (very close to SYSCAL table averages) – method for calculation of PWV(t) time series from the science SPWs of solar scans. Thanks to different spectral dependence of absorption in the terrestrial atmosphere and solar (apparent or actual) brightness changes we are able to decompose both effects and provide 'the net' PWV(t) time series. The procedure has been implemented to a collection of Python/CASA routines which, in principle, may be integrated to the general solar ALMA library *sun_reductionUtils.py*. The goal is to use this PWV(t) time profiles as a replacement for direct WVR signal, which is (almost always) unreachable during the solar on-source sub-scans.

2.4 Science SPW-based PWV(t) vs. direct WVR measurements

The question arises, to what extent our PWV(t) time series, calculated by approach developed gradually in previous Section 2.3, indeed correspond to the PWV as measured more directly by means of WVRs.

For the band-pass (or phase, which, however, has quite short scans) calibrator we have been easily able to test the validity of our *Approach 2* based on our differential dPWV(t) proxy to the WV emission/absorption and its subsequent impact to the phase delays. That test case is crystalline clear as we have the 'ground truth' – the exact phase solutions for the sources. Moreover, we may (and we did) compare directly the dPWV(t) with WVR data extracted (and transformed to differential phase delays) by means of *wvrgcal()*.

In the solar case it is more difficult because the only knowledge about the phase delays of the Sun comes from our *Approach 1*, where we infer them by self-calibration on different time-scales separating the troposhperic turbulence from intrinsic solar dynamics. In fact, the case of BP calibrator is the same, but the BP calibrator has a trivial source model, what is very far from truth with the Sun, which has a complex structure that even changes on the scales only about 10x longer then available cadence of observation. Using solar phase from self-calibration is thus possible, but we actually mix two our approaches (1 and 2) together and it is hard to disentangle their possible errors in the result. Hence, verification of PWV(t) calculations from science SPWs through their impact on phase delays is a bit ambiguous.

The direct comparison with WVR data seemed hard or impossible. In order to make it – at least approximately – we have devised a specific observing test: The 12m array is split into two sub-arrays – most of the antennas watching the Sun, but a few of them (4 - 5), specifically these being part of a close couple with another antenna, look a bit aside off the Sun and measure the (unsaturated) WVR signal synchronously with the solar observation done by the main sub-array. The risky point here is that we observe (i) two a bit different sources – the main array the target at the solar disc, the little sub-array the 'empty-sky' source 2 degs appart, and, (ii), the 'WVR' antenna, albeit selected very close to its 'solar' sibling, is not exactly co-located, so they look through a bit different mass of the atmosphere.

Nevertheless – by an incredibly good luck – we have found yet another possibility of direct comparison of our PWV(t) series with the direct WVR measurements: The 2023 test data has been taken very close to the solar limb. Because the WVR FoV is a bit separated from the Band 5 receiver's FoV, by chance the WVR were pointing out of the Sun, however very close. The WVR data were thus somehow contaminated by the scattered solar radiation but not saturated. In the following we explore both found possibilities for direct WVR-based vs. sci SPW-based PWV(t) time series. We start (Section 2.4.1) with the contaminated WVR data of 2023 observation, on which we have developed the generalized method on how to calculate the PWV from the WVR channels even under solar contamination. Then, in Section 2.4.2 we present preparation, performance, and results of the specific sub-array test observations designed exactly to the PWV(t) verification.

2.4.1 WVR data in the solar scan of 2022.1.00031.CSV – detailed analysis and comparison with the dPWV proxy.

As it has been already mentioned above, by chance we have found that the WVR data for 2022.1.00031.CSV Band 5 MS are, surprisingly, not saturated, albeit influence of the direct solar illumination is visible. As discussed with Neil Phillips (Development-study team communication), the reason is in luckily pointing to the target very close to the solar limb, while the WVR FoV is separated from the Band 5 receiver's FoV by ~10arcmin, so the WVR 'missed' the solar disc.

The WVR raw data can be extracted directly from the SPW 4 and they represent four time series for four frequency channels of the WVR (see the MTR and references therein). The valuse extracted frm the WVR channels are already calibrated brightness temperatures of the sky in Kelvins. We present here the WVR brightness temperatures as measured in its 4 channels for the solar scan no. 9 as Fig. 66 . The jumps at times ~120s (+ MJD offset, up) and ~550s (down) are caused by a nature of the solar scan: It has the very first and very last sub-scan pointing off-source (2degs as verified by our routine *sol_offPos()* in sol_wvrgcal.py), while the middle sub-scan(s) points to the target. The reason is in having an additional "atmosphere" (sub)scan but now with the attenuation levels set as for the on-source observation; this is used in calculation of the modified gain (higher-attenuation) gain G^* as explained in Section 2.3.

As one can see, the WVR channel 0 is saturated by the atmospheric WV emission, it is even indifferent to re-pointing of the antenna close to the Sun – there is no jump visible. The remaining channels 1 – 3, however, detect the approaching to the Sun and react by the jumps. Clearly, the signal is contaminated by the solar illumination. Inspired by the success in decomposition of the sky brightness in the science SPWs described in Section 2.3, we attempt here to use similar trick and get rid of the solar-illumination contribution.

The entire process of extraction of the 'net' or 'decontaminated' (from solar illumination) PWV signal was, again, a sort of gradual procedure, with surprising findings, which demanded change of the approach. We present here the complete evolution, for that sake it is split into two parts, corresponding to history of how we were addressing the problem. Basically, our approach has evolved from linear equations for differential quantities (fluctuations of water wapor dw and brightness of parasitic illumination dL) towards non-linear radiative-transfer equations for complete values of PWV(t) and

illumination *L*, very similarly to the development documented for PWV-proxy in the previous Section 2.3. And again, the impatient reader may skip to the sub-section 2.4.1.7, where the final procedure, its implementation, and results are presented.

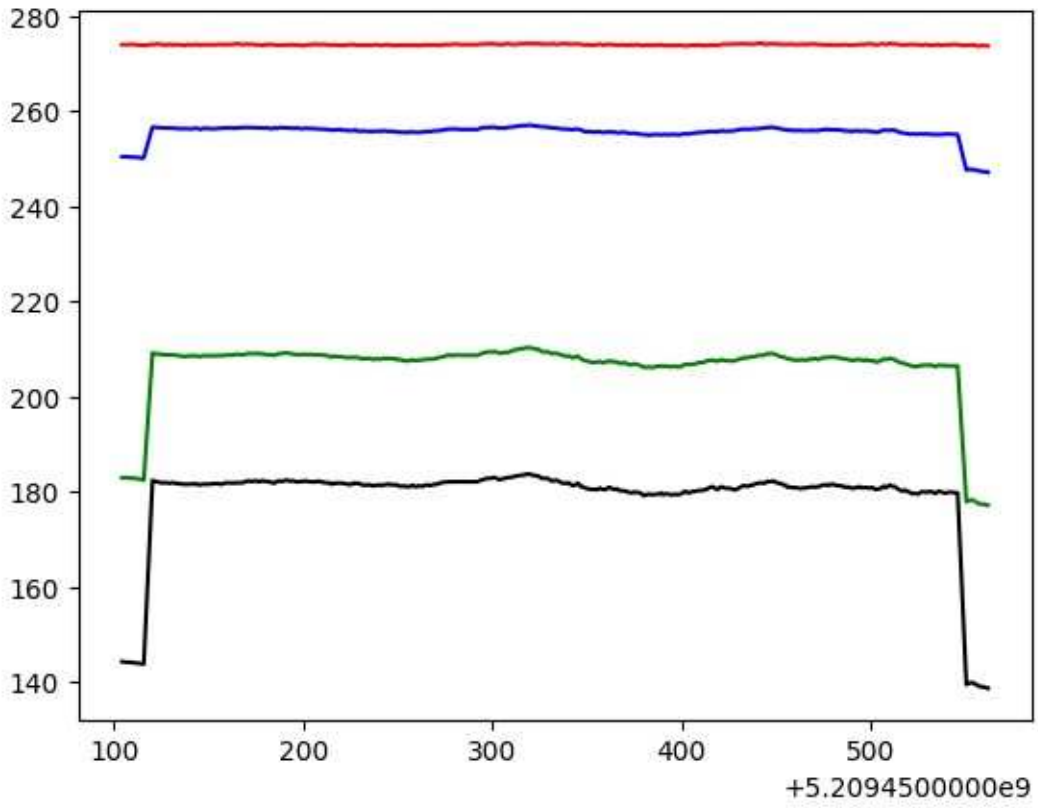


Figure 66: Raw WVR brightness temperatures (in K) as detected in 4 channels in the SPW 4. Channels 0 - 3 are displayed by red, blue, green, and black curves. The jumps at the beginning and end of the solar scan, visible in channels 1 – 3, are caused by moving the antenna towards and outwards of the Sun. Time in the x-axis is in MJDsec.

2.4.1.1 Theory 1: Differential approach

From the jumping WVR intensity, when pointed closer to the Sun, it is clear that the WVR temperatures are contaminated by the solar illumination. Hence, the brightness temperature recorded by the WVR channel contains two components – terrestrial atmosphere irradiation and partial solar illumination, weakened by the absorption

$$T_{WVR}(f,t)=T_{atm}(f)\cdot[1-tr(f,t)]+L_{sol}\cdot tr(f,t) \qquad (4.1.1)$$

Here, as experimentally found, we have to allow the effective kinetic temperature of the atmosphere to be dependent on frequency. For the science SPWs treated in Section 2.3 it was found that T_{atm} calculated

by the ATM model varies by ~1K within entire SPW (e.g., SPW 5), but for the WVR's SPW 4, the channel dependence is much stronger. It is obvious – the closer we are to the WV line center, the bigger is the optical depth, so in the line center we are sounding close (i.e., low) atmospheric layers, while in the far wings the radiation comes from the higher altitudes, too. For now, we may think about solar illumination component $L_{sol} = r.T_{sol}$, where r is a fraction of the solar radiation scattered to the WVR FoV and T_{sol} is the solar disc brightness temperature in Band 5 (~ 6700 K), as being constant, because the solar brightness temperature only weakly depends on frequency (negligible within the relatively narrow bandwidth of the SPW 4) and, one may think that the relative off-pointing of the WVR device from the Sun is kept constant, so there is no reason for the fraction r to be a function of time. However, as we will see in further, this assumption is not true and below, in Part 2, we will have to handle it. But for now we can continue with assuming the illumination L_{sol} being constant.

We can now express the transmissivity from Eq. (4.1.1) as $tr(f,t)=exp[-\tau(f,t)]=exp[-\tau_d(f) -\tau_1(f).w(t)]$ and after decomposing the time-dependent quantities to the time-averaged (over a scan) mean value and the differential variations as we have done in our early dPWV-proxy approach, too, we get an expression for the WVR temperature

$$T_{wvr}(f,t)=T_{atm}(f)+e^{-\langle\tau_d(f)\rangle}\cdot(L_{sol}-T_{atm}(f))\cdot e^{-\tau_1(f)\cdot(\langle w\rangle+dw(t))}$$

which, in a limit *of* $dw(t) << 1$, becomes

$$T_{wvr}(f,t)=T_{atm}(f)+e^{-\langle\tau_d(f)\rangle}\cdot(L_{sol}-T_{atm}(f))\cdot e^{-\tau_1(f)\cdot\langle w\rangle}\cdot\left[1-\tau_1(f)\cdot dw(t)\right]$$

For the variation $dw(t)$ set to zero this expression gives the time-averaged value of the WVR temperature $<T_{wvr}(f)>$, so we eventually arrive to the expressions

$$T_{WVR}(f,t)-\langle T_{WVR}(f)\rangle=-\tau_1(f)\cdot\left(\langle T_{WVR}(f)\rangle-T_{atm}(f)\right)\cdot dw(t) \qquad (4.1.2)$$

As it can be seen, we have effectively excluded the explicit solar illumination L_{sol}, however, it is still implicitly contained in formula (4.1.2) by means of the averaged WVR temperature $<T_{wvr}(f)>$.

The LHS of Eq. (4.1.2) can be easily directly calculated from each WVR channel, as well as most of the quantities at the RHS. An exception is the $T_{atm}(f)$ – if we use the ATM model for its estimation with the mean PWV for the MS (4.6mm), we get different values of $dw(t)$ from different WVR channels 1 – 3. The channel 0 is basically useless because it is intact to anything – both PWV changes or increase/decrease of solar illumination due to antenna off-source/on-source re-pointing do not make any change, it (most likely) just follows the actual air temperature in the low atmosphere around the dish.

Anyway, we can express Eq. (4.1.2) for two different WVR channels and make their ratio. This excludes $dw(t)$ from the expression and predicts that the variations of the $T_{wvr}(f,t)$, i.e. after subtraction of its mean $<T_{wvr}(f)>$, as expressed in the LHS of Eq. (4.1.2), are mutually linearly correlated for different WVR channels. Indeed, the scatter plots of $\delta T_{wvr}(f,t)=T_{wvr}(f,t)-<T_{wvr}(f)>$ for two different frequencies show pretty linear correlation, at least for the DV antennas. We, however, had to exclude

the channel 0 from this analysis, because it has almost zero temperature variations, so its sensitivity to PWV variation is very small

Albeit we are not capable to enumerate the factor standing in front of *dw(t)* on the RHS of Eq. (4.1.2), still we can use Eq. (4.1.2) for the qualitative display of the *dw(t)* curve, just without its absolute scaling. This is useful, as long as we would like to compare it with the indirect/proxy method for (also differential) *dw(t)* based on the data from the science SPWs (see sections 2.2 and 2.3).

2.4.1.2 Results 1: Differential approach

As written above, we may do qualitative comparison between the PWV variations *dw(t)* as derived directly from the WVR channel data using Eq. (4.1.2) above, with the dPWV-proxy calculated from the science SPWs. Because, lacking knowledge of $T_{atm}(f)$ in Eq. (4.1.2), we do not have absolute scaling for the WVR-based *dw(t)*, we scale its values arbitrarily, such a way that the value-ranges for both quantities (WVR-based *dw(t)* and dPWV(t) proxy) are the same. We did it for all antennas equipped with WVR, i.e. except of the CM antennas. The result examples are shown in Figs. 8 and 9, the full ensemble is accessible on-line as *Annex 5*.

Figs. 67 shows the *dw(t)* data calculated from Eq. (4.1.2) in arbitrary scaling as a blue curve. It is the same in both left and right panels, which are displaying XX and YY pols, what, however, has a meaning only for the dPWV proxy calculated from the science SPWs (WVRs have no polarization distinction). The red curves are the corresponding dPWV proxy calculated by a simplified procedure using a single SPW signal, according to Eq. (2.2.14). The figure is produced from the data of antenna DV08. The y-axis is in mm, the time at x-axis is in MJDsec. Apparently, the (qualitative) correlation is quite good even though the PWV is changing ‘wildly’.
Fig. 68 displays the same result, but now for antenna DA42. In this case, the correlation is much worse, one can barely find any similar features between the red (dPWV-proxy) and blue (WVR-based) curves. If one sees the full ensemble of results, there is striking difference between the DA and DV antennas: While the DV antennas show, in general, very good amount of correlation, the DA antennas are much weaker – Figs. 67 and 68 here are just illustrative examples, but they express the general trend.

Since only apparent difference in construction between DA and DV antennas lies in the number and positions of the supports that hold the sub-reflector, we are thinking about additional scattering (or shadowing) of the sunlight at these constructions, which, as the Sun moves on the sky, may be varying with time as the antennas have alt-azimutal mounts. Likelihood of this explanation has been confirmed during informal discussion with our advisory team on occasion of the test-observation preparation in January 2026. The main consequence is that the assumption of constant contribution by the solar illumination to the WVR temperatures is no longer valid and we had to find another approach.

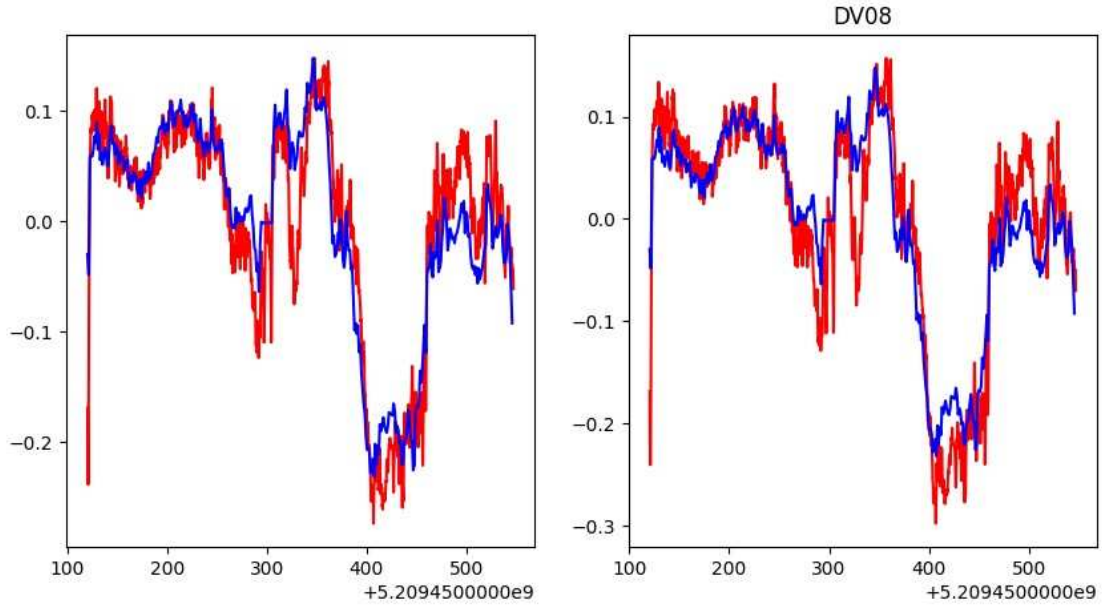


Figure 67: WVR-based PWV variations around its mean (blue, arbitrary scaling) vs. the dPWV proxy calculated from absorption in science SPWs (red, in mm) for antenna DV08. Left/right: dPWV proxy for XX/YY pol.

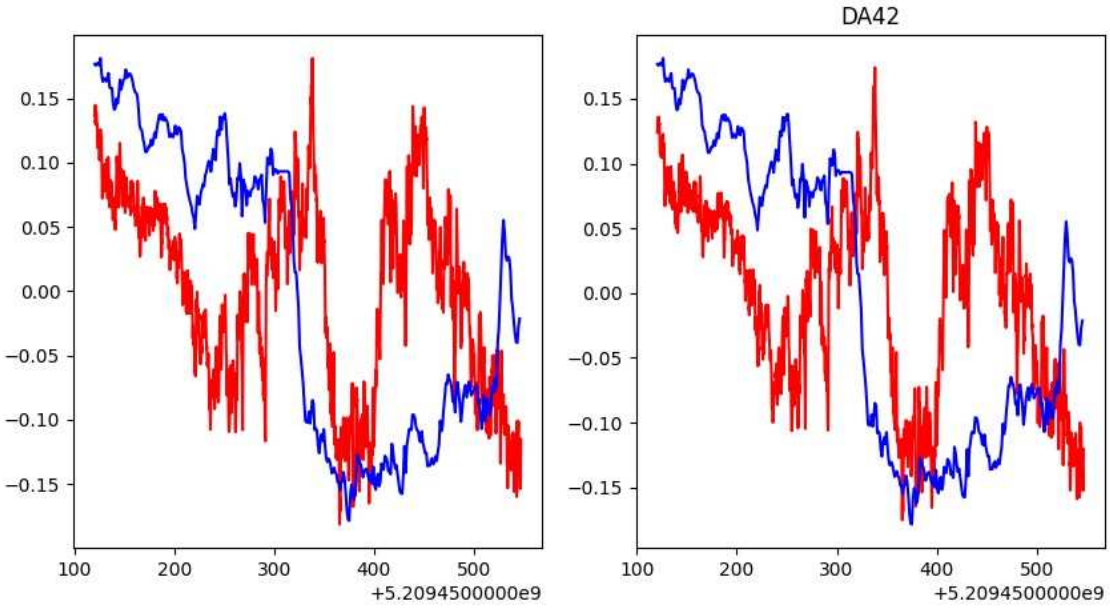


Figure 68: The same as in Fig. 67, now for antenna DA42.

2.4.1.3 Theory 2: Non-linear RT equations for full PWV(t) and L(t)

With having another free/uncontrolled variable in the system it looked like hopeless to try extract the net PWV contribution from the solar-illumination-contaminated raw WVR data. Anyway, our success with decomposition of the (apparent) source brightness dynamics from the science-SPW signal as presented in Section 2.3, inspired us to attempt here the similar task.

First, we have to allow the total solar illumination L_{sun} to be now a function of time, because the fraction r of the solar radiation, which enters the WVR aperture, may vary with time due to uncontrolled reflections/scattering at the antenna construction. Second, the concept of effective *kinetic* temperature of the atmosphere ($T_{atm}(f)$ in Eq. (4.1.2)) has been found as problematic, because we have no means how to evaluate it. For that reasons, the basic Eq. (4.1.1) above is rewritten as

$$T_{WVR}(f,t) = T_{BS}[f, w(t)] + L_{sol}(t) \cdot tr[f, w(t)] \qquad (4.1.3)$$

Here, the $T_{BS}[f,w(t)]$ is the sky brightness temperature, which can be approximated by the ATM model via *at.getTebbSky(wh2o=w)* method, parametricaly, for user-defined PWV. In Eq. (4.1.3) we explicitly indicate that both the $T_{BS}()$ and transmissivity *tr()* depend as functions on time not directly, but by means of time-varying PWV, the *w(t)*.

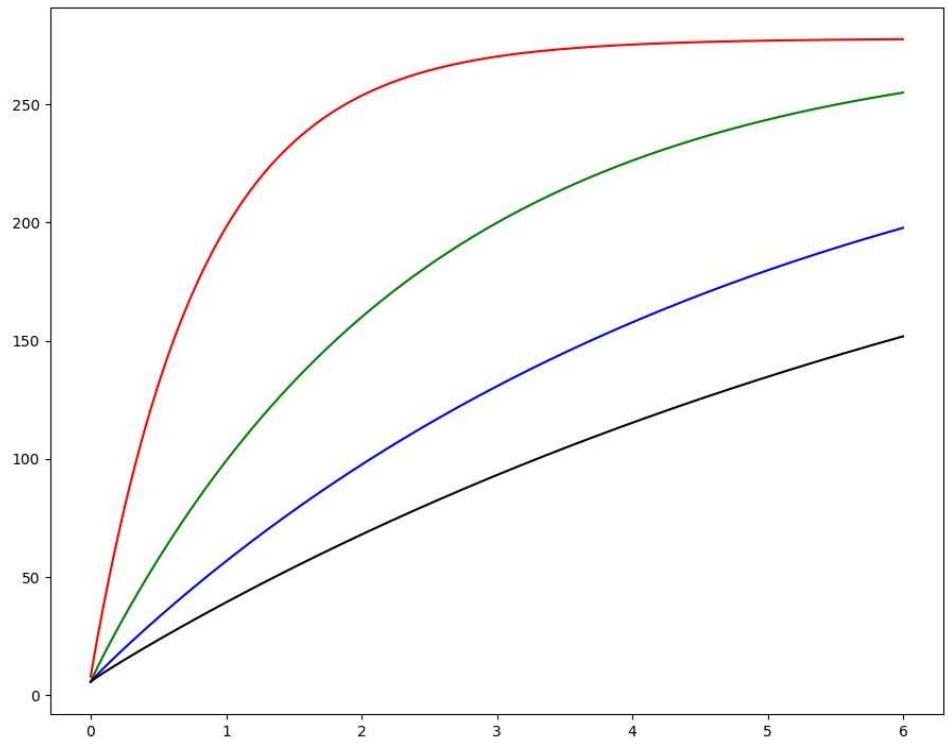


Figure 69: Sky brightness temperature (y-axis, in K) for varying PWV (x-axis, in mm) calculated via ATM model accessed by at.getTebbSky(). The red, green, blue, and black curves are for frequencies corresponding to the WVR channels 0,1,2, and 3, respectively.

We have calculated the sky brightness temperature T_{BS}, approximated by *at.getTebbSkySpec()* as a function of PWV for the 4 WVR channels. The ATM model was matched to the real atmospheric conditions by meteo parameters obtained from ALMA Meteo Central station, the airmass has been set by the elevation of the Sun – see *setAtmTool()* in sol_wvrgcal.py (*Annex 4*). The result is shown in Fig. 69. If there was no solar contamination, the second term in Eq. (4.1.3) would disappear and we could match the measured WVR temperatures in SPW 4 to the curves in Fig. 69 to fit as much as possible the values at the curves and get this way the actual PWV. The curves look like close to exponentials approaching some limiting value for PWV going to infinity (found to be 277.5 K for scan no. 9), what can be easily shown when instead of T_{BS} as a function of w we plot the sky brightness temepratures in another representation: $ln(T_{BS}(\infty)–T_{BS}(w))$ as a function of w. This is shown in Fig. 70, and indeed, except of the channel 0, which we had to drop anyway (because of saturation), the curves are linear within a wide range of PWVs.

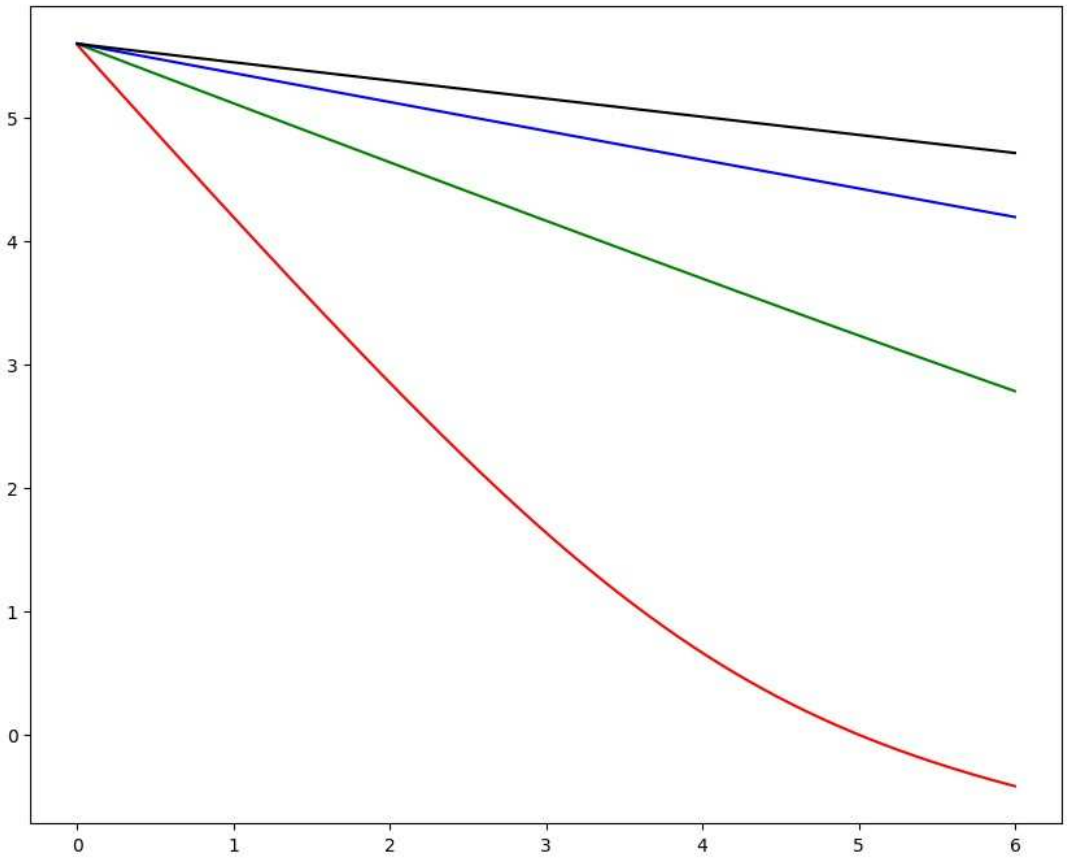


Figure 70: Like in Fig. 69, but now, instead of brightness temperature, we present its logarithm after subtraction from the limiting value of 277.5 K which is reached for infinity PWV (see text).

The logarithmic representation facilitates coupling of the actual T_{wvr} temperatures in channels 1 – 3 to the $T_{BS}(w)$ calculated from the ATM model, because of (almost perfect) linearity. The linearity assures that if we know the $T_{BS}(w_o)$ at some 'working/operating point' $PWV=w_o$, we can easily calculate it in another point $PWV=w$ via relation

$$\ln\left(T_{BS}(\infty)-T_{BS}(w)\right)-\ln\left(T_{BS}(\infty)-T_{BS}(w_o)\right)=-\tau_1^{eff}\cdot(w-w_o) \qquad (4.1.4)$$

where we designated the slope of the straight lines in Fig. 11 as $-\tau_1^{\mathrm{eff}}$ because, as it will come clear from further, it has a meaning of effective wet opacity per 1 mm of PWV, scaled by air-mass for the current scan. If the relations displayed in Fig. 70 were perfectly linear, τ_1^{eff} would be constant. Because there is apparently some non-linearity caused by dependence of the optical depth on w (so that with changing w we see a bit different layer of the atmosphere with different temperature), τ_1^{eff} is actually very weakly dependent on w, too. However this dependence becomes prominent only in the channel 0 for PWVs higher then ~3 mm (see a knee of the red curve in Fig. 70). Anyway, we are searching the coupling between measured T_{wvr} temperatures and brightness sky temperatures T_{BS} calculated from the ATM model iteratively, via Newton-Raphson (NR) method (see *dWdLs()* in sol_wvrgcal.py – *Annex 4*) and calculate the slope numerically in the vicinity of the current 'operating point', so its weak dependence on w does not play a role. Moreover, the almost-linear logarithmic representation facilitates convergence of NR iterations a lot.

Eq. (4.1.4) can be rewritten into form

$$\frac{T_{BS}(w)-T_{BS}(\infty)}{T_{BS}(w_o)-T_{BS}(\infty)}=\mathrm{e}^{-\tau_1^{eff}\cdot(w-w_o)}$$

from which $T_{BS}(w)$ can be expressed as

$$T_{BS}(w)=-\left[\mathrm{e}^{\tau_1^{eff}\cdot w_o}\left(T_{BS}(\infty)-T_{BS}(w_o)\right)\right]\cdot\mathrm{e}^{-\tau_1^{eff}\cdot w}+T_{BS}(\infty) \qquad (4.1.5)$$

Formally it complies with the expression

$$T_{BS}(w)=T_{atm}\cdot(1-tr(w))=T_{atm}\cdot(1-e^{-\tau(w)})=-[T_{atm}e^{\tau_d}]\cdot e^{-\tau_1\cdot w}+T_{atm} \qquad (4.1.6)$$

so it is nothing to wonder about and also it explains, why we call the slope of straight lines in Fig. 70 as $-\tau_1^{\mathrm{eff}}$. On the other hand, since the concept of T_{atm} does not work well (when matching Eqs. (4.1.5) to (4.1.6), it appears that the T_{atm} in the coefficient in front of the exponential need not to have the same value as the second T_{atm} in the last, additive term in Eq. (4.1.6); moreover the effective wet opacity τ_1^{eff} is just close but not exactly the same as τ_1 scaled by airmass), we better stay with the 'experimentally' found (i.e. via analyzing the exact ATM solution for *TebbSky*) parameters in the Eq. (4.1.5).

The $T_{BS}(w)$ from Eq. (4.1.5) can be inserted, together with the transmissivity definition, into Eq. (4.1.3). If we assume that we are moving in the linear PWV space close around the 'operating point' w_o, which can be conveniently selected as the PWV mean averaged in time over a scan (i.e. $w_o=\langle w\rangle$), then $dw=w-w_o$ is going to be small, the appropriate exponential factor may be linearised and we eventually receive relation

$$T_{wvr}(f,t) = T_{BS}(f,w_o) + \langle L_{sun}\rangle \cdot tr(f,w_o) + \left[\tau_1^{eff}(f)\cdot(T_{BS}(\infty) - T_{BS}(w_o)) - \langle L_{sun}\rangle \cdot tr(f,w_o)\cdot\tau_1(f)\right]\cdot dw(t) + tr(f,w_o)\cdot dL_{sun}(t) \qquad (4.1.7)$$

Here we again decompose the solar illumination into stationary part $<L_{sun}>$ and its variation $dL_{sun}(t)$ and neglected the mixed term of the second order $dw(t).dL_{sun}(t)$. The frequency dependence indicates that Eq. (4.1.7) can be formed for each of the WVR channels extra. Actually, for the aforementioned reasons, we omit the saturated channel 0 and use just channels 1 – 3. Anyway, even with the three channels we are able to form over-determined system of three linear equations for two unknown variables $dw(t)$ and $dL_{sun}(t)$.

The Eq. (4.1.7), respectively the set of its realizations for three independent frequencies of WVR channels 1-3, represents a cornerstone for decomposition of the measured WVR signal T_{wvr} in SPW 4 into PWV variations and variable solar illumination. It is somewhat similar to the Eqs. (3.1.4) and (3.1.5) for decomposition of T_{sky} in science SPWs into PWV-induced absorption-variation and the source brightness dynamics, by which it is inspired. On the other hand, there are two distinctions: We have a stationary part in the 1st line of the RHS in Eq. (4.1.7) and this stationary part itself is unknown – we do not know the scan-averaged PWV value w_o and neither the mean solar illumination $<L_{sun}>$. For that reason, even the matrix coefficients in the 2nd and 3rd line of the RHS of Eq. (4.1.7) are not known.

In order to solve the system (4.1.7), we proceed by means of a two-step algorithm, which is based on the following ideas:

1. The unknown variations $dw(t)$ and $dL_{sun}(t)$ are, by design, a small fluctuations around the mean value of the respective quantity. If they have (in the found solution) a non-zero mean, it can be added/transferred to $w_o = <w>$ and $<L_{sun}>$, respectively.

2. In the corrected system, with the variations $dw(t)$ and $dL_{sun}(t)$ having already zero mean (as per Item 1 above), the stationary part at the 1st row of Eq. (4.1.7) must correspond to the time-averaged mean of the T_{wvr}, i.e. $\langle T_{wvr}(f)\rangle = T_{BS}(f,w_o) + \langle L_{sun}\rangle \cdot tr(f,w_o)$.

3. From the equality for stationary parts in the above Item 2 we can calculate $<L_{sun}>$ for any value of w_o using the ATM model (*atmosphere* tool in CASA) – T_{BS} is calculated as *at.getTebbSkySpec(wh2o=*w_o*)*, the transmissivity via *at* methods for wet and dry opacity. The averaged $<T_{wvr}>$ we simply calculate from SPW 4 data, for channels 1 – 3 extra.

4. The point is, that for any input value of w_o we get three values of $<L_{sun}>$ – each WVR channel (1 -3) gives us one solution. In fact, we aim at a single value, so we evaluate the STDDEV of those three independent solutions and aim at minimizing the dispersion in iterations. This gives us a good estimate of $w_o = <w>$ and $<L_{sun}>$. This is the end of the Step 1.

5. Now, with having the stationary part solved, we can calculate the matrix elements (the sensitivity factors for $dw(t)$ and $dL_{sun}(t)$) and the RHS vector (the measured T_{wvr} minus the stationary part at the 1st line of Eq. (4.1.7)). This is beginning of the Step 2.

6. Then we solve the system (4.1.7) and find the time variations *dw(t)* and $dL_{sun}(t)$. Ideally, they should have zero mean value (time average), but it is not quite true.
7. So, we correct the mean values $w_o = \langle w \rangle$ and $\langle L_{sun} \rangle$ by the means of variations found in the Item 6 and continue again from the Item 5 (rebuilding the linear system). When the means calculated in Item 6 are close to zero, we finish.

In practical arrangement, we start with initial guess of w_o by the method *getPWVforAnt()* from our sol_wvrgcal.py collection, which has been added in order to estimate the scan averaged PWV. It uses linear interpolation of PWV values as found in the SYSCAL table, which are 'bracketing' the scan (in time – see Fig 23). This, we believe, does better job then *aU.getMedianPWV()*, which provides a single PWV characterization for entire MS (i.e. averaged over entire ALMA array and all scans).

Minimizing of the $\langle L_{sun} \rangle$ dispersion (STDDEV) from the three independent values in the *Item 4* is performed via Newton-Raphson iterations. An illustration, on how the dispersion depends on varying value of w_o is shown in Fig. 71.

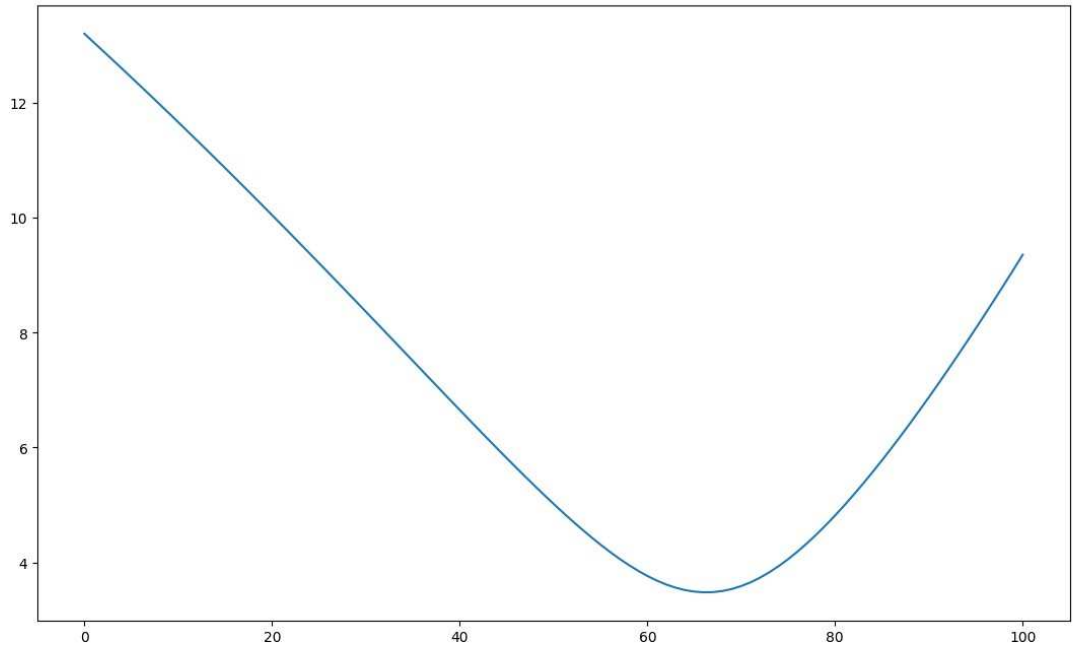


Figure 71: STDDEV of $\langle L_{sun} \rangle$ in Kelvins in a dependence on PWV. The x-axis is expressed in the PWV array index: We have calculated the dispersion for PWVs in the range 4.0 - 5.0 mm, with a step of 0.01mm. Hence, the minimum dispersion of 3.7 K is reached for PWV 4.64 mm (index 65).

The entire solution procedure is implemented as a method *dWdLs()* in *sol_wvrgcal.py* (*Annex 4*).

2.4.1.4 Results 2: Full PWV(t) and L(t) series

The new approach, which implements solution of the system (4.1.7) via iterative algorithm described in 2.4.1.3 as *dWdLs()* routine, has multiple advantages. First of all, it does for what it has been designed – removes the varying solar illumination from the WVR signal and leaves corrected 'net' PWV variations. But there is more in it: Unlike the approach based on Eq. (4.1.2) in Subsection 2.4.1.1, which extracts the fluctuations of PWV up to an unknown multiplicative factor, here we have full scaling and we are able to extract complete PWV, including its mean value. Results of the WVR-signal decomposition to PWV and solar illumination for antennas DA42 and DV08 (we have chosen the same as in *Results 1* for easy comparison) are shown in Figs. 72 – 75.

Fig. 72 shows the full PWV time profile over the on-source sub-scan of the solar scan 9 for DA42 (i.e., one of the 'problematic' DA antennas). We have selected just the on-source data in order to do comparison with our dPWV proxy derived form the science SPWs. Already at the first look it is apparent that the profile differs significantly from the blue curve in Fig. 68, which presents the WVR data without correction to variable solar illumination.

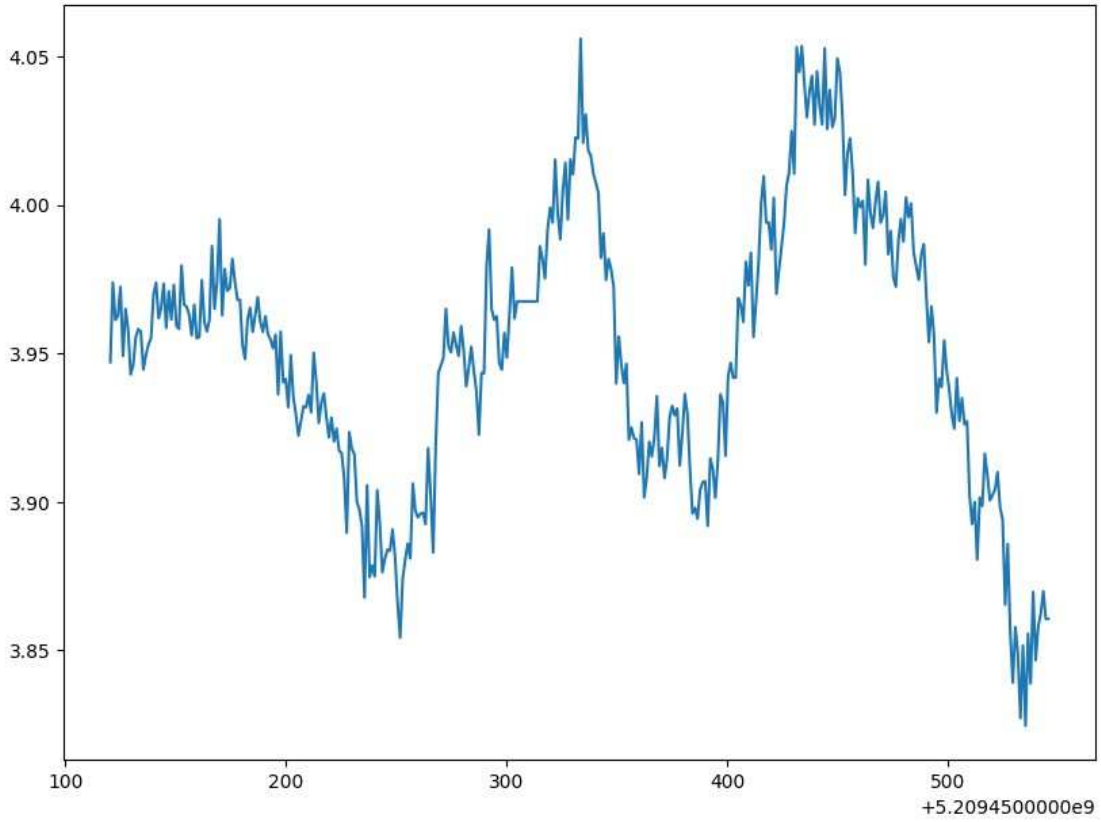


Figure 72: PWV for the antenna DA42 in mm, time axis in MJDsec.

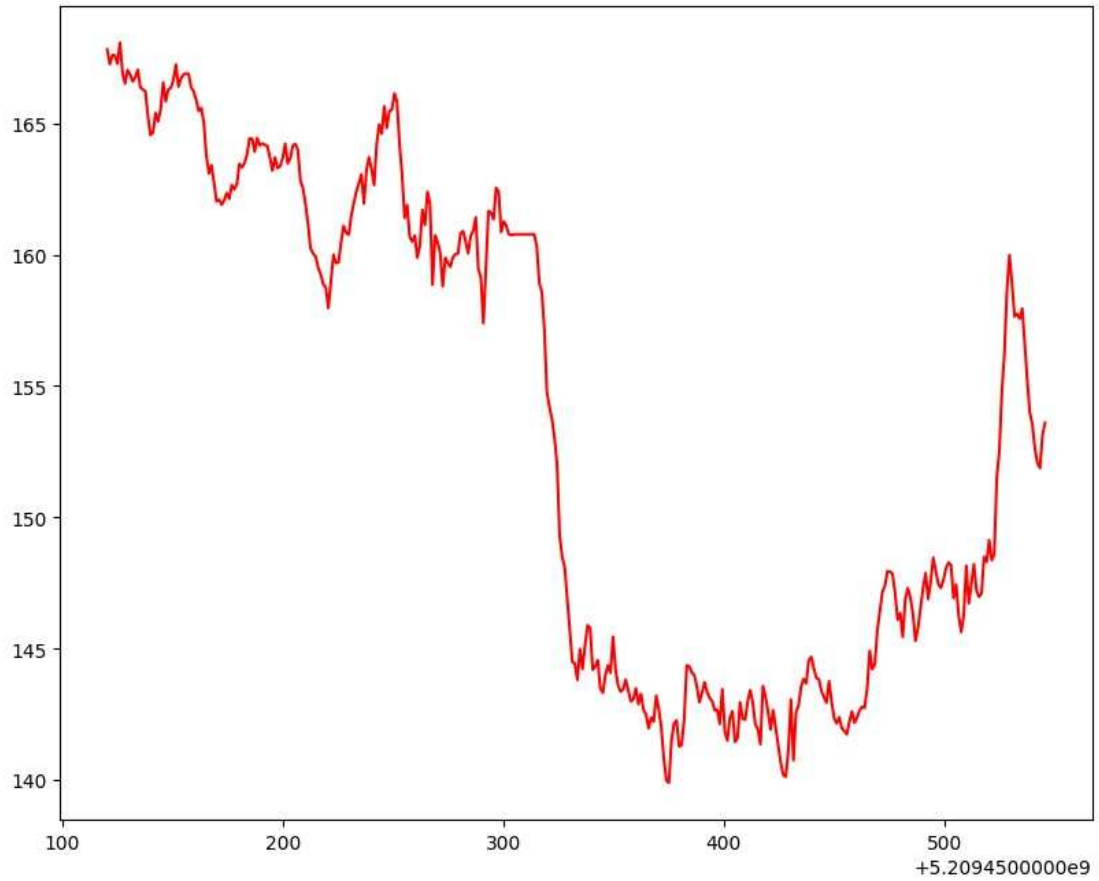


Figure 73: Solar illumination contribution to the WVR signal in Kelvins for antenna DA42. Time in x-axis as MJDsec.

Fig. 73 shows the $L_{sun}(t)$ contribution (solar illumination) to the WVR signal. Note that in line with Eq. (4.1.3) this is the 'net' illumination – for calculating its actual contribution to the WVR data it has to be scaled by atmospheric transmissivity that is specific for each WVR channel. The data shows abrupt drop by ~20K after time 300 s (+MJD offset) and then recovery at ~520s.

Figs. 74 and 75 present the same as Figs. 72 and 73, respectively, now for the DV08 antenna (which has shown good correlation between WVR and dPWV proxy already in the differential calculation by Eq. (4.1.2) in Fig. 67). As it can be seen in Fig. 74, the PWV profile remain qualitatively the same as in Fig. 67 (blue curve), but now we have its absolute scaling in mm, including the mean-value offset to the fluctuations.

Fig. 75 shows less contribution to the solar illumination for DV08 than Fig. 73 does for antenna DA42, but, most notably, the illumination is much more steady, varying only in units of Kelvins.

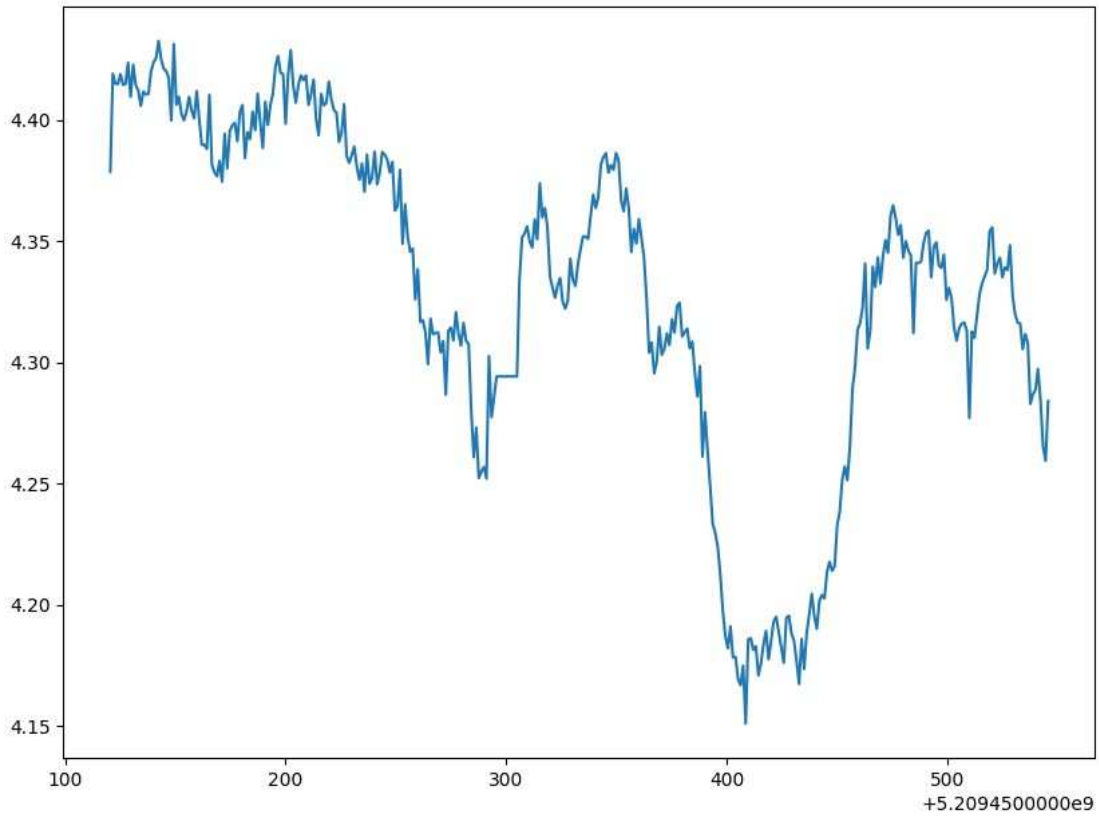


Figure 74: PWV profile for DV08, all notations like in Fig. 72.

The solution of system (4.1.7) – finding of the stationary parts w_o=<*w*> and <L_{sun}>, and their fluctuations *dw(t)* and $dL_{sun}(t)$, separately for WV emission and solar illumination, is implemented in *dWdLs()* per antenna. In order to make comparison of PWV variations *dw(t)* (just the fluctuating part), corrected by elimination of the variable solar illumination, with the dPWV-proxy calculated by Eq. (3.1.4) from the science SPWs for entire array, we have implemented a method *dpwvXsolWVR_raw()*, which iterates over all antennas (skipping the CM ones) and plots the comparisons the same way as displayed in Figs. 67 and 68 (these, were, actually, produced by an older version of the same method, which internally called solution to Eq. (4.1.2) for the arbitrarily scaled *dw(t)*). Results for antenna DA48 and DV07 are presented in Figs. 76 and 77. We have selected one of the DA antennas, which exhibited very weak or no correlation between WVR-based PWV variations and their science-SPW-based dPWV proxy counterpart in the previous approach (*Results 1*, without the varying solar contamination) and one DV antenna, which were qualitatively fine already with the simpler differential method, but anyway, they lacked the amplitude scaling (which is now solved). We admit that these

examples are one of best cases, but one can easily inspect the full dataset in online *Annex 5* to see that the correlation is now really very good for all antennas.

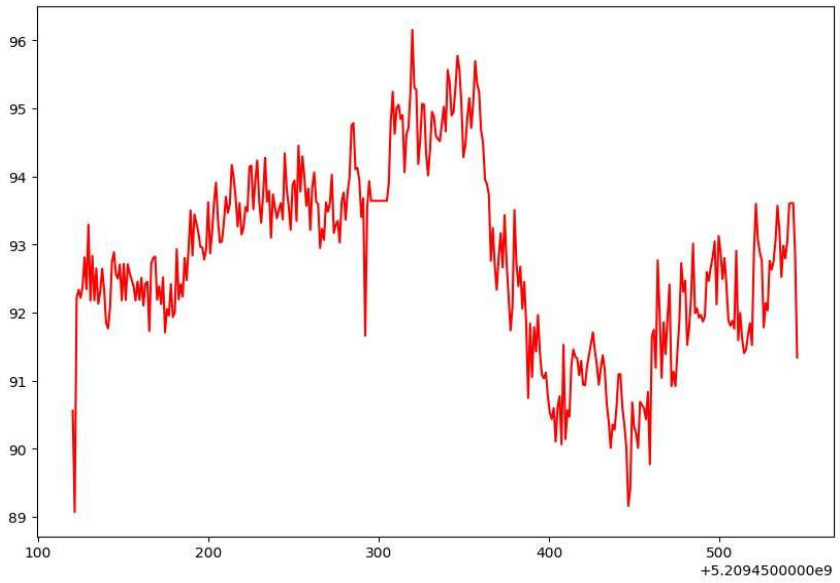


Figure 75: Solar illumination for DV08, all notations like in Fig.73.

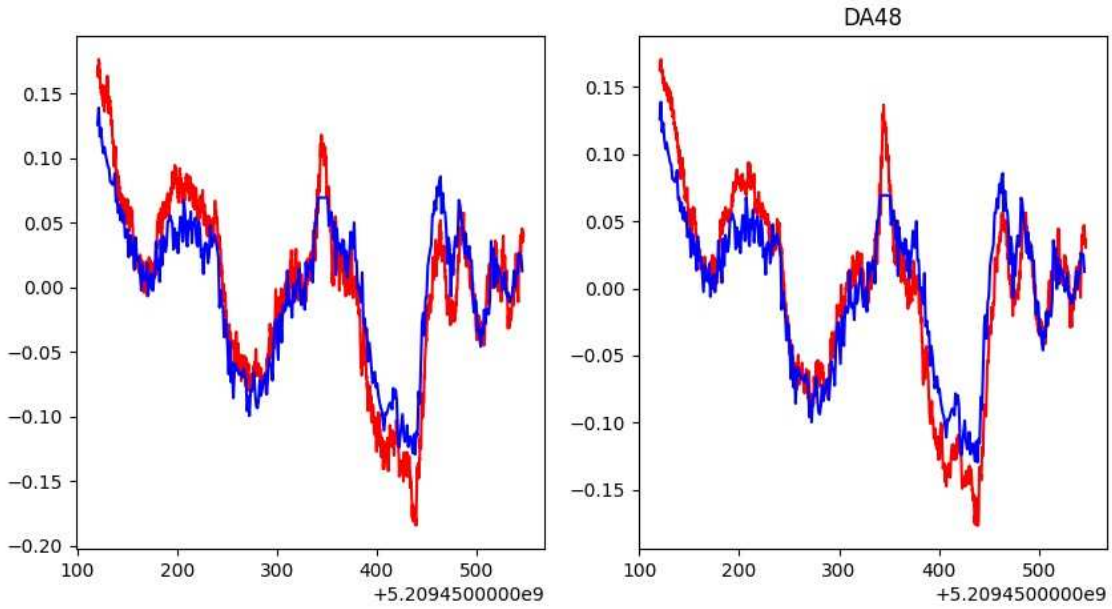


Figure 76: Antenna DA48: PWV fluctuations dW(t) as per raw WVR data (blue curve, mm) and their dPWV(t) proxy from the science SPWs (red, mm). The dPWV proxy has XX and YY pols, for that sake we have two panels. Basically the same display as in Fig. 67, but now the blue curve: (i) has corrected solar variable illumination, and (ii) need not to be scaled to range, it has absolute scaling in mm.

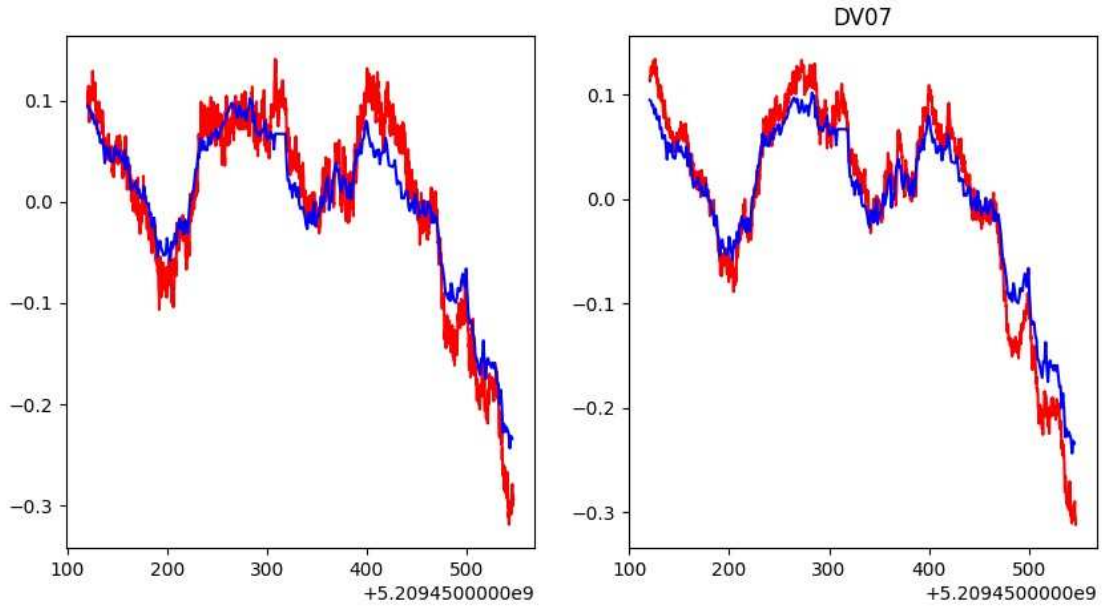


Figure 77: Like Fig. 76, now for antenna DV07.

2.4.1.5 Testing/Verification

In order to verify the method, we did some 'extreme' tests. First, we tried to detect the solar-contamination signal in the non-solar scan no. 5 (which observes the BP calibrator), where we, naturally, expect no solar contamination. Results for antennas DV08 and DV42 are shown in Figs. 78 and 79, respectively.

In particular, the DV08 result is nice, with very low mean around 3K. The DA42 antenna shows a bit higher contribution (instrumental?) around 18K, but anyway, both are pretty stationary with very low variations.

As another test we tried to make the solar illumination extraction from the WVR data for *entire* solar scan, i.e. including the first and last sub-scans that are off-source, and the including re-pointing of antennas between off- and on-source and back. This is actually beyond the assumptions for Eq. (4.1.7), because it was derived with neglected second order term $dw(t).dL_{sun}(t)$. This is not true in this case because the change in solar contamination is actually quite large (~90 K) during the re-pointing as can be expected already from the jumps in Fig. 73. The results are shown in Figs. 80 and 81. Fig. 80 shows clear rapid increase (and drop at the end of scan) of the solar illumination contribution, exactly as expected. Its value in the off-position is ~5K. Whether it is an error or there is still some minimal solar illumination that affects the WVR signal (we are 2 degs from the solar disc center, i.e., 1.75 deg from the limb) is a question. Anyway, in aspect of identification of what part of the total WVR signal is the solar contamination, the method performs well. Fig. 81 then shows the corrected net PWV variation after the solar-contamination removal. In ideal case it would be a smooth (well, up to ordinary

fluctuations around 0.05 mm) curve, with no jumps. This is not really true – the re-pointing of the antenna is still visible in the data by the jumps of about 0.2mm PWV in the times when the pointing has changed. Anyway, taking into account that we are beyond the assumptions for the approach, it is still very good result – vast majority of the raw WVR signal has been removed into solar illumination jumps, without that, the PWV would drop by tens or hundreds of mm (with a usual coupling of the raw WVR data to ATM model, neglecting the variable solar contamination).

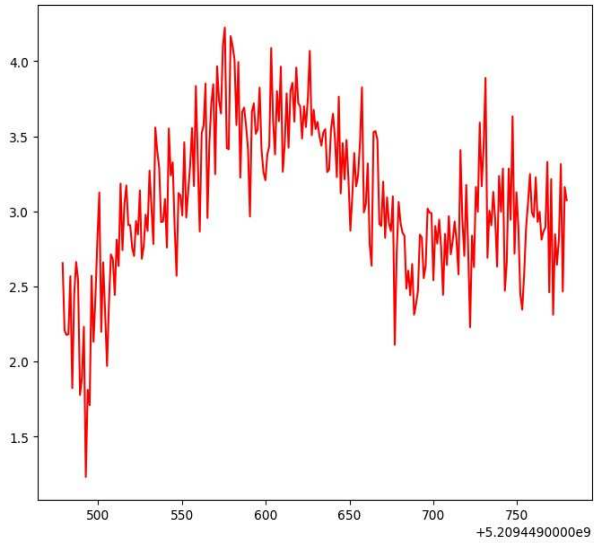


Figure 78: "Solar contamination" for the non-solar scan no. 5 (BP calibrator) at antenna DV08. Y-axis in K, time in x-axis as MJDsecs.

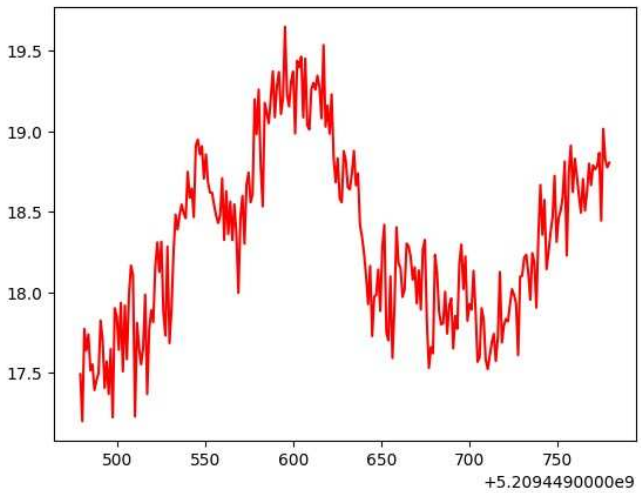


Figure 79: The same like in Fig. 78, now for DA42.

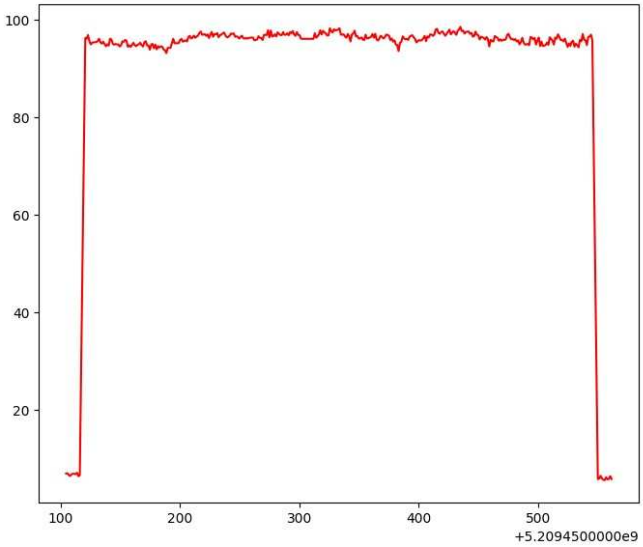


Figure 80: Solar illumination in Kelvins for entire solar scan 9 at antenna DV17. The jumps reflect antenna re-pointing between on-source and off-source positions.

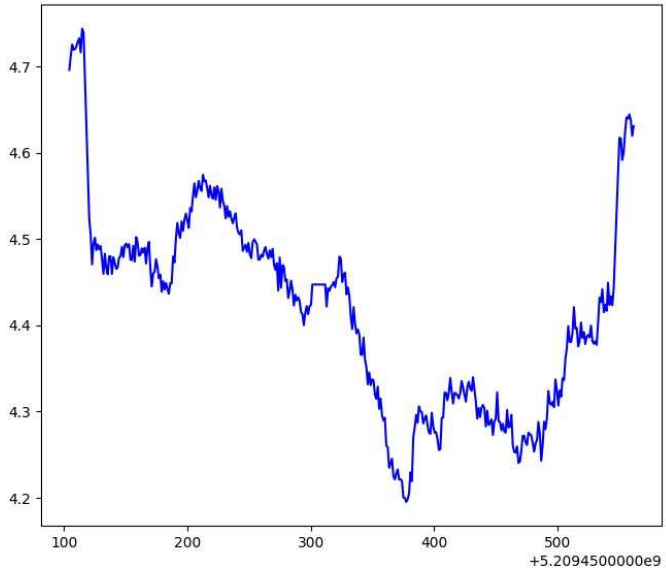


Figure 81: Net PWV time profile (in mm) at antenna DV17 for entire solar scan 9, after removing most of the raw WVR-signal jump into the solar illumination part in Fig. 80.

2.4.1.6 Our vs. SYSCAL PWV values

With the apparatus derived in Section 2.4.1.3 ("*Theory 2*") we are essentially able to calculate the PWV values at any time, for which we have the raw WVR data. For the scans not directed towards the Sun we may neglect the solar contamination L_{sun} and use direct coupling between the $T_{wvr}(f)$ temperatures, as measured in channels of SPW 4, and the sky brightness temperature $T_{BS}(f, w)$ as calculated by the ATM model for various WVR channel frequencies f and varying PWV value w. Conveniently, we can do it in logarithmic representation of $T_{BS}(f, w)$ and use Eq. (4.1.4) in an iterative way, to find the PWV, where $T_{BS}(f, w)$ matches $T_{wvr}(f)$ in each WVR channel f separately. The three values of PWV from the WVR channels 1 - 3 (we dropped the saturated channel 0) are then averaged to obtain a single PWV value in any time instant. This is implemented in routine *pwvCalc()* in the *sol_wvrgcal.py* collection (*Annex 4*). We assume that the standard ALMA procedure for obtaining the PWV values from the raw WVR temperatures is similar, but due to lack of open documentation we are not sure about the details.

In addition to our PWV calculations, there are SYSCAL-provided PWV values available at some sparse time instants over the MS. They are accessible from the MS/ASDM_CAWVR table.

Since we are interested in relation between the raw WVR data and SYSCAL PWV, we have over-plotted their values over entire period of execution block into one display in Fig. 82.

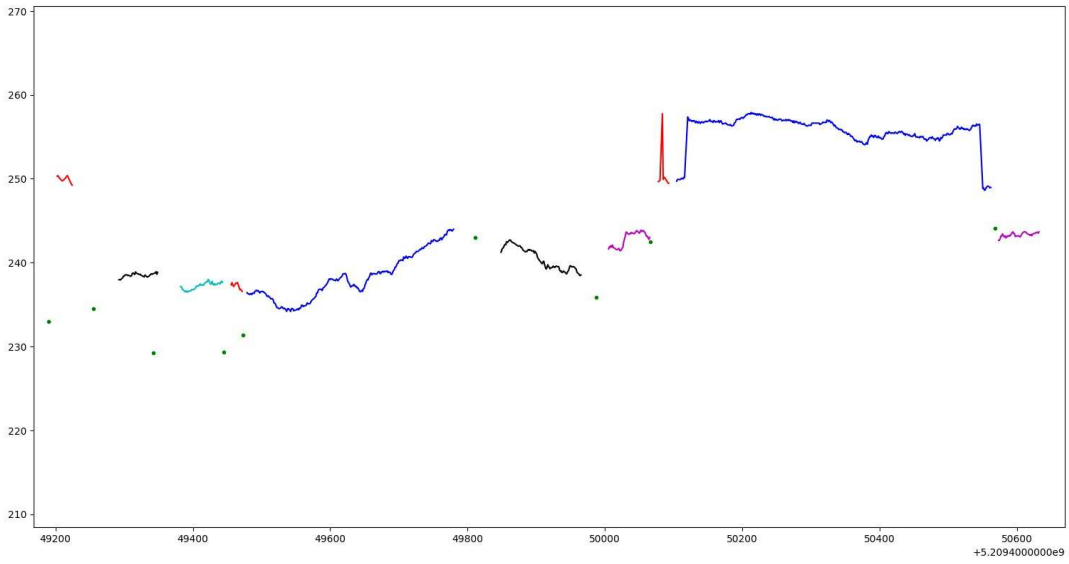


Figure 82: Raw WVR channel 1 temperatures (in K) as continuous curves of various colors for various scans (red - ATM scan, blue - BP and target, magenta - PhaseCal,...) over-plotted by arbitrarily scaled values of PWV, as read-out from the ASDM_CALWVR table – the green dots. All for antenna DV17.

We do not have ambition to use Fig. 82 for quantitative correlation of those two quantities, we are aware that other factors like air-mass (different for different scans) play role here. So we use the single

WVR channel (channel 1) and scaled the PWV data from the SYSCAL table by arbitrary factor (50x), just to boost them in the same range of ‘temperatures’ (around 240 K), having the green points in Fig. 82 close (in y-dimension) to the WVR curves. The purpose was to see, at what times (relative to WVR scans) are the SYSCAL PWVs evaluated. Apparently, they are evaluated at the edges of the scans or even in between them (which is a bit surprising for us).

Anyway, we have implemented a method *getPWVforAntenna(scan, antID)* in our collection (*Annex 4*), which for a given *scan* and *antID* takes the two SYSCAL-based PWVs that are ‘bracketing’ the scan in time, and using linear interpolation it calculates the PWV value interpolated to the central time of that scan. Then, we can use our method *pwvCalc()*, which calculates PWV from the raw WVR data at any time instant and extract the time average (over the scan) of those WVR-based PWVs for the same scan. This way we have approximately synchronized the SYSCAL-based and ours WVR-based PWVs, they are both located in the central time of the scan.

We can do it for all antennas (except of CM) and all scans that have non-contaminated WVR (all but the solar scan 9) and make a scatter plot between ‘our’ PWVs and the system-calculated (SYSCAL).

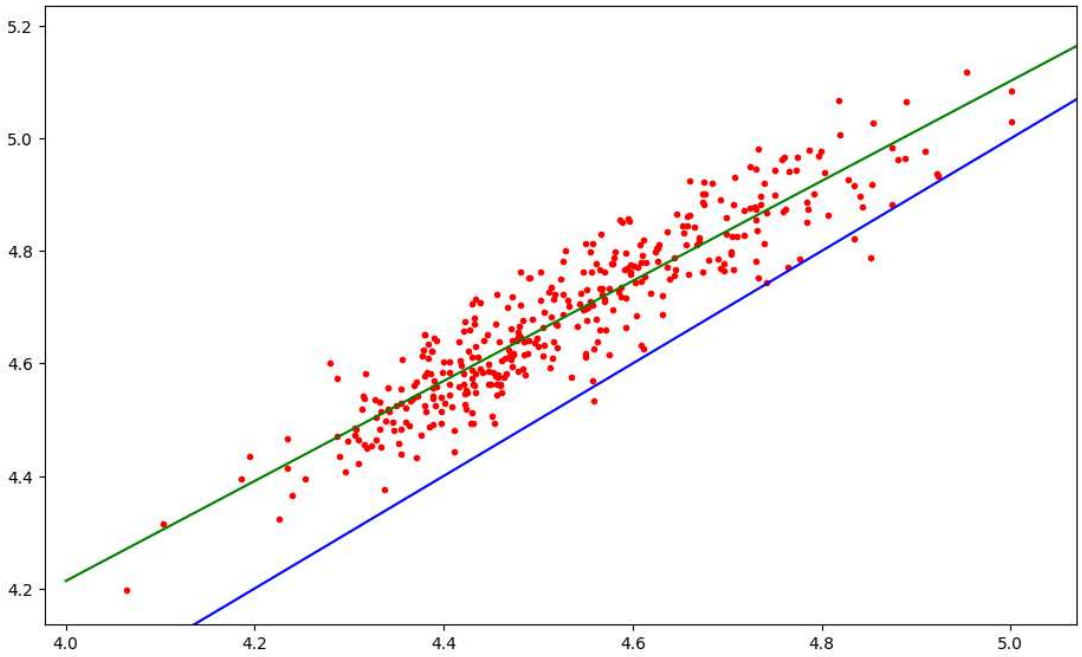


Figure 83: The red dots: PWV values obtained from ASDM_CALWVR (x-axis) vs. PWVs calculated by our method from the rae WVR data (y-axis), both in mm. The blue line represents ideal balance y=x, the green line is the best linear fit y=0.89x+0.79

This procedure has been implemented in *pwvCalcVerify()* and its result is shown in Fig. 24, together with the ideal-status line *y=x* (blue) and actual best fit (green). It is clear that our PWV values are systematically overestimated (w.r.t. the system-defined PWVs) by ~0.2mm, but at the same moment our slope is lower (~0.9) which means that larger fluctuations of T_{wvr} would map to lesser fluctuation in PWV than if the system was capable to measure them for any time instant as well.

We were a bit dis-certained by this finding and have tried explanation for it. We have re-checked the procedure and found three minor issues in our methodology:

1. We have left the default value Hw=1.0km for the height scale, with which water-vapor concentration decreases with height. After checking literature (Stirling et al. (2006): ALMA Memo 541) we have corrected it to more typical value for Chajnantor Hw=1.5km.
2. We have noticed that the four WVR channels in SPW 4 are of uneven widths (ranging from 1.5 to 2.5 GHz, which we have not realized before). We have corrected the ATM model for SPW 4 and the procedure *pwvCalc()* accordingly.
3. We should include the forward- and rear- spillover antenna efficiencies for the WVR SPW 4 data into radiative-transfer model, as we do for the science SPWs.

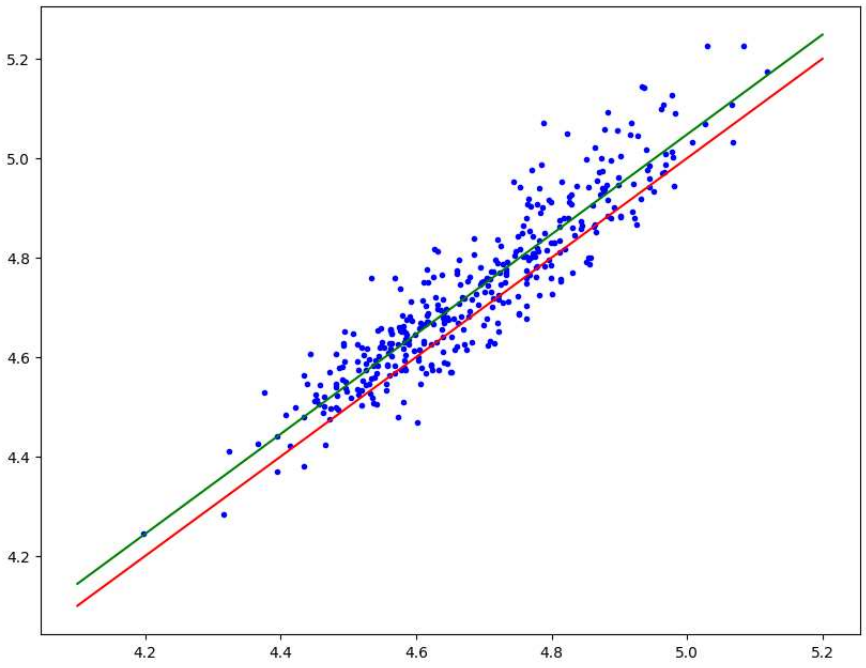


Figure 84: Improved correlation between PWV values (i) obtained by pwvCalc() from the WVR data in SPW 4 and (ii) values read out from ASDM_CALWVR table after corrections in procedure. Blue dots: our WVR-based PWVs (y) vs. SYSCAL PWVs (x), red line: y=1.0 . x, green line: regression with K=1.004 , Q=0.027.

As can be seen from Fig. 84, the correlation between the PWVs, as measured by our method implemented in *pwvCalc()* after incorporation of the fixes 1 – 3 identified above, and the SYSCAL-provided values, has improved a lot and basically the values are statistically the same: The relation is expressed by linear regression $PWV_{wvr}=K\ .\ PWV_{syscal}+Q$, with regression coefficients K=1.004 and Q=0.027 (reminder: before these corrections, the regression coefficients were K=0.89 and Q=0.79).

2.4.1.7 Further development – methodological convergence to the PWV-proxy calculations

As it can be seen from governing equations for the PWV-proxy calculations and for real PWV extraction from the raw WVR data contaminated by solar illumination – i.e., Eqs. 3.3.4 and 4.1.7, both approaches are formally actually very similar. Hence, inspired by the PWV(t)-proxy calculations, developed to a higher level in Section 2.3, we have implemented similar upgrades also to the extraction of PWV form the raw WVR data. Namely, the procedure *dWdLs()* has been consecutively improved by inclusion of

- non-linear solver via Jacobian matrix (mathematically rigorous approach),
- antenna efficiencies, and
- correction of calibration – with having the WVR data already in brightness temperature scale, the instrumental factors – the gain G and istrumental offset P_{inst} in WVR equation analogical to Eq. (3.3.17) are to be set to $G=1.0$ and $P_{inst}=0.0$ in "ACD calibration equations". We may, however, relax from having exactly those values and let the instrumental parameters slightly varying.

Resulting implementations (via consecutive upgrades) are in routines *dWdLs2()* and *dWdLs3()*, the latter one including the idea of "floating calibration". Essentially, they do not differ much from the dWdBdG() used for science-SPW PWV(t)-proxy calculations – the main difference is in just 4 WVR channels without sub-divison (no counterpart to 128-channel AC SPW) and lack of polarization information. We may say that the mathematical procedure for finding PWV(t) in presence of varying source brightness (whether it is real variation, scintillation or time-dependent scaterring at antenna construction) from science SPWs and form the WVR raw data converged together.

2.4.2. WVR vs. sci SPW-based PWV: Specifically designed test observation

As mentioned in the introduction to this part of the study, in order to verify the methodology for PWV(t) proxy-calculations from the science SPWs against the standard WVR data in solar case we need a specific test observation, where part of the antennas are observing slightly off-source in order to enable unsaturated WVR data for the solar scan. The main advantage that motivated design of this test lies in possibility to avoid calculation of the solar illumination/contamination, which brings additional uncertainty to the entire procedure as presented in the previous Section 2.4.1. We start with observation design, and then we are going to describe the actual performance and results.

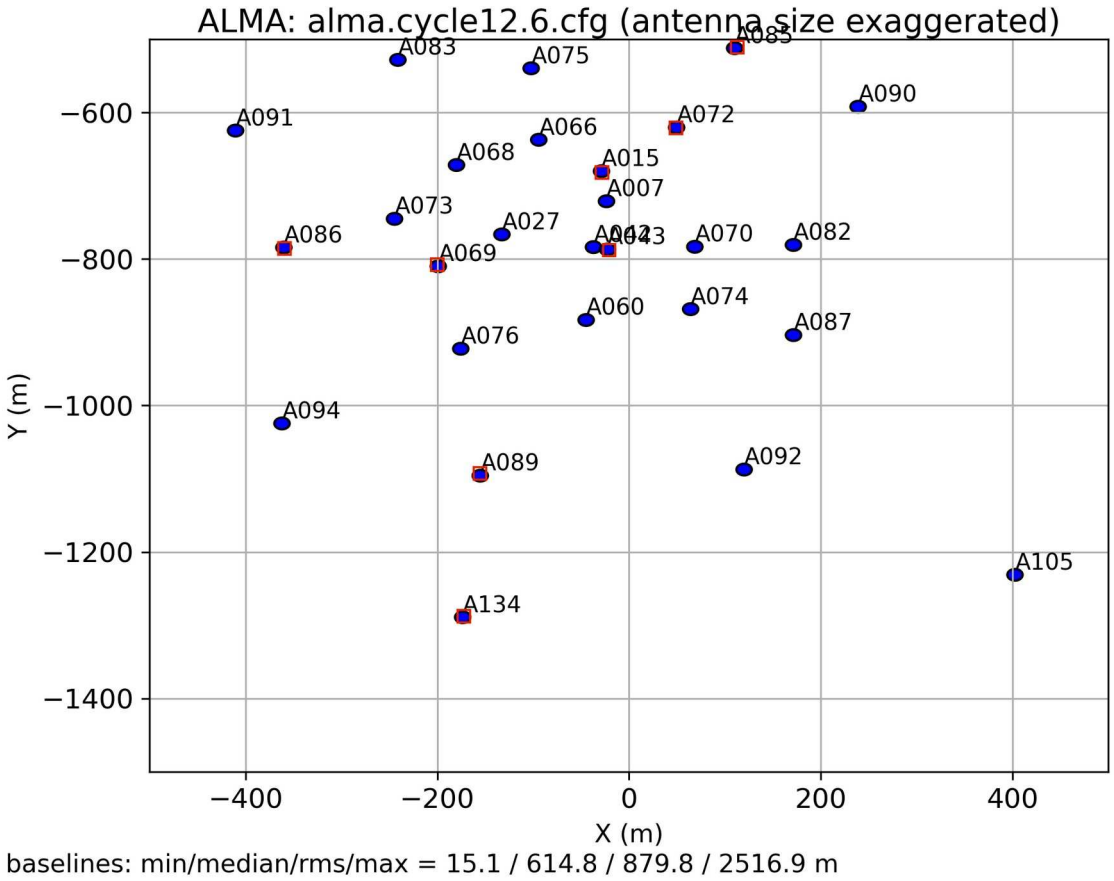


Figure 85: Zoom to the central part of the ALMA 12m array in configuration C-6. Red squares show the antennas that are not present for solar imaging – either deliberately, in order to make off-source WVR measurement or because of technical issues/flagging.

2.4.2.1 Setup of the observation

In order to asses performance of the PWV proxy for predicting the actual water-vapor variations in solar case, we need to compare them to the directly measured WVR signal. Both on-source and off-source signals need to be collected ideally from the same air column. This is not possible to be done exactly, but in order to approach the ideal situation, we identify close antenna pairs in the 12m array, which are located rather in the central part, as we do not want to loose our high resolution. One antenna in the pair is observing the Sun, the second is pointed to the sky a bit off-source. There is also one antenna pair, where both antennas point off-source but into different places at the sky. Still they keep pointing separation the same as the Sun-sky antenna pairs do. This is required as a ‘control sample’ for precluding the false-negative result: If PWV-proxy signal is very different from the WVR in the Sun-sky pair, the reason may be both in the fact that the PWV-proxy approach fails for the solar observation, but alternatively also because the exactly same air column (which we cannot have) is necessary for reproducing the WVR signal. If the control sky-sky pair shows (approximately) the same

WVR signal in spite of antenna-position and target-pointing separations, and the Sun-sky pair heavy discrepancy between PWV-proxy (on-source) and WVR (off-source), the situation is clear – only the first option (i.e. fail of the approach for the solar scan) remains valid.

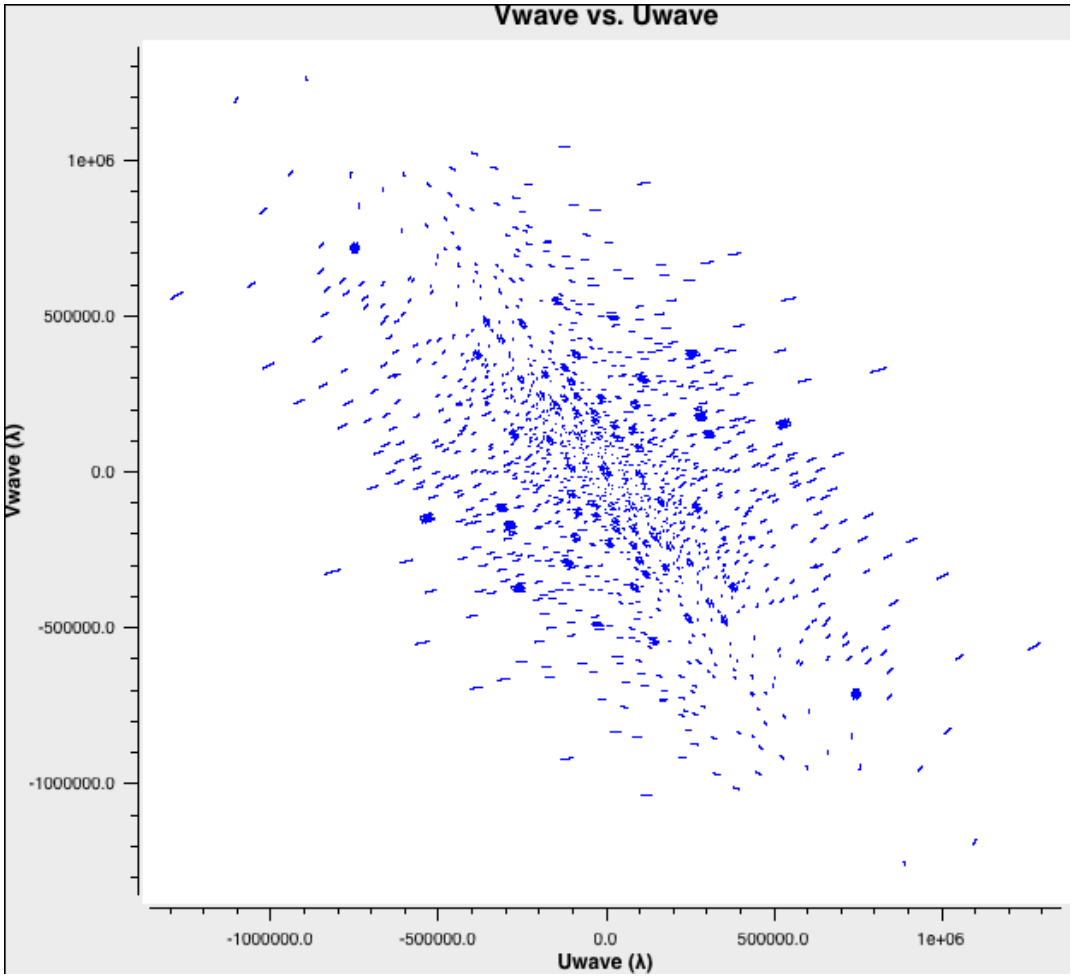


Figure 86: The uv coverage for the C-6 array combined with the ACA, including the mixed baselines. The antennas marked at the previous figure have been removed from calculation.

We would like to have at least two Sun-sky pairs for the case that one antenna fails – even bad T_{sys} is killing for correct finding of the dPWV, as explained above. On the other hand, the high-resolution observations are demanding on the total antenna count as the *uv* coverage is the critical factor for successful imaging – our simulations presented in the MTR have shown that if array configuration is extended just by one step above the limits found in this study (e.g., from C-6 to C-7 for Band 5), we are not able to reconstruct the image even for the case of zero-pressure atmosphere. In order to asses, how many antennas may be taken from the imaging and used for the off-source WVR measurement, we performed simulation as in the Section 2.1 above, but now with reduced set of antennas (the 'WVR antennas' taken away).

The first run has been done with total 8 antennas removed. This is the total 'non-presence' count, for any reason – WVR measurement as described here, but includes also antenna-not-ready or flagged cases. We used configuration C-6 and the antenna positions are identified by their pads name. The

antennas removed from the set has been located at pads A015, A043, A069, A072, A085, A086, A089, and A134.

Figure 85 shows a zoom-in picture of the 12-m main array configuration. Since we want to keep long baseline for the *uv*-coverage, the antenna picked-out are residing in the central part and they are marked by a red square.

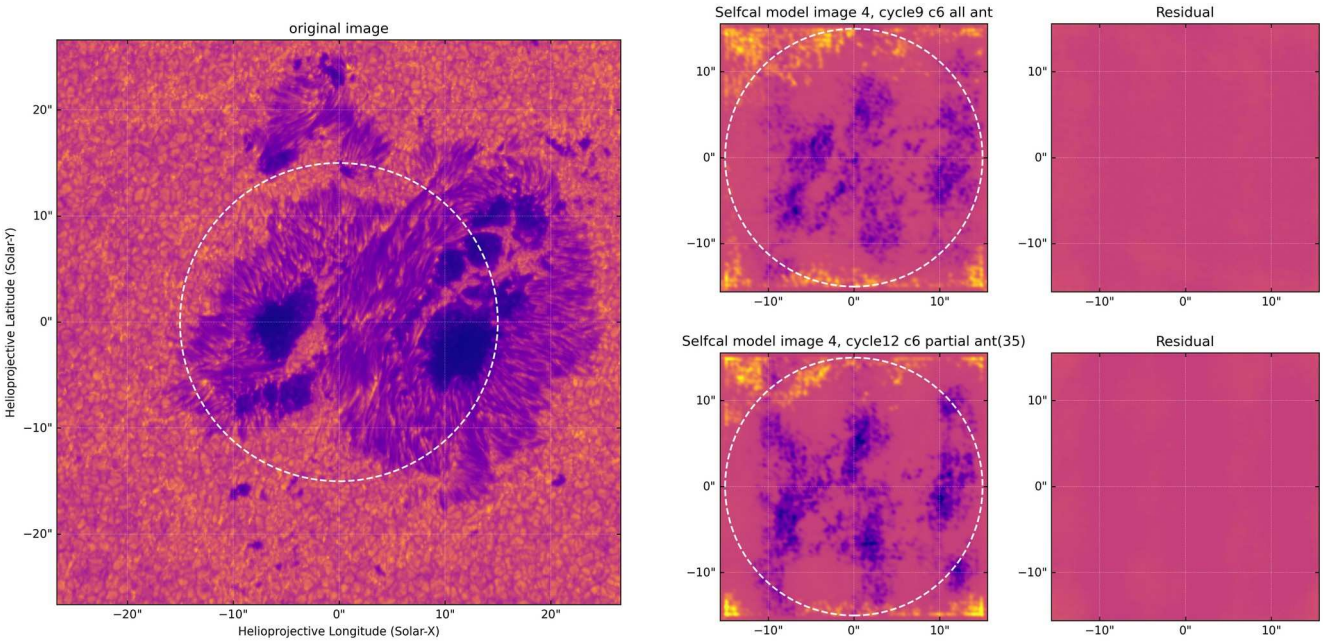


Figure 87: Simulation of the solar imaging in Band 5 in complete vs. reduced configuration C-6 using the phase-screen model for the atmospheric turbulence. The RMS over 1 min reference interval has been set again to 30 deg, as in the Section 2.1. The bottom right panels show reconstructed image and its residual in the reduced array (8 antennas missing) in comparison to their counterparts in the complete array. The 'ground truth' initial model is in left.

The corresponding *uv* coverage for C-6 configuration, including the combined 12m-7m baselines, but lacking the removed 8 antennas on the pads listed above, is shown in Fig. 86.

Finally, Fig. 87 shows the comparison between the fully self-calibrated images from this simulation and the one that uses the entire array (Cycle 9 configuration C-6). It can be seen that the quality of the image dropped significantly with the 8 antennas removed in comparison with the full array. In order to find still acceptable balance between the imaging quality and sufficient number of antennas we have run another simulation with just 5 antennas removed. This is more or less marginal case as we need at least 3 antennas for the WVR observation as described above and 1 – 2 antennas are typically flagged (for various reasons) in the solar datasets. As can be seen in Fig. 88, the image quality is, nevertheless, much better in this case.

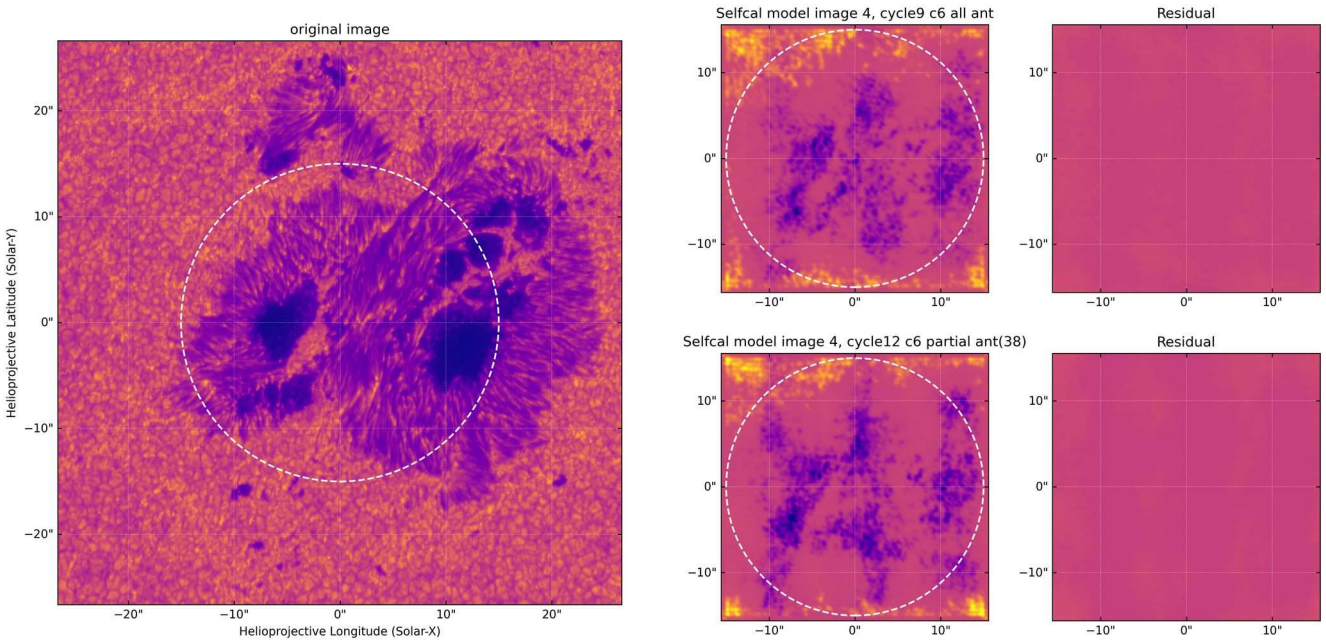


Figure 88: Like at the figure above, now with just five antennas removed from the main array.

2.4.2.2 The observations

The observation setup has been prepared in cooperation with our advisory team, P2Gs and AoDs and submitted as the project 2025.1.0005.CSV. The documents for scientific reasoning for the test, the AOT files, ephemerides, etc., are available either as the online *Annex 5* in the Development study data repository at http://wave.asu.cas.cz/shared/ESO/final_report or at the JIRA ticket for the specific test project 2025.1.0005.CSV at https://jira.alma.cl/browse/CSV-4018.

The first attempt has been done in Winter 2025 during configuration C-5. Nevertheless, setting up the sub-arrays for synchronous observations took some time (and failed attempts). The procedure is demanding for coordination because the solar-target ephemerides are only short-living and we would like to thanks once more the P2G team and AoDs for kind cooperation even during the Christmas break. Eventually, all has been ready in January 2026, when ALMA has already moved to more compact configuration C-3 and moreover, the weather was really bad. Anyway, the test has been performed as 'technical' under non-observing conditions (PWV 10 – 12 mm, rain) just in order to demonstrate viability of the co-temporal observations of two different targets in two sub-arrays.

The test performed well and the procedure has become the basis for the already scientific-merit test planned for next C-6 configuration in Summer 2026. What is remarkable, that even though the non-observable weather with the rain (we were able to detect it as increased absorption gradually covering the array as time progressed), we were still able to get the images of the Sun! As it is not of direct interest for this Study, we present the results just on-line – interested reader may find a brief descriptive text, QA2 report, and the calibrated (just standard way) and self-calibrated movies from this observation in Repository http://wave.asu.cas.cz/shared/ESO/final_report (*Annex 5*).

The second execution of the project (2 concurrent SBs – one for the solar target, one for the WVR off-the-Sun sub-array) took place on July 30th, 2026 under much better conditions. We have reached ALMA in desired configuration C-6, the weather conditions were excellent: PWV 0.6 – 0.9mm. The *listobs()* output and QA2 report can be found in the Repository http://wave.asu.cas.cz/shared/ESO/final_report, again.

In order to keep as much close air-mass as possible between the on-source and off-the-Sun (WVR) measurements (we wish to minimize the influence of different air column between the two synchronously executed SBs), specific ephemeris for the WVR SB has been prepared with offset of 2 degs form the Sun solely in azimuthal direction. For that sake, a routine *off_ephem.py* for generation of this specific ephemeris has been devised (with use of CASA *measures* toolkit and *Analysis Utils*). It uses the solar-disc center ephemeris from the *Solar Ephemeris Generator* (Skokic et al., 2015) as an input, tranforms to the local-ALMA Az/El frame, adds the offset in azimuth, and transforms back to RA/Dec. It can be found in the code listings attached as *Annex 4*.

2.4.2.3 Results of the July 2026 observations

The July 30 observation has bee executed under exceptionally (for the solar observing standards) good weather conditions with PWV in a range 0.6 – 0.9mm. Even with a diminished number of antennas (47 in 12m+7m mixed array, one of them being flagged during calibration/QA2) the imaging went smoothly and self-calibration loop ran with minimum failed *gaincal()* solutions. The results - QA2 report, MS listings, and, above all, self-calibrated solar movie with high resolution – are available in the online Repository at http://wave.asu.cas.cz/shared/ESO/final_report. Having in mind the main purpose of the test, we focus here, in the main text of the report, to the question of how much the PWV(t)-proxy time series (calculated from signal in science SPWs) correspond to PWV derived form the direct WVR measurements.

Comparison of direct and indirect PWV measurements using two concurrent subarrays

The key purpose of this test was to investigate, whether our PWV proxy as calculated from differential absorption of signal in the science SPWs (reminder: the most sophisticated method for this purpose, *dWdBdG2()*, decomposes the signal variations to parts attributed to PWV variations, scintillation, and intrinsic change of solar brightness temperature, and is even capable of few-percent corrections of the instrumental gains and power offsets) indeed corresponds to the PWV directly measured by WVRs. Here we present the results of direct comparison between direct WVR measurements obtained by off-the-Sun observation with Array 2 (antennas DA50, DA52, DA62, and DV03) with the PWV-proxy measured by Array 1 during the on-the-Sun target observation. Moreover, we add the WVR measurements of Array 1 for the (sub)scans, when the Array 1 was not pointed to the solar target – namely leading and trailing phase-cal scan, ATM scan preceding the solar scan, and the two short target sub-scans at the very beginning and very end of the solar scan, which are pointed 2 degs off-the-Sun.

First, we had to handle the problem with different than expected configuration (population of the antenna pads with antennas differs form our expectation in 2.4.2.1) of the Array 1. To that end, we have prepared an Excell table (available online in *Annex 5*) with antenna positions as per *listobs()*. The four antennas of Array 2 (off-Sun WVR measurement) are in the upper part, the lower part is composed by Array 1. For each antenna of Array 2 we have calculated distance to each antenna in Array 1 (right

columns) and perform sorting by that distance. Tables of the antennas of Array 1 closest to each antenna of Array 2 are in a separate sheet (per Array2 antenna) and the respective column of mutual distances is highlighted. This way we have found that, for example, the closest on-Sun antennas (Array 1) to the antenna DV03 in Array 2 (off-Sun WVR) are antennas DA42 (separated from DV03 by 38 meters) and DV01 (at 45 m distance).

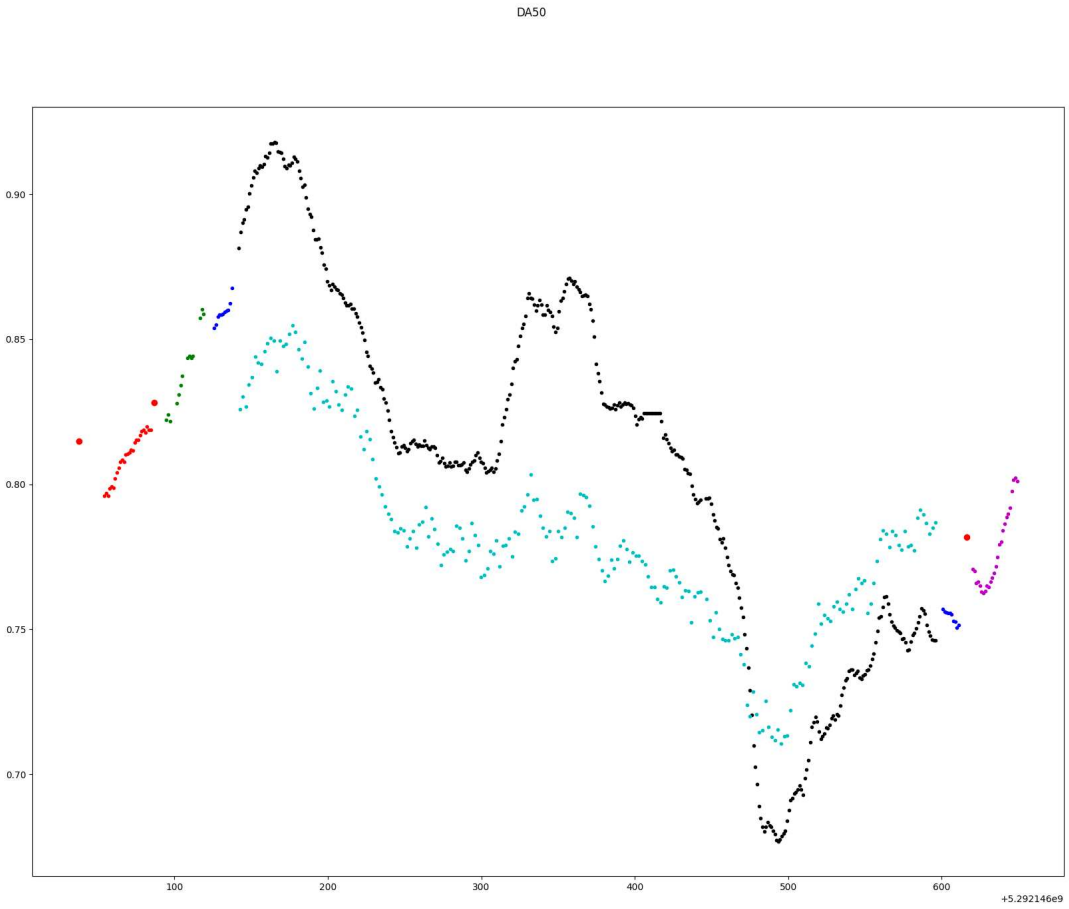


Figure 89: PWV curve (time in MJDsecs, PWV in mm) for four consecutive scans around the target scan. See description in the text.

In the following we present comparisons between selected Array1 – Array 2 pairs. Fig. 89 shows the PWV as calculated by *dWdLs3()* method in our collection (see *Annex 4*) from direct WVR measurement at antenna DA50 in the Array 2 – i.e. pointing 2 degs in azimuth off-the-Sun. The *dWdLs3()* method represents an improved procedure for inferring the PWV value from the raw WVR temperatures, which (inspired by our proxy calculation *dWdBdG()*) accounts for possible little (small units of percents) inaccuracies in determination of WVR brightness temperatures from the very primitive WVR data. The curves shown by different color codes as small filled circles are PWVs (in mm) for the following (sub)scans, respectively: Red = leading phase-cal scan, green = the ATM scan, blue = leading and trailing off-source sub-scans of the target scan (specific for solar observations),

black = the on-source sub-scans of the target scan (reminder: the target ‘on-source’ for Array 2 is an ‘empty space’ 2 degs apart of the Sun), and magenta = the trailing phase-cal scan. Time in the x-axis is in MJDsecs. The total PWV curve over all four scans is nicely (almost) continuous, the very little jumps may be attributed to the fact that we are pointing to slightly different positions at the sky (phase-cal is ~5 deg separated from the Sun). The three distinct bigger red circles (at times approx. 50, 80 and 620) are PWVs as read directly from the ASDM_CALWVR table and are added for comparison.

Even though we are pointing out of the Sun, we can still use our proxy method *dWdBdG()* for estimation of PWV from the science SPWs: We did the same job at the early phases of this study with band-pass calibrator data. In this case, the water vapor manifests itself by emission and because the science SPWs are rather far from the WV line center, the estimation has bigger error (as explained in Section 2.2). This proxy-based PWV curve, still for the same antenna DV50, during the ‘on-source’sub-scans is presented in the same figure as the cyan curve.

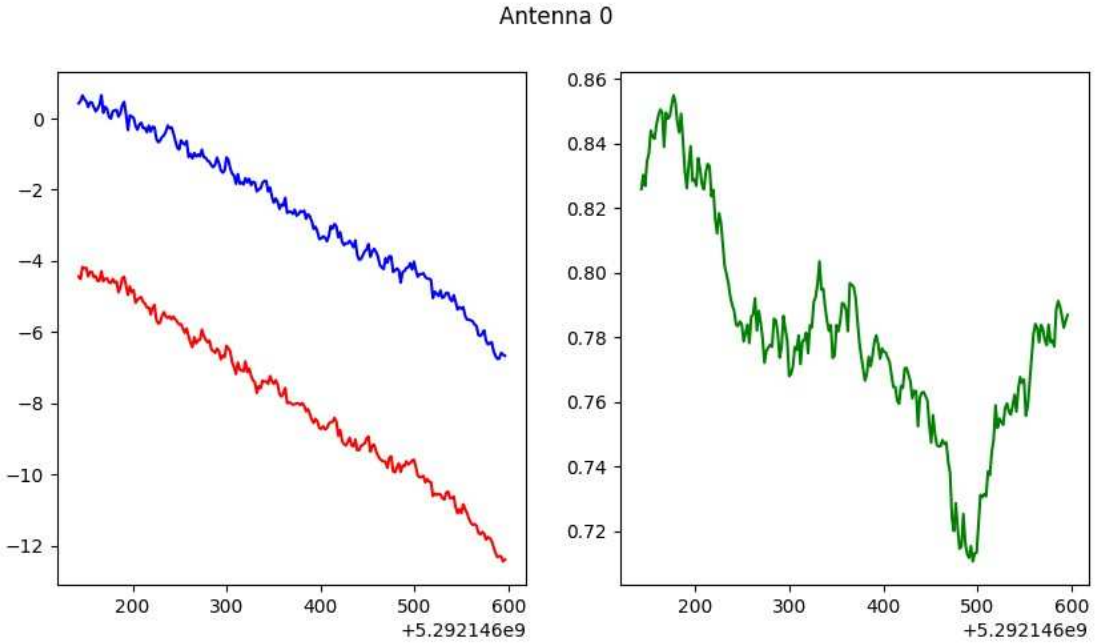


Figure 90: DA 50 (Array 2/off-the-Sun). Right: PWV (in mm), determined by our proxy method from the science SPWs, as a function of time (MJDSec). The green curve is identical with the cyan curve in Fig. 89. Left: The 'by-product' of the proxy method dWdBdG() is the decomposed solar contribution to the total signal. It is close to zero, as expected for the off-the-Sun observation by antenna DA50, despite we started our iterative solution with the normal solar temperature 6700K. Curves: red XX, blue YY polarization, values in Kelvins.

Albeit the large uncertainties in calculation of the PWV from the emission in science SPWs (we subtract two almost identical numbers, loaded by errors, and finally make a ratio of two such uncertain differences), the cyan proxy-based curve reproduces the WVR-based PWVs (black dots) surprisingly well. In order to check, how much our proxy method *dWdBdG()* is robust, we show another representation of the PWV-proxy curve, now alongside with the decomposed varying solar-brightness

contribution. The results for the same antenna DA50 are shown in the next Fig. 90. The right panel just reproduces the cyan curve from Fig. 89. On the left we see the decomposed solar-brightness contribution. As we are pointing 2 degs off the Sun, we expect it should be close to zero. And, indeed, it is. What is noteworthy on the result? We have started our iterative solution with expected solar brightness temperature of 6700K. Anyway it converged close to zero, demonstrating how robust the procedure is.

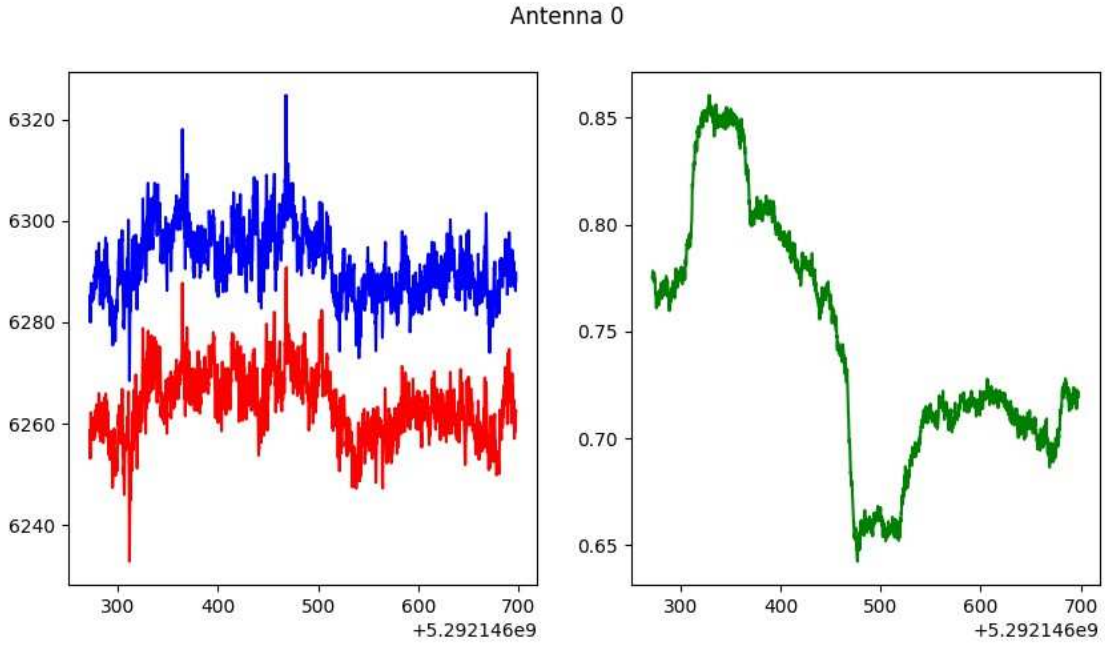


Figure 91: Like in Fig. 90, now for on-the Sun pointing antenna CM01. PWV-proxy (the green curve) corresponds quite well to direct WVR measurement in neighboring antenna DA50 – see the black curve in Fig. 89 – when the black curve is watched at the same time (start at approx. 270 MJDsecs).

Now, we can compare these PWVs obtained directly from WVR at antenna DA50 pointing off-the-Sun with the close antenna in Array 1, pointing to the Sun. Per Excell table sheet DA50 (*Annex 5*), the closest antenna in the Array 1 is CM01, separated from DA50 by 31 m. It does not have WVR measurement, but for our purpose it is irrelevant – we are comparing the real WVR from DA50 with sci-SPW PWV-proxy of the antenna CM01. The proxy-based PWV (right panel) together with decomposed solar contribution to the total sci-SPW signal (left) is shown in Fig. 91. Now we are pointing to the Sun, so the target brightness temperature is correctly ~6300K, varying on short time-scale most likely because of (tropospheric) scintillation. As it can be seen, the green PWV (proxy @CM01) curve shows, indeed, similar features as the black PWV curve (direct WVR @DA50), when we take into account the time-shift ~2min between execution of the SB1 and SB2 for on- and off- the Sun observations.

In order to see the correlation yet better, we put both measurements into a single figure in Fig. 92. The filled-circle curves (red, green, blue, black, and magenta) are – as in Fig. 89 – the PWVs as per direct

WVR @DA50 (off the Sun). Empty cyan squares represent sciSPW-based PWV proxy @CM01 (on the Sun; basically the green curve from Fig. 91). Finally, the yellow empty squares show sciSPW-based PWV proxy as (inaccurately) estimated from WV emission @DA50 (off-the-Sun; basically the green curve from Fig. 90). Correlation or at least strong similarity between the curves is obvious.

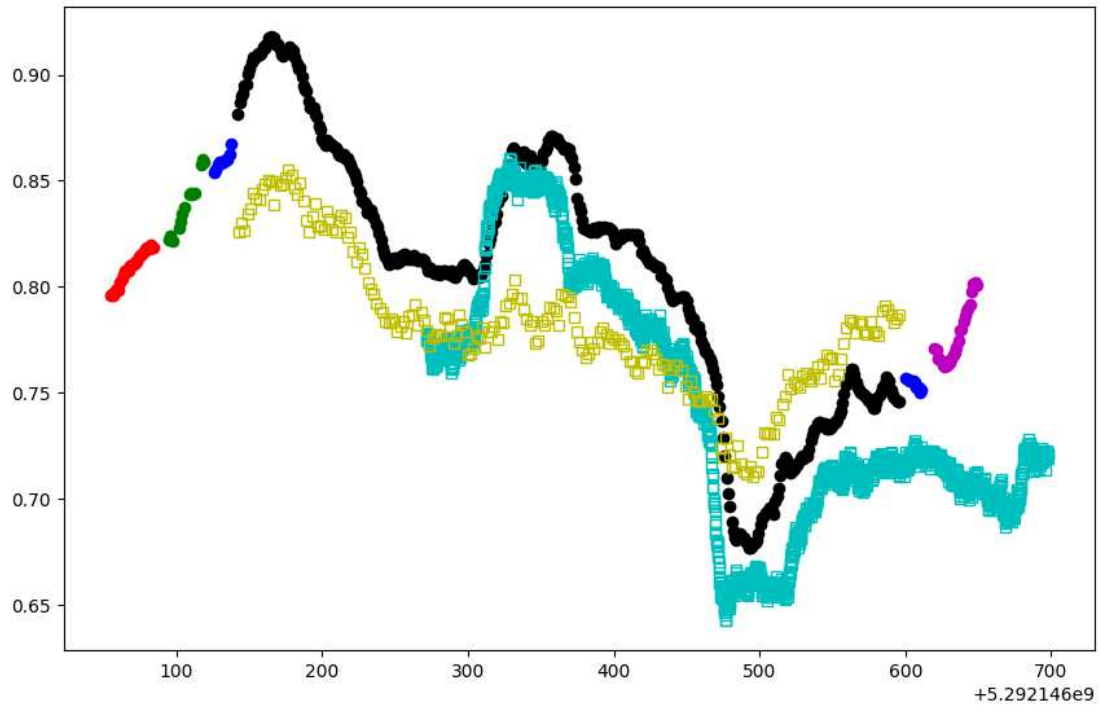


Figure 92: Composed Figs. 89 and 91 (right panel): Comparison between WVRs @DA50 (off Sun) and PWV proxy @CM01 (on Sun). See details in text.

In order to asses the difference between the WVR at Array 2 (off-Sun) antenna and PWV-proxy at its partner in Array 1 (on-Sun), e.g., the difference between the black and cyan curbes in Fig. 92, we would like to address yet another question: To what extent the PWV curves between the two antennas in a pair are different as a consequence of the antenna mutual separation. To that sake we need a close pair of antennas that are both equipped with WVRs. The Excell table with antenna positions shows closest pair of such antennas to be formed by DV03 (Array 2, off-Sun) and DA42 (Array 1, on-Sun), separated by 38m. We have done very similar composed figure to Fig. 92, but now it is enriched by the WVR measurements of the DA42 antenna for all the scans and sub-scans, which are not pointed to the Sun. The results are presented in Fig. 93.

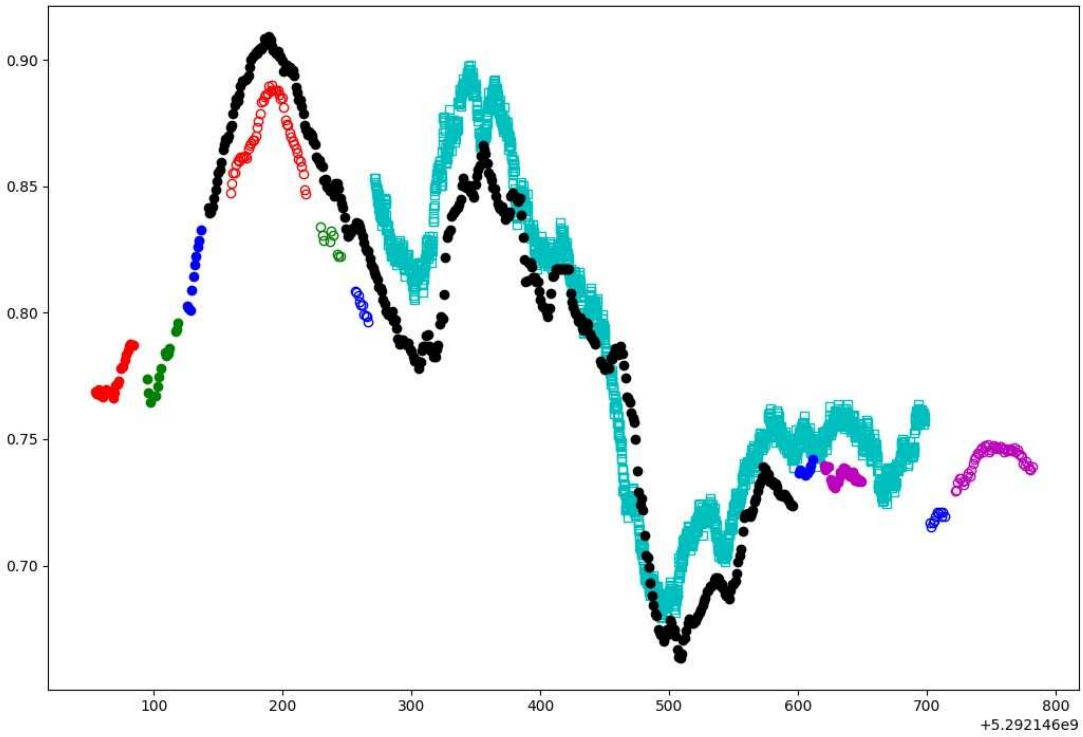


Figure 93: Comparison of WVRs and PWV-proxy for antennas DV03: off-the Sun , filled symbols and DA42: on-the-Sun, empty symbols. See the details in the text.

Meaning of the curves is as in Fig. 89, but now we have two sets: Filled symbols are for antenna DV03 (off-Sun) and empty symbols for antenna DA42 (on-Sun). The red, green, blue, black and magenta are, again, WVR-based PWVs for leading phase-cal, ATM, tgt-off-source, tgt-on-source, and trailing phase-cal scans/subscans. For antenna DA42 (on-Sun) we clearly do not have black (empty) symbols, because the on-source subscan has saturated WVRs. The empty-squares cyan curve is the PWV proxy calculated from WV absorption in the science SPWs at antenna DA42. Again, similarity of the curves is striking. Moreover, now we have time overlap between direct WVR measurements by both antennas DV03 and DA42 in the period of ~150 – 270 MJDsecs, so we can compare the difference between the PWVs to be attributed not to the different methodology (direct WVR vs. sciSPW proxy) but to different antennas, and, namely, their separation (different atmosphere in the ray-path). Basically, they are of the order of ~0.05mm of PWV, which is very similar to the difference between the black (WVR @DV03) and cyan (PWV-proxy @DA42) curves. So, in this sense, the proxy gives the same result as WVR within the uncertainty caused by scanning different atmospheric mass.

Summary: Results of the test demonstrate reliability and robustness of the sciSPW-based proxy method for the PWV calculation. We got similar PWV curves by both methods (direct WVR vs. proxy), the difference between them is not significantly higher than the difference between two direct WVR measurements at two mutually separated antennas. The correlation works surprisingly well even in this

extremely (for the solar observations) low PWV dataset (with median PWV less than 0.8mm). Together with our older comparison of PWV proxy vs. direct WVR carried out at the Sun-contaminated but still valid WVR data of the December 2023 observation (with – in contrast to the current dataset – quite a high PWV) we may say that the method is usable along a broad span of PWVs (0.8 – 5.0mm).

2.5 Application

In the previous sections we have developed method for calculation of the PWV(t) time series from the signal present in science SPWs in Band 5, implemented it as a library of Python routines, and – in our humble opinion – quite convincingly demonstrated that the PWV calculated by this method is tightly correlated to the PWV inferred from the WVR measurements. Now, we are just the last step from reaching the final goal – making use of the results for the phase corrections, i.e. , providing the solar counterpart to *wvrgcal()* gain-calibration method. We are going to make this last step in this section.

2.5.1 Calculation of the optical-path dependence on PWV for different frequencies (dispersion).

Just a brief re-cap: Insofar we have used the factor of *-25.0* for recalculation of the PWV variations in mm into phases in radians (see, e.g., Section 2.2 above). It has come from the factor between PWV and the additional optical-path: 6.7 – this value has been reported by Richard Hills in his original intro material (Annex to MTR). It is then scaled by relation between the optical path and the phase, which for the wavelength about 1.57 mm (190GHz), gives, indeed, a combined factor of approx. value of -25.

We would like to first verify that the scaling factor is correct and sufficiently general over entire pass-band of the Band 5, and second, provide more exact routine for recalculation of the PWV to the phase delay. To that end we have implemented a new routine *wv2path()* in sol_wvrgcal.py collection. The method internally calls *at.getDispersivePathLength()* and *at.getNonDispersivePathLength()* in a loop for all channels of our science SPWs. We have been in particular interested in the dispersive part – i.e. whether the factor 6.7 can be really considered as a constant over frequency domain of Band 5 – the general knowledge says that the refractive index of dielectrics has an irregularity at ‘resonant’ frequencies, where the absorption occurs – known as Kramers-Kronig (KK) relations, so our intuition tends to contradict the statement.
Result: Indeed, the non-dispersive part is constant over SPW 5 and its value is 6.677e-3 m per 1 mm of PWV. The dispersive part is very slightly changing around its average value of 1.0e-4 m per 1 mm PWV and thus, represents merely a negligible correction. We are probably too far from the line center so that the KK-relations have no significant effect on the refractive index. Both paths (dispersive and non-dispersive) have been found to depend linearly on the PWV, so their effective (per 1 mm PWV) value does not depend on the PWV parameter in the *wv2path()* routine. To sum up: A constant factor 6.7 as a scaling between PWV and additional optical path is appropriate.

Anyway, for the correct calculations we switch to the more exact approach, which respects (a small, but anyway some) dispersivity and also the dependence of the phase not only on the optical paths but also on the wavelength (which, is, again, quite small as the total bandwidth of Band 5 covers 16 GHz; at

f_{LO}=198GHz it represents variation in the interval +/- 4%). Namely, we use our *wv2path()* method for calculation of the phase delays for 1mm of PWV and then we sum both contributions (dispersive + non-dispersive) to find the frequency-dependent phase factor $Pf^{spw}(f)$ defined in Eq. 20. Its value, indeed, varies between -25.0 and -29.0 over entire range of all four SPWs, so the initial approximation -25 is not as bad. Because *applycal()* takes care about frequency dependence of phases within a single SPW, the gain calibration tables (*G*- or *T*-type) are not frequency dependent (just on the SPW, but not on their channels). For that reason we calculate the $PF^{spw}(f=f_c)$ at the central frequency of each SPW, having just a single number representing the SPW.

These *PF* factors are subsequently multiplied by the actual PWV(t) as obtained by solution of the RT problem using the most sophisticated *dWdBdG()* method. This way we can calculate the absolute atmospheric phase delay for each antenna

$$\varphi(ant, spw, t) = PF(spw) \cdot PWV(ant, t) \qquad (20)$$

If we subtract the value for the reference antenna, we get the relative phase delay w.r.t. the reference. This value can be in the form of complex gain $exp[i(\varphi_{ant}-\varphi_{ref})]$ already written into the calibration table.

2.5.2 sol_pwvGcal(): The solar counterpart to wvrgcal()

The above-described procedure is implemented in the new routine *sol_pwvGcal()*, which may be considered as the replacement of *wvrgcal()* for the solar scans. It copies the header and entire structure of the WVRGCAL table without its data (using *norows=True* in *tb.copy()*) and subsequently fills its columns by relevant data – times, antenna_1 IDs (antenna_2 is the refant) and the complex gains calculated as complex units (i.e. just the phase calibration) $exp(i.\varphi)$, where φ is determined by Eq. (20). For SNR column we adopted the WVRGCAL strategy and set value of 5.0, preventing thus flagging in subsequent *applycal()* due to SNR below 3.0 (the default threshold).

We were interested, what is the relation of our time-dependent WV-calibration to the standard phase calibration applied to the target, i.e. the linear-interpolated phase solution between the two adjacent phase-calibrator scans. The results are shown in Figs. 94 and 95. Fig. 94 shows the phase solution calculated directly by Eq. (20), i.e. including the possible wrap-overs over 360deg as blue curve, and already wrapped-up linear interpolation for the scan-based (solint='inf') phase solutions for the phase calibrator for antenna 7 (CM10) in SPW 11. Fig. 95 shows the same for antenna 20 (DA55) in SPW 7. The red and green lines are phase-cal interpolation for XX and YY polarizations. It seems that the phase-calibrator solutions interpolated in time between the adjacent phase-cal scans (before and after the target scan) at least mimic the trend of the WV-based phase delays. However, for the more distant antenna DA55 the phase solution is clearly wrapped-over.

Because the linear trend seems to be contained in the WV-calibration, one may think about dropping the standard phase-cal interpolated solution. However, we cannot fully abandon it, because it contains info about relative phase delay between XX and YY polarizations which is, likely, of instrumental nature. So, we have to somehow combine both solutions before applying them to the solar target – we discuss this in further text.

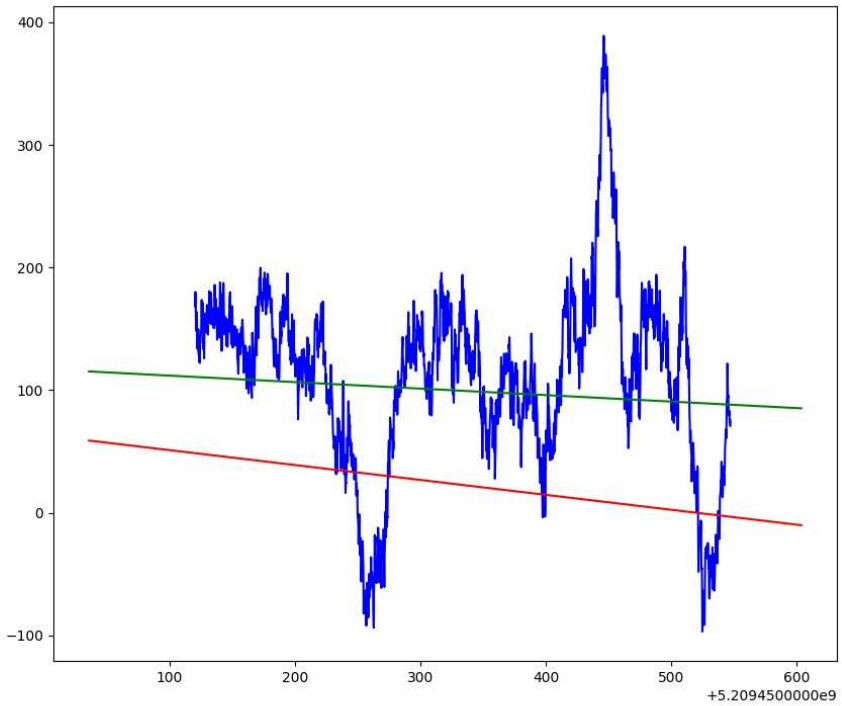


Figure 94: Comparison of the WV-based phase delay relative to refAnt (blue) with XX (red) and YY (green) linear interpolations for phaseCal solutions in the adjacent phaseCal scans for CM10 in SPW11. Phase delays are in degs.

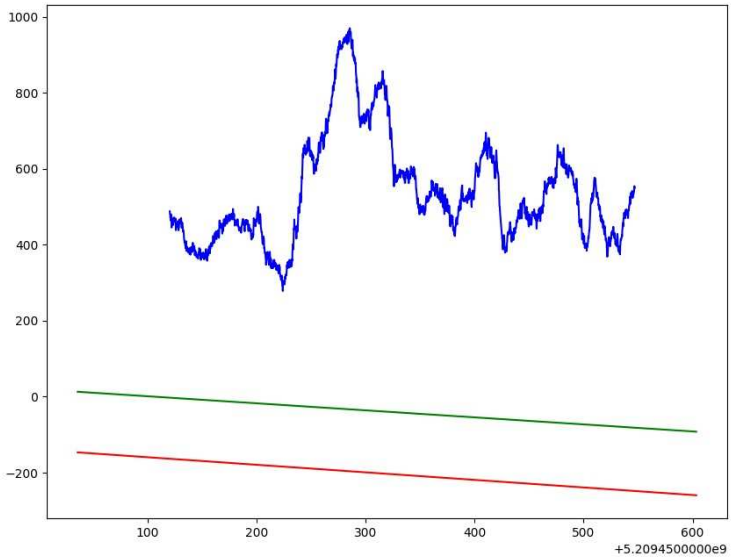


Figure 95: Like in Fig. 94 *, now for DA55 in SPW7.*

As the last result we present comparison of the phase delays extracted from the gaintable calculated by *sol_pwvGcal()* with the self-cal gaintable for the solar target. This is the most relevant result as it aims to direct application of the entire procedure for solar target phase corrections. Results are shown in Figs. 96 (for antenna id3) and 97 (for antenna id44). We are showing the phases extracted from the complex gains contained in gain caltables using our helper method *getPhase()*, so the initial phases for both *sol_pwvGcal()* (a single solution displayed as the green curve) and gaincal() (two solutions XX and YY displayed as read and blue curves respectively) are already wrapped to –π to +π interval. However, as we use *np.unwrap()* the phases may (and namely in case of self-cal gain caltable they do) leave that interval as time proceeds.

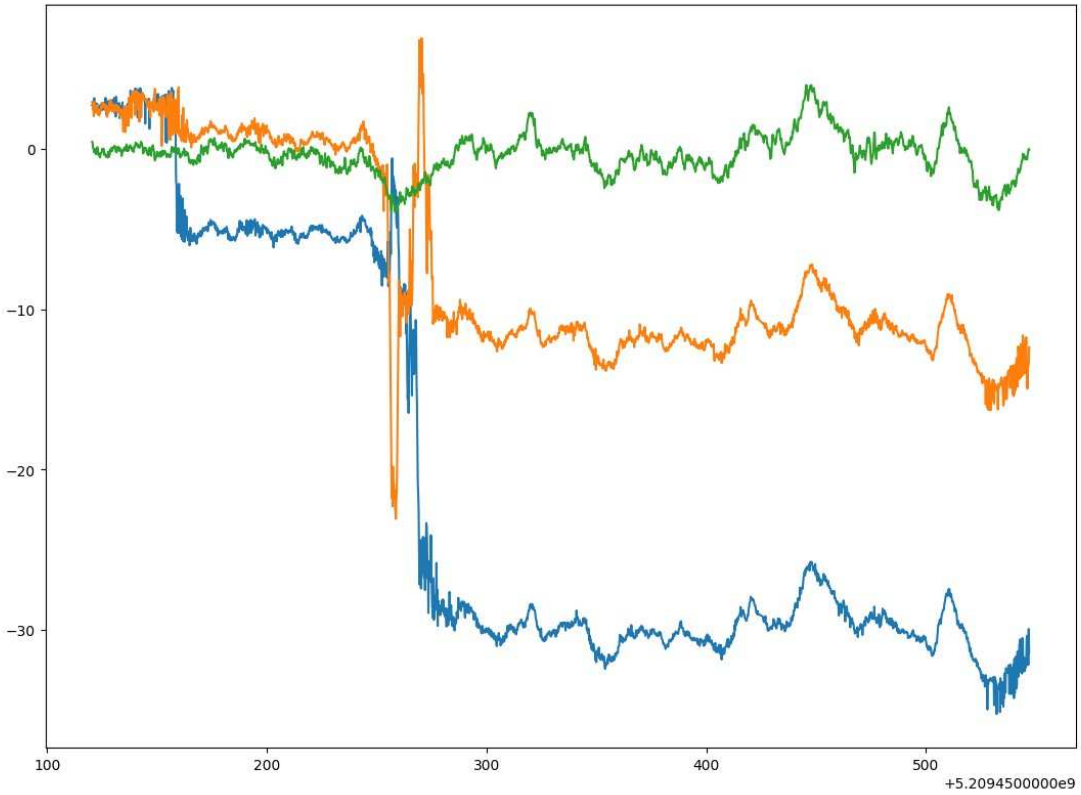


Figure 96: Phases of the complex gains from the WV-gcal table (green) and self-cal gaintable – for XX (red) and YY (blue) polarizations as a function of time (MJD Secs). The phases are displayed in radians(!). Curves are for antenna id3.

If one neglects the jumps in self-cal gaintable phases caused by failing self-cal solutions at specific time instants, qualitative similarity between the curves is clearly apparent.

If the phases for more quiet (with less self-cal jumps) antenna id44 are subtracted, i.e., WV-gcal minus self-cal is plotted, we get a basically straight line with some oscillations – see Fig. 98. It looks like the slope of this line exactly corresponds to the linear trend for the adjacent phase-calibrator scans. The remaining oscillations may be attributed to intrinsic dynamics of the solar atmosphere, the sudden short jumps originate in 'jumpy' self-cal solution. The red and blue curves are for XX and YY (respectively) polarizations of the phase-differences calculated as PWV-based minus self-cal-based solutions, the accompanying straight lines are the linear interpolations of the phase-cal INF solution in the adjacent phase scans (as in Fig. 95). Clearly, the self-cal solution has been performed at the 'standard'-calibrated data, so the instrumental XX – YY difference is already removed (that is why the red and blue curves are almost identical). What remains – in addition to intrinsic solar dynamics, which changes the phases, too (and we do not see it in our self-cal table that is based on the entire-scan, i.e. INF, image model) – is the linear trend: In the solar self-cal solution it has been removed by call to *applycal(interp='linear')* for the phase-calibrator INF gain caltable in the standard calibration, while the PWV-based phase solution contains the full phase variation, including the trend.

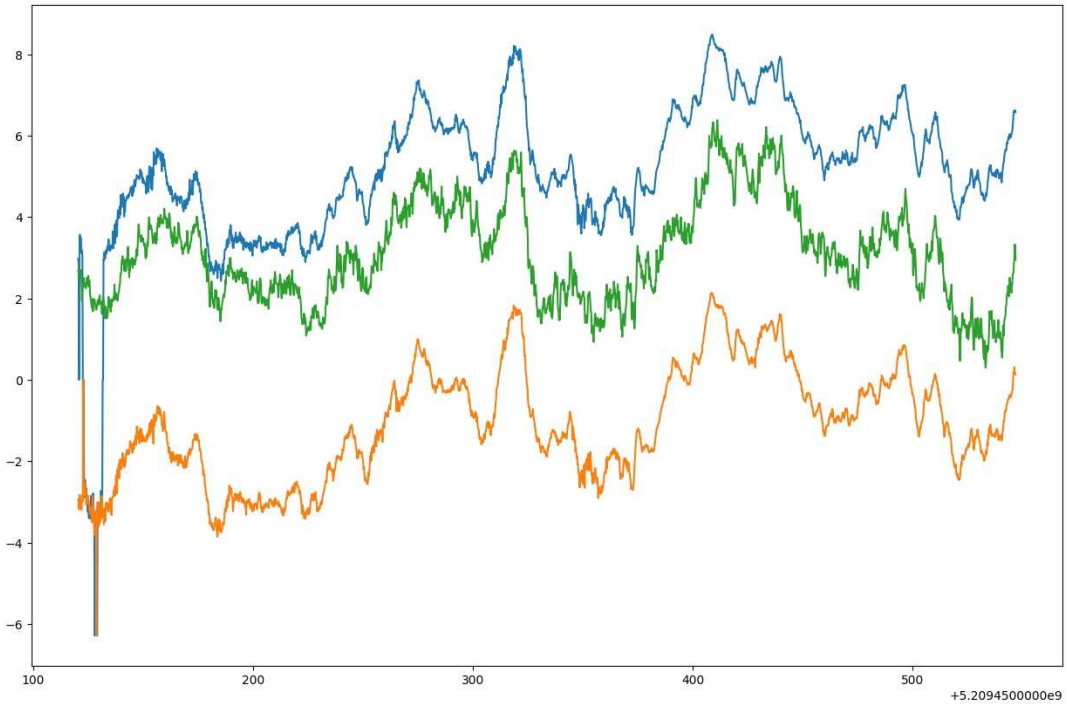


Figure 97: Like in Fig. 14, now for antenna 44.

Summary: We are able to produce solar equivalent of the WVRGCAL table and use it for corrections (via the standard *applycal()* taks) of the phase delays in the solar scan. Still some work is needed in

combining the standard linear-interpolated phase-calibrator solutions (phase-inf gain caltable from the standard calibration procedure) and our WV-gcal caltable. Clearly, if we apply the WV-gcal calibration on top of the fully calibrated (without self-cal) MS, which has the phase-inf gain caltable applied already, we apply the trend twice. (Note that this does not happen in non-solar data: There we apply the *wvrgcal()* solution both to target and phase calibrator, so the trend compensates; in solar case we apply the solar WV-gcal to the solar data only, the phase calibrator does not use *wvrgcal()* in solar datasets). This double-application of the trend is visible in the difference between WV-gcal and self-cal gaincal(). On the other hand, we cannot drop the standard phase-calibrator based solution completely, as it contains also info about relative instrumental delay between XX and YY polarizations. Merging of both calibration will come in next section.

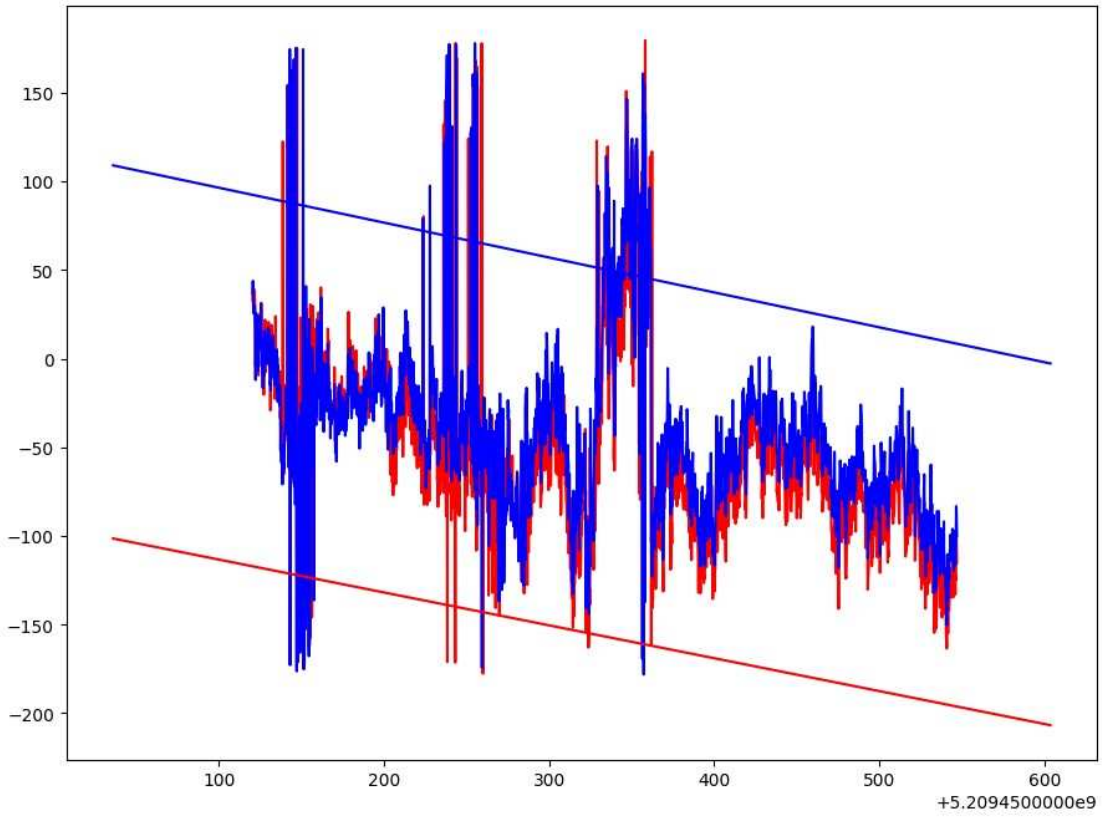


Figure 98: Solar phase solutions (in degs): PWV-based minus self-cal-based (the curves), overploted by linear interpolation of the phase-calibrator-solution (straight lines). Red: XX, blue YY polarization, all for antenna id 44 and SPW 5. The difference between PWV and self-cal have been wrapped into -180 – +180 deg interval in order to prevent phase jumps, which are present as a consequence of self-cal troubles at specific time instants.

2.5.3 Final improvements

Receiving the second, recent, test dataset from July 30th 2026 has appealed on further possible improvements of the method. They are motivated by Fig. 65, which displays the scan-average solar brightness temperatures in dependence on antenna ID. The impression from that figure says that there may be systematic dependence of T_B on the antenna type, most likely related to their different efficiencies. Just for recap: We are using the default values of 0.95 and 0.90 for rear- and forward-efficiency of all antennas in the set, respectively, in line with White et al. (2017). As the July 2026 data show similar distribution, we decided to make two further improvements of the method, related to this observation: Defining the specific antenna efficiencies for different antenna categories, and, after this correction, make use of the solar brightness temperature averaged not only over the scan, but also over the ALMA array. Details are described in the next two sub-sections. They are followed the last subsection on this topic – description of the promised improved *sol_pwvGcal2()* method, which combines the atmospheric and instrumental (polarization-dependent) delays into a single gain cal table.

2.5.3.1 Improved antenna efficiencies

We have done similar analysis as is for the 2023 dataset presented in Fig. 65 also for the new data from July 2026. In spite the arrays for both observations are not identical, vast majority of the antennas are present in both observations, so we can compare their 'performance'. In the XLS file *project22_25_Tb.ods* located in the on-line repository http://wave.asu.cas.cz/shared/ESO/final_report (*Annex 5*), we have separate sheets for 2023 data, 2026 data, and their combination. We are calculating relative normalized efficiency factors for each antenna based on its scan-averaged solar T_B value divided by expected theoretical T_B for the observation (this depend namely on the position angle – see the formula by Alissandrakis et al., 2020). It can be seen that the grouping of antennas by their type (CM, DA, and DV) is present in both datasets. Moreover, for specific antennas DA64 and DV08, which are present in both datasets, we have visible 'under-performance': Their normalized T_Bs are significantly smaller in both datasets and both polarizations than the rest of the antennas. The other antennas also exhibit some variations, but they are randomly up- and down- for different dataset and/or polarization, while DA64 and DV08 shows sort of systematic behavior (if one dares to speak about it in case of just 4 data series).

We have excluded clear outliers and calculated the correction factors for efficiencies for all mentioned categories (CM, DA, DV, and the two outlying antennas). The results are summarized in the following table:

Antenna	η_I corrected	η_F corrected
CM	0.941	0.891
DA	0.947	0.897
DV	0.956	0.905
DA64	0.929	0.880
DV08	0.943	0.894

We count with this new values in our improved routines for both sci-SPW and WVR-based PWV(t) calculations *dWdBdG2()* and *dWdLs4()*.

2.5.3.2 Making use of array-averaged T_B

Related to the previous topic, we came to an idea to use another 'boundary condition' for our optimization (least-square) system: Once the antenna efficiencies are corrected, the Fig. 65 re-calculated with improved efficiencies should give more-or less flat distribution of T_B. In other words, the solar brightness temperature may vary from one antenna to the other on short timescales (because of the scintillation revealed in Section 2.3), but there is no reason why the scan-averaged values should differ – all antennas should measure the same value. In order to enforce such requirement, we postulate requirement

$$\langle T_B(t)\rangle = \frac{1}{NS}\sum_{t=0}^{NS-1} T_B(t) = T_{tgt} \qquad (5.3.1)$$

where NS is the number of time samples and T_{tgt} represents the target (the Sun) brightness temperature averaged over *antenna array*. We now add this equation to the already extended grand-system formed by set of equations (3.3.17) and (3.3.18), which represent the radiative-transfer equations and ACD calibration equations, respectively. Visually it means that the matrix in Fig. 54 gets extended by further lower belt composed by identical simple 3x2 matrices of the form (0,1,0/0,0,1) – in the Python implementation we call them *M-blocks* (from array *M*ean) – which, again may repeat, in order to increase their weight in the system; the repetition factor is called *mWeight*. The matrix J^T *x* J from Fig. 55 gets a bit more complicated by this additional equations, namely the upper-left main N-block receives also off-diagonal components (insofar they have been zero). Nevertheless, still it is possible to make very easy and low-CPU-demand inversion (again found with help of the Gemini AI) and thus solve the system block-wise.

This trick has been incorporated to methods for calculation of both science-SPW based PWV(t) – we have implemented a new routine *dWdBdG2()* for that sake – as well as for calculations of PWV directly from the WVR raw data (a new method d*WdLs4()*). While for the sci-SPW PWV calculation we set the array-mean target temperature at the RHS of Eq. (5.3.1) to the averaged solar brightness temperature over entire array, in case of WVR calculations via *dWdLs4()* we force zero target temperature in case of WVRs pointing well off the Sun.

We shall present the results obtained with this upgraded procedures in the next subsection, where we have accumulated one more minor improvement.

2.5.3.3 Correct weights for the grand-system of equations

During implementation of the last described improvement we have revised the weights, with which the individual equations enter the system. In course of this procedure we have realized that the SQLD SPWs, which have much higher cadence (we have 0.192 s for the AC spws, while 12x higher cadence in SQLDs), have also correspondingly smaller received powers, both in radiative-transfer equations (3.3.17) and the ACD calibration equations (3.3.18). Hence, they enter the system with lower weights than the equations for the AC SPWs. There is a simple cure to that issue: We scale all equations of the combined set (3.3.17) and (3.3.18) by inverse gain. This way, all the equations are expressed in equivalent temperatures, which are mutually co-mensurable (all radiative-transfer eqs. have their RHS around 6000K, as well as the all array-mean equations, the ACD calibration equations move around 300 K). This upgrade comes only to the procedure *dWdBdG2()*, as the raw WVR data, handled in *dWdLs4()*, are already scaled (calibrated) to temperatures.

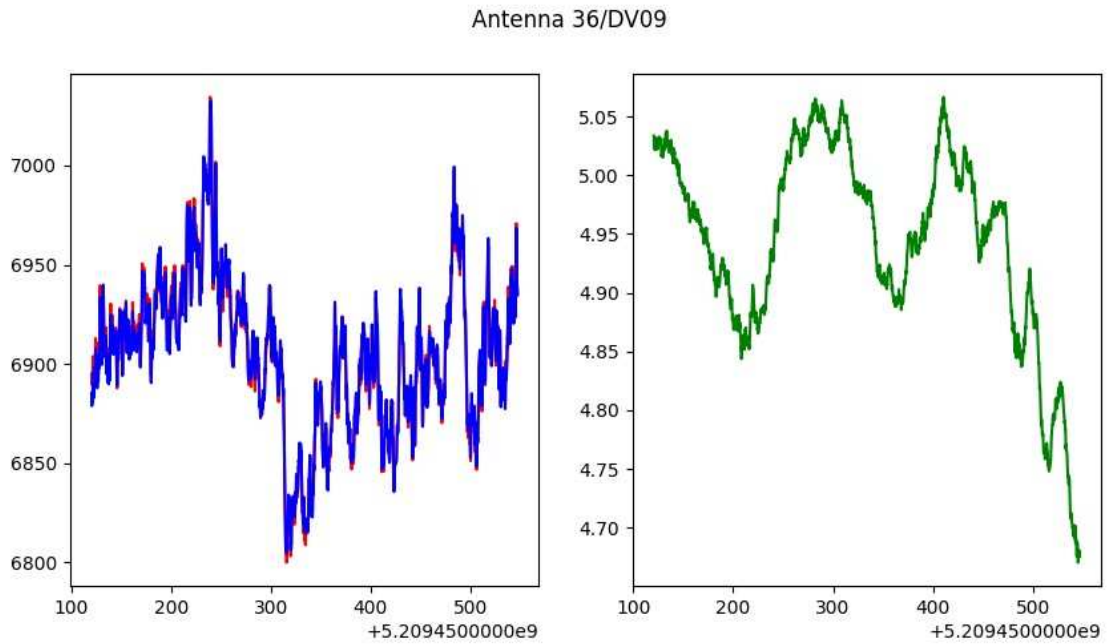


Figure 99: Left: Solar (apparent/scintillating) brightness temperature (K) in XX/YY (red/blue) for 2023 dataset and antenna DV09. Right PWV (mm) for the same data. Time is expressed as MJDsecs.

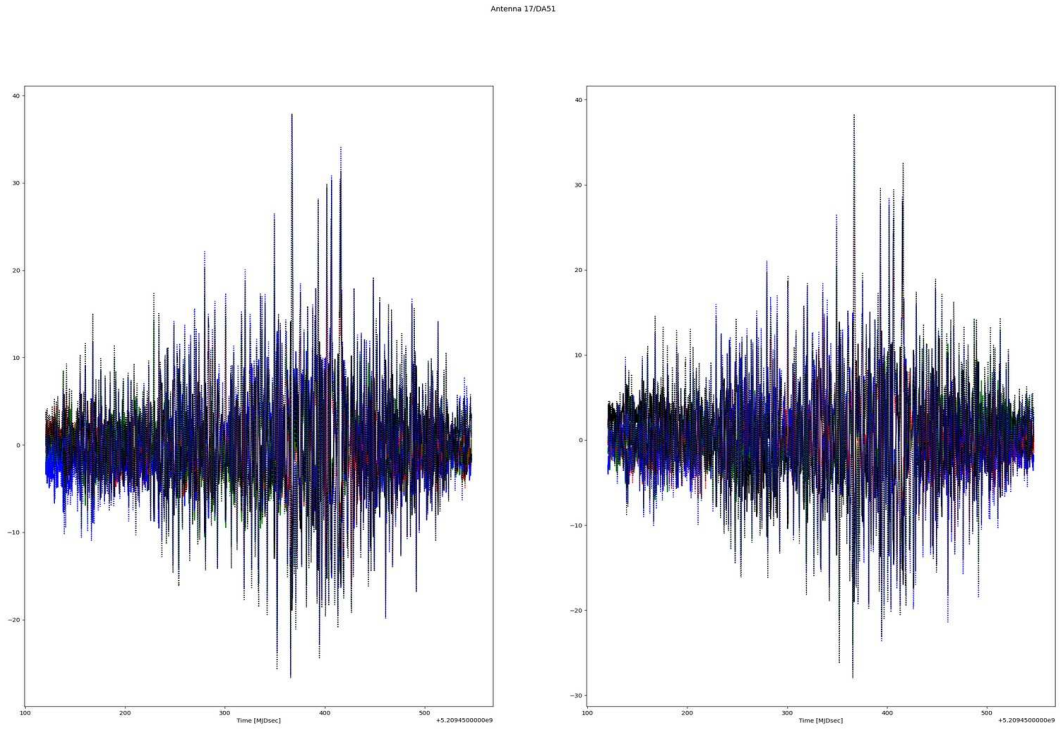


Figure 100: Residuals/errors as equivalent temperature in Kelvins for 2023 dataset and antennaDA51. Left: XX, right: YY polarizations, the individual SPWs are color-coded like in Fig. 31. For most antennas the RMS is well withon 10 K, which represents ans improvement.

We present here some selected results of the improved procedures. The full set is available online in the Repository (*Annex 5*). Actually, impact of the upgrades is significant namely what concerns the T_B(t) profiles for XX and YY polarizations: These are now very close each other and oscillate close to the array-mean T_B value. This can be seen in Fig. 99, which shows the T_B(t) for XX/YY polarizations for the solar scan of the 2023 dataset for antenna DV09 as red/blue curves in the left panel and PWV(t) time series in the right.

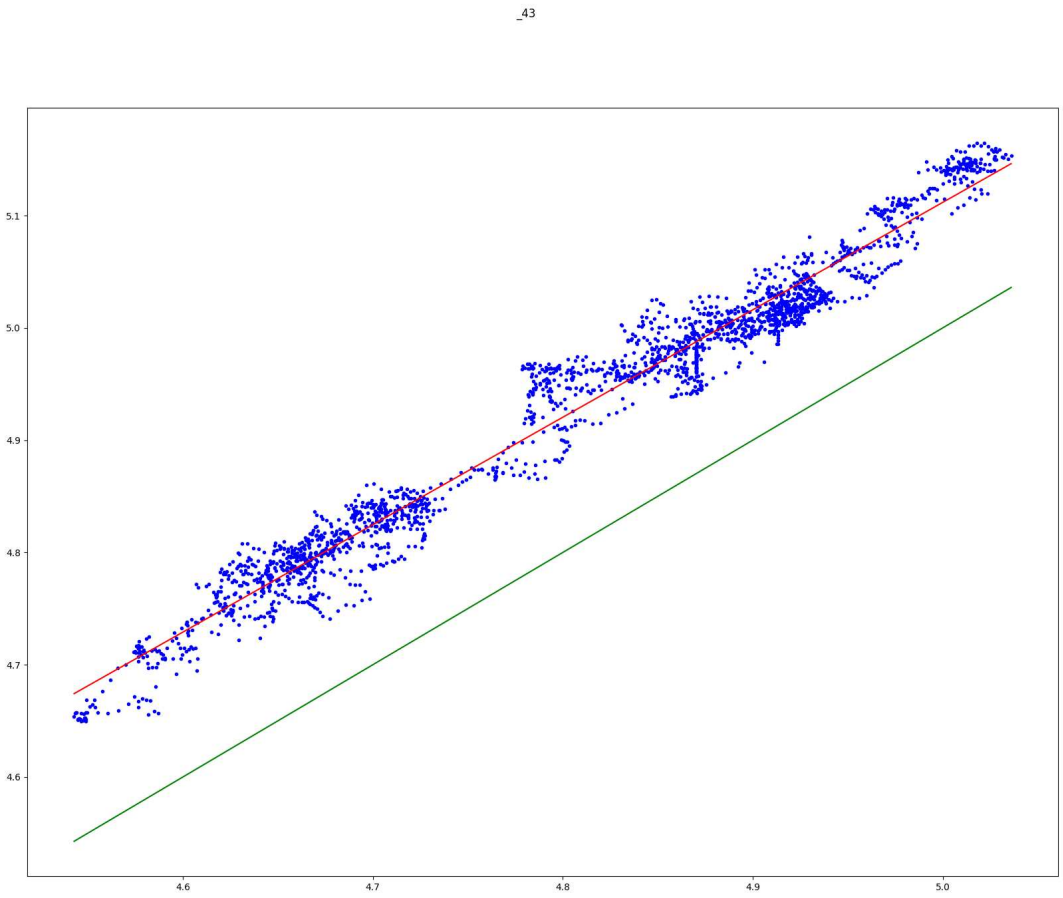


Figure 101: PWV in mm as obtained from (de-cantamined) WVR data - at x-axis, vs. the same quantity as calculated from science SPW signal (y-axis). The red line correspond to the best fit, the green one represents the ideal correlation y=x. For the antenna id 43 and scan 9 in the 2023 dataset.

Further improvement is visible also in the residuals/errors of the least-square solutions expressed in term of effective temperatures. This is shown in next Fig. 100, which may be compared with figures displaying the same quantity in the gradual improvement of the method form Section 2.3, for example with Fig. 49. The residuals significantly diminished and are typically with RMS less then 10 K.

Finally, in Fig. 101 we present cross-correlation between the PWV(t) as obtained from the science SPW signal via our upgraded *dWdBdG2()* method, and the same quantity derived from the (solar-contaminated) WVR signal and extracted (disentangled from the variable solar illumination) by also

upgraded *dWdLs4()* method. The data are for antenna id 43 (DV17) recording the scan 9 (the Sun) in the 2023 dataset. In general, the correlation between PWVs derived by those two different methods improved, most of the linear regression coefficients is in the interval 0.95 – 1.05, with small systematic offset ~0.2mm – the science SPW-based PWVs are higher by that value. Full database of cross-correlation results is available as XLS table (and also as figures similar to Fig. 101) in the online Repository (*Annex 5*).

2.5.3.4 Combined instrumental and atmospheric delays

In Section 2.5.2 we faced the problem of combination of the calibration using our PWV(t) and its recalculation to the phase delays with the ‘standard’ phase-calibrator based *gaincal()*. The point is that the phase-calibrator gaintable applies correction for (i) instrumental, polarization dependent delays together with (ii) linear trend of the atmosphere-related delays. Our *sol_pwvGcal()* routine, which is an analog to *wvrgcal()* for the solar data, creates a gaintable, that fixes only the atmosphere-based phase distortions. Now, if we do the ‘standard’ calibration, i.e. applying the PWV-precalibration first (like application of the wvrgcal() gaintable does), followed by the phase-calibrator based gaincal(), the atmospheric average delay and its linear trend are applied twice to the data. As we have already mentioned, it is not a problem in standard wvrgcal() in non-solar data, because the atmospheric delay is applied to the phase calibrator, too, before it is even used for gaincal(). Hence, this application of the atmospheric part fo the delay compensates.

With the solar data we need to do something similar. To that end, we use *dWdLs4()* for calculation of the PWV(t) series not just for the solar scan, but also (and namely! – recall, it was just the matter fo good luck that we have WVR data for the solar on-source subscan) for the surrounding scans: The ATM scan preceding the solar-target, and the leading and trailing phase-calibrator scans. Once we have PWV(t) calculated for the phase-cal scan (via *dWdLs4()* from the WVR data) and for the Sun (via *dWdBdG2()* from sci SPW signal), we may apply correction for the atmospheric part of the delay both to the phase calibrator and the Sun, exactly as *wvrgcal()* does. This extended procedure is implemented in the routine *sun_pwvGcal2()* in our collection (listings in *Annex 4*, source code as a file in online repo as *Annex 5*). We follow the standard wvrgcal()-based procedure and apply the *sun_pwvGcal2()* fix to the phase calibrator *prior* it is used for generating gaincaltable, i.e. in early steps of the calibration script, so that the applied correction gets exposed via *split(datacolumn=’corrected’).*

An example of such combined time series of PWV(t) in the scans surrounding the solar scan in 2023 dataset, as calculated with upgraded routines *dWdLs4()* and *dWdBdG2(),* are shown in Fig. 102.

To sum up, the strategy for “Solar WVR gain calibration” is as follows:

1. In the *Step 4* of the standard solar calibration script (i.e., the same place, where non-solar calibration script for manual data reduction calls *wvrgcal()*) we call *sun_pwvGcal2()*.
2. This routine prepares a standard *T*-type (i.e., gain with a single polarization channel) calibration table *sun_pwv_gcal.W.*

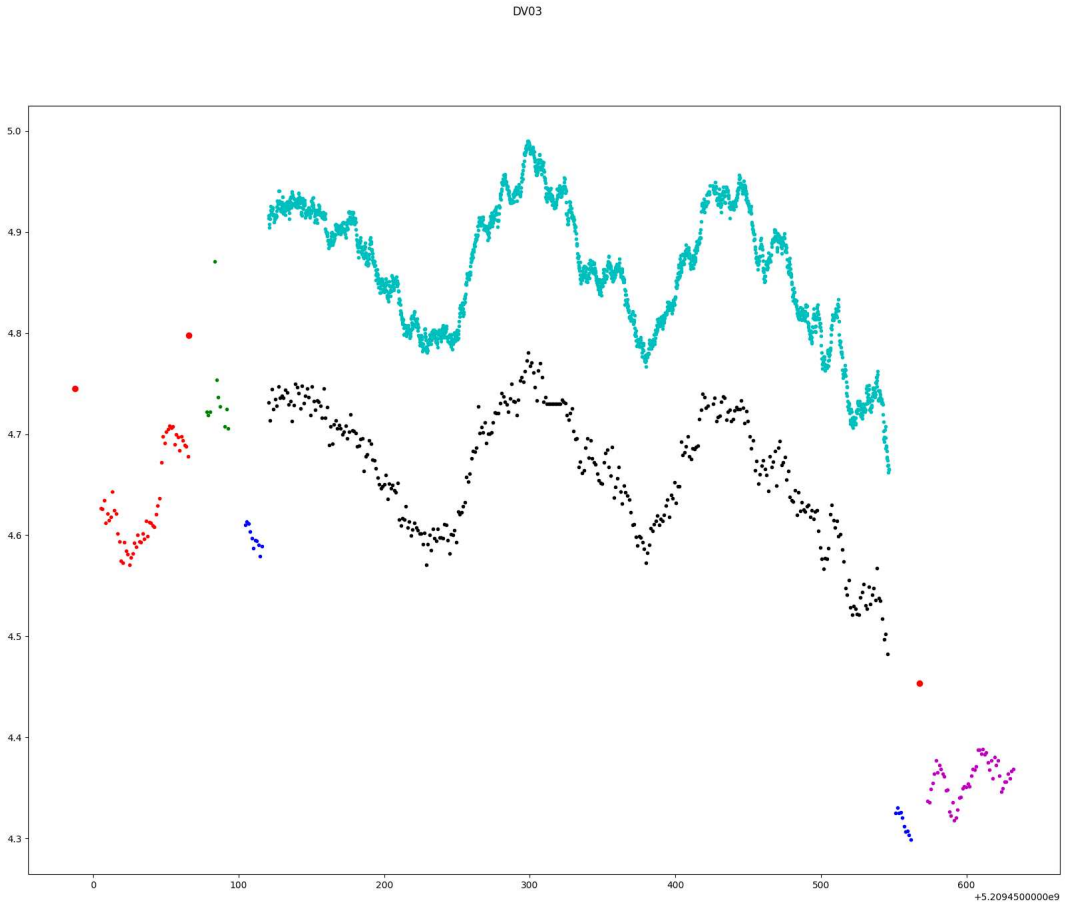


Figure 102: Calculated PWV in mm as function of time (MJDsecs) for the 2023 dataset for antenna DV03. The little dots in red, green, blue, black, blue, and magenta represent the PWV as calculated form the WVR data using latest method dWdLs4() in leading phase-cal scan, ATM scan, solar-off leading subscan, solar on-source subscans, solar-off trailing subscan, and trailing phase-cal scan, respectively. The cyan dots represent the PWV(t) as calculated from science SPWs in solar on-source subscans via upgraded dWdBdG2() method. The larger red circles are PWV values as directly read from the SYSCAL table. The same data for the new July 2026 dataset are shown in Fig. 93.

3. This *T*-type gaintable table (the same type as *wvrgcal()* produces for non-solar data) contains data – complex gains with atmosphere induced phase delays – just for two fields: The phase calibrator and the Sun.

4. The atmosphere-induced phase delays for the Sun are internally calculated by the method *dWdBdG2()* from the signal in science SPW. The atmospheric delays for the phase calibrator are internally calculated by *dWdLs4()*. [Note: The current routines names, burdened by legacy of the development process, might get renamed in the fully operation version of the software package deployed to *sun_reductionUtils()* library.]

5. The *sun_pwv_gcal.W* gain table is – together with Tsys+ Tant and Tantpos tables – applied to the phase calibrator and the Sun in the *Step 6* of solar calibration script, i.e. at the same moment, where wvrgcal()-generated gaintable is being applied in non-solar data.
6. The call to MS *split(datacolumn = 'corrected')* in *Step 8* assures that all the corrections are involved in the further calibration steps.

2.5.4 Results of applied PWV-based gain corrections

We have tested application of the corrections from the gain-table generated by *sun_pwvGcal2()* by direct comparison between movies of the data calibrated just by the standard procedure, and those corrected for the atmospheric phase variations via *sun_pwvGcal2()*. The movies for 2023 dataset are in the on-line data repository and the better quality of the phase-corrected movie is visible at first glance. Even in the case of this relatively high PWV dataset (~5mm) the correction removed most of the fast (frame-to-frame) flickering, leaving just the intrinsic dynamics of the solar atmosphere at its much slower timescale. The PWV-based correction may be compared also with the movie obtained by the *Approach 1*, i.e. standard calibration followed by time-scale decomposed self-calibration. From qualitative point of view it seems that the self-calibration, in comparison with the PWV-based correction, tends to reduce the dynamic features that are most likely related to intrinsic solar evolution.

The data from July 30th 2026 are yet under processing and will be delivered to the on-line repository ASAP.

3. Conclusions, suggestions for implementation, and outlook

Instead of concluding remarks, we would like to briefly recapitulate the main results obtained by carrying out this project/study and, based on them, formulate a set of recommendations, or rather suggestions or a “wish-list”, as we may likely lack the technical knowledge, which is necessary to asses the feasibility of implementations of our suggestions at engineering level. As we feel further potential for improvement, we finish the report with outlook for post-study development in this broader topic.

3.1 A brief recap and summary of obtained results

Before reporting the specific outcomes, let us mention the list of analytical tools implemented as software utilities or libraries (see the docs in *Annex 3*), which has been developed in order to provide answers to the questions that initiated this Study:

- The SW package for solar ALMA simulations, which, for the first time, uses realistic setup for solar observations involving
    - mixed 12m+7m arrays in all realistic configurations (up to C-8),
    - atmospheric corruption of the visibilities using two different advanced models, namely the phase-screen.
- Procedure for self-calibration of solar observations with very high cadence (0.192s) based on time-scale separation between the fast atmosphere-induced phase variations and slower intrinsic solar dynamics. Implemented as a set of scripts (implementation of *Approach 1* of this Study).
- Analytical tools and metrics for quality assessment of the self-calibration solutions (see *Annex 6*).
- A broad collection of routines for analysis of the WV emission and absorption features in the science SPW and WVR channels and their impact on the science data.
- As a by-product of the study, we have developed the first working version of *Solar Script Generator* (SSG, see the MTR in *Annex 1*) for assisted (highly automated) generation of solar calibration scripts. After its extension by further features (polarization, single-dish full-disc and regional-map) by Masumi Shimojo, the SSG has become *de facto* standard in the community of manual data reducers for solar data.

With the above tools we have come to the following main outcomes:

- Self-calibration/imaging loop that utilizes the time-scale separation between apparent (atmospheric) and intrinsic dynamics in the solar images – *Approach 1* of this study – was capable to process Band 3 data observed in C-7 configuration and Band 5 data in C-6. This boosts the insofar resolution almost five times! Successful completion has been demonstrated

on three observing datasets (one Band 3 from December 2023 and two Band 5: one from 2023 and second from July 2026).

- The advanced solar simulator indicates that this is actually the hard limit – reconstructed images for simulated observation in Band 3 in configuration C-8 have been already of poor quality even with atmospheric perturbation switched off. Unless an alternative to standard *tclean()* will be employed (e.g. BRAIN/*resolve()*), the low *uv* density in combination with rather fast time-variability of the solar images becomes the hard-stop limiting factor for the maximum usable configuration.
- We have clearly proved that atmospheric WV and its variations are clearly pronounced in the science SPW data and can be used for calculation of the atmosphere-induced phase delays. We have developed and tested advanced mathematical model (implemented as an extensive set of Python routines – *Annex 4*) for extraction of PWV(t) time profiles from the science SPWs. and These high-cadence PWV(t) time series are subsequently directly used for atmospheric phase-delays calculations – *Approach 2* of this study. The model is based on an extended optimization problem, which involves, in addition to science SPW data, also ACD calibration equations and allows thus for fine tuning of the instrumental parameters: gains and instrumental power-offsets. This methodology for better-fitted gains has potential for application in non-solar data, too.
- In case of the calibrators (BP, phase) as targets, the WV variations are seen as a weak variable emission in the science data, for the Sun as much stronger absorption signal. In spite the weak WV sci-SPW emission in the calibrator scans, not reaching much above the instrumental noise (increased T_{RX} in the solar MD mode), we were able to find striking correlation between the WV emission variations and the calibrator's phase fluctuation for most antennas.
- For the Sun the WV absorption signal is much stronger and even though we do not have the 'ground truth' for bench-marking of our calculated WV-induced phase variations (except of the 'soft' self-cal based solution), we have found strong correlation between our PWV(t) series with the PWV calculated from the direct WVR measurements. This comparison has been achieved thanks to specifically designed test observation in July 2026 (two sub-arrays) and confirmed even for the 2023 test dataset thanks to incredibly good luck (WVR pointed a bit off-the-Sun during solar scan near the limb).
- The above findings have been incorporated into routine *sun_pwvGcal2()*, which represents 1-to-1 counterpart to *wvrgcal()* for the solar datasets.
- As a sort of by-product of the PWV(t) analysis we have revealed the effect of troposheric scintillation in solar mm data: Variations of the signal (auto-correlations and SQLDs) in science SPW cannot be attributed solely to WV fluctuations but it has another, faster component most likely connected with lensing (focusing/defocusing) effects of the tropospheric bubbles/eddies. Our methodology count with this effect and the routines are designed to separate both components (i.e. varying PWV and scintillation). This finding may have broader impact out of solar ALMA observing.

3.2 Suggestions for implementation of high-resolution solar ALMA Observing mode

Based on the results summarized in the previous section, we suggest the following implementation changes that for the solar ALMA observing mode. Some of them are directly related to the new mode of high-resolution observations, but some represent general improvements that may help already in the current setup. We start with the high-resolution mode suggestions:

- Both simulations and the test data show that ALMA configuration may be extended to
  - C-7 in Band 3
  - C-6 in Band 5

  We suggest to start high-resolution solar observation in this setup.
- Simulations show that for other current or future Bands used for solar observing the maximum limit is given by the *uv* dilution. Consequently, the limiting configuration is given proportionally to the above findings for Bands 3 and 5 by relation for the longest baseline L_{max}<1500 kλ. If possible, we would recommend to prove the limiting configuration in the insofar untested Bands also by a short test dataset, before starting scientific observation.
- Both the simulations and test observation demonstrate that very high cadence is vital for time-scale separation between solar intrinsic dynamics and faster apparent changes in the image caused by atmospheric turbulence. The high cadence is anyway a longer-term demand in the community (see, e.g., previous *Solar Development Study* by Wedemeyer et al, 2020, or projects for ALMA 2040 vision). The cost is in higher data-rate, but its is, perhaps, still acceptable having in mind low spectral resolution of TDM mode. We suggest to use 0.192s cadence for high-resolution observations as default. For case of the ‘standard’ solar observations (i.e., e.g., Band 3 up to C-4 configuration) the higher cadence could be possible on PIs request.
- Simulations with phase-screen perturbation indicate that the phase-RMS on the long time scale, used traditionally as go/no-go test criterion, has a little relevance for the solar observations – the self-cal procedure is capable of reconstruction as long as high-cadence time sampling is performed. We suggest to skip/ignore this test for high-cadence solar data.
- We have, as a result of this study, a well tested procedure for deriving PWV(t) time series from science SPWs and their subsequent application to atmospheric phase delay corrections, very analogical to the standard *wvrgcal()* procedure. We suggest to have the *sun_pwvGcal()* routine (and its sub-routines), which plays the role of *wvrgcal()* in solar datasets, as a part of standardized *sun_reductionUtils.py* library. We are ready to help with integration and deployment.
- The three datasets observed for the tests in December 2023 and July 2026 have been fully calibrated and self-calibrated and contain high-quality and unique data (there is no previous ALMA solar observation, which would at least approach such a high spatial and temporal

resolution). The data are likely to be of high interest of scientific solar community and may revive general interest (now a bit decaying, due to frustration from low success rate) for using ALMA in solar research. We suggest to release them as CSV datasets (or similar open format) and we are ready to assist with actions that are necessary for that (metadata creation, documentation preparation, etc.).

During carrying out the Study, we have found few other, more general (not directly related to high-resolution mode) items, whose implementation could improve performance of solar calibration or observations:

- The standard routine *sol_ampCal2()* from *sun_reductionUtils.py* ignores the opacity factor that multiplies the term Tsys+Tant (e.g. White et al., 2017). For most of observations it is not a problem as the dry opacity is very small and, consequently, the correction approaches 1.0. For Band 5, this is, however, not correct. Our Tsys+Tant values in SPW 5 (next to the WV line center) for 2023 test data with quite a high PWV~5mm are around 3000K, i.e. clearly diminished by absorption. We suggest to include the opacity term in calculations and are ready to implement it.
- For the solar on-source sub-scans the solar mode uses higher attenuation settings. The attenuation factor between low and high attenuation is for calibration purposes derived from SQLD spws as a ratio $AF=(P_{sky}-P_o)/(P_{off}-P_o)$, where P_{sky} is the power measured in the ATM scan in the low-attenuation regime, P_{off} is the power measured in the solar off-source sub-scans (there are two – leading and trailing the silar on-source subscan) in the high-attenuation regime, and P_o is the instrumental (zero-signal) offset in the SQLD spw. This method has an issue caused by the fact, that P_{off} may vary in time as the atmospheric emission varies. Namely, in Band 5, where WV opacity/emissivity is strong, the emissivity variance corresponding to quite fast changes of PWV can have strong impact, as we have seen in our 2023 dataset. For that reason we started using just the leading off-source sub-scan, which follows immediately after the ATM scan in our study, in order to avoid larger change of atmospheric emissivity during the long (~7min) on-source subscan. Note, however, that the *sol_ampCal2()* in *sun_reductionUtils.py* uses the average of leading and trailing subscans for the calculation of the attenuation factor. As a cure, we suggest the following change of the procedure: During the ATM scan, there are present also ACD hot- and ambient-load subscans. We suggest to have the hot-load subscan repeated, the second run to be performed with the high-attenuation (solar) regime. First, the hot load has higher temperature than sky, so the determination of the attenuation factor as

  $AF=(P^{LA}_{hot}-P_o)/(P^{HA}_{hot}-P_o)$,

  where *LA* and *HA* superscripts denote the low- and high-attenuation regimes, should be more precise, second, the hot-load temperature does not vary significantly with time.
- By processing the ‘technical’ test dataset – Band 5 observations in the sub-array regime of configuration C-3 under non-observing condition (PWV 10 – 12, rain) we have found that by

applying our self-cal procedure, we are still able to extract somewhat valuable solar movie (see online *Annex 5*). Based on that experience and in the view of solar community frustrated by enormously small number of accepted solar proposals, we would like to suggest consideration of creating specific back-up observing queue for bad-weather tolerant projects. Both the solar community and ALMA observatory may benefit from such setup: Solar community would get their proposals observed, ALMA could increase the number of on-source hours per year – the bad weather conditions form the most significant reason for no-observation.

3.3. Future directions of further development

The Study is formally concluded by this report, but its results immediately initiate considerations on further improvements. We are keen on to continue work in this direction in the post-project period and contribute further to soar ALMA ObsMode improvement. We are fully aware that the results of our study – even if fully accepted – will not come into implementation until the current major WSU update is finished. Taking this fact positively, we would like to use this time between the project delivery and implementation of its results for further improvement. Here are few ideas for continuation of the work:

- The PWV(t) series calculations and their utilization for phase corrections is not limited to extended array configuration – it can be tested in any solar Band 5 data. We plan to do so in order to increase credibility of our findings by application to the larger dataset.
- Related to previous: At the two test Band 5 dataset used in our Study we have found likely systematic variations of averaged solar brightness temperatures across antennas. This effect has been ascribed to per-antenna varying antenna efficiencies and we have used it for prerapring a table of corrected antenna efficiencies by the antenna type (+ two outliers). Again, such a task is not limited to long-baseline configuration and any solar Band 5 dataset can be used for that.
- During discussions at the very end of our project we have touched an issue of possible side-band ratio effect to our calculations of PWV(t). Namely, whether the side-band ratio may influence the spectral slope of data between USB and LSB spws. The current result of consultations is that there should be none or negligible influence. Anyway, we would like to check it more in detail.
- The simulations shows that the diluted *uv* coverage, in conjunction with solar dynamics at rather fast timescale, is the blocker for pushing the bar of more extended array configurations further. Nevertheless, there may be solution in using something different than classical *tclean()*. We have started consultation with Fabriza Guglielmini (ESO) and Thorsten Ensslin (DZA) on possible utilization of BRAIN/*resolve()* cleaner for the solar data. This is going to be a larger project, but we hope to have it finished before the solar high-resolution mode is approved and deployed.

References

[1] Alissandrakis, C.E. , A. Nindos,T. S. Bastian , and S. Patsourakos (2020): Modeling the quiet Sun cell and network emission with ALMA, A&A 640, A57

[2] Bárta, M. (2019): *Towards high-resolution solar ALMA images*, ALMA Development Workshop, held 3-5 June, 2019 at ESO Garching, http://adsabs.harvard.edu/abs/2019adw..confE...4B

[3] Bastian, T. S. (2002): ALMA and the Sun, Astronomische Nachrichten 323, p. 271

[4] Bastian, T. S., Bárta, M., Brajša, R., Chen, B., Pontieu, B. D., Gary, D. E., Fleishman, G. D., Hales, A. S., Iwai, K., Hudson, H., Kim, S., Kobelski, A., Loukitcheva, M., Shimojo, M., Skokić, I., Wedemeyer, S., White, S. M., Yan, Y. (2018): *Exploring the Sun with ALMA*, The Messenger 171, p. 25

[5] Bastian, T. S. (2019): Solar ALMA Team communication.

[6] Benz, A.O.– ALMA DRSP (2006): in ALMA Design Reference Science Plan https://home.strw.leidenuniv.nl/~alma/review/drsp20_review.shtml

[7] Brajša, R., Skokić, I., Bárta, M., Bastian, T.S., Shimojo, M., White, S., Iwai, K. (2017): *Solar Research with ALMA – ESO Development Study,* Final Report, ESO

[8] Brajša, R., Sudar, D., Benz, A. O., Skokić, I., Bárta, M., De Pontieu, B., Kim, S., Kobelski, A., Kuhar, M., Shimojo, M., Wedemeyer, S., White, S., Yagoubov, P., Yan, Y. (2018): First analysis of solar structures in 1.21 mm full-disc ALMA image of the Sun, Astronomy and Astrophysics 613, p. A17

[9] ESO (2017): *ALMA starts observing the Sun*, ESO Press Release no. 1703, https://www.eso.org/public/news/eso1703/

[10] Fomalont, E. B. et al. – ALMA Partnership (2015): *The 2014 ALMA Long Baseline Campaign: An Overview,* The Astrophysical Journal 808, p. L1

[11] Heinzel, P., Berlicki, A., Bárta, M., Karlický, M., Rudawy, P. (2015): *On the Visibility of Prominence Fine Structures at Radio Millimeter Wavelengths*, Solar Physics 290, p. 1981

[12] Hills, R. (2019): Solar ALMA Team communication.

[13] Hudson, H. (2019): Solar ALMA Team communication.

[14] Karlický, M., Bárta, M., Dąbrowski, B. P., Heinzel, P. (2011): *Solar Research with ALMA*, Solar Physics 268, p. 165

[15] Koval, A., Yao Chen, Takuya Tsugawa, Yuichi Otsuka, Atsuki Shinbori, Michi Nishioka, Anatoliy Brazhenko, Aleksander Stanislavsky, Aleksander Konovalenko, Qing-He Zhang (2019): Direct Observations of Traveling Ionospheric Disturbances as Focusers of Solar Radiation: Spectral Caustics, The Astrophysical Journal 877, vol. 2.

[16] Loukitcheva, M. A., Solanki, S. K., White, S. (2008): ALMA as the ideal probe of the solar chromosphere, Astrophysics and Space Science 313, p. 197

[17] Maud, L. T., Fomalont, E. B, Tilanus, R. P. J., Dent, B. (2017a): *ALMA & Allegro Phase Metrics Workshop Report*, ALMA Memo No. 606 https://library.nrao.edu/public/memos/alma/main/memo606.pdf

[18] Maud, L. T., Tilanus, R. P. J., van Kempen, T. A., Hogerheijde, M. R., Schmalzl, M., Yoon, I., Contreras, Y., Toribio, M. C., Asaki, Y., Dent, W. R. F., Fomalont, E., Matsushita, S. (2017b): *Phase correction for ALMA. Investigating water vapour radiometer scaling: The long-baseline science verification data case study*, Astronomy and Astrophysics 605, p. A121

[19] Maud L. T., Andrés F. Pérez-Sánchez, Yoshiharu Asaki, Felix Stoehr, Bill Dent, and María Díaz Trigo (2023): ALMA Memo 624, *Updates to ALMA Site Properties: using the ESO-Allegro Phase RMS database*. https://library.nrao.edu/public/memos/alma/main/memo624.pdf

[20] Otarola, A.; De Breuck, C.; Travouillon, T.; Matsushita, S.; Nyman, L.-Å.; Wootten, A.; Radford, S. J. E.; Sarazin, M.; Kerber, F.; Pérez-Beaupuits, J. P. (2019): *Precipitable Water Vapor, Temperature, and Wind Statistics At Sites Suitable for mm and Submm Wavelength Astronomy in Northern Chile*, PASJ 131, pp. 045001.

[21] Pardo, J.R, José Cernicharo, and Eugene Serabyn (2001): Atmospheric Transmission at Microwaves (ATM): An Improved Model for Millimeter/Submillimeter Applications, IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, VOL. 49, NO. 12

[22] Pety et al, 2001, ALMA Memo Series 398

[23] Richards, A.M.S., E. Moravec, S. Etoka , E. B. Fomalont, A. F. Perez-Sanchez , M. C. Toribio and R. A. Laing(2022): ALMA Memo 620, *Self-calibration and improving image fidelity for ALMA and other radio interferometers.* https://library.nrao.edu/public/memos/alma/main/memo620.pdf

[24] Shimojo, M. et al.. (2017): *Observing the Sun with the Atacama Large Millimeter/submillimeter Array (ALMA): High-Resolution Interferometric Imaging*, Solar Physics 292, p. 87

[25] Shimojo, M. (2019): Solar ALMA Team communication.

[26] Wedemeyer, S. et al.. (2016): *Solar Science with the Atacama Large Millimeter/Submillimeter Array—A New View of Our Sun*, Space Science Reviews 200, p. 1

[27] Wedemeyer, S., Bastian, T.S., Shimojo, M., and Bárta, M. (eds.) (2020): *Proc. of 1st International Solar ALMA Imaging Workshop in Oslo*, March 2-6, 2020 (in preparation as ALMA Memo)

[28] White, S. M. et al. (2017): *Observing the Sun with the Atacama Large Millimeter/submillimeter Array (ALMA): Fast-Scan Single-Dish Mapping*, Solar Physics 292, p. 88

Annex 1

Complete Mid-term report, including all its annexes

ESO ALMA Development Study

HIGH-RESOLUTION SOLAR OBSERVATIONS WITH ALMA

Mid-term report

Miroslav Bárta & Yi Chai

EU ARC – Czech node

Astronomical Institute of the Academy of Sciences

Rev. 3, July, 2024

Table of Contents

1. Introduction – brief summary of the project goal....3
2. Formal basis....6
2.1 Related documents....6
2.2 List of abbreviations....6
3. So-far project timeline, coordination, and monitoring....7
3.1 Progress meetings/reports....7
3.2 Informal consultations/workshops....7
4. Deliverables for the milestone M1....8
4.1 Setup of the CASA simulations for larger baselines (DS1.1.1)....8
4.1.1. Input (unbiased) model....9
4.1.2. Generating the *simulated* MS and its subsequent corruption by the atmospheric turbulence....11
4.1.3. Calibration, imaging, self-calibration....16
4.1.4. Results....17
4.2 Comparison of the selfCal-corrected results with the original unbiased model images – quality assessment (DS1.1.2)....19
4.2.1 Image fidelity....19
4.2.2 Self-cal-corrected vs. uncorrected phase RMS....19
4.2.3 Scale analysis: Fourier (power) spectra of the input-model, simulated clean image, and its residual....20
4.2.4 Results....20
4.3 Solar specific Go/noGo criterion (DS1.1.3)....25
4.4 Analytical relation between *absorption* in the water vapour and the phase delay in Band 5 (DS1.2.1)....28
5. Verification of the project approaches at the CSV data....28
5.1 Verification of self-cal corrections for short-integration observations....29
5.1.1. Observation description....29
5.1.2. Calibration, basic imaging....29
5.1.3. Self-calibration / imaging loop....30
5.2 Replacement of the WVR signal by differential absorption of the solar mm radiation in Band 5: Indication of feasibility....35
6. Development side product: Solar Script Generator....36
7. Conclusions and plans for the next period....38
References....41

Annex 1: Statement of Work (SoW)....43
Annex 2: Proposal – Scientific & technical description....55
Annex 3: WV line sensitivity calculations by Richard Hills....68
Annex 4: Code listings 71
Annex 5: Observation reports 85

1. Introduction – brief summary of the project goal

High spatial resolution for observations performed with ALMA has been one of the big promises for the community of astronomers and astrophysicist in general, and the experts in solar research represent no exception in this respect. While the colleagues studying the 'standard', cold ALMA targets can benefit from this supreme quality – unprecedented at the mm/sub-mm wavelengths and far better than provided by any instrument in the pre-ALMA era – the solar observations have remained quite constrained in this respect. Spatial resolution in the solar ALMA ObsMode has never exceeded ~0.6 arcsec, given the limits for available (as confirmed by the earlier CSV projects performed in 2014 – 2017) array configurations: C-4 for Band 3, C-3 for Bands 5 and 6, and C-2 for Band 7.

ALMA observations with higher spatial resolution have potential to answer many key and so far open questions in solar physics – among them, e.g., the issue of the heating of the upper layers of the solar atmosphere (the well known problem of *coronal heating*, which is important for stellar physics in general, not just for the Sun) – the dissipative shock waves that – according to conjectures based on the (R)MHD modelling – provide the key process for the heating, have definitely sub-arcsecond dimensions. Similarly, the expected filamentary (multi-thread) nature of solar prominences, could be revealed (or disproved) only by observations at higher resolution. ALMA is very specific in this respect as the brightness temperature at mm/sub-mm wavelengths can be much more directly converted into kinetic temperature of the solar plasma than it is possible at other wavelengths, where difficult and very uncertain non-LTE inversions have to be employed. Motivation in the solar ALMA community for higher spatial resolution, is therefore, indeed, very high.

The main goal of this project is to meet the wishes of the community and overcome the current limitations and, in this sense, *'long-baseline'* or *'high-resolution'* terms, used within this report, refer to anything significantly better than the so far posed constraints. **As it can be seen further, we are speaking about extending the array baselines to configurations, say, C-6/C-7** (the actual new limits are being subject of the main final deliverable of this study), which is not – in general ALMA terminology – really long-baseline, but for the solar observations it represents a significant leap ahead towards high spatial resolution. This is – as it has been already written – one of the very desired quality wished by the community. In addition to meeting this wish, there is an additional benefit in the increased flexibility in scheduling of solar observations: Frequently an interval of plausible array configurations is requested in the ALMA OT proposal and broadening the interval of configurations suitable for the solar observations would relax form necessity to schedule them into confined and rather narrow periods of year.

There are basically two main reasons, which limit the array extension for observations of the Sun:

(1) Inability to correct short phase variations (between two phase-cal scans) by means of WVR signal. This is the standard calibration step for the cold targets, which is not working for the Sun, whose brightness temperature exceeds that of the atmospheric WV line more than by order of magnitude and leads to the WVR saturation. The fast phase variations related to the

tropospheric turbulence lead to the *de-coherence* of the signal, if its integration time exceeds the typical time-scale for phase variation and if the fast-varying phase errors cannot be corrected. For most of the science targets, one actually needs a reasonably long integration time in order to get a significant signal. On the other hand, for the Sun – thanks to it high brightness – just very short snapshots (actually their high-cadence sequence) are sufficient for reaching comfortably high SNR. With the short-integration (high-cadence) observations, the atmosphere can be considered as frozen-in. Consequently, the signal stays *coherent* (at least if we ignore the time-varying deformation of the wavefront at the scales smaller than the single antenna diameter), just the visibilities are loaded by a time-dependent phase errors. These could be, potentially, corrected by self-calibration procedure. This simple observation also makes a basis for the main methodological approach of this study: Self-calibration of the series of short-int snapshots of the solar target.

(2) Second issue is related to the structure of the solar images, namely their broad span of spatial scales leading to the wide continuum in the Fourier space. Since the smallest expected scale in the solar atmosphere (the physically determined dissipation scale) is still many orders of magnitude smaller than what we can expect to resolve in our images, each extension of spatial resolution (i.e., extension of the array configuration) leads to further extension (widening) of the Fourier spatial spectrum. Consequently, its coverage by the actual visibilities in the *uv* domain gets diluted, in other words, the ratio of the 'information contents' present in our observed data (represented by the count of instantaneous visibilities) to the same quantity for the reconstructed image (number of grids in the *uv*-space with non-zero signal), drops down. One can, thus, expect a limit in extending the array configuration even for the idealized case of the observations without influence of the Earth's atmosphere. The effect of sidelobes in the synthesised beam, becoming more pressing with the diluted *uv* space, has, moreover, an influence on the actual SNR at the smallest scales, as it increases uncertainty of the brightness temperature in the small pixels of the highly-resolved image. And, clearly, even more this issue may get significant in the real case of the visibilities loaded by the errors implying from the tropospheric turbulence.

To sum up, the so-far imposed quite strict limits for array configurations (see above) do not count with the possibility to correct a sequence of distorted images (each short-int frame being captured, however, with the 'frozen' Earth's atmosphere) *posteriori*, by self-calibration, as described in the Item (1) above. The study aims at pushing the limits of the maximum baseline length as much as possible, nevertheless, with respect to the possible limitations described in the Item (2). Clearly, the new limiting baseline lengths will depend – in addition to the observing frequency – also on the tropospheric conditions. These are normally characterized by the phase variations of the band-pass calibrator, measured at its standard time-scale during the *Go/noGo* decision procedure (usually from the band-pass scan of the preceding EB observation). This is schematically illustrated in the Fig. 1, which is not yet in its final version (this is, basically, contents of the pre-final milestone M4) but the displayed lines are, to a large extent, resulting from the so far performed simulations (See Section 4.1).

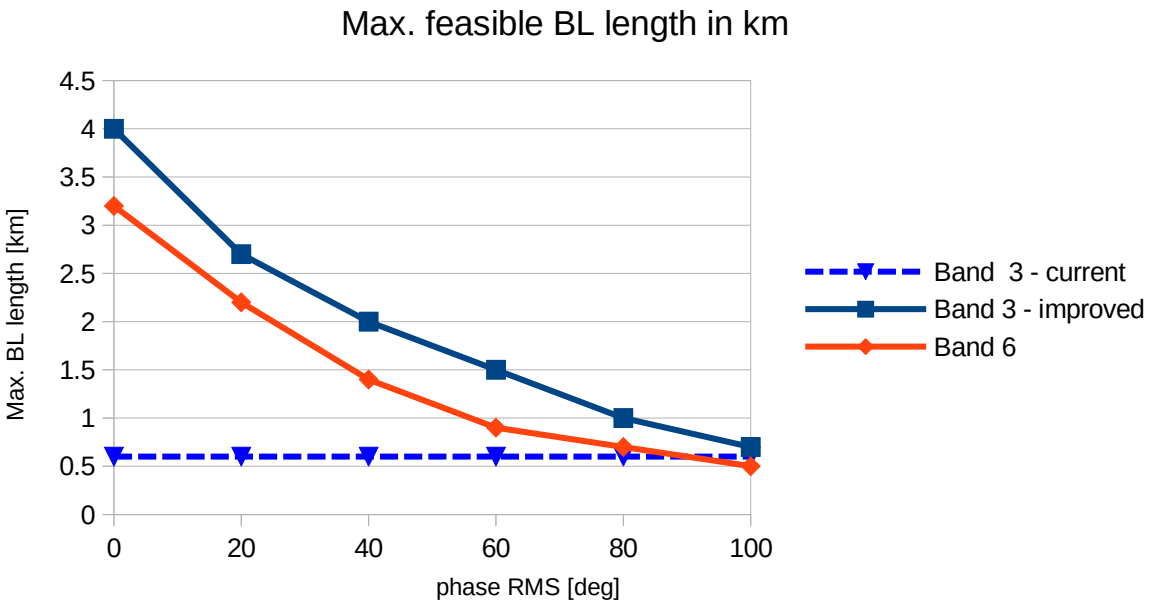


*Figure 1: Schematic illustration of the main goal of this study: For each observing band and the phase RMS as measured by the **standard** Go/noGo procedure (at the BP scan of the previous execution) find the maximum extension of the ALMA configuration still usable for solar single-pointing interferometry. The figure represents an illustrative graphical expression for the **solar** Go/noGo - the points below the line are 'Go', the lines represent the marginal conditions.*

In another turn, the Figure 1 represents (so far just an illustration of) the revised *Go/noGo* criterion specifically for the solar single-pointing interferometric observations – the lines being the marginal cases. Thus, exactly such a figure (when finished) represents the main ultimate goal of this study.

When mentioning the inability to use WVR for the correction of rapidly varying phase errors in solar observations, we should mention also the idea of Richard Hills, who suggested to use differential (i.e. per spectral channel in the TDM mode) absorption of the solar mm radiation in the wing of the WV line as replacement. This approach may be working in the Band 5 observations and it is thus less general. Nevertheless, it can further improve the situation in Band 5 data and this study pays attention to exploration of this possibility, too. In the SoW for this study and within this report this supplementary methodology is referred to as *Approach 2*, while the general-use method based on the posteriori self-cal correction of time series of the short-integration solar images is called *Approach 1*.

More details about motivation, basic ideas, and methodological approaches to this development study can be found in the *Scientific & technical description* of the original proposal to the study, which is attached as *Annex 2*.

2. Formal basis

The research is performed under the *Collaboration agreement no. 102502/19/114084/HNE* between ESO and the Astronomical Institute of the Czech Academy of Sciences, which is the hosting institution of the Czech node of the EU ARC.

2.1 Related documents

- Statement of Work for this study (SoW) – attached as *Annex 1*.
- Project proposal: Scientific & technical description (rev. 2020) – attached as *Annex 2*.
- Richard Hill's calculations of sensitivity of the phase delays by the WV in the ray-path vs. differential spectral absorption – *Annex 3*.
- Listings of the simulation, calibration and imaging scripts – attached as *Annex 4*.
- Observation shift-log report, `listobs()` output, and the QA2 report – attached as *Annex 5*.
- Mid-term review plan containing the revised/clarified deliverables as agreed during the Kick-off meeting in September 2022 – sent as an invitation to the panelists and observers
- General parent JIRA ticket for the project: https://jira.alma.cl/browse/CSV-3831
- Child-thread for the CSV observations: https://jira.alma.cl/browse/CSV-3832

Further resources are available through the references/citations or as the hypertext links directly in the text.

2.2 List of abbreviations

The standard and notorious abbreviations used in the ALMA documents are exploited also in this report, without specific definition. Here we define just these less usual.

ASU	Astronomical Institute (of the Czech Academy of Sciences)
BP	Band-pass (calibrator, scan)
Cy *N*	(ALMA observing) Cycle no. *N*
EB	Execution block
INT	Interferometry/interferometric
FoV	Field of view
KOM	Kick-off Meeting
MTR	Mid-term report / Mid-term review
PB	Primary beam

PM/PR	Progress meeting / progress report
PWV	Precipitable water vapour (a measure of, in mm)
QA2	Quality Assurance Level 2
(R)MHD	(Radiative) Magneto-Hydro Dynamics
SOT	Solar Optical Telescope (onboard Hinode satellite)
SoW	Statement of Work
SP	Single-pointing
SSG	Solar Script Generator
WV/WVR	(Earth atmospheric) water vapour / Water-vapour radiometer.

3. So-far project timeline, coordination, and monitoring

The project has been officially started by the Kick-off meeting (KOM) on September 29, 2022. Its execution is regularly monitored and advised by the Project advisory board: Eric Villard, Carlos de Breuck, and Pavel Yagoubov.

3.1 Progress meetings/reports

After the KOM, we have held so far six Progress meetings (PM) connected with reporting and advising for the further activities. The PMs are held online and usually last around 2 hours in order to find enough time for discussion. The progress reports (PR) usually in two versions (pre- and post-meeting) are loaded in the ESO PDM system.

3.2 Informal consultations/workshops

In addition to the formal PMs we have held several further informal shorter consultations – offline (by e-mail), on-line and even face-to-face: With Eric Villard and Luke Maud we have had an opportunity to discus specific issues of the project during the EU ARC All-hand meeting in Bologna in October 2023.

Moreover, in December 2023 we (the team at ASU and the Project advisory board) had a brief on-line workshop with the team at University of Oslo (UiO), which has been performing preceding solar ALMA Development Study High cadence imaging of the Sun. The goal of the workshop was to avoid useless duplicate research (namely in simulations) and to compare the obtained results in the areas of overlap between both projects. The outcomes of the meeting are recorded in our PR no. 5.

4. Deliverables for the milestone M1

According to revised SoW (described in *Annex 3*), the milestone M1 in the mid-term of the project consists of four deliverables/activities: DS1.1.1, DS1.1.2, D.1.1.3 (related to the main *Approach 1* based on self-calibration of the sequences of short-integration snapshots), and DS1.2.1 (related to the specific treatment of the WV absorption signal in Band 5 – in the SoW referred as *Approach 2*). These are now presented in the following paragraphs 4.1 – 4.4.

4.1 Setup of the CASA simulations for larger baselines (DS1.1.1)

Simulations can be seen as to some extent as an initial task, whose purpose is to provide an apriori estimated map in the parameter space (maximum baseline length, phase RMS, observing frequency): To draw a border line between setups, for which the extended configuration still should be feasible, and those, where we cannot expect good results – basically making a apriori version of Fig. 1. Without this preliminary knowledge it would be difficult to request observing time with just a randomly picked array configuration.

Since this task in many respects represents an extension of the work done by the preceding ALMA Development Study High cadence imaging of the Sun, we start with briefly summarizing the main differences in the approach/setup and the progress done beyond the previous study:

- **Extended configurations**. Clearly, contrary to the preceding Development study, we have focused on the so far unexplored configurations: C5 – C8 in Band 3, C4 – C7 in Band 5, and C3 – C7 in Band 6.
- **Heterogeneous arrays**. Because the instantaneous *uv* coverage is essential for successful imaging of the Sun with extended configurations (extremely broad range of spatial scales in the longer-baseline solar image in conjunction with the fast internal dynamics of the solar atmosphere demands it), we have used very consequently CASA **`simulator`** *tool*, avoiding the **`simalma()`** *task* entirely. This enables us to simulate mixed 12m + 7m (a.k.a. "kitchen-sink") array, which is actually used for the solar observations, and whose heterogeneous baselines have been ignored in the previous simulations. The mixed array (ACA + 12m) has been used in all our simulated array configurations.
- **Atmospheric perturbation/corruption model**. Realistic model of the atmospheric turbulence, which influences namely the phase errors, is essential, too. In this study we do not use **`sm.settrop()`** and **`sm.setnoise()`** but rather a Python based **`ATMOS.py`** model (Ed Fomalont 2021, by courtesy of Eric Villard). This model produces mangled gain calibration table, with the phase delays corresponding to the atmospheric model controlled by a (low-dimensional) set of free parameters. Subsequent call to **`applycal()`** projects the perturbations into the simulated measurement set. Further enhancement is envisaged, using the improved atmospheric model by Yoshiharu Asaki & Luke Maud (2023) in the near future (see Section 7).

- **Input idealized model**. Unlike the previous study, which uses results of the numerical simulations as an input model image, we use real solar images in white light from the Hinode/SOT.

The individual parts of the setup will be discussed more in detail in the subsequent paragraphs.

4.1.1. Input (unbiased) model

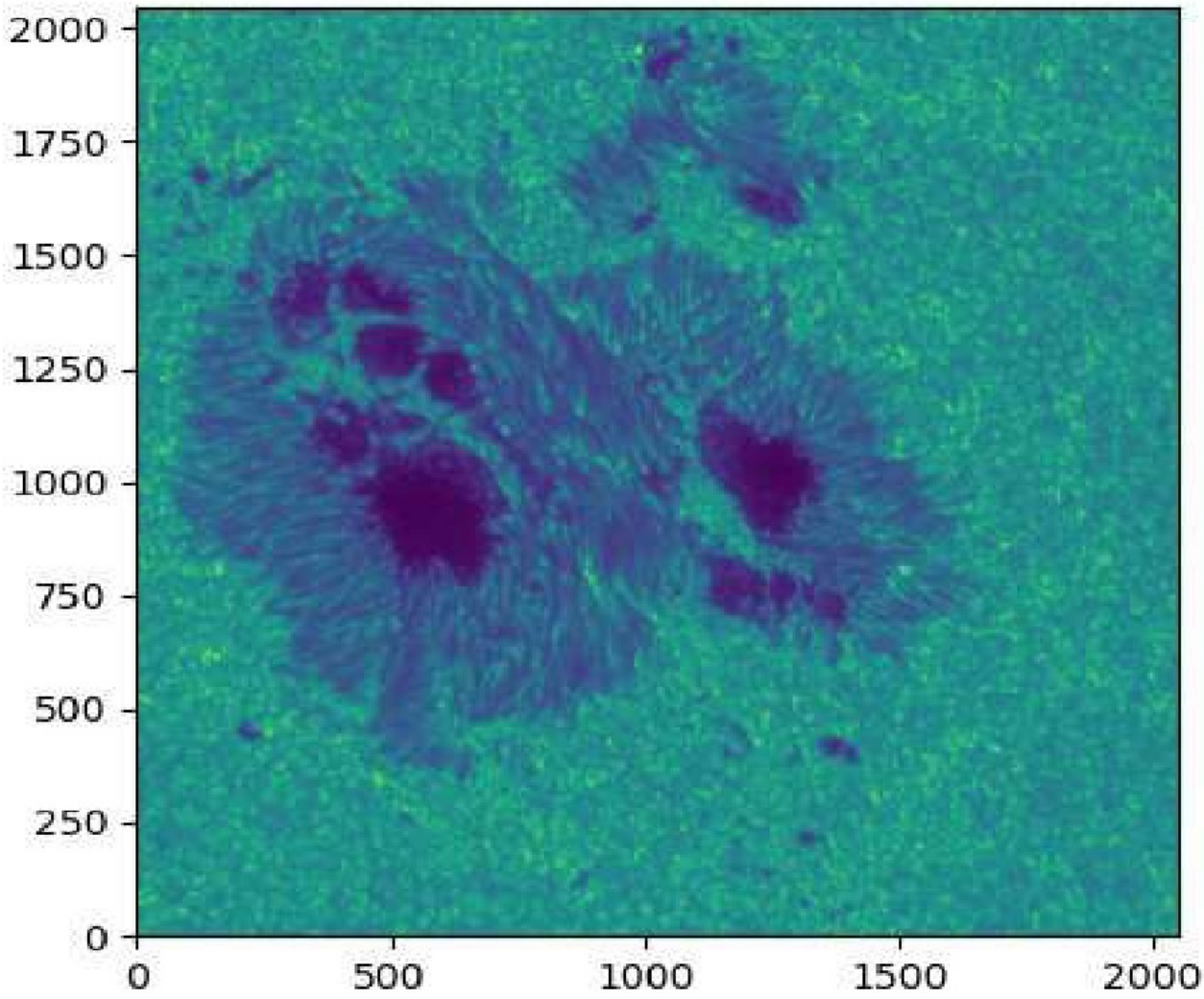


Figure 2: Sunspot image from the SOT/Broadband Filter Imager (a continuum centered around wavelength of 430 nm).

We are using real images of the Sun in white-light as an input model. The images has been taken by the Solar Optical Telescope (SOT) onboard the Hinode satellite. The main advantage to the inputs from the (R)MHD modeling (as used in the previous study) is the sufficiently broad range of scales covered by the image: The span between its resolution (about 0.05 arsec) and FoV (109 arcsec) covers the expected

range of scales even for the pretty extended ALMA configuration (C7 in Band 3). For the yet longer baselines (C-8 and beyond) we had to artificially down-scale the image by a factor of ~2 in order to reach the resolution (ALMA synthesized beam size) and, consequently 'extend' the FoV of Hinode/SOT using mirroring/flipping the image along its boundaries so that it is larger than that of the ALMA 12m antenna primary beam. As a result, the natural range of scales present in the solar atmosphere (albeit shifted towards the higher spatial frequencies for the extra-long baselines) is preserved. Note that the span of the scales is independent on the observing frequency – with increasing frequency both its high-end (the resolution given by the synthesized beam size) and low-end (the FoV determined by the primary-beam of the 12m antenna) just shifts to higher spatial frequencies, but the span (the ratio between low and high ends) remains constant. To that end, we can use the same model image for various frequency bands, rescale its pixel size such a way that it fits the ALMA resolution and adapt the FoV accordingly. On the other hand, with increasing length of the longest baseline the ratio between the smallest and largest scale increases and the model image need some manipulation – for more compact configuration it can be cropped, for more extended such as C-8 and beyond, it has to be extended as described above.

In contrast, the MHD-based input model images do not have their size sufficient to cover entire ALMA FoV and have to be 'tiled' to a bigger image (which is possible thanks to the periodic boundary conditions/BC used in the underlying RMHD model). Such a technique increases the size of the image, on the other hand, the Fourier domain of such a tiled-up image is not broadened. Since one of the main issues/risks of the current project is connected with the question, whether we will be able to reconstruct the image from the diluted (as a consequence of extended configuration) visibilities in the *uv* space, we consider the natural span of the scales as present in our input model based on the high-resolution optical observation, to be a significant feature for confidence of the simulations.

Honestly, there are disadvantages of our approach, too: The structures in the photosphere (as observed by the SOT) need not to correspond to those in the higher layers of the solar atmosphere, which are visible namely in Band 3. Also, the brightness in the optical image is not so closely related to the brightness temperature at mm wavelengths. And last but not least – the initial setup uses a static image (albeit we are going to use high-cadence SOT movies in the next period – see Section 7) so for now we have ignored the quick *real* small-scale features (e.g., thin, moving shock-fronts) and focused on the *apparent* fast changes in the image introduced by the turbulence in the Earth's atmosphere. Nevertheless, we still believe that the structure function of the *solar* atmosphere is represented quite truly by the input model image, and – above all – in its full span of scales that can be visible by ALMA.

Figure 2 shows the sunspot image captured by the *Broadband Filter Imager* of the SOT. The original image was cropped to 2048 px by 2048 px with the Sunspot at the center. The image size corresponds to the expected range of scales (cell-size – FoV) for a single-pointing INT image taken in the configuration C-7. As it has been written above, the image has been extended by its mirror/flipped copy for the yet longer-baselines configurations (C-8). The intensity map has been taken as is – i.e. in the arbitrary units, without any transformation into brightness temperature. Such a transformation would just introduce linear scaling of the values by some constant factor, which does not influence the results. We have used this high-resolution and rather broad FoV image in the visible light (centered around wavelength of 430 nm) namely for the above described nice property – that it provides a natural range of scales for the structures in the solar atmosphere. It may be objected, on the other hand, that it captures a bit different layer of the solar atmosphere – ALMA is sensitive rather to the chromosphere (Band 3) or the layers around the temperature minimum (Band 6; see, e.g., [14] or [20]). More relevant comparison is possible with, e.g., AIA image at 30.4 nm – what we are actually doing with the test

observation data (see Section 5.1.3) in Figure 22 – as the observation at this wavelength captures similar layer of the solar atmosphere as ALMA in Band 3. Nevertheless, while the SOT optical images are available in the high resolution and the wide field enough to provide sufficient span of scales for simulations, the rather low-resolution AIA images lack this property. However, the requirement for the scale coverage is, obviously, an essential one for the image that serves as the simulation input model, and thus it determined our selection.

4.1.2. Generating the simulated MS and its subsequent corruption by the atmospheric turbulence

In order to set up the simulation sequence, we tried the CASA default simulation tasks such as '***simobserve***', '***simanalyze***', and '***simalma***' first. However, due to the complexity of solar observation, the standard tasks is not suitable for appropriate simulation procedure that as much as possible mimics the actual solar observations. Namely, the combined 12m+7m array cannot be reached with the standard ***simalma()*** setup.. Therefore, we turned to the CASA ***simulator tool*** in order to take the opportunity of more free parameters when creating a measurement set from scratch.

The workflow of the whole simulation sequence can be briefly described as follows:

a) Preparing the image which will be the input model for the simulation.
b) Using CASA simulator tools to simulate the ALMA solar observation with a combined array of both 12-m and 7-m antennas and generate the measurement set.
c) Applying Phase error to the measurement set created in step b) to simulate the atmospheric disturbance during observation.
d) Performing a self-calibration routine to the corrupted measurement set, and compare it with the original model image. In the self-cal loop the *solint* interval is consecutively diminished.

As mentioned in Introduction (Issue no. 2 related to the diluted *uv* coverage), one can expect an existence of the very upper limit for the longest baseline, beyond which the image synthesis (using the traditional deconvolution algorithms) is no more possible because in the solar images we have (i) broad range of scales that effectively continuously fill entire Fourier space and (ii) sparse instantaneous sampling at the same moment. The large fraction of non-sampled flux from the extended source like the Sun clearly represents a limiting factor for convergence of imaging, at least with the standard CLEAN algorithm including its sub-variants like *mtmfs-clean*. In order to find this very limit independent on the tropospheric conditions for each observing Band, we have started the preliminary analysis with simulations without the Step (c), i.e.,with no phase errors due to the Earth atmospheric turbulence, letting just the issue of diluted *uv* space to act.

For the sake of stressing this issue yet a bit more, we have performed the simulations both with mixed 12m+7m array (what is the actual case in solar ALMA observations) and the 12m array only. This experiment has been chosen in order to decide, whether the advantage of more dense *uv* space in the combined array eventually prevails over the complications that are connected with heterogeneous-array imaging (e.g. mosaic gridding even for the single-pointing images or the uneasy primary-beam correction).

The results are displayed in Figure 3 and Figure 4. Because the simulated brightness temperature is in arbitrary units, we plot all images (T_B maps) as normalized into the same range 0.0 – 1.0 .For the C-8

configuration we had to expand the model image in line with diminishing its native pixel size in order to match the resolution and FoV for ALMA Band 3 in C-8. The C-7 configuration does not need this model-image manipulation (see the troposphere-perturbed case in Figure 5) but in this case we kept it for the sake of direct comparison of the two images. The parameters for ***tclean()*** are – except of the cell-size and image-size – the same as used through the entire study and can be read in the code listings in the *Annex 4: The scripts*.

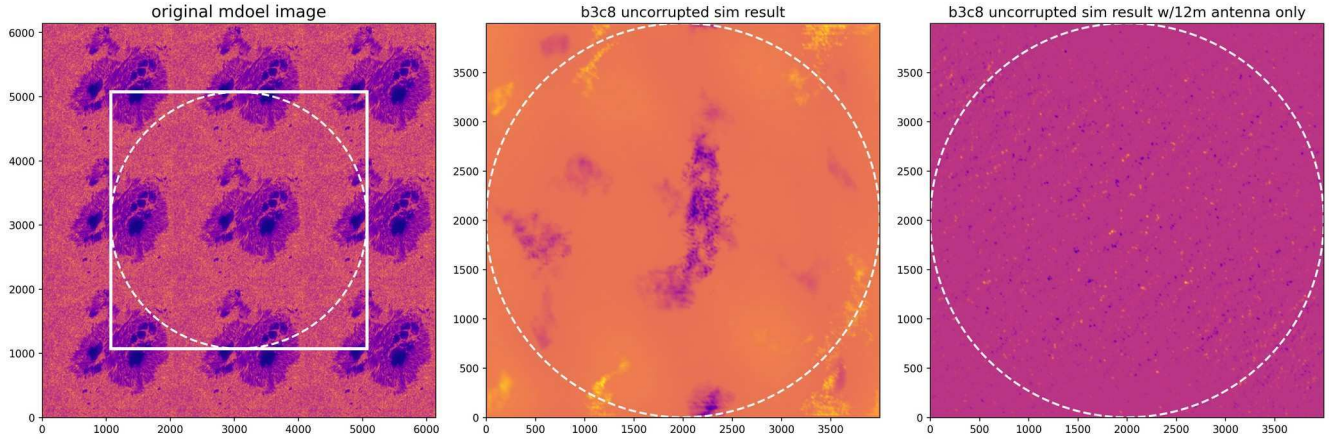


Figure 3: Idealised (no troposphere) simulation of Band 3 in configuration C-8. Left: Tho model image expanded in order to fit the ALMA FoV, middle: Deconvolved simulated ALMA image with combined 12m + 7m array, right: deconvolved simulated ALMA image with 12m array only. The circle in the box shows FoV of a 12m ALMA antenna for Band 3, which is ~1 arcmin. One pixel at the x and y axes thus corresponds ~0.016 arcsec (which is 6x oversampling the synthesized-beam size of 0.096 arcsec in configuration C-8). Maximum recoverable scale for the solely 12m array (right panel) is ~1.4 arcsec, corresponding to 85 pixels. The color scale is in units normalized to the interval 0.0 – 1.0 and the same for all panels – lighter color corresponding higher brightness temperature – see the color-scale bar in the Figure 8.

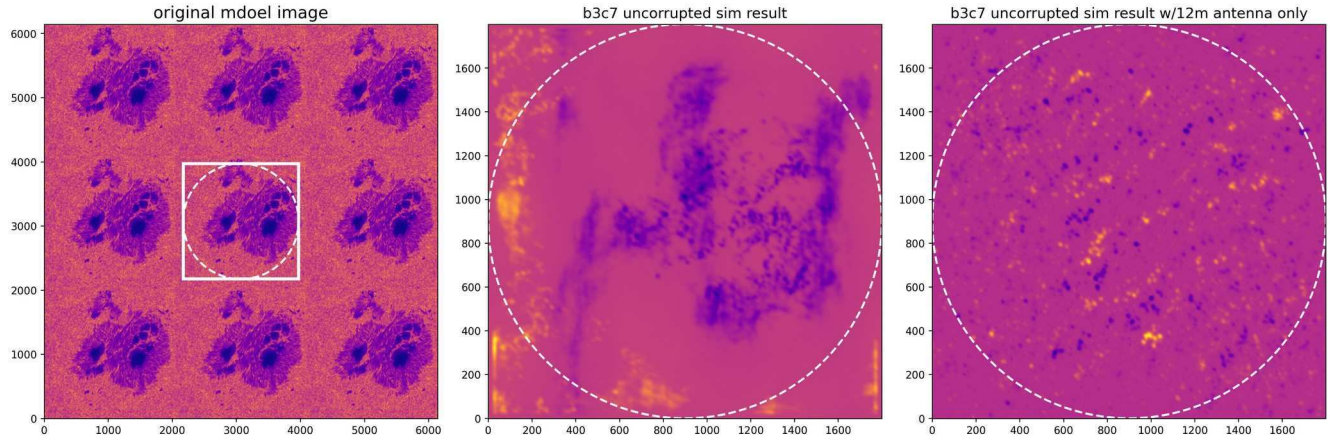


Figure 4: Like in Figure 3, now for the array configuration C-7. One pixel (unit at the axes) corresponds to 0.033 arcsec, the boxed circle (Band 3 PB for 12m antenna) is ~ 1arcmin.

The results clearly show importance of the 7m array – without 7m antennas the synthesized image both in C-8 and C-7 even remotely does not resemble the original model. With the C-7 and combined 12m+7m baselines, the image qualitatively approaches to the original model: We did not have done fidelity-image analysis for this troposphere-less simulation but we did it for the same but perturbed case (see Figure 5 below) and indeed, after self-calibration the relative difference between the model and simulated deconvolved image is plausible.

On the other hand, in the configuration C-8, the proximity between the model and reconstructed images is not reached even at the qualitative level. It should be noted that the maximum recoverable scale (MRS) for solely 12m array in the C-8 configuration is just 1.4 arcsec. Clearly, presence of the 7m array is therefore crucial. We did not investigate the case where both arrays are present but the mixed 12m—7m baselines are missing. However, this case was tried by colleagues in Oslo with their simulation setup and they found even the C-6 configuration as the last plausible for the Band 3 observations (in our case C-7 is still OK), which demonstrates importance of the heterogeneous baselines for improvement of imaging. In the case of full mixed-baselines array, MRS – as given by the shortest baselines – is the same as the FoV and, moreover, the intermediate scales are covered well, as was proved by analysis of the observed test dataset (see Figure 21).

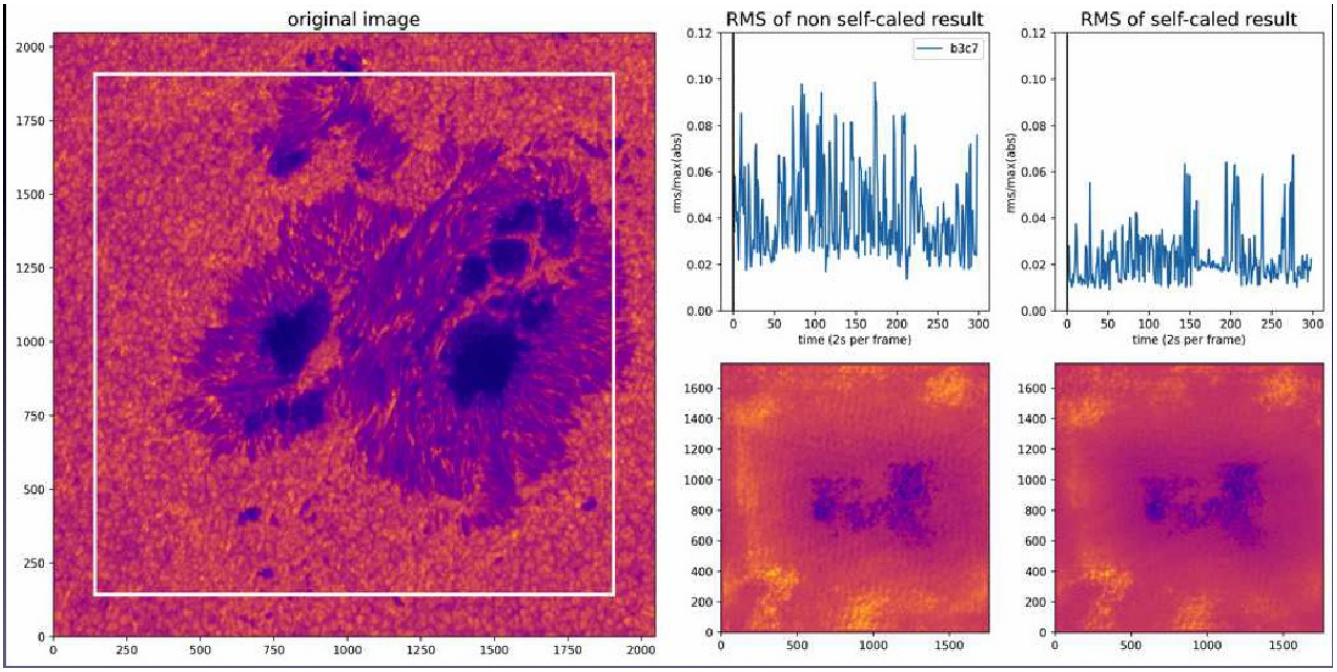


Figure 5: ALMA simulation of Band 3 in C-7 configuration corrupted by the tropospheric turbulence. Self-calibration still allows to reconstruct the image. Scaling at axes as in Figure 4.

In addition to failed convergence, the sparse antenna array in extended configuration has a yet one more potential risk for the image quality – increased uncertainty in determination of the brightness temperature due to the more pronounced side-lobes of the synthesized beam – this results into higher **noise in the image**, which can be – only to some extent – 'cured' by appropriate weighting scheme.

The higher spatial resolution is not itself a limiting factor that would prevent us from reaching a sufficient SNR, since the sensitivity of the array is independent of the lengths of the baselines. Also the brightness-temperature contrast can be expected at hundreds of Kelvins even at the smallest resolved scales, at least for the structures, which are of our utmost interest to be observed with the high resolution, like the chromospheric shock waves and discontinuities or the fine emission threads in prominences. Nevertheless, the synthesized side-lobes may add further noise to the clean image and deteriorate its quality. In the extreme case, we can expect the CLEAN algorithm even fail to deconvolve the image and in this sense both issues are related. Here we present the extreme situation, where it was clearly impossible to reproduce the original model image even in its rough features. The finer issue of increased noise at smallest scales in the images that were successfully reconstructed, has to be yet investigated by more detailed analysis of the fidelity images. We plan to perform wavelet analysis on the fidelity image in order to find out, whether the relative difference between the model and reconstructed images tends to grow under specific circumstances – either the small-scale limit of spatial frequencies or in the specific parts (e.g., with high or low signal) in the image.

Similar analysis has been done for Band 5 and it was found that the C-6 is the last array configuration, where we have reached at least qualitative agreement between the model and reconstructed simulated image. This preliminary analysis has shrunk the space of possible configurations for our further more detailed investigations, now with the inclusion of the tropospheric effects. The complete procedure can be illustrated on the case of ALMA simulated observation in Band 5 in the configuration C-6.

1. We calculated the synthesis beam using the wavelength and array configuration information, and determined the time and pointing direction of the ‘pseudo’ observation. Normally we put the model image exactly at the center of the Sun at a given time when ALMA can observe it.

2. We then converted the model image from fits file to CASA image file using *importfits*, during which the beam information and a 4D coordinate axes that contains directions and spectral values will be given to the image (e.g. ['06h06m56.98', '+23d25m45.5', '198GHz', 'I']).

3. Inspired by the examples presented in the webpage of CASA simulator (https://casadocs.readthedocs.io/en/stable/api/tt/casatools.simulator.html), we combined the ALMA 12-m and 7-m antenna array and simulate a solar observation with pre-set integration time of 2 second and duration of 10 min. An ideal measurement set was generated during the process and ready to be corrupted by the atmospheric turbulence.

We modified the script `ATMOS.py` (Fomalont, Villard 2023 – see the adapted code in *Annex 4*) in order to generate a series of Gaussian-like phase disturbance based on the baseline length from a given reference antenna, and applied them to the ideal measurement set using ***gaincal*** command in CASA. Figure 6 shows the scatter plot of the phase rms as well as the phase error vs. the baseline length measured as a distance from the given reference antenna. The left panel shows the underlying model of the atmospheric structure function: The phase rms increases rapidly (with the power index 0.8) until it reaches a given baseline length for which we specify the phase error (30 degs for this specific case) and saturates there (see the slower rise with the index 0.25) . The typical phase error can be tuned in

order to represent different observing conditions. As we are interested in correction of the fast phase variations in self-calibrated movies, we focus on characterization of the observing conditions by the phase rms error rather than PWV. However, these two quantities are somewhat related as reported by L. Maud et al. in ALMA Memo 624 [24] – see Fig. 13 and discussion at page 19 of this ALMA Memo. The median phase rms for the data without corrections to WVR is reported to be ~0.2mm for 1 km baseline and 120s integration; this corresponds to ~30 degs for the Band 3 data. In this sense, the values of phase rms errors used in our simulations are rather typical: 30 degs, which is the value used for the most of our results shown in the report, being close to median, the higher modeled values corresponding to a bit worse weather conditions. Per the phase-rms/PWV scatter plot in Fig. 19 in [24] the phase errors used in our study should also fall into range of the typical values for the PWV.

Figure 7 shows the plot of phase gain cal table which was then applied to the ideal measurement set to form the ‘corrupted’ one that simulates the atmospheric disturbance in observations. After applying the phase error and generating the ‘corrupted’ measurement set, we then performed the self-calibration/imaging loop with the ‘corrupted’ measurement set. In view that the solar limb observations have many issues already in the current, compact configuration setup, we did not aim at them with the higher resolution. Sure, there is a lot of scientific motivation for having them, too (e.g. the structure and dynamics of solar spicules or the already mentioned fine threads in prominences) but the limb data require careful combination of the INT and regional TP observations and this task aims beyond the possibilities of the current study.

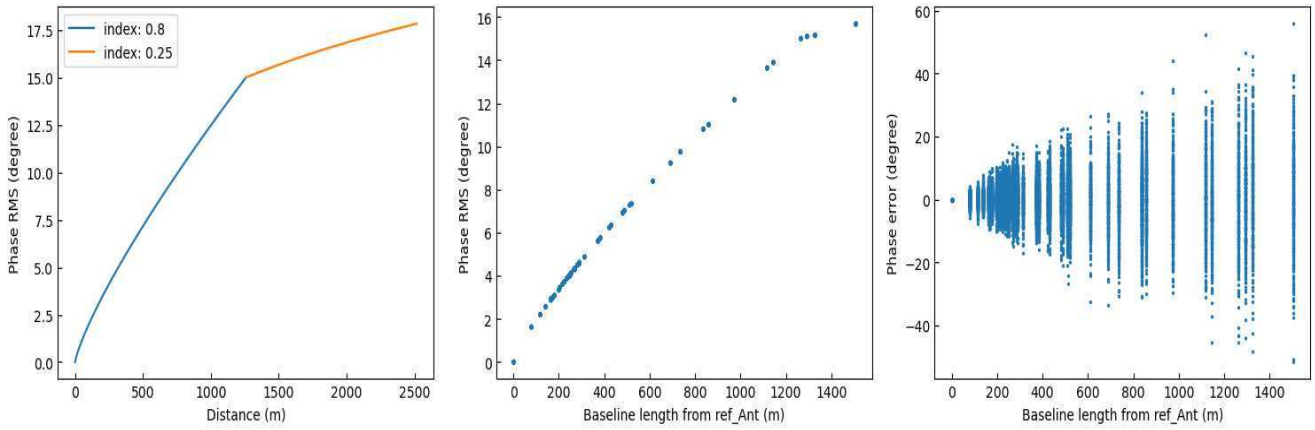


Figure 6: An example of the Gaussian phase error generated by the script. Left panel: Used model for the atmospheric structure function. Middle panel: Realization of that model for the array configuration C-6 (Note: The baselines lengths are measured from the ref. antenna, which is close to the array center; the longest distance is thus ~1500m). Right: Resulting randomized (Gaussian) phase errors for the tested configuration C-6.

4.1.3. Calibration, imaging, self-calibration

Simulated dataset contains just the phase errors introduced by the ATMOS code – these have no long-term trend to be removed by phase calibration (the phase calibrator is not contained in the MS either). The standard calibration step is thus skipped with the simulated data and we continue directly to self-calibration/imaging loop.

Based on the *ALMA memo 620* (see [23]), which discusses the general idea behind the self-calibration of radio interferometers as well as our previous experience in processing the ALMA solar observational data, we carried out a self-calibration scheme which uses a decreasing solution interval as the self-cal proceeds.

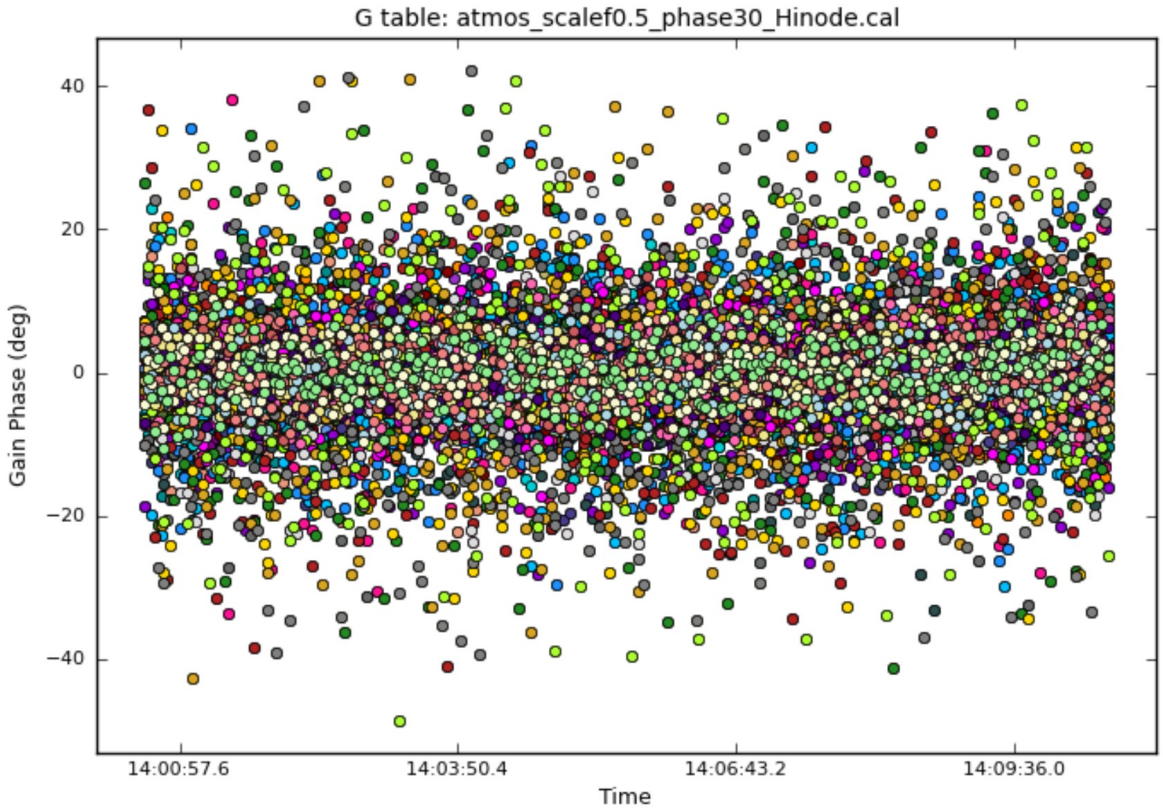


Figure 7: Phase gain cal table plot. Different color represent different antenna. The plot has been acquired for the phase RMS of 30 degrees. This is considered to be corresponding to a typical weather condition at AOS – see the text (Luke Maud, personal communication and [24]).

The general self-calibration scheme can be described as follows:

A. Split out target data, so this now forms the DATA column of a new MS file.
B. Make an image from the entire scan period using the ***tclean*** task in CASA, to use as a model for self-calibration. The model is stored in the dataset for later use.
C. Determine phase corrections for each solution interval relative to the model given by step B using the ***gaincal*** task in CASA, and generate a self-calibration table.
D. Apply the phase corrections to the DATA column using the ***applycal*** task in CASA and generate an intermediate CORRECTED data column.
E. Split out the CORRECTED data column to a new MS file, then repeat from step B until the result is satisfactory.
F. When this loop is finished, repeat the procedure for diminished solution interval. We change the `solint` parameter gradually down to the integration length.

The stopping threshold for self-calibration often depends on the residual as well as the self-calibration table, which will be demonstrated later in section 4.2.

4.1.4. Results

In order to compare the result under certain conditions, some general ***tclean*** parameters that come from previous self-calibration scripts for solar data are kept to the same value throughout the whole routine. The complete set of the parameters can be found in the code listings in the *Annex 4: The scripts.* For example, while doing the simulation of Band 5 observation under array configuration 4, 5 and 6, we modified the cell size as well as the *imsize* for each configuration while keeping the remaining ***tclean*** parameters unchanged. However due to the large image size in pixels at longer baseline configuration, the self-calibration results from Band 5 configuration 6 is not as good as the others, which can be seen from Figure 8 to Figure 10. The simulated brightness temperature (in arbitrary units), has been, again, normalized to the range 0.0 – 1.0. The color coding used in all subsequent simulated brightness maps is shown by the bar next to the left panel of Figure 8. The improvement between raw image and self-caled ones are minimal in this large configuration case. This issue is mainly caused by the model image was not cleaned enough during the self-calibration process. Further comparison and analysis will be given in section 4.2.

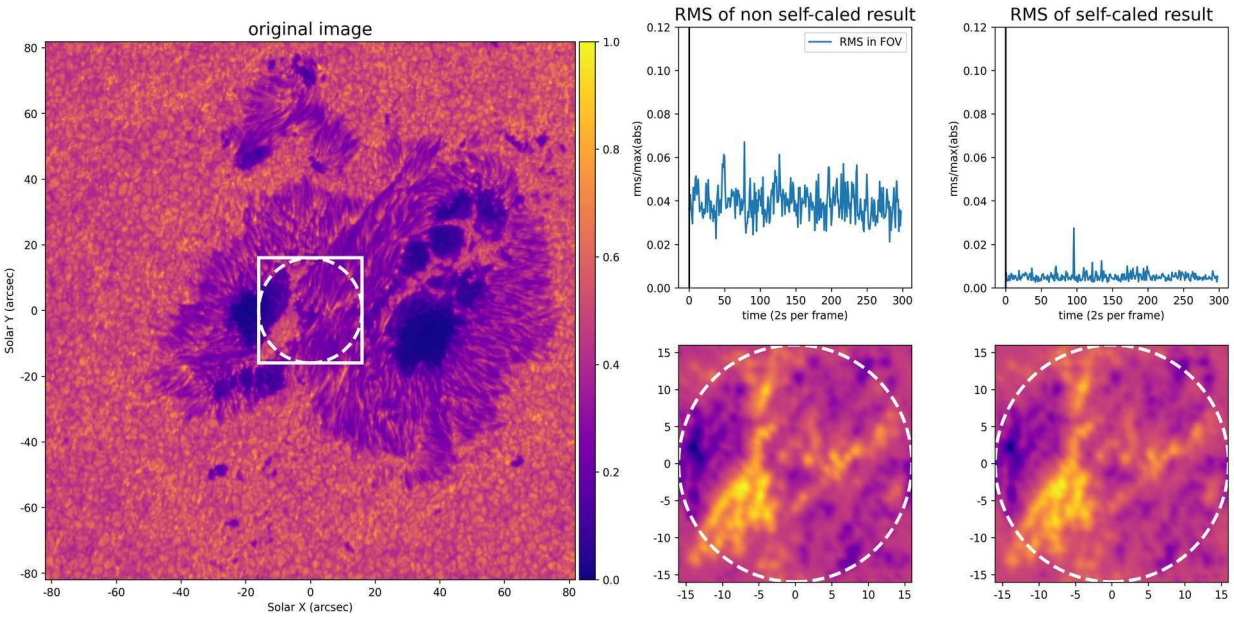


Figure 8: Simulation results of ALMA Band 5 configuration 4. Square in the center represents a bounding-box for the circular FoV – PB of 12m antenna in Band 5. Axes are in arcsecs, their origin (offset) has been arbitrarily set to the image center. The color-coding scheme for this and related figures is shown by the bar next to the left panel.

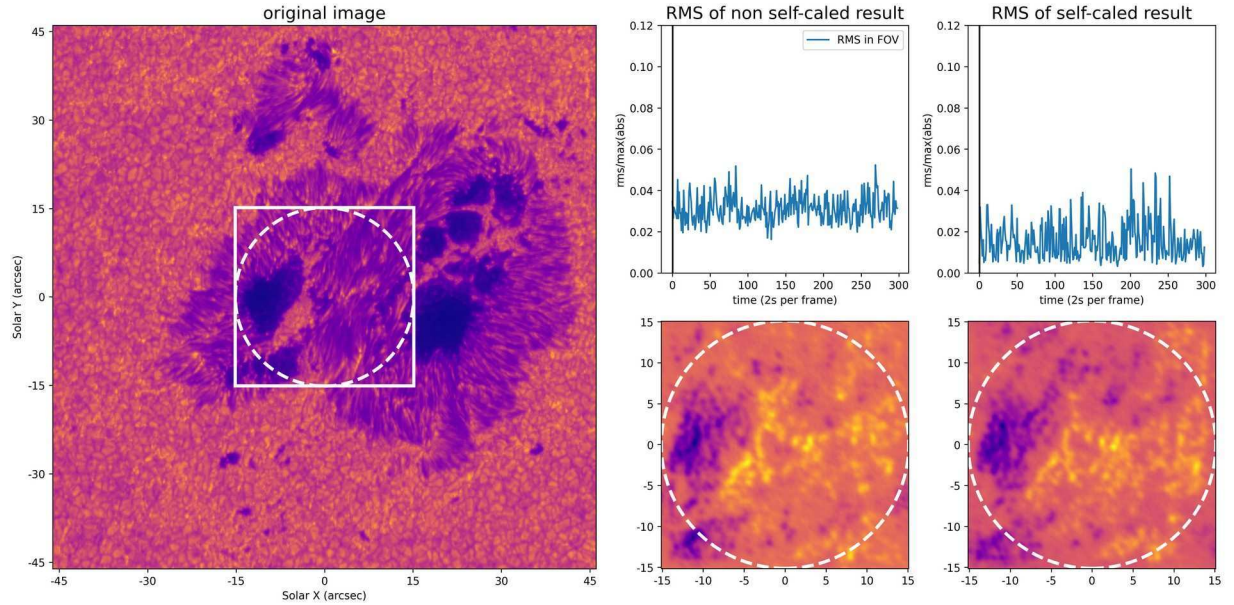


Figure 9: Simulation results of ALMA Band 5 configuration 5. .

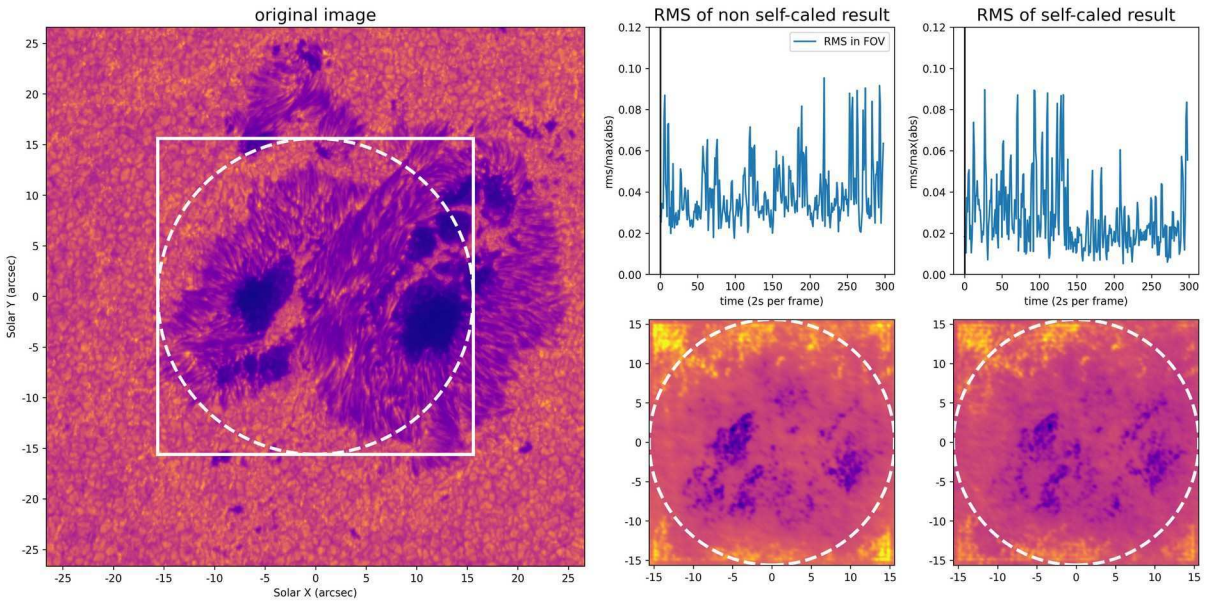


Figure 10: Simulation results of ALMA band 5 configuration 6.

4.2 Comparison of the selfCal-corrected results with the original unbiased model images – quality assessment (DS1.1.2)

In order to asses quality of the imaging and self-calibration procedure, we are using various metrics for reconstructed image features. Some – like standard image fidelity – rely on the knowledge of the ideal model (that has been used as a simulation input), the others are more general and try to asses the imaging/selfcal quality just from the resulting synthesized images. Such metrics are suitable for the observations (where have no ideal model to be compared with the result), too. The goal is to study the correlations between the standard image fidelity and a model-independent metrics and pick one that correlates the best for future quality assessment of the images/movies obtained from actual observations.

4.2.1 Image fidelity

Standard quantity defined in CASA docs for **`simanalyze()`** – ratio between reconstructed image and its (absolute) difference the ideal input model. Clearly, both the model and reconstructed images need to be known for calculation of this metric. In our results in, e.g., Figure 12 and following we are using "normalized difference", which is just the inverted standard fidelity.

4.2.2 Self-cal-corrected vs. uncorrected phase RMS

Phase-based metric: Phase RMS time series for raw and self-calibrated image sequences. Does not measure the quality of the image itself but rather 'convergence' of the self-cal/imaging loop.

4.2.3 Scale analysis: Fourier (power) spectra of the input-model, simulated clean image, and its residual

Analysis of the image structure: Which scales present in the original ideal image have been truly transferred to the synthesized image and what remained yet in the residuals. Can be used also just for clean and residual image, without apriori knowledge of the ideal model.

4.2.4 Results

For comparison between self-calibrated images and non self-calibrated ones, we then use the model image to create the fidelity images and study the difference between these two results.

We show here ALMA simulation of Band 5 and array configuration C-6 as an example. We first generate the "normalised difference" images (which are just the inverted standard fidelity) by subtracting the normalized model image from the normalized results, which could be either the unself-caled or the self-caled image. Then calculate the standard deviation as well as the mean value of each fidelity image and plot these values as dependence on time.

We found that the mean value of this series of fidelity images from the self-cal procedure (Figure 12) has a less spiky feature compared with the unself-calibrated ones (Figure 11), which indicates that the self-calibration does help reduce the random noise which simulates the atmospheric disturbance.

However, since the model image is not changing according to time, if the random noise is fully cleaned, the standard deviation of the fidelity images will stay the same while the mean value shows almost no spiky features in the figure.

As we looked into our self-calibration procedure, we noticed that due to the long baseline we used in simulation, the image size in pixel (1200 px by 1200 px) is far larger than that of previous ALMA solar observations, which, in the case of ALMA band 3 configuration 4, is only ~ 400 px by 400 px.

Thus, the self-cal result shown in Figure 12 is not cleaned enough with current `niter` (= 40,000) setting, which also proved by Figure 13 showing the gain table plot of the last round of the self-cal procedure. Unlike the normal gain table plot we see from the last round of self-calibration, the phase correction is still too large, especially at the end part of the simulation.

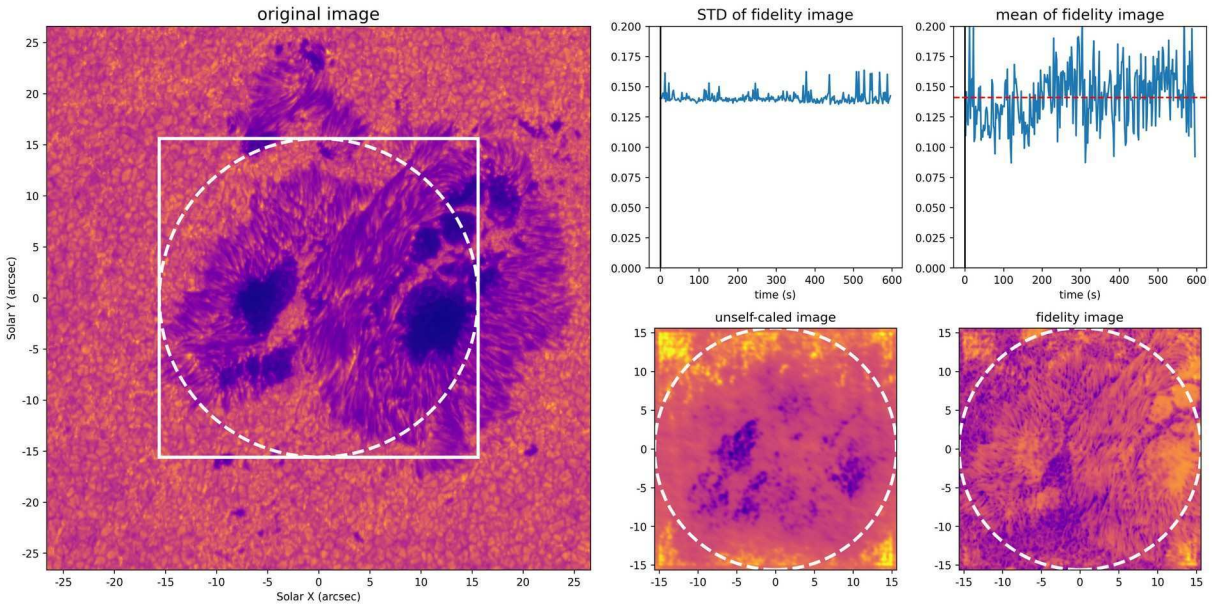


Figure 11: Comparison between model image and unself-caled results. Band 5 in the C-6 configuration.

With the new `niter=200k` setting, we plotted Figure 15, which shows the comparison between model image and self-caled result as well as the corresponding fidelity image. The standard deviations of the fidelity images are almost constant, represented by the horizontal line in the upper middle panel of Figure 15. While there is a general pattern for the mean values of the fidelity images, the spiky features seen in Figure 11 and Figure 12 top right panel, are almost gone as compared to that in Figure 15, indicating that the random noise, which is caused by the atmospheric turbulence, is mostly cleaned from the dataset.

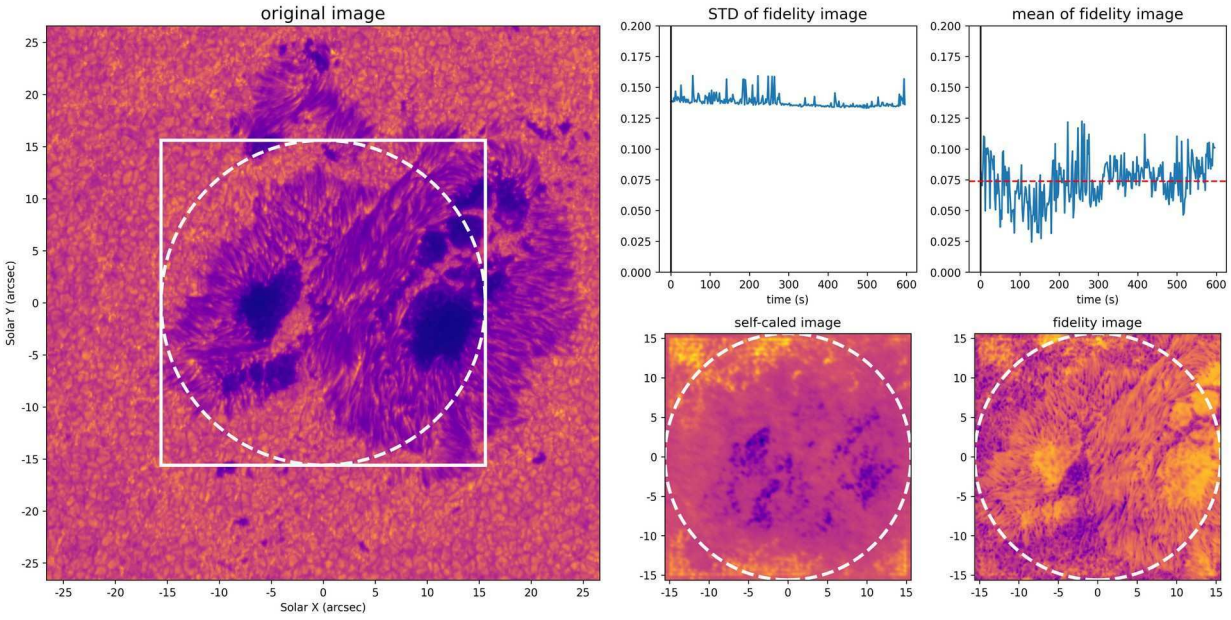


Figure 12: Comparison between model image and self-caled results. Cleaning iterated over 40k cycles.

To further analyze the simulation results, we made power spectrum for all the three cases we presented above: unself-caled raw image, self-caled image with small niter number (40,000), and self-caled image with large `niter` number (200,000). The power spectra were made from the first frame of the snap-shot movie from the whole 10 min simulated observation to serve as an example.

Unlike the raw image and less self-caled one, the power spectrum of the deeply self-caled image shows a strong signal near the small k number, which corresponds to large features in the map. This result proves that the large niter setting is useful in recovering the overall shape of the model image (Figure 15).
In order to measure the qualitative 'spikiness' of the frame-to-frame jumps as shown in Figure 11 and Figure 12 in more quantitative manner, we devised a possible metrics for measuring it. We do the differences between the adjacent snapshots (basically a time derivative) and plot the mean and standard deviation of the difference images as a next plot. Basically, a normalized (to the length of the series) integral of the difference curves provides a number that measures the frame-to-frame jumps. Again, we show two examples for self-calibrated series and its non-self-calibrated counterpart in Figure 16 and Figure 17.

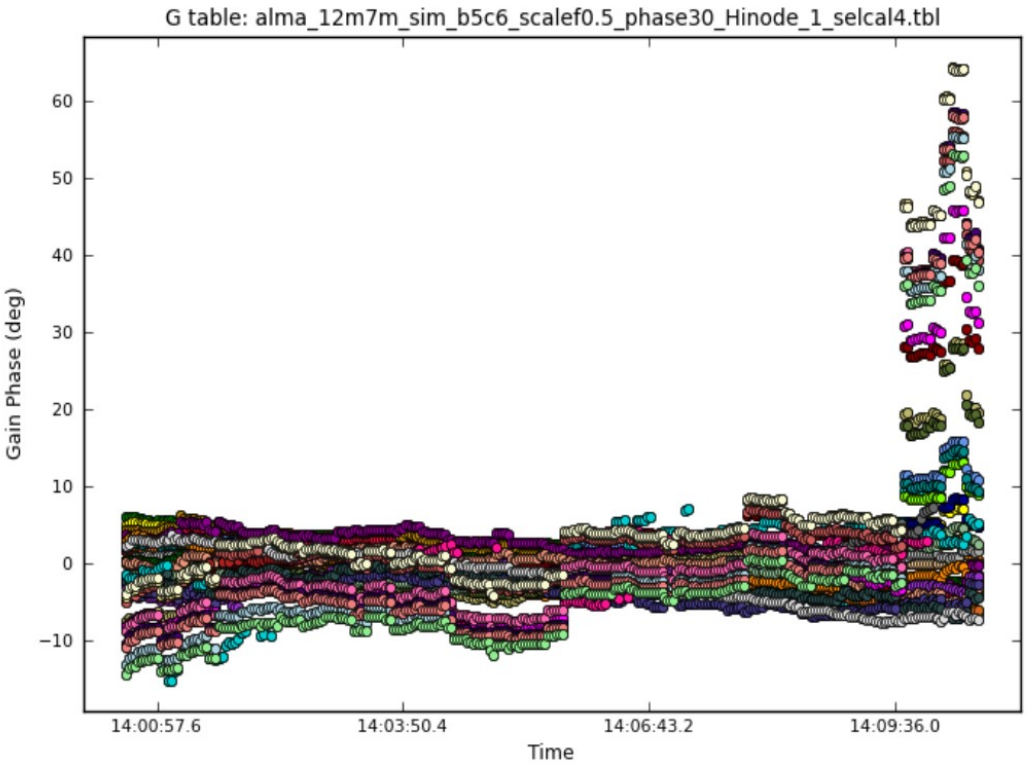


Figure 13: Gain table plot for the last round of self-calibration which uses niter = 40,000.

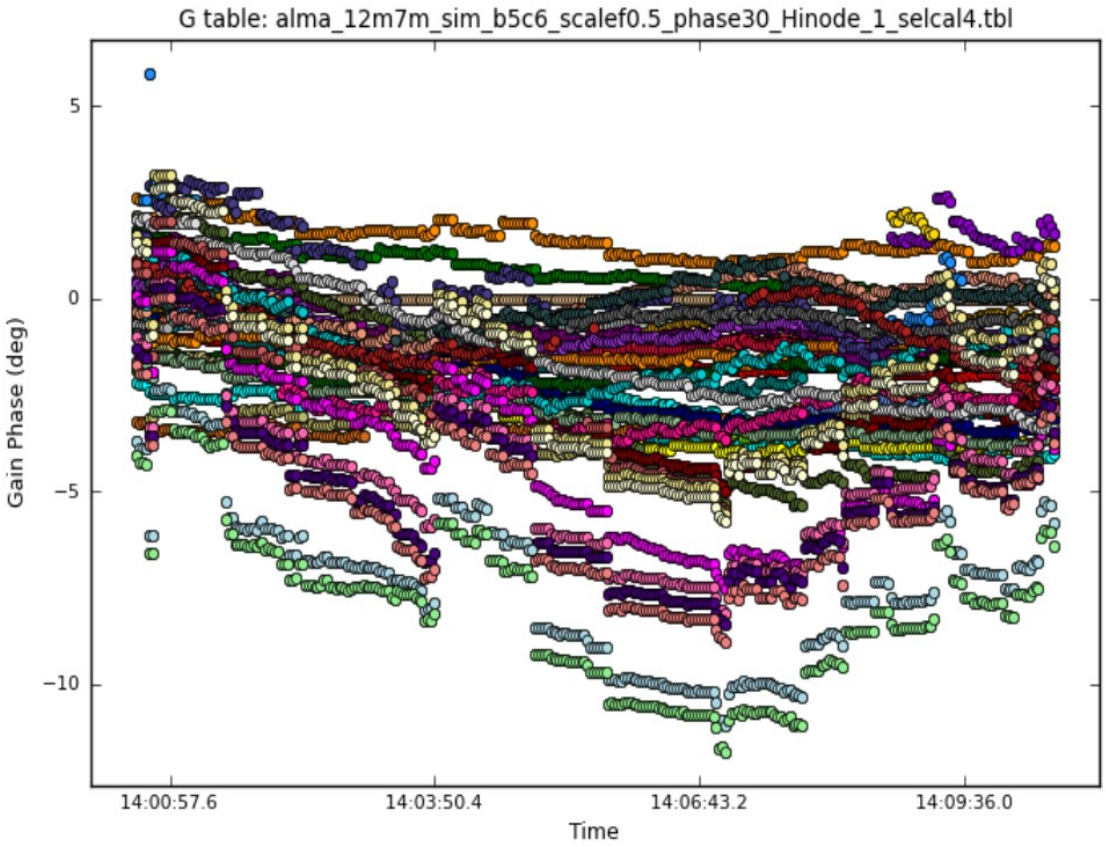


Figure 14: Gain table plot for last round of self-calibration which using `niter` *= 200,000.*

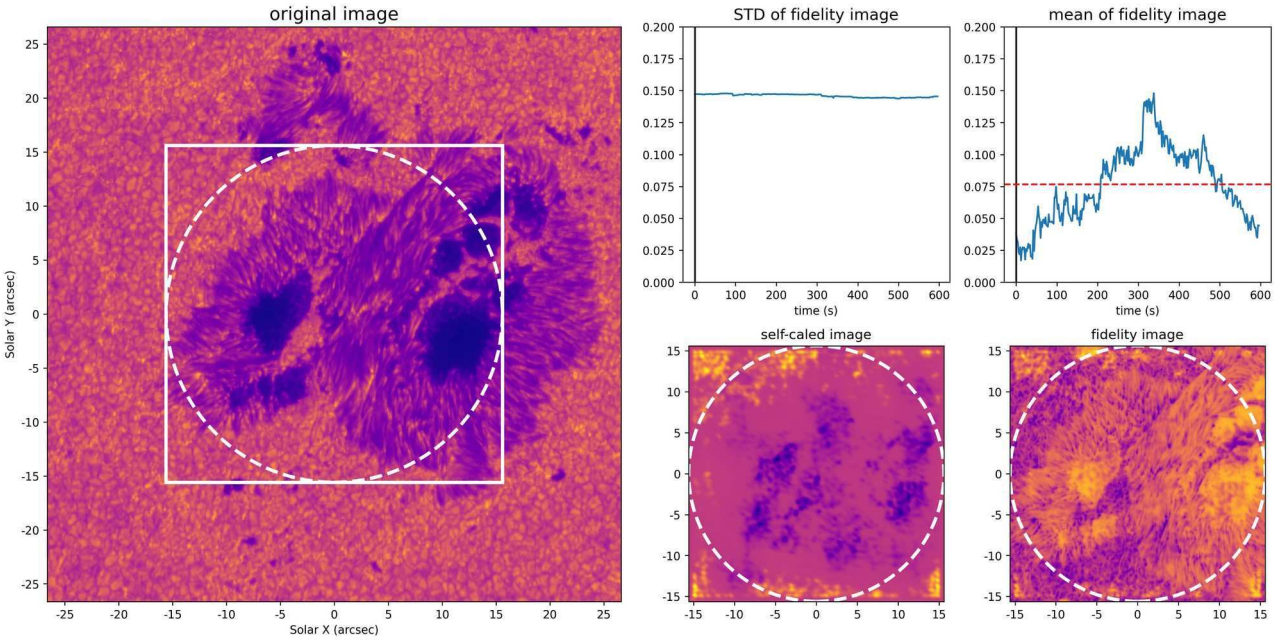


Figure 15: Comparison between model image and self-caled results with niter set to 200,000.

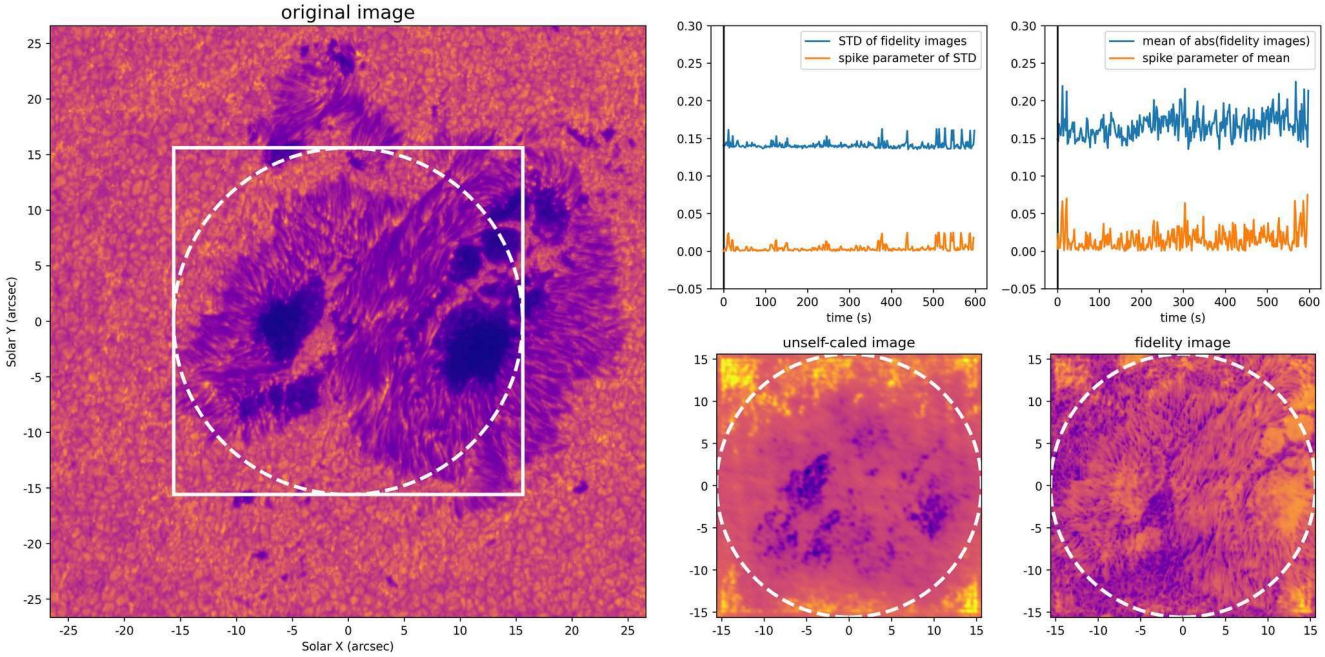


Figure 16: More quantitative measure of the spikiness as displayed in Figure 11 and Figure 15 obtained from running differences – the orange curve. Its normalized integral provides a metric for characterizing frame-to-frame jumps in the time series. Result for the time series without self-calibration correction.

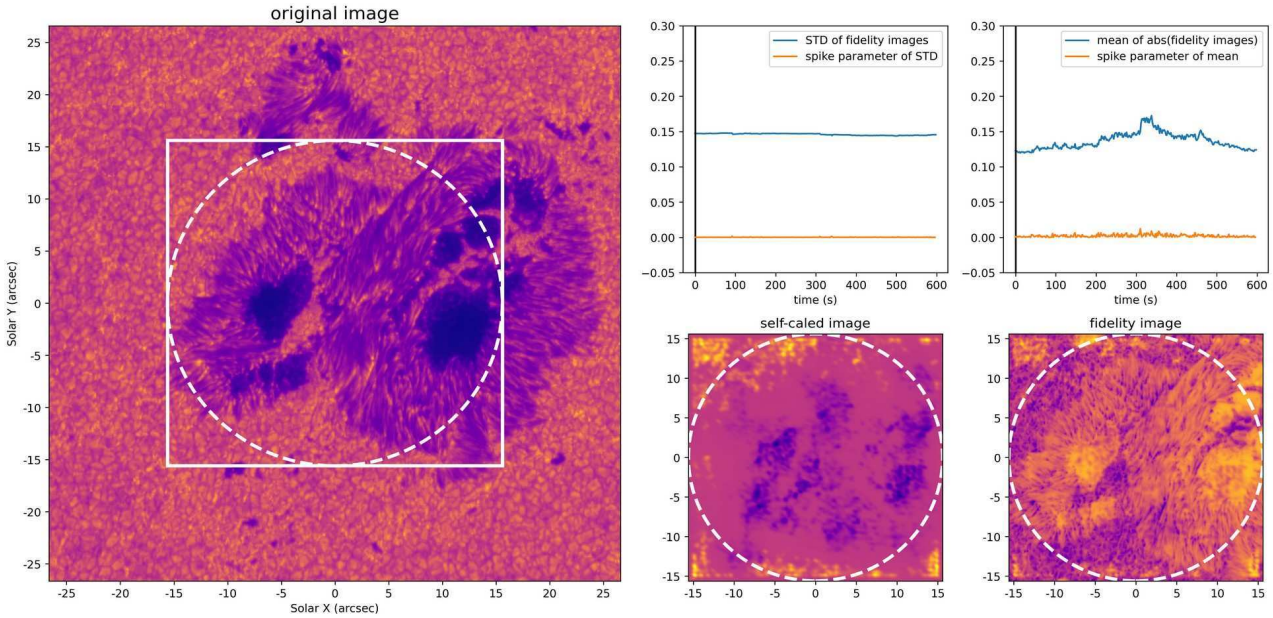


Figure 17: The same like in Figure 16 but now for the self-calibrated series (cleaned with 200k cycles).

4.3 Solar specific Go/noGo criterion (DS1.1.3)

As it has been written in Section 1, the main goal of this study is to find, whether solar observation with a given array configuration and in a given frequency band is possible for the given phase metrics (RMS), as established from the BP phase variations of the preceding execution. The boundary/marginal cases should be expressed by the lines as those, which are schematically shown in the illustration in Figure 1.

Using the simulations in the setup described above in paragraph 4.1 in conjunction with the measures that define the quality of the imaging and self-calibration procedures of time sequences of solar snapshots presented in paragraph 4.2, we have started to explore the parametric space [base-line length; BP phase RMS; observing frequency] in order to find the points of the boundary lines for solar Go/noGo. If one of the following (negative) criteria was met, the respective point in the parametric space was marked as "noGo":

- The images in series has been poorly reconstructed – the low image fidelity. Most likely supercritical dilution of the uv coverage, imaging failed.
- The phase variations not well removed by the self-calibration procedure – self-calibration loop not converging.

Otherwise, the point was marked as "Go".

Insofar we have been mapping the phase RMS to the solar Go/noGo from the 'standard' Go/noGo measurements, i.e. the RMS of the BP-calibrator scan form previous observation is measured on the timescale of minutes (typically 5 min.). Since for the solar observations the much shorter timescales (on the sub-second scale, that are, however, not directly reachable form the BP scan) are relevant, and at the same moment the modified Go/NoGo procedures are started being utilized by ALMA, measuring the RMS at much shorter (a few tens of seconds, typically 30s), the future variants of the Figure 1 shall include mapping between solar Go/noGo and phase RMS as measured at the shorter timescales, too.

First, it has been found that even with the unperturbed case of the phase RMS=0 – see Figure 3 – we cannot use more extended configurations than C-8 for Band 3: Already for this marginal configuration even qualitative agreement between the input model and the reconstructed image is not reached. The reason is in specifics of the solar observations – broad, continuous range of spatial scales in the image in connection with the sparse instantaneous *uv* sampling. It is possible that future deconvolution algorithms may overcome this limit but for the standard CLEAN-based procedures the uv-space dilution represents the very upper limit Then different frequencies under different weather condition (as characterized by the phase variations) have been tried. The so-far reached results can be summarized as follows:

- **Band 3:** Marginal (already rather "noGo") configuration for unperturbed case (0 deg phase variations) is **C-8**.
- **Band 3**: Limiting configuration is **C-7** with 30deg variations.

- **Band 5**: Limiting configuration is **C-6** with 120deg variations.
- **Band 6**: Limiting configuration is **C-5** with 120deg variations.

The results are displayed in Figure 27, which is a still developing but actual realization of the scheme in Figure 1. Further calculations (for further vector of parameters) are on their way, however, we would like to first change the model for atmospheric phase corruption. The main reason – in addition to higher fidelity of the new model (Python code by Asaki & Maud, 2023) – is the inclusion of the integration-time dependence in the model. It has been found (e.g., [15]) that the phase correlations for a given band and baseline decrease with shortened integration time. Since the Sun is going to be observed with higher cadence (typically 0.1 – 0.2 sec) than the BP calibrator (2 – 6 sec), whose phase variations are at the x-axis in Figure 27, the model should include this difference.

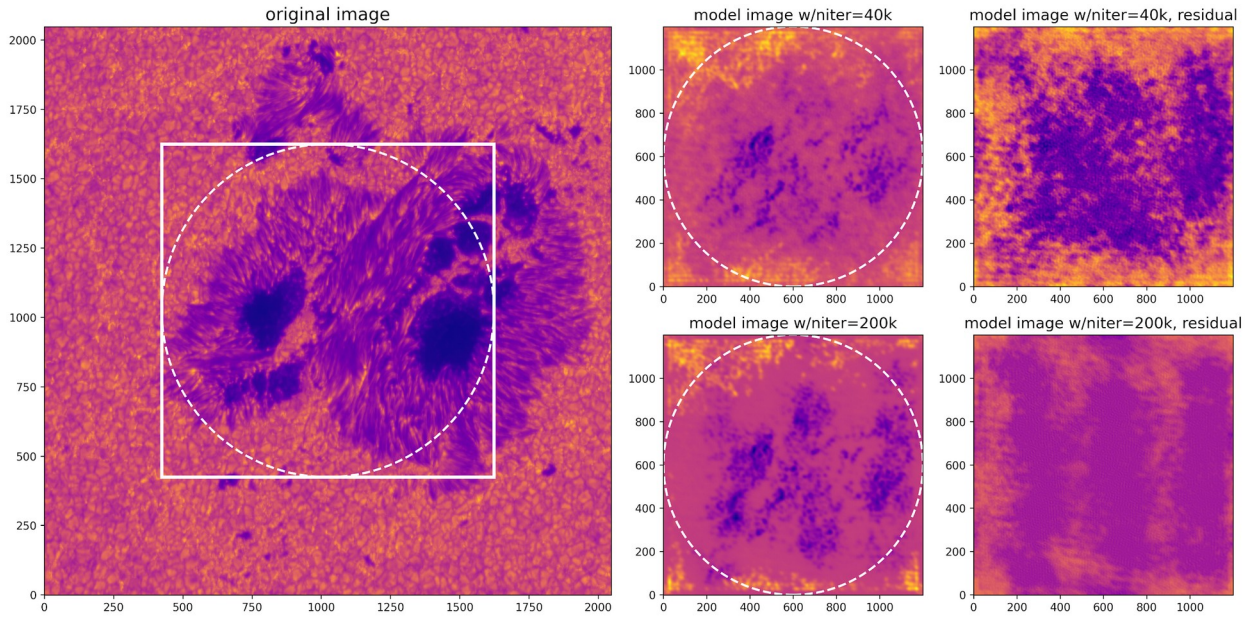


Figure 18: Comparison of model and residual between two different niter cases. Both residual images are plotted under same scale.

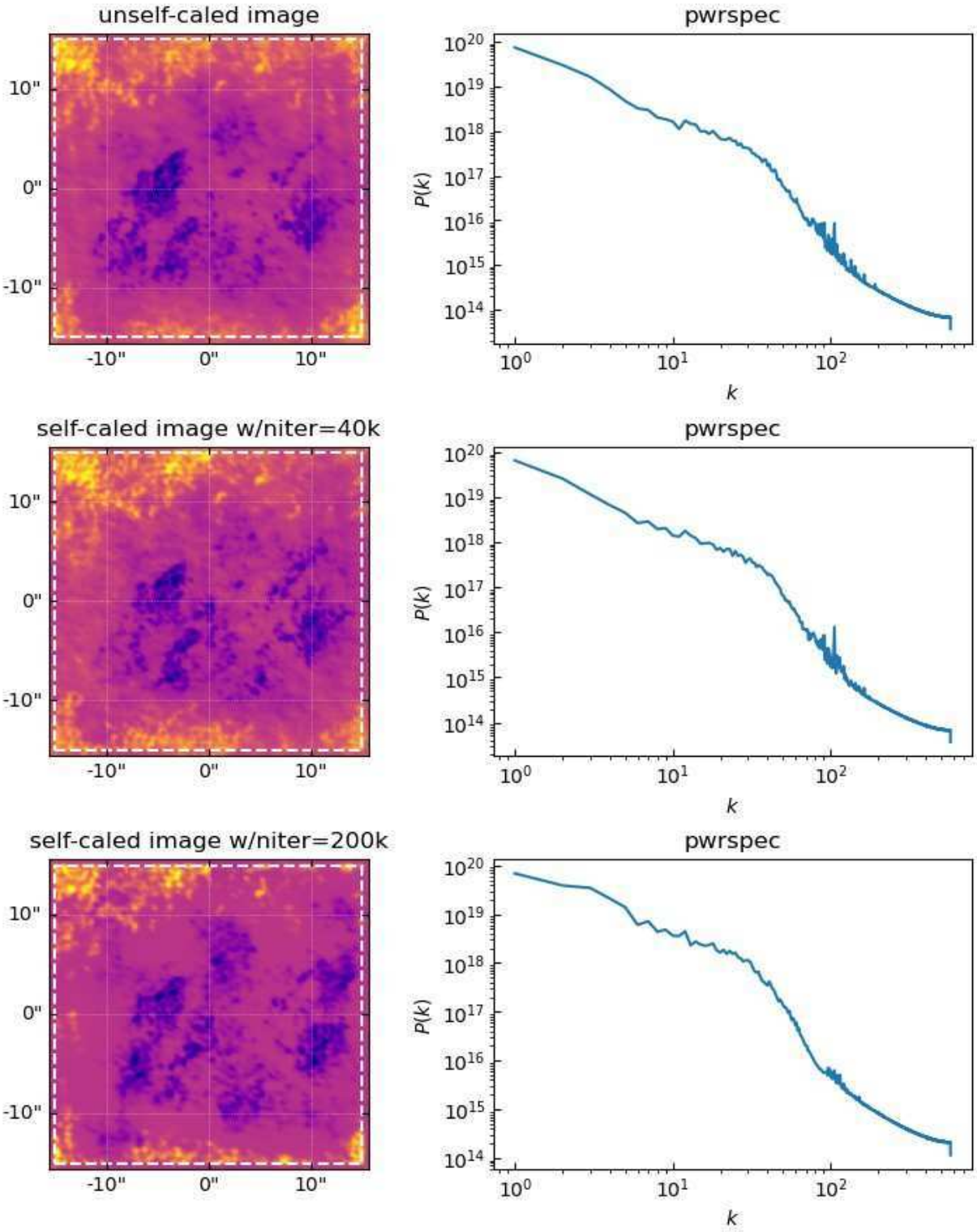


Figure 19: Power spectrum comparison between unself-caled and two self-caled images.

4.4 Analytical relation between *absorption* in the water vapour and the phase delay in Band 5 (DS1.2.1)

This part represent legacy of Richard Hills, a member of the development project team, who, sadly, passed away before the project has started.

Absorption of the solar radiation transmitting through the atmosphere column above each antenna in the contained WV cannot be measured absolutely: We need to record the absorption with the cadence of integration, and because we do not have any amplitude calibration at this short time-scale, its decrease with time hardly could be attributed just to increased absorption. On the other hand, the differential measure (a spectral slope), which includes relative increase/decrease in the amplitude as compared through all frequency channels or their representative sample, is in principle feasible. Alternatively, one can compare the signal as integrated per SPWs, as it is indicated in Figure 27 from the real CSV data (see Section 5.2).

Following the Richard's derivation, whose full text is attached as *Annex 3: Differential absorption in WVR line & the phase delays*, the phase difference introduced by the WV column is related to its fractional absorption by relation

$$d\varphi = 2\pi/\lambda \,.\, r_{eff} \,.\, (-1/\tau 1) \,.\, dT_{sig} / T_{sig}$$

where r_{eff} is the effective refractivity, to be approximated by a value of 6.3 for reasonably broad range of parameters and $\tau 1$ is the opacity for 1mm PWV.

The detailed tables with calculation of the sensitivity of $d\varphi$ to the change of differential absorption done by Richard are to be found here: http://wave.asu.cas.cz/barta/ALMA/DS2022/wv-cal/.

Having now the Band 5 data, we are aiming at verifying these calculations by scatter plot of phase-error per antenna and integration as found by self-calibration vs. the differential absorption, also per antenna and integration, as found from the auto-correlation spectra for each antenna.

Let us note that albeit this procedure is devised and being tested (see Section 5.2) in the ALMA Band 5, it is of more general use and can be – if conceptually proven in Band 5 – also, for example, in Band 7 for the WV line at 324 GHz.

5. Verification of the project approaches at the CSV data

To our greatest pleasure, JAO/ALMA accepted our proposal for the CSV observations and with the kind help of our Project advisory board and Hugo Messias at ALMA the observing plan has been negotiated by means of JIRA CSV-3832. The project got code 2022.1.00031.CSV and has been executed at the end of November (Band 3 data in a transitional configuration close to C-7) and mid December (Band 5 data in C-6). The basic data processing started at the end of the year and advanced analysis is still ongoing. However, already now we can present practical verification (or at least indication of feasibility) of the approaches/activities that are described above in *Deliverables* (Section 4).

5.1 Verification of self-cal corrections for short-integration observations

5.1.1. Observation description

We present detailed results for Band 3 observations here. The Band 5 data are still under processing (self-calibration).

Observation in **Band 3** has been performed on November 29, 2023, on 13:34:39 – 14:01:57 UT. The data have been taken in the (almost) C-7 configuration, under “marginal weather condition” as reported at JIRA ticket. The time on-source includes just a single scan of 10 minutes, with integration of 192 ms. More than 50 antennas have been used for analysis, just one (of 53 available) has been fully flagged during the processing. The observing sequence is shown in Figure 20 (output of **`listobs3()`**). Full MS info is attached in *listobs.txt* in *Annex 5: Observation reports*. In addition to the listobs() output, *Annex 5* contains shift-log report for the entire project, and the QA2 report from the data calibration.

Band 5 execution (the successful one) took place on December 16, 2023, on 13:12:26 – 13:37:25 UT (one scan of 7 minutes on source) in configuration C-6. There were 52 antennas in combined 7m+12m array, 2 of them (one 7m and one 12m) had to be flagged. The PWV was 4.5mm, Array configuration and the *uv*-space coverage are shown in Figure 21. Importance of the combined (heterogeneous) array mentioned in Section 4.1 and summarized in Section 7 is clearly visible: The 7m-12m baselines create “nests” of point in the uv space leading to its denser coverage. Again, the detailed info is in *Annex 5*.

5.1.2. Calibration, basic imaging

Calibration has been assisted by the Solar Script Generator (SSG – see Section 6) – a tool whose development has been eventually finished in connection with processing of the CSV data of this study (the observed datasets brought a new motivation).

The procedure of solar ALMA data calibration is described at the Confluence page:

https://confluence.alma.cl/display/DMG/Solar

With respect to the standard calibration, it includes few more steps, above all a call to the **`sol_ampcal2()`** routine form the collection *sun_reduction_utils.py.* This assures the proper amplitude calibration taking into account both T_{sys} and T_{ant} for the Sun (see Shimojo, 2017). The last calibration step includes, moreover, attributing the correct reference sky coordinates (RA, Dec) to the solar-target fields at the reference time. This is not really solar specific feature and it is present in any ephemeris (solar-system) object.

One antenna has been flagged because extremely high T_{sys} values. At the very end of calibration procedure, **`aU.generateQA2Report()`** has been called and its results checked. Generated QA2 report is attached in the *Annex 5*. The calibration result has been *QA2 passed* and basic imaging

(integrated over the science scan) was all right, too. Hence, we continued with separation to the integration-based time sequence, and self-calibration loop.

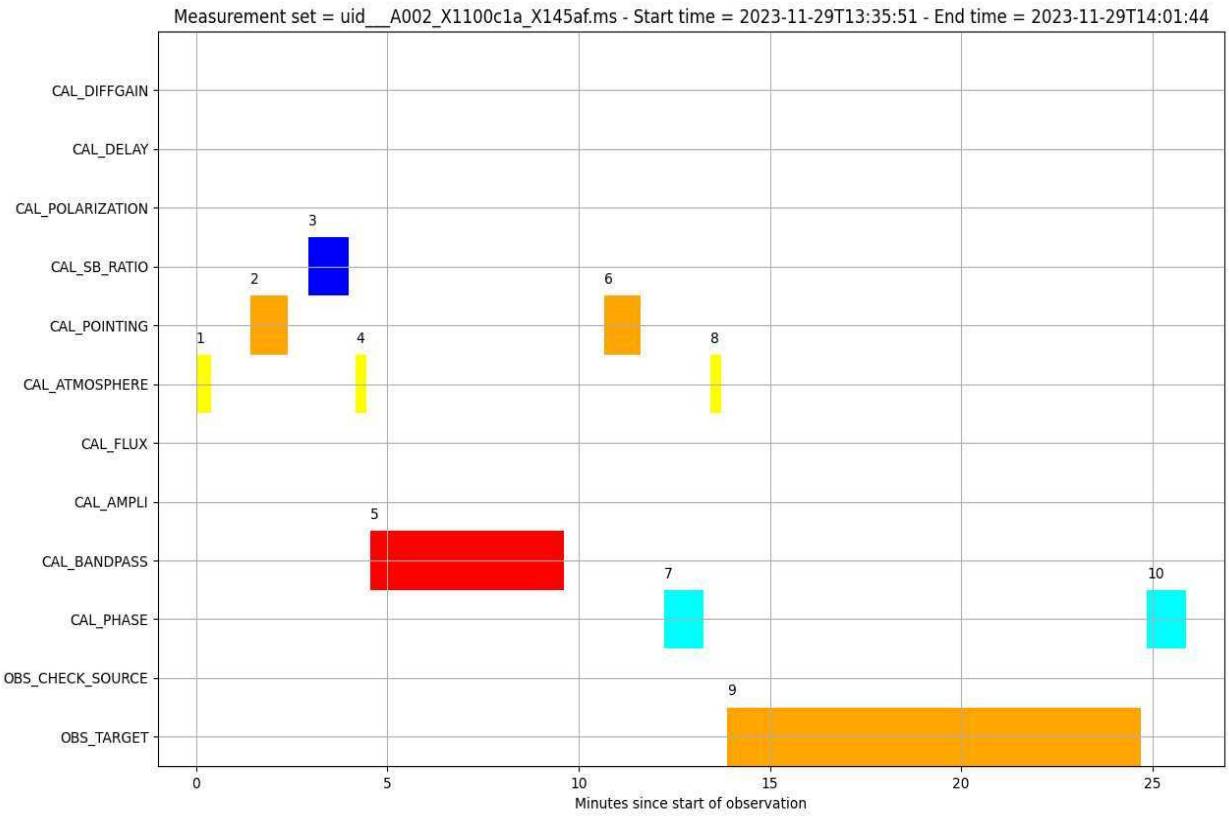


Figure 20: ALMA Band 3 test observation with colored blocks showing different target.

5.1.3. Self-calibration / imaging loop

The scheme of a general self-calibration for solar observation that contains multiple scientific scans can be described as follows:

a) split the data that have been corrected using standard calibration into sub-datasets according to the scan index.

b) For each sub-dataset, make an image from the entire scan period using the ***tclean*** task in CASA, to use as a model for self-calibration. The model is stored in the dataset for later use.

c) Determine phase corrections for each solution interval relative to the model given by step b using the gaincal task in CASA, and generate a self-calibration table.

d) Apply the phase corrections to each image in the sub-dataset using the ***applycal*** task in CASA and generate an intermediately corrected sub-dataset.

e) Iteratively repeat steps b-d, substituting in each intermediate sub-dataset, until no further correction is needed, as determined by checking the calibration table for reduction of the residuals.

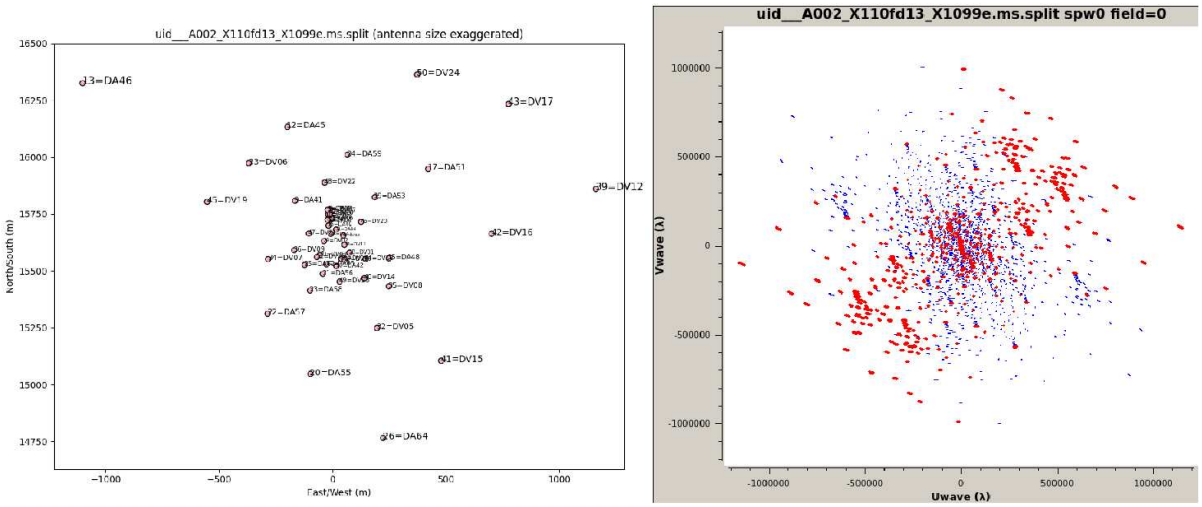


Figure 21: Antenna array configuration and the uv-coverage for the solar target for the Band 5 observation.

For the test solar observation, since we only have one on target scan, we will neglect step (a) and focus on the rest. We will also repeat step (b) to (d) for a fixed 5 times and compare the self-calibration results between different setups, just like what we did for the simulations.

Illustration of the successfully reconstructed image (a single snapshot – the entire-scan spanned self-calibrated movie is still in production) is displayed in Figure 22. The four panels show the ALMA clean brightness map, its corresponding residuum, and – for the sake of comparison – also simultaneously observed data of the same region at the Sun from two other instruments – *Atmospheric Imaging Assembly* (AIA), and the *Helioseismic and Magnetic Imager* (HMI), both onboard The Solar Dynamics Observatory (SDO). The cleaning has been stopped after 200k iterations and the image quality has been subsequently checked. Namely, the residual RMS is less then 10% of the RMS value in the clean image, which is the primary cleaning threshold for us. As the secondary criteria of the image-reconstruction quality we use (similarly as with the simulated data, see – section 4.2.3 and Figure 19) the power spectra in the clean and residual images – see Figure 24 below. This auxiliary measure indicates the successful reconstruction, too. Moreover, from the intensity contours overlaid on the top right panel of Figure 22, we can see that there are some overlaps between the features seen in ALMA Band 3 map and AIA 304Å image. Since AIA 304Å is also sensitive to the solar chromosphere, these similarity in features represent another indication of the quality of the self-calibration results.

During the self-calibration procedure, the image size and cell size are set according to the suggestion given by qa2 report, which is [2048 px, 2048 px] and 0.04 arcsec for the Band 3 configuration C-7 case. As it has been revealed already in the simulated datasets (Sections 4.1.3 and 4.1.4) ***tclean*** for large images (2kpx x 2kpx and larger), whose size implies from the higher resolution, needs

significantly more iterations (typically 200k cycles are required to reach the above stopping criterion) comparing to the 'standard' so-far performed solar imaging of the data taken solely in the compact configurations. Consequently, the whole routine is time consuming has to be be conducted with CASA parallel processing.

While we performed the self-calibration procedure, a "mistake" has been made in the sense that the "standard" number of ***tclean()*** iteration, as typically performed in the QA2 solar imaging script, has been used (40k cycles).This eventually led to early termination of each round of the self-calibration. Though we found out and corrected the issue in the script, we present the deficient result here, too, as it could serve as a good example of how `niter` setting really affects the self-cal result of actual observational data. The issue is illustrated in Figure 23, which shows the ALMA map generated during the first round of the erroneous self-calibration routine. Along with the clean image, the corresponding residual is displayed, as well as the power spectrum for both clean and residuum images. As it can be seen, the residual level is comparable to the clean map itself (and far from our stopping criterion for residual to be less than 10% of the clean image), indicating the failure of this self-calibration. The secondary quality metric – the power spectrum – also doesn't show any sudden drop at the tail part, like we can see in the well self-caled ones – see Figure 24 below for comparison.

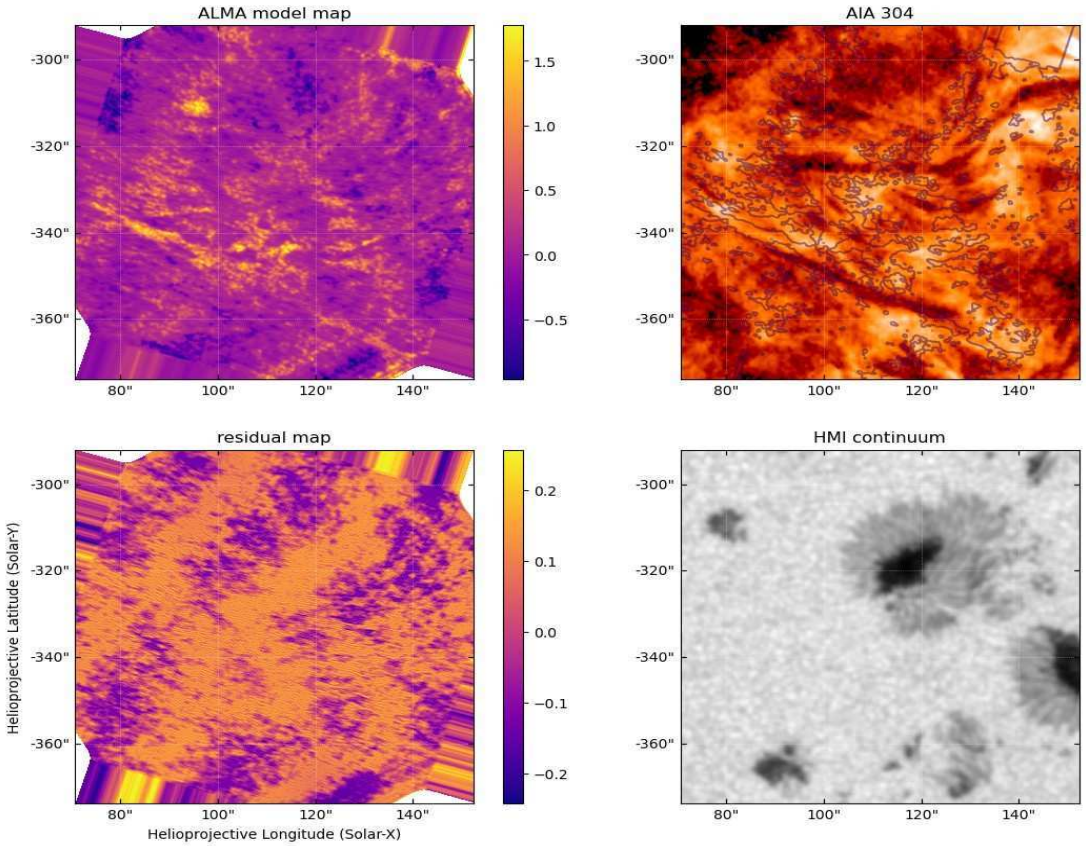


Figure 22: ALMA map and its residual in comparison with AIA 304Å image and HMI continuum image. The AIA wavelength is sensitive to the solar chromosphere, basically we can expect to see similar structures as visible in the ALMA Band 3. The HMI optical continuum shows the underlying photospheric structure – the small sunspot and serves as the context image.

As a direct comparison to Figure 23, which displays the self-cal results with the `niter` settings used in the QA2 imaging of the so-far compact-configuration observations, Figure 24 shows the ALMA map and its residual from the first round of self-cal procedure when the cleaning process is finalized with the niter parameter suitable for the large, high-resolution images. The residual is much lower and much less structured compared to the one in Figure 23. The power spectrum also include a break (the slope change) followed by a sudden drop at roughly k = 100, which indicate the scale of the smallest feature is about 15 pixels. Yet better is the insignificance of the residuum for proper *tclean()* settings (i.e., the correspondingly higher number of iterations) seen in Figure 25, where the clean image and its residuum are displayed in the same range of values (note the same color-scale bar aside).

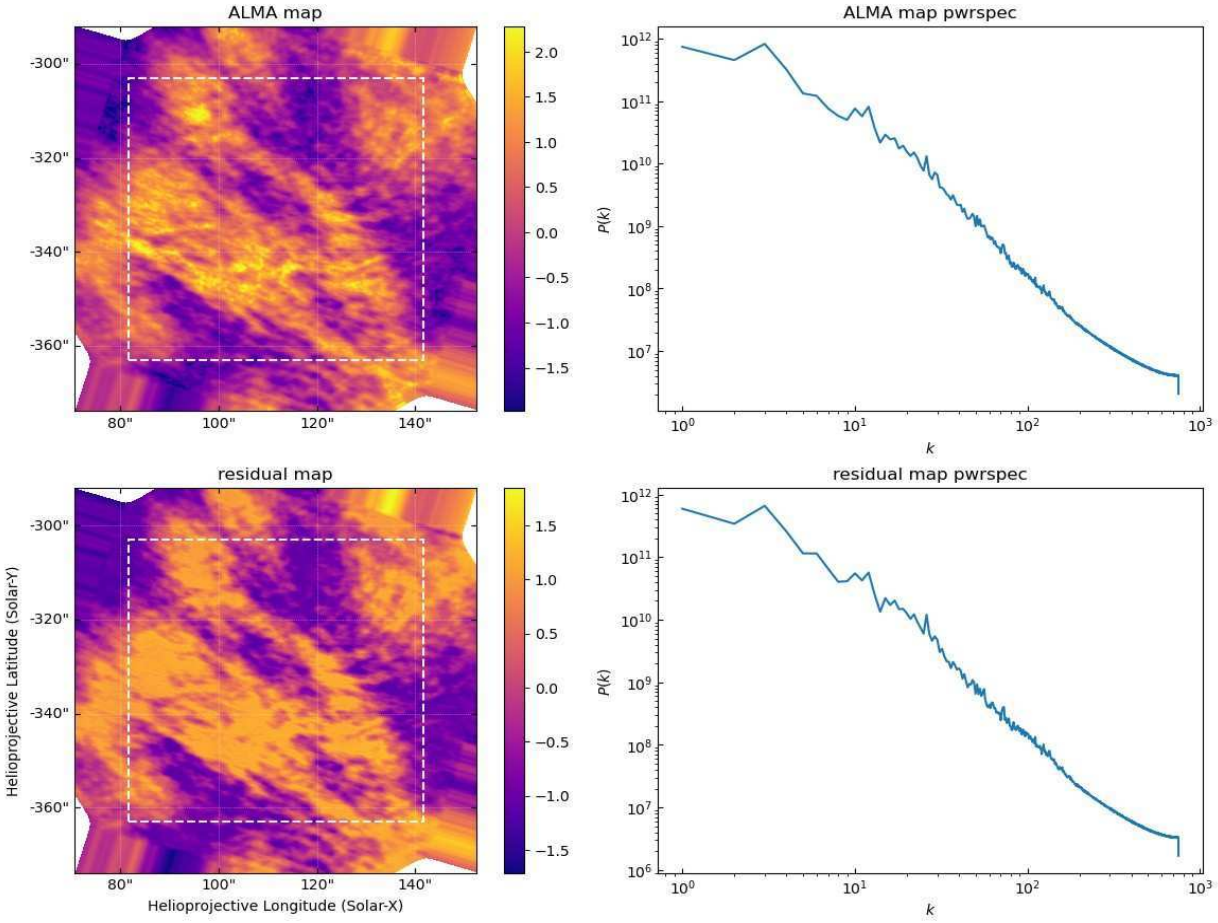


*Figure 23: ALMA map generated in the first round of self-cal routine and the corresponding residual image. The **tclean** process in the self-cal loop has been interrupted prematurely by mistake. The residuum still contains a lot of structures because of that error what is also demonstrated at its Fourier power spectrum.*

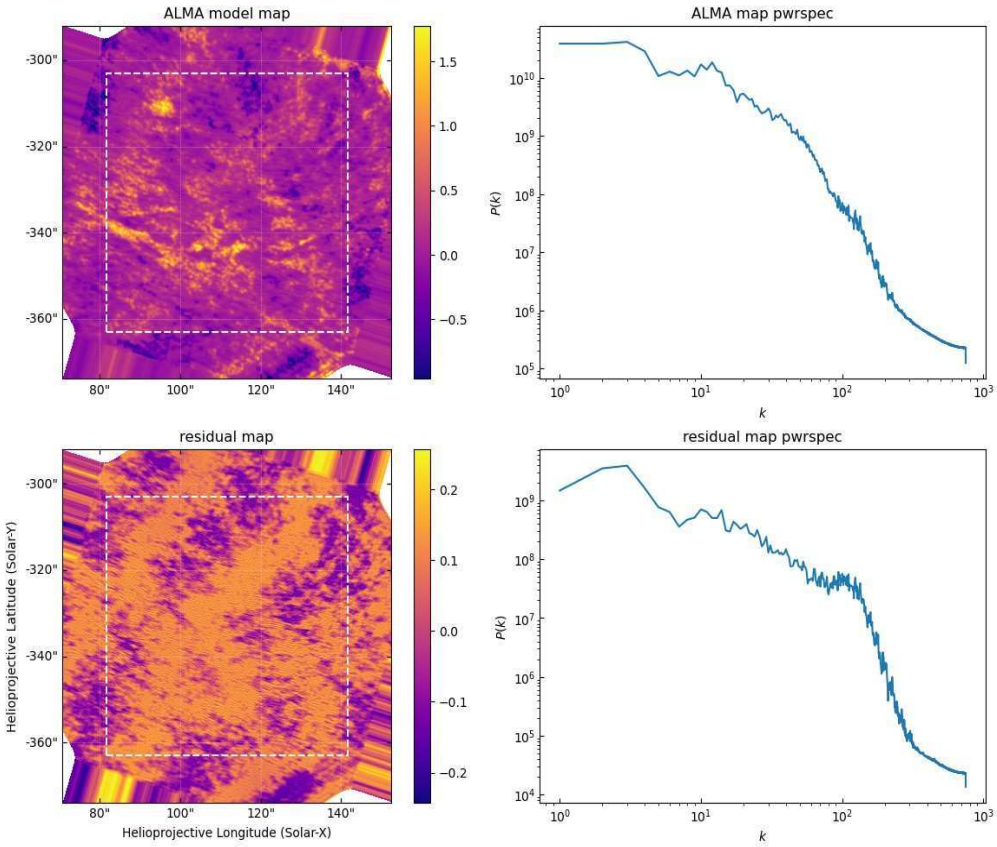


Figure 24: Fully-cleaned ALMA map and its residual, together with corresponding power spectrum.

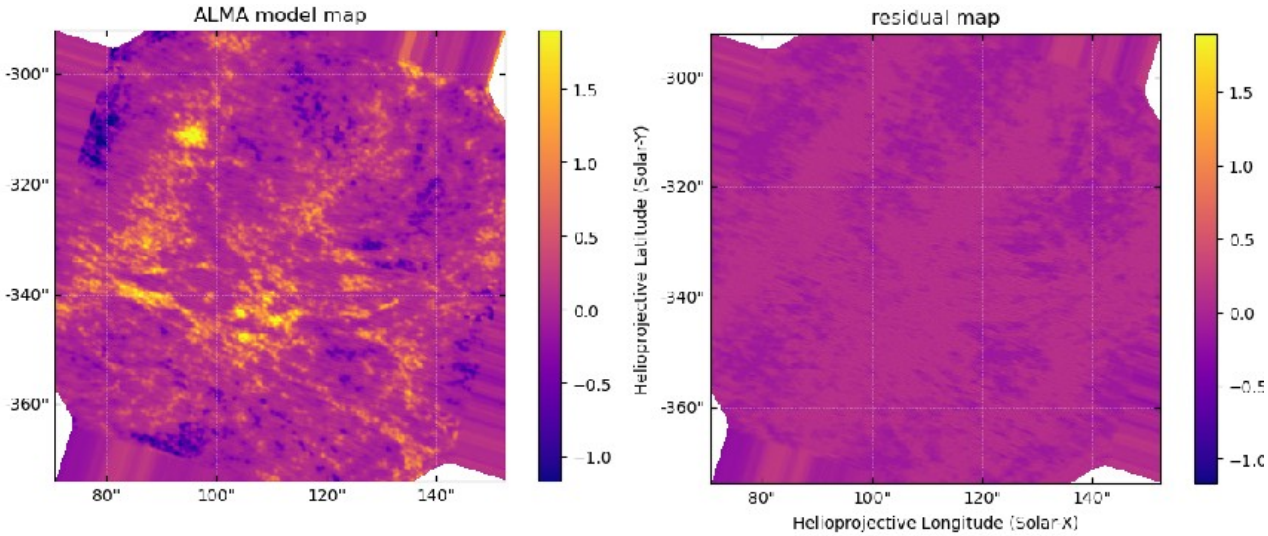


*Figure 25: Clean (left) and residuum (right) maps for the Band 3 data with appropriate number of iterations in **tclean()**. Essentially the left panels of Figure 24, now displayed in the same range of values (and corresponding color scale) .*

5.2 Replacement of the WVR signal by differential absorption of the solar mm radiation in Band 5: Indication of feasibility

Richard Hills estimations for relation between WV differential absorption and phase delay, as presented in Section 4.4, rely on the significant slope of the solar spectrum in the wing of the WV line at 183 GHz. As we are still in the middle of processing of the quite recent Band 5 data, just the preliminary results are presented here.

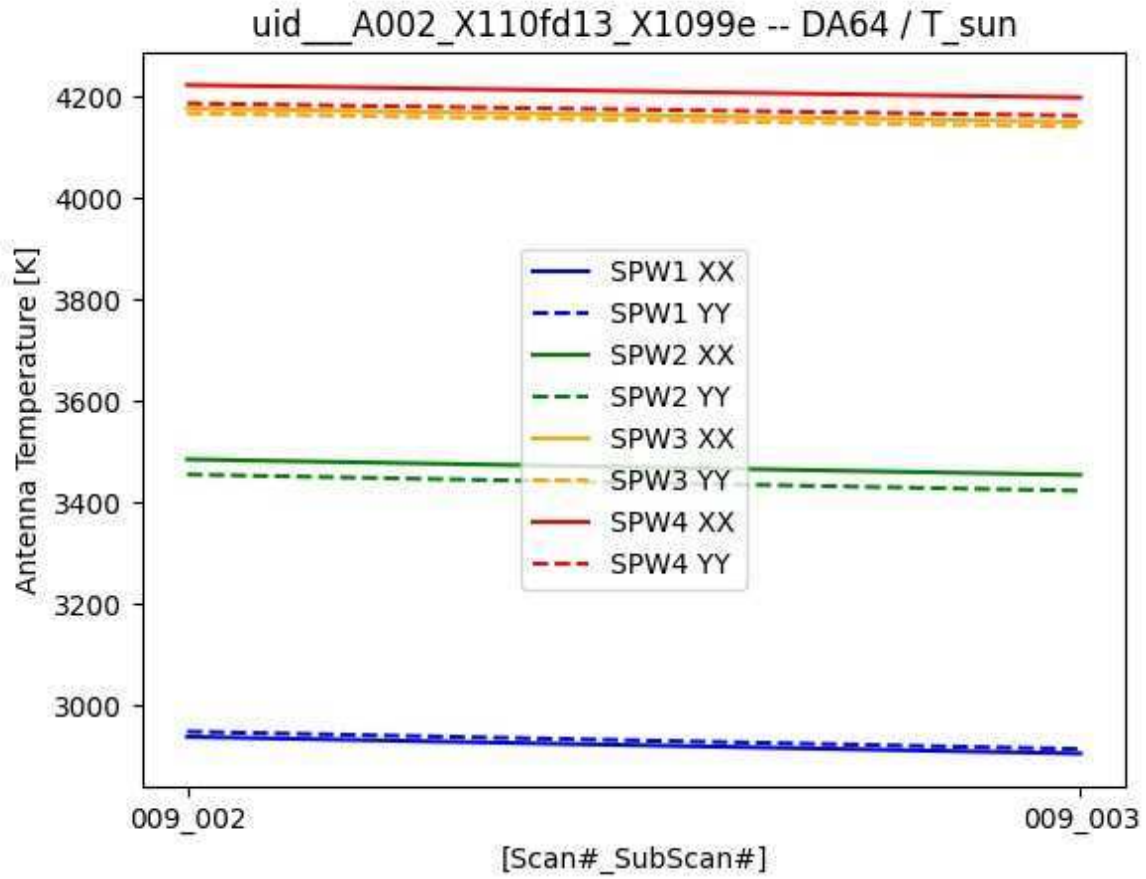


Figure 26: Spectral dependence of T_{ant} – differential influence of the WV line in spw0 and spw1 (in the legend of the figure as SWP1 And SPW2, respectively) is visible. The frequencies of the SPWs increases from SPW1 to SPW4, meaning that we are going apart of the WV line with increasing SPW id. The WV line is most pronounced in the SPW1, its wing partly covers SPW2, too.

An intermediate step in solar calibrations requires calculation of the antenna temperature T_{ant} (Shimojo et al., 2017). This is basically the brightness temperature of the target field averaged of the FoV, but – at this stage – before applied band-pass correction. Hence, it contains the influence of the atmospheric WV line on the spectrum formation. The following Figure 26 shows clear decrease of the T_{ant} in SPWs 0 and 1, as approaching towards the WV line core. This is preliminary indication of the spectral slope. As a next step, we shall resume the (now flagged) auto-correlations and try to recover the

spectral slope in its time dependence for each antenna. Its value is going to be correlated with the phase error as found from the self-calibration loop in Band 5 data (so far running).

6. Development side product: Solar Script Generator

Insofar, the solar ALMA observations are processed (in frame of the QA2) fully manually. It means that the calibration and imaging scripts are copied from one processed EB to another and used as template that has to be manually edited with each new analysis. Since Cy6 the Czech node of the EU ARC has introduced significant amount of automation directly into the scripts (finding observing band, specific calibrators fields and scans, automatic estimation of the image- and cell-sizes, etc.). However, still many parts requested attention of the data analyst, the script execution had to be interrupted at many break-points in order that the analyst manually finds and fills requested values (e.g., the flux density in **`setjy()`**). Moreover, the script itself got extended (almost twice) by this repeating 'service' lines of the code and became complicated and cumbersome.

Clearly, the way out has been found in taking these parts of the code that basically takes care of automated generation of those parts of the script, which depend on the specific processed EB, out of the script itself and create the *reduction script generator for the solar observations*, exactly as it has been defined in the collection of *Analysis Utils* (**`es.generateReducScript()`** by Eric Villard). This can be – in line with the historical development of the 'standard' data processing – considered as the first step towards the standard ALMA pipeline processing for the solar data.

Although this task does not belong to the goals of this development study, the new CSV datasets obtained as a part of the project, have helped us to finalize the development of the *Solar Script Generator* (SSG), which started already in 2020 but has been interrupted because of other, more pressing business and also lack of motivation (there were no EU solar observation to be processed between 2020 and 2023).

In spite of the clear inspiration by Eric's Villard script generator in the *aU* library, we have selected a bit more relaxed (or lazy) approach and used implementation based on the *static template* of the calibration script, which contains all the repeating code, not specific to any processed EB. This general lines of the codes are interleaved with the *keys* – firmly defined placeholder strings, put into the points, where the EB-specific part of the code should appear. The *script generator* itself then reads this static template and – based on the analysis of MS (meta)data – it generates a Python dictionary, which map the placeholder keys to the actual lines of the code. At the final stage it replaces the keys with the values and prints the final script into a file.

The code is available here: http://wave.asu.cas.cz/barta/sw/ALMA/ssg/ and it is self-documented by heavy commenting inside both the script and the template. Its usage (after properly setting the paths in the CASA init file) is similar to that of the reduction script generator in the *aU* module:

<in CASA>

```
import SolSG as ssg

sso = ssg.stuffForSolarDataReduction()

sso.generateSolCalScript(asdmName,doQA2=True)
```

Our motivation to finish this not fully related task as an auxiliary/preparatory activity within this study has been primarily given by increasing divergence between calibration and imaging scripts in the community of ALMA data (QA2) analysts: There existed multiple versions differing by the ARC and the last cycle that the script has been in usage. The reduction of the CSV data has been thus preceded by comparative research in the existing scripts. This unsatisfactory situation is clearly caused by low number of the solar projects observed – typically one per cycle world-wide (so only one ARC is performing solar QA2 per cycle). Hopefully, introduction of the SSG will not only boost the solar QA2 towards ALMA pipeline processing in the future, but it shall also help to keep coherence in the solar data reduction procedures across the ARCs and the consecutive ALMA cycles.

The SSG generator has been successfully used for preparation of the final version of calibration scripts for the CSV data reduction in this study and moreover tested on the older (2018) solar-prominence mosaic.

7. Conclusions and plans for the next period

The study is a bit behind half of its nominal duration, so the final results should yet come. But already now there are interesting preliminary outcomes:

- The simulations shows that **solar observations in Band 3 should be possible up to configuration C-7 and in Band 6 up to C-6**. More extended arrays (C-8 and beyond in Band 3) give poor results in reconstructed images even for the idealized limit of the zero atmospheric turbulence. Most likely, the dilution of the *uv* coverage mentioned in the Section 1, is already critical for more extended configurations and it is hard to invert the visibilities into synthesized image. Interestingly enough, the group at Uni Oslo (UiO) just tried to run their simulations for the preceding study oriented at high-cadence observations with the extended ALMA configurations, too and they found similar limits, just one 'size' smaller: C-6 for Band 3 and C-5 for Band 6. In our opinion, this small difference may be attributed to the different simulation setup; namely our simulations include **heterogeneous array** (12m + 7m, including mixed baselines), while the simulations at UiO included just 12m array (or perhaps, also 7m antennas, but separately). The higher number of baselines at the middle range in our mixed array might led to better performance in **`tclean()`** and, consequently, push the limit to the more extended configuration (see Figure 21). Nevertheless, it is more or less clear that longer baselines than those in C-7 are out of capability of the Solar ALMA ObsMode, at least with the so far used traditional CLEAN algorithm, including its sub-variants. We summarize the so-far obtained results in this direction in Figure 27

- The so far obtained **CSV observations** in Bands 3 (performed in the transitional configuration close to C-7) and 5 (done in C-6) **confirm the simulation results**, at least in the sense of the lower limit: It was possible to perform interferometric synthesis with those baselines and results of the self-calibration procedure in Band 3 revealed really great amount of details. Interestingly, the weather during the Band 3 observation, has been reported as just marginal. In spite of that, the self-calibrated series obtained form that observation are still scientifically valuable (we plan to publish the dataset as the CSV data, upon finished time-domain imaging and self-calibration) – this **opens the possibility to use the standard 'noGo' time for the simple solar observations** and boost the observatory science time, provided that self-calibration is included into the solar QA2 procedure.

- The results are also the **first observational proof that ALMA is capable of high-cadence observing**: The CSV data in both Bands 3 and 5 have been taken with an integration time of 192 milliseconds and yet calibrate and image well. The older CSV data taken for the preceding "High cadence" study did not pass QA2 (because of a bug with drifting/jumping phase-center with 30s period, related to the general-relativity correction), so these are the first high-cadence data ever.

- Albeit the **differential decrease of the antenna temperature** (basically the brightness temperature of the Sun as integrated over the PB *before* the BP calibration) **towards the line**

core of the WV is detected, we are still not able to correlate its time variations with the time-varying phase error for each antenna. For that we have to finish the self-calibration of the Band 5 series, which is – up to the report delivery – still running (and CPU-time demanding) process.

Clearly, many activities are yet in their run and we hope to present their results at the MT Review presentation and update the MT Report in response to the reviewers comments accordingly. This immediate task defines our work (that is actually already running) for the nearest future.

On the longer horizon of next two months we would like to concentrate on

- Replacement of the so far used ATMOS code for corruption of the simulated MS with the phase errors, by the new model of the phase errors introduced through the atmospheric-turbulence, kindly provided by Yoshi Asaki & Luke Maud. We received the code with documentation and training MS in December, basically at the same moment, when we got also the CSV data. For obvious reasons we have focused on the data processing first, postponing the simulation activity to the next period.
- Correlation between image fidelity (direct comparison between reconstructed image and its original unbiased model) and those metrics of the imaging quality (like power spectra of clean/residual), which are not dependent on apriori knowledge of the idealized input model. In the simulations we can always use both types of metrics for estimating the image quality, however, in the observations we will have just those, which cannot rely on comparison to the model. For this sake, the metric that gives the best coincidence to the direct comparison in the simulation case, shall be selected for the observed data, too.
- Improved performance in cleaning of the large solar images. As it has been described above, the large images (actually, the issue seems to be related rather to the broad spatial Fourier spectrum – presence of the flux at many spatial scales than to the size itself) makes the procedure of per-frame (per integration) imaging and self-calibration very (CPU-) time demanding. We shall explore broader range of parameters for the multi-scale and MTMFS cleaning than which has been so far used for in the ‘standard’ solar imaging. Another possibility we see in using manually-controlled multi-scale cleaning: First do the larger scales with a bigger cell-size and using *uvtaper*. Such low-resolution images shall also remain stable over multiple integrations/frames. This is achieved by time-scale separation between actual dynamical processes in the solar atmosphere and the (much faster) phase distortions caused by the Earth’s troposphere turbulence. The spatial and temporal dynamic scales in the solar atmosphere are roughly related by the sound speed: 1 arcsec in the image corresponds to ~ 700 km at the Sun and this spatial scale is changing at the timescale of ~20 seconds, if we take into account a typical chromospheric sound speed of 10 km/s and the rather safe Mach number in the shocks M=3. The pair-wise phase differences between antennas are contained even in the tapered image since no antenna is so far from the rest of the array that it has no baseline point in the tapered *uv* space. Hence, its phase delay is still corrected by the self-calibration procedure. This

low-resolution and low-cadence series, which should be much less CPU-demanding to obtain, will be subsequently used as an input model for the full-resolution (both space and time) deep cleaning. Since the flux at larger scales will be already removed, the small-scale cleaning should run also much faster.

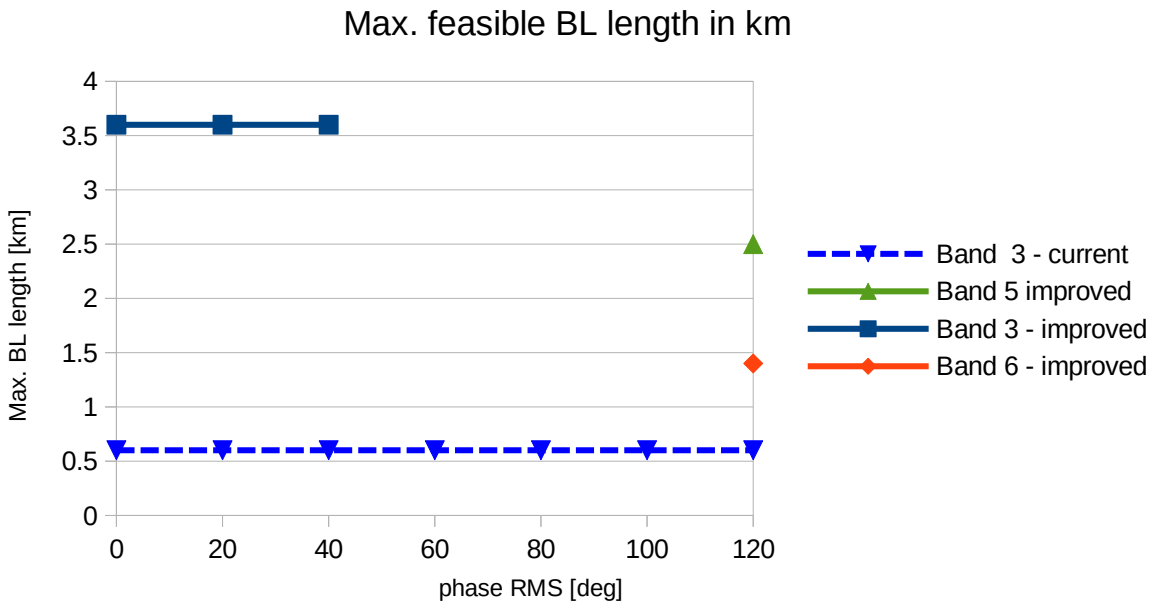


Figure 27: The so far obtained upper limits for solar observations if the fast-cadence sampling followed by post-processing that involves self-calibration loop is employed. Most of the points are base on the simulation results, the point for Band 3 at configuration C-7 (BL length of 3500m) at phase RMS 40 deg has been confirmed by the test observation data. For the idealized no-troposphere case (RMS = 0 deg) the simulation found that configuration C-8 in Band 3 is already beyond the plausible limit, therefore the nearest lower configuration C-7 with the longest baseline of 3600 m is displayed. However, it is possible that some hybrid configuration between C-7 and C-8 would still be manageable. The maximum baseline length for Band 5 and 6 corresponds to configurations C-6 and C-5, respectively.

References

[1] Bárta, M. (2019): *Towards high-resolution solar ALMA images*, ALMA Development Workshop, held 3-5 June, 2019 at ESO Garching, http://adsabs.harvard.edu/abs/2019adw..confE...4B

[2] Bastian, T. S. (2002): ALMA and the Sun, Astronomische Nachrichten 323, p. 271

[3] Bastian, T. S., Bárta, M., Brajša, R., Chen, B., Pontieu, B. D., Gary, D. E., Fleishman, G. D., Hales, A. S., Iwai, K., Hudson, H., Kim, S., Kobelski, A., Loukitcheva, M., Shimojo, M., Skokić, I., Wedemeyer, S., White, S. M., Yan, Y. (2018): *Exploring the Sun with ALMA*, The Messenger 171, p. 25

[4] Bastian, T. S. (2019): Solar ALMA Team communication.

[5] Benz, A.O.– ALMA DRSP (2006): in ALMA Design Reference Science Plan https://home.strw.leidenuniv.nl/~alma/review/drsp20_review.shtml

[6] Brajša, R., Skokić, I., Bárta, M., Bastian, T.S., Shimojo, M., White, S., Iwai, K. (2017): *Solar Research with ALMA – ESO Development Study,* Final Report, ESO

[7] Brajša, R., Sudar, D., Benz, A. O., Skokić, I., Bárta, M., De Pontieu, B., Kim, S., Kobelski, A., Kuhar, M., Shimojo, M., Wedemeyer, S., White, S., Yagoubov, P., Yan, Y. (2018): First analysis of solar structures in 1.21 mm full-disc ALMA image of the Sun, Astronomy and Astrophysics 613, p. A17

[8] ESO (2017): *ALMA starts observing the Sun*, ESO Press Release no. 1703, https://www.eso.org/public/news/eso1703/

[9] Fomalont, E. B. et al. – ALMA Partnership (2015): *The 2014 ALMA Long Baseline Campaign: An Overview*, The Astrophysical Journal 808, p. L1

[10] Heinzel, P., Berlicki, A., Bárta, M., Karlický, M., Rudawy, P. (2015): *On the Visibility of Prominence Fine Structures at Radio Millimeter Wavelengths*, Solar Physics 290, p. 1981

[11] Hills, R. (2019): Solar ALMA Team communication.

[12] Hudson, H. (2019): Solar ALMA Team communication.

[13] Karlický, M., Bárta, M., Dąbrowski, B. P., Heinzel, P. (2011): *Solar Research with ALMA*, Solar Physics 268, p. 165

[14] Loukitcheva, M. A., Solanki, S. K., White, S. (2008): ALMA as the ideal probe of the solar chromosphere, Astrophysics and Space Science 313, p. 197

[15] Maud, L. T., Fomalont, E. B, Tilanus, R. P. J., Dent, B. (2017a): *ALMA & Allegro Phase Metrics Workshop Report*, ALMA Memo No. 606 https://library.nrao.edu/public/memos/alma/main/memo606.pdf

[16] Maud, L. T., Tilanus, R. P. J., van Kempen, T. A., Hogerheijde, M. R., Schmalzl, M., Yoon, I., Contreras, Y., Toribio, M. C., Asaki, Y., Dent, W. R. F., Fomalont, E., Matsushita, S. (2017b): *Phase correction for ALMA. Investigating water vapour radiometer scaling: The long-baseline science verification data case study*, Astronomy and Astrophysics 605, p. A121

[17] Otarola, A.; De Breuck, C.; Travouillon, T.; Matsushita, S.; Nyman, L.-Å.; Wootten, A.; Radford, S. J. E.; Sarazin, M.; Kerber, F.; Pérez-Beaupuits, J. P. (2019): *Precipitable Water Vapor, Temperature, and Wind Statistics At Sites Suitable for mm and Submm Wavelength Astronomy in Northern Chile*, PASJ 131, pp. 045001.

[18] Shimojo, M. et al.. (2017): *Observing the Sun with the Atacama Large Millimeter/submillimeter Array (ALMA): High-Resolution Interferometric Imaging*, Solar Physics 292, p. 87

[19] Shimojo, M. (2019): Solar ALMA Team communication.

[20] Wedemeyer, S. et al.. (2016): *Solar Science with the Atacama Large Millimeter/Submillimeter Array—A New View of Our Sun*, Space Science Reviews 200, p. 1

[21] Wedemeyer, S., Bastian, T.S., Shimojo, M., and Bárta, M. (eds.) (2020): *Proc. of 1st International Solar ALMA Imaging Workshop in Oslo*, March 2-6, 2020 (in preparation as ALMA Memo)

[22] White, S. M. et al. (2017): *Observing the Sun with the Atacama Large Millimeter/submillimeter Array (ALMA): Fast-Scan Single-Dish Mapping*, Solar Physics 292, p. 88

[23] A. M. S. Richards, E. Moravec, S. Etoka , E. B. Fomalont, A. F. Perez-Sanchez , M. C. Toribio and R. A. Laing(2022): ALMA Memo 620, *Self-calibration and improving image fidelity for ALMA and other radio interferometers.* https://library.nrao.edu/public/memos/alma/main/memo620.pdf

[24] Luke T. Maud, Andrés F. Pérez-Sánchez, Yoshiharu Asaki, Felix Stoehr, Bill Dent, and María Díaz Trigo (2023): ALMA Memo 624, *Updates to ALMA Site Properties: using the ESO-Allegro Phase RMS database*. https://library.nrao.edu/public/memos/alma/main/memo624.pdf

Annex 1: Statement of work

Atacama Large Millimeter / submillimeter Array

ESO ALMA Support Centre

Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations

ESO-ALMA Document Number: ESO-395609

Document Version: 1

Document Type: Statement Of Work (SOW)

Released On: 2021-11-10

Document Classification: ESO Internal [Confidential for Non-ESO Staff]

Prepared by:	Yagoubov, Pavel	Released by ESO PDM SYSTEM
Validated by:	De Breuck, Carlos	
Approved by:	Testi, Leonardo	

Name

Authors

Name	Affiliation
P. Yagoubov	ESO

Change Record

Affected Section(s)	Changes / Reason / Remarks
All	First revision

Contents

1. Introduction 4
2. Scope 4
3. Related Documents 5
   3.1 Applicable Documents 5
   3.2 Reference Documents 6
4. Requirements 6
   4.1 Project definition 6
      4.1.1 Project Milestones 6
      4.1.2 Deliverables of the Study 7
   4.2 Tasks Applicable to all Project Phases 9
      4.2.1 Project Management & Control 9
      4.2.1.1 Project Management 9
      4.2.1.2 Project Reporting 9
      4.2.1.3 Progress Meetings and Reviews 10
      4.2.2 Configuration Control & Data Management 10
      4.2.2.1 Configuration Control 10
      4.2.2.2 Change Request Procedure 10
      4.2.2.3 Document Format and Identification 11

1. Introduction

The Atacama Large Millimeter/submillimeter Array (ALMA), an international astronomy facility, is a partnership of Europe, North America and East Asia in cooperation with the Republic of Chile. ALMA is funded in Europe by the European Organization for Astronomical Research in the Southern Hemisphere (ESO), in North America by the U.S. National Science Foundation (NSF) in cooperation with the National Research Council of Canada (NRC) and the National Science Council of Taiwan (NSC) and in East Asia by the National Institutes of Natural Sciences (NINS) of Japan in cooperation with the Academia Sinica (AS) in Taiwan. ALMA operations are led on behalf of Europe by ESO, on behalf of North America by the National Radio Astronomy Observatory (NRAO), which is managed by Associated Universities, Inc. (AUI) and on behalf of East Asia by the National Astronomical Observatory of Japan (NAOJ).

ALMA's primary function is to observe and image with unprecedented sensitivity and angular resolution the Universe at millimetre and sub-millimetre wavelengths.

This document outlines a work plan for the development of new Solar ALMA observing modes that would allow to overcome some of the current system limitations.

2. Scope

In the framework of the ESO “Advanced Study for Upgrades of the Atacama Large Millimeter/submillimeter Array (ALMA)” (CFP/ESO/19/25417/HNE), the present Statement of Work (SOW) is applicable to the:

Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations

This activity consists of the following elements:

1. Change of the Go/noGo procedure followed by the selfCal-based posterior phase corrections for times series of the high-cadence solar interferometric images. This shall be the main and universal approach. This approach will include the following activities:
    - Set up CASA-based simulations including atmospheric turbulence and add selfCal-based corrections of the fast phase variations. The simulations should be compared with the primary (model) images used as input.
    - Expand these simulations for longer baselines.
    - Verify the ability to separate the real and apparent dynamics of the Sun when using larger baselines.
    - Analyse the impact of sparser uv coverage in the larger configurations using these simulations.

- Determine the maximum yet acceptable configuration for each ALMA band depending on the phase metrics measured during the solar-specific Go/NoGo test, integration/sub-scan duration (cadence), and spatial setup (single pointings, size of small mosaics).
- Request, and if obtained, analyse short real test data (e.g. actual Go/NoGo test and science observation of the Sun) and verify/analyse if the altered go/no-go test is appropriate.

2. Utilization of the atmospheric WV line seen in absorption on the edge of the science SPW of Band 5. This is considered as a supplemental approach, which can provide further improvement, but it is usable just for the Band 5 data. This approach will include the following activities:
   - Identify an appropriate archival Band 5 dataset in configuration 2 to test the proposed procedure.
   - Adapt the mathematical model in wvrgcal() to relate the absorption of the solar continuum radiation to the water vapour column density and consequently to the atmospheric phase delay.
   - Apply the previous method to the analysis of existing Band 5 solar test observations in configuration C-2 or alternatively, new observation in configuration up to C-4.
   - Use archival non-solar Band 5 observations with strong bandpass calibrators to verify the water vapour column derived from absorption with those from the radiometer and compare both with the observed phase variations.
   - Optionally, verify if this technique can be used in other bands with strong water vapour lines such as the 324 GHz line in Band 7.
   - Based on the new Go/NoGo criteria for solar observations, determine the limitations of the present method, and specify the conditions in which it can be used.

The technical characteristics of the study are provided in the proposal [AD1].

3. Related Documents

3.1 Applicable Documents

The following documents of the exact issue shown form part of this SOW to the extent specified herein. In the event of conflict between the documents referenced herein and the present document, the terms of the Statement of Work shall be considered as superseding requirements.

AD1 CFP/ESO/19/25417/HNE “Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations”, Advanced Study for Upgrades of the Atacama Large Millimeter/Submillimeter Array (ALMA), Author: M. Bárta;

AD2 ALMA Documentation Standards

ALMA-80.02.00.00-003-G-STD

3.2 Reference Documents

The following documents, of the exact version shown herein, are listed as background references only. They are not to be construed as a binding complement to the present document.

RD1 Alma Documentation Control Plan

ALMA-80.02.00.00-011-F-PLA

4. Requirements

4.1 Project definition

ESO

The term “ESO” (European Organisation for Astronomical Research in the Southern Hemisphere) is the Executive Organization that has concluded with the Contractor for the Work.

Contractor

For all purposes related to this SOW and associated documentation, the term Contractor shall mean the Contractor’s firm, its employees, its paid and/or unpaid associates, its subcontractors and sub-subcontractors, consultants, suppliers, service providers and all other contracting and otherwise associated parties.

Technical Proposal

For the purpose of this SOW Technical Proposal means the Proposal [AD1].

4.1.1 Project Milestones

The work shall be organized according to the schedule milestones in Table 1. The milestone definitions being:

M0 Kick off. Starting of the activity

M1 Delivery of the Mid-Project review data package: in preparation for milestone M2 the data package as specified under deliverable DS1 shall be delivered to ESO

M2 Mid-Project review: based on the data package delivered under milestone M1, a review under the guidance of ESO will be held

M3 Delivery of the Final study review data package: in preparation for milestone M4 the data package as specified under deliverable DS2 shall be delivered to ESO

M4 Final Review: based on the data package delivered under milestone M6 a review under the guidance of ESO will be held

M5 Provisional Acceptance of the Documentation Package: provisional acceptance of the documentation package is granted once all actions from the Final Review, milestone

M4, have been resolved and Final data package delivered to ESO as specified under deliverable DS3.

Table 1 - Milestones

Milestones	Date
M0 - Kick off	T_0
M1 - Delivery of the Mid-Project review data package	T_0 + 11 months
M2 - Mid-Project review	T_0 + 12 months
M3 - Delivery of the Final study report review data package	T_0 + 23 months
M4 - Final Review	T_0 + 24 months
M5 - Provisional Acceptance of the Documentation Package	T_0 + 25 months

4.1.2 Deliverables of the Study

As part of the project, the contractor will deliver documents and support reviews as shown in Table 2. Final versions of all documentation shall be delivered at milestone M5 latest.

All the documents to be delivered within the scope of ESO contracts are under the managerial responsibility of ESO and shall be reviewed and/or approved by ESO.

Table 2 - Deliverables

ID	Deliverable description	Delivery for Milestone

DS1	Delivery of the Mid-Project review data package:
	Preliminary report containing results from both approaches:
	• Approach 1
	  ○ Setup of the simulation sequence in CASA for larger baselines;
	  ○ Comparison of the “blind-shot” selfCal-based correction results with the original unbiased model images;
	  ○ Measurement of the phase metrics from a changed go/no-go test on a short timescale;
	• Approach 2
	  ○ For Band 5, present analytical calculations/model that binds the absorption signal to the water vapour column density to the phase delay.
		M1

DS2	Delivery of the Final study review data package: Analytical study/report that will devise feasible (alternative) ways, which would enable solar ALMA observations with more extended configurations. For each of the alternatives the following shall be documented and compared to test observations where available: • Benefits/constraints: Maximum resolution (max. baseline length) reached vs. sci/tech (side-effect) constraints introduced by the implementation. The ultimate goal: Finding the maximum yet acceptable baseline length in dependence on (i) the observing frequency, (ii) the phase metrics as measured by Go/NoGo test prior to the observation, (iii) observing cadence, and (iv) spatial setup (single-pointing or a small mosaic); • Implementation requirements and suggested implementation procedure; • Estimation of implementation costs (albeit it may be only in the form of test time and manpower).	M3
DS3	Final data package: The Final data package shall consist of the documentation delivered under DS1 and DS2 but updated for the issues and actions agreed at Mid-Project review (M2) and Final Review (M4) respectively.	M5

4.2 Tasks Applicable to all Project Phases

This Chapter defines the basic tasks that are common to all project phases.

4.2.1 Project Management & Control

4.2.1.1 Project Management

The Contractor shall establish and maintain an effective project management structure to achieve the objectives of the Agreement.

The Project Office shall coordinate and control all technical, and contractual activities, the project resources, and manage all disciplines required for the successful completion of the Study. To be able to do so, the Project Office shall have effective control and support from appropriate senior management of the Contractor's organization.

The **Key Personnel** of the project shall consist of people with experience in the specific technologies and/or domains required by the scope of work. Appointment and exchange of key personnel, in particular Project Manager, during the execution of the Study shall be agreed with ESO in due time.

The Contractor shall assign a **Project Manager** with authority over the involved personnel and resources. The project manager shall be given full authority to negotiate with ESO and their Sub-Contractors and/or suppliers all issues related to the Agreement and shall be the **Single Point of Contact (SPC)** for all formal matters.

If events occur which could impact on the project milestones, e.g. technical problems or changes requests initiated by ESO or by itself, the Contractor shall evaluate every possible solution to avoid or limit the impact on the schedule.

4.2.1.2 Project Reporting

a) Progress Report

The Contractor shall provide ESO with Progress Reports, duly approved and signed by the Project Manager:

1) before the 5th of every second month following the project Kick-off.

The Progress Report shall clearly summarize the current status of the work in progress It shall briefly describe the results achieved in this period and planned for the next period. It shall also give an account on every problem detected at any level (i.e. technical, programmatic, finance) of the project during the reporting period(s) and identify the corrective actions.

The Progress Report shall also contain the Action Item List (AIL) and the status of all Action Items which shall also be reviewed at each progress meeting. The Progress Reports shall be sent by email to the ESO Technical Representative no later than three working days after the end of the reporting period. In the course of critical phases, ESO reserves the right to ask the Contractor additional reports.

b) Red Flag Report

Independently and in addition of to the regular Progress Reports, the Contractor shall report any event with significant implications on schedule, design or cost. A Red Flag Report shall be issued by the Contractor within 24 hours after occurrence of a major problem jeopardizing delivery, the achievement of the milestones defined in this Agreement, or the

achievement of technical performance and requiring the immediate attention of ESO. This reporting shall apply to problems at all project levels.

4.2.1.3 Progress Meetings and Reviews

Progress Meeting shall be held either at Contractor premises, at ALMA European premises, or by tele or video conference (to be agreed case-by-case) on a bi-monthly basis, in any case the week following the receipt of a monthly Progress Report by the assigned ESO Technical Representative. The **Agenda** and the **List of Participants** shall be communicated to ESO at least three days before the meeting. ESO reserves the right to call extraordinary Progress Meeting, one week in advance.

The purpose of the PM is to review the progress of the work, to highlight and discuss detected and reported problems or concerns, and to determine corrective measures to be taken.

Unless specifically requested otherwise by ESO, the Contractor shall write the **Minutes of Meeting (MOM)**. The Minutes shall be signed by all the parties at the end of the Meeting. The signature of such minutes indicates solely that the wording is correct and properly reflects the outcomes. The signature shall however not be construed as a formal, contractual agreement. Any matter having contractual implications shall be handled in accordance with the regulations of the Agreement. The Minutes shall include a **List of Action Items (AI)** with appropriate deadlines and responsible, and shall be under configuration control.

The Contractor shall plan and prepare project reviews in consultation with ESO, particularly the location, agenda, and participants of the reviews will have to be agreed.

The following reviews shall be held:

- The Mid-Project review (M2) is a review to assess the work performed in the first phase of the study;
- The Final review (M4) is a review to assess the work performed in the second phase of the study.

A review is closed when all issues and comments raised by the review board and accepted by ESO have been taken into account.

4.2.2 Configuration Control & Data Management

4.2.2.1 Configuration Control

The plan for the management of the ALMA documentation is provided for information in [AD2]. The Contractor is invited to adhere to this plan. As part of it, the Contractor shall maintain the Configured Items Data List. The CIDL identifies all the official and valid documentation (“configured items”) of the Project. It is a key element of the Configuration Control process and for the particular project shall include the list of project reports (e.g. included in Review Data Packages.

4.2.2.2 Change Request Procedure

Both parties, the ESO and the Contractor, may at any time request a change, which may affect the Contractual Conditions:

- Agreement

- Statement of work
- Documentation
- Schedule

In response to a Change Request initiated by ESO, the Contractor shall return a reasonable and justified Change Proposal within 4 weeks (subject to negotiations between parties), and, if agreed by ESO, implement the change defined in the requested change. In the case of a Change Request initiated by the Contractor, ESO have no obligation to accept the proposed change and will, in any case, provide a reply to such requests within 4 weeks. In any case, the implementation of a Change Request shall not start prior to the written authorization or approval by ESO in accordance with the contractual requirements.

When preparing a Change Request, the Contractor shall investigate the proposed change and provide ESO with all information for a decision, including, but not limited to:

1. The reason for the change.
2. An assessment of the technical feasibility, including an analysis of all the impacts on the products and on the total system (i.e. performance, specifications, interfaces, etc.).
3. The necessary modifications of the contractual documents (Statement of Work,) shall be clearly indicated by quoting the OLD and NEW versions of the text and/or the drawing.
4. The total schedule impact, i.e. on any contractual milestones.
5. The total cost impact, giving differential costs changes on labor, operations and capital, for each affected Work Elements/Work Packages.
6. Other related factors such as reliability, safety, integrity, maintenance, etc.
7. Any additional documents deemed necessary.

A template of ALMA Change Request is provided on the ALMA document server (EDM or its replacement). The procedure for the submittal and processing of Change Request is given for information in [RD1]. Each proposed change must be identified by an individual and unique number, which shall be used by all subsequent correspondence as reference. The Contractor shall keep a record with all initiated changes with their status (approved, rejected, pending, closed) and provide an updated status together with the Progress Reports.

4.2.2.3 Document Format and Identification

All written documentation and information issued by ESO will be forwarded to the Contractor Project Office for further distribution within the Contractor organization as necessary.

The project documentation shall comply with the document standards described in [AD2].

--- End of document ---

Annex 2: Proposal – Scientific & technical description

Call for Proposals CFP/ESO/19/25417/HNE

ALMA DEVELOPMENT STUDIES 2019

TOWARDS HIGH-RESOLUTION SOLAR ALMA IMAGES

Overcoming current ObsMode limitations

Revised version

as a follow-up of the *1st International Solar ALMA Imaging Workshop* held in Oslo, March 2nd – 6th, 2020

PRINCIPAL INVESTIGATOR:

RNDr. Miroslav Bárta, Ph.D., Head of the Department of Solar Physics

e-mail: barta@asu.cas.cz
Telephone +42 0 323 620 324

Address: EU ARC – Czech node, Astronomical Institute of the Czech Academy of Sciences, Fričova Str. 298, CZ-25165 Ondřejov, Czechia

BIDDER:

Name: Astronomical Institute of the Czech Academy of Sciences, represented by its director Prof. RNDr. Vladimír Karas, D.Sc.,

Address: Astronomical Institute of the Czech Academy of Sciences, Fričova Str. 298, CZ-25165 Ondřejov, Czechia

e-mail: vladimir.karas@cuni.cz, director@asu.cas.cz
Telephone: +42 0 323 620 116

TOWARDS HIGH-RESOLUTION SOLAR ALMA IMAGES
Overcoming current ObsMode limitations

Preamble to the revised version

This proposal for *ALMA Development Study* is a revised version of the project submitted in September 2019. The revision has been motivated by an ESO clarification request with the purpose to:

(i) Identify the best up-to-date practices and methods for accomplishing the project objectives, based on outcome of the discussion held during the *1st International Solar ALMA Imaging Workshop in Oslo* in March 2020.

(ii) Down-select the number of alternative approaches suggested in the original proposal, in order to focus right on the few most promising ways as identified in (i).

(iii) Describe the methods used for the project solution and resources requested for it (namely in terms of the EoC test time and manpower at ALMA Observatory) in more detailed and specific way.

As a result of the Workshop summary and subsequent project team discussion, we decided to concentrate only on two approaches:

1. Change of the *Go/noGo* procedure followed by the selfCal-based posterior phase corrections for times series of the high-cadence solar interferometric images. This shall be the main and universal approach.
2. Utilization of the atmospheric WV line seen in absorption on the edge of the science SPW of Band 5. This is considered as a supplemental approach, which can provide further improvement, but it is usable just for the Band 5 data.

Alternative approaches (Items 2,3,5, and 6 in the original proposal) are abandoned in favor of detailed investigation of the two above selected (Items 1 and 4 in the original proposal). Relevant parts of this revised proposal have been upgraded accordingly to this decision and are described in detail below. A new section *"Other requested resources"* have been appended at the end of this document. Requested EoC time and help of ALMA/ESO staff is specified and summarized there. Because of increased focus on *Approach 1*, the core team has been strengthened by relevant expertise of L. Maud (ESO) and S. Wedemeyer (Uni Oslo).

Summary

Goal of the project

Solar ALMA ObsMode is currently severely limited to compact array configurations only, allowing observations with max. baselines corresponding to C43-3 (C43-4 in Band 3). This fact limits the resolution to ~1", preventing a lot of key solar research with ALMA. The project aims at breaking the limitations – we plan to design and in-detail investigate possible ways for an upgrade of the *Solar ALMA Observing Mode* that would overcome its current unsatisfactory status. As a result, we shall get the new range of possible configurations depending parametrically on the observing ALMA Band and atmospheric conditions measured as phase-error metrics.

Timeline

Total duration: Two years from the kick-off of the project. Mid-term review after one year.

Requested resources (from ESO/ALMA)

- Personal costs for one 100% FTE dedicated researcher (postdoc) for entire duration of the project (2 years) + 0.4 FTE to support the effort in the involved institutions.
- Mobility/cooperation travel support (between involved institutions; visiting ESO).
- Allocated testing/EoC time (~2 – 3 hours) at ALMA.

Deliverable

Analytical study/report that will devise feasible (alternative) ways, which would enable solar ALMA observations with more extended configurations. For each of the alternatives we plan having documented its

- Benefits/constraints: Maximum resolution (max. baseline length) reached vs. sci/tech (side-effect) constraints introduced by the implementation. The ultimate goal: **Finding the maximum yet acceptable baseline length in dependence on (i) the observing frequency, and, (ii) the phase metrics as measured by *Go/NoGo* test prior to the observation.**
- Implementation requirements and suggested implementation procedure.
- Estimation of implementation costs (albeit it may be only in the form of test time and manpower).

The study is intended as a guide and starting point for the later actual implementation of the *Solar ALMA Observing Mode* upgrade.

Motivation – Overcoming the current-state limitations

High angular resolution is undoubtedly one of the key and prominent qualities that ALMA provides for scientific observations. Indeed, already since the successful *Long-baseline CSV campaign* (ALMA Partnership – Fomalont et al., 2015) many branches of astrophysics are greatly profiting from break-through discoveries allowed by ALMA observations at extended configurations – let us name, among others, at least the example of unique collection of observations of protoplanetary discs, which got broadly known even in the general public.

The solar scientific observations have been planned already from the very beginning of the ALMA project preparation and already the early-stage relevant documents (e.g., Bastian, 2002; *ALMA DRSP* – Benz, 2006; Loukitcheva et al., 2008, Karlický et al., 2011; Heinzel et al. 2015; Wedemeyer et al., 2016) mention many key science cases, for which the high angular resolution is essential. Again, let us name just one example – the long-standing mystery of the mechanism for heating of upper layers of the solar atmosphere: Since actual dissipation of energy takes place at small scales, high-resolution is essential for this (and many others) key research problem.

The *Solar ALMA ObsMode* has been officially approved in December 2016 (also thanks to the EU ARC contribution by the project "Solar Research with ALMA" carried out in frame of the previous ESO *ALMA Development Study* programme) and the Cycle 4 thus allowed the first solar science with ALMA (ESO, 2017; Brajša et al., 2017, Shimojo et al., 2017; White et al. 2018; Bastian et al., 2018). In order to enable the promised solar science with ALMA to the community soon, the *ObsMode* has been offered as a "*non-standard"*, with many limitations (compact arrays, Bands 3 and 6 only, TDM, no polarization, etc.).

One of the most severe constraints is the maximum baseline length ~500m – the array configurations allowed for the mode are up to C43-3 (exceptionally C43-4 in Band 3 since Cy7), what prevents solar physicists of profiting from the high-resolution ALMA capability, which brought so much new results in other astrophysical research. Where this limitation for solar observations comes from? First, already the fact that we use the day-light observing time naturally brings higher values of PWV and, consequently, the higher phase RMSs. For the non-solar targets (even those observed at day time) there is, nevertheless, a possibility – used in the standard calibration procedure – to correct the fast phase variations between the phase-calibrator scans via water-vapour radiometers (WVRs) measurements and posterior *wvrgcal()* procedure. For the Sun this is, however, impossible as the brightness temperature of the Sun is much higher than contribution of the atmospheric water vapour, the WVRs get even saturated. Since we are lacking possible correction mechanism for fast phase variations in the solar *ObsMode*, the standard-interferometry *Go/noGo* tests based on phase RMS measurements (on a long time scale) preclude solar observations with longer baselines.

Many solar science cases, for which the resolution is not so critical, has been already successfully observed – repeating it with merely different (but by nature the same) target will likely bring no new information. Since new capabilities including the high-resolution observations are not being offered, some frustration and loss of interest in solar ALMA observations is visible inside the solar physics community; this can be seen in a significant drop of solar-oriented proposals during the last Cycles. On the other hand, there is a strong call for the high-resolution solar ALMA observations among researchers. Addressed by this challenge we propose the corresponding upgrade of the Solar *ObsMode*. The project aims at finding the possible ways out of the above mentioned limitation and the deep investigation of their feasibility, implementation requirements, limits of their applicability, and estimated implementation and running costs (in broader sense, e.g., loss of another quality). There are already a few ideas on the table that have emerged within the *Solar ALMA Development Team.* These have have been recently discussed during the Oslo Workshop in March 2020, namely in frame of the short topical session devoted to this project. From around six possible approaches (partly overlapping), the two described in the next section were identified as the most promising and selected for further detailed elaboration. Nevertheless, we expect improvements and new ideas to appear, when the topic will be really focused and systematically investigated in frame of the project.

Upgrading the *Solar ObsMode*: Most promising approaches

The goal of this study is to present detailed analysis of two realizations (one "main" and one "supplemental") for the *Solar ObsMode* upgrade in the sense of higher angular resolution with all their strong and weak aspects, implementation requirements and procedure, and estimation of the implementation costs. It shall be the responsibility of ALMA and its regulating bodies which, and if any, of those suggested ways will be eventually implemented. Here we present the ideas that already emerged and have been discussed within the project team, but definitely need to be

investigated in detail in frame of the proposed project. We cannot, however, preclude that another smart solution to this problem emerges during the project execution.

1. Alternation of the *Go/noGo* procedure for solar observations and posterior selfCal-based phase corrections in high-cadence interferometric images

Specifics of the solar observations (namely the high brightness of the Sun) bring many difficulties, including lack of the standard WVR corrections for the fast phase variations, what is essential with respect to the goal of high-resolution solar images. On the other hand, they can also yield opportunities that are impossible in the standard interferometry of relatively weak targets. Namely, the high brightness temperature of the Sun breaks – sometimes unconscious – "paradigm" of interferometric observations, which claims that we need rather long observing (total signal integration) times for reaching sufficiently high S/N ratio. Since the image integration in the "standard" interferometry is being performed over period largely exceeding the time-scale of the phase variations, the resulting image is *blurred* and the resolution (long-duration coherence) is lost. On the other hand, with the Sun, which provides enough signal even for the shortest so-far possible integration times (0.16 s as per current correlator limit), we can get *series* of relatively sharp (see below) but *distorted* images instead of a single blurry image (Bárta 2019). Of course, the distortion is also varying on the time-scale of the (Earth) atmospheric turbulence, but the apparent motion (and the brightness variations) can be to some extent distinguished and separated from the real dynamics of the Sun: The large-scale solar structures change at much longer timescale (minutes) than the integration time (a fraction of seconds). For this sake, there is already well established and tested method of *Self-calibration with cascading time-scale separation*: Self-cal is first done for the solution intervals covering entire scan, then its halves, quarters, and so on, till the (atomic) integration time scale. Its results in the form of impressive solar ALMA movies, made from the series of self-calibrated per-integration images, have been presented at the Oslo workshop in March

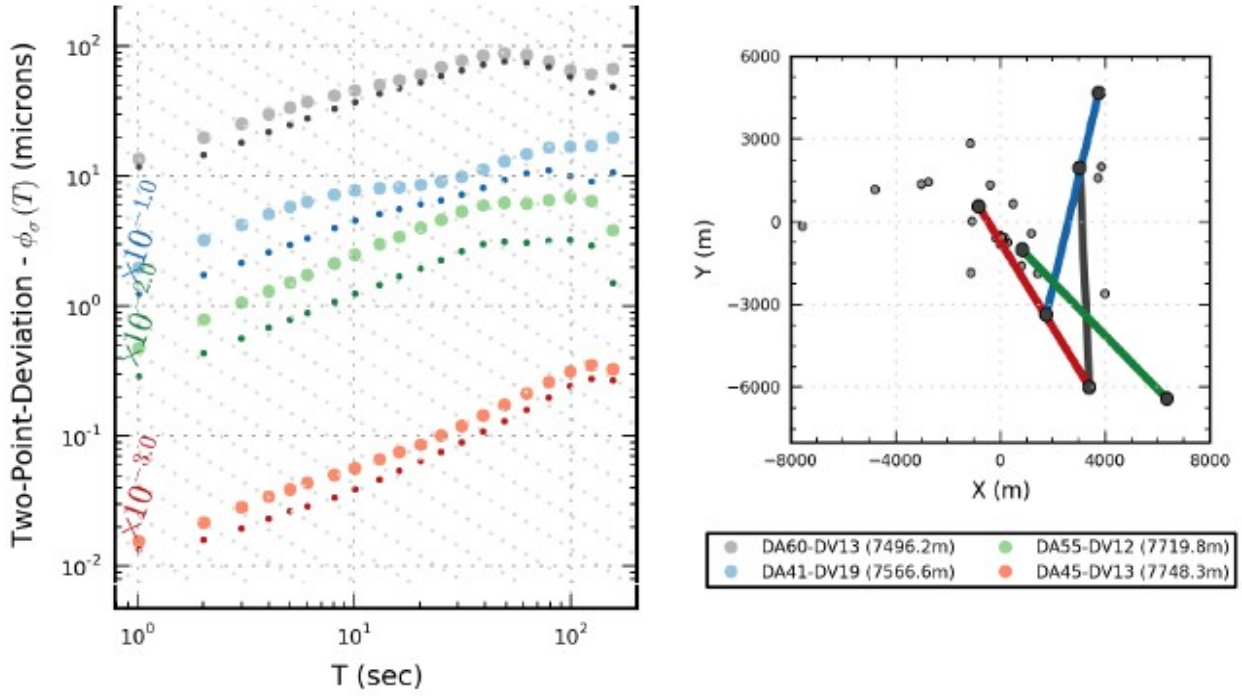


***Fig. 1**: The fast phase variations measured by means of two-point deviations (TPD) for four selected baselines vs. the timescale. Adopted from Maud et. al. (2017b). Raw (big bullets) and WVR-corrected (small ones) data have been used as an input.*

2020 (e.g., Wedemeyer+Szydlarski, Shimojo, Bastian – Workshop summary is planned to be published as an ALMA Memo).

There is also a couple of studies of the fast phase variation, their influence to the coherence and image quality and counter-acting measures – e.g. Maud et al. (2017a, 2017b). The authors found – among other results – that the phase variations measured via the metrics of the *two-point deviations* (TPD) decrease strongly with the time lag between the two points of phase-error measurements – see Fig. 1. Putting it another way: The fast variations are not so harmful for the short integration times (i.e., the *sub-scan* duration in the solar mode). The authors of the ALMA Memo no. 606 (Maud et al., 2017a) even literally recommend to use *short* Go/noGo test, with the time-scale corresponding to the sub-scan time. The Sun is pretty bright, so the sufficient duration for collecting the enough signal is on the sub-second timescale. The *Go/noGo* test, which uses band-pass calibrator, likely will need a bit longer duration to reach sufficient S/N ratio in order to provide trustworthy results of the test, however, we should tend to as short time-scale as possible for the test, in order to approach the real timing in the subsequent solar science observations.

To sum up: It is well possible that we have made the (basically non-existent) problem of "impossibility" of solar observations on the longer baselines ourselves, just because applying the standard (long time-scale) *Go/noGo* phase-variation test in an inappropriate situation. Should the proposed study show that the above is correct, merely relaxation from the too strict *Go/noGo* test for solar observations and postponing the problem of distorted images to the post-processing (solar-specific *self-cal* loop) would be the simplest solution of the entire problem. However, detailed investigation via simulations and real testing is required. The details are described below in *"Methods"*. We plan to focus namely at this approach because it could work universally, for all ALMA observing bands.

2. Using absorption water lines in science SPWs of Band 5

Because of low additional "costs" (the Band 5 solar data are already there, we can just analyze them within the team with no demand on the EoC time nor help of the ALMA staff), we suggest a supplemental approach for the Band 5 solar data, which could, to some extent, further improve the fast phase-error corrections similarly like WVRs do so for the non-solar data.

Atmospheric water line centered at 183 GHz partly (by its wing) reaches to the science SPWs of the Band 5. This fact, on one hand, degrades some frequency channels in those SPWs (BB1 in the currently used setup with the LO frequency set to 198 GHz), on the other hand, it brings the possibility to use those channels for measuring of the absorption of solar continuum in the water vapour directly (Hills 2019). This could be used for estimation of the WV contents without using the WVRs. The signal is not saturated in this case – the science SPWs are attenuated electronically via the MD technique. Instead of that we face another problem: Adaptation/generalization of the mathematical model, which in the standard *wvrgcal()* ties the water-vapour *emission* with the phase delay, in the way that *absorption* of the solar continuum radiation in the atmospheric water vapour is related to the water-vapour column density and consequently to the atmospheric phase delay.

There have been several methods suggested for recalculation of measured signal variations to the phase delay in Hills (2019), e.g., total power at a single freq. channel, in the entire SPW, or in an entire sideband, or detailed analysis of (the slope of) the auto-correlation spectra. Hills (2019) also estimated sensitivity of the TP variations measurements to phase delay vs. frequency for various set of other parameters (receiver gain stability, PWV, etc.) – one such example is shown in Fig. 2. This

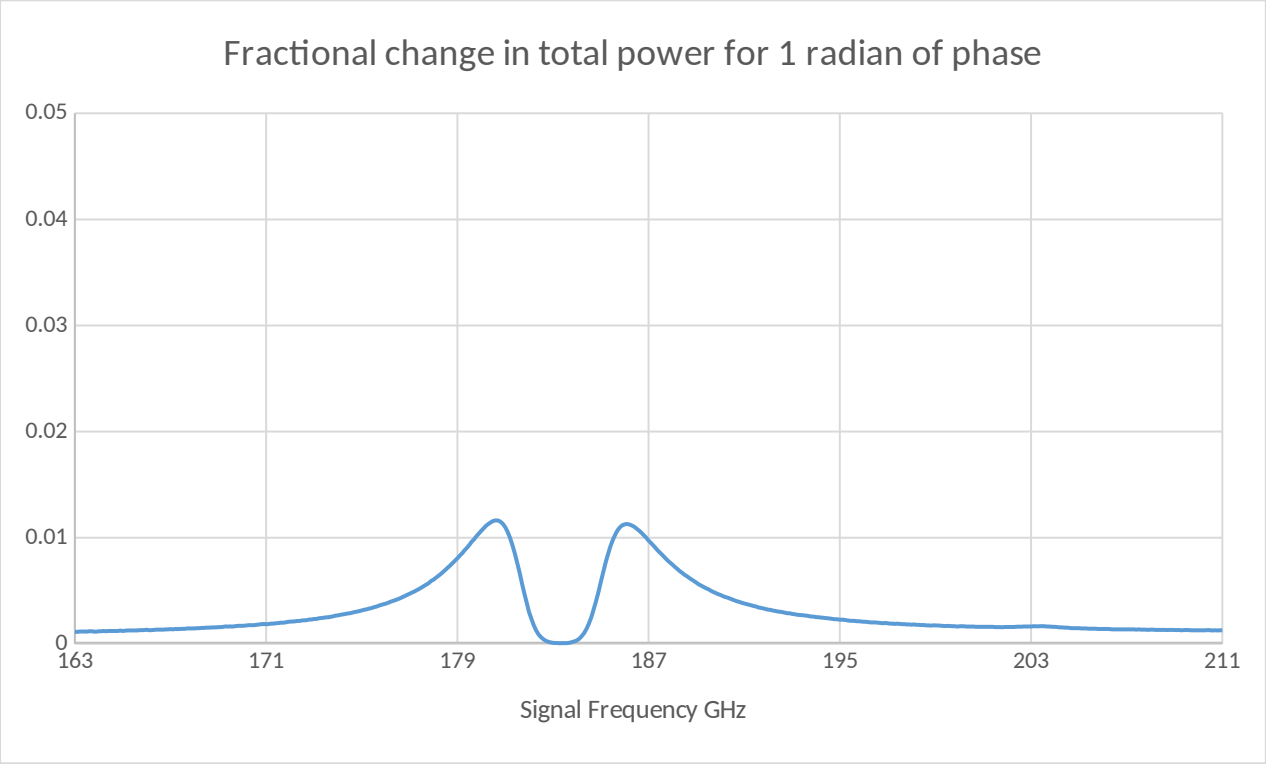


***Fig. 2**: Response of total power in Band 5 to the increase of PWV causing the phase delay of 1 rad. From Hills (2019).*

method needs further elaboration and testing, but has no demands on hardware change, EoC time nor ALMA staff help.

Methodology

The *Approach 1* will be investigated dominantly by means of the CASA-based simulations, which shall include atmospheric turbulence mixed with the (simulated) proper solar dynamics, followed by the *selfCal*-based corrections of the fast phase variations. The solar-specific self-calibration loop, which is essential for correction of the series of the distorted images, is already now quite well developed for the single-pointing data (and work is underway for small mosaics), the most intensive effort being done at Uni Oslo in frame of the currently performed ALMA Development Study "High-cadence solar ALMA imaging" (PI S. Wedemeyer). In this respect, the currently proposed project will follow up the results of this on-going study. CASA-simulated solar ALMA visibilities will be created for longer-baseline arrays and realistic atmospheric conditions. Time series of self-calibrated maps will be then compared with the primary (model) images used as an input to CASA simulations. We shall concentrate not only at our ability to separate the real and apparent dynamics of the Sun when using longer baselines, but also resolve the issue of yet more sparse *uv*-space coverage, which becomes more critical for extended configurations. This can make a problem for the solar images as they naturally contain emission structures at all possible scales from the pixel size to the size of FOV. Using the simulations we shall **find the maximum yet acceptable configuration depending parametrically on the observing ALMA Band and phase metrics measured during the solar-specific *Go/noGo* test**. After the optimum procedure for self-cal loop shall be identified in the simulated data, which would enable the maximum possible baseline lengths, we would like to try it also on the real test data – very short sequence (~10 minutes of on-

source time per Band) of solar observation would be sufficient. Of course, the altered *Go/noGo* test have to be tried, too. We discussed this possibility during the Oslo workshop – the outcome is that such a test can be done during any ALMA technical break (e.g. because of array configuration change – A. Hales, workshop discussion).

The supplemental *Approach 2* needs substantial modification of the *wvrgcal()* underlying model and procedure – this will be done in collaboration with its authors/developers. Tests, to what extent the procedure helps, can be done on already existing solar Band 5 data in C-2 configuration. Having another set of solar Band 5 data taken at the longer baselines (> C-4) would be helpful, but not vital. Instead of that, we shall search for appropriate archive Band 5 data of non-solar observations and compare the WV-radiation (from the WVRs) and WV-absorption seen in the data for the band-pass calibrator, with the phase variations of the same body.

The team and its resources

The project team is formed by its core, represented by researchers working more intensively on the topic, and external collaborators, whose particular expertise will the core-workers occasionally request for consultations.

The core:

- Miroslav Bárta (PI), Astronomical Institute, EU ARC.CZ, Ondřejov, Czechia
- ***Full-time dedicated researcher***, Astronomical Institute, EU ARC.CZ, Czechia
- Tim Bastian, National Radio Astronomy Observatory, USA
- Roman Brajša, Hvar Observatory, Faculty of Geodesy, University of Zagreb, Croatia
- Dale Gary, New Jersey Institute of Technology, USA
- Antonio Hales, Joint ALMA Observatory, Chile
- Richard Hills, Cavendish Laboratory, Cambridge, UK
- Hugh Hudson, School of Physics and Astronomy, University of Glasgow, UK
- Luke Maud, ESO, Garching, Germany (phase variations, phase metrics)
- Masumi Shimojo, NAOJ, Tokyo, Japan
- Ivica Skokič, Hvar Observatory, Faculty of Geodesy, University of Zagreb, Croatia
- Sven Wedemeyer, Institute of Theoretical Astrophysics, University of Oslo, Norway

The *Full-time dedicated project researcher* has to be appointed first, we plan to publish an open call for the position in case of this proposal gets approved. This new position is essential for the project success as the other core members – though having good expertise in the topic (but at the same moment frequently busy with other duties) – cannot devote their time entirely to the detailed, particular or routinely project tasks.

External collaborators:

- Shin'ichiro Asayama, East Asia ALMA Support Center, Tokyo, Japan
- Bin Chen, New Jersey Institute of Technology, USA
- Bart De Pontieu, LMSAL, USA;
- Gregory Fleishman, New Jersey Institute of Technology, USA
- Akihiko Hirota, Joint ALMA Observatory, Chile
- Kazumasa Iwai, National Institute of Information and Communications Technology, Japan
- Sujin Kim, Korea Astronomy and Space Science Institute, Daejeon, Republic of Korea
- Neil Philips, ESO, Garching, Germany

- Tsuyoshi Sawada, Joint ALMA Observatory, Chile
- Giorgio Siringo, Joint ALMA Observatory, Chile
- Stephen White, AFRL, USA
- Pavel Yagoubov, ESO, Garching, Germany
- Yihua Yan, NAO, Chinese Academy of Sciences, Beijing, China.

The entire Team (core + external experts) is formed from the members of the *Solar ALMA Development Team* (e.g., ESO, 2017) – the researchers, who actively took part in the basic development of the *Solar ALMA ObsMode*. Moreover, we have asked also Dr. Luke Maud (ESO) for kind sharing of his expertise on phase-variation metrics. The *Solar Team* is still active, having regular bi-weekly telecons (led by Tim Bastian) and sharing the documents (see in *References*). Moreover, in addition to the development tasks, the *Solar Team* members are involved in organization of ALMA solar scientific observations and regularly work on the solar data processing/ QA2 (entirely manual data reduction). To sum up, there is a good amount of required expertise in the current-project Team: For example, the PI (M.B.) worked as the project manager of the previous ALMA Development Study "Solar Research with ALMA" (and overtook the role of the bidder-representative, after Prof. M. Karlický had retired), R. Brajša was the PI of that project, I. Skokič the dedicated full-time researcher. T. Bastian and M. Shimojo have been leading the NA ARC and EA ARC participation, respectively, R. Hills is one of the leading experts on ALMA hardware and technology, S. Wedemeyer is leading related project, whose results will be very useful for accomplishing the goals of this one, etc. The bidding institution will also take advantage of being involved into EU ARC and ESO for direct cooperation with the ESO-based researchers, also with those not explicitly listed above.

The collaborating institutes are well equipped with resources required for carrying out the project, namely a hardware for the test-data processing and CASA simulations – e.g., the Czech node of the EU ARC in Ondřejov (EU ARC.CZ) has dedicated high-performance cluster for ALMA data processing (used for the QA2 services) and 10 years lasting experience with CASA.

Expected timeline

If approved, the project would come into its preparation phase. In this period the core team will search for a suitable candidate for the open postdoc position. After filling the position the project is expected to start by the *Kick-off Meeting* at ESO, where the details could be yet negotiated. The mid-term review is planned one year after the project kick-off. Till then we plan to have finished procedures for generating CASA-simulated series of high-cadence images with longer (~ 3 – 5 km) baselines for *Approach 1* and comparison of their "blind-shot" selfCal-based correction results with the original unbiased model images. We shall also have measured the phase metrics from changed *Go/noGo* test on a short timescale. In parallel we shall work on the analytical calculations/model that binds absorption signal of the WV line to the phase delays (*Approach 2*).

The 2nd half of the project is planned to be devoted to more systematic parametric study of *Approach 1* with the aim to find *optimized procedure* and maximum possible extent of the array configuration for a given observing frequency and phase RMS as measured by the specific solar s*hort time-scale Go/noGo* test. In parallel we shall apply *Approach 2* to the existing Band 5 solar data and study its effect. Also the non-solar archive Band 5 data for strong band pass calibrators (if some suitable are found) shall be used for comparison among (i) WVR data, (ii) Absorption in science SPW in the WV-line wing, and (iii) actual phase variations found for the band-pass calibrator. Finally, the *Approach 1* shall be tested on the real EoC data – we shall verify at least

a few points of limiting line for the allowed baseline length in the parametric space that spans over the observing frequency and (short-timescale) phase RMS.

At the final review (2 years after kick-off) the study serving as a background for further decision on upgrade of the *Solar ObsMode*, shall be delivered. Last but not least: Focusing at the topic for the first time more systematically in frame of a dedicated project, new, unexpected, ideas on how to overcome the current limitations of the solar mode, may emerge.

Project costs

Requested financial support from ESO for the project are planned to cover namely: (1) The dedicated postdoc researcher, working at 100% for the project tasks, and, (2) Collaboration (team meetings, f2f visits, travelings to ESO) among the team members. Obviously, a lot of effort is going to be "voluntary", actually done in frame of the research supported by involved institutions.

Calculations for the requested support form ESO are as follows: Personal costs for postdoc based at the Astronomical Institute in Ondřejov – table-based monthly salary + bonuses for researcher of the category V3B: 1900 EUR (gross); +36% deductions for the health-care and social insurance, times 24 month of the project duration gives approx. 62 kEUR. We take this amount as a base for other FTE-related costs calculations: There will be next 0.4 FTE allocated for the work at the project paid from the ESO project resources (plus further 1.4 FTE supported by the involved institutions) giving 0.4 x 62 = 24.8 kEUR. The total personal costs for the project execution are thus approx. 87 kEUR. The collaborating institutions (Zagreb, Cambridge, Glasgow) will use 2 kEUR each for travel to the project-team meetings, the bidding institution (EU ARC – Czech node) would need yet some extra money for (i) traveling more frequently to ESO, (ii) because of hosting the PI and dedicated postdoc, in total 6 kEUR. The total project costs requested from ESO are thus 87 + 3 x 2 + 6 = 99 kEUR.

The spread of the costs (in kEUR) among involved institutions (ESO share + involved-inst. contributions) is summarized in the following table:

Institution	Total FTE for the project	FTE requested from ESO	Total FTE costs	FTE costs req. from ESO	Travel costs (all req. from ESO)
Ondrejov	1.2	1.1	74.4	68.4	6
Zagreb	0.4	0.1	24.8	6.2	2
Cambridge	0.2	0.1	12.4	6.2	2
Glasgow	0.2	0.1	12.4	6.2	2
NRAO	0.2	0	12.4	0	0
NAOJ	0.2	0	12.4	0	0
NJIT	0.2	0	12.4	0	0
ALMA	0.2	0	12.4	0	0
Total	**2.8**	**1.4**	**173.6**	**87**	**12**

Other requested resources

In addition to the financial support for being able to carry out the study, our team will need support in the following immaterial resources:

- EoC test time at ALMA. We shall need a really short test of the procedure described in *Approach 1*: **One science scan = ~10 min on-source observation for each allowed Band** (3, 5, 6, and 7) of the Sun with as much complete 12m+7m array, in the (limiting) configuration specified later as a result of our simulations and their analyses. In addition to that, **a short test of modified *Go/noGo* procedure** – measuring the phase RMS at as much short as possible time scale. This can be done during technical break (A. Hales – discussion during the Oslo workshop) **with quite incomplete array**. For supplemental *Approach 2* (WV line in absorption): The existing Band 5 CSV data (2019.1.0005+13.CSV; array configuration: C43-2, LO freq. 198 GHz) are useful for rough estimation of the efficiency of phase error corrections based on the atmospheric WV line absorption. However, it would be fine (but not vital) to have another EoC observation with longer baselines and a bit lower LO in order to be closer to the WV line center in BB1. Alternatively, observation of a strong bandpass calibrator with an **incomplete array** (e.g., during the technical break) containing mid-size baseline at the same spectral setup (TDM continuum centered around LO frequency ~196 GHz) would be helpful for testing the procedure. In total: We request (as a minimum) **2 hours of total time with complete array** + some "cheap" time during the technical breaks, in amount given by opportunity.
- Expert consultations. It may happen that the team would need expert advice from ALMA or ESO staff beyond involvement of the experts working in the team (A. Hales, P. Yagoubov, L. Maud). Except of the tests described above and (seldom) request for external expertise sharing our project should not take any other manpower of ALMA/ESO staff.

References

Bárta, M. (2019): *Towards high-resolution solar ALMA images*, ALMA Development Workshop, held 3-5 June, 2019 at ESO Garching, http://adsabs.harvard.edu/abs/2019adw..confE...4B

Bastian, T. S. (2002): ALMA and the Sun, Astronomische Nachrichten 323, p. 271

Bastian, T. S., Bárta, M., Brajša, R., Chen, B., Pontieu, B. D., Gary, D. E., Fleishman, G. D., Hales, A. S., Iwai, K., Hudson, H., Kim, S., Kobelski, A., Loukitcheva, M., Shimojo, M., Skokić, I., Wedemeyer, S., White, S. M., Yan, Y. (2018): *Exploring the Sun with ALMA*, The Messenger 171, p. 25

Bastian, T. S. (2019): Solar ALMA Team communication.

Benz, A.O.– ALMA DRSP (2006): in ALMA Design Reference Science Plan https://home.strw.leidenuniv.nl/~alma/review/drsp20_review.shtml

Brajša, R., Skokić, I., Bárta, M., Bastian, T.S., Shimojo, M., White, S., Iwai, K. (2017): *Solar Research with ALMA – ESO Development Study*, Final Report, ESO

Brajša, R., Sudar, D., Benz, A. O., Skokić, I., Bárta, M., De Pontieu, B., Kim, S., Kobelski, A., Kuhar, M., Shimojo, M., Wedemeyer, S., White, S., Yagoubov, P., Yan, Y. (2018): First analysis of solar structures in 1.21 mm full-disc ALMA image of the Sun, Astronomy and Astrophysics 613, p. A17

ESO (2017): *ALMA starts observing the Sun*, ESO Press Release no. 1703, https://www.eso.org/public/news/eso1703/

Fomalont, E. B. et al. – ALMA Partnership (2015): *The 2014 ALMA Long Baseline Campaign: An Overview*, The Astrophysical Journal 808, p. L1

Heinzel, P., Berlicki, A., Bárta, M., Karlický, M., Rudawy, P. (2015): *On the Visibility of Prominence Fine Structures at Radio Millimeter Wavelengths*, Solar Physics 290, p. 1981

Hills, R. (2019): Solar ALMA Team communication.

Hudson, H. (2019): Solar ALMA Team communication.

Karlický, M., Bárta, M., Dąbrowski, B. P., Heinzel, P. (2011): *Solar Research with ALMA*, Solar Physics 268, p. 165

Loukitcheva, M. A., Solanki, S. K., White, S. (2008): ALMA as the ideal probe of the solar chromosphere, Astrophysics and Space Science 313, p. 197

Maud, L. T., Fomalont, E. B, Tilanus, R. P. J., Dent, B. (2017a): *ALMA & Allegro Phase Metrics Workshop Report*, ALMA Memo No. 606 https://library.nrao.edu/public/memos/alma/main/memo606.pdf

Maud, L. T., Tilanus, R. P. J., van Kempen, T. A., Hogerheijde, M. R., Schmalzl, M., Yoon, I., Contreras, Y., Toribio, M. C., Asaki, Y., Dent, W. R. F., Fomalont, E., Matsushita, S. (2017b): *Phase correction for ALMA. Investigating water vapour radiometer scaling: The long-baseline science verification data case study*, Astronomy and Astrophysics 605, p. A121

Otarola, A.; De Breuck, C.; Travouillon, T.; Matsushita, S.; Nyman, L.-Å.; Wootten, A.; Radford, S. J. E.; Sarazin, M.; Kerber, F.; Pérez-Beaupuits, J. P. (2019): *Precipitable Water Vapor, Temperature, and Wind Statistics At Sites Suitable for mm and Submm Wavelength Astronomy in Northern Chile*, PASJ 131, pp. 045001.

Shimojo, M. et al.. (2017): *Observing the Sun with the Atacama Large Millimeter/submillimeter Array (ALMA): High-Resolution Interferometric Imaging*, Solar Physics 292, p. 87

Shimojo, M. (2019): Solar ALMA Team communication.

Wedemeyer, S. et al.. (2016): *Solar Science with the Atacama Large Millimeter/Submillimeter Array—A New View of Our Sun*, Space Science Reviews 200, p. 1

Wedemeyer, S., Bastian, T.S., Shimojo, M., and Bárta, M. (eds.) (2020): *Proc. of 1st International Solar ALMA Imaging Workshop in Oslo*, March 2-6, 2020 (in preparation as ALMA Memo)

White, S. M. et al. (2017): *Observing the Sun with the Atacama Large Millimeter/submillimeter Array (ALMA): Fast-Scan Single-Dish Mapping*, Solar Physics 292, p. 88

Annex 3: WV line sensitivity – Calculations by Richard Hills

Sensitivity of Phase Correction based on Absorption Fluctuations

We assume that we are pointing at a fixed position on the Sun and making an aperture synthesis image. To calculate the phase correction, we assume that the observed fluctuations in the total power are due to variations in the atmospheric transmission caused by fluctuations in the amount of water along the line of sight from each antenna to the Sun. We want to find out how much error we will make in the correction if, instead, the variations are due to something else such as receiver gain fluctuations or variations in the solar emission. (Note that the latter will not be important if all the antennas have the same beam pattern and are pointing at the same region of the Sun.)

We write the total power received as:

$$Tsig = \eta \,.\, Tsun \,.\, Trans + Tatm \,.\, (1 - Trans) + Trx \qquad (1)$$

where Tsun is the Sun's brightness temperature, η is the appropriate beam efficiency, Tatm is the physical temperature of the water in the atmosphere, and Trx is the effective receiver temperature (which would include things like the losses in the antenna).

Trans is the transmission of the atmosphere, which is:

$$Trans = \exp[\, -(w.\tau_1 + \tau_{dry})\,] \qquad (2)$$

where w is the amount of water in the line of sight to the Sun in mm, τ_1 is the opacity for 1mm of water, and τ_{dry} is the opacity due to the "dry" components – mainly oxygen and ozone. These opacities are of course strong functions of frequency but relatively weak functions of temperature, pressure, etc. I have used some numbers I had from the atm model which should be good for ALMA conditions but I have not checked the details.

By substituting for Trans in (1) and differentiating one finds that the variation in total power due to a small change in w is

$$dTsig = -\tau_1 \,.\, (\eta.Tsun - Tatm) \,.\, Trans \,.\, dw$$

This means that, so long as the first term in equation (1) dominates, i.e. $(\eta.Tsun - Tatm) \,.\, Trans \gg Tatm + Trx$, the fractional change in the total power is

$$dTsig / Tsig = -\tau_1 \,.\, dw \,.$$

The change in the path to the antenna (in mm) due to the extra water is just given by $dp = r_{eff}\, dw$ where r_{eff} is the effective refractivity. To a good approximation this is equal to 6.3: there is a weak temperature dependence and a small frequency variation (dispersion) which we will neglect. The phase change is therefore

$$d\phi = 2\pi/\lambda \,.\, r_{eff} \,.\, dw = 2\pi/\lambda \,.\, r_{eff} \,.\, (-1/\tau_1) \,.\, dTsig / Tsig$$ where λ is the observing frequency.

So the conversion from fractional change in power to phase depends only on τ_1 and not on all the other quantities in equations (1) and (2).

The assumption that the first term in equation (1) is dominant will be true so long as we have a reasonably good receiver temperature and the opacity is not very large. At the center of the 183GHz line, however, the opacity can get quite large. I have therefore does the sums numerically in the spreadsheet assuming a small change (1 micron) in the amount of water. The other parameters are listed and can be changed but, apart from the amount of water, these have very little effect on the results. Separate sheets are given for the cases w = 1, 2 and 4.

I considered four possible ways of making the correction. In each case I assumed (guessed) a possible value for what the instrumental errors, e.g. gain fluctuations, might be and calculated what the corresponding

phase error would be. Remember that these measurements have to be made separately on each antenna and the correction is made to the phase of the signal from that antenna. The loss of coherence is therefore cos(sqrt(2).dϕ), i.e. if dϕ has an rms variation of 0.3 radians we lose about 10% coherence. This is probably a reasonable goal.

Exactly how one does the full observing sequence which includes moving to a phase calibrator from time to time (and perhaps making measurements with the conventional water-vapour radiometers at those points) is not clear to me, but presumably we need to follow the variation in power over the time between observations of the phase calibrator. The assumed values for the gain variations, etc., should therefore correspond to that period.

The cases are:

1) Total power. One simply chooses some suitable frequency in the observing band and measures the total power fluctuations. I took 10^{-3} for the stability in this case. The plot shows that if the stability is as poor as this, one would have to put one of the observing bands quite close to the water line, where the opacity is probably too large to use it for astronomy.

2) Fluctuations in the slope across a spectral window. This could be measured using the auto-correlation spectrum and the stability should be pretty good. I assumed that the slope is measured in units of change per GHz and took a value of 10^{-4} for the stability. One sees that, for the larger values of w, there are places in the plot of the fluctuation, dTsig/Tsig, against frequency has turning points, so measurements that rely on the slope will not work there. So long as w < 4mm, however, LO frequencies in the range ~195 to 200 GHz look OK.

3) Difference between two neighbouring spectral windows. Here one would use the total power counters associated with the pair of windows in a sideband. One sees that with a stability of 2.10^{-4} (in the difference) this looks OK with LO between about 194 and 199GHz.

4) Finally, one can take the fluctuations in the difference between the total power in the upper and lower sidebands. This gives a bigger change in signal but the errors are likely to be larger because less of the hardware is in common. For 5.10^{-4} this looks OK from about 193 to 198GHz.

   I must emphasize that these stability estimates are complete guesses. Obviously we need experiments to find out what the reality is and which is the most promising method. In all cases, however, these values are much larger than the errors due just to radiometric noise: remember that for 1 second integration the radiometric fluctuations are down to 10^{-4} for a bandwidth of just 100MHz. The errors that I expect to dominate will be due to things like temperature variations, LO level changes, etc.

   One could perhaps use a combination of 2) to 4) above to get a better results and estimate the errors. We also have two polarization channels so we get two values for each measurement. These will have independent noise but partially correlated other errors, so this will probably not help much.

   Conclusion

   The tension here is between having the LO frequency low enough to be able to measure the changes in absorption and being far enough from the line to avoid having excessive absorption and slope across the lowest spectral window. With the LO at 198GHz, the transmission in spw0 is about 89% for w = 1mm and the slope is less than 2% per GHz. I think this would be no problem. For w = 4mm the transmission is down to ~63% and the slope is about 5% per GHz, which is getting pretty high but my guess is that the standard calibration procedures would still cope reasonably well. It may be that the stability is hopeless in the daytime when the water is that high anyway.

Richard Hills 4th Aug 2019

Annex 4: Code listings

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
      MAKE UNCORRUPTED VISIBILITIES FPOM THE IMPORTED OPTICAL SOLAR IMAGE
================================================================================

Created on Thu Apr 13 12:42:12 2023
@author: yichai
"""
# import os
# import glob
# import numpy as np
# from simutil import simutil
# u=simutil()

#casa6
import os
import glob
import numpy as np
from casatools import simulator, image, table, coordsys, measures, componentlist, quanta, ctsys
from casatasks.private import simutil
sm = simulator()
u = simutil.simutil()

#workding directory
#os.chdir('/Users/yichai/Documents/ALMA_simulation/CRISP_8542_20210819_b3c4_image/')
os.chdir('/Users/yichai/Documents/ALMA_simulation/alma_cycle9_configuration_files/sim_b5c6_Hinode
_p30/')

#parameters
#keywords = 'CRISP_8542_20210819_b3c4'
keywords = 'Hinode_SOT_20150623T102846_b5c6'
#reftime = '2021/08/19/17:28:00' #CRISP
#reftime = '2015/06/23/10:28:00' #Hinode
reftime = '2015/06/23/14:00:00' #Hinode modified time
#tstep = 10 #second
overwrite = True
defaultaxes = True
#phasecenter = 'J2000 09h55m30.91 12d37m47.9'
#phasecenter = 'J2000 06h06m20.46 23d25m48.8'
phasecenter = 'J2000 06h06m56.98 23d25m45.5' #Hinode modified coordinate
#array configuration file location
ALMA_main = '../alma.cycle9.6.cfg'
ALMA_ACA = '../aca.cycle9.cfg'

#import fits file
#band3 100GHz; band5 198GHz; band6 239GHz;
#fitsimage = '../Hinode_SOT_20150623T102846_single_mod_9.fits'
fitsimage = '../../Hinode_SOT_20150623T102846_single_mod.fits'
#imagename = 'Hinode_SOT_20150623T102846_single.image'
imagename = 'Hinode_SOT_20150623T102846_single_mod_b5c6.image'
bandname = 'band5' #ALMA band name
ref_freq = '198GHz' #center frequency
delta_freq = "1GHz"
freq_resolution = "1GHz"
int_time = '2s' #integration time
etime = qa.quantity(60,'s') #duration of observation
out_put_ms = 'alma_12m7m_sim_b5c6_test_Hinode_mod.ms'
#defaultaxesvalues = ['09h55m30.91', '+12d37m47.9', '100GHz', 'I']
#defaultaxesvalues = ['06h06m20.46', '+23d25m48.8', '100GHz', 'I']
defaultaxesvalues = ['06h06m56.98', '+23d25m45.5', '198GHz', 'I'] #band3 100Ghz band5 198GHz
band6 239GHz band7 346.6GHz
beam = ['0.16arcsec', '0.16arcsec', '0deg'] #modify beam according to configuration and
wavelength
os.system('rm -rf '+ imagename)
importfits(fitsimage = fitsimage,
           imagename = imagename,
```

```
            overwrite = True,
            defaultaxes = True,
            whichhdu = 1,
            defaultaxesvalues = defaultaxesvalues,
            beam = beam)


#changing image parameters so it suit the simulated ALMA configuration
#for single fits file
imagename = imagename

t = qa.time(qa.quantity(qa.convert(reftime,'s')['value'],'s'),form='ymd')
imhead(imagename = imagename,mode='put',hdkey='object',hdvalue='sun')
imhead(imagename = imagename,mode='put',hdkey='bunit',hdvalue='Jy/pixel')
imhead(imagename = imagename,mode='put',hdkey='date-obs',hdvalue=t[0])

# #center coordinate from CRISP image
# xc = imhead(imagename = 'CRISP_8542_20210819_band3_config4.image',mode='get',hdkey='crval1')
# yc = imhead(imagename = 'CRISP_8542_20210819_band3_config4.image',mode='get',hdkey='crval2')
# #change Hinode image coordinate to CRISP image
# imhead(imagename = imagename, mode='put',hdkey='crval1',hdvalue=xc)
# imhead(imagename = imagename, mode='put',hdkey='crval2',hdvalue=yc)

bmaj = imhead(imagename=imagename,mode='get',hdkey='beammajor')
cell = str(qa.div(bmaj,6)['value'])+'arcsec'
imhead(imagename=imagename,mode='put',hdkey='cdelt1',hdvalue=cell)
imhead(imagename=imagename,mode='put',hdkey='cdelt2',hdvalue=cell)

#read antenna config files, the antenna cfg file has one addition column of padnames
x,y,z,d,padnames,trash1,telescope,posobs = u.readantenna(ALMA_main)
x2,y2,z2,d2,padnames2,trash2,telescope2,posobs2 = u.readantenna(ALMA_ACA)

out_put_ms = out_put_ms
os.system('rm -rf '+out_put_ms)
print('now processing: ' + str(out_put_ms))

sm.open(out_put_ms)
sm.setconfig(telescopename="ALMA",x=np.append(x,x2),y=np.append(y,y2),z=np.append(z,z2),dishdiame
ter=np.append(d,d2),mount=['alt-az'],
padname=np.append(padnames,padnames2).tolist(),coordsystem='global', referencelocation=posobs)
sm.setspwindow(spwname=bandname,
freq=ref_freq,deltafreq=delta_freq,freqresolution=freq_resolution,nchannels=1,stokes='XX YY')
sm.setfeed(mode='perfect X Y',pol=[''])
sm.setlimits(shadowlimit=0.01, elevationlimit='10deg')
sm.setauto(0.0)

#sm.setfield(sourcename='sun',sourcedirection='J2000 09h55m30.91
+12d37m47.9',calcode='OBJ',distance='0m')
#sm.setfield(sourcename='sun',sourcedirection='J2000 06h06m20.46
+23d25m48.8',calcode='OBJ',distance='0m')
sm.setfield(sourcename='sun',sourcedirection=phasecenter,calcode='OBJ',distance='0m')
sm.settimes(integrationtime=int_time,usehourangle=False,referencetime=me.epoch('UTC', t[0]))
etime = etime
sm.observe(sourcename="sun", spwname=bandname,starttime=qa.quantity(0,"s"),stoptime=etime)
sm.setoptions(ftmachine="mosaic")

sm.predict(imagename=imagename)
sm.close()
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
    MANGLE THE GAIN CALIBRATION TABLE BY SIMULATED ATMOS. PHASE DELAYS
================================================================================

Created on Mon Mar 27 12:52:13 2023
Based on the ATMOS code provided by Eric
@author: yichai
"""
import os
import glob
import math
from math import pi,cos,sin,tan,atan
import sys
import random
#import commands
import numpy as np
import datetime
import scipy as sp
from scipy import optimize
import cmath
import string
# from simutil import simutil
# u = simutil()

#casa6
from casatools import simulator, image, table, coordsys, measures, componentlist, quanta, ctsys
from casatasks.private import simutil
u = simutil.simutil()
sm = simulator()

#workding directory
#os.chdir('/Users/yichai/Documents/ALMA_simulation/alma_cycle9_configuration_files/sim_b3c4_10min/')
#os.chdir('/Users/yichai/Documents/ALMA_simulation/alma_cycle9_configuration_files/sim_b3c4_combined/')
os.chdir('/Users/yichai/Documents/ALMA_simulation/alma_cycle9_configuration_files/sim_b5c6_Hinode_p30/')


scaleF = 0.5                    # multiplication factor for tropospheric phases
#key_words = 'alma_12m7m_sim_b3c4_scalef0.5_phase30_test_combined'
key_words = 'alma_12m7m_sim_b5c6_scalef0.5_phase30_Hinode_test'
orig_data = 'alma_12m7m_sim_b5c6_test_Hinode_mod.ms' # INPUT ms
out_data = 'alma_12m7m_sim_b5c6_scalef0.5_phase30_Hinode_test.ms'
cal_table = 'atmos_scalef0.5_phase30_Hinode_test.cal'

#orig_data = 'alma_12m7m_sim_b3c4_test_combined.ms'
#out_data = 'alma_12m7m_sim_b3c4_scalef0.5_phase30_test_combined.ms'

# key_words2 = 'alma_12m7m_sim_b3c7_pwv1_Hinode_smooth'
# out_data2 = 'alma_12m7m_sim_b3c7_pwv1_Hinode_smooth.ms'
# cal_table2 = 'atmos_pwv1_Hinode.cal'

#frefant = '13'                 # reference antenna
frefant = '8'
fconf = '../alma.cycle9.6.cfg'      # Configuration file
fconf2 = '../aca.cycle9.cfg'

#use half maximum baseline as threshold?
maxbl = np.max(u.baselineLengths(fconf))
#calculate other factor
degree = 30 #phase error
k1 = degree/((maxbl/2.)**0.8)
k2 = degree/((maxbl/2.)**0.25)
```

```
#   prms = PRMS(antbl,scaled):
#      antbl is antenna separation from reference antenna
#      prms is the rms phase for that antenna separation
def PRMS(antbl,scaleF):
    #  The structure function here gives 30 deg phase rms at 10000m = 10km
    #
    #Lrms = 1.0/52.83 * antbl**0.8     # phase rms ~0.8 power to 10 km, 1/52.83*10000^0.8 ~ 30
    Lrms = k1 * antbl**0.8
    #Hrms = 3.0 * antbl**0.25        # phase rms `0.25 power beyond 10 km
    Hrms = k2 * antbl**0.25
    # if antbl < 10000.0:
    #    prms = scaleF*Lrms
    # if antbl > 10000.0:
    #    prms = scaleF*Hrms
    if antbl < maxbl/2.0:
        prms = scaleF*Lrms
    if antbl >= maxbl/2.0:
        prms = scaleF*Hrms
    return prms

os.system('rm -rf '+out_data)
os.system('cp -r '+ orig_data+' '+out_data)


zant = []
zx = []
zy = []
zz = []
zpad = []

#read main array antenna info
f = open(fconf)
lines = f.readlines()
nlines = len(lines)
f.close()
for i in range(3,nlines):
    stuff = lines[i].split()
    zx.append(float(stuff[0]))
    zy.append(float(stuff[1]))
    zz.append(float(stuff[2]))
    zpad.append(stuff[4][0:4])

#read ACA info
f2 = open(fconf2)
lines2 = f2.readlines()
nlines2 = len(lines2)
f2.close()
for j in range(3,nlines2):
    stuff2 = lines2[j].split()
    zx.append(float(stuff2[0]))
    zy.append(float(stuff2[1]))
    zz.append(float(stuff2[2]))
    zpad.append(stuff2[4][0:4])


nant = len(zx)
zxref = []
zyref = []
zzref = []
zztot = []
nref = int(frefant)
for i in range(0,nant):
    zxref.append(zx[i]-zx[nref])
    zyref.append(zy[i]-zy[nref])
    zzref.append(zz[i]-zz[nref])
for i in range(0,nant):
    # The distance antenna is from reference
    # Used for getting antenna structure function
```

```
    zztot.append(np.sqrt(zxref[i]**2+zyref[i]**2+zzref[i]**2))


# make gaintable 'atmos.cal' to be filled with atmospheric phase
tb.close()
os.system('rm -rf '+ cal_table)
gaincal(vis = out_data,
        caltable = cal_table,
        refant = frefant,
        minsnr = 0.001,
        #solint = '40s',    # should be time-scale of troposphere change?
        solint = 'int',
        calmode = 'p')

# Obtain certain 'atmos.cal' columns
#   ans fill in antenna-based antmospheric phases with
#
tb.open(cal_table,nomodify=False)
yant = tb.getcol('ANTENNA1')
ytime = tb.getcol('TIME')
ycparam = tb.getcol('CPARAM')
Nycparam = ycparam
nant = len(yant)

print('     ant    time   antsep    rmsphase')

antbl_arr = []
prms_arr = []
perror_arr = []
for i in range(0,nant):
    antbl = zztot[yant[i]]
    # get rms phase for each antenna
    prms = PRMS(antbl,scaleF)
    # determine random GAUSSIAN phase error from rms phase
    perror = random.gauss(0,prms)
    # PRINT OUT ANT PHASE ERRORS
    #print('%3d %3d %7.1f %7.1f %7.1f %7.1f ')%(i, yant[i],ytime[i]-ytime[0],antbl,prms,perror)
    print('%3d %3d %7.1f %7.1f %7.1f %7.1f ' % (i, yant[i],ytime[i]-ytime[0],antbl,prms,perror))
    # put random phase in gaintable column CPARAM
    rperror = np.cos(perror*pi/180.0)
    iperror = np.sin(perror*pi/180.0)
    Nycparam[0][0][i] = 1.0*np.complex(rperror,iperror)   #X POL
    Nycparam[1][0][i] = 1.0*np.complex(rperror,iperror)   #Y POL  ASSUMED SAME
    #save parameters
    antbl_arr.append(antbl)
    prms_arr.append(prms)
    perror_arr.append(perror)

np.save('antbl.npy',np.array(antbl_arr))
np.save('prms.npy',np.array(prms_arr))
np.save('perror.npy',np.array(perror_arr))

tb.putcol('CPARAM',Nycparam)
tb.flush()
tb.close()

#apply the atmospheric phases to uvdata
applycal(vis = out_data, gaintable = cal_table)
print('done')

fig_file = key_words+'_cal.png'
os.system('rm -rf '+ fig_file)
# plotcal(caltable = cal_table,
#         xaxis = 'time',
#         yaxis = 'phase',
#         showgui = False,
#         figfile = fig_file)
```

```
# plotms(vis = cal_table,
#        xaxis = 'time',
#        yaxis = 'phase',
#       showgui = False,
#       overwrite = True,
#        plotfile = fig_file)
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
          IMAGING & SELF-CAL LOOP FOR A SINGLE IMAGE
================================================================================
Created on Thu Mar 30 10:42:53 2023
@author: yichai
"""
#general parameters
#msco = 'alma_12m7m_sim_b3c4_scalef0.5_phase30_test.ms'
msco = 'alma_12m7m_sim_b5c6_scalef0.5_phase30_Hinode_test.ms'
keywords = 'alma_12m7m_sim_b5c6_scalef0.5_phase30_Hinode_test'

#msoc = 'alma_12m7m_sim_b7c4_pwv1_Hinode.ms'
#keywords = 'alma_12m7m_sim_b7c4_pwv1_Hinode'

#clean parameters
cell_size =  '0.026arcsec' #b5c6
imsize = [1200,1200]
robust = 1.0
#threshold = '2.0Jy'
#niter = 40000
niter = 200000 #more niter for large images
#niter2 = 40000
pbmask = 0.2
gain = 0.025
#spw = '0,1,2,3'
#stokes = 'I'
field = '0'
#phasecenter = 0
#phasecenter = 'J2000 09h55m30.91 12d37m47.9'
phasecenter = 'J2000 06h06m56.98 23d25m45.5' #modified for Hinode data
#deconvolver = 'clark'
deconvolver = 'multiscale'
scales = [0,6,18]

#snr setup
minsnr=3.0

#refant = 'DA44'
refant = '8'
#reftime = '2021/08/19/17:28:00'
reftime = '2015/06/23/14:00:00' #modified time for Hinode data

#selcal_div = [1,4,15,60,240]
t_interval = ['inf','60','int','int','int']
#t_interval = ['int','int','int','int','int']
#t_interval = ['inf'] #for comparison between un-selfcaled data
#---------
# selcal_div: the duration for selcal:
#   1 : 1/1 scan (~8mins@Band6)
#   4 : 1/4 scan (~2mins@Band6)
#  15 : 1/15 scan (~32secs@Band6)
#  60 : 1/60 scan (~8secs@Band6)
# 240 : 1/240 scan (~2secs@BAnd6)
#---------

import analysisUtils as aU
es = aU.stuffForScienceDataReduction()

import os

#obtain scan numbers
myms = aU.createCasaTool(mstool)
myms.open(msco)
mymsmd = aU.createCasaTool(msmdtool)
mymsmd.open(msco)
```

```
tgt_scans = mymsmd.scansforintent('*OBSERVE_TARGET*')
mymsmd.close()
myms.close()

os.system('rm -rf '+keywords+'_?')
os.system('rm -rf '+keywords+'_*_selcal?.ms')
os.system('rm -rf '+keywords+'_*_selcal?.ms*')
os.system('rm -rf '+keywords+'_*_selcal?.tbl*')

sc = tgt_scans[0]
msc = keywords+'_'+str(sc)+'_selcal0.ms'

#------------- Preparing the initial MS

sc = tgt_scans[0]
msc = keywords+'_'+str(sc)+'_selcal0.ms'

# split(vis = msco,
#      outputvis = msc,
#      datacolumn = 'data',
#      scan = sc,
#      keepflags = True)

split(vis = msco,
      outputvis = msc,
      datacolumn = 'corrected',
      scan = sc,
      keepflags = True)

#fixplanets(msc, field ='0', fixuvw = True, refant=refant, reftime = reftime)
#tb.open(msc+'/POINTING', nomodify = False)
#a = tb.rownumbers()
#tb.removerows(a)
#tb.close()

tb.open(msco)
raw_data = tb.query('SCAN_NUMBER == %d && ANTENNA1 == 0 && ANTENNA2 == 2 && DATA_DESC_ID == 0' %
sc)
tt_data = raw_data.getcol('TIME')
tb.close
tt_st = qa.time(qa.quantity(tt_data[0]-1, 's'), form="ymd",prec=8)[0]
tt_ed = qa.time(qa.quantity(tt_data[-1]+1, 's'), form="ymd",prec=8)[0]
tt_timerange = '%s~%s' % (tt_st, tt_ed)

for i in range(0,len(t_interval)):
    msc = keywords+'_'+str(sc)+'_selcal'+str(i)+'.ms'
    imgname_base = keywords+'_'+str(i)+'/'+keywords+'_'+str(sc)+'_selcal'+str(i)
    if os.path.exists(keywords+'_'+str(i)) == False:
        os.system('mkdir '+keywords+'_'+str(i))

    imagename = imgname_base
    print('Now synthsizing : '+imagename)

    tclean(vis = msc,
           #spw = spw,
           #stokes = stokes,
           field = field,
           timerange = tt_timerange,
           phasecenter = phasecenter,
           imagename = imagename,
           cell = cell_size,
           imsize = imsize,
           interactive = False,
           usemask = 'pb',
           pbmask = pbmask,
           pblimit = pbmask,
           specmode = 'mfs',
           gridder = 'mosaicft',
```

```
            weighting = 'briggs',
            robust = robust,
            niter = niter,
            deconvolver = deconvolver,
            scales = scales,
            #threshold = threshold,
            cycleniter = 1000,
            parallel = True,
            gain = gain)

    tclean(vis = msc,
            #spw = spw,
            #stokes = stokes,
            field = field,
            timerange = tt_timerange,
            phasecenter = phasecenter,
            imagename = imagename,
            cell = cell_size,
            imsize = imsize,
            interactive = False,
            usemask = 'pb',
            pbmask = pbmask,
            pblimit = pbmask,
            specmode = 'mfs',
            gridder = 'mosaicft',
            weighting = 'briggs',
            robust = robust,
            niter = 0,
            deconvolver = deconvolver,
            scales = scales,
            #threshold = threshold,
            calcres = False,
            calcpsf = False,
            savemodel = 'modelcolumn',
            gain = gain)

    impbcor(imagename=imagename + '.image',
             pbimage=imagename+'.pb',
             outfile=imagename+'.pbcor', mode='divide')

    exportfits(imagename = imagename + '.image',fitsimage = imagename + '.fits')

    #-------- create and apply calibration table
    t_int = t_interval[i]
    gaincal(vis = msc, caltable = keywords+'_'+str(sc)+'_selcal'+str(i)+'.tbl',gaintype =
'G',calmode = 'p',refant = refant,
             solint = t_int, minsnr = minsnr)
    #es.checkCalTable(keywords+'_'+str(sc)+'_selcal'+str(i)+'.tbl', msName = msc, interactive =
False)
    applycal(vis = msc, gaintable = keywords+'_'+str(sc)+'_selcal'+str(i)+'.tbl',interp =
'nearest', applymode = 'calonly', calwt = False)

    #-------- create MS for next step of self calibration
    nextmsc = keywords+'_'+str(sc)+'_selcal'+str(i+1)+'.ms'
    split(vis = msc, outputvis = nextmsc, datacolumn = 'corrected', keepflags = True)

    #remove older measurement set
    os.system('rm -rf '+msc)


#generate snap-shot images
domovie = 0
if domovie == 1:

    #------------ Loop of synthesizing final images
    if os.path.exists(keywords+'_F') == False:
        os.system('mkdir '+keywords+'_F')
```

```
    msc = nextmsc
    imgname_base = keywords+'_F/'+keywords+'_'+str(sc)+'_selcalF'

    img_step = 1
    t_step = 2 #normal 10min scan: 2s, short-int scan:0.2s
    for synth_num in range(0, tt_data.size):
        st_tim = qa.time(qa.quantity(tt_data[synth_num*img_step]-t_step/2., 's'), form="ymd",
prec=8)[0]
        ed_tim = qa.time(qa.quantity(tt_data[(synth_num+1)*img_step-1]+t_step/2., 's'),
form="ymd", prec=8)[0]
        timerange = '%s~%s' % (st_tim, ed_tim)
        imagename = imgname_base + '_%03d' % synth_num
        print('Now synthsizing ('+str(synth_num+1)+'): '+imagename)


        tclean(vis = msc,
                #spw = spw,
                #stokes = stokes,
                field = field,
                timerange = timerange,
                phasecenter = phasecenter,
                imagename = imagename,
                cell = cell_size,
                imsize = imsize,
                interactive = False,
                usemask = 'pb',
                pbmask = pbmask,
                pblimit = pbmask,
                specmode = 'mfs',
                gridder = 'mosaicft',
                weighting = 'briggs',
                robust = robust,
                niter = niter,
                deconvolver = deconvolver,
                scales = scales,
                #threshold = threshold,
                cycleniter = 1000,
                parallel = True,
                gain = gain)

        impbcor(imagename=imagename + '.image',
                pbimage=imagename+'.pb',
                outfile=imagename+'.pbcor', mode='divide')

        exportfits(imagename= imagename + '.image',
                   fitsimage= imagename + '.fits')
```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
          IMAGING & SELF-CAL LOOP FOR A MOVIE
================================================================================

Created on Tue Oct 17 10:51:22 2023
@author: yichai
"""
#general parameters
msco = 'alma_12m7m_sim_b5c6_scalef0.5_phase30_Hinode_test.ms'
keywords = 'alma_12m7m_sim_b5c6_scalef0.5_phase30_Hinode_test'

#clean parameters
cell_size =  '0.026arcsec' #b5c6
imsize = [1200,1200]
robust = 1.0
#threshold = '2.0Jy'
#niter = 40000
niter = 200000 #more niter for large images?
#niter2 = 40000
pbmask = 0.2
gain = 0.025
#spw = '0,1,2,3'
#stokes = 'I'
field = '0'
#phasecenter = 0
#phasecenter = 'J2000 09h55m30.91 12d37m47.9'
phasecenter = 'J2000 06h06m56.98 23d25m45.5' #modified for Hinode data
#deconvolver = 'clark'
deconvolver = 'multiscale'
scales = [0,6,18]

#snr setup
minsnr=3.0

#refant = 'DA44'
refant = '8'
#reftime = '2021/08/19/17:28:00'
reftime = '2015/06/23/14:00:00' #modified time for Hinode data

#selcal_div = [1,4,15,60,240]
t_interval = ['inf','60','int','int','int']
#t_interval = ['int','int','int','int','int']
#t_interval = ['inf'] #for comparison between un-selfcaled data
#---------
# selcal_div: the duration for selcal:
#   1 : 1/1 scan (~8mins@Band6)
#   4 : 1/4 scan (~2mins@Band6)
#  15 : 1/15 scan (~32secs@Band6)
#  60 : 1/60 scan (~8secs@Band6)
# 240 : 1/240 scan (~2secs@BAnd6)
#--------

import analysisUtils as aU
es = aU.stuffForScienceDataReduction()
import os

#obtain scan numbers
myms = aU.createCasaTool(mstool)
myms.open(msco)
mymsmd = aU.createCasaTool(msmdtool)
mymsmd.open(msco)
tgt_scans = mymsmd.scansforintent('*OBSERVE_TARGET*')
mymsmd.close()
myms.close()
```

```
#------------ Loop of synthesizing final images
if os.path.exists(keywords+'_F') == False:
    os.system('mkdir '+keywords+'_F')

sc = tgt_scans[0]
i = len(t_interval)-1
nextmsc = keywords+'_'+str(sc)+'_selcal'+str(i+1)+'.ms'
print(nextmsc)
msc = nextmsc
imgname_base = keywords+'_F/'+keywords+'_'+str(sc)+'_selcalF'

tb.open(msco)
raw_data = tb.query('SCAN_NUMBER == %d && ANTENNA1 == 0 && ANTENNA2 == 2 && DATA_DESC_ID == 0' %
sc)
tt_data = raw_data.getcol('TIME')
tb.close

#generate snap-shot images
domovie = 1
if domovie == 1:
    img_step = 1
    t_step = 2 #normal 10min scan: 2s, short-int scan:0.2s
    for synth_num in range(0, tt_data.size, img_step):
        st_tim = qa.time(qa.quantity(tt_data[synth_num]-t_step/2., 's'), form="ymd", prec=8)[0]
        ed_tim = qa.time(qa.quantity(tt_data[synth_num]+t_step/2., 's'), form="ymd", prec=8)[0]
        timerange = '%s~%s' % (st_tim, ed_tim)
        imagename = imgname_base + '_%03d' % synth_num
        print('Now synthsizing ('+str(synth_num+1)+'): '+imagename)


        tclean(vis = msc,
                #spw = spw,
                #stokes = stokes,
                field = field,
                timerange = timerange,
                phasecenter = phasecenter,
                imagename = imagename,
                cell = cell_size,
                imsize = imsize,
                interactive = False,
                usemask = 'pb',
                pbmask = pbmask,
                pblimit = pbmask,
                specmode = 'mfs',
                gridder = 'mosaicft',
                weighting = 'briggs',
                robust = robust,
                niter = niter,
                deconvolver = deconvolver,
                scales = scales,
                #threshold = threshold,
                cycleniter = 1000,
                parallel = True,
                gain = gain)

        impbcor(imagename=imagename + '.image',
                pbimage=imagename+'.pb',
                outfile=imagename+'.pbcor', mode='divide')

        exportfits(imagename= imagename + '.image',
                   fitsimage= imagename + '.fits')
```

Annex 5: Observation reports

2022.1.00031.CSV **Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations (CSV-3832)**

Miroslav Barta
<barta@asu.cas.cz>

Project state changes

Role	UTC	State
hmessias	2023-11-27T19:24:22	CSVReady

Timeline

Wednesday, 29 Nov 2023

	UTC	Author	Details
SBEX	14:01		***Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations (CSV-3832) (Version 1.2)*** *--SUCCESS--*

Start/End 13:34:39 - 14:01:57
Project Code 2022.1.00031.CSV — **PI** mbarta
SchedBlock Sun_a_03_INT — **ExecBlock** uid://A002/X1100c1a/X145af
SB UID uid://A001/X362b/Xee0 — **QA0 Status**
Band ALMA_RB_03
ALMA Build ONLINE-CYCLE10-B-28-2023-11-04-05-00-00
Array Name Array90-BLC — **Array/Corr** 12 [m]/64-antenna

Saturday, 16 Dec 2023

	UTC	Author	Details
SBEX	12:59		***Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations (CSV-3832) (Version 2.5)*** *--FAIL--*

Start/End 12:58:05 - 12:59:12
Project Code 2022.1.00031.CSV — **PI** mbarta
SchedBlock Sun_a_05_INT — **ExecBlock** uid://A002/X110fd13/X108e3
SB UID uid://A001/X362b/Xee1 — **QA0 Status**
Band ALMA_RB_05
ALMA Build ONLINE-CYCLE10-B-28-2023-11-04-05-00-00
Array Name Array82-BLC — **Array/Corr** 12 [m]/64-antenna
JIRA PRTSPR-70876

	UTC	Author	Details
SBEX	13:04		***Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations (CSV-3832) (Version 2.5)*** *--FAIL--*

Start/End 13:03:20 - 13:04:05
Project Code 2022.1.00031.CSV — **PI** mbarta
SchedBlock Sun_a_05_INT — **ExecBlock** uid://A002/X110fd13/X1099c
SB UID uid://A001/X362b/Xee1 — **QA0 Status**
Band ALMA_RB_05
ALMA Build ONLINE-CYCLE10-B-28-2023-11-04-05-00-00
Array Name Array82-BLC — **Array/Corr** 12 [m]/64-antenna
JIRA PRTSPR-70876

	UTC	Author	Details
SBEX	13:37		***Towards High-resolution Solar ALMA Images Overcoming current ObsMode limitations (CSV-3832) (Version 2.5)*** *--SUCCESS--*

Start/End 13:12:26 - 13:37:25
Project Code 2022.1.00031.CSV — **PI** mbarta
SchedBlock Sun_a_05_INT — **ExecBlock** uid://A002/X110fd13/X1099e
SB UID uid://A001/X362b/Xee1 — **QA0 Status**
Band ALMA_RB_05
ALMA Build ONLINE-CYCLE10-B-28-2023-11-04-05-00-00
Array Name Array84-BLC — **Array/Corr** 12 [m]/64-antenna

```
================================================================================
           MeasurementSet Name:
/scratch/barta/LB_Solar/Project_2022.1.00031.CSV/Sun_10_a_05_INT/X1099e/uid___A002_X110fd13_X1099
e.ms      MS Version 2
================================================================================
   Observer: mbarta     Project: uid://A001/X362b/Xede
Observation: ALMA
Data records: 71494020       Total elapsed time = 1431.65 seconds
   Observed from   16-Dec-2023/13:13:21.8   to   16-Dec-2023/13:37:13.4 (UTC)

   ObservationID = 0         ArrayID = 0
  Date        Timerange (UTC)          Scan  FldId FieldName             nRows     SpwIds
Average Interval(s)    ScanIntent
  16-Dec-2023/13:13:21.8 - 13:13:45.3     1      0 Sun                     1082224
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24]  [0.016, 0.016, 0.016, 0.016,
1.15, 0.192, 0.192, 0.192, 0.192, 0.192, 0.192, 0.192, 0.192, 0.016, 0.016, 0.016, 0.016, 0.192,
0.192, 0.192, 0.192, 0.192, 0.192, 0.192, 0.192]
[CALIBRATE_ATMOSPHERE#AMBIENT,CALIBRATE_ATMOSPHERE#HOT,CALIBRATE_ATMOSPHERE#OFF_SOURCE,CALIBRATE_
ATMOSPHERE#REFERENCE,CALIBRATE_WVR#AMBIENT,CALIBRATE_WVR#HOT,CALIBRATE_WVR#OFF_SOURCE,CALIBRATE_W
VR#REFERENCE]
              13:14:51.6 - 13:15:49.3     2      1 J1256-0547              856700
[4,25,26,27,28,29,30,31,32,33,34,35,36]  [1.15, 0.016, 0.016, 0.016, 0.016, 2.02, 1.01, 2.02,
1.01, 2.02, 1.01, 2.02, 1.01] [CALIBRATE_POINTING#ON_SOURCE,CALIBRATE_WVR#ON_SOURCE]
              13:16:21.5 - 13:17:24.7     3      1 J1256-0547             4247984
[0,1,2,3,4,5,6,7,8,9,10,11,12]  [0.016, 0.016, 0.016, 0.016, 1.15, 0.192, 0.192, 0.192, 0.192,
0.192, 0.192, 0.192, 0.192]
[CALIBRATE_SIDEBAND_RATIO#IMAGE,CALIBRATE_SIDEBAND_RATIO#SIGNAL,CALIBRATE_WVR#IMAGE,CALIBRATE_WVR
#SIGNAL]
              13:17:35.2 - 13:17:52.8     4      1 J1256-0547              811772
[0,1,2,3,4,5,6,7,8,9,10,11,12]  [0.016, 0.016, 0.016, 0.016, 1.15, 0.192, 0.192, 0.192, 0.192,
0.192, 0.192, 0.192, 0.192]
[CALIBRATE_ATMOSPHERE#AMBIENT,CALIBRATE_ATMOSPHERE#HOT,CALIBRATE_ATMOSPHERE#OFF_SOURCE,CALIBRATE_
WVR#AMBIENT,CALIBRATE_WVR#HOT,CALIBRATE_WVR#OFF_SOURCE]
              13:17:58.3 - 13:23:00.9     5      1 J1256-0547            21307624
[0,1,2,3,4,5,6,7,8,9,10,11,12]  [0.016, 0.016, 0.016, 0.016, 1.15, 0.192, 0.192, 0.192, 0.192,
0.192, 0.192, 0.192, 0.192] [CALIBRATE_BANDPASS#ON_SOURCE,CALIBRATE_WVR#ON_SOURCE]
              13:24:08.1 - 13:26:06.0     6      2 J1624-3213              2141776
[4,25,26,27,28,29,30,31,32,33,34,35,36]  [1.15, 0.016, 0.016, 0.016, 0.016, 2.02, 1.01, 2.02,
1.01, 2.02, 1.01, 2.02, 1.01] [CALIBRATE_POINTING#ON_SOURCE,CALIBRATE_WVR#ON_SOURCE]
              13:26:44.7 - 13:27:45.8     7      3 J1700-2610              4261556
[0,1,2,3,4,5,6,7,8,9,10,11,12]  [0.016, 0.016, 0.016, 0.016, 1.15, 0.192, 0.192, 0.192, 0.192,
0.192, 0.192, 0.192, 0.192] [CALIBRATE_PHASE#ON_SOURCE,CALIBRATE_WVR#ON_SOURCE]
              13:27:57.3 - 13:28:14.3     8      0 Sun                      811720
[0,1,2,3,4,5,6,7,8,9,10,11,12]  [0.016, 0.016, 0.016, 0.016, 1.15, 0.192, 0.192, 0.192, 0.192,
0.192, 0.192, 0.192, 0.192]
[CALIBRATE_ATMOSPHERE#AMBIENT,CALIBRATE_ATMOSPHERE#HOT,CALIBRATE_ATMOSPHERE#OFF_SOURCE,CALIBRATE_
WVR#AMBIENT,CALIBRATE_WVR#HOT,CALIBRATE_WVR#OFF_SOURCE]
              13:28:23.8 - 13:36:03.2     9      0 Sun                    31711160
[0,1,2,3,4,5,6,7,8,9,10,11,12]  [0.016, 0.016, 0.016, 0.016, 1.15, 0.192, 0.192, 0.192, 0.192,
0.192, 0.192, 0.192, 0.192] [OBSERVE_TARGET#OFF_SOURCE,OBSERVE_TARGET#ON_SOURCE]
              13:36:12.7 - 13:37:13.4    10      3 J1700-2610              4261504
[0,1,2,3,4,5,6,7,8,9,10,11,12]  [0.016, 0.016, 0.016, 0.016, 1.15, 0.192, 0.192, 0.192, 0.192,
0.192, 0.192, 0.192, 0.192] [CALIBRATE_PHASE#ON_SOURCE,CALIBRATE_WVR#ON_SOURCE]
           (nRows = Total number of rows per scan)
Fields: 4
  ID   Code Name                RA               Decl           Epoch   SrcId      nRows
  0    none Sun                 17:32:41.451514 -23.13.55.18836 ICRS    0       33605104
  1    none J1256-0547          12:56:11.166570 -05.47.21.52520 ICRS    1       27224080
  2    none J1624-3213          16:24:59.619634 -32.13.24.43829 ICRS    2        2141776
  3    none J1700-2610          17:00:53.154063 -26.10.51.72542 ICRS    3        8523060
Spectral Windows:  (37 unique spectral windows and 2 unique polarization setups)
  SpwID  Name                                      #Chans   Frame   Ch0(MHz)  ChanWid(kHz)
TotBW(kHz) CtrFreq(MHz) BBC Num  Corrs
  0      X498997425#ALMA_RB_05#BB_1#SQLD               1   TOPO  191021.000   2000000.000
2000000.0 191021.0000        1  XX  YY
  1      X498997425#ALMA_RB_05#BB_2#SQLD               1   TOPO  192979.000   2000000.000
2000000.0 192979.0000        2  XX  YY
  2      X498997425#ALMA_RB_05#BB_3#SQLD               1   TOPO  203021.000   2000000.000
```

```
2000000.0 203021.0000        3  XX  YY
  3      X498997425#ALMA_RB_05#BB_4#SQLD                1   TOPO  204979.000   2000000.000
2000000.0 204979.0000        4  XX  YY
  4      WVR#NOMINAL                                    4   TOPO  184550.000   1500000.000
7500000.0 187550.0000        0  XX
  5      X498997425#ALMA_RB_05#BB_1#SW-01#FULL_RES    128   TOPO  192013.188    -15625.000
2000000.0 191021.0000        1  XX  YY
  6      X498997425#ALMA_RB_05#BB_1#SW-01#CH_AVG        1   TOPO  191005.375   1796875.000
1796875.0 191005.3750        1  XX  YY
  7      X498997425#ALMA_RB_05#BB_2#SW-01#FULL_RES    128   TOPO  193971.188    -15625.000
2000000.0 192979.0000        2  XX  YY
  8      X498997425#ALMA_RB_05#BB_2#SW-01#CH_AVG        1   TOPO  192963.375   1796875.000
1796875.0 192963.3750        2  XX  YY
  9      X498997425#ALMA_RB_05#BB_3#SW-01#FULL_RES    128   TOPO  202028.812     15625.000
2000000.0 203021.0000        3  XX  YY
  10     X498997425#ALMA_RB_05#BB_3#SW-01#CH_AVG        1   TOPO  203005.375   1796875.000
1796875.0 203005.3750        3  XX  YY
  11     X498997425#ALMA_RB_05#BB_4#SW-01#FULL_RES    128   TOPO  203986.812     15625.000
2000000.0 204979.0000        4  XX  YY
  12     X498997425#ALMA_RB_05#BB_4#SW-01#CH_AVG        1   TOPO  204963.375   1796875.000
1796875.0 204963.3750        4  XX  YY
  13     X498997425#ALMA_RB_05#BB_1#SQLD                1   TOPO  187021.000   2000000.000
2000000.0 187021.0000        1  XX  YY
  14     X498997425#ALMA_RB_05#BB_2#SQLD                1   TOPO  187021.000   2000000.000
2000000.0 187021.0000        2  XX  YY
  15     X498997425#ALMA_RB_05#BB_3#SQLD                1   TOPO  187021.000   2000000.000
2000000.0 187021.0000        3  XX  YY
  16     X498997425#ALMA_RB_05#BB_4#SQLD                1   TOPO  187021.000   2000000.000
2000000.0 187021.0000        4  XX  YY
  17     X498997425#ALMA_RB_05#BB_1#SW-01#FULL_RES    128   TOPO  188013.188    -15625.000
2000000.0 187021.0000        1  XX  YY
  18     X498997425#ALMA_RB_05#BB_1#SW-01#CH_AVG        1   TOPO  187005.375   1796875.000
1796875.0 187005.3750        1  XX  YY
  19     X498997425#ALMA_RB_05#BB_2#SW-01#FULL_RES    128   TOPO  188013.188    -15625.000
2000000.0 187021.0000        2  XX  YY
  20     X498997425#ALMA_RB_05#BB_2#SW-01#CH_AVG        1   TOPO  187005.375   1796875.000
1796875.0 187005.3750        2  XX  YY
  21     X498997425#ALMA_RB_05#BB_3#SW-01#FULL_RES    128   TOPO  207986.812     15625.000
2000000.0 208979.0000        3  XX  YY
  22     X498997425#ALMA_RB_05#BB_3#SW-01#CH_AVG        1   TOPO  208963.375   1796875.000
1796875.0 208963.3750        3  XX  YY
  23     X498997425#ALMA_RB_05#BB_4#SW-01#FULL_RES    128   TOPO  207986.812     15625.000
2000000.0 208979.0000        4  XX  YY
  24     X498997425#ALMA_RB_05#BB_4#SW-01#CH_AVG        1   TOPO  208963.375   1796875.000
1796875.0 208963.3750        4  XX  YY
  25     X498997425#ALMA_RB_05#BB_1#SQLD                1   TOPO  196021.000   2000000.000
2000000.0 196021.0000        1  XX  YY
  26     X498997425#ALMA_RB_05#BB_2#SQLD                1   TOPO  197979.000   2000000.000
2000000.0 197979.0000        2  XX  YY
  27     X498997425#ALMA_RB_05#BB_3#SQLD                1   TOPO  208021.000   2000000.000
2000000.0 208021.0000        3  XX  YY
  28     X498997425#ALMA_RB_05#BB_4#SQLD                1   TOPO  209979.000   2000000.000
2000000.0 209979.0000        4  XX  YY
  29     X498997425#ALMA_RB_05#BB_1#SW-01#FULL_RES    128   TOPO  197013.188    -15625.000
2000000.0 196021.0000        1  XX  YY
  30     X498997425#ALMA_RB_05#BB_1#SW-01#CH_AVG        1   TOPO  195997.562   1781250.000
1781250.0 195997.5625        1  XX  YY
  31     X498997425#ALMA_RB_05#BB_2#SW-01#FULL_RES    128   TOPO  198971.188    -15625.000
2000000.0 197979.0000        2  XX  YY
  32     X498997425#ALMA_RB_05#BB_2#SW-01#CH_AVG        1   TOPO  197955.562   1781250.000
1781250.0 197955.5625        2  XX  YY
  33     X498997425#ALMA_RB_05#BB_3#SW-01#FULL_RES    128   TOPO  207028.812     15625.000
2000000.0 208021.0000        3  XX  YY
  34     X498997425#ALMA_RB_05#BB_3#SW-01#CH_AVG        1   TOPO  207997.562   1781250.000
1781250.0 207997.5625        3  XX  YY
  35     X498997425#ALMA_RB_05#BB_4#SW-01#FULL_RES    128   TOPO  208986.812     15625.000
2000000.0 209979.0000        4  XX  YY
  36     X498997425#ALMA_RB_05#BB_4#SW-01#CH_AVG        1   TOPO  209955.562   1781250.000
```

1781250.0 209955.5625 4 XX YY

Sources: 300

ID	Name	SpwId	RestFreq(MHz)	SysVel(km/s)
0	Sun	0	-	-
0	Sun	1	-	-
0	Sun	2	-	-
0	Sun	3	-	-
0	Sun	4	-	-
0	Sun	37	-	-
0	Sun	38	-	-
0	Sun	39	-	-
0	Sun	40	-	-
0	Sun	41	-	-
0	Sun	42	-	-
0	Sun	43	-	-
0	Sun	44	-	-
0	Sun	45	-	-
0	Sun	46	-	-
0	Sun	47	-	-
0	Sun	48	-	-
0	Sun	49	-	-
0	Sun	50	-	-
0	Sun	51	-	-
0	Sun	52	-	-
0	Sun	53	-	-
0	Sun	54	-	-
0	Sun	55	-	-
0	Sun	56	-	-
0	Sun	57	-	-
0	Sun	58	-	-
0	Sun	59	-	-
0	Sun	60	-	-
0	Sun	61	-	-
0	Sun	62	-	-
0	Sun	63	-	-
0	Sun	64	-	-
0	Sun	65	-	-
0	Sun	66	-	-
0	Sun	67	-	-
0	Sun	68	-	-
0	Sun	69	-	-
0	Sun	70	-	-
0	Sun	71	-	-
0	Sun	72	-	-
0	Sun	73	-	-
0	Sun	74	-	-
0	Sun	75	-	-
0	Sun	76	-	-
0	Sun	77	-	-
0	Sun	78	-	-
0	Sun	79	-	-
0	Sun	80	-	-
0	Sun	81	-	-
0	Sun	82	-	-
0	Sun	83	-	-
0	Sun	84	-	-
0	Sun	85	-	-
0	Sun	86	-	-
0	Sun	87	-	-
0	Sun	88	-	-
0	Sun	5	191000	0
0	Sun	6	191000	0
0	Sun	7	193000	0
0	Sun	8	193000	0
0	Sun	9	203000	0
0	Sun	10	203000	0
0	Sun	11	205000	0
0	Sun	12	205000	0

0	Sun	13	–	–
0	Sun	14	–	–
0	Sun	15	–	–
0	Sun	16	–	–
0	Sun	17	191000	0
0	Sun	18	191000	0
0	Sun	19	193000	0
0	Sun	20	193000	0
0	Sun	21	203000	0
0	Sun	22	203000	0
0	Sun	23	205000	0
0	Sun	24	205000	0
1	J1256-0547	4	–	–
1	J1256-0547	37	–	–
1	J1256-0547	38	–	–
1	J1256-0547	39	–	–
1	J1256-0547	40	–	–
1	J1256-0547	41	–	–
1	J1256-0547	42	–	–
1	J1256-0547	43	–	–
1	J1256-0547	44	–	–
1	J1256-0547	45	–	–
1	J1256-0547	46	–	–
1	J1256-0547	47	–	–
1	J1256-0547	48	–	–
1	J1256-0547	49	–	–
1	J1256-0547	50	–	–
1	J1256-0547	51	–	–
1	J1256-0547	52	–	–
1	J1256-0547	53	–	–
1	J1256-0547	54	–	–
1	J1256-0547	55	–	–
1	J1256-0547	56	–	–
1	J1256-0547	57	–	–
1	J1256-0547	58	–	–
1	J1256-0547	59	–	–
1	J1256-0547	60	–	–
1	J1256-0547	61	–	–
1	J1256-0547	62	–	–
1	J1256-0547	63	–	–
1	J1256-0547	64	–	–
1	J1256-0547	65	–	–
1	J1256-0547	66	–	–
1	J1256-0547	67	–	–
1	J1256-0547	68	–	–
1	J1256-0547	69	–	–
1	J1256-0547	70	–	–
1	J1256-0547	71	–	–
1	J1256-0547	72	–	–
1	J1256-0547	73	–	–
1	J1256-0547	74	–	–
1	J1256-0547	75	–	–
1	J1256-0547	76	–	–
1	J1256-0547	77	–	–
1	J1256-0547	78	–	–
1	J1256-0547	79	–	–
1	J1256-0547	80	–	–
1	J1256-0547	81	–	–
1	J1256-0547	82	–	–
1	J1256-0547	83	–	–
1	J1256-0547	84	–	–
1	J1256-0547	85	–	–
1	J1256-0547	86	–	–
1	J1256-0547	87	–	–
1	J1256-0547	88	–	–
1	J1256-0547	5	191000	0
1	J1256-0547	6	191000	0
1	J1256-0547	7	193000	0

1	J1256-0547	8	193000	0
1	J1256-0547	9	203000	0
1	J1256-0547	10	203000	0
1	J1256-0547	11	205000	0
1	J1256-0547	12	205000	0
1	J1256-0547	25	–	–
1	J1256-0547	26	–	–
1	J1256-0547	27	–	–
1	J1256-0547	28	–	–
1	J1256-0547	29	–	–
1	J1256-0547	30	–	–
1	J1256-0547	31	–	–
1	J1256-0547	32	–	–
1	J1256-0547	33	–	–
1	J1256-0547	34	–	–
1	J1256-0547	35	–	–
1	J1256-0547	36	–	–
1	J1256-0547	0	–	–
1	J1256-0547	1	–	–
1	J1256-0547	2	–	–
1	J1256-0547	3	–	–
2	J1624-3213	4	–	–
2	J1624-3213	37	–	–
2	J1624-3213	38	–	–
2	J1624-3213	39	–	–
2	J1624-3213	40	–	–
2	J1624-3213	41	–	–
2	J1624-3213	42	–	–
2	J1624-3213	43	–	–
2	J1624-3213	44	–	–
2	J1624-3213	45	–	–
2	J1624-3213	46	–	–
2	J1624-3213	47	–	–
2	J1624-3213	48	–	–
2	J1624-3213	49	–	–
2	J1624-3213	50	–	–
2	J1624-3213	51	–	–
2	J1624-3213	52	–	–
2	J1624-3213	53	–	–
2	J1624-3213	54	–	–
2	J1624-3213	55	–	–
2	J1624-3213	56	–	–
2	J1624-3213	57	–	–
2	J1624-3213	58	–	–
2	J1624-3213	59	–	–
2	J1624-3213	60	–	–
2	J1624-3213	61	–	–
2	J1624-3213	62	–	–
2	J1624-3213	63	–	–
2	J1624-3213	64	–	–
2	J1624-3213	65	–	–
2	J1624-3213	66	–	–
2	J1624-3213	67	–	–
2	J1624-3213	68	–	–
2	J1624-3213	69	–	–
2	J1624-3213	70	–	–
2	J1624-3213	71	–	–
2	J1624-3213	72	–	–
2	J1624-3213	73	–	–
2	J1624-3213	74	–	–
2	J1624-3213	75	–	–
2	J1624-3213	76	–	–
2	J1624-3213	77	–	–
2	J1624-3213	78	–	–
2	J1624-3213	79	–	–
2	J1624-3213	80	–	–
2	J1624-3213	81	–	–
2	J1624-3213	82	–	–

2	J1624-3213	83	–	–
2	J1624-3213	84	–	–
2	J1624-3213	85	–	–
2	J1624-3213	86	–	–
2	J1624-3213	87	–	–
2	J1624-3213	88	–	–
2	J1624-3213	5	191000	0
2	J1624-3213	6	191000	0
2	J1624-3213	7	193000	0
2	J1624-3213	8	193000	0
2	J1624-3213	9	203000	0
2	J1624-3213	10	203000	0
2	J1624-3213	11	205000	0
2	J1624-3213	12	205000	0
2	J1624-3213	25	–	–
2	J1624-3213	26	–	–
2	J1624-3213	27	–	–
2	J1624-3213	28	–	–
2	J1624-3213	29	–	–
2	J1624-3213	30	–	–
2	J1624-3213	31	–	–
2	J1624-3213	32	–	–
2	J1624-3213	33	–	–
2	J1624-3213	34	–	–
2	J1624-3213	35	–	–
2	J1624-3213	36	–	–
3	J1700-2610	4	–	–
3	J1700-2610	37	–	–
3	J1700-2610	38	–	–
3	J1700-2610	39	–	–
3	J1700-2610	40	–	–
3	J1700-2610	41	–	–
3	J1700-2610	42	–	–
3	J1700-2610	43	–	–
3	J1700-2610	44	–	–
3	J1700-2610	45	–	–
3	J1700-2610	46	–	–
3	J1700-2610	47	–	–
3	J1700-2610	48	–	–
3	J1700-2610	49	–	–
3	J1700-2610	50	–	–
3	J1700-2610	51	–	–
3	J1700-2610	52	–	–
3	J1700-2610	53	–	–
3	J1700-2610	54	–	–
3	J1700-2610	55	–	–
3	J1700-2610	56	–	–
3	J1700-2610	57	–	–
3	J1700-2610	58	–	–
3	J1700-2610	59	–	–
3	J1700-2610	60	–	–
3	J1700-2610	61	–	–
3	J1700-2610	62	–	–
3	J1700-2610	63	–	–
3	J1700-2610	64	–	–
3	J1700-2610	65	–	–
3	J1700-2610	66	–	–
3	J1700-2610	67	–	–
3	J1700-2610	68	–	–
3	J1700-2610	69	–	–
3	J1700-2610	70	–	–
3	J1700-2610	71	–	–
3	J1700-2610	72	–	–
3	J1700-2610	73	–	–
3	J1700-2610	74	–	–
3	J1700-2610	75	–	–
3	J1700-2610	76	–	–
3	J1700-2610	77	–	–

3	J1700-2610	78	-	-	
3	J1700-2610	79	-	-	
3	J1700-2610	80	-	-	
3	J1700-2610	81	-	-	
3	J1700-2610	82	-	-	
3	J1700-2610	83	-	-	
3	J1700-2610	84	-	-	
3	J1700-2610	85	-	-	
3	J1700-2610	86	-	-	
3	J1700-2610	87	-	-	
3	J1700-2610	88	-	-	
3	J1700-2610	29	-	-	
3	J1700-2610	30	-	-	
3	J1700-2610	31	-	-	
3	J1700-2610	32	-	-	
3	J1700-2610	33	-	-	
3	J1700-2610	34	-	-	
3	J1700-2610	35	-	-	
3	J1700-2610	36	-	-	
3	J1700-2610	0	-	-	
3	J1700-2610	1	-	-	
3	J1700-2610	2	-	-	
3	J1700-2610	3	-	-	
3	J1700-2610	5	191000	0	
3	J1700-2610	6	191000	0	
3	J1700-2610	7	193000	0	
3	J1700-2610	8	193000	0	
3	J1700-2610	9	203000	0	
3	J1700-2610	10	203000	0	
3	J1700-2610	11	205000	0	
3	J1700-2610	12	205000	0	

Antennas: 52:

ID	Name	Station	Diam.	Long.	Lat.	Offset from array center (m) East	North
		ITRF Geocentric coordinates (m)					
Elevation		x		y	z		
0	CM01	N602	7.0 m	-067.45.17.4	-22.53.22.3	8.8037	-527.8580
22.2055	2225080.355228	-5440132.958128	-2481524.789864				
1	CM02	J502	7.0 m	-067.45.17.7	-22.53.23.0	2.1071	-549.4456
22.1477	2225070.958281	-5440127.672185	-2481544.655258				
2	CM03	J503	7.0 m	-067.45.17.4	-22.53.23.2	9.2481	-555.0631
22.1325	2225076.735107	-5440122.933576	-2481549.824492				
3	CM04	N605	7.0 m	-067.45.17.4	-22.53.23.9	9.6880	-575.8356
22.0879	2225074.068012	-5440115.250608	-2481568.943967				
4	CM06	N606	7.0 m	-067.45.17.1	-22.53.23.6	19.1994	-566.5678
22.1031	2225084.241158	-5440114.999494	-2481560.411854				
5	CM07	N601	7.0 m	-067.45.17.0	-22.53.22.5	21.0632	-532.5809
22.1877	2225091.000261	-5440126.601762	-2481529.134056				
6	CM09	N603	7.0 m	-067.45.17.7	-22.53.22.3	-0.0497	-527.8645
22.1928	2225072.155639	-5440136.296531	-2481524.791008				
7	CM10	J501	7.0 m	-067.45.17.4	-22.53.22.9	10.0861	-545.4953
22.1622	2225078.929912	-5440126.086166	-2481541.021606				
8	CM11	N604	7.0 m	-067.45.17.8	-22.53.23.7	-0.2657	-571.8963
22.0852	2225065.434567	-5440120.434627	-2481565.313800				
9	DA41	A083	12.0 m	-067.45.22.8	-22.53.21.0	-143.5434	-488.2362
25.7594	2224946.423770	-5440207.923211	-2481489.670186				
10	DA42	A047	12.0 m	-067.45.16.4	-22.53.30.3	38.4531	-775.2182
21.5979	2225071.159752	-5440032.160449	-2481752.434638				
11	DA44	A016	12.0 m	-067.45.16.4	-22.53.25.1	37.4651	-614.5603
21.7887	2225093.970126	-5440090.538761	-2481604.502872				
12	DA45	A101	12.0 m	-067.45.24.0	-22.53.10.5	-177.3499	-163.1864
25.7562	2224962.988323	-5440337.730653	-2481190.209469				
13	DA46	A122	12.0 m	-067.45.55.6	-22.53.04.3	-1079.8558	29.4330
6.2507	2224149.167422	-5440732.003760	-2481005.166290				
14	DA47	A022	12.0 m	-067.45.17.2	-22.53.25.7	14.4394	-633.3728
21.9426	2225069.942470	-5440092.613603	-2481621.893817				
15	DA48	A082	12.0 m	-067.45.08.3	-22.53.29.2	269.0423	-740.9535
16.2833	2225287.767033	-5439952.669973	-2481718.801163				

```
  16   DA49  A066      12.0 m   -067.45.17.6  -22.53.24.6          3.3950     -597.3309
21.8251  2225064.986682 -5440109.670174 -2481588.644429
  17   DA51  A135      12.0 m   -067.45.02.2  -22.53.16.5        442.8905     -348.0857
12.9372  2225505.357228 -5440025.428568 -2481355.567250
  18   DA52  A015      12.0 m   -067.45.15.3  -22.53.26.0         68.8256     -640.1823
21.5282  2225119.131525 -5440069.220242 -2481628.005984
  19   DA53  A085      12.0 m   -067.45.10.5  -22.53.20.5        207.6862     -472.2073
16.8002  2225270.737975 -5440073.091611 -2481471.418756
  20   DA55  A134      12.0 m   -067.45.20.4  -22.53.45.6        -76.0039    -1249.4239
24.2069  2224896.300495 -5439906.962910 -2482190.302240
  21   DA56  A062      12.0 m   -067.45.18.5  -22.53.31.4        -21.4628     -809.5726
22.5655  2225010.984344 -5440043.297652 -2481784.459798
  22   DA57  A094      12.0 m   -067.45.27.0  -22.53.37.1       -264.4594     -984.7315
19.8550  2224759.344760 -5440069.898985 -2481944.768298
  23   DA58  A076      12.0 m   -067.45.20.5  -22.53.33.8        -77.9916     -882.7187
24.5723  2224948.593622 -5440040.070937 -2481852.625688
  24   DA59  A093      12.0 m   -067.45.14.7  -22.53.14.5         86.9869     -286.0414
22.1124  2225188.290980 -5440190.332841 -2481301.976054
  25   DA62  A069      12.0 m   -067.45.21.3  -22.53.30.2       -101.4810     -770.1050
23.8043  2224943.168875 -5440088.854228 -2481748.582478
  26   DA64  A110      12.0 m   -067.45.09.2  -22.53.54.8        244.7432    -1530.9482
13.5074  2225147.947357 -5439675.039018 -2482445.481380
  27   DA65  A045      12.0 m   -067.45.17.9  -22.53.30.1         -5.4184     -767.4395
22.6051  2225032.052228 -5440052.427624 -2481745.660395
  28   DV01  A011      12.0 m   -067.45.14.4  -22.53.28.4         95.9112     -716.4989
21.0931  2225132.809848 -5440031.119522 -2481698.143402
  29   DV02  A024      12.0 m   -067.45.18.3  -22.53.26.8        -16.3966     -665.3089
21.5565  2225036.565223 -5440092.459725 -2481651.164861
  30   DV03  A027      12.0 m   -067.45.19.0  -22.53.28.7        -35.0467     -726.6009
21.6047  2225010.295225 -5440077.493756 -2481707.648797
  31   DV04  A042      12.0 m   -067.45.15.6  -22.53.29.3         60.4757     -743.9313
20.6733  2225095.827142 -5440034.299665 -2481723.252096
  32   DV05  A092      12.0 m   -067.45.10.1  -22.53.39.1        217.6020    -1047.7078
20.8520  2225196.572935 -5439865.592923 -2482003.171782
  33   DV06  A096      12.0 m   -067.45.29.9  -22.53.15.7       -347.1456     -322.7952
22.7984  2224781.300736 -5440342.024685 -2481336.103148
  34   DV07  A086      12.0 m   -067.45.27.0  -22.53.29.3       -262.5029     -744.4838
26.4714  2224798.839058 -5440161.303679 -2481726.016465
  35   DV08  A087      12.0 m   -067.45.08.3  -22.53.33.2        269.0949     -864.0691
16.2389  2225269.667982 -5439908.285863 -2481832.203194
  36   DV09  A073      12.0 m   -067.45.22.9  -22.53.28.1       -147.0734     -705.3304
23.7576  2224910.493113 -5440129.394064 -2481688.890878
  37   DV10  A043      12.0 m   -067.45.15.1  -22.53.29.4         75.1901     -747.1637
20.6663  2225108.967309 -5440027.559553 -2481726.227190
  38   DV11  A007      12.0 m   -067.45.15.1  -22.53.27.3         74.0150     -681.2930
21.3248  2225117.809595 -5440052.281397 -2481665.800107
  39   DV12  A121      12.0 m   -067.44.36.3  -22.53.19.3       1181.3913     -435.7649
-24.4919  2226162.864243 -5439682.292505 -2481421.785374
  40   DV14  A074      12.0 m   -067.45.12.1  -22.53.32.0        161.8142     -828.6189
18.7740  2225176.483822 -5439963.825760 -2481800.531180
  41   DV15  A105      12.0 m   -067.45.00.2  -22.53.43.8        500.1150    -1191.4347
14.9707  2225434.811677 -5439701.866358 -2482133.288250
  42   DV16  A108      12.0 m   -067.44.52.4  -22.53.25.7        722.5812     -632.9600
1.9541  2225718.424978 -5439807.607943 -2481613.738634
  43   DV17  A116      12.0 m   -067.44.49.8  -22.53.07.3        796.8150      -62.7364
2.4465  2225871.284785 -5439985.176851 -2481088.601483
  44   DV18  A070      12.0 m   -067.45.11.9  -22.53.29.3        166.1828     -743.4929
19.3809  2225193.275369 -5439993.337992 -2481722.345469
  45   DV19  A104      12.0 m   -067.45.36.4  -22.53.21.2       -530.9026     -492.4907
20.3387  2224585.379851 -5440348.386669 -2481491.481328
  46   DV20  A072      12.0 m   -067.45.12.6  -22.53.24.0        147.1735     -580.5877
18.1844  2225199.254476 -5440058.163787 -2481571.803516
  47   DV21  A068      12.0 m   -067.45.20.6  -22.53.25.7        -82.4247     -631.7802
23.0805  2224980.922220 -5440130.825161 -2481620.869248
  48   DV22  A088      12.0 m   -067.45.18.2  -22.53.18.4        -13.0909     -408.0098
23.6750  2225078.250111 -5440185.644924 -2481414.949662
  49   DV23  A060      12.0 m   -067.45.15.9  -22.53.32.5         52.7921     -843.4863
21.7260  2225074.421988 -5440002.261967 -2481815.375939
```

50	DV24	A111	12.0 m	−067.45.03.9	−22.53.03.1	394.2957	66.9171	17.6692	2225523.141145	−5440197.241490	−2480975.072800
51	DV25	A033	12.0 m	−067.45.19.4	−22.53.29.0	−47.3625	−735.6347	21.8852	2224997.664006	−5440079.142558	−2481716.080288

```
************************************************************
                        QA2 REPORT
************************************************************

             MEDIAN TSYS WITH SOURCE/ELEVATION
                 (see plots for details)
Scan   Fid               Source      Elev    Median T_x  Median T_y
   1     0                  Sun   46.1      1249        1253
   4     1            J1256-0547   62.6      1094        1083
   8     0                  Sun   49.4      1236        1240



          MEDIAN TSYS versus ANTENNA/SPW and OUTLIERS

             TSYS MEDIAN               >3-sigma OUTLIERS
       XPOL              YPOL               OUTLIERS
 SPW    T   rms        T   rms      antenna  Pol  tsys  n-sigma

 0  5   1426 1190      1450 1229   No outliers
 1  7   1159  944      1140  990   No outliers
 2  9    985  852       966 1048   No outliers
 3 11   1244 1702      1206 1718   No outliers

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

         SUMMARY INFORMATION FOR  uid___A002_X110fd13_X1099e.ms.split

   Experiment Duration:  2023/12/16/13:17:58 to
                         2023/12/16/13:37:13

   Processed from ms: uid___A002_X110fd13_X1099e.ms.split
   Written to file:   NewListobs.txt

                                SCAN LISTING

  Scan FdId srcId FieldName         StartTime    StopTime     Int(s) Elev  ScanIntent
    5    1    1   J1256-0547        13:17:58.5 - 13:23:00.9   0.19   62.0  Cal Bandpass
    7    3    3   J1700-2610        13:26:45.4 - 13:27:45.8   0.19   56.7  Cal Phase
    9    0    0   Sun               13:28:40.4 - 13:35:47.0   0.19   50.3  Obs Target
   10    3    3   J1700-2610        13:36:13.0 - 13:37:13.4   0.19   58.8  Cal Phase


                             FIELD INFORMATION

 Fid  Srd  Field                       RA  (J2000)    DEC       Fld Time  #Scans
                                                                  (min)
   0    0  Sun                   00:00:00.00000 +00.00.00.0000   7.11      1
   1    1  J1256-0547            12:56:11.16657 -05.47.21.5252   5.04      1
   2    2  J1624-3213            16:24:59.61963 -32.13.24.4383   0.00      0
   3    3  J1700-2610            17:00:53.15406 -26.10.51.7254   2.02      2


                      FREQUENCY INFORMATION

spw  nchan      -----Frequencies (GHz)-------     --Channel Width--
               First         Last     Bandwidth     MHz     km/s     POLN

  0   128   192.013187   190.028812   2.000     -15.625   -24.39  ['XX', 'YY']
  1   128   193.971188   191.986813   2.000     -15.625   -24.14  ['XX', 'YY']
  2   128   202.028812   204.013187   2.000      15.625    23.18  ['XX', 'YY']
  3   128   203.986813   205.971188   2.000      15.625    22.96  ['XX', 'YY']


                     ANTENNA INFORMATION
```

ID	Name	Pad	Size (m)	Longitude	Latitude	E-off (m)	N-off (m)	Elev (m)
0	CM01	N602	7.0	-67.45.17.44	-22.53.22.32	-12.8	15768.3	8.2
1	CM02	J502	7.0	-67.45.17.67	-22.53.23.02	-19.5	15746.7	8.2
2	CM03	J503	7.0	-67.45.17.42	-22.53.23.20	-12.3	15741.1	8.1
3	CM04	N605	7.0	-67.45.17.40	-22.53.23.87	-11.9	15720.3	8.1
4	CM06	N606	7.0	-67.45.17.07	-22.53.23.57	-2.4	15729.6	8.1
5	CM07	N601	7.0	-67.45.17.01	-22.53.22.47	-0.5	15763.5	8.2
6	CM09	N603	7.0	-67.45.17.75	-22.53.22.32	-21.6	15768.3	8.2
7	CM10	J501	7.0	-67.45.17.39	-22.53.22.89	-11.5	15750.6	8.2
8	CM11	N604	7.0	-67.45.17.75	-22.53.23.74	-21.8	15724.2	8.1
9	DA41	A083	12.0	-67.45.22.78	-22.53.21.04	-165.1	15807.9	11.8
10	DA42	A047	12.0	-67.45.16.39	-22.53.30.32	16.9	15520.9	7.6
11	DA44	A016	12.0	-67.45.16.43	-22.53.25.12	15.9	15681.6	7.8
12	DA45	A101	12.0	-67.45.23.97	-22.53.10.53	-198.9	16132.9	11.8
13	DA46	A122	12.0	-67.45.55.64	-22.53.04.30	-1101.4	16325.6	-7.7
14	DA47	A022	12.0	-67.45.17.24	-22.53.25.73	-7.1	15662.8	8.0
15	DA48	A082	12.0	-67.45.08.30	-22.53.29.21	247.5	15555.2	2.3
16	DA49	A066	12.0	-67.45.17.63	-22.53.24.57	-18.2	15698.8	7.8
17	DA51	A135	12.0	-67.45.02.20	-22.53.16.51	421.3	15948.0	-1.0
18	DA52	A015	12.0	-67.45.15.33	-22.53.25.95	47.3	15655.9	7.5
19	DA53	A085	12.0	-67.45.10.46	-22.53.20.52	186.1	15823.9	2.8
20	DA55	A134	12.0	-67.45.20.41	-22.53.45.65	-97.6	15046.7	10.2
21	DA56	A062	12.0	-67.45.18.50	-22.53.31.43	-43.0	15486.6	8.6
22	DA57	A094	12.0	-67.45.27.02	-22.53.37.09	-286.0	15311.4	5.9
23	DA58	A076	12.0	-67.45.20.48	-22.53.33.79	-99.6	15413.4	10.6
24	DA59	A093	12.0	-67.45.14.69	-22.53.14.50	65.4	16010.1	8.1
25	DA62	A069	12.0	-67.45.21.31	-22.53.30.15	-123.0	15526.0	9.8
26	DA64	A110	12.0	-67.45.09.16	-22.53.54.75	223.2	14765.2	-0.5
27	DA65	A045	12.0	-67.45.17.93	-22.53.30.07	-27.0	15528.7	8.6
28	DV01	A011	12.0	-67.45.14.38	-22.53.28.42	74.3	15579.6	7.1
29	DV02	A024	12.0	-67.45.18.32	-22.53.26.76	-38.0	15630.8	7.6
30	DV03	A027	12.0	-67.45.18.97	-22.53.28.75	-56.6	15569.5	7.6
31	DV04	A042	12.0	-67.45.15.62	-22.53.29.31	38.9	15552.2	6.7
32	DV05	A092	12.0	-67.45.10.11	-22.53.39.13	196.0	15248.4	6.9
33	DV06	A096	12.0	-67.45.29.93	-22.53.15.69	-368.7	15973.3	8.8
34	DV07	A086	12.0	-67.45.26.96	-22.53.29.32	-284.1	15551.6	12.5
35	DV08	A087	12.0	-67.45.08.30	-22.53.33.19	247.5	15432.1	2.3
36	DV09	A073	12.0	-67.45.22.91	-22.53.28.06	-168.6	15590.8	9.8
37	DV10	A043	12.0	-67.45.15.11	-22.53.29.41	53.6	15549.0	6.7
38	DV11	A007	12.0	-67.45.15.15	-22.53.27.28	52.4	15614.8	7.3
39	DV12	A121	12.0	-67.44.36.29	-22.53.19.34	1159.8	15860.4	-38.5
40	DV14	A074	12.0	-67.45.12.07	-22.53.32.04	140.2	15467.5	4.8
41	DV15	A105	12.0	-67.45.00.19	-22.53.43.77	478.5	15104.7	1.0
42	DV16	A108	12.0	-67.44.52.39	-22.53.25.72	701.0	15663.2	-12.0
43	DV17	A116	12.0	-67.44.49.78	-22.53.07.28	775.2	16233.4	-11.5
44	DV18	A070	12.0	-67.45.11.91	-22.53.29.29	144.6	15552.6	5.4
45	DV19	A104	12.0	-67.45.36.38	-22.53.21.18	-552.5	15803.6	6.4
46	DV20	A072	12.0	-67.45.12.58	-22.53.24.03	125.6	15715.5	4.2
47	DV21	A068	12.0	-67.45.20.64	-22.53.25.68	-104.0	15664.3	9.1
48	DV22	A088	12.0	-67.45.18.20	-22.53.18.45	-34.7	15888.1	9.7
49	DV23	A060	12.0	-67.45.15.89	-22.53.32.52	31.2	15452.6	7.7
50	DV24	A111	12.0	-67.45.03.91	-22.53.03.09	372.7	16363.0	3.7
51	DV25	A033	12.0	-67.45.19.41	-22.53.29.04	-68.9	15560.5	7.9

****

MEDIAN GAIN VALUE = 0.168

RELATIVE ANTENNA GAIN FOR BANDPASS OBSERVATION

Antenna	SPW 0		SPW 1		SPW 2		SPW 3	
	X	Y	X	Y	X	Y	X	Y
0-CM01	0.58**	0.61**	0.58**	0.61**	0.58**	0.61**	0.57**	0.61**
1-CM02	0.57**	0.60**	0.57**	0.61**	0.57**	0.60**	0.57**	0.60**

2-CM03	0.58**	0.61**	0.58**	0.60**	0.58**	0.60**	0.58**	0.61**
3-CM04	0.57**	0.61**	0.57**	0.61**	0.57**	0.60**	0.58**	0.60**
4-CM06	0.56**	0.61**	0.57**	0.61**	0.57**	0.60**	0.57**	0.61**
5-CM07	flagged	flagged	flagged	flagged	flagged	flagged	flagged	flagged
6-CM09	0.57**	0.58**	0.57**	0.59**	0.58**	0.60**	0.56**	0.57**
7-CM10	0.57**	0.60**	0.57**	0.60**	0.57**	0.60**	0.57**	0.60**
8-CM11	0.55**	0.59**	0.56**	0.60**	0.56**	0.58**	0.56**	0.59**
9-DA41	0.98	1.04	0.98	1.04	0.98	1.03	0.99	1.03
10-DA42	0.97	1.03	0.98	1.03	0.97	1.02	0.97	1.02
11-DA44	0.99	1.04	0.98	1.04	0.99	1.04	0.98	1.04
12-DA45	0.95	1.03	0.97	1.03	0.99	1.03	0.97	1.04
13-DA46	0.95	1.05	0.99	1.04	0.99	1.04	0.98	1.04
14-DA47	0.99	1.05	1.00	1.05	1.00	1.04	0.99	1.06
15-DA48	flagged	flagged	flagged	flagged	flagged	flagged	flagged	flagged
16-DA49	0.97	1.04	0.99	1.04	0.99	1.03	0.98	1.03
17-DA51	0.96	1.00	0.97	1.02	0.96	1.02	0.97	1.01
18-DA52	0.97	1.04	0.99	1.05	0.98	1.05	0.98	1.06
19-DA53	0.97	1.03	0.98	1.03	0.96	1.02	0.97	1.03
20-DA55	0.99	1.03	0.99	1.04	0.98	1.03	0.98	1.03
21-DA56	0.98	1.02	0.98	1.03	0.97	1.02	0.96	1.02
22-DA57	0.99	1.05	0.99	1.05	0.99	1.04	0.99	1.04
23-DA58	0.99	1.04	0.99	1.04	0.99	1.03	0.99	1.04
24-DA59	0.99	1.03	0.99	1.03	0.98	1.03	0.98	1.03
25-DA62	0.98	1.03	0.99	1.04	0.98	1.04	0.99	1.04
26-DA64	0.99	1.05	1.00	1.05	0.99	1.04	0.99	1.05
27-DA65	0.99	1.04	1.00	1.04	0.98	1.04	0.99	1.03
28-DV01	1.01	1.06	1.01	1.07	1.01	1.07	1.01	1.07
29-DV02	1.00	1.05	1.01	1.06	1.01	1.07	1.00	1.06
30-DV03	1.00	1.05	1.01	1.06	1.00	1.05	1.00	1.06
31-DV04	1.01	1.06	1.01	1.06	1.01	1.06	1.01	1.07
32-DV05	1.01	1.06	1.01	1.05	0.99	1.04	1.00	1.03
33-DV06	1.00	1.06	1.00	1.06	1.00	1.06	1.00	1.05
34-DV07	1.00	1.04	0.99	1.06	1.00	1.05	0.99	1.04
35-DV08	0.99	1.04	1.00	1.05	0.99	1.04	0.99	1.04
36-DV09	1.01	1.03	1.02	1.05	1.01	1.07	1.01	1.05
37-DV10	1.00	1.06	1.01	1.06	1.00	1.06	1.01	1.06
38-DV11	1.00	1.06	1.01	1.06	0.99	1.05	1.00	1.05
39-DV12	0.97	1.04	0.99	1.04	0.97	1.03	0.98	1.03
40-DV14	1.00	1.06	0.99	1.05	1.00	1.05	1.00	1.06
41-DV15	0.99	1.03	0.99	1.04	0.99	1.05	0.99	1.04
42-DV16	0.96	1.04	0.98	1.03	0.99	1.04	0.98	1.04
43-DV17	0.98	1.02	0.99	1.04	0.99	1.03	0.98	1.04
44-DV18	0.99	1.05	1.01	1.05	0.99	1.05	0.99	1.04
45-DV19	0.99	1.03	1.00	1.05	0.99	1.05	0.99	1.05
46-DV20	0.98	1.04	0.98	1.03	0.97	1.00	0.98	0.99
47-DV21	0.99	1.05	0.99	1.04	1.00	1.05	1.00	1.03
48-DV22	0.99	1.05	1.00	1.05	0.99	1.05	1.00	1.05
49-DV23	1.01	1.05	1.00	1.06	1.00	1.05	0.99	1.04
50-DV24	0.95	1.00	0.96	1.00	0.94	1.00	0.94	0.99
51-DV25	0.99	1.02	0.99	1.04	1.00	1.02	1.00	1.01

** means outside of normalized range: 0.90 to 1.10

****

NORMALIZED BANDPASS AMPLITUDE RMS OVER 128 CHANNELS
using table uid___A002_X110fd13_X1099e.ms.split.bandpass

Antenna	SPW0		SPW1		SPW2		SPW3	
	X	Y	X	Y	X	Y	X	Y
0-CM01	2.507	2.359	2.483	2.362	2.488	2.361	2.515	2.367
1-CM02	2.530	2.372	2.531	2.362	2.533	2.380	2.536	2.388
2-CM03	2.504	2.337	2.488	2.374	2.500	2.397	2.486	2.358
3-CM04	2.553	2.365	2.511	2.370	2.520	2.391	2.490	2.386
4-CM06	2.564	2.367	2.511	2.364	2.511	2.377	2.534	2.365

5-CM07	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6-CM09	2.513	2.466	2.525	2.415	2.501	2.392	2.563	2.514
7-CM10	2.532	2.383	2.544	2.375	2.543	2.394	2.550	2.414
8-CM11	2.632	2.433	2.600	2.414	2.604	2.461	2.586	2.431
9-DA41	1.355	1.270	1.356	1.268	1.364	1.276	1.351	1.278
10-DA42	1.373	1.276	1.369	1.280	1.384	1.299	1.377	1.301
11-DA44	1.347	1.262	1.356	1.266	1.349	1.270	1.356	1.267
12-DA45	1.409	1.285	1.374	1.287	1.350	1.285	1.379	1.261
13-DA46	1.413	1.251	1.345	1.274	1.343	1.263	1.369	1.271
14-DA47	1.339	1.252	1.330	1.258	1.330	1.264	1.349	1.241
15-DA48	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
16-DA49	1.371	1.268	1.352	1.266	1.351	1.275	1.361	1.278
17-DA51	1.387	1.326	1.378	1.298	1.386	1.301	1.383	1.312
18-DA52	1.373	1.260	1.339	1.252	1.357	1.248	1.357	1.238
19-DA53	1.380	1.284	1.366	1.285	1.387	1.289	1.379	1.289
20-DA55	1.345	1.277	1.351	1.270	1.361	1.282	1.359	1.280
21-DA56	1.367	1.298	1.369	1.287	1.376	1.295	1.388	1.297
22-DA57	1.342	1.249	1.343	1.253	1.345	1.261	1.348	1.266
23-DA58	1.341	1.271	1.337	1.274	1.348	1.277	1.343	1.262
24-DA59	1.351	1.274	1.345	1.277	1.364	1.281	1.353	1.285
25-DA62	1.353	1.280	1.347	1.268	1.363	1.272	1.347	1.259
26-DA64	1.341	1.253	1.334	1.253	1.339	1.264	1.344	1.255
27-DA65	1.338	1.262	1.329	1.271	1.353	1.264	1.343	1.279
28-DV01	1.309	1.240	1.312	1.227	1.305	1.225	1.308	1.217
29-DV02	1.326	1.253	1.307	1.238	1.317	1.231	1.327	1.238
30-DV03	1.334	1.253	1.317	1.238	1.329	1.248	1.323	1.243
31-DV04	1.318	1.239	1.319	1.240	1.317	1.238	1.318	1.228
32-DV05	1.310	1.241	1.312	1.251	1.337	1.269	1.329	1.286
33-DV06	1.324	1.241	1.326	1.240	1.328	1.241	1.335	1.247
34-DV07	1.329	1.268	1.340	1.240	1.335	1.258	1.343	1.273
35-DV08	1.343	1.270	1.322	1.256	1.347	1.262	1.340	1.260
36-DV09	1.308	1.276	1.304	1.249	1.305	1.230	1.308	1.249
37-DV10	1.322	1.232	1.316	1.233	1.325	1.236	1.310	1.241
38-DV11	1.326	1.233	1.315	1.234	1.339	1.246	1.332	1.250
39-DV12	1.374	1.266	1.348	1.263	1.383	1.279	1.365	1.280
40-DV14	1.322	1.238	1.337	1.248	1.335	1.251	1.333	1.244
41-DV15	1.341	1.282	1.341	1.269	1.344	1.254	1.337	1.274
42-DV16	1.401	1.269	1.364	1.276	1.352	1.273	1.359	1.264
43-DV17	1.361	1.296	1.348	1.269	1.349	1.279	1.355	1.265
44-DV18	1.344	1.247	1.319	1.249	1.349	1.251	1.343	1.260
45-DV19	1.341	1.278	1.333	1.249	1.344	1.257	1.349	1.258
46-DV20	1.359	1.265	1.357	1.278	1.371	1.321	1.363	1.349
47-DV21	1.341	1.256	1.340	1.269	1.325	1.258	1.325	1.279
48-DV22	1.343	1.249	1.324	1.254	1.336	1.258	1.328	1.249
49-DV23	1.315	1.252	1.322	1.238	1.333	1.247	1.338	1.266
50-DV24	1.408	1.331	1.403	1.332	1.427	1.335	1.431	1.351
51-DV25	1.344	1.296	1.338	1.262	1.334	1.292	1.334	1.311

BANDPASS RMS MEDIAN VALUE = 1.325
Consider using BPOLY, or B with pre-averaging (e.g. solint="8MHz")

RMS OUTLIERS >3.0 SIGMA

SPW	Xpol		Ypol		
	mean	rms	mean	rms	OUTLIERS
0	1.3445	0.5219	1.2682	0.4897	No Outliers
1	1.3420	0.5179	1.2669	0.4885	No Outliers
2	1.3491	0.5166	1.2692	0.4922	No Outliers
3	1.3487	0.5190	1.2690	0.4949	No Outliers

****

Flux Density Determinations

Reference source J1256-0547

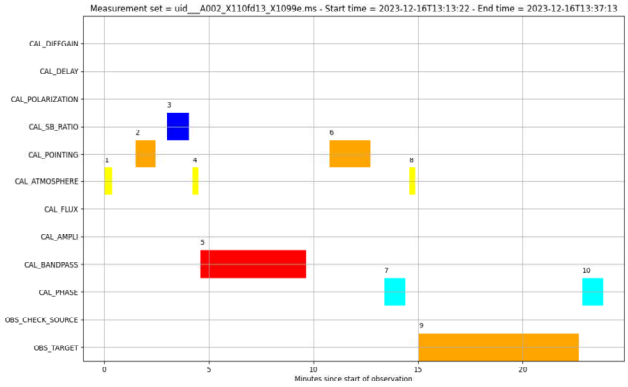
J1700-2610 in SpW=0 (1.91021e+11 Hz) is: 0.715959+-0.00749206 (SNR=95.5624, N=100)
J1700-2610 in SpW=1 (1.92979e+11 Hz) is: 0.690351+-0.00681293 (SNR=101.329, N=100)
J1700-2610 in SpW=2 (2.03021e+11 Hz) is: 0.647312+-0.00554967 (SNR=116.64, N=100)
J1700-2610 in SpW=3 (2.04979e+11 Hz) is: 0.643687+-0.00657912 (SNR=97.8378, N=100)
Fit spectrum J1700-2610: 0.673232+-0.00379604 (197.907 GHz) spidx: a_1 (spectral index)
=-1.42638+-0.187176 covariance matrix the fit: covar(0,0)=2.36333e-05 covar(0,1)=-0.000142162
covar(1,0)=-0.000142162 covar(1,1)=0.138079

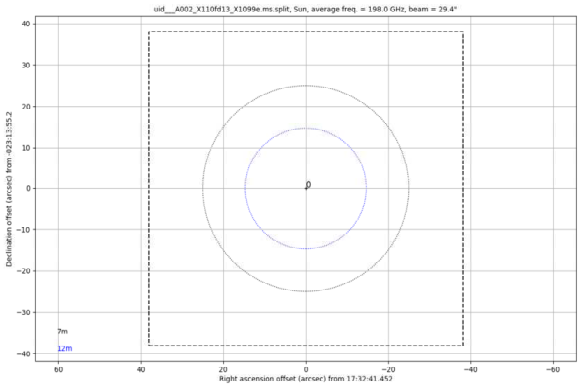
    FLAGGING STATISTICS

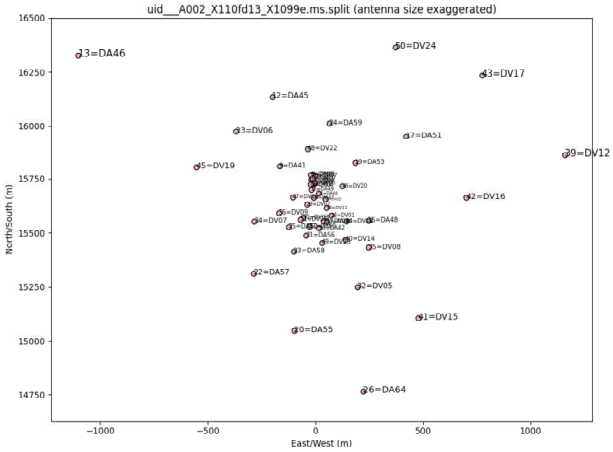
Overall -> 33.41 percent

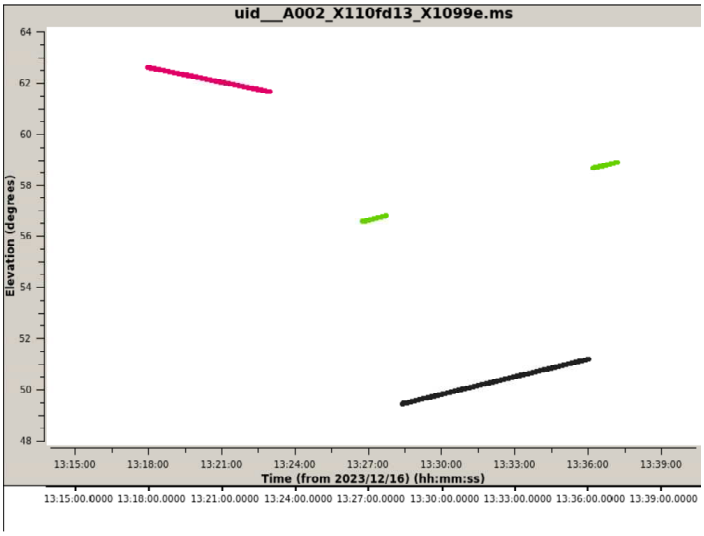
Per spw (over total of dataset):
0 -> 9.23
1 -> 8.05
2 -> 8.23
3 -> 7.90
Per antenna (over total of dataset):
CM01 -> 1.64
CM02 -> 0.99
CM03 -> 0.99
CM04 -> 0.98
CM06 -> 1.06
CM07 -> 3.77
CM09 -> 2.32
CM10 -> 0.99
CM11 -> 0.98
DA41 -> 1.30
DA42 -> 0.98
DA44 -> 0.98
DA45 -> 0.98
DA46 -> 0.98
DA47 -> 0.98
DA48 -> 3.77
DA49 -> 0.98
DA51 -> 0.98
DA52 -> 0.98
DA53 -> 0.98
DA55 -> 1.30
DA56 -> 0.98
DA57 -> 0.98
DA58 -> 0.98
DA59 -> 0.98
DA62 -> 0.98
DA64 -> 0.98
DA65 -> 0.98
DV01 -> 0.98
DV02 -> 0.98
DV03 -> 0.98
DV04 -> 2.31
DV05 -> 1.80
DV06 -> 0.98
DV07 -> 0.98
DV08 -> 0.98
DV09 -> 0.98
DV10 -> 0.98
DV11 -> 0.98
DV12 -> 1.97
DV14 -> 0.98
DV15 -> 0.98
DV16 -> 1.29
DV17 -> 1.29

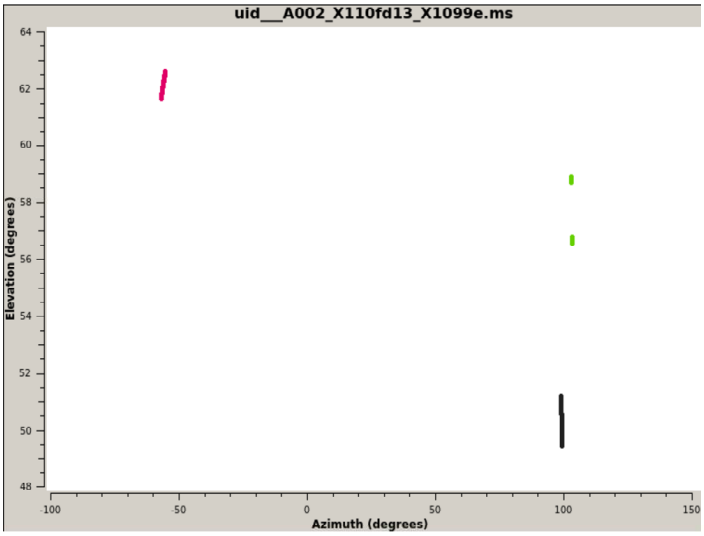
```
DV18 -> 0.98
DV19 -> 0.98
DV20 -> 0.98
DV21 -> 0.98
DV22 -> 0.98
DV23 -> 0.98
DV24 -> 0.98
DV25 -> 0.98


************************************************************

        CHECK OF A TARGET IMAGE AND SENSITIVITY

    longest baseline       =  2308.8 (meters)
    recommended cellsize   =  0.0300 (arcsec)
    primary beam FWHM      =  31.25 (arcsec)
    recommended image size =  1600

    target                 =  0
    resolution             =  0.438 x  0.310 in pa   73.2
    time on target         =   7.08 (min)
    peak on image          = 2924.497 (mJy)
    rms on image           = 1195.578 (mJy)
    expected sensitivity   =   0.428 (mJy) (aggregate bandwidth)


<!-- No flux measurements for the ALMA database since no intent CALIBRATE_FLUX#ON_SOURCE was
found -->
```

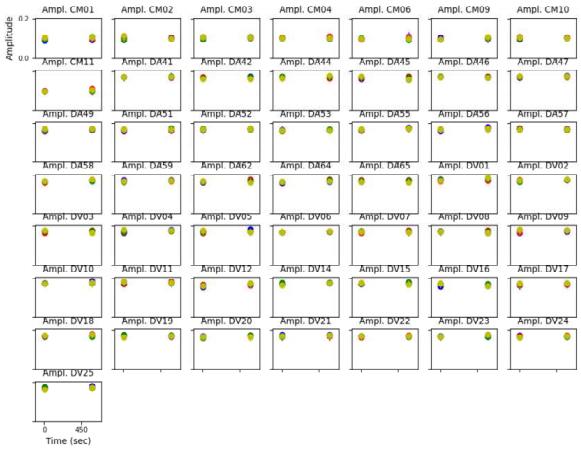

Measurement set = uid___A002_X110fd13_X1099e.ms - Start time = 2023-12-16T13:13:22 - End time = 2023-12-16T13:37:13
CAL_DIFFGAIN
CAL_DELAY
CAL_POLARIZATION
CAL_SB_RATIO
CAL_POINTING
CAL_ATMOSPHERE
CAL_FLUX
CAL_AMPLI
CAL_BANDPASS
CAL_PHASE
OBS_CHECK_SOURCE
OBS_TARGET
Minutes since start of observation

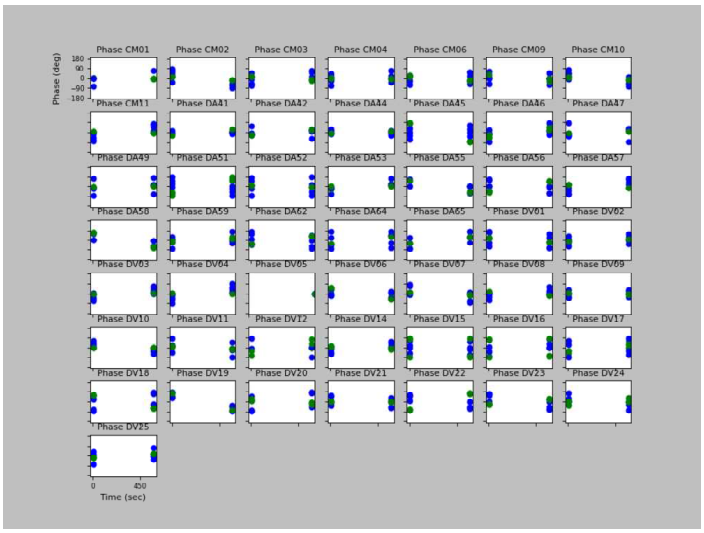

uid___A002_X110fd13_X1099e.ms.split, Sun, average freq. = 198.0 GHz, beam = 29.4"
Declination offset (arcsec) from 423:11:55.2
7m
12m
Right ascension offset (arcsec) from 17:32:41.452

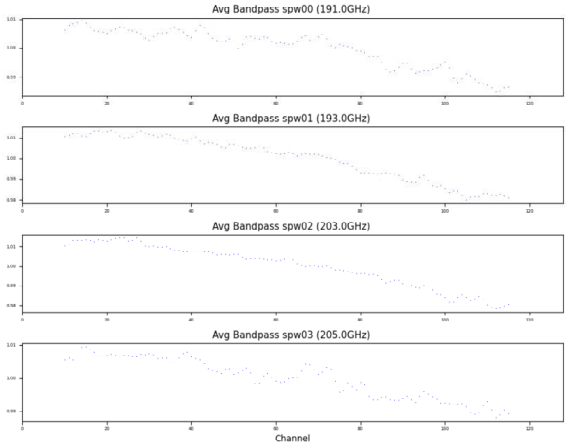

uid___A002_X110fd13_X1099e.ms.split (antenna size exaggerated)
13=DA46
0=DV24
3=DV17
2=DA45
4=DA59
3=DV06
7=DA51
8=DV22
9=DV12
5=DV19
0=DA41
9=DA53
6=DV20
2=DV16
4=DV07
6=DA48
5=DV08
2=DA57
2=DV05
1=DV15
0=DA55
6=DA64
North/South (m)
East/West (m)

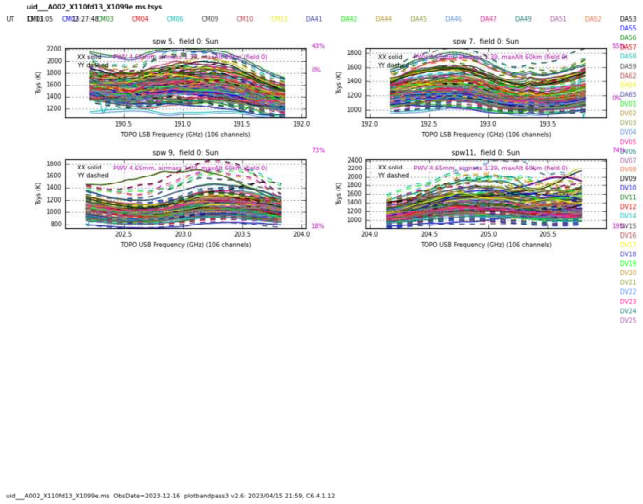

uid___A002_X110fd13_X1099e.ms
Elevation (degrees)
Time (from 2023/12/16) (hh:mm:ss)
13:15:00.0000 13:18:00.0000 13:21:00.0000 13:24:00.0000 13:27:00.0000 13:30:00.0000 13:33:00.0000 13:36:00.0000 13:39:00.0000

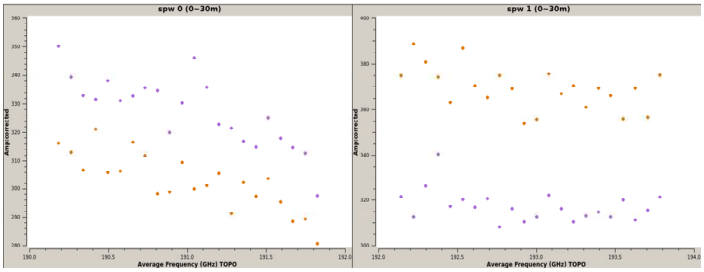

uid___A002_X110fd13_X1099e.ms
Elevation (degrees)
Azimuth (degrees)

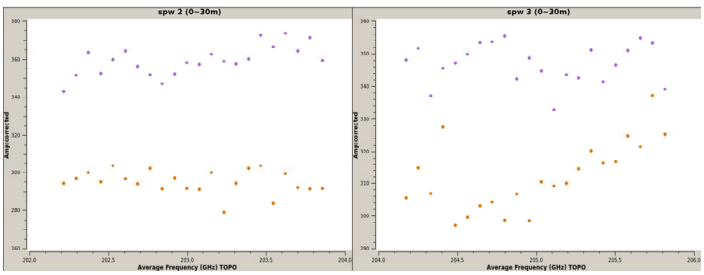

Ampl. CM01
Ampl. CM02
Ampl. CM03
Ampl. CM04
Ampl. CM06
Ampl. CM09
Ampl. CM10
Ampl. CM11
Ampl. DA41
Ampl. DA42
Ampl. DA44
Ampl. DA45
Ampl. DA46
Ampl. DA47
Ampl. DA49
Ampl. DA51
Ampl. DA52
Ampl. DA53
Ampl. DA55
Ampl. DA56
Ampl. DA57
Ampl. DA58
Ampl. DA59
Ampl. DA62
Ampl. DA64
Ampl. DA65
Ampl. DV01
Ampl. DV02
Ampl. DV03
Ampl. DV04
Ampl. DV05
Ampl. DV06
Ampl. DV07
Ampl. DV08
Ampl. DV09
Ampl. DV10
Ampl. DV11
Ampl. DV12
Ampl. DV14
Ampl. DV15
Ampl. DV16
Ampl. DV17
Ampl. DV18
Ampl. DV19
Ampl. DV20
Ampl. DV21
Ampl. DV22
Ampl. DV23
Ampl. DV24
Ampl. DV25
Amplitude
0.2
0.0
0
450
Time (sec)

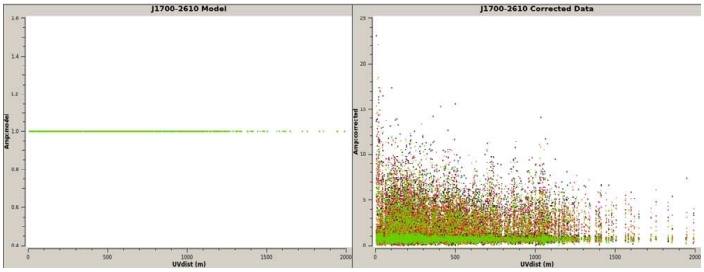

Phase CM01
Phase CM02
Phase CM03
Phase CM04
Phase CM06
Phase CM09
Phase CM10
Phase CM11
Phase DA41
Phase DA42
Phase DA44
Phase DA45
Phase DA46
Phase DA47
Phase DA49
Phase DA51
Phase DA52
Phase DA53
Phase DA55
Phase DA56
Phase DA57
Phase DA58
Phase DA59
Phase DA62
Phase DA64
Phase DA65
Phase DV01
Phase DV02
Phase DV03
Phase DV04
Phase DV05
Phase DV06
Phase DV07
Phase DV08
Phase DV09
Phase DV10
Phase DV11
Phase DV12
Phase DV14
Phase DV15
Phase DV16
Phase DV17
Phase DV18
Phase DV19
Phase DV20
Phase DV21
Phase DV22
Phase DV23
Phase DV24
Phase DV25
Phase (deg)
180
90
0
−90
−180
0
450
Time (sec)

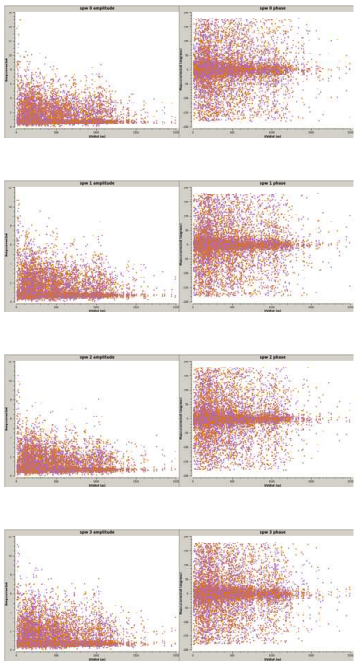

Avg Bandpass spw00 (191.0GHz)
Avg Bandpass spw01 (193.0GHz)
Avg Bandpass spw02 (203.0GHz)
Avg Bandpass spw03 (205.0GHz)
Channel

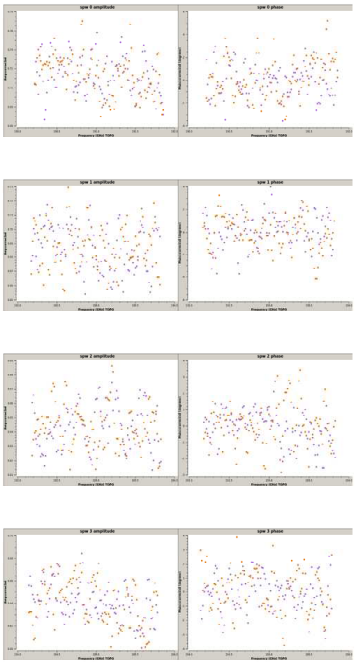

spw 5, field 0: Sun
spw 7, field 0: Sun
spw 9, field 0: Sun
spw11, field 0: Sun
XX solid
YY dashed
Tsys (K)
TOPO LSB Frequency (GHz) (106 channels)
TOPO USB Frequency (GHz) (106 channels)

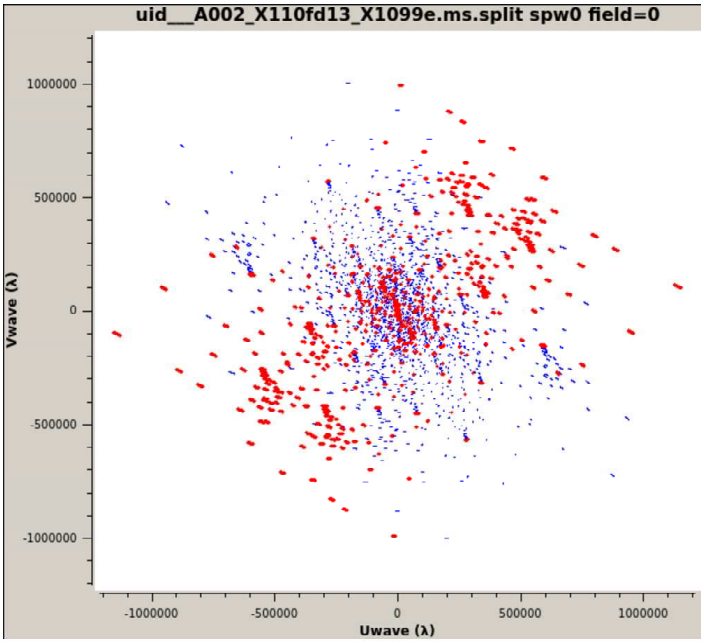

spw 0 (0~30m)
spw 1 (0~30m)
Amp corrected
Average Frequency (GHz) TOPO

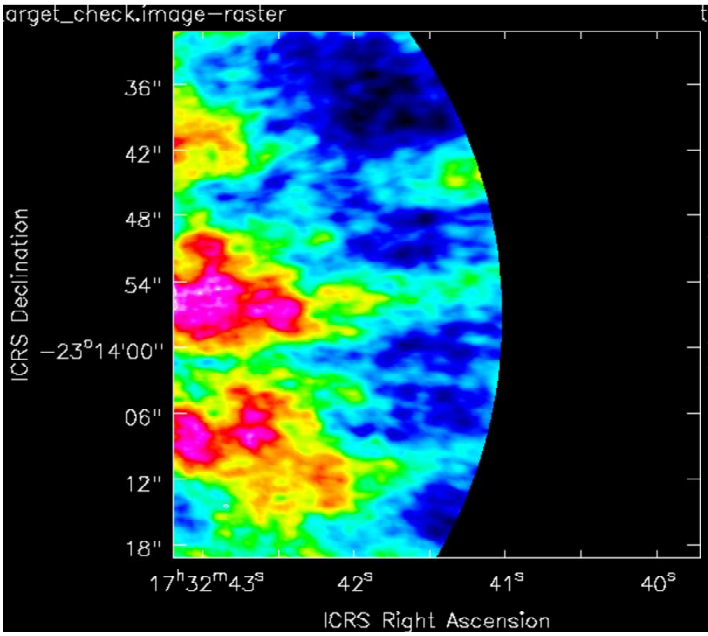

spw 2 (0~30m)
spw 3 (0~30m)
Amp corrected
Average Frequency (GHz) TOPO

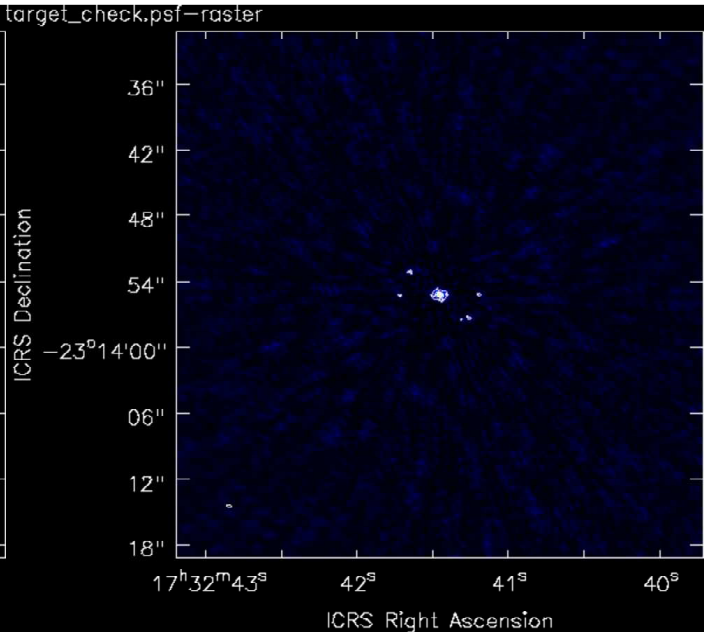

J1700-2610 Model
J1700-2610 Corrected Data
Amplitude
Amp corrected data
UVdist (m)
UVdist (m)






uid___A002_X110fd13_X1099e.ms.split spw0 field=0
Vwave (λ)
Uwave (λ)
1000000
500000
0
-500000
-1000000


target_check.image-raster
ICRS Declination
ICRS Right Ascension
36"
42"
48"
54"
−23°14'00"
06"
12"
18"
17h32m43s
42s
41s
40s


target_check.psf-raster
ICRS Declination
ICRS Right Ascension
36"
42"
48"
54"
−23°14'00"
06"
12"
18"
17h32m43s
42s
41s
40s

Annex 2

Recommendation for the 2nd half of the Study

Document Number	N/A
Document Title	Mid-term report
Document Date	2024.02.15
Document Revision	1.0

Review Panel	Affiliation	Acronym
Carlos De Breuck	ESO	CDB
Eric Villard (chair)	ESO	EVI
Ruediger Kneissl	ESO	RKN
Stephen White	USAF	SWH
Pavel Yagoubov	ESO	PYA
Observers	**Affiliation**	**Acronym**
Maria Diaz Trigo	ESO	MDT
Liz Humphreys	ESO	LHU
Tomek Kaminski	Lab. for Astrophysics in Torun	TKA
Hugo Messias	ESO	HME
Neil Phillips	ESO	NPH
Martin Zwaan	ESO	MZW
Responders	**Affiliation**	**Acronym**

Reviewer Comment Classes	Key
Typo	T
Review Item Discrepancy	RID
Review Item Clarification	RIC
General Comment	G
Other	O

No.	Page	Section	Reviewer Comment	Class	Originator	Severity	Response Study team	Feedback Reviewer	Description/Action	Issue closed?
23	15	4.2.4	The description of the mean of the fidelity image should be made more quantitative, statements as 'less spiky' are not very solid, and the difference in the mean values of 0.075 compared to 0.14 are not even mentioned. This is important to end up with a quantifiable go/no-go criterion	RIC	CDB	Major			AI Miro to define additional metrics to assess the quality of the reconstruction of the image of the Sun	No
34	30	7	A useful extension of the go/no-go criterion would be to extend the frequency bands to B1, B2 and B8-B10.	G	CDB	Major			AI Miro to determine go/nogo limits for B1/B2 (best effort)	No
35	11,18	4.2	With a simulated observation it should be possible to compare the phase solutions from calibration (phase referencing and self-cal) with the known phase corruption that was applied. The phase vs. time in Fig. 13 (only long time variations) doesn't look at all like the phase corruption in Fig. 4 (close to white noise). I suggest doing a comparison of the phase gain table obtained by self-cal with that that was applied to corrupt the data, and using the similarity (e.g.	RID	NP	Major			AI Miro to investigate the possibility of comparing the selfcalibration with the initial corruption (best effort)	No
37	29	7, 4th bullet	Related to my RIX 35, the testing of the WV extraction from the Band 5 data seems dependent on the self-cal solution reliability. Is there a feeling for how reliable this will be, and is it a risk to the WV testing? Could there be any independent means of testing, e.g. comparing retrieved WV with that from a WVR in a nearby antenna not pointing toward the Sun?	G	NP	Minor			AI Miro to work on this during second part of study	No
56	12	4.1.4	But an important factor for this question is whether multi-resolution clean was used: I can't find that stated anywhere. It should help to keep down the number of CC's needed.	RIC	SWH	Major			AI Miro to investigate the possibiity of adding one more/larger size value to the multiclean command	No
61			Since I am no expert on solar observing could you please clarify what data products have scientifc value in terms of small angular scale, such as low or high resolution images at different times, large scale range images at one instance, power spectra in corresponding scale ranges, etc.	RIC	RKN	Major			AI Miro to investigate the possibility of deriving a power spectrum from the visibilities	No
63	3	1	Maybe related to the uv dilution problem, if there was any value in only high spatial resolution information without a large range of scales on the solar structure then a single 12m configuration could be considered for the imaging, while the heterogenous array might be more difficult to use for imaging.	RIC	RKN	Major			AI Miro to simulate an observation+selfcal+imaging with just 12m configuration	No

65	4	1	Fig. 1: I understand that at a reasonable phase rms, say 30deg or better for good images, one would limit Band 6 to 2km and Band 3 to 2.5km from the plot, but should there not be a reference scale with the phase rms of the x-axis, and could the plot not directly adopt a max. acceptable phase rms.	RIC	RKN	Minor			AI Miro to think of ways to improve plot (e.g. specify configuration, swap axes)	No
72			The task of investigating small mosaic seems less advanced than other tasks already close to completion, and maybe there are challenges still to be addressed.	RID	RKN	Minor			AI Miro to work on this during second part of study	No
73	21	4.4	The use of Band 5 absorption for the correlated PWV is as stated in the MTR not demonstrated yet on the observations. I understand it is on-going and it would be good to see this soon in an application to the data.	RID	RKN	Major			AI Miro to work on this during second part of study	No

Annex 3

Documentation for Solar ALMA Utils developed in frame of the study

1. Solar Script Generator (SSG)
2. sol_wvrcal.py: collection (routines for Approach 2)
3. off_ephemeris.py: Setup for sun-array test observation

Solar ALMA utils

Generated by Doxygen 1.17.0

1 Namespace Index 1

1.1 Namespace List . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 Namespace Documentation 5

3.1 off_ephem Namespace Reference . . . 5

3.1.1 Detailed Description . . . 6

3.1.2 Function Documentation . . . 6

3.1.2.1 edit_coordinates() . . . 6

3.1.2.2 edit_coordinates2() . . . 6

3.1.3 Variable Documentation . . . 7

3.1.3.1 modified_line . . . 7

3.2 sol_wvgcal Namespace Reference . . . 7

3.2.1 Detailed Description . . . 8

3.2.2 Function Documentation . . . 9

3.2.2.1 acd_caltab_plots() . . . 9

3.2.2.2 acd_calTabs() . . . 10

3.2.2.3 acd_calTabs2() . . . 11

3.2.2.4 blindPhiFix() . . . 11

3.2.2.5 dB_sum() . . . 11

3.2.2.6 dPWV() . . . 12

3.2.2.7 dPWV_spwPart() . . . 12

3.2.2.8 dpwvXphi_corrs() . . . 13

3.2.2.9 dpwvXphi_plots() . . . 13

3.2.2.10 dpwvXsolWVR_raw() . . . 13

3.2.2.11 dpwvXwvr_corrs() . . . 14

3.2.2.12 dWdB() . . . 14

3.2.2.13 dWdB_Tbs() . . . 15

3.2.2.14 dWdB_Tbs_NL() . . . 16

3.2.2.15 dWdBdG() . . . 16

3.2.2.16 dWdBdG2() . . . 17

3.2.2.17 dWdBdG2_weights() . . . 18

3.2.2.18 dWdBdG_calWeights() . . . 19

3.2.2.19 dWdLs() . . . 20

3.2.2.20 dWdLs2() . . . 21

3.2.2.21 dWdLs3() . . . 21

3.2.2.22 dWdLs3_calWeight() . . . 22

3.2.2.23 dWdLs4() . . . 22

3.2.2.24 dWdLs4_weights() . . . 23

3.2.2.25 extMatInv() . . . 23

3.2.2.26 G_off_cal() . . . 24

3.2.2.27 getMeteo() . . . 24
3.2.2.28 getMeteoForScan() . . . 24
3.2.2.29 getPhase() . . . 25
3.2.2.30 getPhaseMS() . . . 25
3.2.2.31 getPWVforAnt() . . . 25
3.2.2.32 paregression() . . . 26
3.2.2.33 phiFixCompare() . . . 26
3.2.2.34 phiFixCompare3() . . . 27
3.2.2.35 plotPhases() . . . 28
3.2.2.36 pwv4scans() . . . 28
3.2.2.37 pwv4ScansCompare() . . . 28
3.2.2.38 pwv4scansRestore() . . . 29
3.2.2.39 pwvCalc() . . . 29
3.2.2.40 pwvCalcVerify() . . . 29
3.2.2.41 pwvCompare() . . . 30
3.2.2.42 rebin() . . . 30
3.2.2.43 rebin2() . . . 30
3.2.2.44 restoreSol_dWdB() . . . 31
3.2.2.45 saveSol_dWdB() . . . 31
3.2.2.46 saveSol_dWdB_Tbs() . . . 31
3.2.2.47 setAtmosSciSPWs() . . . 32
3.2.2.48 setAtmTool() . . . 32
3.2.2.49 setAtmTool4SPWs() . . . 33
3.2.2.50 smooth() . . . 33
3.2.2.51 sol_offPos() . . . 33
3.2.2.52 sol_PhaseFix() . . . 34
3.2.2.53 sol_pwv2selfCal() . . . 34
3.2.2.54 sol_pwvGcal() . . . 34
3.2.2.55 solTgtPos() . . . 35
3.2.2.56 sqld_P0() . . . 35
3.2.2.57 TbStats() . . . 35
3.2.2.58 tgt_pow() . . . 36
3.2.2.59 tgt_Tb() . . . 36
3.2.2.60 tgtPow_crossPlots() . . . 37
3.2.2.61 unwrap2() . . . 37
3.2.2.62 wopacity() . . . 37
3.2.2.63 wrapBack() . . . 38
3.2.2.64 wv2path() . . . 38
3.2.2.65 wvr_dataRaw() . . . 38
3.2.2.66 wvrPhase() . . . 39

4 Class Documentation **41**

4.1 SolSG.calibrators Class Reference . . . 41
4.1.1 Detailed Description . . . 41
4.1.2 Constructor & Destructor Documentation . . . 42
4.1.2.1 __init__() . . . 42
4.1.3 Member Function Documentation . . . 42
4.1.3.1 allCalsLst() . . . 42
4.1.3.2 allCalsStr() . . . 42
4.1.3.3 fluxCalQualityMetrics() . . . 42
4.1.3.4 printInfo() . . . 43
4.2 SolSG.observation Class Reference . . . 43
4.2.1 Detailed Description . . . 43
4.2.2 Member Function Documentation . . . 43
4.2.2.1 printInfo() . . . 43
4.2.2.2 rfStr() . . . 44
4.2.3 Member Data Documentation . . . 44
4.2.3.1 obsDate . . . 44
4.3 SolSG.scienceTarget Class Reference . . . 44
4.3.1 Detailed Description . . . 45
4.3.2 Constructor & Destructor Documentation . . . 45
4.3.2.1 __init__() . . . 45
4.3.3 Member Function Documentation . . . 45
4.3.3.1 isMosaic() . . . 45
4.3.3.2 printInfo() . . . 46
4.3.3.3 targetFields() . . . 46
4.4 SolSG.stuffForSolarDataReduction Class Reference . . . 46
4.4.1 Detailed Description . . . 46
4.4.2 Member Function Documentation . . . 46
4.4.2.1 generateSolCalScript() . . . 46

Index **47**

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

off_ephem 5
sol_wvgcal 7

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

SolSG.calibrators 41
SolSG.observation 43
SolSG.scienceTarget 44
SolSG.stuffForSolarDataReduction 46

Chapter 3

Namespace Documentation

3.1 off_ephem Namespace Reference

Functions

- edit_coordinates (time_str, ra_str, dec_str, azShift=2.0)
- edit_coordinates2 (time_str, ra_str, dec_str, azShift=2.0)

Variables

- **TIME_START**
- **TIME_END**
- **RA_START**
- **RA_END**
- **DEC_START**
- **DEC_END**
- int **DEG_TO_RAD** = math.pi/180.
- int **RAD_TO_DEG** = 180/math.pi
- int **HRS_TO_RAD** = math.pi/12.
- float **RAD_TO_HRS** = 1.0/HRS_TO_RAD
- **observatory**
- int **phi** = alma_lat∗DEG_TO_RAD
- str **input_file** = "./SC.txt"
- str **output_file** = "./off_pos.txt"
- bool **inside_ephemeris** = False
- **stripped_line** = line.rstrip("\n")
- **time_val** = stripped_line[TIME_START:TIME_END]
- **ra_val** = stripped_line[RA_START:RA_END]
- **dec_val** = stripped_line[DEC_START:DEC_END]
- **new_ra_val**
- **new_dec_val**
- tuple modified_line

3.1.1 Detailed Description

This code reads the ephemeris of the solar-disc center from the file SC.txt and
for each its line it ads a predefined offset (default is +2deg) in *Azimuth* !
Shifted positions (transformed back to RA/Dec) are then written to the modified
ephemeris file off_pos.txt. The original solar-center ephemeris is obtained
by Solar Ephemeris Generator.

How does it work? For each line of the original ephemeris we
1. Convert RA/Dec to Az/El.
2. Add the desired offset (+2degs is the default) to Az (scaled by 1/cos(EL))
3. Convert back to Ra/Dec.

Rationale: For the sake of test of compatibility between our PWV proxy and WVR
data, we observe the Sun with two sub-arrays, Array 1 (~37 antennas) observes
defined solar target, Array 2 (2 -3 antennas) measures co-temporally WVRs in
the postion off-the-Sun. We, however, need that the offset from the Sun is
constant in Azimuth in order to preserve the same airmass for both arrays.

Two alternative approaches have been tried for the AZ-shift calculations:
(i) edit_coordinates(), which uses the aU utils for transforming to AzEl frame
and back, and (ii), edit_coordinates2(), which uses the *measure* CASA tool for
the same purpose. The results are similar (differences within 30 arcsecs), we
use the latter one as it is more accurate.

3.1.2 Function Documentation

3.1.2.1 edit_coordinates()

```
edit_coordinates (
              time_str,
              ra_str,
              dec_str,
              azShift = 2.0)
```

Convert to AzEl and shift by *azShift* degs (positive means westward at
north hemisphere). Then convert back to RaDec.
We use methods from Analysis Utils for doing so.

The shift means the actual angular separation from the solar center in the
direction of AZ, so it needs to be scale by 1/cos(EL)!

3.1.2.2 edit_coordinates2()

```
edit_coordinates2 (
              time_str,
              ra_str,
              dec_str,
              azShift = 2.0)
```

A more accurate alternative to edit_coordinates(). Now we use
methods of *measure* tool from the CASA toolkit.

Convert to AzEl and shift by *azShift* degs (positive means westward at
north hemisphere). Then convert back to RaDec.

The shift means the actual angular separation from the solar center in the
direction of AZ! The measure.shift() method takes care about it.

3.1.3 Variable Documentation

3.1.3.1 modified_line

```
tuple off_ephem.modified_line
```

Initial value:

```
=  (stripped_line[:RA_START]
                        + new_ra_val
                        + stripped_line[RA_END:DEC_START]
                        + new_dec_val
                        + stripped_line[DEC_END:])
```

3.2 sol_wvgcal Namespace Reference

Functions

- rebin (array1D, downSampleFactor)
- rebin2 (array1D, origX, downSampledX)
- smooth (array1D, kernel)
- paregression (arrX, arrY)
- unwrap2 (phase)
- wrapBack (phases)
- extMatInv (Ainv, Lb, Rb, LRc)
- sqld_P0 (vis, ant, spw, myMS=None, myMsmd=None)
- acd_calTabs (vis, atmScan, ant, spw, tgtScan=None, myMS=None, myMsmd=None, acdLoads=None)
- G_off_cal (vis, atmScan, tgtScan, ant, acSpw, myMS=None, myMsmd=None)
- acd_calTabs2 (vis, atmScan, ant, spw, myAtm=None, myMS=None, myMsmd=None, acdLoads=None)
- acd_caltab_plots (vis, atmScans, sciSPWs)
- tgt_pow (vis, tgtScan, ant, spw, myMS=None, myMsmd=None, subScans='on')
- tgtPow_crossPlots (vis, ant, tgtScan, spw1, spw2)
- getPhase (gTable, antID, spw)
- getPhaseMS (vis, antID, spw)
- plotPhases (vis, gTable, spw)
- tgt_Tb (vis, atmScan, tgtScan, antID, spwID, myMS=None, myMsmd=None, acdLoads=None, myAtm=None)
- setAtmTool (myAtm, vis, spw, scan, myMsmd=None, wvHeightScale=1.5, tempDropPerKm=-5.6)
- setAtmTool4SPWs (myAtm, vis, scan, myMsmd=None, wvHeightScale=1.5, tempDropPerKm=-5.6)
- wopacity (vis, spw, scan, userPWV=-1.0)
- setAtmosSciSPWs (vis, tgtScan, spwList)
- getPWVforAnt (vis, tgtScan, antID, myMsmd=None)
- getMeteo (vis)
- getMeteoForScan (vis, scan, myMsmd=None)
- wv2path (vis, spw, scan, pwv)
- dPWV (vis, atmScan, tgtScan, antID, spwAvg=6, spwRes=5)
- dPWV_spwPart (vis, atmScan, tgtScan, antID, spwRes=5, startChan=8, stopChan=120)
- dWdB (vis, atmScan, tgtScan, atmModel, antID, spwList)
- dWdB_Tbs (vis, atmScan, tgtScan, myAtm, antID, spwList, flaggedData=[ ], myMS=None, myMsmd=None, acdLoads=None)
- dWdBdG (vis, atmScan, tgtScan, myAtm, antID, spwList, flaggedData=[ ], myMS=None, myMsmd=None, acdLoads=None)
- dWdBdG2 (vis, atmScan, tgtScan, myAtm, antID, spwList, T_tgt=-1.0e6, flaggedData=[ ], myMS=None, my↩ Msmd=None, acdLoads=None)
- dWdB_Tbs_NL (vis, atmScan, tgtScan, myAtm, antID, spwList)

- dWdBdG2_weights (vis, atmScan, tgtScan, myAtm, antID, spwList, T_tgt=-1.0e6, flaggedData=[], myMS=None, myMsmd=None, acdLoads=None)
- dWdBdG_calWeights (vis, atmScan, tgtScan, myAtm, antID, spwList, flaggedData=[], myMS=None, myMsmd=None, acdLoads=None)
- saveSol_dWdB (vis, atmScan, tgtScan, spwList)
- saveSol_dWdB_Tbs (vis, atmScan, tgtScan, spwList)
- restoreSol_dWdB (vis, filePath, runID=100000)
- dB_sum ()
- TbStats (resCG, resVG)
- dpwvXphi_plots (doSun=False, useSQLD=False)
- dpwvXphi_corrs (doSun=False, useSQLD=False)
- blindPhiFix (doSun=False, useSQLD=False)
- wvrPhase (vis, antID, spw)
- pwvCompare ()
- phiFixCompare (vis, atmScan, tgtScan, refAnt, spwID, spwRes=-1)
- phiFixCompare3 (vis, atmScan, tgtScan, refAnt, spwLow, spwRes=-1)
- dpwvXwvr_corrs (SQLD=False)
- sol_PhaseFix (antID, spw, pwvMap)
- sol_pwv2selfCal (antID, spw)
- sol_pwvGcal (pwvMap, vis, refAnt, pwvCalTableName='sol_pwvCalTable.W')
- sol_offPos (vis, tgtScan, refAnt)
- solTgtPos (vis, tgtScan, refAnt)
- wvr_dataRaw (vis, tgtScan, antID, subScans='')
- pwvCalc (vis, atmTool, scan, antID)
- pwvCalcVerify ()
- dWdLs (antID, myAtm, fullPWV=False)
- dWdLs2 (myAtm, antID, tgtScan, flaggedWVRchannels=[], ssType='', solIllumination=True, myMsmd=None, acdLoads=None)
- dWdLs3 (vis, myAtm, antID, tgtScan, flaggedWVRchannels=[], ssType='', solIllumination=True, myMsmd=None, acdLoads=None)
- dWdLs4 (vis, myAtm, antID, tgtScan, flaggedWVRchannels=[], ssType='', solIllumination=True, myMsmd=None, acdLoads=None)
- dWdLs4_weights (vis, myAtm, antID, tgtScan, flaggedWVRchannels=[], ssType='', solIllumination=False, myMsmd=None, acdLoads=None)
- dWdLs3_calWeight (myAtm, antID, tgtScan, flaggedWVRchannels=[], ssType='', solIllumination=True, myMS=None, myMsmd=None, acdLoads=None)
- pwv4scans (vis, solWVR=False)
- pwv4scansRestore (vis, hasSolWVR=True, runID=100000)
- dpwvXsolWVR_raw ()
- pwv4ScansCompare (vis)

3.2.1 Detailed Description

```
@file sol_wvgcal.py
@brief Use WV line absorption/emission in AC/sqld SPWs for phase correction.

@author barta@asu.cas.cz
@date 2026-04-03 (last update)

Experimentation with WV-line features - emission in case of the weak source
(typically calibrator) and absorption for the Sun - in the science SPWs: either
auto-correlations (ACs) of SQLD. We aim to use those features for eventual
phase-error corrections on the short time-scales, as an alternative to WVR data
and wvrgcal(), which are unavailable (saturated) in case of solar on-source
scans.

The collection of routines here is devised for the following group of tasks:
```

```
1. Multi-purpose (non-specific) helpers: rebin(), smooth(), paregression().
2. Perform dual-load/ACD calibration: acd_calTables(); apply it to MS powers
   to convert them to calibrated T_B: tgt_Tb(). Diagnostics: acd_caltab_plots().
3. Extract the raw data from MSs and calTables: tgt_pow() and getPhase().
   Diagnostics: plotPhases().
4. Setup the atmosphere (ATM) model and use it for calculation of opacities:
   setAtmTool(), wopacity().
5. Use all above + mathematical mode derived in PR11 to calculate the variations
   of PWV (proxy/model): dPWV()
6. Analysis:  Compare the dPWV proxy vs. phase variations
7. Verification: Compare our dPWV proxy for the BP-calibraror with an actual
   PWV/WVR data measured in ASDM_CALWVR (accessed via wvrgcal())

7.1 Specific for 2022.1.00031.CSV-Sun10a-05-INT SB: Analysis of unsaturated
    raw WVR data

-----
Notes
=====

Ad (2): Note on comparison with sol_sqld_mean_2() in sun_reduction_utils.py.
We extract the full AC and SQLD data from the MS (using ms.select()) and make
their averaging here using numpy.mean(). The sun_reduction_utils.py uses
ms.statistics() internally in its sol_sqld_mean_2() helper. For SQLDs, the
results of np.mean() are very similar to those obtained by sol_sqld_mean_2().
The difference between our approach and that in sol_sqld_mean_2(), which uses
ms.statistics() internally, is less than 0.015% (the relative error) and looks
to be related by rounding inaccuracy (less accurate representation of the
float data in our approach). Our approach is, however, more general - in the
already mentioned sense of its applicability to both SQLD and AC (channelized
or integrated) spws, but also because it can be used for the concrete selected
antenna: sol_sqld_mean_2() accepts the antenna-names list, too, but it
MUST be a complete list of *all* antennas for the given MS, otherwise mapping
between antenna name and its ID (int number), which is internally used in the
sol_sqld_mean_2() method, gets broken.

Ad (3): Note to experimenting with Dual-Load (DL: Hot, Ambient, Sky) calibration
using ALMA ACD. We extract P_hot, P_amb, and P_sky for a given antenna and ATM
scan. For the Sun, we extract also P_off, which is basically P_sky, but with the
attenuation set to the high, solar-level, value. This is not present in the
ATM scan preceding the solar-target scan, but in the Sun-scan itself: In its
very first and very last sub-scan (see Fig. 4 in Shimojo et al., 2017).
We extract also ACD load temperatures and  - for SQLD spws - also the P_zero
offset for the scan and calculate the gain G and T_rx according to formulae
(7) and (9), and "solar" (i.e. in hi-attenuation setup) gain G* according to
Eq. (8) in White et al. (2017).

The main difference to solar flux calibration implemented in sol_ampcal2() is
that we can use here also
1. autocorrelation (AC) data and not just the SQLDs,
2. we may extract ACs in the ATM scans in SPWs with the channelized data
    in order to reconstruct the spectrally resolved  Tsky+Tsrc (128 channels).

The obtained ACD calibration (gains G and G*, T_rx and in case of SQLD
data also the offset P_zero) is subsequently applied to the target scan in
order to convert measured raw powers (AC or SQLD) into brigthness 'sky'
temperature - the sum of source (attenuated by the terrrestrial atmosphere)
and the contribution of our atmosphere.
```

3.2.2 Function Documentation

3.2.2.1 acd_caltab_plots()

```
acd_caltab_plots (
              vis,
              atmScans,
              sciSPWs)
```

A diagnostic routine that plots G/G* and T_rx as found in acd_calTabs() as
a function of frequency in the channel-resolved SPWs for each antenna.

3.2.2.2 acd_calTabs()

```
acd_calTabs (
              vis,
              atmScan,
              ant,
              spw,
              tgtScan = None,
              myMS = None,
              myMsmd = None,
              acdLoads = None)
```

Calculates the (inverse) gains (G and, possibly, G* in White et al. 2017),
and T_rx for a given *ant* and *spw* from the ATM scan *atmScan* using its
three sub-scans for Sky, Ambient, and Hot loads.

The *ant*, *spw*, and *atmScan* parameters MUST be set as *int* (not string),
otherwise ms.select(), used internally, returns completely wrong selection -
silently, without any warning!
The 'global' tools *myMS* and *myMsmd* may be conveniently supplied, attached
to the MS *vis*. If not, local tools are created and destryed with the
routine, with some additional cost on CPU time. The same holds for the
global dictionary *acdLoads* that is obtained by call to aU.getAntennaLoads().

If the *atmScan* precedes directly the solar scan, we calculate G*, too.
For the G*, we use (in addition to the three ATM subscans) also the
'sky'/off-source sub-scans (the very first and last) in the
immediatelly-following solar scan.

Returns: A dictionary with the inverse gain G (Eq. 7 in White et al.), inverse
'solar' gain G* (Eq. 8), T_rx (Eq. 9), and effective total 'instrumental'
temperature T_inst=T_rx+Po/G. The G and T_rx components are
numpy arrays of shape nPols x nChans, i.e., 2 x 128, if we use the science
SPWs (5,7,9,11) and 2 x 1 for channel-integrated SPWs (6,8,10,12) or
SQLDs (0,1,2,3). The first dimension represents polarization and the second
the channel number.

For G*, we provide its two values - at the very beginning and very end of the
solar scan (the very 1st and very last sub-scans in each solar scan are the
off-source subscans). Hence, the shape of the returned array is
2 x nPol x nChans, the first dim =2 representing the beginning and end values.
According to Shimojo et al. (2017), the G* should be calculated from the SQLD
spws solely (it is said that P_off is too low for AC spws). Consequently, it
should be an array of the shape 2 x 2 x 1. However, we are experimenting here
with the auto-corellations (AC) SPWs, too, both for spectral-resolved and
channel-integrated spws, so the return shape may be well also 2 x 2 x 128.
The reason is roooted in the fact that the attenuation factor G*/G found in
the Band 5 data is just about 1.07.In case of the ATM scan precedes non-solar
scan (e.g., BP-cal scan), we return [[[1.0],[1.0]],[[1.0],[1.0]]] - i.e.
factor of 1,with the shape corresponding to the channel-integrated SPWs.
Nevertheless: It has been found - see detailed comment in tgt_Tb(), in the
section starting with 'if(doSolarGain)' - that PWV changing during the solar
scan affects measured P_off and consequently the attenuation factor and
resulting gain G*. Sinze the attenuation factor AF=(P_sky-Po)/(P_off-Po),
it is bettre to take just the leading-subscan P_off measurement, because
it follows immediatelly after the P_sky measutement in preceding ATM scan.

3.2.2.3 acd_calTabs2()

```
acd_calTabs2 (
              vis,
              atmScan,
              ant,
              spw,
              myAtm = None,
              myMS = None,
              myMsmd = None,
              acdLoads = None)
```

Improved version of acd_calTabs() for solar AC data. It uses transfer of relative gain-drop between P_sky and P_off scans from SQLD to AC spws - see experimentation in G_off_cal().

Because of corrections to frequency-dependent transmissivity and intrinsic radiation of the terrestrial atmosphere in channel-resolved AC spws, we need to suply the ATM model *myAtm* for the solar G* calibration. It is the *attool* set for our atmosphere, for all 4 resolved sci SPWs, and it can be conveniently obtained via setAtmTool4SPWs() - see also dWdB_Tbs().

If we do not calibrate for the Sun, i.e. when *atmScan* is not followed by a solar scan, *myAtm* parameter may remain empty. Similarly, if the *spw* is SQLD, we do nothing more on top of acd_calTabs(), so *myAtm* is ignored.

3.2.2.4 blindPhiFix()

```
blindPhiFix (
              doSun = False,
              useSQLD = False)
```

*** So far just for the solar MS project 2022.1.00031.CSV, Band 5!!! ***

Basically, we do the same calculations like in dpwvXphi_plots() but now we try to make blind corrections of the phases by dPWV with two options:

1. We fix phase by subtracting dPWV scaled by the theoretical factor -25.0.
2. Like in case (1.), but use the factor found by the correlation analysis.

Phases fixed by both methods are plotted into a single chart for comparison. We would like to see, what is residal phase variation after the correction.

@todo: For the Sun we do NOT have yet 'indpendent' phase solution to compare, we do it for the phases of the BP cal. Of course, it has no sense to compare the dPWV curve (scaled by -25.0) for the Sun with the phases for the BP-cal. The only goal is to see, whether we have similar amplitude and typical dynamics comparable or not.

3.2.2.5 dB_sum()

```
dB_sum ()
```

Sum the brightness variations as found by dWdB() for all antennas.

Idea: If there is an intrinsic T_B variation of the Sun, we arrive to this 'coherent' contribution in the sum. If not, and the dB changes are just because of atmosperic lensing, the average of dB over all antennas should tend to zero.

3.2.2.6 dPWV()

```
dPWV (
              vis,
              atmScan,
              tgtScan,
              antID,
              spwAvg = 6,
              spwRes = 5)
```

Calculate dPWV - our proxy to differential PWV (water-vapour column) based on the variations of received power. See formulae (13) and (14) in the Progress Report no. 11 (March 27, 2025). We may use both the auto-correlation (AC) channel-integrated SPW (i.e. SPW 6 in solar case) or, almost equally, the lowest SQLD spw (SPW 0) for that purpose. What spw is actually used is controlled by *spwAvg* parameter.
The *tgtScan* parameter defines, for what object is dPWV calculated, the *atmScan* argument is the scan used for ACD calibration (see acd_calTabs(); atmScan should precede the tgtScan as close as possible, ideally by one). Antenna, for which dPWV is calculated, is represented by its integer *antID*.

Returns: A tupple of three 1-D arrays, each of the size of nIntegs: First represents the time-axis as MJD (merely reproduction of the axis returned by tgt_pow()), the other two are values of dPWV for XX and YY polarizations, respectivelly.

3.2.2.7 dPWV_spwPart()

```
dPWV_spwPart (
              vis,
              atmScan,
              tgtScan,
              antID,
              spwRes = 5,
              startChan = 8,
              stopChan = 120)
```

Find dPWV - our proxy to the WV contents but now by using just a part of the SPW. Unlike in dPWV() we now use the channel-resolved SPW *spwRes*, by default that one closest to the WV line center (i.e. SPW 5).

Because the single-channel data are too noisy (that is the reason why spw-integrated ACs or SQLDs are being used), we do integration over multiple channels, but not entirely over full SPW. The frequency range for integration is defined by *startChan* and *stopChan* parameters, which default to the first and last 'usable' channel (the firts and last 8 channels are typically flagged in any analysis). Note: For SPWs with reversed order the start/stop chan is also swapped so that the startChan always means the low-frequency boundary.

Idea: We may avoid O3 lines or use just that part of the SPW 5 closer to the WV line center and increase sensitivity to the WV contents. Suggested on PM12 in May 2025 (see PR12 - post meeting Action item 6), implemented in line with formula (1) in PR13 (August 2025).

3.2.2.8 dpwvXphi_corrs()

```
dpwvXphi_corrs (
              doSun = False,
              useSQLD = False)
```

*** So far just for the solar MS project 2022.1.00031.CSV, Band 5!!! ***

Calculate actual correlations between dPWV and phases. Just indicative plots
are performed by dpwvXphi_plots() with anticipated coefficient of linear
regression to be -25.0 (see the study in PR11, March 2025). Here, we perform
actual correlation calculations, finding the coefficients of linear regression
between dPWV and phases. However, the situation is not that simple: the
preliminary analysis has shown (see PR11) that the phases tend to (perhaps
linearly) grow also with time and time-based de-trending of the phase series
should be performed. In order to do that, parametrized linear regression
is computed here using paregression().

We may use ACs or SQLD for that - see dPWV(); this is controlled by *useSQLD*.
Whether we do this for the BP-calibrator or the Sun is controlled by
doSun.

3.2.2.9 dpwvXphi_plots()

```
dpwvXphi_plots (
              doSun = False,
              useSQLD = False)
```

*** So far specific just for the solar MS 2022.1.00031.CSV, Band 5 !!! ***

Plot the phase variations opverplotted by our dPWV proxy, scaled by
the theoretical factor dPhi/dPWV=-25.0 for each antenna - see the PR11.
This is just the quick-check diagnostics.

We may use ACs or SQLD for that - see dPWV(); this is controlled by *useSQLD*.
Whether we do this for the BP-calibrator or the Sun is controlled by *doSun*.

3.2.2.10 dpwvXsolWVR_raw()

```
dpwvXsolWVR_raw ()
```

A simple 'by-eye' comparison our dPWV proxy for the *solar* scan with
the *raw* WVR data in the solar scan.

dPWV proxy is calculated from *science SPWs* using dWdB(). We utilize here the
pre-calculated output saved in dWdB_data.npy

WVR is calculated from raw WVR data (SPW 4) via dWdLs().

We plot both time profiles as lef and right panel of a single figure to
see the differences.

*** Specific for 2022.1.00031CSV project! ***

This method cannot be used outside of the project - it was just a good luck
that the WVR were ponting out of the Sun because the Band 5 receiver was
pointed close to the limb - see fixSolWVRs() for details.

3.2.2.11 dpwvXwvr_corrs()

```
dpwvXwvr_corrs (
              SQLD = False)
```

***So far specific just for the solar project 2022.1.00031.CSV, Band 5 !!! ***

Find correlation and linear trends between our dPWV()-proxy calculated phase correction and that found by the WVRgcal().

Very similar approach as used in dpwvXphi_corrs(), now used for correlation analysis between two phase corrections cacluated by two different methods.

This routine cannot be even theoretically used for the Sun (no wvrgcal table).

3.2.2.12 dWdB()

```
dWdB (
              vis,
              atmScan,
              tgtScan,
              atmModel,
              antID,
              spwList)
```

*** The main routine for calculation of the dW(t) for solar scans.***

For the BP calibrator we may use dPWV() or its dPWV_spwPart() variant as there is no further source of signal variations except of the atmopsheric column mass changes manifested as variations in PWV.

For the Sun, this is, however, not true - it can have its own brightness changes. Here we attempt to decompose the signal variations using a collection of science SPWs in *spwList* (may be AC integrated, AC resolved, SQLD or any combination, which incorporates at least two frequency domains) into:

(i) dW = total variations of the PWV - like in dPWV, and
(ii) dB = variations of the (apparent) source brightness.

The latter may involve both intrinsic changes of Tb_src and the lensing effects of the turbulent atmosphere (a.k.a. caustics in low-frequency radio) - these two are not easilly mutually distinguishable.

In order to do the disentagling, we need to use multiple SPWs.
In the single-channel SPWs we omit the correction implying from the slope of Tsky(f) and tau(f), it has been experimentally found to be less then 5% for dPWV().

The *atmModel* is a set of static 'tabelised' profiles (frequency-dependent) for opacities, Tatm, etc., as returned by wopacity(). We have it as an array for all science channel-resolved SPWs (5,7,9, and 11). It is supplied as an parameter because it is the same for all antennas: We do not want to re-creaate it inside this routine whenever it is called for another antenna, as it is usually done in a loop over entire array, e.g., in saveSol_dWdB(). When this routine is invoked in a loop for all array, the *atmModel* can be conveniently pre-set by call to setAtmosSciSPWs().

The overdetermined system (we have just two variables - dW and dB, and up to 12 equations for 12 SPWs) is then solved by the least-square principle, leading to a minimizer of a quadratic form (which is another linear system of eqs., but now regular one: 2x2 matrix with a nTimeSamples-long collection of 2-component RHS vectors).

Least-square residuals and other analytics are plotted in a compound 2 x 3
panels graph as a by-product of calculations. In the './dWdB/' subdir we plot
the solutions dW(t) and dB(t).

Commented-out is a perspective extension, which tries to decompose the
total dW variations into two parts, caused by (1) relative-humidity changes,
and (2) total air-preessure variations. It has been experimentally found,
however, that the system is then ill-conditioned for this decomposition. It
can be improved by changing the vector basis and decomposing into total wet
and total dry variations. However, even with this setup, the total wet
contribution to the signal remains almost unchanged (so it is quite a robust
quantity), the total dry component exhibits a large error and cannot be used.

@todo: In the ASDM_CALATMOPHERE table we have Tatm (a single value) per
SPW. Try to compare its value with our Tatm as calculated by wopacity().

3.2.2.13 dWdB_Tbs()

```
dWdB_Tbs (
              vis,
              atmScan,
              tgtScan,
              myAtm,
              antID,
              spwList,
              flaggedData = [],
              myMS = None,
              myMsmd = None,
              acdLoads = None)
```

*** Need to be edited if used for obsevartion different from 2023.12.16 ! ***
Namely the ehpemeris-dependend data at the beginning of the script (indicated
in comment).

An alternative approach to dWdB(), which, instead of using Tatm in the
expression Tbs=Tatm*(1-tr) for atmospheric contribution to the sky brightness,
uses directly Tbs=at.TebbSky(), smiliraly to our approach for PWV calculations
from the WVR temperatures, like in pwvCalc() or dWdLsun().

It should be of the same order of accuracy and complexity, because the Tatm
is calculated from the same ATM model anyway, just we have to scale it by
the factor (1-tr).

The big advantage of this alternative approach is that we can easilly find
also the scan-averaged PWV value, not just its differential variations.

We use the same approach as in dWdLsun(): Linearizing the radiative-transfer
equation around its "operation point", which is defined by the scan-averaged
PWV w0 and scan-averaged 'luminosity' of the Sun Tsun0. The linear set of
equations is overdetermined for more than two SPWs (if they occupy distinct
frequency domains), so we solve the set of 'normal' equations, ipmlying
from the least-square principle, like in dWdB(). However, in our linearized
set of equations, the variations need not to have zero mean. So we add the
mean to the stationary 'operation point' defined by [w0,Tsun0] and make a new
iteration, like in dWdLsun(), until we reach zero-mean variations. This way,
both the variations and their stationary part is found. This approach has been
verified via dWdB_Tbs_NL(), which solves the non-linear RT equation directly
by Newton-Raphson iterations: Results of both differ insignificantly.

We do appropriate integration in the frequency domain for all model-based
quantities. The solar QSR spectrum is being scaled by formula (4) in
Alissandrakis et al. (2020), the original power-law estimation has been
abandoned beacuse of large residua (see below in the code).

Moreover, we apply consequentlythe antenna efficiencies eta_l and eta_F
(see White et al. 2017): For the fluctuations within in a single SPW they
may be incorporated into effective gain and 'instrumental' temnperature,
but here, when using all SPWs and absolute calibration to solar T_B and full
PWV, it is not possible.

And the last innovation: The PWV values should be common for both XX and YY
polarization. On the pther hand, the solar brightness T_B may differ in
XX and YY channels (e.g., the apparent changes due to atm.-turbulence lensing
may be polarization dependent, even the solar FoV may have a small amount of
inrinsic polarization). Therefore, we have 3 independent variables:
PWV, TB_XX, and TB_YY.

Arguments (these that differ from dWdB()):

myAtm: Here, myAtm is the *attool* set for our atmosphere, for all 4
resolved sci SPWs. It is a difference to dWdB(), where we use already
pre-calculated ('tabelized') model, i.e., 'static' arrays for opacities, Tatm,
transmissivity etc. for a specific value of time-averaged PWV for the scan.
Here it is not possible as the mean PWV (the w0 parameter) is one of the
unknowns (in dWdB() we use its ASDM_CALWVR value interpolated to the center
of the scan). The *myAtm* attool set for the frequency domain defined by the
AC channnel-resolved SPWs can be conveniently obtained via setAtmTool4SPWs().

flaggedData: A list of strings in the form of (example) ['9:XX','3:YY'] with
the spw:polarization pairs that should be ignored in the analysis. This can be
used for multi-passage of this method, where in the first run we find the
final-solution residua and the SPW:Pol pair, whose residuum is more than
desired distance (e.g., 3 sigma) appart from the mean, can be flagged for
the repeated run.

myMsmd, *myMS*: Conveniently we can pass the tools already attached ton the
MS *vis*. If nothing is provided, the method allocates and attaches its own
local tools (whoch costs some CPU time)

acdLoads: A dictionary of ACD temperatures (got by aU.getLoadTemperatures())
from which, for example, the ambient-load temperature can be extracted by
Tamb = acdLoads[antID][atmScan]['amb']. If not provided, it is generated
inside the routine (cumbersome, if we call it in the loop for all antennas).

3.2.2.14 dWdB_Tbs_NL()

```
dWdB_Tbs_NL (
              vis,
              atmScan,
              tgtScan,
              myAtm,
              antID,
              spwList)
```

Like dWdB_Tbs(), but now we do not use linearization. We apply brutal force to
solve the full, overdetermined system of non-linear equations in least-square
sense. Intended just for verification of the linearization method.

Results: It has been found that the difference in PWV is max. 0.01 mm and in
brightness temperature ~3 K - see PR16. As a result, we further develop just
the linearized version dWdB_Tbs().

3.2.2.15 dWdBdG()

```
dWdBdG (
              vis,
```

```
                atmScan,
                tgtScan,
                myAtm,
                antID,
                spwList,
                flaggedData = [],
                myMS = None,
                myMsmd = None,
                acdLoads = None)
```

Extended dWdB_Tbs(): Solve for antenna gains and instrumental powers, too.

The most complex treating the RT problem for the Sun: Like with dWdB_Tbs() we get the time-series of full PWV and the (apparent, i.e. including the atmospheric lensing) solar brightness temperature - actually its reference value for f_LO=198.0GHz - in XX and YY polarizations. But, here, we moreover set the antenna gains *G* and the instrumental power *P_inst=G*Trx+Po* as unknown parameters to be solved by the extended set of equations.

Rationale: dWdB_Tbs() seems to perfectly reproduce the variations in the solar brightness temeprature and PWV: They are very similar in all SPWs in both XX and YY polarizations. Yet, there is a stationary part of the T_B, which is different among SPWs and pols - see the residuum plots. Actually, this static-part residua have been significantly reduced, already when we have introduced the corrected ACD calibration method acd_calTabs2() for getting measured Tsky+src from directly recorded powers: The improved ACD calibration fixes the AC-spw P_off-based (i.e., with solar attenuation level) gain G* by transferring the gain-drop level (given by P_off/P_sky ratio) from the adjunged (the same frequency domain) SQLD spw. Hence, it looks these differences are due to imperfect ACD gain estimation: 1% error in the gain easilly produces ~70K difference in the solar T_B. Therefore, this method has been devised to try to improve the gain calibration.

How it works: We assume that the antenna gains and P_inst are constant in time over a scan. Moreover, for the spectral AC SPWS, we assume that the normalised gain g(f)=G(f)/<G(f)>, as used in acd_calTabs2(), is correctly calculated by the ACD calibration, and here we calculate just its scaling. Hence, for NT time-samples and NDC = nSPW x nPols = 24 data-channels, we have 2xNDC unknown instrumental variables (G + P_inst per SPW and pol) and, 3xNT 'natural' variables (PWV, T_B_XX, T_B_YY). This is still significantly less then number of equations, which is NT x NDC. Nevertheless, the matrix of such system has very high condition number. It is connected with the fact that we have no anchor for the instrumental parameters (namely gain scale). Situation is improved a lot if we add a few further equations - the equations for the ADC calibration - into the system. This way we solve the ATM-RT problem and (correction to) ACD calibration simultaneously. For that reason, the 'grand' system is really large - we have (NT + 2) x NDC equations (after linearization of the NL RT equations), where '+2' is for the ambient- and hot-load ACD calibration equations, so it is overdetermined. We use the 'normal' equation approach, i.e., solving minimizer for the quadratic form of the least-squares. Still the normal square matrix is of the order 3 x NT + 2 x NDC and its direct inversion takes ages. However, it has been found that the normal matrix for the extended system (natural variables + instrumental parameters) can be conveniently inverted with knowledge of the (easilly calculated) inversion of the original matrix for the natural variabless (as used in dWdB_Tbs()) - the procedure is demonstrated in extMatInv() auxilliary routine at the beginning of this script. Lastly, we express the temperatures in kiloKelvins in order to have comparable sensitivities for dW adn dT_B (which play rolse of matrix elements) and improve thus the condition number of the matrix.

3.2.2.16 dWdBdG2()

```
dWdBdG2 (
                vis,
```

```
                atmScan,
                tgtScan,
                myAtm,
                antID,
                spwList,
                T_tgt = -1.0e6,
                flaggedData = [],
                myMS = None,
                myMsmd = None,
                acdLoads = None)
```

```
The dWdBdG() extended by a bond to the expected time-averaged target
brightness temperature *T_tgt* (in Kelvins).

Rationale: When observing the target, e.g. the Sun, we expect to obtain the
same value of its time-averaged brightness temperature at all antennas.
The time variations of T_B might be a bit different per antenna due to
scintillation, but the time average over ~7minute scan should give the same
value. If we use dWdBdG(), which calculates T_B(t) independently per
antenna (and yet per XX/YY polarization), it is not very true: The T_Bs
are somehow centerred around the theoretical value calculated by Alissandrikis
formula (see below in the code) but they have per-antenna scatterring around
that value with RMS ~200 K - see Fig. 11 in PR16. For that sake, we complete
our grand-system of equations, described in dWdBdG(), yet by two additional
equations (one per polarization), which enforce that the <T_B(t)_XX/YY> equals
to the prescribed value of *T_tgt*, if it is supplied (the default value -1MK
means "do not use it").

Implementation: The new equations make the grand-system yet more complicated,
namely, the 'natural' part of the (J^T x J) matrix is no longer block-diagonal
but still it is posssible to invert the matrix relativelly 'cheap' way.
In addition to N-, G, and A- blocks, knowm from dWdBdG(), we introduce yet
the M-block for the two additioonal "Mean" (over the array) equations.
Similarly to the A-block equations, that enforce fulfilling of the ACD
calibration equalities, and the 'relative strength', of how much we push to
the ACD equations to be exactly fulfilled, is expressed by the factor
'calWeight', here we introduce a factor 'mWeight', which quantifies the
relative strength of the <T_B(t)^XX/YY>=T_tgt in the grand-total system of
equations.

Moreover:

1. Based on the results shown in the PR16, Fig. 11, we have introduced
a bit modified antenna efficiencies, set by antenna type (CM, DA, and DV).
The goal is to diminish apparent systematic (i.e by antenna type) <T_B(t)>
variations - the 'steps' in Fig. 11/PR16. The relative antenna efficiencies
have been studied also in the XLS table project22_25_Tb.ods.

2. SQLD spws have significantky shorter integrations and therefore also
received powers and gains (because G=dP/dT). In order to compensate this,
we scale all equations by the gains, i.e., we express them in temperatures.

3. A new diagnostic tool has been added in order to auto-detect corrupted
data. It is based on the error-vector per SPW/pol channel. The SPW:Pol
data with extreme error are now reported, in future version they will be
auto-flagged.
```

3.2.2.17 dWdBdG2_weights()

```
dWdBdG2_weights (
                vis,
                atmScan,
                tgtScan,
                myAtm,
```

```
              antID,
              spwList,
              T_tgt = -1.0e6,
              flaggedData = [],
              myMS = None,
              myMsmd = None,
              acdLoads = None)
```

Essentially the same as dWdBdG2(), but now we compose & solve the grand-system
for entire range of calWeights (from 100 to 1000) and mWeights (100 - 10000)
run in a loop.

Why we do not make the dWdBdG2() parametrized by *cal/mWeight* and run just
the wrapping function, which calls dWdBdG2() in a loop with varying weights?
The reason is technical: The most time-consuming operation is reading
and processing the sci-SPW data. In this function we have a loop just over
the grand-system solution, which represents less then 10% of CPU time spent
in dWdBdG2().

The goal is to find, how much the mean parameters <PWV> and <TsunXX/YY> are
sensitive to the calWeight/mWeight selection. Note, that the fluctuations dW,
dTBxx, and dTbyy, are quite robust - they do not significantly change even
between various dPWV proxies like dWdB(), dWdB_Tbs(), dWdBdG[2]() or
dWdB_Tbs_NL() and are, to a large extent, independent of their respective
means.

Moreover, we are checking more in detail (with much finer cal/mWeight sampling
over the selected interval), how the particular errors (Xi-squares) for
the natural (pwv, T_bXX/YY),instrumental (G, P_inst), and array-mean (T_tgt)
parts of the solution vector depend on cal/mWeight. Lastly, the number of
iterations needed for the convergence is being watched.

Implementation: We vary always just one weight (cal- or mWeight): the second
is set static to its optimised value found by variation. Selection is
performed in the code - comment/uncomment selection of *theWeight* below.
Weight range and sampling is controlled also manually by wLow and wHigh range
and the number of samples code in wItSize.

Results: It has been found that the optimum value for calWeight lies between
100 and 300. At low values, the Xi^2 for instrumental variables is high and
is improving with increasing calWeight. Nevertheless, around 200 the
improvement of Xi^2 inst. slows down, while, at the same moment, the 2nd part
of the error vector, the Xi^2 for natural variables, start rapidly increasing.
The <PWV> and <TsunXX/YY> are pretty flat in this region of calWeights, so
it is safe to select catWeght=200. This is actually the value eventually used
in production version of dWdBdG2().

3.2.2.18 dWdBdG_calWeights()

```
dWdBdG_calWeights (
              vis,
              atmScan,
              tgtScan,
              myAtm,
              antID,
              spwList,
              flaggedData = [],
              myMS = None,
              myMsmd = None,
              acdLoads = None)
```

--- OBSOLETTED---

This is an older version of dWdBdG2_weights(), which supersedes this function.

3.2.2.19 dWdLs()

```
dWdLs (
              antID,
              myAtm,
              fullPWV = False)
```

*** Specific for 2022.1.00031CSV project! ***

Decompose the WVR raw variations for antenna *antID* in the solar scan into
PWV variations and varying illumination of the WVRs by scaterred solar
irradiance Lsun. These two should be separable because we have 4 WVR channels
(three of them usable, one saturated) and dPWV and dLsun have different
frequency dependence - PWV-related signal increases towards the WV line center
while the solar illumination (thinned by atmospheric absorption) decreases.
The atmosphere tool *myAtm* must be set beforehand using setAtmTool() for
the scan no 9 (the solar scan) and SPW 4 (WVR data).
The boolean switch *fullPWV* controls, whether just variations around the
means shall come out for the extracted PWV data (the default is false, which
is usefull for comparison with dPWV proxy) or full PWV value with the mean
component added should come out. The Lsun contribution is always returned
including its mean.

Reasoning: By comparison between our dPWV data and raw WVR signal it has been
found that the DV antennas exhibit, in general, excellent correlation between
those two quantities (WVR raw data and dPWV from science SPWs), while the DA
antennas are obviously much worse. It looks like the reason is to be found
in a different antenna construction - namely supports of the sub-reflector.
These obstacles may lead to shadowing or, vice versa, additional reflection
of the solar illumination to the WVR. Hence, the solar illumination needs
to be handled as variable (cannot be fully subtracted in the stationary part).

We use formula (16) in PR15 to make the decomposition. Alhorithm for numerical
solution of Eq. (16) is the following:

1. Make scan-averaged (may be decomposed to off-source and on-source subscans)
WVR channel temperatures <Twvr>. Since dW(t) and dL(t) variations should
have (by definition) zero time average, three equations

<Twvr>=T_BS(w) + <Lsun>*tr(w)

should hold for WVR channels 1 - 3. We neglect the WVR channel 0, as it is
saturated, exhibiting minimum variations (even the off-source -> on-source
transition and back is not visible in the data). We vary the PWV content w
such a way, that dispersion of three results for <Lsun> is minimized.
The sky brigtness T_BS(w) is approximated by at.getTebbSkySpec() using the
myAtm model of the atmosphere, the transmisivity tr(w) is calculated from
the same model. Newton-Raphson (N-R) iterations are used for minimizing the
<Lsun> dispersion over the WVR channels.

2. Once optimized <PWV> and <Lsun> are found, we calculate the matrix
coefficients and the RHS for the overdetermined system of equations
(three equations for 3 usable WVR channels and 2 variables - dW and dLsun)
and use the standard quadratic-form miminizing to find the solution
(see the same procedure in dWdB() for details).
The solutions dW(t) and dLsun(t) are checked for zero time-average: If
a non-negligible value is found, the mean PWV as found in Step 1 is
corected

<PWV>_new=<PWV>_old + <dW(t)>

and likewise the <Lsun>, and Step 2 is repeated until near-zero avarages are
reached.

3.2.2.20 dWdLs2()

```
dWdLs2 (
              myAtm,
              antID,
              tgtScan,
              flaggedWVRchannels = [],
              ssType = '',
              solIllumination = True,
              myMsmd = None,
              acdLoads = None)
```

*** Specific for 2022.1.00031CSV project! ***

An updated version of dWdLs(), which is improved by the following features:

- Included antenna efficiencies.
- Improved solar spectrum as in dWdB_Tbs()/dWdBdG().
- Improved solution of the system using Jacobians as in dWdB_Tbs()/dWdBdG().

Essentially, we may use the same basic equation for Tsrc+sky as we did for sci SPWs in dWdB_Tbs() - the scattered solar radiation (both forward or rear) contributes always by term of the form r_eff*tr(w(f))*Tsun(f), where r_eff is the fraction of solar illumination effectivelly scatterred into WVR detector and this factor already contains the antenna efficiencies. Since r_eff is time dependent and unkonwn, we calculate just the product (r_eff(t)*Tsun0(t)). Here, we absorb r_eff into the solar temperature and we designate the entire that product simply by Tsun0(t). As in dWdBdG() we write Tsun(f,t) by means of the normalised spectrum as T(f,t)=Tsun0(t)*SF(f). Hence, differences to dWdBdG() are not so big and they are contained in the following:

- We do not have instrumental vars (gains/Pinst), the normalised spectral gains are set to be uniform and equal to 1.0.

- There is no polarization separation as the WVR data are just Stokes I.

- We have only 4 WVR channels modelled by 4 ATM spws: No data->model SPW map.

- Tsun0(t), representing just the part of solar radiation scaterred to WVR, is typically 50 - 100 K only.

As explained in dWdLs(), this procedure is bound to the specific MS, where we had good luck and the WVRs are not saturated thanks to close-to-limb target observation.

3.2.2.21 dWdLs3()

```
dWdLs3 (
              vis,
              myAtm,
              antID,
              tgtScan,
              flaggedWVRchannels = [],
              ssType = '',
              solIllumination = True,
              myMsmd = None,
              acdLoads = None)
```

Not strictly specific for the project 2022.1.00023.CSV (Band 5 data), but for the other Band 5 datasets it gives nonsense for the solar on-source

subscan because of the WVR saturation (only the above mentioned project
is 'lucky' as the WVR has been pointing sligthly off-limb - see dWdLs().)

An extended version of dWdLs2(): Similarly to dWdBdG() we now allow for
limited corrections of the 'gains' and 'intrumental offsets' of WVR
radiometers. The reason is similar to the sci-SPW data: The calibration
is not exact and the gains may be wrong within a couple of percents.
Combination with the science data can help to fix these little discrepancies.

Since the WVR data contain already ACD-calibrated temperatures, we have
just a relative gains (close to 1.0) and offsets (close to 0.0) to become
new instrumental variables of the system. The 'ACD' calibration equations
thus take the form:

G*Tamb+P_i=Tamb
G*Thot+P_i=Thot

If left alone, they have solution G=1.0, P_i=0.0. But now they are used in
combination with the science data in the grand-system like in dWdBdG().

3.2.2.22 dWdLs3_calWeight()

```
dWdLs3_calWeight (
              myAtm,
              antID,
              tgtScan,
              flaggedWVRchannels = [],
              ssType = '',
              solIllumination = True,
              myMS = None,
              myMsmd = None,
              acdLoads = None)
```

--- OBSOLETTED ---

Replaced/superseded by a new dWdLs4_weights() function.

3.2.2.23 dWdLs4()

```
dWdLs4 (
              vis,
              myAtm,
              antID,
              tgtScan,
              flaggedWVRchannels = [],
              ssType = '',
              solIllumination = True,
              myMsmd = None,
              acdLoads = None)
```

Improved version of dWdLs3().

Inspired by dWdBdG2(), we now handle the situation of enforced zero solar
illumination (setting *solIllumination=False*) more appropriatelly.
The insofar procedure in dWdLs3() sets Tsun=0.0 hard and 'degrades' the T_B
variations dB from a variable to a constant (we narrow the J_wb matrix by
cutting the dB column) and it is set as a 'hard-wired solution' to 0.0.

In this new implementation we keep the T_B to be a variable, but we set its
time-averaged value Tsun to 0 by implicit setting of parameter, corresponding
to *T_tgt* in dWdBdG2(), to zero. In other words, we add the equations that
force the time-averaged solar-illumination Tsun to zero, still keeping its
small variations dB. As a result, the zero-illumination condition is not
super-seded to other equations, but it has its relative strength in the
entire system, expresssed by internal (experimentaly found) parameter mWeight.

As in dWdBdG2(), we have specific antenna-efficiencies setup per antenna type:
This is another 'upgrade' to dWdLs3(). Unlike the dWdBdG2() we need not
to introduce a 'natural' weight factor WF=1/G, because our 'raw' measured WVR
signal is already expressed as temperature and all 4 WVR channels have the
same weight (this is not true fro sci SPWs, handled in dWdBdG2(), where SQLD
spws have different integration time and, consequenctly, powers and gains).

3.2.2.24 dWdLs4_weights()

```
dWdLs4_weights (
              vis,
              myAtm,
              antID,
              tgtScan,
              flaggedWVRchannels = [],
              ssType = '',
              solIllumination = False,
              myMsmd = None,
              acdLoads = None)
```

Similar to dBdWdG2_weights(). We are varying the *calWeight* and *mWeight*
parameters (one at a time, selected by commenting/ucommenting the line
with *theWeight* below in the code) in order to find their optimized values,
which somehow minimize the combined error (residuum) of natural, instrumental,
and mean-array parts of our grand-system of equations.

Implementation: Basically, we use the dWdLs4(), where the main solution-finder
iteration loop is nested in an outer loop of varying calWeight/mWeight. Here,
some explaining comments are abbreviated - see dWdLs4() for full detail.
Moreover, the plots of all time-dependent quantties are omitted.

3.2.2.25 extMatInv()

```
extMatInv (
              Ainv,
              Lb,
              Rb,
              LRc)
```

If we know inversion *Ainv* of a square regular matrix AA, we can
efficiently calculate inversion the matrix AA extended by a lower belt *Lb*,
right belt *Rb*, and a lower-right corner *LRc*, i.e. of a compound matrix
M = [[AA,Rb],[Lb,LRc]]

This is used in dWdBdG() method below, where a large, block-tridagonal matrix
(whose inversion is easy to compute) is extended by the (relativelly narrow)
belt-matrices at its right and low ends connected with the instrumental
variables.

Returns: M^(-1), i.e. the square matrix (2-d ndarray) that is inversion
of the extended matrix M = [[AA,Rb],[Lb,LRc]].

3.2.2.26 G_off_cal()

```
G_off_cal (
              vis,
              atmScan,
              tgtScan,
              ant,
              acSpw,
              myMS = None,
              myMsmd = None)
```

An experiment, how to transfer relative gain-drop between G* and G in the *sky* and *off* scans (due to switching the increased attenuation level on) from SQLD to the corresponding (occupying the same frequency domain) AC SPWs. Findings reached by experimentation with this method are then used in improved ACD calibration method for the Sun acd_calTabs2().

In theory, the AC SPWs have low dynamic range for being able to measure P_off in the highly-attenuated off-Sun position (Shimojo et al. 2017, p. 10, right above Eq. (6)). Here, by means of linear proportionality between SQLD and AC powers (P_sqld = K*P_ac+Q) for the same frequency domain, where the parameter K depends on the ratio G*_sqld/G*_ac, we evaluate the the derived G*_ac and compare it with its (suspicious) value obtained by measuring P_off directly in the AC spws. The scaling factor between the sqld-to-ac transfer derived G* and the AC-P_off calculation is returned.

For the spectral-resolved AC spws we assume that the spectral gain normalized to unity remains constant during switching to the higher attenuation level, i.e., that only its scale is changing. Here we apply as much as exact calculation, which takes into account frequency dependence not only of this normalised gain, but also absorption/emission of the terrestrial atmosphere, and the normalized spectrum of the solar incident radiation. These factors enter into calculations of P_ac averaged over the resolved SPW as wieghts in the sums over the SPW channels. It has been experimentaly found, that the influence of solar spetral slope is negligible in comparison with the other two channel-dependent factors (normalised spectral gain and transmissivity). Hence, based on this research it is neglected in the improved acd_calTabs2() routine.

3.2.2.27 getMeteo()

```
getMeteo (
              vis)
```

Get time series of the basic meteo characteristics. They are not per antenna as there is just a single meteo station for entire array. Still we have much better time coverage than the ASDM_CALATMOPHERE table. The latter has, seemingly, pr-antenna values but in fact they are the same for all antennas and differ only for different times. There are much less time samples in ASDM_CALATMOSPHERE than in WEATHER table.

We return the meteo characteristics as a time series in a map.

3.2.2.28 getMeteoForScan()

```
getMeteoForScan (
              vis,
              scan,
              myMsmd = None)
```

The meteo characteristics as extracted by getMeteo() form the MS *vis*, now averaged and interpolated to the center of the *scan*. The msmd tool *myMmsmd* may be conveniently supplied if it is already attached to *vis* - it improves performance.

Basically, it is intended as a more accurate replacement for aU.getWeather(). What is the difference between the two? The aU.getWeather() finds the meteo values 'bracketted' by the beginning and end of the scan and makes their average. Thsi approach forgets even the recent history - if there is a meteo measurement point just a second before the scan begins, its value is neglected. The same holds on the 'future' end. In contrast, getMeteoForScan() samples the scan by 100 time samples, for each of them makes linear interpolation of the meteo values and eventually, this array is averaged. This approach takes into account the pre- and post-scan meteo measurements, too, with their weights naturally set by the time distance of the meteo record from the actula time-sample of the scan.

For the solar scans the tiny differences are not so important, the signal is dominated by the brightness temparature of the Sun, the T_atm correction is negligible. On the other hand, for the BP scans, the (T_sky-T_atm) expression, which stands in the sensitivity factor for dPWV, has a typical value about 3K, so difference in T_atm of 0.5K makes ~20% error!

3.2.2.29 getPhase()

```
getPhase (
              gTable,
              antID,
              spw)
```

Extract phase variations as a function of time from the gain-cal calibration table *gTable* for antenna with *antID* and in the spectral window *spw*.

3.2.2.30 getPhaseMS()

```
getPhaseMS (
              vis,
              antID,
              spw)
```

Instead of using the gaincal() solution contained in the gainTable (as getPhase() does), try to extract phases directly from the MS using the DATA and MODEL columns.

We are doing this trick just for the solar MS, so the refAnt and tgtScan parameters are hardwired at the beginning of this routine.

3.2.2.31 getPWVforAnt()

```
getPWVforAnt (
              vis,
              tgtScan,
              antID,
              myMsmd = None)
```

In wopacity() we set averaged PWV over entire array and entire EB duration.
For estimation of T_atm (which depends on atmospheric transmissivity) this
is probably OK.

On the other hand, in dWdB() we have the matrix coefficents for (apparent)
source brightness variations (dB) depending on the transmissivity, too, now
calculated per-antenna. To that end, we extract the PWV values right
before and right after the *tgtScan* for specific antenna *antID*. We
use the MS/ASDM_CALWVR table to get the PWV per time and antenna. We find two
PWV values that most closely (in time) embrace the *tgtScan* and return their
average - value interpolated in time to the time-center of the *tgtScan*.
Moreover, as a second value in the tupple, we return the linear slope of
the PWV change during the *tgtScan*

See also aU.readPWVFromMS() and related functions for explanation, how we
extract the PWV data.

3.2.2.32 paregression()

paregression (
arrX,
arrY)

Generalized/parametrized linear regression calculation between the arrays
arrX and arrY. In addition to the standard lin. regression (handled, e.g.,
by np.polyfit()), here the values y_i of the arrY may be dependent not only
on x_i, but also on the index i itself.
Mathematically, we are finding the best fit of the data in the form

y_i-t*i=K*x_i+Q

Rationale: If both arrays X and Y are time series (with regular cadence),
the Y values may be dependent not only at X, but also on time (represented
by the index i). Here we expect that there is - in addition to linear
dependence on X - also a linear trend of Y-array in time. The goal is to
optimize the parameter 't' of linear trend in time such a way, that we obtain
the best linear correlation between the X-array and time-detrended Y-array.

Math behind the calculations here is described in PR12 (May 2025). Application
in the context of this research is for correlation between phases and PWV
variations as calculated in dwXdpi_corrs() below.

Input: arrX, arrY - 1D float np arrays (time series of the same size).

Returns: A dictionary with the values of

tTrend: Time-trend parameter.
K: Coefficient of linear correlation between X and time-detrended Y
Q: Offset for linear correlation between X and time-detrended Y.
Xi2: Xi-square - how good is the fit.

3.2.2.33 phiFixCompare()

phiFixCompare (
vis,
atmScan,
tgtScan,
refAnt,
spwID,
spwRes = -1)

Compare standard WVRgcal()-based correction of *BP-cal* phases with that based
on our dPWV() proxy.

The dPWV() can be calculated either from SQLD or AC SPWs, the AC still
having two sub-options: Channel-integrated or channel resolved SPW.
In the latter case, dPWV_spwPart() is called internally, with integration
only over the lower half of the SPW channels.

Which data are used, it is auto-detected from the *spwID* parameter.
In case of the channel-integrated data (namely SQLDs) we need to specify
the SPW with channel-resolved data in addition: This is used (1) for the
calculation of (usually small) correction factor dTsky/df that implies from
integration over the SPW bandwidth in case that tau_1 and Tsky are function
of frequency, (ii) for comparison with raw and wvrgcal()-corrested phases;
these are measured always in spec-resolved SPWs. The correspsonding resolved
SPW is given by the parameter *spwRes*. If spwID is a SQLD spw, then
this *spwRes* MUST be set (we have no chance to deduce it). On the other hand,
in AC-integrated data it can be inferred as spwID-1 and in case of AC-resolved
data it is just directly the *spwID*.

First part of the job basically replicates the code in blindPhiFix() in order
to see comaprison between ucorrected phase and its improved counterpart fixed
using our dPWV() proxy and the theoretical factor -25.0 (called "Method 1" in
blindPhiFox())

The second part lies in taking the p-cal table (pre-Bandpass) with data
corrected also by WVR calibration (the 'standard' p-cal table for the BP
calibrator is NOT corrected for WVR) as the source. The following sequence
of steps has been performed in order to obtain the WVR-corrected p-cal table
mso+'.ap_pre_bandpass-wvrFix' in CASA shell:

@code{sh}
clearcal('uid___A002_X110fd13_X1099e.ms',field='1')

wvrgcal(vis=vis,caltable='wvr_gcal.W', segsource=True, sourceflag=['Sun'],
statsource='J1256-0547',refant='DV10',maxdistm=1.0,minnumants=3)

from casarecipes.almahelpers import tsysspwmap
tsysmap = tsysspwmap(vis = mso, tsystable = mso+'.tsys', tsysChanTol = 1)

gaincal(vis=mso,caltable=mso+'.ap_pre_bandpass-wvrFix',field='1',scan='5',
solint='1.92s',refant='DV10',calmode='p',
gaintable=[mso+'.antpos',mso+'.tsys','wvr_gcal.W'],
gainfield=['','1','1'],interp=['','linear,spline','nearest'],
spwmap=[[],tsysmap,[]])

@endcode

This way, antpos, Tsys, and WVR corrections has been applied on-the-fly.

This method cannot be used - even hypothetically - for the solar scans
as we have no wvrgcaltable for the Sun to be compared with the dPWV() proxy.

3.2.2.34 phiFixCompare3()

phiFixCompare3 (
vis,
atmScan,
tgtScan,
refAnt,
spwLow,
spwRes = -1)

Like phiFixCompare(), but we use dWdP() - the method that takes into account

the dry-component-caused power fluctuations - for determining the PWV
variations instead of dPWV() (which takes just a single SPW for the power
fluctuations).

Since dWdP() works currently only for chan-integared SPWs, the spwLow may
be just the lowest SQLD or AC-integrated SPW.

dWdP() omits the correction for slopes of <Tsky(f)> and tau_1(f) so it does
not need the spwRes. However, we still need it for the raw and wvrgcal()-fixed
phases to be compared with our corrections.

3.2.2.35 plotPhases()

```
plotPhases (
              vis,
              gTable,
              spw)
```

Extract phases from the table *gTable* for a given *spw* using getPhase() and
make series of plots (as PNG files) for all antennas contained in the current
MS *vis*.

3.2.2.36 pwv4scans()

```
pwv4scans (
              vis,
              solWVR = False)
```

Calculate PWV(t) from the raw WVR data using dWdLs2()/dWdLs3() in four
consecutive scans around the solar scan - start with leading phase-scan and
finish with the trailing phase-cal scan.

For comparison, calculate the sciSPW-based PWV(t) proxy by dWdBdG() for the
solar on-source subscans, too.

WVRs in the solar on-source subscan are disabled by default (*solWVR* = F),
because, in general, they are saturated. May be set to T for the specific
MS: Band 5 data of 2022.1.00023.CSV (WVR are ~10 arcmin off-limb)

Save WVR + sciSPW data to './pwv4scans.npy' for 12m antennas and sciSPW data
extra for all antennas to './PWV_TB_GP_DATA-antEff.npy'. Both files are open
in 'append mode, the recent-most results are the last (the older may be used
for comparison of different approaches, e.g.dWdLs2() vs. dWdLs3(), solar
contamination in off-the-Sun scans allowed or not, etc.)

3.2.2.37 pwv4ScansCompare()

```
pwv4ScansCompare (
              vis)
```

More sofisticated comarison between PWV as determined from sci SPWs
via dWdBdG() method with the value calculated from the raw WVR temperatures
via dWdLs3().

We compare for all 12m antennas (the 9m CM antennas do not have WVRs) and
for sake of efficiency we use pre-calculated PWVs stored in './pwv4scans.npy'
via pwv4scans() and re-read by pwv4scansRestore().

3.2.2.38 pwv4scansRestore()

```
pwv4scansRestore (
              vis,
              hasSolWVR = True,
              runID = 100000)
```

Extract data saved by pwv4scans(). The parameter *runID* controls, which run
of pwv4scans() should be extracted (pwv4scans() appends the new results to
the end of existing files). The default value of 100000 means to take the
recent-most data. The *runID* starts counting from 1.

Parameter *vis* is here for determining the antenna array.

The flag *hasSolWVR* (default = T) indicates, whether the data contain also
the on-source subscan WVR data.

3.2.2.39 pwvCalc()

```
pwvCalc (
              vis,
              atmTool,
              scan,
              antID)
```

Calculate PWV for the *scan* and *antID* using raw WVR data. We use
linearized formula (13) in the PR15 in an iterative way to converge
to dW=w-w0 to be zero - basically we use the Newton-Raphson (N-R) method. The
atmosphere tool *atmTool* must be set beforehand via setAtmTool() for SPW 4.
The sky brightness temperature Tbs is calculated from the ATM model using
at.getTebbSky() method. We have also tried at.getTrjSky() as an alternative
and compared the results with the (rare) SYSCAL PWVS via pwvCalcVerify() -
see the comment on comparison results there.

Returns: PWV obtained by averaging the partial results from WVR channels 1-3
and its stddev in a tupple. We skip channel 0 (the line core) as it gives
outlied data from the rest (frequently also NaNs). Essentially, for the
'solar observing conditions' with PWV typically > 3.0 it is, anyway, mostly
saturated and, hence, insensitive to smaller PWV variations.

3.2.2.40 pwvCalcVerify()

```
pwvCalcVerify ()
```

Try to 'verify' our PWV calculation in pwvCalc() against the ALMA-calculated
values in ASDM_CALWVR table, accessed by getPWVforAnt().

*** Specific for this MS! ***

We calculate the PWVs by both methods for all antennas and all non-solar scans
and make a scatter plot. We can expect some differences as our pwvCalc()
method uses WVR temperatures ove a scan, while the system-calculated PWVs
are a bit opaque - neither the method of their calculation is clear (nothing
found in literature except they use the ATM model) nor the strange times at
which we have their values in ASDM_CALWVR table are explained. The goal is
to see, whether the dicscrepancies are small and do not have systematic
trends.

Results: The correclation works basically well, but there are, indeed,
systematic differences: If at.getTebbSky() is used for sky brightness
estimation (to be interpretted as Twvr values in WVR raw data), our pwvCalc()
method gives systematically underestimated PWV values. When at.getTrjSky() is
used, then our PWVs are systematically over-estimated. Detailed
results are in ./wvr_raw/pwv-SYS_vs_ours directory.

3.2.2.41 pwvCompare()

```
pwvCompare ()
```

```
*** So far just for the solar MS project 2022.1.00031.CSV, Band 5!!! ***

Compare systematically (for all antennas) atmospheric phase delays as
predicted from:

1. Our dPWV() proxy (calculated from signal of science SPWs),
2. wvrgcal()-produced caltable (calculated from the hi-cadence WVR data).

We can do it just for the BP calibrator, so no free source paramater in this
case. We are using the channel-integrated AC windows (not SQLD) in dPWV().

Creates two-panel plot (left for XX, right for YY) for each antenna,
having our dPWV() prediction in red and wvrgcal() calculation in blue.
```

3.2.2.42 rebin()

```
rebin (
              array1D,
              downSampleFactor)
```

```
This is a general-purpose helper analogical to the IDL's routine rebin().
It down-samples 1D-array *array1D* by the *downSampleFactor*. The dimension
of the array1D must be evenly divisible by the downSampleFactor! If not,
the original (non-resampled) array is returned and a warning issued. In such
a case, it is advisable to use more complex rebin2() method.

Rationale for utilization here: We are frequently trying to find correlations
between quantities like phase variations and our model PWV-proxy, and these
need not to have the same sampling rate (typically our PWV-proxy from
auto-correlations has cadence 0.192s; from the SQLD data yet shorter, by an
integer factor of 12, while the BP-calibrator phases are solved at interval
of 1.92 sec).
```

3.2.2.43 rebin2()

```
rebin2 (
              array1D,
              origX,
              downSampledX)
```

```
The rebin() method generalized for the case that down-sampled array elements
cannot be simply calculated as averages over *downSampleFactor* elements
of the original array, because the dimensions do not agree.

We submit three argument: *array1D* are the 1D data (e.g., recorded powers
in our SQLD spw), *origX* is the x-grid of the original data (i.e., the times
at which the SQLD powers are measured in the above example), and the
*downSampledX* is the new, downsampled x-grid (in the used example it is
typically the lower-cadence time-axis for the corresponding AC resolved spw).

We first define a new high-cadence x-axis with a gridding that is (i) as much
close to the original x-grid as possible, and, at the same moment (ii) the
new grid represents an even division of the low-cadence grid. Then we
can already call our standard rebin() method.

Rationale: Like with rebin(). This more generalised routine has been
introduced namely because of the non-comensurable (no even division) time
meshes for SQLD and the AC spws in some MSs.
```

3.2.2.44 restoreSol_dWdB()

```
restoreSol_dWdB (
              vis,
              filePath,
              runID = 100000)
```

Load the results of dWdB() & friends for entire ALMA array from a file and store it as a map with antenna as its key. Restoring works as a reverse routine to both saveSol_dWdB() and saveSol_dWdB_Tbs()

The *filePath* points to the NPY file with data.

The *runID* parameter (starting from 1) controls, which of the appended records should be read.

We need *vis* in order to determine the antenna array (Todo: Think about re-implementation of the corresponding save_() routines so that thay store antenna array at the beginning of the file).

It is a solar-specific routine.

See also saveSol_dWdB() / saveSol_dWdB_Tbs().

3.2.2.45 saveSol_dWdB()

```
saveSol_dWdB (
              vis,
              atmScan,
              tgtScan,
              spwList)
```

Create ATM model framework and then run dWdB() for all antennas in the array. Save the complete results (dW, dB, matrix, and RHS) into a binary NPY file.

See np.save() and np.load().

3.2.2.46 saveSol_dWdB_Tbs()

```
saveSol_dWdB_Tbs (
              vis,
              atmScan,
              tgtScan,
              spwList)
```

As saveSol_dWdB(), but now for data produced by either dWdB_Tbs() or dWdBdG().

The main difference is in the ATM model: While dWdB() requires static opacities (i.e. arrays) pre-calculated by setAtmosSciSPWs(), the dWddB_Tbs() and dWdBdG() routines require the full ATM tool (attool) as their parameter, here set and coupled to actual meteo situation via setAtmTool4SPWs().

3.2.2.47 setAtmosSciSPWs()

```
setAtmosSciSPWs (
              vis,
              tgtScan,
              spwList)
```

Calculate the ATM profiles using wopacity() for all channel-resolved sci SPWs.
Then, using the spw-mapping attribute the appropriate ATM profile to every
science SPW (chan-resolved and integrated ACs and SQLDs) in the *spwList*.
Use *tgtScan* for determining the approporate ATM model - we have scan-
dependent meteo parameters and airmass.

Used as a single-shot function in the methods that iterate over the antenna
array, like saveSol_dWdB(): We do initiation of the atm model for all spws
just once.

3.2.2.48 setAtmTool()

```
setAtmTool (
              myAtm,
              vis,
              spw,
              scan,
              myMsmd = None,
              wvHeightScale = 1.5,
              tempDropPerKm = -5.6)
```

Set the atmosphere tool (casatools::at) *myAtm* for later calculation of the
opacity (wet & dry component) as a function of frequencies for the single
spectral window *spw*. The *spw* may be either one of the AC-resolved sci SPWs
or the SPW=4 for the WVR data. The attool effectivelly represents CASA
interface to the underlying terrestrial-atmosphere radiative-transfer model
ATM by J.R. Pardo.

Input:
The spw setup and the atmosphere (meteo) profile model are extracted
from the input measurement set *vis*. Parameter *scan* determines (i) the
time-range, for which we seek the meteo parameters, and (ii) the target
object and its elevation (H) - we need it for the airmass (AM = 1/sin(H))
calculation. The WV height scale and temperature drop per 1km are
set to their default values, which give the best fit between our PWVs
as calculated by pwvValc() from the raw WVR data and the SYSCAL PWV values
for the 2023 Band 5 data - see pwvCalcVerify() for details.

Output:
The atmTool *myAtm* gets modified in this routine so that it is ready for
further use in, e.g., wopacity().
With this flexibility, we can later force the PWV value to be set by a user.

Our attool interface has been loosely inspired by aU.plotAtmosphere().
The aU.plotAtmosphere() utility, however, uses aU.getMedianPWV(vis)
eventhough its *pwv* arg is set, if the vis argument is present, so the pwv
argument is actually ignored there. Moreover, aU.plotAtmosphere() returns
just the representative-frequency (scalar) values; we need the frequency-
dependent opacity values (see wopacity()).

See aU::plotAtmosphere() and casatools::at for details.

3.2.2.49 setAtmTool4SPWs()

```
setAtmTool4SPWs (
              myAtm,
              vis,
              scan,
              myMsmd = None,
              wvHeightScale = 1.5,
              tempDropPerKm = -5.6)
```

Similar to setAtmTool() but now it makes the setup for the 4 channel-resolved
AC SPWS (i.e. no WVR spw possible to be set here).

As result, *myAtm* has 4 spws 0 - 3 mapped from the sci SPWs 5,7,9, and 11.
For the temperature, pressure and humidity profile we use the same setup as
in setAtmTool().

3.2.2.50 smooth()

```
smooth (
              array1D,
              kernel)
```

A simple smoothing of *array1D* performed by a convolution with the
much shorter 1-D array *kernel*. Kernel may be a box or at riangle function or
whatever similar with the finite (and rarther short, in comparison with
the array1D length) support.

In order to handle the edge effects, we apply 'mirror boundary condition' to
array1D prior smoothing - we extend the original array at both edges prior
the convolution. This prevents wild behaviour at the edges (so the
edges do not affect, e.g., plotting y-range), but the edge data carry
inherently lower/degraded information and must be take cautiously.

3.2.2.51 sol_offPos()

```
sol_offPos (
              vis,
              tgtScan,
              refAnt)
```

Trying to find the minimum off-source separation from the Sun so that
WVRs are not affected by direct solar illumination.

We calculate, how far from the Sun (the disc center) we are when pointing
OFF_SOURCE during the solar scan. This is the position taken at the first and
last subscan of the solar scan and its purpose is to the basically sky/atm
measurement, but now with the attenuator settings used for the solar on-source
observation - see the G-star calculation in acd_calTabs()
It seems that the psoition of the firts sub-scan is identical with those
taken in the preceding ATM scan.

3.2.2.52 sol_PhaseFix()

```
sol_PhaseFix (
              antID,
              spw,
              pwvMap)
```

Compare the suggested phase-fix based on the full-PWV calculations as
performed by dWdBdG(), with the linear interpolation of the pahse-inf solution
obtained from the adjacent (bracketing) phase-cal scans. We would like to see
whether the phase-inf interpolated solution has at least some significance or
whether it is totally 'out', e.g. the phase wraps over 2PI in reality.

Results:
(1) The trend in PWV-based phases (Sun) roughly corresponds to the linear
interpolated phi-Cal solution.
(2) There is, however, an offset that may be even as large as 720 degs (i.e.
2 lambdas in path).
(3) A bit worrysome is the difference between XX and YY polarizations in
phi-Cal solution.
(4) Albeit the trend is corresponding between both, the PWV-based phases
exhibit variation over the scan that are easilly bigger then 360 deg.

3.2.2.53 sol_pwv2selfCal()

```
sol_pwv2selfCal (
              antID,
              spw)
```

Comparison of PWV-based phase calibration of the Sun with self-cal gaintable
solution. Because the self-cal phases are already apriori corrected by
the linear trend (via applycal() of phase-cal gaintable), we plot also the
phase-cal linearly-interpolated solution.

3.2.2.54 sol_pwvGcal()

```
sol_pwvGcal (
              pwvMap,
              vis,
              refAnt,
              pwvCalTableName = 'sol_pwvCalTable.W')
```

Create the phase calibration table for the Sun in the same style as
wvrgcal() makes it for non-solar objects.

We use the full-PWV time series per antenna as found by dWdBdG() routine
and stored in *pwvMap*.

@todo: Generalize for the case we have multiple solar scans.

3.2.2.55 solTgtPos()

```
solTgtPos (
              vis,
              tgtScan,
              refAnt)
```

Similar to sol_offPos(), but now we produce a graphic output of the actual
target-pointing relative to the solar disc center (i.e., angular separation
of those two) as a function of time over entire *tgtScan* for a 'reference'
antenna (the result should be the same for all) *refAnt*.

3.2.2.56 sqld_P0()

```
sqld_P0 (
              vis,
              ant,
              spw,
              myMS = None,
              myMsmd = None)
```

Find the P-zero offset for SQLD detector at antenna *ant*.
The *spw* parameter designates the spw for *standard* SQLD ATM scans [0,1,2,3]
and maps to the spws for the scan no. 1 (the 'zero' scan) [13,14,15,16]
in an obvious way [0,1,2,3] -> [13,14,15,16].

Returns: An array of shape 2 x 1, the first dimension being polarization.

See also acd_calTabs().

3.2.2.57 TbStats()

```
TbStats (
              resCG,
              resVG)
```

The mean brightness temperature of the Sun, as derived from dWdB_Tbs()
and its generalization dWdBdG(), should be essentially the same for all
antennas - we expect (like in dB_sum() above) that the 'scintillation'
burst/drops because of turbulence focusing/de-focusing effects are in average
zero over the scan.

Here we make staticstics over all antennas for two sets of the results:
One obtained by dWdB_Tbs() with constant gains determined by ACD calibration,
and the other, resulting from dWdBdG(), where we allow for the gain correction
via minimizing the grand-system (science + calibration measurements together).
The parameters *resCG* and *resVG* are the maps/dicts obtained as a result of
restoreSol_dWdB() for the constant-gain and variable-gain solutions,
respectivelly.

3.2.2.58 tgt_pow()

```
tgt_pow (
              vis,
              tgtScan,
              ant,
              spw,
              myMS = None,
              myMsmd = None,
              subScans = 'on')
```

Extract total-power data in *spw* for target defined by the *tgtScan* number
at the antenna *ant*.
The *myMS* and *myMsmd* tools attached to *vis* may be supplied for speed-up.
The *subScan* is relevant just for the solar target and may be empty
(entire scan) or 'on'/'off' for the on-source (the default) and off-source
sub-scans selection.

Target, antenna, and spw params MUST be int, not strings - see acd_calTabs().

Performs MS selection in order to extract the raw auto-correlations (AC) or
SQLD data (depending on the SPW selected). We get the time-series of the
total powers (either AC or SQLD); if the SPW is channel-resolved (spws
5, 7, 9, and 11), we have even time series of the spectra. If the *tgtScan*
is a solar scan, we have to select just the 'internal' subscans, omitting the
very first and very last sub-scans, as they are taken in the off-source
position for calibration (see acd_calTabs() for detail).

Note: If intended to be used with acd_calTabs(), the ATM scan in acd_calTabs()
should as much close as possible precede the target scan in tgt_pow().
This is the case, for example, with the arguments *atmScan* and *tgtScan*
in tgt_Tb().

Returns: A tupple with two components.
The 1st component: Times for each amplitude as MJD - an array of size nIntegs.
The 2nd component: Array of powers in the form 2 x nChans x nIntegs.
I.e., 2 x 1 x nIntegs for the SQLD and chan-integrated SPWs, and
2 x 128 x nIntegs for channel-resolved SPWs. The first dim (2) means
polarization (XX, YY).

3.2.2.59 tgt_Tb()

```
tgt_Tb (
              vis,
              atmScan,
              tgtScan,
              antID,
              spwID,
              myMS = None,
              myMsmd = None,
              acdLoads = None,
              myAtm = None)
```

Calculate ACD-calibrated 'sky' brightness temperatures from the measured
total-power (AC or SQLD) data. The 'sky' Tb is a sum of the atmosphere
contribution and the attenuated (by passage through the same atmosphere)
signal of the source (target) defined by the *tgtScan* parameter. The
atmScan should be the nearest ATM scan preceding to the target scan.

Return: A tupple of three arrays. The first one is a time-axis, the 1-D array
of size nInteg, merely reproduction of the time-axis (as MJD) from internally
called tgt_pow(). The other two arrays - Txx and Tyy, are each of the shape
nChans x nIntegs, nChans is either 1 (SQLD and chan-integrated AC SPWs)
or 128 (chan-resolved AC SPWs) in case of Solar Band 5 hi-res MS.

3.2.2.60 tgtPow_crossPlots()

```
tgtPow_crossPlots (
              vis,
              ant,
              tgtScan,
              spw1,
              spw2)
```

Compare the power variations (mean-subtracted powers) for target in *tgtScan*
in two different SPWs *spw1* and *spw2* (should be of the same type: either
AC integrated or SQLDs - not checked!) for a given antenna *ant*.
Specifically, we do scatter plot of power variations in *spw2* vs. powers
in *spw1*.

Idea: Do we see dry-component or source intrinsic contribution to the power
variations in the SPWs further from the WV line center? The ultimate question
is, which of the approaches to the dPWV proxy shall be used - whether
dPWV() / dPWV_spwPart() or dWdB(), which counts with the (apparent)
variability of the source, too (and may accound for adding the dry-component).

Result: When used for the BP calibrator, we see just a single linear
correlation of the signal between the *spw1* and *spw2*. Its slope more or
less corresponds to theory, given the big relative uncertainty in Tsky-Tatm.
For the Sun, however, we see a sort of two timescales, each with a different
slope. It has been found that the slower-variations linear slope corresponds
to the ratio of matrix coefficients for dW, while the faster one for dB,
when decomposing via dWdB() method is applied. Seee dWdB() for details.

3.2.2.61 unwrap2()

```
unwrap2 (
              phase)
```

Sometimes we have phase jumps exactly by 180 deg. These seem to be artificial
but not cured by the standard np.unwrap(). This method aims to remove also
these residuals that are well less the 2PI (which are cured by np.unwrap()).

Input: phase in rads - 1-D numpy ndarray
Output: phase_unwrap in rads - 1-D numpy ndarray with the same dimensions.

How does it work: We differentiate the original array, fix the high jumps
(positive and negative extra) in the array of differences and then
're-integrate' the corrected difference series back via the cummulative sum.
In order to skip seekeing and evaluating the 'super-high' jumps (2PI and more)
we call first the standard np.unwrap() prior to our residual-jump treatment.

3.2.2.62 wopacity()

```
wopacity (
              vis,
              spw,
              scan,
              userPWV = -1.0)
```

```
Uses the atmosphere tool (casatools::at) for calculation of the opacity
(wet & dry component) as a function of frequencies for the science spectral
window *swp*. We preset the atTool via setAtmTool().


Return value: dictionary with the following components:
airMass: 1/sin(h) for the object observed in the scan *scan*
tau_1: 'Wet' component of the zenith opacity for PWV=1.0 as a function of
frequency channels (1d-array).
tau_d: Dry component of the zenith opacity as function of channels.
tau_w: Wet zenith opacity as a fubction of channel for the actual PWV.
T_atm: Effective temperature of the atmosphere (used in (1-trans)*Tatm expr.).
```

3.2.2.63 wrapBack()

```
wrapBack (
              phases)


An 'inverse' function to np.unwrap()
```

3.2.2.64 wv2path()

```
wv2path (
              vis,
              spw,
              scan,
              pwv)


We use the factor -25.0 for conversion between dW and dPhi, as it implies
from the Richard Hill's original draft. (RH uses effective refractive index
r_eff=6.7, independent on frequency. When the optical path is recalculated
to phase delay for average lambda ~1.5 mm, we arrive to factor -25.0).

This routine verifies, whether this assumption is correct and what are its
limitations. We use ATM modela and its get***Path() routines.

Result: Indeed, the non-dispersive part is constant over SPW 5 and its value
is ~6.7e-3 m(!) per 1 mm of PWV. The dispresive part is very slightly
changing around its average value of 1.0e-4 m per 1 mm PWV and thus,
represents a small correction. Both paths depend linearly on pwv, so
their effective (per 1 mm PWV) value does not depend on the *pwv* parameter.
On the other hand, they slightly depend on the airmass (i.e. on the *scan*
parameter). With rising SPW the total (disp.+ non-dispersive) path per 1mm
rises (up to ~6.9e-3 m per 1 mm PWV). So, for approximate calculations it is
fine to use the factor of approx -25 (actually -27). However, for real
application, where we use correct central frequencies for each SPW, it is
better to calculate the phase-delays directly from the ATM model.
```

3.2.2.65 wvr_dataRaw()

```
wvr_dataRaw (
              vis,
              tgtScan,
              antID,
              subScans = '')
```

Extract the raw WVR data from SPW 4 for the antenna *antID*

The *subScans* parameter can be either an empty string (the default), or
have one of the "on" or "off" values. If empty, entire scan is used,
otherwise, just on-source or off-source subscans are involved, respectivelly.

The four WVR channels (temperatures) are extracted as a function of time.
Return value: A tupple of two ndarrays - time (1d) as MJDsecs of dimension
nTimeSamples, WVR (2d) of dimensions 4xnTimeSamples.

3.2.2.66 wvrPhase()

wvrPhase (
vis,
antID,
spw)

Extract WVR-predicted phase delays for the PB-calibrator as a function of
time for the WVR gain table that has been created by the call

@code{sh}
wvrgcal(vis=vis,caltable='wvr_gcal.W', segsource=True, sourceflag=['Sun'],
statsource='J1256-0547',refant='DV10',maxdistm=1.0,minnumants=3)
@endcode

Basically, very similar to the getPhase() above, just we specify also the
field inside the table-selection.

The getPhase() method is called as well inside - the only purpose is to get
easilly the time-range for the BP scan and refine the selection of the
WVR-gcal table rows.

The last two args tp wvrgcal() (added after PM12 in may 2025) prevents
wvrgcal() from making caltables in antennas that do not have the WVR device
(namely 7m antennas) or have it broken (without those switches wvrgcal()
produces a 'spurious' table for non-working WVR by interpolation of the
neighborhood antennas).

Chapter 4

Class Documentation

4.1 SolSG.calibrators Class Reference

Public Member Functions

- __init__ (self, obsMsmd)
- allCalsStr (self)
- allCalsLst (self)
- printInfo (self)
- fluxCalQualityMetrics (fCalField)

Public Attributes

- str **fluxCalField** = str(fcFields[0])
- **phaseCalField** = str((obsMsmd.fieldsforintent("∗PHASE∗"))[0])
- **bpassCalField** = str((obsMsmd.fieldsforintent("∗BANDPASS∗"))[0])
- **bpassCalScan** = str((obsMsmd.scansforintent("∗BANDPASS∗"))[0])
- **ampCalField** = self.bpassCalField
- tuple **ampCalSrcName** = (obsMsmd.sourcenames())[ampCalRow]

4.1.1 Detailed Description

```
Encapsulation of calibrators.

Attributes:

fluxCalField (string): Field for intent CALIBRATE_FLUX.
phaseCalField (string): Field for intent CALIBRATE_PHASE.
bpassCalField (string): Field for intent CALIBRATE_BANDPASS.
ampCalField (string): Field to be actually used in setjy() - may be
                      bpassCalField or fluxCalField.
bpassCalScan (string): Which scan has been used for B-pass calibration.
ampCalSrcName (string): Source-name for actual flux calibrator.
```

4.1.2 Constructor & Destructor Documentation

4.1.2.1 __init__()

```
SolSG.calibrators.__init__ (
              self,
              obsMsmd)
```

Initialize the calibrator attributes

4.1.3 Member Function Documentation

4.1.3.1 allCalsLst()

```
SolSG.calibrators.allCalsLst (
              self)
```

All calibrators (a unique set) as a Python list (of strings).
To be used in a loop over applycal(field=allCals[i]).

4.1.3.2 allCalsStr()

```
SolSG.calibrators.allCalsStr (
              self)
```

All calibrators (a unique set) as a single comma-separated string.
To be used in gaincal(field=allCals)

4.1.3.3 fluxCalQualityMetrics()

```
SolSG.calibrators.fluxCalQualityMetrics (
              fCalField)
```

Returns a single float number as a measure of quality for the
flux calibrator. The quality is based on (1) flux density and,
(2) time separation between the last measurements in the ALMA calibrator
database and the date of observations.
Rationale: Quite often we have declared (primary) flux calibrator (with
intent CALIBRATE_FLUX) in solar science data which, however, is either
weak or its measurement in database is old. Hence, the band-pass
calibrator is frequently use as a primary calibrator: The function
sol_calflux2() even assumes this as a standard.
This method should calculate a score (the higher, the better) to help
to decide whcih calibrator should be used for setjy() and subsequent
fluxscale()

4.1.3.4 printInfo()

```
SolSG.calibrators.printInfo (
              self)
```

```
Print info on calibrators.
```

The documentation for this class was generated from the following file:

- SolSG.py

4.2 SolSG.observation Class Reference

Public Member Functions

- **__init__** (self, asdmName)
- rfStr (self)
- printInfo (self)

Public Attributes

- **theSun** = scienceTarget(obsMsmd)
- **theCalibrators** = calibrators(obsMsmd)
- **band** = aU.bandforspw(refSPW,mymsmd=obsMsmd)
- tuple **refFreq** = (obsMsmd.reffreq(refSPW))["m0"]
- obsDate

4.2.1 Detailed Description

```
Basic minimum data about the observation that are necessary for data
reduction & imaging. Contains the children objects scienceTarget and
calibrators plus a few further service data.

Attributes:

theSun (class scienceTarget): Encapsulated properties of the target
theCalibrators (class calibrators): Encapsulate properties of calibrators.
band (int): Observing band.
refFreq (CASA quanta): Representing (reference) observing frequency.
obsDate (string): Date of obsrevation in YYYY-MM-DD format.
```

4.2.2 Member Function Documentation

4.2.2.1 printInfo()

```
SolSG.observation.printInfo (
              self)
```

```
Prints basic info about the observation.
```

4.2.2.2 rfStr()

```
SolSG.observation.rfStr (
               self)
```

Returns the reference frequency as a string (value + unit).

4.2.3 Member Data Documentation

4.2.3.1 obsDate

```
SolSG.observation.obsDate
```

Initial value:

```
= qatool().time(self.theSun.refTime,
                                  form=["fits","no_time"])[0]
```

The documentation for this class was generated from the following file:

- SolSG.py

4.3 SolSG.scienceTarget Class Reference

Public Member Functions

- __init__ (self, obsMsmd)
- isMosaic (self)
- targetFields (self)
- printInfo (self)

Public Attributes

- **allFields** = allFieldsArray.tolist()
- **sciFields** = sciFieldsArray.tolist()
- **refTime** = qatool().quantity(sciScanTimes[0],'s')
- **refField** = fieldRef
- **fieldsDirections** = dict()

4.3.1 Detailed Description

```
Attributes & methods for the science target (the Sun).

Attributes:

allFields (list of ints): List of all target fields. May contain purelly
                  OFF_SOURCE field 0 that is attached to ephemeris and
                  serves as a reference to which the science fields in
                  a mosaic have offsets in the FIELD table.

sciFields (list of ints): Just the science fields (the intent
                          OBSERVE_TRAGET#ON_SOURCE)


refField (int): The reference field attached to ephemeris.


refTime (CASA qanta): The time when the first science field started to be
                      scanned. For exactly this time the ephemeris is
                      interpolated and RA/DEC found by this interpolation
                      are set to the reference field. It is also the same
                      date/time that is later written as DATE_OBS keyword in
                      the header of final product FITS image - can be used
                      for transformation to helioprojective X,Y.


fieldsDirections (dictionary of measures): Ra/Dec attributed to each field
                                           at the reference time.


Methods:

bool isMosaic(): Returns T if the target is a mosaic, F if single-pointing.

string targetFields(): Returns the all target fields as a CASA-string.

void printInfo(): Prints information about science target to terminal.
```

4.3.2 Constructor & Destructor Documentation

4.3.2.1 __init__()

```
SolSG.scienceTarget.__init__ (
              self,
              obsMsmd)


Constructor.
@param obsMsmd: MS meta-data.
```

4.3.3 Member Function Documentation

4.3.3.1 isMosaic()

```
SolSG.scienceTarget.isMosaic (
              self)


Returns True if the target is a mosaic, False if single-pointing.
```

4.3.3.2 printInfo()

```
SolSG.scienceTarget.printInfo (
              self)

Print info/summary about the target for the user.
```

4.3.3.3 targetFields()

```
SolSG.scienceTarget.targetFields (
              self)

Returns a string to be passed as a 'field' argument in applycal()
and tclean().

Note: We have to include the field 0 (in case of mosaic) whether it
is associated with a scicence scan or not. The 'phantom' field=0, albeit
may not even be present in the mosaic as a science target field
(MS.split.cal contains no data associated with this field), it is still
in the FIELD table and THIS is the field that is clipped to ephemeris
(even if it is purelly OBSERVE_TARGET#OFF_SOURCE intent!)
It is usually located in the mosaic center, but need not be.
```

The documentation for this class was generated from the following file:

- SolSG.py

4.4 SolSG.stuffForSolarDataReduction Class Reference

Public Member Functions

- generateSolCalScript (self, asdmName, doQA2=True)

4.4.1 Detailed Description

```
Encapsulation for Solar ALMA Data Reduction Utils
```

4.4.2 Member Function Documentation

4.4.2.1 generateSolCalScript()

```
SolSG.stuffForSolarDataReduction.generateSolCalScript (
              self,
              asdmName,
              doQA2 = True)

Generates the script for solar INT data calibration (single EB)

Implementation is very simple:
It reads the static template (the 'sceleton', which remains constant
for all solar INT EBs) and adds/replaces the lines with the EB-specific
code. Result is written into the 'standard' solar calibration script.

Inputs:
asdmName (string): Name of the ASDM to reduce
doQA2 (bool): Generate the script for the QA2 analyst (True) or for
              the archive (False).
```

The documentation for this class was generated from the following file:

- SolSG.py

Index

__init__
    SolSG.calibrators, 42
    SolSG.scienceTarget, 45

acd_caltab_plots
    sol_wvgcal, 9
acd_calTabs
    sol_wvgcal, 10
acd_calTabs2
    sol_wvgcal, 10
allCalsLst
    SolSG.calibrators, 42
allCalsStr
    SolSG.calibrators, 42

blindPhiFix
    sol_wvgcal, 11

dB_sum
    sol_wvgcal, 11
dPWV
    sol_wvgcal, 11
dPWV_spwPart
    sol_wvgcal, 12
dpwvXphi_corrs
    sol_wvgcal, 12
dpwvXphi_plots
    sol_wvgcal, 13
dpwvXsolWVR_raw
    sol_wvgcal, 13
dpwvXwvr_corrs
    sol_wvgcal, 13
dWdB
    sol_wvgcal, 14
dWdB_Tbs
    sol_wvgcal, 15
dWdB_Tbs_NL
    sol_wvgcal, 16
dWdBdG
    sol_wvgcal, 16
dWdBdG2
    sol_wvgcal, 17
dWdBdG2_weights
    sol_wvgcal, 18
dWdBdG_calWeights
    sol_wvgcal, 19
dWdLs
    sol_wvgcal, 20
dWdLs2
    sol_wvgcal, 20
dWdLs3
    sol_wvgcal, 21
dWdLs3_calWeight
    sol_wvgcal, 22
dWdLs4
    sol_wvgcal, 22
dWdLs4_weights
    sol_wvgcal, 23

edit_coordinates
    off_ephem, 6
edit_coordinates2
    off_ephem, 6
extMatInv
    sol_wvgcal, 23

fluxCalQualityMetrics
    SolSG.calibrators, 42

G_off_cal
    sol_wvgcal, 23
generateSolCalScript
    SolSG.stuffForSolarDataReduction, 46
getMeteo
    sol_wvgcal, 24
getMeteoForScan
    sol_wvgcal, 24
getPhase
    sol_wvgcal, 25
getPhaseMS
    sol_wvgcal, 25
getPWVforAnt
    sol_wvgcal, 25

isMosaic
    SolSG.scienceTarget, 45

modified_line
    off_ephem, 7

obsDate
    SolSG.observation, 44
off_ephem, 5
    edit_coordinates, 6
    edit_coordinates2, 6
    modified_line, 7

paregression
    sol_wvgcal, 26
phiFixCompare
    sol_wvgcal, 26

phiFixCompare3
sol_wvgcal, 27
plotPhases
sol_wvgcal, 28
printInfo
SolSG.calibrators, 42
SolSG.observation, 43
SolSG.scienceTarget, 45
pwv4scans
sol_wvgcal, 28
pwv4ScansCompare
sol_wvgcal, 28
pwv4scansRestore
sol_wvgcal, 28
pwvCalc
sol_wvgcal, 29
pwvCalcVerify
sol_wvgcal, 29
pwvCompare
sol_wvgcal, 29

rebin
sol_wvgcal, 30
rebin2
sol_wvgcal, 30
restoreSol_dWdB
sol_wvgcal, 30
rfStr
SolSG.observation, 43

saveSol_dWdB
sol_wvgcal, 31
saveSol_dWdB_Tbs
sol_wvgcal, 31
setAtmosSciSPWs
sol_wvgcal, 31
setAtmTool
sol_wvgcal, 32
setAtmTool4SPWs
sol_wvgcal, 32
smooth
sol_wvgcal, 33
sol_offPos
sol_wvgcal, 33
sol_PhaseFix
sol_wvgcal, 33
sol_pwv2selfCal
sol_wvgcal, 34
sol_pwvGcal
sol_wvgcal, 34
sol_wvgcal, 7
acd_caltab_plots, 9
acd_calTabs, 10
acd_calTabs2, 10
blindPhiFix, 11
dB_sum, 11
dPWV, 11
dPWV_spwPart, 12
dpwvXphi_corrs, 12
dpwvXphi_plots, 13
dpwvXsolWVR_raw, 13
dpwvXwvr_corrs, 13
dWdB, 14
dWdB_Tbs, 15
dWdB_Tbs_NL, 16
dWdBdG, 16
dWdBdG2, 17
dWdBdG2_weights, 18
dWdBdG_calWeights, 19
dWdLs, 20
dWdLs2, 20
dWdLs3, 21
dWdLs3_calWeight, 22
dWdLs4, 22
dWdLs4_weights, 23
extMatInv, 23
G_off_cal, 23
getMeteo, 24
getMeteoForScan, 24
getPhase, 25
getPhaseMS, 25
getPWVforAnt, 25
paregression, 26
phiFixCompare, 26
phiFixCompare3, 27
plotPhases, 28
pwv4scans, 28
pwv4ScansCompare, 28
pwv4scansRestore, 28
pwvCalc, 29
pwvCalcVerify, 29
pwvCompare, 29
rebin, 30
rebin2, 30
restoreSol_dWdB, 30
saveSol_dWdB, 31
saveSol_dWdB_Tbs, 31
setAtmosSciSPWs, 31
setAtmTool, 32
setAtmTool4SPWs, 32
smooth, 33
sol_offPos, 33
sol_PhaseFix, 33
sol_pwv2selfCal, 34
sol_pwvGcal, 34
solTgtPos, 34
sqld_P0, 35
TbStats, 35
tgt_pow, 35
tgt_Tb, 36
tgtPow_crossPlots, 36
unwrap2, 37
wopacity, 37
wrapBack, 38
wv2path, 38
wvr_dataRaw, 38
wvrPhase, 39

SolSG.calibrators, 41
    __init__, 42
    allCalsLst, 42
    allCalsStr, 42
    fluxCalQualityMetrics, 42
    printInfo, 42
SolSG.observation, 43
    obsDate, 44
    printInfo, 43
    rfStr, 43
SolSG.scienceTarget, 44
    __init__, 45
    isMosaic, 45
    printInfo, 45
    targetFields, 46
SolSG.stuffForSolarDataReduction, 46
    generateSolCalScript, 46
solTgtPos
    sol_wvgcal, 34
sqld_P0
    sol_wvgcal, 35

targetFields
    SolSG.scienceTarget, 46
TbStats
    sol_wvgcal, 35
tgt_pow
    sol_wvgcal, 35
tgt_Tb
    sol_wvgcal, 36
tgtPow_crossPlots
    sol_wvgcal, 36

unwrap2
    sol_wvgcal, 37

wopacity
    sol_wvgcal, 37
wrapBack
    sol_wvgcal, 38
wv2path
    sol_wvgcal, 38
wvr_dataRaw
    sol_wvgcal, 38
wvrPhase
    sol_wvgcal, 39

Annex 4

Full listings of the SW

1. sol_wvrcal.py: collection (routines for Approach 2)
2. off_ephemeris.py: Setup for sun-array test observation

```
#!/usr/bin/env python
"""
@file sol_wvgcal.py
@brief Use WV line absorption/emission in AC/sqld SPWs for phase correction.

@author barta@asu.cas.cz
@date 2026-09-28 (last update)

Experimentation with WV-line features - emission in case of the weak source
(typically calibrator) and absorption for the Sun - in the science SPWs: either
auto-correlations (ACs) of SQLD. We aim to use those features for eventual
phase-error corrections on the short time-scales, as an alternative to WVR data
and wvrgcal(), which are unavailable (saturated) in case of solar on-source
scans.

The collection of routines here is devised for the following group of tasks:


1. Multi-purpose (non-specific) helpers: rebin(), smooth(), paregression().
2. Perform dual-load/ACD calibration: acd_calTables(); apply it to MS powers
   to convert them to calibrated T_B: tgt_Tb(). Diagnostics: acd_caltab_plots().
3. Extract the raw data from MSs and calTables: tgt_pow() and getPhase().
   Diagnostics: plotPhases().
4. Setup the atmosphere (ATM) model and use it for calculation of opacities:
   setAtmTool(), wopacity().
5. Use all above + mathematical mode derived in PR11 to calculate the variations
   of PWV (proxy/model): dPWV()
6. Analysis:  Compare the dPWV proxy vs. phase variations
7. Verification: Compare our dPWV proxy for the BP-calibraror with an actual
   PWV/WVR data measured in ASDM_CALWVR (accessed via wvrgcal())

7.1 Specific for 2022.1.00031.CSV-Sun10a-05-INT SB: Analysis of unsaturated
    raw WVR data

- - - - -
Notes
=====

Ad (2): Note on comparison with sol_sqld_mean_2() in sun_reduction_utils.py.
We extract the full AC and SQLD data from the MS (using ms.select()) and make
their averaging here using numpy.mean(). The sun_reduction_utils.py uses
ms.statistics() internally in its sol_sqld_mean_2() helper. For SQLDs, the
results of np.mean() are very similar to those obtained by sol_sqld_mean_2().
The difference between our approach and that in sol_sqld_mean_2(), which uses
ms.statistics() internally, is less than 0.015% (the relative error) and looks
to be related by rounding inaccuracy (less accurate representation of the
float data in our approach). Our approach is, however, more general - in the
already mentioned sense of its applicability to both SQLD and AC (channelized
or integrated) spws, but also because it can be used for the concrete selected
antenna: sol_sqld_mean_2() accepts the antenna-names list, too, but it
MUST be a complete list of *all* antennas for the given MS, otherwise mapping
between antenna name and its ID (int number), which is internally used in the
sol_sqld_mean_2() method, gets broken.

Ad (3): Note to experimenting with Dual-Load (DL: Hot, Ambient, Sky) calibration
using ALMA ACD. We extract P_hot, P_amb, and P_sky for a given antenna and ATM
scan. For the Sun, we extract also P_off, which is basically P_sky, but with the
attenuation set to the high, solar-level, value. This is not present in the
```

```
ATM scan preceding the solar-target scan, but in the Sun-scan itself: In its
very first and very last sub-scan (see Fig. 4 in Shimojo et al., 2017).
We extract also ACD load temperatures and  - for SQLD spws - also the P_zero
offset for the scan and calculate the gain G and T_rx according to formulae
(7) and (9), and "solar" (i.e. in hi-attenuation setup) gain G* according to
Eq. (8) in White et al. (2017).

The main difference to solar flux calibration implemented in sol_ampcal2() is
that we can use here also
1. autocorrelation (AC) data and not just the SQLDs,
2. we may extract ACs in the ATM scans in SPWs with the channelized data
    in order to reconstruct the spectrally resolved  Tsky+Tsrc (128 channels).

The obtained ACD calibration (gains G and G*, T_rx and in case of SQLD
data also the offset P_zero) is subsequently applied to the target scan in
order to convert measured raw powers (AC or SQLD) into brigthness 'sky'
temperature - the sum of source (attenuated by the terrrestrial atmosphere)
and the contribution of our atmosphere.
"""
import numpy as np
import math
import os
import sys
import matplotlib.pyplot as plt
from copy import deepcopy

# Where (in which Tsys SPW) we can find ATM/ACD data for our science SPW?
from casarecipes.almahelpers import tsysspwmap


#-----------------------------------------------------------------------------
#     1. 'Universal' helpers - general-purpose auxilliary functions
#-----------------------------------------------------------------------------

def rebin(array1D,downSampleFactor):
  """
  This is a general-purpose helper analogical to the IDL's routine rebin().
  It down-samples 1D-array *array1D* by the *downSampleFactor*. The dimension
  of the array1D must be evenly divisible by the downSampleFactor! If not,
  the original (non-resampled) array is returned and a warning issued. In such
  a case, it is advisable to use more complex rebin2() method.

  Rationale for utilization here: We are frequently trying to find correlations
  between quantities like phase variations and our model PWV-proxy, and these
  need not to have the same sampling rate (typically our PWV-proxy from
  auto-correlations has cadence 0.192s; from the SQLD data yet shorter, by an
  integer factor of 12, while the BP-calibrator phases are solved at interval
  of 1.92 sec).
  """

  origDim=array1D.size

  # Check, whether we can do the down-sampling. Return original array if not
  # and print warning.
  if (origDim % downSampleFactor):
    print("rebin(): Array dimension is not divisible by the re-bin factor!")
    return array1D
```

```
  # Down-sampled size of the array.
  newDim=origDim//downSampleFactor

  # Make an artificial 2nd axis inside the new broader bin.
  array2D=array1D.reshape((newDim,downSampleFactor))

  # Now 'contract' this 2nd axis by averaging operation.
  array1DResampled=array2D.mean(axis=1,dtype=np.float64)


  return array1DResampled

#-----------------------------------------------------------------------------

def rebin2(array1D,origX,downSampledX):
  """
  The rebin() method generalized for the case that down-sampled array elements
  cannot be simply calculated as averages over *downSampleFactor* elements
  of the original array, because the dimensions do not agree.

  We submit three argument: *array1D* are the 1D data (e.g., recorded powers
  in our SQLD spw), *origX* is the x-grid of the original data (i.e., the times
  at which the SQLD powers are measured in the above example), and the
  *downSampledX* is the new, downsampled x-grid (in the used example it is
  typically the lower-cadence time-axis for the corresponding AC resolved spw).

  We first define a new high-cadence x-axis with a gridding that is (i) as much
  close to the original x-grid as possible, and, at the same moment (ii) the
  new grid represents an even division of the low-cadence grid. Then we
  can already call our standard rebin() method.

  Rationale: Like with rebin(). This more generalised routine has been
  introduced namely because of the non-comensurable (no even division) time
  meshes for SQLD and the AC spws in some MSs.
  """

  #-- Find the (equidistant) gridding.

  # Original-array grid.
  oSize=origX.size
  oDx=(origX[oSize-1]-origX[0])/oSize

  # Resampled-array grid.
  rSize=downSampledX.size
  rDx=(downSampledX[rSize-1]-downSampledX[0])/rSize

  #-- Create a new grid, evenly-divisible to the down-sampled mesh.

  # Down-sampling factor, as much close to the original cadence-ratio.
  dsFactor=round(rDx/oDx)

  # A new cell-size.
  nDx=rDx/dsFactor

  # A new grid/x-axis.
  nxGrid=nDx*np.arange(dsFactor*rSize)+downSampledX[0]
```

```
  #-- Interpolate the data to the new grid.
  nArray1D=np.interp(nxGrid,origX,array1D)


  #-- Now we can call rebin().
  return rebin(nArray1D,dsFactor)


#-------------------------------------------------------------------------


def smooth(array1D,kernel):
  """
  A simple smoothing of *array1D* performed by a convolution with the
  much shorter 1-D array *kernel*. Kernel may be a box or at riangle function or
  whatever similar with the finite (and rarther short, in comparison with
  the array1D length) support.

  In order to handle the edge effects, we apply 'mirror boundary condition' to
  array1D prior smoothing - we extend the original array at both edges prior
  the convolution. This prevents wild behaviour at the edges (so the
  edges do not affect, e.g., plotting y-range), but the edge data carry
  inherently lower/degraded information and must be take cautiously.
  """

  #-- Prepare the kernel and array for a convolution.

  # Normalize the smoothing kernel to 1.0
  kNorm=1.0/kernel.sum()
  kernel*=kNorm

  # Find the kernel 'half-width'.
  khw=kernel.shape[0]//2

  # Array last index.
  aLast=array1D.shape[0]-1

  # Form the mirrorred 'ghost' edges.
  edgeLeft=np.flip(array1D[1:1+khw])
  edgeRight=np.flip(array1D[aLast-khw:aLast])

  # Prepend/append them to the array1D.
  array1D_ext=np.append(edgeLeft,array1D)
  array1D_ext=np.append(array1D_ext,edgeRight)

  #-- Convolve, preserving the original array shape.

  aSmooth=np.convolve(array1D_ext,kernel,'valid')

  return aSmooth


#-------------------------------------------------------------------------

def paregression(arrX,arrY):
  """
  Generalized/parametrized linear regression calculation between the arrays
  arrX and arrY. In addition to the standard lin. regression (handled, e.g.,
```

```
by np.polyfit()), here the values y_i of the arrY may be dependent not only
on x_i, but also on the index i itself.
Mathematically, we are finding the best fit of the data in the form

y_i-t*i=K*x_i+Q


Rationale: If both arrays X and Y are time series (with regular cadence),
the Y values may be dependent not only at X, but also on time (represented
by the index i). Here we expect that there is - in addition to linear
dependence on X - also a linear trend of Y-array in time. The goal is to
optimize the parameter 't' of linear trend in time such a way, that we obtain
the best linear correlation between the X-array and time-detrended Y-array.

Math behind the calculations here is described in PR12 (May 2025). Application
in the context of this research is for correlation between phases and PWV
variations as calculated in dwXdpi_corrs() below.

Input: arrX, arrY - 1D float np arrays (time series of the same size).

Returns: A dictionary with the values of

tTrend: Time-trend parameter.
K: Coefficient of linear correlation between X and time-detrended Y
Q: Offset for linear correlation between X and time-detrended Y.
Xi2: Xi-square - how good is the fit.
"""

 # Size of the time series
 N=arrX.size

 # Check that the series are compatible (of the same size).
 if (arrY.size != N):
   print("Sizes of input arrays must be equal!")
   return -1

 # Create the 'time' array - a vector of indices starting from 1.
 iVec=np.arange(N)+1

 #-- Evaluate quantities that repeat in further calculations.

 xMean=arrX.mean()
 yMean=arrY.mean()

 ix=iVec*arrX
 ixSum=ix.sum()

 iy=iVec*arrY
 iySum=iy.sum()

 xx=arrX*arrX
 xxSum=xx.sum()

 xy=arrX*arrY
 xySum=xy.sum()

 #-- Evaluate the 2x2 linear system matrix for 't' and 'K'
```

```
  A11=xxSum-N*xMean*xMean
  A12=ixSum-0.5*N*(N+1.0)*xMean

  A21=A12
  A22=(N-1.0)*N*(N+1.0)/12.0

  # Right-hand side of that lin. system.
  C1=xySum-N*xMean*yMean
  C2=iySum-0.5*N*(N+1.0)*yMean

  #-- Solve the system by Cramer's rule.

  iD=1.0/(A11*A22-A12*A21) # Inverse determinant of the system.

  K=iD*(C1*A22-C2*A12)
  tTrend=iD*(C2*A11-C1*A21)

  # Find the 3rd parameter, expresed analytically in PR12.
  Q=yMean-K*xMean-tTrend*0.5*(N+1.0)


  #-- How good is the fit? Calculate Xi-square.

  errs=arrY-tTrend*iVec-K*arrX-Q
  Xi2=(errs*errs).sum()


  #-- Form the result as a dictionary/map and return it.
  rCoeffs={'tTrend':tTrend, 'K':K, 'Q':Q ,'Xi2':Xi2}

  return rCoeffs

#-------------------------------------------------------------------------------

def unwrap2(phase):
  """
  Sometimes we have phase jumps exactly by 180 deg. These seem to be artificial
  but not cured by the standard np.unwrap(). This method aims to remove also
  these residuals that are well less the 2PI (which are cured by np.unwrap()).

  Input: phase in rads - 1-D numpy ndarray
  Output: phase_unwrap in rads - 1-D numpy ndarray with the same dimensions.

  How does it work: We differentiate the original array, fix the high jumps
  (positive and negative extra) in the array of differences and then
  're-integrate' the corrected difference series back via the cummulative sum.
  In order to skip seekeing and evaluating the 'super-high' jumps (2PI and more)
  we call first the standard np.unwrap() prior to our residual-jump treatment.
  """

  # The difference array, with 0.0. prepended, it contains phase[0] in its
  # 1st element. Usefull when reverting back using np.cumsum()
  phiDiff=np.diff(np.unwrap(phase),prepend=0.0)

  # Identify and fix the positive jumps. Threshold 2.5 is approx. 80% of PI.
  posJumps=np.where(phiDiff > 2.5)
  phiDiff[posJumps]-=math.pi
```

```
  # Fix the negative jumps the same way.
  negJumps=np.where(phiDiff < -2.5)
  phiDiff[negJumps]+=math.pi

  # Return the array re-integrated back.
  return np.cumsum(phiDiff)

#-----------------------------------------------------------------------

def wrapBack(phases):
  '''
  An 'inverse' function to np.unwrap()
  '''
  wPhases = np.arctan2(np.sin(phases), np.cos(phases))

  return wPhases

#-----------------------------------------------------------------------

def extMatInv(Ainv,Lb,Rb,LRc):
  """
  If we know inversion *Ainv* of a square regular matrix AA, we can
  efficiently calculate inversion the matrix AA extended by a lower belt *Lb*,
  right belt *Rb*, and a lower-right corner *LRc*, i.e. of a compound matrix
  M = [[AA,Rb],[Lb,LRc]]

  This is used in dWdBdG() method below, where a large, block-tridagonal matrix
  (whose inversion is easy to compute) is extended by the (relativelly narrow)
  belt-matrices at its right and low ends connected with the instrumental
  variables.

  Returns: M^(-1), i.e. the square matrix (2-d ndarray) that is inversion
  of the extended matrix M = [[AA,Rb],[Lb,LRc]].
  """

  # Pre-calculate frequently repeating terms Ainv x Rb and Lb x Ainv
  AinvRb=np.matmul(Ainv,Rb)
  LbAinv=np.matmul(Lb,Ainv)

  # Calculate the Schur complement and its inversion.
  S=LRc-np.matmul(Lb,AinvRb)
  Sinv=np.linalg.inv(S)

  #- Calculate the blocks of the inverted matrix.

  # Start with the lower-left block as it repeats in upper-left block, too.
  Minv21=-np.matmul(Sinv,LbAinv)

  # Upper-left block.
  Minv11=Ainv-np.matmul(AinvRb,Minv21)

  # Upper-right block.
  Minv12=-np.matmul(AinvRb,Sinv)

  # Lower-right block.
  Minv22=Sinv

  #- Compose the Minv matrix from its blocks
```

```
  # Compose the rows.
  Minv1=np.hstack([Minv11,Minv12])
  Minv2=np.hstack([Minv21,Minv22])

  # Now the rows together
  Minv=np.vstack([Minv1,Minv2])

  return Minv


#-------------------------------------------------------------------------
#                 2. ACD/dual-load (DL) calibration routines
#-------------------------------------------------------------------------

def sqld_P0(vis,ant,spw,myMS=None,myMsmd=None):
  """
  Find the P-zero offset for SQLD detector at antenna *ant*.
  The *spw* parameter designates the spw for *standard* SQLD ATM scans [0,1,2,3]
  and maps to the spws for the scan no. 1 (the 'zero' scan) [13,14,15,16]
  in an obvious way [0,1,2,3] -> [13,14,15,16].

  Returns: An array of shape 2 x 1, the first dimension being polarization.

  See also acd_calTabs().
  """

  # P zero scan. Has to be looked in listobs.txt
  PoScan=1

  # ms and msmd tools flags.
  ownMS=False
  ownMsmd=False

  # Open own tools if 'global' not provided
  if (not myMS):
    # Open the ms tool
    myMS=mstool()
    myMS.open(vis)
    ownMS=True

  if (not myMsmd):
    # Open the ms tool.
    myMsmd=msmdtool()
    myMsmd.open(vis)
    ownMsmd=True


  # Find the data_desc_id for a given SPW.
  # Perform the [0,1,2,3] -> [13,14,15,16] mapping.
  dataDescID=int(myMsmd.datadescids(spw=spw+13)[0])

  # Get sub-scan info for the zero-scan (scan no.1).
  sub_info=aU.computeDurationOfScan(PoScan,mymsmd=myMsmd,
                                    returnSubscanTimes=True)

  # The sub_info indicates 4 subscans. However, the only non-zero selection
  # is for the sub-scan no. 2 - the others say
```

```
  # RuntimeError: MSSelectionNullSelection : The selected table has zero rows.
  # Hence, select just the sub-scan no. 2 by its time-span.
  timeRange=sub_info[2][2]

  # Init pre-selection by the SPW id.
  myMS.selectinit(datadescid=dataDescID)

  # Select 'autocorrelations' for a given antenna, the zero-scan number,
  # and the sub-scan defined by its time-range.
  myMS.select({'antenna1':ant,'antenna2':ant,'scan_number':PoScan,
               'time':timeRange})

  # Extract the data from the reduced/filtered MS.
  acdPow=myMS.getdata("data")["data"]

  # The above array has dimensions 2 x nChans x nIntegs, where nIntegs is
  # a number of integrations in the subscan (20 for sci spws, 240 for SQLDs).
  # The 1st dim represents polarization, the 2nd is the freq.-channel number.
  # The array is complex but its imaginary parts are zeroes (experimentally
  # verified).
  # We now take just the real part and do averaging over integrations.
  acdPowAvg=acdPow.real.mean(axis=2,dtype=np.float64)

  #- Clean-up the tools.

  # Reset always - it resets the selection criteria.
  myMS.reset()

  # Close only if we use 'local' tools.
  if (ownMsmd): myMsmd.done()
  if (ownMS): myMS.close()

  # Return the time-averaged P_zero for XX and YY as the 2 x 1 array.
  return acdPowAvg


#-----------------------------------------------------------------------------

def acd_calTabs(vis,atmScan,ant,spw,tgtScan=None,myMS=None,myMsmd=None,
                acdLoads=None):
  """
  Calculates the (inverse) gains (G and, possibly, G* in White et al. 2017),
  and T_rx for a given *ant* and *spw* from the ATM scan *atmScan* using its
  three sub-scans for Sky, Ambient, and Hot loads.

  The *ant*, *spw*, and *atmScan* parameters MUST be set as *int* (not string),
  otherwise ms.select(), used internally, returns completely wrong selection -
  silently, without any warning!
  The 'global' tools *myMS* and *myMsmd* may be conveniently supplied, attached
  to the MS *vis*. If not, local tools are created and destryed with the
  routine, with some additional cost on CPU time. The same holds for the
  global dictionary *acdLoads* that is obtained by call to aU.getAntennaLoads().

  If the *atmScan* precedes directly the solar scan, we calculate G*, too.
  For the G*, we use (in addition to the three ATM subscans) also the
  'sky'/off-source sub-scans (the very first and last) in the
  immediatelly-following solar scan.
```

```
    Returns: A dictionary with the inverse gain G (Eq. 7 in White et al.), inverse
    'solar' gain G* (Eq. 8), T_rx (Eq. 9), and effective total 'instrumental'
    temperature T_inst=T_rx+Po/G. The G and T_rx components are
    numpy arrays of shape nPols x nChans, i.e., 2 x 128, if we use the science
    SPWs (5,7,9,11) and 2 x 1 for channel-integrated SPWs (6,8,10,12) or
    SQLDs (0,1,2,3). The first dimension represents polarization and the second
    the channel number.

    For G*, we provide its two values - at the very beginning and very end of the
    solar scan (the very 1st and very last sub-scans in each solar scan are the
    off-source subscans). Hence, the shape of the returned array is
    2 x nPol x nChans, the first dim =2 representing the beginning and end values.
    According to Shimojo et al. (2017), the G* should be calculated from the SQLD
    spws solely (it is said that P_off is too low for AC spws). Consequently, it
    should be an array of the shape 2 x 2 x 1. However, we are experimenting here
    with the auto-corellations (AC) SPWs, too, both for spectral-resolved and
    channel-integrated spws, so the return shape may be well also 2 x 2 x 128.
    The reason is roooted in the fact that the attenuation factor G*/G found in
    the Band 5 data is just about 1.07.In case of the ATM scan precedes non-solar
    scan (e.g., BP-cal scan), we return [[[1.0],[1.0]],[[1.0],[1.0]]] - i.e.
    factor of 1,with the shape corresponding to the  channel-integrated SPWs.
    Nevertheless: It has been found  - see detailed comment in tgt_Tb(), in the
    section starting with 'if(doSolarGain)' - that PWV changing during the solar
    scan affects measured P_off and consequently the attenuation factor and
    resulting gain G*. Sinze the attenuation factor AF=(P_sky-Po)/(P_off-Po),
    it is bettre to take just the leading-subscan P_off measurement, because
    it follows immediatelly after the P_sky measutement in preceding ATM scan.
    """


    #- Global tools & data

    # ms and msmd tools flags, checked at closing
    ownMS=False
    ownMsmd=False

    # Open own tools if 'global' not provided
    if (not myMS):
      # Open the ms tool.
      myMS=mstool()
      myMS.open(vis)
      ownMS=True

    if (not myMsmd):
      # Open the ms tool.
      myMsmd=msmdtool()
      myMsmd.open(vis)
      ownMsmd=True

    # Create our dictionary with ACD load temperatures (ambient, hot) if needed.
    if (not acdLoads):
      acdLoads=aU.getLoadTemperatures(vis,doplot = False,
                                      warnIfNoLoadTemperatures = True,
                                      mymsmd=myMsmd)

    #-- Find the data_desc_ids for the given sciSPW and its Tsys: Data and Tsys
    #   may have different SPWs.
```

```
# Description ID for the science data SPW.
sciDataDescID=int(myMsmd.datadescids(spw=spw)[0])

# The CALIBRATE_ATMOSPHERE scan (our *atmScan*) used for the ACD has data
# present only in some SPWs. Check, whether ours is among them.

acdSpws=myMsmd.spwsforscan(atmScan)

if (spw in acdSpws):
  # OK, we have atm/tsys data in the sciSPW
  tsysSPW=spw

  print('ACD calibration: Using spw ',spw, 'for ACD (Tsys) data.')

else:
  # We do not have atm measurement, must use a mapped SPW
  tsysmap = tsysspwmap(vis, tsystable = vis+'.tsys', tsysChanTol = 1)

  # Find correct SPW for the atmScan correcponding to *spw*.
  # Note: tsysspwmap() above maps the channel-integrated spws to their
  # resolved counterparts although we have Tsys/atm for the integrated spws,
  # too. Fix this by adding 1 to integrated spws: Chan-integrated SWPs will
  # be mapped to chan-integrated Tsys SPWs
  tsysSPW=tsysmap[spw]-spw%2+1

  print('ACD calibration: Mapping spws for obtaining the ACD (Tsys) data: ',
        spw,' --> ',tsysSPW)


# Description ID for the corresponding Tsys/ATM SPW.
tsysDescID=int(myMsmd.datadescids(spw=tsysSPW)[0])

#--- Extract the ACD powers for sky, ambient and hot loads for Tsys spws

# Get sub-scan info for our (atm) scan.
sub_info=aU.computeDurationOfScan(atmScan,mymsmd=myMsmd,
                                  returnSubscanTimes=True)

# An empty dictionary for calibration powers.
PowCals={}
# Map the sub-scan numbers to names: The keys to our dictionary.
# Note: P_src need not to be present. It has been introduced at ~2023 in order
# to cope with the 'renormalization issue' (emission seen in autocorrs,
# typically in strong lines; the Sun has strong autocorr signal elsewhere,
# that is why we use T_ant calculations - the P_src sub-scan is irrelevant
# for the solar observations).
ssNames={1:'P_sky',2:'P_amb',3:'P_hot',4:'P_src'}

#- Iterate the atm scan sub-scans.
for key,value in sub_info[2].items():

  # Initialize the MS selector by the Tsys SPW.
  myMS.selectinit(datadescid=tsysDescID)

  # The sub-scan is defined by its time range. Here as [MJDs_start, MJDs_stop]
  timeRange=value

  # Select autocorrelations for a given antenna, the scan number,
```

```
  # and the sub-scan defined by its time-range.
  myMS.select({'antenna1':ant,'antenna2':ant,'scan_number':atmScan,
               'time':timeRange})

  # Extract the data from the reduced/filtered MS.
  acdPow=myMS.getdata("data")["data"]

  # The above array has dimensions nPols x nChans x nIntegs, where nIntegs is
  # a number of integrations in the subscan.
  # Do time average over integrations.
  acdPowAvg=acdPow.real.mean(axis=2,dtype=np.float64)

  # Append the result to a dictionary.
  PowCals[ssNames[key]]=acdPowAvg

  # Reset all so-far applied selectors/filters before the next round.
  myMS.reset()

#--> endfor(subScans) in ATM scan.

#-- Get the Tsys SPW definition (frequency channels in GHz). This defines
#   also the channels, for which we have the gain and Trx.
freqs=myMsmd.chanfreqs(tsysSPW)*1.0e-9 # in GHz


#--- Check our SPW type. For SQLDs we need also the power offset Po.

# Get the power offset Po. It is non-zero just for the SQLD spws.
# Set the isSQLD flag as well, it will be used later.
if spw in [0,1,2,3,13,14,15,16,25,26,27,28]:
  isSQLD=True
  Pzero=sqld_P0(vis,ant,spw,myMS,myMsmd)
else:
  Pzero=np.array([[0.0],[0.0]])
  isSQLD=False

# Add the value to our collectuon of calibration powers.
PowCals['P_zero']=Pzero

#--- If we solve gains for the solar scan, extract also P_off necessary
#    for the G* calculation.

# If target scan is not specified, use the one following after ATM scan
if (not tgtScan): tgtScan=atmScan+1

# All Sun scans.
solScans=myMsmd.scansforintent('OBSERVE_TARGET#ON_SOURCE')

# Do the solar off-scan and G* calculation only if the Sun scan follows.
doSolarGain=(tgtScan in solScans)

if (doSolarGain):

  print("Solving for solar gain G* ...")


  #-- Extract the P_off values. Do it even if we are in the AC SPW and these
  #   values are suspicious - in this case we fix it later by SQLD transfer.
```

```
# Extract sub-scan information for the solar scan.
sunSub_info=aU.computeDurationOfScan(tgtScan,mymsmd=myMsmd,
                                     returnSubscanTimes=True)

# The off-source (i.e., hi-attenuated sky/atm) sub-scans are the first
# and the last. So, we need to find the number of sub-scans. It is the 2nd
# item of the info - see computeDurationOfScan() in sun_reduction_util.py.
nSubScans=sunSub_info[1]

# The off-source scans. Note: Keys in subInfo start with 1.
offSubScan1=1
offSubScan2=nSubScans

# Extract their time-ranges - we select the data later by them.
tRange1=sunSub_info[2][offSubScan1] # Time-range for the leading subscan.
tRange2=sunSub_info[2][offSubScan2] # Time-range for the trailing subscan.

#- Process the leading off-source sub-scan

# Initialize the MS selector by the science SPW.
myMS.selectinit(datadescid=sciDataDescID)

# Select autocorrelations for a given antenna, the scan number,
# and the sub-scan defined by its time-range.
myMS.select({'antenna1':ant,'antenna2':ant,'scan_number':tgtScan,
            'time':tRange1})

# Extract the data from the reduced/filtered MS.
sol_off_pow=myMS.getdata("data")["data"]


# The above array has dimensions nPols x nChans x nIntegs, where nIntegs is
# a number of integrations in the subscan.
# Do time average over integratuons of the subscan.
P_off_lead=sol_off_pow.real.mean(axis=2,dtype=np.float64)

# Reset the selection.
myMS.reset()

#- Process the trailing off-source sub-scan

# Initialize the MS selector by the scioence SPW.
myMS.selectinit(datadescid=sciDataDescID)

# Select autocorrelations for a given antenna, the scan number,
# and the sub-scan defined by its time-range.
myMS.select({'antenna1':ant,'antenna2':ant,'scan_number':tgtScan,
            'time':tRange2})

# Extract the data from the reduced/filtered MS.
sol_off_pow=myMS.getdata("data")["data"]

P_off_trail=sol_off_pow.real.mean(axis=2,dtype=np.float64)

# Reset the selection.
myMS.reset()
```

```
  # Combine the leading and trailing value into a single numpy array.
  # The 1st dimension is now time: 0=leading, 1=trailing.
  P_off=np.array([P_off_lead, P_off_trail])

  # Add the value to our collection of calibration powers for completion.
  PowCals['P_off']=P_off

  # P_off has been calculated from solar (target-scan) data, these may have
  # different frequency resolution than the Tsys used for basic ACD calibs:
  # Get the solar SPW definition (frequency channels in GHz).
  sunFreqs=myMsmd.chanfreqs(spw)*1.0e-9 # in GHz

#--> endif(doSolarGain)


#--- Extract the corresponding load temperatures.

# We select just the values for our ATM scan and antenna.
# For the Sun we scale to kiloKelvins!!!
T_amb = 0.001*acdLoads[ant][atmScan]['amb']
T_hot = 0.001*acdLoads[ant][atmScan]['hot']

#--- Calculate gains G & G*, and T_rx


# Calculate the gain (Eq. 7 in White et al. 2017).
# In fact, we wish to have 1/G, because we shall later calculate T_sky+src
# from the measured power P_sky+src.

# Inverse power-difference.
iPowDiff=1.0/(PowCals['P_hot']-PowCals['P_amb'])

# The (inverse) gain: 1/G
iGain=(T_hot-T_amb)*iPowDiff
# Calculate T_rx (Eq. 9 in White et al. 2017).
T_rx=(T_hot*(PowCals['P_amb']-Pzero)-T_amb*(PowCals['P_hot']-Pzero))*iPowDiff
# Calculate effective instrumental T_inst combining T_rx and G-scaled Pzero.
T_inst=T_rx+Pzero*iGain

#-- The (inverse) solar gain: 1/G* (Eq. 8 in White et al. 2017)
if (doSolarGain):

  # Actual calculation, the result is of the shape
  # 2(lead/trail) x 2(XX/YY) x nChans(=1 for SQLDs)

  # Numerators of the fraction in Eq. 8
  iGStarXX=iGain[0]*(PowCals['P_sky'][0]-Pzero[0])
  iGStarYY=iGain[1]*(PowCals['P_sky'][1]-Pzero[1])

  # We shall divide this by numerator (P_off-Pzero), which has channel
  # structure as the solar-target scan SPW. This may differ from
  # the Tsys spw (check tsysmap). In order to solve this mixed-channel issue
  # (gain and Trx at Tsys spw, P_off at solar spw), we interpolate everything
  # to the solar spw. Must be done per polarization, as np.interp() works for
  # 1-d arrays only.
  if (tsysSPW!=spw):

    print('--- SPW mapping: Interpolating everything to solar SPW channels!')
```

```
      # ------- !!! Attention !!! -------
      #
      # @todo This needs to be reconsiderred because we have troubles when
      # interpolating the LSB spws (5 and 7), which have decreasing order
      # of frequencies. The np.interp() requires the *freqs* array to be
      # sorted in ascending order. Suggested solution: Flip the arrays,
      # make inerpolation, and flip the arrays back in order to have them
      # sorted in 'natural' order as they come from correlator. The 2023 solar
      # data are not affected, they use no spw mapping and need no interpolation
      # but for general handling this should be sorted out.

      # Numerators of the fraction in Eq. 8 to be intrepolated
      iGStarXX=np.interp(sunFreqs,freqs,iGStarXX)
      iGStarYY=np.interp(sunFreqs,freqs,iGStarYY)

      # Let us interpolate the iG and T_rx, too.
      iGain[0]=np.interp(sunFreqs,freqs,iGain[0])
      iGain[1]=np.interp(sunFreqs,freqs,iGain[1])

      T_rx[0]=np.interp(sunFreqs,freqs,T_rx[0])
      T_rx[1]=np.interp(sunFreqs,freqs,T_rx[1])

      # Replace the spw channels definition for entire caltab
      freqs=sunFreqs

    #-->endif(tsys spw mapping)

    # Denominator (P_off-Pzero) is already at sun-spw freq. resolution.
    # This is our G* as inferred from P_off subscans.
    iGainStar=np.array([[iGStarXX,iGStarYY],[iGStarXX,iGStarYY]])/(P_off-Pzero)

    # Fix effective instrumental T_inst: Combine T_rx and G_star-scaled Pzero.
    '''
   # Use time-average for the G-star.
   T_inst[0]=T_rx[0]+0.5*Pzero[0]*(iGainStar[0,0]+iGainStar[1,0])
   T_inst[1]=T_rx[1]+0.5*Pzero[1]*(iGainStar[0,1]+iGainStar[1,1])
    '''
    # Use the leading P_off beacuse it is closer to P_sky - see the comment
    # inside tgt_Tb(), after 'if(doSolarGain)'.
    T_inst[0]=T_rx[0]+Pzero[0]*(iGainStar[0,0])
    T_inst[1]=T_rx[1]+Pzero[1]*(iGainStar[0,1])

  else:
    # Return a neutral factor of 1.0, just organised in the expected shape.
    iGainStar=np.array([[[1.0],[1.0]],[[1.0],[1.0]]])


# Data are extracted, we can close the tools (if local).
if (ownMsmd): myMsmd.done()
if (ownMS): myMS.close()


# Return inverse gains iG/iG*, T_rx, and T_inst for each freqs channel in
# a dictionary. T_inst differs from T_rx only for SQLD SPWs, which have
# Pzero offset in power.
return {'freqs':freqs,'iG':iGain, 'iG*':iGainStar, 'T_rx':T_rx,
        'T_inst':T_inst,'calPows':PowCals}
```

```
#-----------------------------------------------------------------------------

def G_off_cal(vis,atmScan,tgtScan,ant,acSpw,myMS=None,myMsmd=None):
  """
  An experiment, how to transfer relative gain-drop between G* and G in the
  *sky* and *off* scans (due to switching the increased attenuation level on)
  from SQLD to the corresponding (occupying the same frequency domain) AC SPWs.
  Findings reached by experimentation with this method are then used in improved
  ACD calibration method for the Sun acd_calTabs2().

  In theory, the AC SPWs have low dynamic range for being able to
  measure P_off in the highly-attenuated off-Sun position (Shimojo et al. 2017,
  p. 10, right above Eq. (6)). Here, by means of linear proportionality
  between SQLD and AC powers (P_sqld = K*P_ac+Q) for the same frequency domain,
  where the parameter K depends on the ratio G*_sqld/G*_ac, we evaluate the
  the derived G*_ac and compare it with its (suspicious) value obtained by
  measuring P_off directly in the AC spws. The scaling factor between the
  sqld-to-ac transfer derived G* and the AC-P_off calculation is returned.

  For the spectral-resolved AC spws we assume that the spectral gain normalized
  to unity remains constant during switching to the higher attenuation level,
  i.e., that only its scale is changing. Here we apply as much as exact
  calculation, which takes into account frequency dependence not only of this
  normalised gain, but also absorption/emission of the terrestrial atmosphere,
  and the normalized spectrum of the solar incident radiation. These factors
  enter into calculations of P_ac averaged over the resolved SPW as wieghts in
  the sums over the SPW channels. It has been experimentaly found, that the
  influence of solar spetral slope is negligible in comparison with the other
  two channel-dependent factors (normalised spectral gain and transmissivity).
  Hence, based on this research it is neglected in the improved acd_calTabs2()
  routine.
  """

  # Mapping between SPWs of the same frequency domain: AC -> SQLD
  spwMap={5:0,6:0,
          7:1,8:1,
          9:2,10:2,
          11:3,12:3}

  sqldSpw=spwMap[acSpw]

  # For non-resolved SPWs we have a single channel, for resolved integrate
  # over a range with the edges trimmed.
  if (acSpw in [6,8,10,12]):
    fChans=0
    resolved=False
  else:
    fChans=range(8,120)
    resolved=True

  # Powers for the on-source subscans: AC and SQLD
  acData=tgt_pow(vis,tgtScan,ant,acSpw,myMS=myMS,myMsmd=myMsmd,subScans='on')
  sqldData=tgt_pow(vis,tgtScan,ant,sqldSpw,myMS=myMS,myMsmd=myMsmd,
                   subScans='on')

  # SQLD powers rebined to te same cadence as ACs.
  sqPowXX=rebin(sqldData[1][0,0],12)
```

```
sqPowYY=rebin(sqldData[1][1,0],12)

# ACD calibration table: AC
acCtb=acd_calTabs(vis,atmScan,ant,acSpw,myMS=myMS,myMsmd=myMsmd)
sqldCtb=acd_calTabs(vis,atmScan,ant,sqldSpw,myMS=myMS,myMsmd=myMsmd)


# G* as per P_off in AC spw (in question))
acGxx=0.5/acCtb['iG*'][0,0,fChans]+0.5/acCtb['iG*'][1,0,fChans]
acGyy=0.5/acCtb['iG*'][0,1,fChans]+0.5/acCtb['iG*'][1,1,fChans]

# Trx
acTRxx=acCtb['T_rx'][0,fChans]
acTRyy=acCtb['T_rx'][1,fChans]

# ACD calibration table: SQLD

# G* in SQLD spw (trusted)
sqGxx=0.5/sqldCtb['iG*'][0,0,0]+0.5/sqldCtb['iG*'][1,0,0]
sqGyy=0.5/sqldCtb['iG*'][0,1,0]+0.5/sqldCtb['iG*'][1,1,0]

sqTRxx=sqldCtb['T_rx'][0,0]
sqTRyy=sqldCtb['T_rx'][1,0]

sqPoXX=sqldCtb['calPows']['P_zero'][0,0]
sqPoYY=sqldCtb['calPows']['P_zero'][1,0]


if (not resolved):
  #-- Intregrated AC SPWs are easy

  # AC powers.
  acPowXX=acData[1][0,fChans]
  acPowYY=acData[1][1,fChans]

  # Spectral correction factors. Trivial here, we have no spectral dependency
  # within the integrated AC SPW.
  SpCxx=1.0
  SpCyy=1.0

  # Offset Q: Assumed SQLD power at the zero AC power in the correlation
  # P_sqld=K*P_ac+Q. Essentially Po, but may be modified if Trx differs
  # between the AC and corresponding SQLD spw.
  P00xx=sqGxx*(sqTRxx-acTRxx)+sqPoXX
  P00yy=sqGyy*(sqTRyy-acTRyy)+sqPoYY

else:
  #-- Resolved AC SPWs need much more care - we take into account spectral
  #   weighting by normalized spectral gain, atmospheric tranmsmissivity
  #   (+ atm. radiation), and solar (QSR) spectrum.

  # AC powers averaged over channels.
  acPowXX=acData[1][0,fChans].mean(axis=0)
  acPowYY=acData[1][1,fChans].mean(axis=0)


  #- Spectral gain
```

```
# Low-attenuation (P_sky) gains in the AC SPW (limited to fChans range).
laGxx=1.0/acCtb['iG'][0,fChans]
laGyy=1.0/acCtb['iG'][1,fChans]

# Spectral *P_sky* gains normalised to 1.0
# We assume these remain untouched by the attenuation level switching,
# i.e., they are the same also for the *P_off* sub-scans.
gXX=laGxx/laGxx.mean()
gYY=laGyy/laGyy.mean()

#- Atmospheric transmissivity, atm. radiation, and instrumental effects.

# Calculate the ATM model for our actual PWV as per SYSCAL table.
pwv=getPWVforAnt(vis,tgtScan,ant,myMsmd)['pwvAvg']
atmParams=wopacity(vis,acSpw,tgtScan,userPWV=pwv)

# Estimated atmospheric (spectral) transmissivity for the acSpw.
# Limit the array to our fChans range.
tr=atmParams['tr'][fChans]

# Relative spectral transmissivity normalized to unity.
trn=tr/tr.mean()

# Atmospheric (spectral) effective BB brightness temperature within fChans.
T_atm=atmParams['T_atm'][fChans]
TeBB=T_atm*(1.0-tr)

# Rear-spillover + ohmic-loss antenna efficiency.
eta_l=0.95
# Forward-spillover antenna efficiency.
eta_F=0.90

# Ambient-load temperature for rear-spillover estimation.
TLoad = aU.getLoadTemperatures(vis, doplot = False,
                               warnIfNoLoadTemperatures = True,
                               mymsmd=myMsmd)
Tamb=TLoad[ant][atmScan]['amb']


#- Solar normalised spectrum

# Reference T_b for solar disc center @198GHz (see solar SD cal script).
TsunCenter=6300.0  # K
# Solar radius
Rsun=696000 # km

# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.

# Apparent angular solar radius in arcsec.
```

```
r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

# mu=cos(theta)
mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

b_2=-1127.0 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

# Brightness temperature @f_LO (198GHz) corrected to limb brightenning.
Tsun00=TsunCenter+0.5*b_2*np.log(mu)

# LO frequency - the reference frequency for TsunCenter
f_LO=1.98e11 # Hz

freqs=myMsmd.chanfreqs(acSpw)[fChans] # Hz

# Make SPW arrays in increasing order for the reverse-channel (LSB) SPWs.
# We flip arrays related to measurements in acSpw only, the models
# (like ATM) are already in the correct order.
if (freqs[0] > freqs[1]):

  print("Reversed SPW.")

  freqs=np.flip(freqs)

  gXX=np.flip(gXX)
  gYY=np.flip(gYY)

  acTRxx=np.flip(acTRxx)
  acTRyy=np.flip(acTRyy)

# Central frequency of the AC SPW
f0=freqs.mean()

# Mean solar T_B at the current SPW central frequency.
Tsun0=Tsun00+b_2*np.log(f0/f_LO)

print('Tsun=',Tsun0)

# Normalized (to unity) solar spectrum (linearized).
sf=1.0+b_2/Tsun0*(freqs/f0-1.0)

#- Spectral correction factors

SpCxx=1.0/np.mean(gXX*trn*sf)
SpCyy=1.0/np.mean(gYY*trn*sf)


plt.plot(gXX,'r-')
plt.plot(trn,'g-')
plt.plot(sf,'b-')

print('Spectral correction factors: ',SpCxx,SpCyy)

#- Offsets (P_sqld for zero P_ac)

# Separate the individual contributions in order to compare their weights.
# Pol XX
TrxDiff=sqTRxx-SpCxx*np.mean(gXX*acTRxx)
```

```
    TskyDiff=(1.0-eta_F+eta_l)*(TeBB.mean()-SpCxx*np.mean(gXX*TeBB))
    TspillDiff=(1.0-eta_l)*Tamb*(1.0-SpCxx)

    P00xx=sqGxx*(TrxDiff+TskyDiff+TspillDiff)+sqPoXX

    print('XX offset components: dTrx=',TrxDiff,'dTsky=',TskyDiff,
          'dTspill=',TspillDiff,'Po/G=',sqPoXX/sqGxx)

    # Pol YY
    TrxDiff=sqTRyy-SpCyy*np.mean(gYY*acTRyy)
    TskyDiff=(1.0-eta_F+eta_l)*(TeBB.mean()-SpCyy*np.mean(gYY*TeBB))
    TspillDiff=(1.0-eta_l)*Tamb*(1.0-SpCyy)

    P00yy=sqGyy*(TrxDiff+TskyDiff+TspillDiff)+sqPoYY

    print('YY offset components: dTrx=',TrxDiff,'dTsky=',TskyDiff,
          'dTspill=',TspillDiff,'Po/G=',sqPoYY/sqGyy)


  #--> endif(resolved)


  # Add the [0.0, P00] point to the arrays of measured powers in order
  # to extent the proportionality line.
  acPowXXext=np.append(acPowXX,0.0)
  sqPowXXext=np.append(sqPowXX,P00xx)

  acPowYYext=np.append(acPowYY,0.0)
  sqPowYYext=np.append(sqPowYY,P00yy)

  # Find the correlation slopes
  Kxx,Qxx=np.polyfit(acPowXXext,sqPowXXext,1)
  Kyy,Qyy=np.polyfit(acPowYYext,sqPowYYext,1)

  # AC solar-attenuated gains as recalculated from their SQLD counterparts.
  acGxxM=SpCxx*sqGxx/Kxx
  acGyyM=SpCyy*sqGyy/Kyy


  #-- The result: Gain correction factors
  gainCorrectionXX=acGxx.mean()/acGxxM
  gainCorrectionYY=acGyy.mean()/acGyyM

  print('direct/derived G* XX=',gainCorrectionXX)
  print('direct/derived G* YY=',gainCorrectionYY)

  return (gainCorrectionXX,gainCorrectionYY)


#--------------------------------------------------------------------------


def acd_calTabs2(vis,atmScan,ant,spw,myAtm=None,myMS=None,myMsmd=None,
                 acdLoads=None):
  """
  Improved version of acd_calTabs() for solar AC data. It uses transfer of
  relative gain-drop between P_sky and P_off scans from SQLD to AC spws - see
  experimentation in G_off_cal().
```

```
Because of corrections to frequency-dependent transmissivity and intrinsic
radiation of the terrestrial atmosphere in channel-resolved AC spws, we need
to suply the ATM model *myAtm* for the solar G* calibration. It is
the *attool* set for our atmosphere, for all 4 resolved sci SPWs, and it can
be conveniently obtained via setAtmTool4SPWs() - see also dWdB_Tbs().

If we do not calibrate for the Sun, i.e. when *atmScan* is not followed by
a solar scan, *myAtm* parameter may remain empty. Similarly, if the *spw* is
SQLD, we do nothing more on top of acd_calTabs(), so *myAtm* is ignored.
"""

#- Global tools & data

# ms and msmd tools flags, checked at closing
ownMS=False
ownMsmd=False

# Open own tools if 'global' not provided
if (not myMS):
  # Open the ms tool.
  myMS=mstool()
  myMS.open(vis)
  ownMS=True

if (not myMsmd):
  # Open the msmd tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True

# Create our dictionary with ACD load temperatures (ambient, hot) if needed.
if (not acdLoads):
  acdLoads=aU.getLoadTemperatures(vis,doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)

#- Get the standard ACD calibration tables.

calDict=acd_calTabs(vis,atmScan,ant,spw,myMS=myMS,myMsmd=myMsmd,
                    acdLoads=acdLoads)


#- Check, what else is necessary

sqldSPWs=[0,1,2,3,13,14,15,16,25,26,27,28]
solScans=myMsmd.scansforintent('OBSERVE_TARGET#ON_SOURCE')
tgtScan=atmScan+1

if ((spw in sqldSPWs) or (tgtScan not in solScans)):
  # It is easy, we may return the standard caltables 'as is'.
  return calDict

#- We are in the AC SPW and doing G* for the Sun. We need to transfer
#  the P_off/P_sky gain-drop from the SQLD spw.

# Check, whether we have ATM tool ready. Exit with an error if not.
if (not myAtm):
```

```
  print("Error: myAtm ATM tool must be defined for solar G* calibration!")
  print("Exiting...")

  return False

#--- OK, continue with G* corrections.

# Mapping between SPWs of the same frequency domain: AC -> SQLD
spwMap={5:0,6:0,
        7:1,8:1,
        9:2,10:2,
        11:3,12:3}

sqldSpw=spwMap[spw]

print("Transferring gains from the SQLD spw",sqldSpw,'...')

# For non-resolved SPWs we have a single channel, for resolved integrate
# the spectra over a range with the edges trimmed.
# Note: We trim the edges symmetrically, so flipping of reversed SPWs
# does not affect the result.
if (spw in [6,8,10,12]):
  fChans=[0]
  resolved=False
else:
  fChans=range(8,120) # Trimm edge channels symmetrically.
  resolved=True

#-- Powers for on-source subscans: AC and SQLD, we use their proportionality.

# Retrieve the AC-data time series.
(acTime,acData)=tgt_pow(vis,tgtScan,ant,spw,myMS=myMS,myMsmd=myMsmd,
                        subScans='on')

# Extract AC powers per polarization, averaged over channels.
acPowXX=acData[0,fChans].mean(axis=0)
acPowYY=acData[1,fChans].mean(axis=0)

# Retrieve the SQLD-data time series.
(sqTime,sqldData)=tgt_pow(vis,tgtScan,ant,sqldSpw,myMS=myMS,myMsmd=myMsmd,
                          subScans='on')

# Extract SQLD powers per polarization, rebin to the same timebase as the ACs.
sqPowXX=rebin2(sqldData[0,0],sqTime,acTime)
sqPowYY=rebin2(sqldData[1,0],sqTime,acTime)

#-- Standard ACD caltables for our AC and adjunged SQLD spws.

#- 1. Standard ACD caltable for our AC SPW - we have it already.

acCtb=calDict

# Low-attenuation (P_sky-based) gains G. The P_off-based  G* is untrusted.
# Limit to fChans range for calculations.
laGxx=1.0/acCtb['iG'][0,fChans]
laGyy=1.0/acCtb['iG'][1,fChans]

# Spectral P_sky-based gains normalised to 1.0 with fChans range.
```

```
# We assume these remain untouched by the attenuation level switching,
# i.e., they are the same also for the P_off sub-scans.
gXXnorm=laGxx.mean()
gYYnorm=laGyy.mean()

gXX=laGxx/gXXnorm
gYY=laGyy/gYYnorm

# Trx in AC. Limited to fChans range.
acTRxx=acCtb['T_rx'][0,fChans]
acTRyy=acCtb['T_rx'][1,fChans]


#- 2. ACD caltable for the adjunged SQLD spw.

sqldCtb=acd_calTabs(vis,atmScan,ant,sqldSpw,myMS=myMS,myMsmd=myMsmd,
                    acdLoads=acdLoads)

# G* in SQLD spw (trusted).
# We have leading and trailing values from the two (lead/trail) P_off subscans
sqGxxLead=1.0/sqldCtb['iG*'][0,0,0]
sqGxxTrail=1.0/sqldCtb['iG*'][1,0,0]

sqGyyLead=1.0/sqldCtb['iG*'][0,1,0]
sqGyyTrail=1.0/sqldCtb['iG*'][1,1,0]

'''
# For an offset calculation (see below) we need also averages over a scan.
sqGxx=0.5*(sqGxxLead+sqGxxTrail)
sqGyy=0.5*(sqGyyLead+sqGyyTrail)
'''

# As it has been found, the high-attenuation gain calcualted form P_off is
# affected by changing WVR during solar scan - see comment in acd_calTabs().
# It is therefore better to take the gain calculated from P_off measurement
# immediatelly following the low-attenuation P_sky ATM scan, i.e., from the
# leading P_off sub-scan of solar scan.
sqGxx=sqGxxLead
sqGyy=sqGyyLead


# Trx in SQLD
sqTRxx=sqldCtb['T_rx'][0,0]
sqTRyy=sqldCtb['T_rx'][1,0]

# Pzero in SQLD
sqPoXX=sqldCtb['calPows']['P_zero'][0,0]
sqPoYY=sqldCtb['calPows']['P_zero'][1,0]


if (not resolved):
  #-- Intregrated AC SPWs are easy

  # Spectral correction factors. Trivial here, we have no spectral dependency
  # within the integrated AC SPW.
  SpCxx=1.0
  SpCyy=1.0
```

```
    # Offset Q: Assumed SQLD power at the zero AC power in the correlation
    # P_sqld=K*P_ac+Q. Essentially Po, but may be modified if Trx differs
    # between the AC and corresponding SQLD spw.
    P00xx=sqGxx*(sqTRxx-acTRxx[0])+sqPoXX
    P00yy=sqGyy*(sqTRyy-acTRyy[0])+sqPoYY

else:
    #-- Resolved AC SPWs need much more care - we take into account spectral
    #   weighting by normalized spectral gain, atmospheric tranmsmissivity
    #   (+ atm. radiation). Solar spectrum slope is negligible, we ignore it.


    #- Atmospheric transmissivity, atm. radiation, and instrumental effects.

    # Map our science SPW to the 0 - 3 range for spws in *myAtm* attool.
    sciAtmMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}
    atmSPW=sciAtmMap[spw]

    # Setup the ATM model for our actual PWV as per SYSCAL table.
    pwv=getPWVforAnt(vis,tgtScan,ant,myMsmd=myMsmd)['pwvAvg']
    myAtm.setUserWH2O(qa.quantity(pwv,'mm'))

    # Extract the empty-sky (atm+CMB) brightness temperature (spectrum) in K.
    # Limit its domain to our fChans range.
    TeBB=myAtm.getTebbSkySpec(atmSPW,qa.quantity(pwv,'mm'))[1]['value'][fChans]

    # Calculate transmissivity.

    # Dry zenith opacity
    tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
    # Water-vapour related 'wet' opacity
    tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
    # Airmass for the current scan.
    airMass=myAtm.getAirMass()

    # Total opacity (spectral). Limit it to fChans range.
    opacity=(airMass*(tau_wet+tau_dry))[fChans]
    # Transmissivity (spectral).
    tr=np.exp(-opacity)

    # Relative spectral transmissivity normalized to unity.
    trn=tr/tr.mean()

    #- Antenna efficiencies

    antNames=myMsmd.antennanames()
    myAntName=antNames[ant]
    antType=myAntName[:2]

    # Map the antenna efficiencies by antenna type. As per investigation from
    # merged T_sun temperatures form 2023 and 2026 data (project_22_25_Tb.ods).
    eta_l_map={'CM':0.941,'DA':0.947,'DV':0.956}
    eta_F_map={'CM':0.891,'DA':0.897,'DV':0.906}

    # Rear-spillover + ohmic antenna efficiency - see project_22_25_Tb.ods
    eta_l=eta_l_map[antType]
    # Forward-spillover antenna efficiency - see project_22_25_Tb.ods
    eta_F=eta_F_map[antType]
```

```
    # We have two exception that seem to have systematically lower
    # antenna efficiencies: DA64 and DV08 - see project_22_25_Tb.ods
    if myAntName=='DA64':
      eta_l=0.929
      eta_F=0.880
    if myAntName=='DV08':
      eta_l=0.943
      eta_F=0.894

    # Rear-spillover temperature, in Kelvins.
    Tamb=acdLoads[ant][atmScan]['amb']

    #- Channels in ascending order.

    freqs=myMsmd.chanfreqs(spw)[fChans] # Hz

    # Make SPW arrays in increasing order for the reverse-channel (LSB) SPWs.
    # We flip arrays related to measurements in the acSpw only, the models
    # (like ATM) are already in the correct order.
    if (freqs[0] > freqs[1]):

      freqs=np.flip(freqs)

      gXX=np.flip(gXX)
      gYY=np.flip(gYY)

      acTRxx=np.flip(acTRxx)
      acTRyy=np.flip(acTRyy)

    #- Spectral correction factors

    SpCxx=1.0/ np.mean(gXX*trn)
    SpCyy=1.0/ np.mean(gYY*trn)


    #- Offsets (P_sqld for zero P_ac)

    # Separate the individual contributions in order to compare their weights.
    # Pol XX
    TrxDiff=sqTRxx- SpCxx* np.mean(gXX* acTRxx)
    # T_rx is in kiloKelvins(!), TeBB and Tamb not: Scale all to kK.
    TskyDiff=0.001* (1.0- eta_F+eta_F* eta_l)* (TeBB.mean()- SpCxx* np.mean(gXX* TeBB))
    TspillDiff=0.001* eta_F* (1.0- eta_l)* Tamb* (1.0- SpCxx)

    P00xx=sqGxx* (TrxDiff+TskyDiff+TspillDiff)+sqPoXX


    # Pol YY
    TrxDiff=sqTRyy- SpCyy* np.mean(gYY* acTRyy)
    TskyDiff=0.001* (1.0- eta_F+eta_F* eta_l)* (TeBB.mean()- SpCyy* np.mean(gYY* TeBB))
    TspillDiff=0.001* (1.0- eta_l)* Tamb* (1.0- SpCyy)

    P00yy=sqGyy* (TrxDiff+TskyDiff+TspillDiff)+sqPoYY


#--> endif(resolved)
```

```
  # Add the [0.0, P00] point to the arrays of measured powers in order
  # to extent the proportionality line.

  acPowXXext=np.append(acPowXX,0.0)
  sqPowXXext=np.append(sqPowXX,P00xx)

  acPowYYext=np.append(acPowYY,0.0)
  sqPowYYext=np.append(sqPowYY,P00yy)

  # Find the correlation slopes
  Kxx,Qxx=np.polyfit(acPowXXext,sqPowXXext,1)
  Kyy,Qyy=np.polyfit(acPowYYext,sqPowYYext,1)


  # AC solar-attenuated gains as recalculated from their SQLD counterparts.
  # These represent the spw-averaged values, i.e. the scales.
  acGxxLead=SpCxx* sqGxxLead/ Kxx
  acGxxTrail=SpCxx* sqGxxTrail/ Kxx

  acGyyLead=SpCyy* sqGyyLead/ Kyy
  acGyyTrail=SpCyy* sqGyyTrail/ Kyy


  # Now use them to scale the spectral normalized gains.
  # We wish now to have the gains G* over full SPW, including the trimmed edge
  # channels. So, produce first the gXX/gYY versions with the edge channels
  # included. Moreover, we have natural channel order - no flipping in LSB spws.
  gXXext=(1.0/ acCtb['iG'][0])/ gXXnorm
  gYYext=(1.0/ acCtb['iG'][1])/ gYYnorm

  # Now apply the scaling.

  gStarXXLead=acGxxLead* gXXext
  gStarXXTrail=acGxxTrail* gXXext

  gStarYYLead=acGyyLead* gYYext
  gStarYYTrail=acGyyTrail* gYYext

  # Replace original iG* in the standard-ACD calDict with the new values.
  calDict['iG*'][0,0]=1.0/ gStarXXLead
  calDict['iG*'][1,0]=1.0/ gStarXXTrail

  calDict['iG*'][0,1]=1.0/ gStarYYLead
  calDict['iG*'][1,1]=1.0/ gStarYYTrail

  # Return the modified calDict.
  return calDict


#--------------------------------------------------------------------------

def acd_caltab_plots(vis,atmScans,sciSPWs):
  """
  A diagnostic routine that plots G/G* and T_rx as found in acd_calTabs() as
  a function of frequency in the channel-resolved SPWs for each antenna.
  """

  #-- This MS-specific defs.
```

```
# Extract info about antennas and SPWs.
myMsmd=msmdtool()
myMsmd.open(vis)

# Find the antenna list (the IDs and corresponding names).
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()

# All Sun scans.
solScans=myMsmd.scansforintent('OBSERVE_TARGET#ON_SOURCE')

myMsmd.done()

# Output PNG directory.
pngDir=os.getcwd+'/acdCals_plots/'

#-- Loop over all atm scans

for aScan in atmScans:

  # The next-target scan: Solar or BP-cal?
  tgtScan=atmScan+1

  # Do the solar off-scan and G* calculation only if the Sun scan follows.
  doSolarGain=(tgtScan in solScans)

  # Set the output PNG file path - the source description (Sun/BP).
  fPath=""

  if doSolarGain:
    fPath=pngDir+"Sun_"
  else:
    fPath=pngDir+"BP_"

  #-- Loop over all antennas

  for myant in antIDs:

    # Report progress.
    print('For scan ',aScan,' calculating antenna ',
          int(myant),'<-->',antNames[myant])


    #-- Init plot for antenna.

    plt.rcParams["figure.figsize"] = (20,80)
    # Make XX and YY plots as subplots (horizontally)
    ff,sps=plt.subplots(4,2)

    # Antenna name as the entire-plot title.
    plt.suptitle(antNames[myant])

    # X-axis label just for the bottom sub-plot.
    sps[3,0].set_xlabel('frequency [GHz]')
    sps[3,1].set_xlabel('frequency [GHz]')

    #-- Loop over SPWs. Access them by index in order to map them to subplots
```

```
for spwIndex in range(len(sciSPWs)):

  spw=sciSPWs[spwIndex]

  calFactors=acd_calTabs(vis,aScan,int(myant),spw)

  # Extract the frequency SOW definition.
  freqs=calFactors['freqs']
  nChans=len(freqs)


  # Extract the gain G/G* and the additive constant T_rx
  Trx_XX=calFactors['T_rx'][0]
  Trx_YY=calFactors['T_rx'][1]

  if doSolarGain:
    # We are doing solar scan, show G* as the 'main' gain.
    # We have two values for the beginning and the end of the solar scan,
    # show their average.

    iG_XX=0.5*(calFactors['iG*'][0][0]+calFactors['iG*'][1][0])
    iG_YY=0.5*(calFactors['iG*'][0][1]+calFactors['iG*'][1][1])

    # Over-plot the MD gain before being scaled by the P_off factor.
    # Prepare it trimmed by edge channels on-the-fly - see below.
    iG0_XX=calFactors['iG'][0][7:nChans-8]
    iG0_YY=calFactors['iG'][1][7:nChans-8]

  else:
    # We are doing BP scan, show G as the gain.
    iG_XX=calFactors['iG'][0]
    iG_YY=calFactors['iG'][1]



  # Trim the edge channels
  spwFreqs=freqs[spwIndex][7:nChans-8]

  Trx_XX=Trx_XX[7:nChans-8]
  Trx_YY=Trx_YY[7:nChans-8]

  iG_XX=iG_XX[7:nChans-8]
  iG_YY=iG_YY[7:nChans-8]


  # Y-axis labels for each sub-plot
  sps[spwIndex,0].set_ylabel('T_rx [K]')
  sps[spwIndex,1].set_ylabel('Inverse gain')


  # Plot T_rx in the left column, XX as red, YY as blue.
  sps[spwIndex,0].plot(spwFreqs,Trx_XX,'-r')
  sps[spwIndex,0].plot(spwFreqs,Trx_YY,'-b')


  # Plot the (inverse) gain in the right column, XX as red, YY as blue.
  sps[spwIndex,1].plot(spwFreqs,iG_XX,'-r')
```

```
        sps[spwIndex,1].plot(spwFreqs,iG_YY,'-b')

        # If we are in solar scan, overplot the non-scaled gain as dashed.
        if (aScan==8):
          sps[spwIndex,1].plot(spwFreqs,iG0_XX,'--r')
          sps[spwIndex,1].plot(spwFreqs,iG0_YY,'--b')


      # Save to a file and close.
      plt.savefig(fPath+antNames[myant]+'.png')
      plt.show()
      plt.close()


  return

#-----------------------------------------------------------------------------
#             3. Extract raw data from the MS & Gain calTables
#-----------------------------------------------------------------------------

def tgt_pow(vis,tgtScan,ant,spw,myMS=None,myMsmd=None,subScans='on'):
  """
  Extract total-power data in *spw* for target defined by the *tgtScan* number
  at the antenna *ant*.
  The *myMS* and *myMsmd* tools attached to *vis* may be supplied for speed-up.
  The *subScan* is relevant just for the solar target and may be empty
  (entire scan) or 'on'/'off' for the on-source (the default) and off-source
  sub-scans selection.

  Target, antenna, and spw params MUST be int, not strings - see acd_calTabs().

  Performs MS selection in order to extract the raw auto-correlations (AC) or
  SQLD data (depending on the SPW selected). We get the time-series of the
  *total powers* (either AC or SQLD); if the SPW is channel-resolved (spws
  5, 7, 9, and 11), we have even time series of the spectra. If the *tgtScan*
  is a solar scan, we have to select just the 'internal' subscans, omitting the
  very first and very last sub-scans, as they are taken in the off-source
  position for calibration (see acd_calTabs() for detail).

  Note: If intended to be used with acd_calTabs(), the ATM scan in acd_calTabs()
  should as much close as possible precede the target scan in tgt_pow().
  This is the case, for example, with the arguments *atmScan* and *tgtScan*
  in tgt_Tb().

  Returns: A tupple with two components.
  The 1st component: Times for each amplitude as MJD - an array of size nIntegs.
  The 2nd component: Array of powers in the form 2 x nChans x nIntegs.
  I.e., 2 x 1 x nIntegs for the SQLD and chan-integrated SPWs, and
  2 x 128 x nIntegs for channel-resolved SPWs. The first dim (2) means
  polarization (XX, YY).
  """

  # ms and msmd tools flags.
  ownMS=False
  ownMsmd=False

  # Open own tools if 'global' not provided
  if (not myMS):
```

```
    # Open the ms tool.
    myMS=mstool()
    myMS.open(vis)
    ownMS=True

if (not myMsmd):
    # Open the ms tool.
    myMsmd=msmdtool()
    myMsmd.open(vis)
    ownMsmd=True


#--- Selection by the sub-scans. This we will need if the source is the Sun:
#    The very firts and very last sub-scans in the solar scan are for
#    taking the P_off. This is like P_sky in dual-load but with high
#    attenuation level, suitable for the solar scans.


# Find the data_desc_id for a given science SPW.
sciDataDescID=int(myMsmd.datadescids(spw=spw)[0])

# Initialize the MS selector by SPW.
myMS.selectinit(datadescid=sciDataDescID)

# Select ACs/SQLDs for a given antenna and the scan number.
myMS.select({'antenna1':ant,'antenna2':ant,'scan_number':tgtScan})

#-- Are we (tgtScan) the solar scan? If so, refine the selection.

# All Sun scans.
solScans=myMsmd.scansforintent('OBSERVE_TARGET#ON_SOURCE')

if (tgtScan in solScans):
    #- In case of the solar scan we have more strict selection by subscans.

    # Get sub-scan info for our tgtScan.
    sub_info=aU.computeDurationOfScan(tgtScan,mymsmd=myMsmd,
                                      returnSubscanTimes=True)

    # How many sub-scans do we have?
    nSubScans=sub_info[1]

    # The on-source subscans are just these in the middle (1st/last are off).
    # Note: This is a key to the map/dictionary of subscans and it starts at 1.
    offSubScans=[1,nSubScans]
    onSubScans=range(2,nSubScans)



    # Make and fill a list of time-ranges (as MJDs) for the on-source sub-scans.
    # We use the string representation of date-time - that is why sub_info[3] is
    # extracted here. This is suitable for ms::msselect() used subsequently -
    # see the comments below.
    tRanges=[]


    if (subScans=='on'):
      for oss in onSubScans:
```

```
        tRanges.append(sub_info[3][oss])
    elif(subScans=='off'):
      for offs in offSubScans:
        tRanges.append(sub_info[3][offs])


    # Make the string-list to be a single comma-separated string.
    trString=','.join(tRanges)

    # On demand, refine selection to on-/off- subscans defined by time-range.
    # Note: We use ms::msselect() (and not ms::select()) here as it allows
    # for non-contiguous union of time intervals for the sub-scans
    if(trString): myMS.msselect({'time':trString})

  #-- Filtration done, now pick-up and process the data.

  # Extract the (complex) data and their 'coordinates' from the filtered MS.
  tgtData=myMS.getdata(["data","axis_info"],ifraxis=True)

  # Get the data (complex amplitudes)
  tgt_cAmp=tgtData["data"]

  # Get their corresponding times as MJD.
  tgt_mjd=tgtData["axis_info"]["time_axis"]["MJDseconds"]

  # The data are complex by nature, but in fact having just the real part.
  tgtPow=tgt_cAmp.real

  # The 'ifraxis=True' in the ms.getdata() above causes expansion of the antenna
  # axis. In our case (auto-corrs or SQLDs) it is degenerated to dimension=1.
  # Let us remove it (its is the 3rd axis of the array).
  tgt_pow=np.squeeze(tgtPow,axis=2)

  #print('Tgt pow: ',tgt_pow)

  # Reset the selection.
  myMS.reset()

  # Data are extracted, we can close the tools (if local).
  if(ownMsmd): myMsmd.done()
  if(ownMS): myMS.close()


  # Return the time axis as MJD.
  # Return the power as a numpy array of nPols x nChans x nIntegs dims.
  return (tgt_mjd,tgt_pow)


#-----------------------------------------------------------------------------

def tgtPow_crossPlots(vis,ant,tgtScan,spw1,spw2):
  """
  Compare the power variations (mean-subtracted powers) for target in *tgtScan*
  in two different SPWs *spw1* and *spw2* (should be of the same type: either
  AC integrated or SQLDs - not checked!) for a given antenna *ant*.
  Specifically, we do scatter plot of power variations in *spw2* vs. powers
  in *spw1*.
```

```
    Idea: Do we see dry-component or source intrinsic contribution to the power
    variations in the SPWs further from the WV line center? The ultimate question
    is, which of the approaches to the dPWV proxy shall be used - whether
    dPWV() / dPWV_spwPart() or dWdB(), which counts with the (apparent)
    variability of the source, too (and may accound for adding the dry-component).

    Result: When used for the BP calibrator, we see just a single linear
    correlation of the signal between the *spw1* and *spw2*. Its slope more or
    less corresponds to theory, given the big relative uncertainty in Tsky-Tatm.
    For the Sun, however, we see a sort of two timescales, each with a different
    slope. It has been found that the slower-variations linear slope corresponds
    to the ratio of matrix coefficients for dW, while the faster one for dB,
    when decomposing via dWdB() method is applied. Seee dWdB() for details.
    """

    # Extract the data
    time1,data1=tgt_pow(vis,tgtScan,ant,spw1)
    time2,data2=tgt_pow(vis,tgtScan,ant,spw2)

    # We use channel-integrated windows - get rid of the fake frequency dimension.
    d1=np.squeeze(data1,axis=1)
    d2=np.squeeze(data2,axis=1)


    # Separate by polarizations, keeping just the varying parts.
    d1MeanXX=d1[0].mean()
    d1VarsXX=d1[0]-d1MeanXX

    d1MeanYY=d1[1].mean()
    d1VarsYY=d1[1]-d1MeanYY

    d2MeanXX=d2[0].mean()
    d2VarsXX=d2[0]-d2MeanXX

    d2MeanYY=d2[1].mean()
    d2VarsYY=d2[1]-d2MeanYY

    plt.plot(d1VarsXX,d2VarsXX,'r-')
    plt.plot(d1VarsYY,d2VarsYY,'b.')

    return


#--------------------------------------------------------------------------

def getPhase(gTable,antID,spw):
    """
    Extract phase variations as a function of time from the gain-cal calibration
    table *gTable* for antenna with *antID* and in the spectral window *spw*.
    """
    # Create the table tool and use it to open the gain caltable.
    cTab=tbtool()
    cTab.open(gTable)

    # Construct a query string: Selection by antenna and spw.
    selection="ANTENNA1=="+str(antID)+" && SPECTRAL_WINDOW_ID="+str(spw)

    # Extract the sub-table that correspond to the selection criteria.
```

```
  # We need just the time and gain columns.
  ctSel=cTab.query(selection,sortlist="TIME",columns="TIME,CPARAM")

  # Extract the column data
  times=ctSel.getcol("TIME")
  gains=ctSel.getcol("CPARAM")


  print("--- Sizes: ", len(times), gains.shape)


  # Data are extracted, we may close the tab tools (incl. the sub-table ctSel).
  ctSel.close()
  cTab.done()

  # The gains array has a shape (nPols x 1 x nTimeSamples) for standard
  # G-tables and only (1 x nTimeSamples) for the WVR/PWV caltables.
  # Check the actual shape.
  nPols=gains.shape[0]

  if (nPols==2):
    # Extract the polarizations separatelly and contract the fake second axis.
    # As a result, we get just 1D array with sizeequal to nTimeSamples for
    # each polarization.
    gXX=gains[0][0]
    gYY=gains[1][0]

    # We are interested just in the phases.
    phaseXX=np.angle(gXX)
    phaseYY=np.angle(gYY)

  else:
    # Just Stokes-I
    gII=gains[0][0]

    # In order to have uniform results, duplicate the phases for XX/YY pols.
    phaseXX=np.angle(gII)
    phaseYY=phaseXX


  # Use np.unwrap in order to get rid of the phase jumps at +/- PI.
  upXX=np.unwrap(phaseXX)
  upYY=np.unwrap(phaseYY)

  # Form the returned result as a dictionary.
  pDict={'mjd':times,'phiXX':upXX,'phiYY':upYY}

  return pDict

#-----------------------------------------------------------------------------

def getPhaseMS(vis,antID,spw):
  """
  Instead of using the gaincal() solution contained in the gainTable (as
  getPhase() does), try to extract phases directly from the MS using the DATA
  and MODEL columns.

  We are doing this trick just for the solar MS, so the refAnt and tgtScan
```

```
parameters are hardwired at the beginning of this routine.
"""

#--- Hard-wired settings for the solar test MS.

# Reference antenna for the solar MS.
refAnt=37
# Solar observe-target scan.
tgtScan=9


myMsmd=msmdtool()
myMsmd.open(vis)

#- 1. Identify the SPW by the data_desc_id
myDataDescID=int(myMsmd.datadescids(spw=spw)[0])

#print('Selected data desc ID: ',myDataDescID)

#- 2. Identify the off-source subscans by their time range.

# Get sub-scan info for our tgtScan.
sub_info=aU.computeDurationOfScan(tgtScan,mymsmd=myMsmd,
                                  returnSubscanTimes=True)
# How many sub-scans do we have?
nSubScans=sub_info[1]

# The on-source subscans are those middle, except the first and last one.
# Note: This is a key to the map/dictionary of subscans and it starts at 1.
onSubScans=range(2,nSubScans)
# Make and fill a list of time-ranges (as MJDs) for the on-source sub-scans.
# We use the string representation of date-time - that is why sub_info[3] is
# extracted here. This is suitable for ms::msselect() used subsequently -
# see the comments below.
tRanges=[]

for onss in onSubScans:
  tRanges.append(sub_info[3][onss])

# Make the string-list to be a single comma-separated string.
trString=','.join(tRanges)

# We are done with the time-range extraction.
myMsmd.done()

#--- Use the above identifiers/selectors for filterring the data from MS.

myMs=mstool()
myMs.open(vis)

# Initialize the MS selector by SPW
myMs.selectinit(datadescid=myDataDescID)

# Matrix of cross-correlations/visibilities is just the upper-triangular:
# The 'antenna2' column starts with antenna that have by 1 bigger ID than
# antenna1 (checked in browsetable()). So, we have to select carefully.

if (refAnt > antID):
```

```
  ant1=antID
  ant2=refAnt
else:
  ant1=refAnt
  ant2=antID


# Select data for a given antenna and the scan number.
myMs.select({'antenna1':ant1,'antenna2':ant2,'scan_number':tgtScan})

# Refine selection by on-source subscans only.
# Note: We use ms::msselect() (and not ms::select()) here as it allows
# for non-contiguous union of time intervals for the sub-scans
myMs.msselect({'time':trString})

#--- Filtration done, now pick-up and process the data.

# Extract the (complex) data and their 'coordinates' from the filtered MS.
allData=myMs.getdata(["data","model_data","axis_info"],ifraxis=True)


# Clean-up the tool MS tool.
myMs.reset()
myMs.close()

#- Separate the complex-structured data to components
#  Remove the degenerated baseline axis of dim=1.

# Get the raw-data visibilities stored as complex amplitudes.
rawVis=np.squeeze(allData["data"],axis=2)
# Get the model visibilities stored as complex amplitudes.
modVis=np.squeeze(allData["model_data"],axis=2)
# Get their corresponding times as MJD secs.
time_mjds=allData["axis_info"]["time_axis"]["MJDseconds"]


#- Data processing: Extract the phases only.

# Complex amplitudes for MODEL - trim the more extended edge channels because
# the MFS in tclean() does not work with them either.

mcAmpXX=modVis[0,20:107,:]
mcAmpYY=modVis[1,20:107,:]

# Complex amplitudes for DATA - keep the same shape as for model.

dcAmpXX=rawVis[0,20:107,:]
dcAmpYY=rawVis[1,20:107,:]

# Process the phases. Correct wraps along the f-axis for each time instant.

# Get the size of t-axis, it is the same for all arrays.
tSize=dcAmpXX.shape[1]

# Arrays for time-evolution of the mean phase delays and their STDdevs.
phiXX=np.empty(tSize)
phiYY=np.empty(tSize)
```

```
  stdPhiXX=np.empty(tSize)
  stdPhiYY=np.empty(tSize)


  # Loop over time (integrations).
  for tt in range(tSize):

    # DATA phases as functions of channel, unwrapped.
    dPhiXX=unwrap2(np.angle(dcAmpXX[:,tt]))
    dPhiYY=unwrap2(np.angle(dcAmpYY[:,tt]))

    # MODEL phases as functions of channel, unwrapped.
    mPhiXX=unwrap2(np.angle(mcAmpXX[:,tt]))
    mPhiYY=unwrap2(np.angle(mcAmpYY[:,tt]))

    # Make a difference DATA - MODEL
    dmPhiXX=dPhiXX-mPhiXX
    dmPhiYY=dPhiYY-mPhiYY

    # Find the mean (over f-channels) and RMS.
    phiXX[tt]=np.mean(dmPhiXX)
    phiYY[tt]=np.mean(dmPhiYY)

    # Find the stddev (over f-channels) and RMS.
    stdPhiXX[tt]=np.std(dmPhiXX)
    stdPhiYY[tt]=np.std(dmPhiYY)

  # Unwrap the phase jumps also in the time dimension.
  pXX=np.unwrap(phiXX)
  pYY=np.unwrap(phiYY)

  return (time_mjds,pXX,pYY,stdPhiXX,stdPhiYY)


#-----------------------------------------------------------------------------


def plotPhases(vis,gTable,spw):
  """
  Extract phases from the table *gTable* for a given *spw* using getPhase() and
  make series of plots (as PNG files) for all antennas contained in the current
  MS *vis*.
  """

  #-- This MS-specific defs.

  # Extract info about antennas and SPWs.
  myMsmd=msmdtool()
  myMsmd.open(vis)

  # Find the antenna list (the IDs and corresponding names).
  antIDs=myMsmd.antennaids()
  antNames=myMsmd.antennanames()

  myMsmd.done()

  # Gain table may reside in another directory (e.g., calibrated,self_cal,...)
  # Do not confuse the outpit dir, use just its base-name for id of outputs.
```

```
  gTableName=os.path.basename(gTable)

  # Output PNG directory. One per gain table.
  pngDir='./phase_plots/'+gTableName+'/'
  if (not os.path.exists(pngDir)): os.makedirs(pngDir)

  #-- Loop over all antennas

  for myant in antIDs:

    # Report progress.
    print('Plotting antenna ',int(myant),'<-->',antNames[myant])

    # Get phases for a given time as the time->phase map (a dictionary).
    tpDict=getPhase(gTable,int(myant),spw)

    time_axis=tpDict['mjd']
    phx=tpDict['phiXX']
    phy=tpDict['phiYY']

    #- Make a plot with an overplot

    # Plot phases in red: XX solid line, YY dashed.
    plt.plot(time_axis,phx,'-r')
    plt.plot(time_axis,phy,'--r')


    # Add the antenna name as a plot title.
    plt.title(antNames[myant])

    # Save to a file and close.
    plt.savefig(pngDir+'phi_'+antNames[myant]+'.png')
    plt.show()
    plt.close()

  return



#-----------------------------------------------------------------------------
#             2.+3. ACD-calibrated data: From powers to temperatures
#-----------------------------------------------------------------------------

def tgt_Tb(vis,atmScan,tgtScan,antID,spwID,myMS=None,myMsmd=None,
           acdLoads=None,myAtm=None):
  """
  Calculate ACD-calibrated 'sky' brightness temperatures from the measured
  total-power (AC or SQLD) data. The 'sky' Tb is a sum of the atmosphere
  contribution and the attenuated (by passage through the same atmosphere)
  signal of the source (target) defined by the *tgtScan* parameter. The
  *atmScan* should be the nearest ATM scan preceding to the target scan.

  Return: A tupple of three arrays. The first one is a time-axis, the 1-D array
  of size nInteg, merely reproduction of the time-axis (as MJD) from internally
  called tgt_pow(). The other two arrays - Txx and Tyy, are each of the shape
  nChans x nIntegs, nChans is either 1 (SQLD and chan-integrated AC SPWs)
  or 128 (chan-resolved AC SPWs) in case of Solar Band 5 hi-res MS.
  """
```

```
#-- MS and msmd tool: Do we have global or local instances?
casalog.post("tgt_Tb()")

# MS and msmd tools flags (local/global)
ownMS=False
ownMsmd=False

# Open own tools if 'global' not provided
if (not myMS):
  # Open the ms tool.
  myMS=mstool()
  myMS.open(vis)
  ownMS=True

if (not myMsmd):
  # Open the ms tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True

if (not acdLoads):
  acdLoads=aU.getLoadTemperatures(vis,doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)


#-- Read ACD calibration tables and the raw powers.

# ACD calibration 'tables': The (inverse) gain(s) and T_rx/T_inst.

gTrx=acd_calTabs2(vis,atmScan,antID,spwID,myAtm=myAtm,myMS=myMS,
                  myMsmd=myMsmd,acdLoads=acdLoads)

# Alternativelly, use the less accurate acd_calTabs().
# gTrx=acd_calTabs(vis,atmScan,antID,spwID,myMS=myMS,myMsmd=myMsmd,
#                  acdLoads=acdLoads)

# Raw total-power data for the target, preceded by their times as MJD
mjd_tgt,Ptgt=tgt_pow(vis,tgtScan,antID,spwID,myMS,myMsmd)


# Power has dims 2 x nChans x nIntegs, where 2 = num of pols.
# Keep just the time (nIntegs) and possibly freq. (nChans) dimensions.
sXX=Ptgt[0]
sYY=Ptgt[1]

# Analyse MS: Resolve between solar and BP tgtScan, find native sci SPW
# resolution for the target.

solScans=myMsmd.scansforintent('OBSERVE_TARGET#ON_SOURCE')
tgtFreqs=myMsmd.chanfreqs(spwID)*1.0e-9

# We can close the tools, if local.
if (ownMsmd): myMsmd.done()
if (ownMS): myMS.close()
```

```
#-- We return time-dependent (and for some SPWs also frequency-dependent)
#   quantities. In order to perform operations with multi-dimensional
#   arrays correctly, we have to use tensor multiplications (np.tensordot()).
#   To that end, the time-dimension needs to be incorporated as a vector.

# Num. of integrations in the target-power data.
nIntegs=Ptgt.shape[2]

# Time-dimension stepping vector [ 0.0, 1.0, 2.0, ... , float(nIntegs-1)]
# (works like findgen() in IDL).
tdVec=np.arange(nIntegs)
# Time-dimension unit vector [1.0, 1.0, 1.0, ... , 1.0]
# (works like std::vector<float>(size_t s, float, val=1.0) in C++).
tdUnitVec=np.ones(nIntegs)


#-- Calculate the time-domain gains.

#Are we (the tgtScan) a solar scan?
doSolarGain=(tgtScan in solScans)

if (doSolarGain):
  # Use G* if we (the tgtScan) are a solar scan.

  #--- Attention !!!
  #
  # Think about linear interpolation in time (expressed as integration index)
  # as we have iG* defined for the beginning and the end of the solar scan.
  # (No frequency-domain interpolation is required, for the solar gains we
  # have done it in acd_calTab() already.)
  #
  # On the other hand !!! Change of the P_off between leading and trailing
  # sub-scan *need not* to be solely because of the G* gain changes but also
  # because T_sky has chnanged between (i) ATM scan preceding the target scan,
  # (ii) leading P_off sub-scan of tgt scan, and (iii) trailing P_off subscan.
  # Derivation of Eq. (8) in White et al. (2017) assumes that T_sky does
  # not change, but as we are in Band 5 (high opacity) and usually with
  # medium to high PWV, change in the PWV may significantly change value
  # of T_sky. In principle, we should be able to infer T_sky(t) from the
  # raw WVR data as per dWdLs(). T_sky(t) can be then used to fixing the
  # P_off subscans w.r.t P_sky of preceding atm scan - this is todo item.
  # Until it is implemented, the safest way is to use just G* derived from
  # P_off of the *leading* sub-scan (that is why iGend below takes index '0').
  # The leading sub-scan follows immediatelly after atm scan and is observed
  # at the same sky position, so the T_sky change between those two should be
  # (hopefully) small.
  iGbeg=gTrx['iG*'][0]
  iGend=gTrx['iG*'][0]

  # To be uncommented after acd_Caltabs() takes into account T_sky time
  # variation inferred from the raw WVR data.
  # iGend=gTrx['iG*'][1]

  # In order to prevent mess with multi-dimensional arrays, we split
  # processing by polarization. As a result, we get just time- and
  # channel-dependent quantities (of the shape nChans x nIntegs)

  iGbegXX=iGbeg[0]
```

```
    iGbegYY=iGbeg[1]

    iGendXX=iGend[0]
    iGendYY=iGend[1]


    # The gain step (from integration-to-integration). 1-D arrays of nChans dim.
    gStepXX=(iGendXX-iGbegXX)/(nIntegs-1.0)
    gStepYY=(iGendYY-iGbegYY)/(nIntegs-1.0)


    # Linearly interpolated (in time) i-gains for XX and YY pols.
    iGainXX=np.tensordot(iGbegXX,tdUnitVec,0)+np.tensordot(gStepXX,tdVec,0)
    iGainYY=np.tensordot(iGbegYY,tdUnitVec,0)+np.tensordot(gStepYY,tdVec,0)


    # Extract T_inst - i.e. Trx corrected by Po/G for SQLD spws than Trx itself.
    # T_inst are constant in time, will be just extracted. The time-dimension
    # will be added below, during calibration.
    Tixx=gTrx['T_inst'][0]
    Tiyy=gTrx['T_inst'][1]

    #--> endif(Sun)

else:
    # Non-solar target. Use iG as a (inverse) gain, no time-interpolation.
    # If the gain/T_rx lives at different frecuency channels (see acd_calTab()),
    # do interpolation in the frequency domain. This shouild never happen in
    # the solar observation, as it has both target powers (sci SPWs) and the
    # Tsys SPW (for ACD cal) in the same shape (TDM = 128 channels).

    # Frequency domain for the gain G and T_rx
    gainFreqs=gTrx['freqs']

    # Just a very basic check - if it does not work, check yet f[0] and df.
    if (len(tgtFreqs) != len(gainFreqs)):

      # Gain and target spw definitions differ: Do interpolate.
      iGxx=np.interp(tgtFreqs,gainFreqs,gTrx['iG'][0])
      iGyy=np.interp(tgtFreqs,gainFreqs,gTrx['iG'][1])

      Tixx=np.interp(tgtFreqs,gainFreqs,gTrx['T_inst'][0])
      Tiyy=np.interp(tgtFreqs,gainFreqs,gTrx['T_inst'][1])

    else:
      # The same channels for gain and tgt data, just extract to a new array.

      iGxx=gTrx['iG'][0]
      iGyy=gTrx['iG'][1]

      Tixx=gTrx['T_inst'][0]
      Tiyy=gTrx['T_inst'][1]

    # Replicate in time-domain: Constant (in time) i-gains for XX and YY pols.
    iGainXX=np.tensordot(iGxx,tdUnitVec,0)
    iGainYY=np.tensordot(iGyy,tdUnitVec,0)

    #-->endelse(non-solar tgt)
```

```
  #-- Now apply the calibration separatelly for XX and YY pols.
  #   Add the time-dimension also to the Trx values.
  Txx=sXX*iGainXX-np.tensordot(Tixx,tdUnitVec,0)
  Tyy=sYY*iGainYY-np.tensordot(Tiyy,tdUnitVec,0)


  # Return the brigtness temperature for XX and YY, preceded by the time-axis
  # in MJD, as a tupple.
  return (mjd_tgt,Txx,Tyy)


#-------------------------------------------------------------------------
#                       4. Interface to ATM model
#-------------------------------------------------------------------------

def setAtmTool(myAtm,vis,spw,scan,
               myMsmd=None,wvHeightScale=1.5,tempDropPerKm=-5.6):

  """
  Set the atmosphere tool (casatools::at) *myAtm* for later calculation of the
  opacity (wet & dry component) as a function of frequencies for the single
  spectral window *spw*. The *spw* may be either one of the AC-resolved sci SPWs
  or the SPW=4 for the WVR data. The attool effectivelly represents CASA
  interface to the underlying terrestrial-atmosphere radiative-transfer model
  ATM by J.R. Pardo.

  Input:
  The spw setup and the atmosphere (meteo) profile model are extracted
  from the input measurement set *vis*. Parameter *scan* determines (i) the
  time-range, for which we seek the meteo parameters, and (ii) the target
  object and its elevation (H) - we need it for the airmass (AM =  1/sin(H))
  calculation. The WV height scale and temperature drop per 1km are
  set to their default values, which give the best fit between our PWVs
  as calculated by pwvValc() from the raw WVR data and the SYSCAL PWV values
  for the 2023 Band 5 data - see pwvCalcVerify() for details.


  Output:
  The atmTool *myAtm* gets modified in this routine so that it is ready for
  further use in, e.g.,  wopacity().
  With this flexibility, we can later force the PWV value to be set by a user.

  Our attool interface has been loosely inspired by aU.plotAtmosphere().
  The aU.plotAtmosphere() utility, however, uses aU.getMedianPWV(vis)
  eventhough its *pwv* arg is set, if the vis argument is present, so the pwv
  argument is actually ignored there. Moreover, aU.plotAtmosphere() returns
  just the representative-frequency (scalar) values; we need the frequency-
  dependent opacity values (see wopacity()).

  See aU::plotAtmosphere() and casatools::at for details.
  """

  #-- Get info about the SPW definition, weather, and target elevation.
  #   We need msmd tool attached to our MS for that.

  # Local msmd-tool flag.
  ownMsmd=False
```

```
if (not myMsmd):
  # Open the ms tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True

#-- Weather & airmass (1/sin(elevation)) - per-scan specific.
#   We have to start with that and set the SPW later, reverted order
#   does not work!

# We use here the standard aU.getWeather() just for extracting target
# elevation in order to calculate airmass. For the meteo characteristics
# we use our (more accurate) approach in getMeteoForScan().

# getWeaher() returns a tupple, the weather dictionary is its 1st component.
wDict = (aU.getWeather(vis, scan=scan, mymsmd=myMsmd))[0]
# getMeteoForScan() returns directly a map/dictionary.
wDict2=getMeteoForScan(vis,scan,myMsmd)


# Elevation/airmass from aU.getWeather()
elevation = wDict['elevation']
airMass = 1/np.sin(np.radians(elevation))

# Meteo from getMeteoForScan()
humidity = wDict2['humidity']
temperature = wDict2['temperature']
pressure = wDict2['pressure']


#-- Telescope location & clima - ALMA-specific

# Default climatic model for ALMA( 'midLatitudeWinter' or 'tropical').
almaClima=aU.translateLatitudeClass('midLatitudeWinter')
# WV scale-height for ALMA [km].
h0=wvHeightScale
# Temperature drop per 1 km
tempGrad=tempDropPerKm

# Altitude of ALMA [m].
altitude = 5059.0

#-- Default ATM model params.

# Top altitude  of the atmosphere [km].
maxAltitude=60.0
# Presurre step [mbar].
dP=5.0
# Pressure-step factor [dimension-less].
dPm=1.1

#- These parameters are requested as quantities in at.initAtmProfile()

tempQa=qa.quantity(temperature,"K")
altQa=qa.quantity(altitude,"m")
pressQa=qa.quantity(pressure,"mbar")
h0Qa=qa.quantity(h0,"km")
```

```
maxAltQa=qa.quantity(maxAltitude,"km")
dPQa=qa.quantity(dP,"mbar")
tempGradQa=qa.quantity(tempGrad,"K/km")


#-- Setup the atmospheric vertical profile.
atmModel= myAtm.initAtmProfile(humidity=humidity,
                               temperature=tempQa,
                               altitude=altQa,
                               pressure=pressQa,
                               h0=h0Qa,
                               dTem_dh=tempGradQa,
                               maxAltitude=maxAltQa,
                               dP=dPQa,
                               atmType=almaClima,
                               dPm=dPm)



#-- SPW definition.

# Extract all SPW channels from metadata.
freqs = myMsmd.chanfreqs(spw)*1.0e-9 # ndarray, in GHz


#- We have to make a specific settings for SPW 4 - the WVR data.
#  Why? It has 4 channels, but they have different bandwidths (see Figure in
#  Appendix A6 (WVRs) of ALMA Technical Handbook.
#  Solution: We define extra ATM SPW for each of the 4 WVR channels.
#  The 'sub-channel' resolution in these ATM SPWs is arbitrary, but it helps
#  us to properly integrate over the band-width of each WVR channel.
#  For the science SPWs we are using the natural channel resolution of the
#  channel-resolved SPWs, even for the integrated ACs and SQLDs, these are
#  always mapped to corresponding resolved SPW in the same frequency domain.

if (spw == 4):
  #-- WVR SPW: 1 ATM SPW per 1 WVR channel, make 100 sub-channels in each.

  print("Specific setup for the WVR SPW.")

  # Extract the WVR channel widths.
  dfs = myMsmd.chanwidths(4)*1.0e-9 # ndarray, in GHz

  #- at.initSpectralWindow() requires some args as quantities.

  fCenterQa=qa.quantity(list(freqs),'GHz')
  fWidthQa=qa.quantity(list(dfs),'GHz')
  # Define 100 (arbitrary) virtual sub-channels per each WVR channel.
  dfQa=qa.quantity(list(0.01*dfs),'GHz')

  # Setup our spectral windows: Frequency-domain for the attool.
  myAtm.initSpectralWindow(len(freqs),fCenterQa,fWidthQa,dfQa)

else:
  #-- Sci SPW: 1 ATM SPW per 1 sci SPW, keep 128-channel resolution.

  # We are always mapped to channel-resolved SPWs (even for single-channel ACs
  # and SQLDs). Calculate the derived SPW characteristics.
```

```
    # Channel are evenly distributed for sci SPWs, so we can determine the
    # channel width as a difference of any two central channel frequencies.
    # If the spw has decreasing frequencies, we revert the order.
    df=abs(freqs[1]-freqs[0])
    # Ref.  rrequency is in the center of the SPW.
    refFreq = np.mean(freqs)

    # This is the full SPW width. The freqs[] array stores *central* frequencies
    # for each channel, so in order to reach the very SPW edges we have to
    # extend by the channel half-width at both sides.
    frequencyRange = [np.min(freqs)-0.5*df, np.max(freqs)+0.5*df]

    #- at.initSpectralWindow() requires some args as quantities.

    fCenterQa=qa.quantity(refFreq,'GHz')
    fWidthQa=qa.quantity(frequencyRange[1]-frequencyRange[0],'GHz')
    dfQa=qa.quantity(df,'GHz')

    # Setup our single spectral window: Frequency-domain for the attool.
    myAtm.initSpectralWindow(1,fCenterQa,fWidthQa,dfQa)
  #--> endelse(Sci SPW)

  # Close the msmd tool if local.
  if (ownMsmd): myMsmd.done()

  #-- Set the target (scan) specific airmass (cosec(elevation)).
  myAtm.setAirMass(airMass)

  """
 # Debug: Print SPW summary.

 nChans=myAtm.getNumChan(0)
 cWidth=myAtm.getChanSep(0)
 for c in range(nChans):
  cFreq=myAtm.getChanFreq(c,0)
   print("Channel ", c, ": central frequency = ",cFreq['value'][0],
        cFreq['unit'],", channel width: ", cWidth['value'][0],cWidth['unit'])

  """

  return


#-----------------------------------------------------------------------------

def setAtmTool4SPWs(myAtm,vis,scan,
                    myMsmd=None,wvHeightScale=1.5,tempDropPerKm=-5.6):
  """
  Similar to setAtmTool() but now it makes the setup for the 4 channel-resolved
  AC SPWS (i.e. no WVR spw possible to be set here).

  As result, *myAtm* has 4 spws 0 - 3 mapped from the sci SPWs 5,7,9, and 11.
  For the temperature, pressure and humidity profile we use the same setup as
  in setAtmTool().
  """

  #-- Get info about the SPW definition, weather, and target elevation.
```

```
#   We need msmd tool attached to our MS for that.

# Local msmd-tool flag.
ownMsmd=False

if (not myMsmd):
  # Open the ms tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True


#-- Weather & airmass (1/sin(elevation)) - per-scan specific.
#   We have to start with that and set the SPW later, reverted order
#   does not work!

# We use here the standard aU.getWeather() just for extracting target
# elevation in order to calculate airmass. For the meteo characteristics
# we use our (more accurate) approach in getMeteoForScan().

# getWeaher() returns a tupple, the weather dictionary is its 1st component.
wDict = (aU.getWeather(vis, scan=scan, mymsmd=myMsmd))[0]
# getMeteoForScan() returns directly a map/dictionary.
wDict2=getMeteoForScan(vis,scan,myMsmd)


# Elevation/airmass from aU.getWeather()
elevation = wDict['elevation']
airMass = 1/np.sin(np.radians(elevation))

# Meteo from getMeteoForScan()
humidity = wDict2['humidity']
temperature = wDict2['temperature']
pressure = wDict2['pressure']


#-- Telescope location & clima - ALMA-specific

# Default climatic model for ALMA( 'midLatitudeWinter' or 'tropical').
almaClima=aU.translateLatitudeClass('midLatitudeWinter')
# WV scale-height for ALMA [km].
h0=wvHeightScale
# Temperature drop per 1 km
tempGrad=tempDropPerKm

# Altitude of ALMA [m].
altitude = 5059.0

#-- Default ATM model params.

# Top altitude  of the atmosphere [km].
maxAltitude=60.0
# Presurre step [mbar].
dP=5.0
# Pressure-step factor [dimension-less].
dPm=1.1

#- These parameters are requested as quantities in at.initAtmProfile()
```

```
tempQa=qa.quantity(temperature,"K")
altQa=qa.quantity(altitude,"m")
pressQa=qa.quantity(pressure,"mbar")
h0Qa=qa.quantity(h0,"km")
maxAltQa=qa.quantity(maxAltitude,"km")
dPQa=qa.quantity(dP,"mbar")
tempGradQa=qa.quantity(tempGrad,"K/km")


#-- Setup the atmospheric vertical profile.
atmModel= myAtm.initAtmProfile(humidity=humidity,
                               temperature=tempQa,
                               altitude=altQa,
                               pressure=pressQa,
                               h0=h0Qa,
                               dTem_dh=tempGradQa,
                               maxAltitude=maxAltQa,
                               dP=dPQa,
                               atmType=almaClima,
                               dPm=dPm)


#-- Setup the 4 ATM SPWs correcponding to chan-resolved ACs.

# Arrays for spw-defining parameters.
refFreq=np.zeros(4)
df=np.zeros(4)
bandWidth=np.zeros(4)

atmSpwIndex=0

for sciSPW in [5,7,9,11]:

  # Extract all SPW channels from metadata.
  freqs=myMsmd.chanfreqs(sciSPW)*1.0e-9 # ndarray, in GHz

  # Channels are evenly distributed for science SPWs, so we can determine the
  # channel width as a difference of any two central-channel frequencies.
  # If the spw has decreasing frequencies, we revert the order.
  df_cur=abs(freqs[1]-freqs[0])
  df[atmSpwIndex]=df_cur
  # Ref. frequency is in the center of the SPW.
  refFreq[atmSpwIndex] = np.mean(freqs)

  # This is the full SPW width. The freqs[] array stores *central* frequencies
  # for each channel, so in order to reach the very SPW edges we have to
  # extend it by the channel half-width at both ends.
  frequencyRange = [np.min(freqs)-0.5*df_cur, np.max(freqs)+0.5*df_cur]
  bandWidth[atmSpwIndex]=frequencyRange[1]-frequencyRange[0]

  atmSpwIndex+=1

#--> endfor (sci SPWs)


#- at.initSpectralWindow() requires some args as quantities. Convert
#  the np.appays to python lists.
```

```
  fCenterQa=qa.quantity(list(refFreq),'GHz')
  fWidthQa=qa.quantity(list(bandWidth),'GHz')
  dfQa=qa.quantity(list(df),'GHz')

  # Setup our spectral windows: Frequency-domain for the attool.
  myAtm.initSpectralWindow(4,fCenterQa,fWidthQa,dfQa)

  # Close the msmd tool if local.
  if (ownMsmd): myMsmd.done()

  #-- Set the target (scan) specific airmass (cosec(elevation)).
  myAtm.setAirMass(airMass)

  return


#-----------------------------------------------------------------------------

def wopacity(vis,spw,scan,userPWV=-1.0):

  """
  Uses the atmosphere tool (casatools::at) for calculation of the opacity
  (wet & dry component) as a function of frequencies for the science spectral
  window *swp*. We preset the atTool via setAtmTool().


  Return value: dictionary with the following components:
  airMass: 1/sin(h) for the object observed in the scan *scan*
  tau_1: 'Wet' component of the zenith opacity for PWV=1.0 as a function of
  frequency channels (1d-array).
  tau_d: Dry component of the zenith opacity as function of channels.
  tau_w: Wet zenith opacity as a fubction of channel for the actual PWV.
  T_atm: Effective temperature of the atmosphere (used in (1-trans)*Tatm expr.).
  """

  # This method is intended just for channel-resolved science SPWs.
  # Return immediatelly if we are trying to calculate it for other.

  if (spw not in [5,7,9,11]):
    print("wopacity() is intended just for resolved sci SPWs, which ",spw,
          " is not.")
    return

  #-- Open and set the atm tool.
  myAtm=attool()
  setAtmTool(myAtm,vis,spw,scan)


  #--- Calculate the opacities for PWV=1.0 (zenith value).

  pwv=1.0
  myAtm.setUserWH2O(qa.quantity(pwv,'mm'))

  #- Extract the opacities from the ATM model.

  # Dry zenith opacity
  tau_d = np.array(myAtm.getDryOpacitySpec(0)[1])
```

```
  # Water-vapour related 'wet' opacity for PWV=1.0
  tau_1 = np.array(myAtm.getWetOpacitySpec(0)[1]['value'])

  #--- We need also the T_atm - estimation of the effective gas temperature.
  #    This to be calculated for the actual PWV during the observation.
  #    PWV may be supplied by the user (e.g., per antenna). If not, we
  #    use median PWV over entire EB and array.

  if (userPWV>0.0):
    pwv=userPWV
  else:
    # Get the median PWV over entire array for entire EB duration.
    pwv=aU.getMedianPWV(vis)[0]

  # Set the actual PWV as to the ATM model.
  myAtm.setUserWH2O(qa.quantity(pwv,'mm'))

  #- Extract the opacities & friends from the ATM model.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(0)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(0)[1]['value'])

  # Brightness temperature (Effective Black-Body) of the athmosphere
  TebbSky = myAtm.getTebbSkySpec(spwid=0)[1]['value']


  # Extract the airmass for the current scan.
  airMass=myAtm.getAirMass()

  # Recalculate it to (effective) gas temparature via T_b = T_atm*(1-trans)
  opacity = airMass*(tau_wet+tau_dry)
  transmission = np.exp(-opacity)

  # Shape: 1D array of size nChans-1. Anyway, it is almost constant, may be
  # easilly represented by a single value (e.g., the mean).
  T_atm = TebbSky /  (1-transmission)

  #--- Compose the return value as a dictionary. Clean-up the atm tool

  atmProp={'airmass':airMass,'tau_d':tau_d,'tau_w':tau_wet,'tau_1':tau_1,
           'tr':transmission,'T_atm':T_atm}

  myAtm.done()

  return atmProp


#------------------------------------------------------------------------------

def setAtmosSciSPWs(vis,tgtScan,spwList):
  """
  Calculate the ATM profiles using wopacity() for all channel-resolved sci SPWs.
  Then, using the spw-mapping attribute the appropriate ATM profile to every
  science SPW (chan-resolved and integrated ACs and SQLDs) in the *spwList*.
  Use *tgtScan* for determining the approporate ATM model - we have scan-
  dependent meteo parameters and airmass.
```

```
  Used as a single-shot function in the methods that iterate over the antenna
  array, like saveSol_dWdB(): We do initiation of the atm model for all spws
  just once.
  """

  # Collect the sub-models for resolved spws as a map.
  atmModelRes={}

  # We do it just for the resolved SPWs
  for rspw in [5,7,9,11]:
    atmModelRes[rspw]=wopacity(vis,rspw,tgtScan)

  #-- Now we map any SPW to the res-spw atm model.

  # Use the *resolved* swps corresponding to our integrated ACs and SQLDs.
  spwRes={0:5,1:7,2:9,3:11,5:5,6:5,7:7,8:7,9:9,10:9,11:11,12:11}

  # Collect the sub-models for requested spws as a map.
  atmModel={}

  for spw in spwList:
    atmModel[spw]=atmModelRes[spwRes[spw]]

  return atmModel

#-----------------------------------------------------------------------------

def getPWVforAnt(vis,tgtScan,antID,myMsmd=None):
  """
  In wopacity() we set averaged PWV over entire array and entire EB duration.
  For estimation of T_atm (which depends on atmospheric transmissivity) this
  is probably OK.

  On the other hand, in dWdB() we have the matrix coefficents for (apparent)
  source brightness variations (dB) depending on the transmissivity, too, now
  calculated per-antenna. To that end, we extract the PWV values right
  before and right after the *tgtScan* for specific antenna *antID*. We
  use the MS/ASDM_CALWVR table to get the PWV per time and antenna. We find two
  PWV values that most closely (in time) embrace the *tgtScan* and return their
  average - value interpolated in time to the time-center of the *tgtScan*.
  Moreover, as a second value in the tupple, we return the linear slope of
  the PWV change during the *tgtScan*

  See also aU.readPWVFromMS() and related functions for explanation, how we
  extract the PWV data.
  """

  #-- Explore the MS metadata

  ownMsmd=False

  if (not myMsmd):
    myMsmd=msmdtool()
    myMsmd.open(vis)
    ownMsmd=True
```

```
    # Scan timing
    scanInfo=aU.computeDurationOfScan(tgtScan,mymsmd=myMsmd,
                                      returnSubscanTimes=True)

    # Antenna ID to name conversion.
    antNames=myMsmd.antennanames()
    antName=antNames[antID]

    if (ownMsmd): myMsmd.done()


    #-- Query the WV data

    # Open the MS/ASDM_CALWVR table
    wvTab = tbtool()
    wvTab.open(vis+"/ASDM_CALWVR")

    # Select just the data for antID, the result is a sub-table.
    wvAntTab=wvTab.query(query="antennaName==\'"+antName+"\'",
                         sortlist="startValidTime",
                         columns="startValidTime,water")

    # Extract time and pwv for the antenna.
    pwvTimes = wvAntTab.getcol('startValidTime')
    pwv = wvAntTab.getcol('water')

    # Close both the per-antenna subtable and the main WV table.
    wvAntTab.close()
    wvTab.done()

    #-- Find the time-range for the tgtScan

    # Num. of subscans in the tgtScan.
    nSubScans=scanInfo[1]

    # First and last subscan: The result is a map starting from 1, not an array.
    ssTimesFirst=scanInfo[2][1]
    ssTimesLast=scanInfo[2][nSubScans]

    # The sub-scan times are 2-element arrays - subscans beginning and end.
    # The full-scan beginning is the beginning of the first sub-scan.
    tBeg=ssTimesFirst[0]
    # The full-scan end is the end of the last sub-scan.
    tEnd=ssTimesLast[1]

    #-- Interpolate PWV to the tgtScan range. Find scan-center average and slope.

    # Interpolated PWV in milimmeters (original value is in meters).
    pwvTgt=1000.0*np.interp([tBeg,tEnd],pwvTimes,pwv)

    # Scan-center average PWV.
    pwvAvg=0.5*(pwvTgt[0]+pwvTgt[1])

    # PWV linear time-trend / slope.
    pwvSlope=(pwvTgt[1]-pwvTgt[0])/(tEnd-tBeg)

    #-- Return the average, time-slope, and the full-PWV array as a map.
    return {'pwvAvg':pwvAvg,'pwvSlope':pwvSlope,'allPWVs':(pwvTimes,pwv)}
```

```
#------------------------------------------------------------------------------

def getMeteo(vis):
  """
  Get time series of the basic meteo characteristics. They are not per antenna
  as there is just a single meteo station for entire array. Still we have much
  better time coverage than the ASDM_CALATMOPHERE table. The latter has,
  seemingly, pr-antenna values but in fact they are the same for all antennas
  and differ only for different times. There are much less time samples in
 ASDM_CALATMOSPHERE than in WEATHER table.

  We return the meteo characteristics as a time series in a map.
  """

  #-- Extract the METEO data

  # Open the MS/ASDM_CALWVR table
  meteoTab = tbtool()
  meteoTab.open(vis+"/WEATHER")

  # Extract time and meteo quantities
  time=meteoTab.getcol('TIME')


  # As described in aU.getWeather(), the times may not be sorted in increasing
  # order and we do not use tb.query() here, which normally assures orderring.
  # We have to do sorting by increasing time manually.

  # Find the permuted indices.
  pIndex = np.argsort(time)

  # Apply the same permutation to all arrays.
  time=time[pIndex]
  pressure=meteoTab.getcol('PRESSURE')[pIndex]
  humidity=meteoTab.getcol('REL_HUMIDITY')[pIndex]
  temperature=meteoTab.getcol('TEMPERATURE')[pIndex]

  # Close the table.
  meteoTab.done()

  # Return the meteo characteristics
  return {'time': time, 'pressure': pressure, 'humidity':humidity,
          'temperature':temperature}

#------------------------------------------------------------------------------

def getMeteoForScan(vis,scan,myMsmd=None):
  """
  The meteo characteristics as extracted by getMeteo() form the MS *vis*, now
  averaged and interpolated to the center of the *scan*. The msmd tool *myMmsmd*
  may be conveniently supplied if it is already attached to *vis* - it improves
  performance.

  Basically, it is intended as a more accurate replacement for aU.getWeather().
  What is the difference between the two? The aU.getWeather() finds the meteo
  values 'bracketted' by the beginning and end of the scan and makes their
  average. Thsi approach forgets even the recent history - if there is a meteo
```

```
measurement point just a second before the scan begins, its value is
neglected. The same holds on the 'future' end. In contrast, getMeteoForScan()
samples the scan by 100 time samples, for each of them makes linear
interpolation of the meteo values and eventually, this array is averaged.
This approach takes into account the pre- and post-scan meteo measurements,
too, with their weights naturally set by the time distance of the meteo
record from the actula time-sample of the scan.

For the solar scans the tiny differences are not so important, the signal is
dominated by the brightness temparature of the Sun, the T_atm correction is
negligible. On the other hand, for the BP scans, the (T_sky-T_atm) expression,
which stands in the sensitivity factor for dPWV, has a typical value about 3K,
so difference in T_atm of 0.5K makes ~20% error!

"""

#-- Find the scan timing.

#- Msmd tool for scan-info extraction.

# Flag indicating, whether we had to create our own msmd tool for scan timing.
ownMsmd=False

if (myMsmd==None):
  # No msmd tool supplied, create a new one and attach it to our MS.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  # Set the flag
  ownMsmd=True
  print('msmdtool attached to ',vis)

scanInfo=aU.computeDurationOfScan(scan,mymsmd=myMsmd,
                                  returnSubscanTimes=True)

# Close only our own local msmd tool, if we have used the 'global' one,
# leave it as is.
if (ownMsmd): myMsmd.done()

#- Analyse the scan-info results.

# Num. of subscans in the tgtScan.
nSubScans=scanInfo[1]

# First and last subscan: The result is a map starting from 1, not an array.
ssTimesFirst=scanInfo[2][1]
ssTimesLast=scanInfo[2][nSubScans]

# The sub-scan times are 2-element arrays - subscans beginning and end.
# The full-scan beginning is the beginning of the first sub-scan.
tBeg=ssTimesFirst[0]
# The full-scan end is the end of the last sub-scan.
tEnd=ssTimesLast[1]

# Create scan time-sampling axis (100 samples per scan).
scanTimes=tBeg+0.01*(tEnd-tBeg)*np.arange(101.0)

#-- Get meteo time series.
```

```
  meteo=getMeteo(vis)

  meteoTimes=meteo['time']
  pressure=meteo['pressure']
  humidity=meteo['humidity']
  temperature=meteo['temperature']

  #-- Interpolate the meteo characteristics to the scan time axis.

  sPressure=np.interp(scanTimes,meteoTimes,pressure)
  sHumidity=np.interp(scanTimes,meteoTimes,humidity)
  sTemperature=np.interp(scanTimes,meteoTimes,temperature)

  #-- Return the scan mean values.

  return {'pressure':sPressure.mean(),
          'humidity':sHumidity.mean(),
          'temperature':sTemperature.mean()}

#------------------------------------------------------------------------------


def wv2path(vis,spw,scan,pwv):
  """
  We use the factor -25.0 for conversion between dW and dPhi, as it implies
  from the Richard Hill's original draft. (RH uses effective refractive index
  r_eff=6.7, independent on frequency. When the optical path is recalculated
  to phase delay for average lambda ~1.5 mm, we arrive to factor -25.0).

  This routine verifies, whether this assumption is correct and what are its
  limitations. We use ATM modela and its get***Path() routines.

  Result: Indeed, the non-dispersive part is constant over SPW 5 and its value
  is ~6.7e-3 m(!) per 1 mm of PWV. The dispresive part is very slightly
  changing around its average value of 1.0e-4 m per 1 mm PWV and thus,
  represents a small correction. Both paths depend linearly on pwv, so
  their effective (per 1 mm PWV) value does not depend on the *pwv* parameter.
  On the other hand, they slightly depend on the airmass (i.e. on the *scan*
  parameter). With rising SPW the total (disp.+ non-dispersive) path per 1mm
  rises (up to ~6.9e-3 m per 1 mm PWV). So, for approximate calculations it is
  fine to use the factor of approx -25 (actually -27). However, for real
  application, where we use correct central frequencies for each SPW, it is
  better to calculate the phase-delays directly from the ATM model.
  """

  #-- Open and set the atm tool.
  myAtm=attool()
  setAtmTool(myAtm,vis,spw,scan)

  # Set the PWV for the model.
  myAtm.setUserWH2O(qa.quantity(pwv,'mm'))

  #-- Calculate the paths per channel relative to 1mm of PWV.

  nChans=myAtm.getNumChan(0)

  # Allocate the arrays for dispersive and non-dispersive paths.
  pDisp=np.zeros(nChans)
```

```
  pNonDisp=np.zeros(nChans)


  for ch in range(nChans):

    pDisp[ch]=myAtm.getDispersivePathLength(nc=ch)['value'][0]/pwv
    pNonDisp[ch]=myAtm.getNonDispersivePathLength(nc=ch)['value'][0]/pwv

  # Close the ATM tool.
  myAtm.done()

  # Return dipersive and non-dispersive paths in mm as a tupple.
  return 1000.0*pDisp,1000.0*pNonDisp


#--------------------------------------------------------------------------
#               5. Calculate the proxies of differential PWV
#--------------------------------------------------------------------------

def dPWV(vis,atmScan,tgtScan,antID,spwAvg=6,spwRes=5):
  """
  Calculate dPWV - our proxy to differential PWV (water-vapour column) based on
  the variations of received power. See formulae (13) and (14) in the Progress
  Report no. 11 (March 27, 2025). We may use both the auto-correlation (AC)
  channel-integrated SPW (i.e. SPW 6 in solar case) or, almost equally, the
  lowest SQLD spw (SPW 0) for that purpose. What spw is actually used is
  controlled by *spwAvg* parameter.
  The *tgtScan* parameter defines, for what object is dPWV calculated, the
  *atmScan* argument is the scan used for ACD calibration (see acd_calTabs();
  atmScan should precede the tgtScan as close as possible, ideally by one).
  Antenna, for which dPWV is calculated, is represented by its integer *antID*.

  Returns: A tupple of three 1-D arrays, each of the size of nIntegs: First
  represents the time-axis as MJD (merely reproduction of the axis returned by
  tgt_pow()), the other two are values of dPWV for XX and YY polarizations,
  respectivelly.
  """


  #-- Power temporal variations.

  # Get the measured power, preceded by times as MJD.
  mjd_src,Psrc=tgt_pow(vis,tgtScan,antID,spwAvg)

  # 1st index is pol., 2nd means channel num. We have just a single channel.
  Pxx=Psrc[0][0]
  Pyy=Psrc[1][0]

  # Subtract the time-averaged value over scan to get the variations only.
  PxxM=Pxx.mean()
  PyyM=Pyy.mean()

  dPxx=Pxx-PxxM
  dPyy=Pyy-PyyM

  #-- Calculate the "sensitivity factor S"

  #- Find G/G*, Trx, and Tinst
```

```
ctACD=acd_calTabs(vis,atmScan,antID,spwAvg)

iG=ctACD['iG']
iGstar=ctACD['iG*']
T_inst=ctACD['T_inst']


# What is the target? Should we use the solar-attenuation gain G* or
# the 'standard' G?
if (iGstar[0][0][0] == 1.0):  # G* not set (has its default value) -> not Sun

  print('Solving for CALIBRATOR scan.')

  # G* was not calculated => non-solar target, use (inverse) G.
  # 1st index is pol., 2nd means channel num. We have just a single channel.
  iGx=iG[0][0]
  iGy=iG[1][0]

else:

  print('Solving for SOLAR scan.')

  # Solar target-scan: Use (inverse) G*, make an average from the values
  # derived for the beginning and the end of the scan.
  # 1st index is 'time' (just begin and end), 2nd means pol. and 3rd the
  # channel number. We have just a single channel.
  iGx=0.5*(iGstar[0][0][0]+iGstar[1][0][0])
  iGy=0.5*(iGstar[0][1][0]+iGstar[1][1][0])


Gx=1.0/iGx
Gy=1.0/iGy

# The 'instrumental' temperature (T_RX + contribution from the SQLD offset).
T_iXX=T_inst[0]
T_iYY=T_inst[1]


#- Find spectral slope (linear) for the source (+atm.) brightness temperature.

# Calculate the brightness temperature as a function of frequency and
# time using ACD calibration.
mjd_tb,Txx,Tyy=tgt_Tb(vis,atmScan,tgtScan,antID,spwRes)

# We use time-averaged (over a scan) values for the slope determination.
Txm=Txx.mean(axis=1)
Tym=Tyy.mean(axis=1)

# Trim the edge channels, these are always problematic.
nChans=len(Txm)

TxM=Txm[7:nChans-8]
TyM=Tym[7:nChans-8]

# Find the corresponding frequency axis. Trim the edges accordingly.
msmd.open(vis)
freqs = myMsmd.chanfreqs(spwRes)*1.0e-9 # in GHz
msmd.done()
```

```
fr=freqs[7:nChans-8]
# Find the entire SPW bandwidth [GHz].
BW=np.max(freqs)-np.min(freqs)

# Use linear (poly=1) regression to find the slope.
dTxx_df=np.polyfit(fr,TxM,1)[0]
dTyy_df=np.polyfit(fr,TyM,1)[0]

#-- Find the opacities and T_atm using the ATM model.

atmParams=wopacity(vis,spwRes,tgtScan)

# 1/sin(h) for the target.
airMass=atmParams['airmass']

# Effective gas temperature of the atmosphere.
T_atm=atmParams['T_atm'].mean()

# Zenith wet opacity for PWV=1.0.
tau_1z=atmParams['tau_1']

# The same quantity scaled by airmass.
tau_1=airMass*tau_1z

#- Extract its mean value and the ranges for high- and low-frequency ends.

tau_1M=tau_1.mean()

# Highest-fequency channel
hiChan=tau_1.size-1

# Opacity span over the SPW.
tau_1Hi=tau_1[hiChan]
tau_1Lo=tau_1[0]

#-- Now we can calculate the S-factors for each polarization extra.
SFxx=-Gx*tau_1M*(iGx*PxxM-T_iXX-T_atm)-Gx*BW/12.0*(tau_1Hi-tau_1Lo)*dTxx_df
SFyy=-Gy*tau_1M*(iGy*PyyM-T_iYY-T_atm)-Gy*BW/12.0*(tau_1Hi-tau_1Lo)*dTyy_df

# Diagnostic prints: Find out, how much each term contributes.


print('Psrc/G = Tsrc = ',iGx*PxxM,iGy*PyyM)
print('Tinst = ',T_iXX,T_iYY)
print('Tatm = ',T_atm)
print('dTsky/df = ',dTxx_df,dTyy_df)
print('SF_corr = ',Gx*BW/12.0*(tau_1Hi-tau_1Lo)*dTxx_df,
      Gy*BW/12.0*(tau_1Hi-tau_1Lo)*dTyy_df)

print('SF = ',SFxx,SFyy)


#-- Eventually, get the diferential modelled PWVs.

dPWVxx=dPxx/SFxx
dPWVyy=dPyy/SFyy

# Return them as a tupple.
```

```
  return (mjd_src,dPWVxx,dPWVyy)


#------------------------------------------------------------------------------


def dPWV_spwPart(vis,atmScan,tgtScan,antID,spwRes=5,startChan=8,stopChan=120):
  """
  Find dPWV - our proxy to the WV contents but now by using just a part
  of the SPW. Unlike in dPWV() we now use the channel-resolved SPW *spwRes*,
  by default that one closest to the WV line center (i.e. SPW 5).

  Because the single-channel data are too noisy (that is the reason why
  spw-integrated ACs or SQLDs are being used), we do integration over multiple
  channels, but not entirely over full SPW. The frequency range for integration
  is defined by *startChan* and *stopChan* parameters, which default to the
  first and last 'usable' channel (the firts and last 8 channels are typically
  flagged in any analysis). Note: For SPWs with reversed order the start/stop
  chan is also swapped  so that the startChan always means the low-frequency
  boundary.

  Idea: We may avoid O3 lines or use just that part of the SPW 5 closer to the
  WV line center and increase sensitivity to the WV contents. Suggested on PM12
  in May 2025 (see PR12 - post meeting Action item 6), implemented in line
  with formula (1) in PR13 (August 2025).
  """

  #-- Power temporal variations.

  # Get the measured power, preceded by times as MJD.
  # Shape of the power Psrc is nPols x nChans x nIntegs
  time_src,Psrc=tgt_pow(vis,tgtScan,antID,spwRes)

  # Separate polarizations.
  Pxx=Psrc[0]
  Pyy=Psrc[1]

  # Find the frequency and time dimensions.
  nChans=Pxx.shape[0]
  nIntegs=Pxx.shape[1]

  #- Subtract <P>, the time-averaged value over scan to get the variations only.

  # Calculate <P>. result is a vector of size nChans.
  PxxM=Pxx.mean(axis=1,dtype=np.float64)
  PyyM=Pyy.mean(axis=1,dtype=np.float64)

  # We need the <P> vector (in frequency) to be a matrix (const in time).
  # Define the time-dimension const (unit) vector.
  tdUnitVec=np.ones(nIntegs)
  # Perform the outer (tensor) product.
  PxxMM=np.tensordot(PxxM,tdUnitVec,0)
  PyyMM=np.tensordot(PyyM,tdUnitVec,0)

  # Now we can subtract - the result is a matrix with shape nChans x nIntegs
  dPxx=Pxx-PxxMM
  dPyy=Pyy-PyyMM
```

```
#- Find G/G*, Trx, and Tinst
ctACD=acd_calTabs(vis,atmScan,antID,spwRes)

# iG and T_rx are of shape nPols x nChans, iGstar (solar) nPols x nChans x 2,
# the last dimension of size 2 representing 2 time instants at which the
# gain is measured (the 1st and last sub-scans of the solar scan).
iG=ctACD['iG']
iGstar=ctACD['iG*']
T_rx=ctACD['T_rx']

# What is the target? Should we use the solar-attenuation gain G* or
# the 'standard' G?
if (iGstar[0][0][0] == 1.0):  # G* not set (has its default value) -> not Sun

  print('Solving for CALIBRATOR scan.')

  # G* was not calculated => non-solar target, use (inverse) G.

  iGx=iG[0]
  iGy=iG[1]

else:

  print('Solving for SOLAR scan.')

  # Solar target-scan: Use (inverse) G*, make an average from the values
  # derived for the beginning and the end of the scan.
  iGx=0.5*(iGstar[0][0]+iGstar[1][0])
  iGy=0.5*(iGstar[0][1]+iGstar[1][1])

# The 'instrumental' temperature - just T_RX in case of the AC SPW.
T_iXX=T_rx[0]
T_iYY=T_rx[1]

#-- Find the opacities and T_atm using the ATM model.

atmParams=wopacity(vis,spwRes,tgtScan)

# 1/sin(h) for the target.
airMass=atmParams['airmass']

# Effective gas temperature of the atmosphere.
# We use the scalar value representing the almost constant array - see
# the comment in wopacity(). Keep it constant but still a vector in frequency.
T_atm=atmParams['T_atm'].mean()*np.ones(nChans)

# Zenith wet opacity for PWV=1.0.
tau_1z=atmParams['tau_1']

# The same quantity scaled by airmass.
tau_1=airMass*tau_1z

#-- We are going to integrate (sum) over the defined frequency range.
#   Some SPWs have, however, reversed frequency axis. For P, T_RX and G/G*
#   this is not an issue (we are just summing backwards). Nevertheless,
#   tau_1 has been calculated by atm model in wopacity() with the frequency
#   axis always in increasing order. Hence, in reversed SPWs we reverse tau_1,
```

```
#   too, in order to make all values having the same per-channel order.
#   Moreover, we swap the args startChan/stopChan in reversed SPWs, so that
#   they keep their intuitive meaning (corresponding to increasing frequency).

# Check the SPW definition.
myMsmd=msmdtool()
myMsmd.open(vis)
freqs = myMsmd.chanfreqs(spwRes)
myMsmd.done()

lastChan=len(freqs)-1

if (freqs[1] < freqs[0]):
  # Reversed-order SPW
  print("Reversed SPW, swapping start/stop channels.")
  startChan,stopChan = lastChan-stopChan,lastChan-startChan

  tau_1=np.flip(tau_1)


#-- In spectraly-resolved SPWs the gain/Trx may live in a different freguency
#   domain than the data - see acd_calTabs(). We have to intrepolate first.
#   Note: tau_1 has the same freq. domain as science data as set in wopacity()
#   just it may be in reversed order (solved above by flipping).


# Reveal the frequency domain for gains.
gainFreqs=ctACD['freqs']

# Just a very basic frequency-domain compatibility check. @todo If it does not
# work, check yet f[0] and df.
if (len(freqs) != len(gainFreqs)):

  print('Interpolating gain/T_rx from ',len(gainFreqs),' to ',len(freqs),
        ' frequency channels.')

  # Gain and target spw definitions differ: Do interpolate.
  iGx=np.interp(freqs,gainFreqs,iGx)
  iGy=np.interp(freqs,gainFreqs,iGy)

  Trx=np.interp(freqs,gainFreqs,Trx)
  Try=np.interp(freqs,gainFreqs,Try)


#-- Apply the integration limits to all relevant arrays.

freqs=freqs[startChan:stopChan]

iGx=iGx[startChan:stopChan]
iGy=iGy[startChan:stopChan]

Ti_XX=T_iXX[startChan:stopChan]
Ti_YY=T_iYY[startChan:stopChan]


dPxx=dPxx[startChan:stopChan]
dPyy=dPyy[startChan:stopChan]
```

```
  PxxM=PxxM[startChan:stopChan]
  PyyM=PyyM[startChan:stopChan]

  tau_1=tau_1[startChan:stopChan]
  T_atm=T_atm[startChan:stopChan]


  #-- Calculate the dPWV by formula (1) in PR13 (August 2025).

  # Again, we need the iG (inv. gain) vector (in freq.) as a matrix - see above.
  iGxM=np.tensordot(iGx,tdUnitVec,0)
  iGyM=np.tensordot(iGy,tdUnitVec,0)

  # Frequency-integrated sky-temperature (not power like in dPWV()) variations.
  dTxx=np.sum(iGxM*dPxx,axis=0)
  dTyy=np.sum(iGyM*dPyy,axis=0)

  # The sensitivity factors. They differ by G-factor from those in dPWV().
  SF_XX=-np.sum((iGx*PxxM-Ti_XX-T_atm)*tau_1)
  SF_YY=-np.sum((iGy*PyyM-Ti_YY-T_atm)*tau_1)

  dPWVxx=dTxx/SF_XX
  dPWVyy=dTyy/SF_YY

  # Return them as a tupple, preceded by time.

  return (time_src,dPWVxx,dPWVyy)



#--------------------------------------------------------------------------------


def dWdB(vis,atmScan,tgtScan,atmModel,antID,spwList):
  """

  *** The main routine for calculation of the dW(t) for solar scans.***

  For the BP calibrator we may use dPWV() or its dPWV_spwPart() variant as there
  is no further source of signal variations except of the atmopsheric column
  mass changes manifested as variations in PWV.

  For the Sun, this is, however, not true - it can have its own brightness
  changes. Here we attempt to decompose the signal variations using a collection
  of science SPWs in *spwList* (may be AC integrated, AC resolved, SQLD or any
  combination, which incorporates at least two frequency domains) into:

  (i) dW = total variations of the PWV - like in dPWV, and
  (ii) dB = variations of the (apparent) source brightness.

  The latter may involve both intrinsic changes of Tb_src and the lensing
  effects of the turbulent atmosphere (a.k.a. caustics in low-frequency radio) -
  these two are not easilly mutually distinguishable.

  In order to do the disentagling, we need to use multiple SPWs.
  In the single-channel SPWs we omit the correction implying from the slope of
  Tsky(f) and tau(f), it has been experimentally found to be less then 5% for
  dPWV().
```

```
The *atmModel* is a set of static 'tabelised' profiles (frequency-dependent)
for opacities, Tatm, etc., as returned by wopacity(). We have it as an array
for all science channel-resolved SPWs (5,7,9, and 11). It is supplied as an
parameter because it is the same for all antennas: We do not want to
re-creaate it inside this routine whenever it is called for another antenna,
as it is usually done in a loop over entire array, e.g., in saveSol_dWdB().
When this routine is invoked in a loop for all array, the *atmModel* can be
conveniently pre-set by call to setAtmosSciSPWs().


The overdetermined system (we have just two variables - dW and dB, and up
to 12 equations for 12 SPWs) is then solved by the least-square principle,
leading to a minimizer of a quadratic form (which is another linear system of
eqs., but now regular one: 2x2 matrix with a nTimeSamples-long collection of
2-component RHS vectors).

Least-square residuals and other analytics are plotted in a compound 2 x 3
panels graph as a by-product of calculations. In the './dWdB/' subdir we plot
the solutions dW(t) and dB(t).

Commented-out is a perspective extension, which tries to decompose the
total dW variations into two parts, caused by (1) relative-humidity changes,
and (2) total air-preessure variations. It has been experimentally found,
however, that the system is then ill-conditioned for this decomposition. It
can be improved by changing the vector basis and decomposing into total wet
and total dry variations. However, even with this setup, the total wet
contribution to the signal remains almost unchanged (so it is quite a robust
quantity), the total dry component exhibits a large error and cannot be used.

@todo: In the ASDM_CALATMOPHERE table we have Tatm (a single value) per
SPW. Try to compare its value with our Tatm as calculated by wopacity().
"""

#-- Static parameters

# Reference frequency for the source spectrum - the LO frequency (in Hz)
f_LO=1.98e11

# Spectral index for the source. Spix=2.0 corresponds to R-J law.
# Spix=0.0 represents 'gray' enhancements due to atmospheric-bubble lensing.
spix=2.0

# Integration limits for SPWs. To be done more selective, e.g., to exclude
# stratospheric lines like O3.

startChan=7
stopChan=120

#-- Average PWV over a scan per antenna: Required for tau calculations.

pwvMean=getPWVforAnt(vis,tgtScan,antID)['pwvAvg']


#-- Matrices (so far empty placeholders) for sets of linear equations.

Axx=np.array([])
Ayy=np.array([])
```

```
#-- RHS vectors (so far empty placeholders) for sets of linear equations.
Cxx=np.array([])
Cyy=np.array([])

#-- Loop over the SPWs.

spwIndex=0

for mySPW in spwList:

  #-- Raw powers: Temporal variations.

  # Get the measured power, preceded by times as MJD.
  mjd_src,Psrc=tgt_pow(vis,tgtScan,antID,mySPW)

  # Separate polarizations.
  Pxx=Psrc[0]
  Pyy=Psrc[1]

  # Find the frequency and time dimensions.
  nChans=Pxx.shape[0]
  nIntegs=Pxx.shape[1]

  #- Subtract <P>, the time-average over a scan to get the variations only.

  # Calculate <P>. Result is a vector of size nChans.
  PxxM=Pxx.mean(axis=1,dtype=np.float64)
  PyyM=Pyy.mean(axis=1,dtype=np.float64)

  # We need the <P> vector (in frequency) to be a matrix (const in time).
  # Define the time-dimension const (unit) vector.
  tdUnitVec=np.ones(nIntegs)
  # Perform the outer (tensor) product.
  PxxMM=np.tensordot(PxxM,tdUnitVec,0)
  PyyMM=np.tensordot(PyyM,tdUnitVec,0)

  # Now we can subtract - the result is a matrix with shape nChans x nIntegs
  dPxx=Pxx-PxxMM
  dPyy=Pyy-PyyMM


  #-- Calibration to Tsky: Find G/G* and Trx/Tinst

  ctACD=acd_calTabs(vis,atmScan,antID,mySPW)

  # iG and T_rx are of shape nPols x nChans, iG* (solar) nPols x nChans x 2,
  # the last dimension of size 2 representing 2 time instants at which the
  # gain is measured (the 1st and last sub-scans of the solar scan).
  iG=ctACD['iG']
  iGstar=ctACD['iG*']
  T_inst=ctACD['T_inst']


  # What is the target? Should we use the solar-attenuation gain G* or
  # the 'standard' G?
  if (iGstar[0][0][0] == 1.0):  # G* not set (has its default value) -> not Sun
```

```
  print('Solving for CALIBRATOR scan.')

  # G* was not calculated => non-solar target, use (inverse) G.
  iGx=iG[0]
  iGy=iG[1]

else:

  print('Solving for SOLAR scan.')

  # Solar target-scan: Use (inverse) G*, make an average from the values
  # derived for the beginning and the end of the scan.
  iGx=0.5*(iGstar[0][0]+iGstar[1][0])
  iGy=0.5*(iGstar[0][1]+iGstar[1][1])
#--> endif(solar)

# The 'instrumental' temperature - just T_RX in case of the AC SPWs.
T_iXX=T_inst[0]
T_iYY=T_inst[1]


#-- Atmosphere: Find the opacities and T_atm using the ATM model.

# 1/sin(h) for the target.
airMass=atmModel[mySPW]['airmass']

# Effective gas temperature of the atmosphere.
# T_atm=atmModel[mySPW]['T_atm'].mean()*np.ones(nChans)
T_atm=atmModel[mySPW]['T_atm']

# Zenith wet opacity for PWV=1.0.
tau_1z=atmModel[mySPW]['tau_1']

# Zenith wet opacity for the actual PWV.
tau_wz=atmModel[mySPW]['tau_w']

# Zenith dry opacity.
tau_dz=atmModel[mySPW]['tau_d']


# Wet unit opacity scaled by airmass.
tau_1=airMass*tau_1z
# Wet actual opacity scaled by airmass.
tau_w=airMass*tau_wz
# Dry opacity scaled by airmass.
tau_d=airMass*tau_dz
# Total opacity scaled by airmass. Do NOT use tau_w - it is an average
# over entire ALMA array, see wopacity(). We rather use meanPwv per antenna
# and tau1.
tau_t=pwvMean*tau_1+tau_d


#-- Check the SPW definition. We branch for resolved and averaged SPWs.

myMsmd=msmdtool()
myMsmd.open(vis)
freqs = myMsmd.chanfreqs(mySPW)
```

```
myMsmd.done()

nChans=len(freqs)

if (nChans > 1):

  #-- This is a channel-resolved SPW.

  # We are going to integrate (sum) over the defined frequency range.
  # Some SPWs have, however, reversed frequency axis. For P, T_RX and G/G*
  # this is not an issue (we are just summing backwards). Nevertheless,
  # tau_* have been calculated by atm model in wopacity() with the frequency
  # axis always in increasing order. Hence, in reversed SPWs we reverse tau
  # too, in order to make all values having the same per-channel order.
  # Moreover, we swap the args startChan/stopChan in reversed SPWs, so that
  # they keep their intuitive meaning (corresponding to increasing
  # frequency).


  lastChan=nChans-1

  if (freqs[1] < freqs[0]):
    # Reversed-order SPW
    print("Reversed SPW, swapping start/stop channels.")
    startChan, stopChan = (lastChan-stopChan),(lastChan-startChan)

    tau_1=np.flip(tau_1)
    tau_w=np.flip(tau_w)
    tau_d=np.flip(tau_d)
    tau_t=np.flip(tau_t)
    T_atm=np.flip(T_atm)

  # In resolved SPWs the gain/Trx may live in a different freguency
  # domain than the data - see acd_calTabs(). We have to intrepolate first.
  # Note: tau_* have the same freq. domain as science data as set in
  # wopacity() just it may be in reversed order (solved above by flipping).

  # Reveal the frequency domain for gains.
  gainFreqs=ctACD['freqs']

  # Just a very basic frequency-domain compatibility check. @todo If it does
  # not work, check yet f[0] and df.
  if (nChans != len(gainFreqs)):
    print('Interpolating gain/T_rx from ',len(gainFreqs),' to ',len(freqs),
          ' frequency channels.')

    # Gain and target spw definitions differ: Do interpolate.
    iGx=np.interp(freqs,gainFreqs,iGx)
    iGy=np.interp(freqs,gainFreqs,iGy)

    Ti_XX=np.interp(freqs,gainFreqs,Ti_XX)
    Ti_YY=np.interp(freqs,gainFreqs,Ti_YY)


  #-- Apply the integration limits to all relevant arrays.

  freqs=freqs[startChan:stopChan]
```

```
    iGx=iGx[startChan:stopChan]
    iGy=iGy[startChan:stopChan]

    Ti_XX=T_iXX[startChan:stopChan]
    Ti_YY=T_iYY[startChan:stopChan]


    dPxx=dPxx[startChan:stopChan]
    dPyy=dPyy[startChan:stopChan]

    PxxM=PxxM[startChan:stopChan]
    PyyM=PyyM[startChan:stopChan]

    tau_1=tau_1[startChan:stopChan]
    tau_w=tau_w[startChan:stopChan]
    tau_d=tau_d[startChan:stopChan]
    tau_t=tau_t[startChan:stopChan]
    T_atm=T_atm[startChan:stopChan]


    #-- Partial sensitivity factors for reolved SPWs

    # Again, we need the iG (inv. gain) vector (in freq.) as a matrix.
    iGxM=np.tensordot(iGx,tdUnitVec,0)
    iGyM=np.tensordot(iGy,tdUnitVec,0)

    # Frequency-averaged sky-temperature (not power like in dPWV()) variations
    # This is going to be an RHS vector of our set of equations.
    dTxx=np.mean(iGxM*dPxx,axis=0)
    dTyy=np.mean(iGyM*dPyy,axis=0)

    # The sensitivity factor for WV variations. Column 1 in matrix.
    SW_XX=-np.mean((iGx*PxxM-Ti_XX-T_atm)*tau_1)
    SW_YY=-np.mean((iGy*PyyM-Ti_YY-T_atm)*tau_1)

    # The sensitivity factor for source-brigthtness vars. Col 2 in matrix.
    SB_XX=np.mean((freqs/f_LO)**spix*np.exp(-tau_t))
    SB_YY=SB_XX

    #--> endif(resolved spw)

else:

    #-- This is a channel-averaged AC or SQLD spw.

    # Calculate channel-averages over the SPW for atmospheric model params.
    tau_1M=tau_1.mean()
    tau_wM=tau_w.mean()
    tau_dM=tau_d.mean()
    tau_tM=tau_t.mean()
    T_atmM=T_atm.mean()

    print('Opacities w1:',tau_1M,'wet:',tau_wM, 'dry: ',tau_dM,
          'Total:',tau_tM, 'Transmisivity:',np.exp(-tau_tM))


    #-- Integrated sensitivity factors for averaged SPWs.
    #   The index '0' represents the single frequency channel we have.
```

```
  # Frequency-averaged sky-temperature variations.
  # This is going to be an RHS vector of our set of equations.
  dTxx=iGx[0]*dPxx[0]
  dTyy=iGy[0]*dPyy[0]

  # The SQLD spws have 12x higher cadence than the ACs. Rebin the SQLD
  # data in time to have a homogeneous series. Do the same for time axis MJD
  if mySPW in [0,1,2,3,13,14,15,16]:
    dTxx=rebin(dTxx,12)
    dTyy=rebin(dTyy,12)
    mjd_src=rebin(mjd_src,12)

  # Shortcut for 'net' sky temnperature, cleaned by atm. and inst. contribs.
  TnetXX=iGx[0]*PxxM[0]-T_iXX[0]-T_atmM
  TnetYY=iGy[0]*PyyM[0]-T_iYY[0]-T_atmM


  # The sensitivity factor for total WV variations. Column 1 in matrix.
  SW_XX=-TnetXX*tau_1M
  SW_YY=-TnetYY*tau_1M

  # The sensitivity factor for source-brigthtness vars. Col 2 in matrix.
  SB_XX=(freqs[0]/f_LO)**spix*np.exp(-tau_tM)
  SB_YY=SB_XX


  """
 Alternative/additional decompositions of a signal. Using either option
 1 or 2 below, we arrive to in total 3 unknown variables:
 [wet-slow, wet-fast, source-brightness] for Option 1, and
 [wet-total, dry, source-brightness] for Option 2.
 @todo: If it shall be implemented, do it for chan-resolved SPW above, too.

 Option 1: Use these two instead of total wet component.

 # Slow WV variations: Humidity / WV rel. abundance.
SWS_XX=-TnetXX*tau_wM
SWS_YY=-TnetYY*tau_wM

 # Fast WV variations: Air-mixture total density.
SWF_XX=-TnetXX*tau_tM
SWF_YY=-TnetYY*tau_tM

 Option 2: Add this expression for the dry component to the total wet vars.

 # Total dry variations.
SD_XX=-TnetXX*tau_dM
SD_YY=-TnetYY*tau_dM

 As it is shown in PR15, both options are mathematically equivalent, we
 just decompose to two different bases. However, none is working - either
 the system is ill-conditioned (Option 1) or we are just ampllifying noise
 in because very low sensitivity factor for dry component (Option 2).
  """
  #--> endelse(averaged spw)
```

```
    #-- Set of linear equations for dW and dB: Assembly the system.

    # Set the sensitivity factors for the current SPW as a row in the matrix.
    Axx=np.append(Axx,[SW_XX,SB_XX])
    Ayy=np.append(Ayy,[SW_YY,SB_YY])

    # Set the sky temperature variations in the current SPW as the RHS vectors.
    Cxx=np.append(Cxx,dTxx)
    Cyy=np.append(Cyy,dTyy)

    spwIndex+=1

#--> endfor(spws)

# The last spw index is the spw count.
numSpw=spwIndex

# Re-shape the matrix to be nVars x numSpw array
Axx=np.reshape(Axx,(numSpw,-1))
Ayy=np.reshape(Ayy,(numSpw,-1))


# Re-shape the RHS vectors to be nTimeSamples x numSpw array
Cxx=np.reshape(Cxx,(numSpw,-1))
Cyy=np.reshape(Cyy,(numSpw,-1))


#-- We most likely have over-determined system. Use the least-square
#   approach to get quadratic form and its minimizer

# Transposed matrices.
AxxT=np.transpose(Axx)
AyyT=np.transpose(Ayy)


# Quadratic-form matrix 2x2 (we have two variables).
AxxK=np.matmul(AxxT,Axx)
AyyK=np.matmul(AyyT,Ayy)

# Modified RHS (nTimeSamples x 2)
CxxK=np.matmul(AxxT,Cxx)
CyyK=np.matmul(AyyT,Cyy)


#-- Solve the nVarsx x nVars set of equations, one per polarization,
#   using Cramer's rule. So far just for 2 x 2 - need to be updated in case
#   of extended decomposition (see commented 'Alternative...' part above).

#- Pol XX

A11=AxxK[0,0]
A12=AxxK[0,1]

A21=AxxK[1,0]
A22=AxxK[1,1]

print('A11=',A11,', A12=',A12)
print('A21=',A21,', A12=',A22)
```

```
C1=CxxK[0]
C2=CxxK[1]

print('C1=',C1)
print('C2=',C2)

iD=1.0/(A11*A22-A12*A21) # Inverse determinant of the system.

print("Inverse determinant (XX): ",iD)

dWWxx=iD*(A22*C1-A12*C2)
dBBxx=iD*(A11*C2-A21*C1)

#- Pol YY

A11=AyyK[0,0]
A12=AyyK[0,1]

A21=AyyK[1,0]
A22=AyyK[1,1]

print('A11=',A11,', A12=',A12)
print('A21=',A21,', A12=',A22)

C1=CyyK[0]
C2=CyyK[1]

print('C1=',C1)
print('C2=',C2)

iD=1.0/(A11*A22-A12*A21) # Inverse determinant of the system.
print("Inverse determinant (YY): ",iD)

dWWyy=iD*(A22*C1-A12*C2)
dBByy=iD*(A11*C2-A21*C1)

#--- Diagnostic prints & plots

#- All variations should essentially have zero mean.

print("dWxx, dWyy mean: ", dWWxx.mean(),dWWyy.mean())
print("dBxx, dByy mean: ", dBBxx.mean(),dBByy.mean())


#-- Make a plot comparing the measured variations, their part attributed to
#   the intrinsic brightness changes, and error in eqs-system solution.

#-Set the plotting device.

plt.rcParams["figure.figsize"] = (30,70)
# Make XX and YY plots as subplots (horizontally)
ff,sps=plt.subplots(3,2)

# X-axis label just for the bottom sub-plot.
sps[2,0].set_xlabel('Time [MJDsec]')
sps[2,1].set_xlabel('Time [MJDsec]')
```

```
# Add the antenna name as a plot title.
plt.suptitle('Antenna '+str(antID))

# Color-map for SPWs in increasing order, incl. line-style.
# We have minimum 2 and maximum 12 SPWs in the spwList argument.
# If we use all 12 spws, linestyle correspondence to the spw type follows:
csline=['-r','-g','-b','-k',          # AC resolved
        '--r','--g','--b','--k',      # AC integrated
        ':r',':g',':b',':k']          # SQLD

#- Calculate the errors for overdetermined set of eqs.

# Compose the variables into a stack of vectors nTimeSamples x 2
varVecXX=np.array([dWWxx,dBBxx])
varVecYY=np.array([dWWyy,dBByy])


# Subtract 'theoretical' and real sky-temperature variations.
errVecXX=np.matmul(Axx,varVecXX)-Cxx
errVecYY=np.matmul(Ayy,varVecYY)-Cyy


#- Make the analytical plots - least-square residuals etc.

# *** 1st row
# Left panel: XX errs
spwCount=0
for spw in spwList:
  sps[0,0].plot(mjd_src,errVecXX[spwCount],csline[spwCount])
  spwCount+=1

# Right panel: YY errs
spwCount=0
for spw in spwList:
  sps[0,1].plot(mjd_src,errVecYY[spwCount],csline[spwCount])
  spwCount+=1


# *** 2nd row
# Left panel: XX RHS
spwCount=0
for spw in spwList:
  sps[1,0].plot(mjd_src,Cxx[spwCount],csline[spwCount])
  spwCount+=1

# Right panel: YY RHS
spwCount=0
for spw in spwList:
  sps[1,1].plot(mjd_src,Cyy[spwCount],csline[spwCount])
  spwCount+=1

# *** 3rd row
# Left panel: XX pol for T_src variations contrib to the total.
spwCount=0
for spw in spwList:
  sps[2,0].plot(mjd_src,Axx[spwCount,1]*dBBxx,csline[spwCount])
  spwCount+=1
```

```
  # Right panel: YY pol for T_src variations contrib to the total.
  spwCount=0
  for spw in spwList:
    sps[2,1].plot(mjd_src,Ayy[spwCount,1]*dBByy,csline[spwCount])
    spwCount+=1


  #- Save the plots to a file and close.
  plt.savefig('./dwdp_4SPWs/errs_ant_'+str(antID)+'.png')
  plt.show()
  plt.close()


  #-- Make the *solution* plots: dW(t) and dB(t).

  # Set the size of the plot
  plt.rcParams["figure.figsize"] = (10,5)
  # Make slow-WV and fast-WV plots as subplots (horizontally)
  ff,(sp1,sp2)=plt.subplots(1,2)
  # Add the antenna name as a plot title.
  plt.suptitle('Antenna '+str(antID))

  # Left panel: Fast variations - apparent target brightness
  sp1.plot(mjd_src,dBBxx,'-r')
  sp1.plot(mjd_src,dBByy,'-b')

  # Right panel: Slow variations - total PWV (compression + humidity).
  sp2.plot(mjd_src,dWWxx,'-r')
  sp2.plot(mjd_src,dWWyy,'-b')

  plt.savefig('./dwdp_4SPWs/dWdB/dWdB_ant_'+str(antID)+'.png')
  plt.show()
  plt.close()


  #-- Return the dW and dB variations preceded by time.
  #   Add the (original over-determined) matrix and the RHS vector, too.

  # Compose the XX and YY polarization component to a single arrays
  dWW=np.array([dWWxx,dWWyy])
  dBB=np.array([dBBxx,dBByy])
  AA=np.array([Axx,Ayy])
  CC=np.array([Cxx,Cyy])

  # Return all as a map/dictionary.
  return {'time':mjd_src,'dW':dWW,'dB':dBB,'A':AA,'C':CC}

#-----------------------------------------------------------------------------

def dWdB_Tbs(vis,atmScan,tgtScan,myAtm,antID,spwList,flaggedData=[],
             myMS=None,myMsmd=None,acdLoads=None):
  """

  *** Need to be edited if used for obsevartion different from 2023.12.16 ! ***
  Namely the ehpemeris-dependend data at the beginning of the script (indicated
  in comment).
```

An alternative approach to dWdB(), which, instead of using Tatm in the expression Tbs=Tatm(1-tr) for atmospheric contribution to the sky brightness, uses directly Tbs=at.TebbSky(), smiliraly to our approach for PWV calculations from the WVR temperatures, like in pwvCalc() or dWdLsun().*

It should be of the same order of accuracy and complexity, because the Tatm is calculated from the same ATM model anyway, just we have to scale it by the factor (1-tr).

The big advantage of this alternative approach is that we can easilly find also the scan-averaged PWV value, not just its differential variations.

We use the same approach as in dWdLsun(): Linearizing the radiative-transfer equation around its "operation point", which is defined by the scan-averaged PWV w0 and scan-averaged 'luminosity' of the Sun Tsun0. The linear set of equations is overdetermined for more than two SPWs (if they occupy distinct frequency domains), so we solve the set of 'normal' equations, ipmlying from the least-square principle, like in dWdB(). However, in our linearized set of equations, the variations need not to have zero mean. So we add the mean to the stationary 'operation point' defined by [w0,Tsun0] and make a new iteration, like in dWdLsun(), until we reach zero-mean variations. This way, both the variations and their stationary part is found. This approach has been verified via dWdB_Tbs_NL(), which solves the non-linear RT equation directly by Newton-Raphson iterations: Results of both differ insignificantly.

We do appropriate integration in the frequency domain for all model-based quantities. The solar QSR spectrum is being scaled by formula (4) in Alissandrakis et al. (2020), the original power-law estimation has been abandoned beacuse of large residua (see below in the code).

Moreover, we apply consequentlythe antenna efficiencies eta_l and eta_F (see White et al. 2017): For the fluctuations within in a single SPW they may be incorporated into effective gain and 'instrumental' temnperature, but here, when using all SPWs and absolute calibration to solar T_B and full PWV, it is not possible.

And the last innovation: The PWV values should be common for both XX and YY polarization. On the pther hand, the solar brightness T_B may differ in XX and YY channels (e.g., the apparent changes due to atm.-turbulence lensing may be polarization dependent, even the solar FoV may have a small amount of inrinsic polarization). Therefore, we have 3 independent variables: PWV, TB_XX, and TB_YY.

Arguments (these that differ from dWdB()):

**myAtm*: Here, myAtm is the *attool* set for our atmosphere, for all 4 resolved sci SPWs. It is a difference to dWdB(), where we use already pre-calculated ('tabelized') model, i.e., 'static' arrays for opacities, Tatm, transmissivity etc. for a specific value of time-averaged PWV for the scan. Here it is not possible as the mean PWV (the w0 parameter) is one of the unknowns (in dWdB() we use its ASDM_CALWVR value interpolated to the center of the scan). The *myAtm* attool set for the frequency domain defined by the AC channnel-resolved SPWs can be conveniently obtained via setAtmTool4SPWs().*

**flaggedData*: A list of strings in the form of (example) ['9:XX','3:YY'] with the spw:polarization pairs that should be ignored in the analysis. This can be used for multi-passage of this method, where in the first run we find the final-solution residua and the SPW:Pol pair, whose residuum is more than*

```
desired distance (e.g., 3 sigma) appart from the mean, can be flagged for
the repeated run.

*myMsmd*, *myMS*: Conveniently we can pass the tools already attached ton the
MS *vis*. If nothing is provided, the method allocates and attaches its own
local tools (whoch costs some CPU time)

*acdLoads*: A dictionary of ACD temperatures (got by aU.getLoadTemperatures())
from which, for example, the ambient-load temperature can be extracted by
Tamb = acdLoads[antID][atmScan]['amb']. If not provided, it is generated
inside the routine (cumbersome, if we call it in the loop for all antennas).
"""

#--- Static parameters

#- Dirs

# Output/diagnostics plots.
plotDir='./sciPWVs'


#- Tools, global data

# Flags used to check, whether we should close the tool later.
ownMS=False
ownMsmd=False

# Open own tools if 'global' not provided
if (not myMS):
  # Open the ms tool
  myMS=mstool()
  myMS.open(vis)
  ownMS=True

if (not myMsmd):
  # Open the ms tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True

# Create our dictionary with ACD load temperatures (ambient, hot) if needed.
if (not acdLoads):
  acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)

#- Source

# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6300.0   # K

# Solar radius.
Rsun=696000 # km

# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
```

```
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.

# Apparent angular solar radius in arcsec.
r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

# mu=cos(theta). Used for limb-brightenning correction below.
mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

# b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
# in Alissandrakis et al. (2020).
b_2=-1127.0 # = 2x (-563.5K) as per Table 2 in Allisandrakis et al. (2020).

# Brightness temoerature corrected to limb brightenning - see
# Alissandrakis et al. (2020).
Tsun00=TsunCenter+0.5*b_2*np.log(mu)

# Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
f_LO=1.98e11


#- SPWs

# Integration limits for resolved AC SPWs.
startChan=8
stopChan=120

#- Antenna

# Rear-spillover + ohmic antenna efficiency (see Tab. 3 in White et al. 2017).
eta_l=0.95
# Forward-spillover antenna efficiency (Tab. 3 in White et al. 2017).
eta_F=0.90

# Extract the ambient temperature for current antenna and scan.
# It determines the rear-spillover contribution to noise.
Tamb = acdLoads[antID][atmScan]['amb']


#- Atmosphere

 # Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
w0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',w0,' @airMass = ',airMass)
print('Estimated T_B = ',Tsun00)
```

```
print('.......')

#- SciSPW to AtmSPW mapping.

# How science SPWs map to the SPW 0 - 3 in the ATM tool.
sciAtmMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}


#--- Read-out the sky+src brightness temperatures for all SPWs in the spwList.

# Store the frequency-averaged sky+src temperatures in a map
# spwId -> np.array(nTimeSamples). Do not use 2D array as we index by the SPW
# id, which is sparse in the set of integers.
Txx={}
Tyy={}


for sciSPW in spwList:

  # Read ACD calibrated sky temperatures for the target defined by targetScan.
  time,TxxS,TyyS=tgt_Tb(vis,atmScan,tgtScan,antID,sciSPW,
                        myMS=myMS,myMsmd=myMsmd,myAtm=myAtm,acdLoads=acdLoads)

  # We have different handling for frequency-resolved and integrated SPWs.
  nChans=TxxS.shape[0]

  if (nChans > 1):

    # Resolved AC SPWs.
    # Trim the spectral edges by posing the integration limits and average
    # over the remaining channels.

    TxxM=np.mean(TxxS[startChan:stopChan],axis=0)
    TyyM=np.mean(TyyS[startChan:stopChan],axis=0)

  else:

    # Integrated ACs or SQLDs.
    # The single channel already represents a mean over the band-pass.

    TxxM=TxxS[0]
    TyyM=TyyS[0]

    # The SQLD spws have 12x higher cadence than the ACs. Rebin the SQLD
    # data in time to have a homogeneous series. Do the same for time axis.
    if sciSPW in [0,1,2,3,13,14,15,16]:
      TxxM=rebin(TxxM,12)
      TyyM=rebin(TyyM,12)
      time=rebin(time,12)

  # Append the frequency-averged time series for each SPW to the Tsky arrays.
  Txx[sciSPW]=TxxM
  Tyy[sciSPW]=TyyM


  print('SPW ',sciSPW,': <Tsky+src_XX> = ',Txx[sciSPW].mean(),'+/-',
        Txx[sciSPW].std())
  print('          <Tsky+src_YY> = ',Tyy[sciSPW].mean(),'+/-',
```

```
        Tyy[sciSPW].std())


#-->endfor(sciSpws in list)

# Count the SPW in the list.
numSpw=len(spwList)
# Count the time samples in data.
nSamples=len(time)

# The data are read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()
if (ownMS): myMS.close()


#-- Allocate arrays to be used for matrix A and the RHS-vector C calculations
#   from the ATM model. They have just nAtmSPWs components.
nAtmSPWs=myAtm.getNumSpectralWindows()
nChans=myAtm.getNumChan(0)

# Allocate the model sky (source + atm) temperature.
TskyXX=np.zeros(nAtmSPWs)
TskyYY=np.zeros(nAtmSPWs)

# Sensitivity factors for dW and dB (Jacobian elements).
SWxx=np.zeros(nAtmSPWs)
SWyy=np.zeros(nAtmSPWs)

SBxx=np.zeros(nAtmSPWs)
SByy=np.zeros(nAtmSPWs)


# Frequencies of channels in our ATM SPWs.
spw_freqs=np.zeros((nAtmSPWs,nChans))

for atmSpw in range(nAtmSPWs):

  fMin=myAtm.getMinFreq(atmSpw)['value'][0]*1.0e9 # GHz to Hz
  df=myAtm.getChanSep(atmSpw)['value'][0]*1.0e6   # MHz to Hz

  spw_freqs[atmSpw]=fMin+df*np.arange(nChans)


#--- Improve a guess of the solar brigthness temperature.

TsolEst=np.zeros(numSpw)

spwCount=0
for sciSPW in spwList:

  # Map the sci SPW to its corresponding ATM counterpart
  atmSPW=sciAtmMap[sciSPW]

  # Sky brightness in a given SPW for the current PWV estimate.
  TbsEst=myAtm.getAverageTebbSky(atmSPW,qa.quantity(w0,'mm'))['value'][0]

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
```

```
    # Water-vapour related 'wet' opacity
    tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
    # Total opacity (spectral).
    opacity=airMass*(tau_wet+tau_dry)

    # Mean (spw-averaged) transmissivity
    transM=np.mean(np.exp(-opacity))
    # Frequency-dependent part of sol-flux spectrum weighted by transmissivity.
    SFtrans=np.mean(np.exp(-opacity)*np.log(spw_freqs[atmSPW]/f_LO))

    # <Tsky+src> estimation from measurements, averaged over time and pols.
    TskyEst=0.5*(Txx[sciSPW].mean()+Tyy[sciSPW].mean())

    # Solar <Tsrc> temperature estimation in the current SPW.
    TsolEst[spwCount]=(TskyEst-(1.0-eta_F+eta_l)*TbsEst
                       -eta_l*eta_F*b_2*SFtrans
                       -(1.0-eta_l)*Tamb)/(eta_l*eta_F*transM)


    spwCount+=1

#--> endfor(sci SPWs)

print("Estimated Tsun from sci SPWS: ",TsolEst)

# Make a new initial guess of Tsun as an average.
Tsun00=TsolEst.mean()

print("Tsun = ",Tsun00,"+/-",TsolEst.std())

# Set it as the starting value for iterations.
Tsun0xx=Tsun00
Tsun0yy=Tsun00

#-- Average Solar-Flux spectrum (dimension-less, normalised to 1.0 at f=f_LO)

# It has the same dimensions as spw_freqs, i.e. nAtmSPWs x nChans
SF=1.0+b_2/Tsun00*np.log(spw_freqs/f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
dwMean_max=0.01
dbMean_max=0.01

# Initial values for the dW and dL means - anyting bigger then the stoppers.
dW_mean=0.5
dB_mean=1.0

# Iteration counter.
itCount=0
```

```
#-- Iterations: Minimize the mean value of variations.

while(((abs(dW_mean) > dwMean_max) or (abs(dB_mean) > dbMean_max))
      and (itCount< 20)):

  #- Prepare/reset the linearized system of equations.

  # (Jacobian linearization) Matrices (so far empty placeholders).
  JJ=np.array([])

  # RHS vectors (so far empty placeholders).
  CC=np.array([])

  # Rezidual mean sky temperature - after subtratced brightness of the
  # atmosphere (+CMB) and (attenuated) source. We keep it for all sci SPWs and
  # both polarizations and print it out at the end of every iteration cycle
  # as a diagnostics of convergence and solution quality.

  resTxx={}
  resTyy={}

  # Mapping between spw IDs and the row number in the system of equations.
  # We need it for final diagnostic plots. It may be non-trivial as some
  # spw:pol pairs may be flagged and ignored, casuing irregularity in the map.

  spw2rowXX={}
  spw2rowYY={}


  #- Re-calculate ATM model with updated <PWV> and <Tsun>.

  # Set current estimation of PWV to the ATM model.
  myAtm.setUserWH2O(qa.quantity(w0,'mm'))

  #- Loop over 4 ATM spws - these correspond to the channel-resolved ACs.
  #  Some factors depend only on the ATM mmodel, not the sciSPW data.
  for atmSPW in range(nAtmSPWs):

    #- Model value of the sky/atm brightness with current pwv.
    TbsW0=myAtm.getTebbSkySpec(atmSPW,qa.quantity(w0,'mm'))[1]['value']

    #- Slope (centered difference) of the atm brightness with current pwv.
    TbsW0H=myAtm.getTebbSkySpec(atmSPW,
                                qa.quantity(w0+dws,'mm'))[1]['value']
    TbsW0L=myAtm.getTebbSkySpec(atmSPW,
                                qa.quantity(w0-dws,'mm'))[1]['value']

    dTdW=(TbsW0H-TbsW0L)/(2.0*dws)


    #- Opacities & transmisivity.

    # Dry zenith opacity
    tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
    # Water-vapour related 'wet' opacity
    tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
    # Wet opacity per 1mm PWV scaled by airmass.
```

```
    tau_1=airMass*tau_wet/w0
    # Total opacity (spectral).
    opacity=airMass*(tau_wet+tau_dry)
    # Transmissivity (spectral).
    trans=np.exp(-opacity)


    #- Sensitivities = Jacobian elements. Weighted averages over the SPW.

    # Sensitivity factors for PWV.
    SWxx[atmSPW]=(-eta_l*eta_F*Tsun0xx*np.mean(trans*tau_1*SF[atmSPW])
                  +(1.0-eta_F+eta_l)*np.mean(dTdW))

    SWyy[atmSPW]=(-eta_l*eta_F*Tsun0yy*np.mean(trans*tau_1*SF[atmSPW])
                  +(1.0-eta_F+eta_l)*np.mean(dTdW))


    # Sensitivity factors for the (apparent) solar-brigthness temperature.
    SBxx[atmSPW]=eta_l*eta_F*np.mean(trans*SF[atmSPW])
    SByy[atmSPW]=eta_l*eta_F*np.mean(trans*SF[atmSPW])

    #- T_sky model. Weighted averages over the SPW.
    TskyXX[atmSPW]=(eta_l*eta_F*Tsun0xx*np.mean(trans*SF[atmSPW])
                    +(1.0-eta_F+eta_l)*np.mean(TbsW0)
                    +(1.0-eta_l)*Tamb)

    TskyYY[atmSPW]=(eta_l*eta_F*Tsun0yy*np.mean(trans*SF[atmSPW])
                    +(1.0-eta_F+eta_l)*np.mean(TbsW0)
                    +(1.0-eta_l)*Tamb)

#-->endfor(ATM spws)


#-- Loop over all sciSPWS in the list.

# Count the actual number of equations - some rows may be flagged.
eqCount=0

for sciSPW in spwList:

    # Map the sci SPW to its corresponding ATM counterpart
    atmSPW=sciAtmMap[sciSPW]

    #- Separate procedure by polarizations

    #- Pol XX

    # Full identificator of data: sciSPW:pol
    sciDataID=str(sciSPW)+':XX'

    if (sciDataID not in flaggedData):

      # Set the sensitivity factors for the current SPW and pol as a row
      # in the linearized-system Jacobian matrix.
      JJ=np.append(JJ,[SWxx[atmSPW],SBxx[atmSPW],0.0])

      # RHS: Txx/Tyy with their stationary (ATM model-based) part subtracted.
      dTxx=Txx[sciSPW]-TskyXX[atmSPW]
```

```
      CC=np.append(CC,dTxx)

      # Residua of sky tempereture for mean PWV and Tsun (operation point).
      resTxx[sciSPW]=dTxx.mean()

      # Mapping between SPW:XX and equation number.
      spw2rowXX[sciSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',sciDataID,' data.')
    #--> endif(not flaggged)


    #- Pol YY

    # Full identificator of data: sciSPW:pol
    sciDataID=str(sciSPW)+':YY'

    if (sciDataID not in flaggedData):

      # Set the sensitivity factors for the current SPW and pol as a row
      # in the linearized-system Jacobian matrix.
      JJ=np.append(JJ,[SWyy[atmSPW],0.0,SByy[atmSPW]])

      # RHS: Txx/Tyy with their stationary (ATM model-based) part subtracted.
      dTyy=Tyy[sciSPW]-TskyYY[atmSPW]
      CC=np.append(CC,dTyy)

      # Residua of sky tempereture for mean PWV and Tsun (operation point).
      resTyy[sciSPW]=dTyy.mean()

      # Mapping between SPW:YY and equation number.
      spw2rowYY[sciSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',sciDataID,' data.')
    #--> endif(not flaggged)

  #--> endfor(all sciSpws in the list)


  # Re-shape the matrix to be nVars x numEqations array
  JJ=np.reshape(JJ,(eqCount,3))

  # Re-shape the RHS vectors to be nTimeSamples x numEquations array
  CC=np.reshape(CC,(eqCount,-1))


  #- We most likely have over-determined system. Use the least-square
  #  approach to get quadratic form and its minimizer ('normal' equations).

  # Transposed matrices.
  JJ_T=np.transpose(JJ)


  # Quadratic-form matrix 3x3 (we have three variables).
```

```
    J_KK=np.matmul(JJ_T,JJ)

    # Modified RHS (nTimeSamples x nVars)
    C_KK=np.matmul(JJ_T,CC)

    # Solve the nVarsx x nVars set of so called normal equations.
    varVec=np.linalg.solve(J_KK,C_KK)

    # Decompose the solution vectror to our components dW, dBxx, and dByy.

    dW = varVec[0]
    dBxx=varVec[1]
    dByy=varVec[2]

    #- All variations should tend to zero means.

    dW_mean=dW.mean()
    dBxx_mean=dBxx.mean()
    dByy_mean=dByy.mean()

    dB_mean=np.sqrt(dBxx_mean**2+dByy_mean**2)

    # Update stationary part of PWV and solar T_B  by means of their variations.
    w0+=dW_mean
    Tsun0xx+=dBxx_mean
    Tsun0yy+=dByy_mean

    #- Convergence diagnostics.

    itCount+=1

    # Updated scan-averaged PWV and Tsun after completed iteration.
    print('....... After iteration no. ',itCount)
    print('Mean PWV for the scan,',tgtScan,' w0 = ',w0,
          ", mean Tsun (XX/YY) = ",Tsun0xx,'/',Tsun0yy)

    # Mean values of dW and dB fluctuations should tend to zero.
    print("dW mean: ",dW_mean)
    print("dB mean: ",dB_mean)

    # Sky-brightness residua should tend to zero, too.
    # Convert the spwId maps to np.arrays first. We have to go via 'list'.
    resTxxArr=np.array(list(resTxx.values()))
    resTyyArr=np.array(list(resTyy.values()))

    print("res Txx: ", resTxx,"mean +/- RMS: ",
          resTxxArr.mean(),"+/-",resTxxArr.std())
    print("res Tyy: ", resTyy,"mean +/- RMS: ",
          resTyyArr.mean(),"+/-",resTyyArr.std())

    print('.......')

#--> endwhile(non-zero-means)

#-- Diagnostic plots.
#   Make a plot comparing the measured fluctuations, their part attributed to
#   the intrinsic brightness changes, and error in eqs-system solution.
```

```
#-Set the plotting device.

plt.rcParams["figure.figsize"] = (30,50)
# Make XX and YY plots as subplots (horizontally)
ff,sps=plt.subplots(3,2)

# X-axis label just for the bottom sub-plot.
sps[2,0].set_xlabel('Time [MJDsec]')
sps[2,1].set_xlabel('Time [MJDsec]')

# Add the antenna name as a plot title.
plt.suptitle('Antenna '+str(antID))

# Color-map for SPWs, incl. line-style.
# We have minimum 2 and maximum 12 SPWs in the spwList argument.
# If we use all 12 spws, linestyle correspondence to the spw type follows:
csline={5:'-r',7:'-g',9:'-b',11:'-k',          # AC resolved
        6:'--r',8:'--g',10:'--b',12:'--k',     # AC integrated
        0:':r',1:':g',2:':b',3:':k'}           # SQLD

#- Calculate the errors for overdetermined set of eqs.

# Subtract 'theoretical' and real sky+src temperature variations.
errVec=np.matmul(JJ,varVec)-CC


#- Make the *analytical* plots - least-square residuals etc.


# XX pol

for spw,eqRow in spw2rowXX.items():

  # Panel row 1: Residuals/errors.
  sps[0,0].plot(time,errVec[eqRow],csline[spw])
  # Panel row 2: RHS vectors = measured Tsky+src minus model Tsky.
  sps[1,0].plot(time,CC[eqRow],csline[spw])
  # Panel row 3: T_b variation contribution to total Tsky+src
  sps[2,0].plot(time,JJ[eqRow,1]*dBxx,csline[spw])

# YY pol

for spw,eqRow in spw2rowYY.items():

  # Panel row 1: Residuals/errors.
  sps[0,1].plot(time,errVec[eqRow],csline[spw])
  # Panel row 2: RHS vectors = measured Tsky+src minus model Tsky.
  sps[1,1].plot(time,CC[eqRow],csline[spw])
  # Panel row 3: T_b variation contribution to total Tsky+src
  sps[2,1].plot(time,JJ[eqRow,2]*dByy,csline[spw])


#- Save the plots to a file and close.
plt.savefig(plotDir+'/errs_ant_'+str(antID)+'.png')
plt.show()
plt.close()
```

```
  #-- Make the *solution* plots: dW(t) and dB(t).

  # Set the size of the plot
  plt.rcParams["figure.figsize"] = (10,5)
  # Make slow-WV and fast-WV plots as subplots (horizontally)
  ff,(sp1,sp2)=plt.subplots(1,2)
  # Add the antenna name as a plot title.
  plt.suptitle('Antenna '+str(antID))

  # Left panel: Fast variations - apparent target brightness
  sp1.plot(time,dBxx,'-r')
  sp1.plot(time,dByy,'-b')

  # Right panel: Slow variations - total PWV (compression + humidity).
  sp2.plot(time,dW,'-g')

  plt.savefig(plotDir+'/dWdB/dWdB_ant_'+str(antID)+'.png')
  plt.show()
  plt.close()

  #-- Prepare resulting data for export.

  # Add variations to the mean (scan-averaged) values.
  TsunXX=Tsun0xx+dBxx
  TsunYY=Tsun0yy+dByy

  pwv=w0+dW

  # 'Compactify' the per-polarization arrays.
  Tsun=np.array([TsunXX,TsunYY])

  return {'time':time,'pwv':pwv,'T_B':Tsun,'J':JJ,'C':CC}

#--------------------------------------------------------------------------

def dWdBdG(vis,atmScan,tgtScan,myAtm,antID,spwList,flaggedData=[],
           myMS=None,myMsmd=None,acdLoads=None):
  """
  Extended dWdB_Tbs(): Solve for antenna gains and instrumental powers, too.

  The most complex treating the RT problem for the Sun: Like with dWdB_Tbs() we
  get the time-series of full PWV and the (apparent, i.e. including the
  atmospheric lensing) solar brightness temperature - actually its reference
  value for f_LO=198.0GHz - in XX and YY polarizations. But, here, we moreover
  set the antenna gains *G* and the instrumental power *P_inst=G*Trx+Po* as
  unknown parameters to be solved by the extended set of equations.

  Rationale: dWdB_Tbs() seems to perfectly reproduce the variations in the solar
  brightness temeprature and PWV: They are very similar in all SPWs in both
  XX and YY polarizations. Yet, there is a stationary part of the T_B, which
  is different among SPWs and pols - see the residuum plots. Actually, this
  static-part residua have been significantly reduced, already when we have
  introduced the corrected ACD calibration method acd_calTabs2() for getting
  measured Tsky+src from directly recorded powers: The improved ACD calibration
  fixes the AC-spw P_off-based (i.e., with solar attenuation level) gain G*
  by transferring the gain-drop level (given by P_off/P_sky ratio) from the
  adjunged (the same frequency domain) SQLD spw. Hence, it looks these
  differences are due to imperfect ACD gain estimation: 1% error in the gain
```

```
easilly produces ~70K difference in the solar T_B. Therefore, this method has
been devised to try to improve the gain calibration.

How it works: We assume that the antenna gains and P_inst are constant in time
over a scan. Moreover, for the spectral AC SPWS, we assume that the normalised
gain g(f)=G(f)/<G(f)>, as used in acd_calTabs2(), is correctly calculated
by the ACD calibration, and here we calculate just its scaling. Hence,
for NT time-samples and NDC = nSPW x nPols = 24 data-channels, we have 2xNDC
unknown instrumental variables (G + P_inst per SPW and pol) and, 3xNT
'natural' variables (PWV, T_B_XX, T_B_YY). This is still significantly less
then number of equations, which is NT x NDC. Nevertheless, the matrix of such
system has very high condition number. It is connected with the fact that
we have no anchor for the instrumental parameters (namely gain scale).
Situation is improved a lot if we add a few further equations - the equations
for the ADC calibration - into the system. This way we solve the ATM-RT
problem and (correction to) ACD calibration simultaneously. For that reason,
the 'grand' system is really large - we have (NT + 2) x NDC equations (after
linearization of the NL RT equations), where '+2' is for the ambient- and
hot-load ACD calibration equations, so it is overdetermined. We use the
'normal' equation approach, i.e., solving minimizer for the quadratic form
of the least-squares. Still the normal square matrix is of the order
3 x NT + 2 x NDC and its direct inversion takes ages. However, it has been
found that the normal matrix for the extended system (natural variables +
instrumental parameters) can be conveniently inverted with knowledge of the
(easilly calculated) inversion of the original matrix for the natural
variabless (as used in dWdB_Tbs()) - the procedure is demonstrated in
extMatInv() auxilliary routine at the beginning of this script. Lastly,
we express the temperatures in kiloKelvins in order to have comparable
sensitivities for dW adn dT_B (which play rolse of matrix elements) and
improve thus the condition number of the matrix.
"""

#--- Static parameters

#- Dirs

# Output/diagnostics plots.
plotDir='./sciPWVs_GP'
if (not os.path.exists(plotDir)): os.makedirs(plotDir)
if (not os.path.exists(plotDir+'/dWdB')): os.makedirs(plotDir+'/dWdB')


#- Tools, global data

# Flags used to check, whether we should close the tool later.
ownMS=False
ownMsmd=False

# Open own tools if 'global' not provided
if (not myMS):
  # Open the ms tool
  myMS=mstool()
  myMS.open(vis)
  ownMS=True

if (not myMsmd):
  # Open the ms tool.
  myMsmd=msmdtool()
```

```
    myMsmd.open(vis)
    ownMsmd=True

  # Create our dictionary with ACD load temperatures (ambient, hot) if needed.
  if (not acdLoads):
    acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                    warnIfNoLoadTemperatures = True,
                                    mymsmd=myMsmd)

  #- Target

  # Initial guess of the solar brightness temperature.
  # SD calibration script: Reference T_b for solar disc center @198GHz
  TsunCenter=6.3  # kiloKelvins!!!

  # Solar radius.
  Rsun=696000 # km

  # *** This part is observation specific - need to read ephemeris !!!

  # Heliocentric coordinates for July 30 2026Band 5 observation.
  '''
  Target:       RA=08 37 18.01 DEC=+18 31 23.0
  Sun center:   RA=08 38 15.79 DEC=+18 28 47.2
  ----------------------------------------------
  diff (arcsec, incl. cos(delta)): 822   155.8
  '''
  x_h=822.0 # arcsec
  y_h=155.8  # arcsec

  # Earth-Sun distance for July 30 2026 Band 5 observation.
  Dsun=1.01508611721886*149597870.700 # km

  # *** End of observation-specific part.

  # Apparent angular solar radius in arcsec.
  r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

  # mu=cos(theta). Used for limb-brightenning correction below.
  mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

  # b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
  # in Alissandrakis et al. (2020). Scaled to kiloKelvins!
  b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

  # Brightness temoerature corrected to limb brightenning - see
  # Alissandrakis et al. (2020).
  Tsun00=TsunCenter+0.5*b_2*np.log(mu)

  # Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
  f_LO=1.98e11

  #- SPWs

  # Integration limits for resolved AC SPWs.
  startChan=8
  stopChan=120
```

```
fChans=range(startChan,stopChan)

#- Antenna

# Rear-spillover + ohmic antenna efficiency (see Tab. 3 in White et al. 2017).
eta_l=0.95
# Forward-spillover antenna efficiency (Tab. 3 in White et al. 2017).
eta_F=0.90

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']


# Weight of the ACD calibration equations in the system. Essentially the
# number, with which appended block of the ACD equations is repeated in order
# to balance the much larger sciData-equations block. Infinity means that
# the instrumental parameters G and P_inst are essentially fixed and given
# by results of the acd_calTabs(). Low weight, on the other hand, leads to
# an ill-coinditioned system.
# The selected value of 200 has been found experimentally: With lower value
# we have a large error in the instrumental part of the solution.
# On the other hand, increasing beyond this value increseases the error in
# the natural part with no significant improvement in the instrumental error.
# See dWdBdG_calWeights(), which has been designed for exerimentation with
# the calWeight parameter.
calWeight=200.0

#- Atmosphere

# Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,' @airMass = ',airMass)
print('Estimated T_B = ',1000.0*Tsun00)
print('.......')

#- SciSPW to AtmSPW mapping.

# How science SPWs map to the SPW 0 - 3 in the ATM tool.
sciAtmMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}

#--- Read-out the sky+src powers for all SPWs in the spwList.

# Store the frequency-averaged sky+src powers in a map.
# spwId -> np.array(nTimeSamples). Do not use 2D array as we index by the SPW
# id, which is sparse in the set of integers.
Pxx={}
```

```
Pyy={}

#- Similar maps for the gains and instrumental powers P_inst=G*T_inst.
#  Here, all gains and P_inst are, in fact, the G* and P_ints* values for the
#  higher-attenuation solar regime. In order to append the ACD calibration
#  equations, whose measurements are performed in the norma-attenuation regime
#  in the preceding ATM scan, we determine also the attenuation factors
#  AF=G*/G and related power offsets AO.

# Frequency-averaged gains. The scale for the spectral-resolved norm. gains.
Gxx={}
Gyy={}

# Attenuation factors G/G*=(P_sky-P0)/(P_off-Po). Attributed to every-spw
# Gxx & Gyy, but actually calculated just in SQLD spws, the rest is mapped.
AFxx={}
AFyy={}

afMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}

# Attenuation-based power offsets AO=-Po*(P_sky-P_off)/(P_off-Po).
# The same holds as for the AFs - we use afMap.
AOxx={}
AOyy={}

# Spectral gain normalized to 1.0. Acual ACD-based values in spectral AC spws,
# a trivial 1.0 ndarray for integrated-AC and SQLD spws.
gXX={}
gYY={}

# Frequency-averaged instrumental powers.
PIxx={}
PIyy={}

#- Finally, the raw ACD-calibration powers.

# Ambient-load powers.
PxxAmb={}
PyyAmb={}

# Hot-load powers.
PxxHot={}
PyyHot={}

#-- Read the data and ACD calibration tables in a loop over all sciSPWs.

for sciSPW in spwList:

  #- Raw data & ACD tables

  # Read raw science powers for the target defined by targetScan.
  time,P_src=tgt_pow(vis,tgtScan,antID,sciSPW,myMS=myMS,myMsmd=myMsmd,
                     subScans='on')

  PxxS=P_src[0]
  PyyS=P_src[1]

  # Read ACD calibrations - gains,Tinst, and calibration powers.
```

```
calDict=acd_calTabs(vis,atmScan,antID,sciSPW,myMS=myMS,myMsmd=myMsmd,
                    acdLoads=acdLoads)

Pcals=calDict['calPows']

PxxAmbS=Pcals['P_amb'][0]
PyyAmbS=Pcals['P_amb'][1]

PxxHotS=Pcals['P_hot'][0]
PyyHotS=Pcals['P_hot'][1]


# We have different handling for frequency-resolved and integrated SPWs.
nChans=PxxS.shape[0]

if (nChans > 1):

  #- Resolved AC SPWs.
  #  Trim the spectral edges by posing the integration limits and average
  #  over the remaining fChans channels.

  # Science-target powers.
  PxxM=np.mean(PxxS[fChans],axis=0)
  PyyM=np.mean(PyyS[fChans],axis=0)


  # Calibration (ACD) powers.
  PxxAmbM=np.mean(PxxAmbS[fChans],axis=0)
  PyyAmbM=np.mean(PyyAmbS[fChans],axis=0)

  PxxHotM=np.mean(PxxHotS[fChans],axis=0)
  PyyHotM=np.mean(PyyHotS[fChans],axis=0)

  #- Instrumental factors.
  #  Store the normalized (to 1.0) spectral gains (arrays, constant) and
  #  their scales (numbers, to be re-fined by the solver) separately.

  # Raw G* gains in a full SPW. We may flip the array (see below) so better
  # to do so before its limiting to fChans - wee need not to think whether
  # the fChans central range is placed symmetrically withn its SPW.
  # The 1st index 0 means we pick the leading P_off subscan - see the note
  # in acd_calTabs2().
  GxxS=1.0/(calDict['iG*'][0,0])
  GyyS=1.0/(calDict['iG*'][0,1])

  # T_inst in full SPW. The same holds as for G*.
  TinstXX=calDict['T_inst'][0]
  TinstYY=calDict['T_inst'][1]

  # Check, whether we need flip them - this is the case of the LSB spws,
  # which have decreasing frequency channels.
  spwFreqs=calDict['freqs']

  if (spwFreqs[0] > spwFreqs[1]):
    # Reversed SPW, flip the gain and T_inst arrays.

    GxxS=np.flip(GxxS)
    GyyS=np.flip(GyyS)
```

```
      TinstXX=np.flip(TinstXX)
      TinstYY=np.flip(TinstYY)

    # Now it is safe to select the central fChans band-pass.
    GxxS=GxxS[fChans]
    GyyS=GyyS[fChans]

    TinstXX=TinstXX[fChans]
    TinstYY=TinstYY[fChans]

    # Calculate the unity-normalized gains by dividing out the mean value.
    GxxM=GxxS.mean()
    GyyM=GyyS.mean()

    gXX[sciSPW]=GxxS/GxxM
    gYY[sciSPW]=GyyS/GyyM

    # Instrumental power averaged over our fChans range.
    PIxxM=np.mean(GxxS*TinstXX)
    PIyyM=np.mean(GyyS*TinstYY)

    # Keep the time-axis of the last resolved SPW as a reference timebase,
    # to which we intrepolate the SQLD signals.
    timeBase=time


else:
    #- Integrated ACs or SQLDs.
    #  The single channel already represents a mean over the spw band-pass.

    # Single-channel science data.
    PxxM=PxxS[0]
    PyyM=PyyS[0]


    # Channel-integrated SPWs may have higher cadence. For SQLDs it is true
    # always, but it may be the case of integrated ACs, too. Let us
    # put all data to the same, slowest, timebase (as per resoved AC spws).

    # Compare the cadence (elementary time-step, a.k.a. integration).
    if ((timeBase[1]-timeBase[0]) > (time[1]-time[0])):

      # Indeed, we have higher cadence. Re-bin in time.
      # Use the timeBase defined as a time-axis for resolved SPW.

      PxxM=rebin2(PxxM,time,timeBase)
      PyyM=rebin2(PyyM,time,timeBase)
      time=timeBase

    #--> endif(higher cadence)

    # The attenuation factors AF=(P_sky-Po)/(P_off-Po) and offsets AO should
    # be, moreover, extracted from the SQLDss (ACs use mapping afMap
    # defined above).
    if sciSPW in [0,1,2,3,13,14,15,16]:

      # Extract the caliibration powers relevant for AF calculation
```

```
      Patm=Pcals['P_sky']
      Poff=Pcals['P_off']
      Pzero=Pcals['P_zero']

      # Calculate AFs.
      # The 1st index in powers is pol, the 2nd means (the single) channel.
      AFxx[sciSPW]=(Patm[0,0]-Pzero[0,0])/(Poff[0,0]-Pzero[0,0])
      AFyy[sciSPW]=(Patm[1,0]-Pzero[1,0])/(Poff[1,0]-Pzero[1,0])

      # Calculates AOs.
      AOxx[sciSPW]=Pzero[0,0]*(AFxx[sciSPW]-1.0)
      AOyy[sciSPW]=Pzero[1,0]*(AFyy[sciSPW]-1.0)
    #--> endif(SQLD spws)

    # Calibration data.
    PxxAmbM=PxxAmbS[0]
    PyyAmbM=PyyAmbS[0]

    PxxHotM=PxxHotS[0]
    PyyHotM=PyyHotS[0]


    #- Instrumental factors.
    GxxM=1.0/(calDict['iG*'][0,0,0])
    GyyM=1.0/(calDict['iG*'][0,1,0])


    # Create a fake spectral normalized gain for integrated SPWs.
    # We use it in products with the ATM arrays (e.g. transmissivity), which
    # are always spectral.
    gXX[sciSPW]=np.ones(len(fChans))
    gYY[sciSPW]=np.ones(len(fChans))

    # Instrumental power: The channel 0 is the average already
    PIxxM=GxxM*calDict['T_inst'][0,0]
    PIyyM=GyyM*calDict['T_inst'][1,0]
  #--> endif(resolved SPW)


  # Append the frequency-averged time series for each SPW to the maps.
  Pxx[sciSPW]=PxxM
  Pyy[sciSPW]=PyyM

  # The calibration powers are already time-averaged.
  PxxAmb[sciSPW]=PxxAmbM
  PyyAmb[sciSPW]=PyyAmbM

  PxxHot[sciSPW]=PxxHotM
  PyyHot[sciSPW]=PyyHotM

  # Set the ACD-based gains (scales) and instrumental contributions.
  Gxx[sciSPW]=GxxM
  Gyy[sciSPW]=GyyM

  PIxx[sciSPW]=PIxxM
  PIyy[sciSPW]=PIyyM
```

```
  print('SPW ',sciSPW,': <Tsky+src_XX> = ',
        (Pxx[sciSPW].mean()-PIxx[sciSPW])/Gxx[sciSPW],'+/-',
        Pxx[sciSPW].std()/Gxx[sciSPW])
  print('          <Tsky+src_YY> = ',
        (Pyy[sciSPW].mean()-PIyy[sciSPW])/Gyy[sciSPW],'+/-',
        Pyy[sciSPW].std()/Gyy[sciSPW])

#-->endfor(all sciSPWs in the list)

# Save the original gains and instrumental powers for comparison with
# the updated values after the solution.
# Note: We have to use copy.deepcopy(), the simple assignment operator
# copies just the reference!

GxxOrig=deepcopy(Gxx)
GyyOrig=deepcopy(Gyy)

PIxxOrig=deepcopy(PIxx)
PIyyOrig=deepcopy(PIyy)

#-- Dimensions of the system.

# Count the SPW in the list.
numSpw=len(spwList)
# Count the time samples in data.
nSamples=len(time)

# How many net equations do we have per block of 1 time-sample.
# Be carefull: Do not flag the data that are *not* present in the spwList!
eqBlockSize=2*numSpw-len(flaggedData)


# The data has been read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()
if (ownMS): myMS.close()

#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).

# Number of the model spectral windows.
nAtmSPWs=myAtm.getNumSpectralWindows()
# Number of all channels in the model SPWs.
nChans=myAtm.getNumChan(0)
# Number of central band-pass (with edges trimmed) channels.
cChans=len(fChans)

# Frequencies of channels in our model SPWs (central band-pass only).
spw_freqs=np.zeros((nAtmSPWs,cChans))

for atmSpw in range(nAtmSPWs):

  fMin=myAtm.getMinFreq(atmSpw)['value'][0]*1.0e9 # GHz to Hz
  df=myAtm.getChanSep(atmSpw)['value'][0]*1.0e6   # MHz to Hz

  spw_freqs[atmSpw]=(fMin+df*np.arange(nChans))[fChans]

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
```

```
TbsW0=np.zeros((nAtmSPWs,cChans))
dTdW=np.zeros((nAtmSPWs,cChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,cChans))
tau_1=np.zeros((nAtmSPWs,cChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/Tsun00*np.log(spw_freqs/f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin

# Initial values for the dW and dL means - anyting bigger then the stoppers.
dW_mean=0.5
dB_mean=0.5

# Iteration counter.
itCount=0

# Set the starting values for iterations.
w0=pwv0
Tsun0xx=Tsun00
Tsun0yy=Tsun00

#-- Iterations: Minimize the mean value of variations.

while(((abs(dW_mean) > dwMean_max) or (abs(dB_mean) > dbMean_max))
      and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample

  # N-block matrix: Jacobian part for [dW,dT_Bxx,dT_Byy] 'natural' variables.
  J_wb=np.zeros((eqBlockSize,3)) # 3 vars repeating in all spws
  # G-block matrix: Jacobian part for [dG,dP_inst] 'instrumental' variables.
  J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each SPW:Pol.
  # A-block matrix: Jacobian part for [dG,dP_inst] for ACD calibration powers.
  J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, ambient.
  J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, hot.

  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))
  # RHS-vector 'appendix' - amb. & hot ACD calibration powers (time-averaged).
  CCA=np.zeros((eqBlockSize,1))
  CCH=np.zeros((eqBlockSize,1))
```

```
# Rezidual mean sky temperature - after subtratced brightness of the
# atmosphere (+CMB) and (attenuated) source. We keep it for all sci SPWs and
# both polarizations and print it out at the end of every iteration cycle
# as a diagnostics of convergence and solution quality.

resTxx={}
resTyy={}

# Mapping between spw IDs and the row number in the system of equations.
# We need it for final diagnostic plots. It may be non-trivial as some
# spw:pol pairs may be flagged and ignored, casuing irregularity in the map.

spw2rowXX={}
spw2rowYY={}

#- Re-calculate ATM model with updated <PWV> and <Tsun>.

# Set current estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

#- Loop over 4 ATM spws - these correspond to the channel-resolved ACs.
#  Calculate factors dependent only on the mmodel, not the data.
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Limit the spectrum to the central fChans reagion as we have trimmed
  # the edge channels in ACD-based spectral normalized gain gXX/gYY.
  # Scale to kiloKelvins.
  TbsW0[atmSPW]=0.001*Tebb[fChans]

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value'][fChans]
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value'][fChans]

  # In kiloKelvins/mm
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Transmissivity (spectral). Limitted to fChans.
  trans[atmSPW]=np.exp(-opacity[fChans])
  # Wet opacity per 1mm PWV scaled by airmass. Limitted to fChans.
  tau_1[atmSPW]=(airMass/w0)*tau_wet[fChans]

#--> endfor(modelSPWs)

#- Loop over all SCI spws - combine model-only dependent factors with
#  sciSPW-dependent data and instrumental factors and build the system.
```

```
    # Count the actual number of equations - some rows may be flagged.
    eqCount=0

    for sciSPW in spwList:

      # Map the sci SPW to its corresponding ATM counterpart
      atmSPW=sciAtmMap[sciSPW]

      #- Model (_M) factors.

      # Tebb and its slope.
      TbsW0_M=TbsW0[atmSPW]
      dTdW_M=dTdW[atmSPW]

      # Transmissivity and unit-PWV opacity.
      trans_M=trans[atmSPW]
      tau_1M=tau_1[atmSPW]

      # Solar-flux model spectrum (normalized).
      SF_M=SF[atmSPW]

      #- Separate procedure by polarizations for combination with real data.

      #- Pol XX

      # Full identificator of data: sciSPW:pol
      sciDataID=str(sciSPW)+':XX'

      if (sciDataID not in flaggedData):

        #- Model + instrumental factors combined.

        # Sensitivity factor for PWV.
        SWxx=(-eta_l*eta_F*Tsun0xx*np.mean(gXX[sciSPW]*trans_M*tau_1M*SF_M)
              +(1.0-eta_F+eta_F*eta_l)*np.mean(gXX[sciSPW]*dTdW_M))*Gxx[sciSPW]

        # Sensitivity factor for the (apparent) solar-brigthness temperature.
        SBxx=Gxx[sciSPW]*eta_l*eta_F*np.mean(gXX[sciSPW]*trans_M*SF_M)

        # Model T_sky as viewed by the receiver with spectral gain gXX.
        TskyXX=(eta_l*eta_F*Tsun0xx*np.mean(trans_M*SF_M*gXX[sciSPW])
                +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M*gXX[sciSPW])
                +eta_F*(1.0-eta_l)*Tamb)

        #- Build the matrix: Natural (dW & dB sensitivities) block.

        # Set the sensitivity factors for the current SPW and pol as a row
        # in the block/tile for the linearized-system Jacobian matrix.
        J_wb[eqCount]=np.array([SWxx,SBxx,0.0])

        #- Build the matrix: Instrumental (G & P_inst sensitivities) block.

        # We have to prepend a proper offset for the G and P_inst vars as
        # they are independendt for each SPW and polarization.
        bOffset=2*eqCount

        J_gp[eqCount,bOffset]=1.0
```

```
    J_gp[eqCount,bOffset+1]=TskyXX

    #- Build the matrix: ACD powers (amb & hot) 2-block.

    # Ambient-power sub-block. Scale between the G* / P_inst* used in
    # J_wb and J_gp, and G / P_inst used in the ACD cal equations by the
    # attenuation factor AF=G*/G=(P_sky-Po)/(P_off-Po)
    af=AFxx[afMap[sciSPW]]

    J_cdA[eqCount,bOffset]=af
    J_cdA[eqCount,bOffset+1]=af*Tamb

    # Hot-power sub-block.
    J_cdH[eqCount,bOffset]=af
    J_cdH[eqCount,bOffset+1]=af*Thot

    #- Build the RHS vector: Target data

    # RHS: Pxx(t) with their stationary (in the working point) instrumental
    # and ATM-based parts subtracted.
    dPxx=Pxx[sciSPW]-PIxx[sciSPW]-Gxx[sciSPW]*TskyXX

    # Set it as the RHS component.
    CC[eqCount]=dPxx

    #- Ammend the RHS vector: Calibration data.

    # Attenuation-based power offset.
    aoff=AOxx[afMap[sciSPW]]

    # Ambient-load power with its working-point part subtracted.
    CCA[eqCount]=PxxAmb[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Tamb+aoff
    # Hot-load power with its working-point part subtracted.
    CCH[eqCount]=PxxHot[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Thot+aoff

    #- Diagnostics

    # Residua of sky tempereture for mean PWV and Tsun (operation point).
    resTxx[sciSPW]=dPxx.mean()/Gxx[sciSPW]

    # Mapping between SPW:XX and equation number.
    spw2rowXX[sciSPW]=eqCount

    eqCount+=1

else: print('Ignoring flagged ',sciDataID,' data.')
#--> endif(not flaggged)

#- Pol YY

# Full identificator of data: sciSPW:pol
sciDataID=str(sciSPW)+':YY'

if (sciDataID not in flaggedData):

    #- Model + instrumental factors combined.

    # Sensitivity factor for PWV.
```

```
SWyy=(-eta_l*eta_F*Tsun0yy*np.mean(gYY[sciSPW]*trans_M*tau_1M*SF_M)
      +(1.0-eta_F+eta_F*eta_l)*np.mean(gYY[sciSPW]*dTdW_M))*Gyy[sciSPW]

# Sensitivity factor for the (apparent) solar-brigthness temperature.
SByy=Gyy[sciSPW]*eta_l*eta_F*np.mean(gYY[sciSPW]*trans_M*SF_M)

# Model T_sky as viewed by the receiver with spectral gain gYY.
TskyYY=(eta_l*eta_F*Tsun0yy*np.mean(trans_M*SF_M*gYY[sciSPW])
        +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M*gYY[sciSPW])
        +eta_F*(1.0-eta_l)*Tamb)

#- Build the matrix: Natural (dW & dB sensitivities) block.

# Set the sensitivity factors for the current SPW and pol as a row
# in the block/tile for the linearized-system Jacobian matrix.
J_wb[eqCount]=np.array([SWyy,0.0,SByy])

#- Build the matrix: Instrumental (G & P_inst sensitivities) block.

# We have to prepend a proper offset for the G and P_inst vars as
# they are independendt for each SPW and polarization.
bOffset=2*eqCount

J_gp[eqCount,bOffset]=1.0
J_gp[eqCount,bOffset+1]=TskyYY

#- Build the matrix: ACD powers 2-block.

# Attenuation factor G*/G.
af=AFyy[afMap[sciSPW]]

# Ambient-power sub-block.
J_cdA[eqCount,bOffset]=af
J_cdA[eqCount,bOffset+1]=af*Tamb

# Hot-power sub-block.
J_cdH[eqCount,bOffset]=af
J_cdH[eqCount,bOffset+1]=af*Thot

#- Build the RHS vector: Target data.

# RHS: Pyy(t) with their stationary (in the working point) instrumental
# and ATM-based parts subtracted.
dPyy=Pyy[sciSPW]-PIyy[sciSPW]-Gyy[sciSPW]*TskyYY

# Set it as the RHS component.
CC[eqCount]=dPyy

#- Ammend the RHS vector: Calibration data.

# Attenuation-based power offset.
aoff=AOyy[afMap[sciSPW]]

# Ambient-load power with its working-point part subtracted.
CCA[eqCount]=PyyAmb[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Tamb+aoff
# Hot-load power with its working-point part subtracted.
CCH[eqCount]=PyyHot[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Thot+aoff
```

```
      #- Diagnostics

      # Residua of sky tempereture for mean PWV and Tsun (operation point).
      resTyy[sciSPW]=dPyy.mean()/Gyy[sciSPW]

      # Mapping between SPW:YY and equation number.
      spw2rowYY[sciSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',sciDataID,' data.')
    #--> endif(not flaggged)

  #--> endfor(all sci SPWs)

  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged SPWs/polarization channels.")
    return False


  #-- Grand system assembly and inversion by parts. We use the approach from
  #   extMatInv() for inversion of extended-system matrix. The procedure
  #   performed here looks mysterious (sometimes a factor nSamples emerges or
  #   the RHS vector summed over time-samples), but all directly implies from
  #   application of the inverted matrix of extended grand-system (preserved
  #   in the commented section below) to the RHS vector. We use advantage of
  #   the symmetry of the original grand-system matrix (JJ_K^T=JJ_K).
  #   Advantage: This block-decomposed solution is O(ns^1) in the number of
  #   time-samples, while the 'brutal-force' direct method in the commented
  #   section below scales as O(ns^3) and takes ages...

  #- Calculate the small-block matrices and their inversions needed for
  #  grand-system inversion.

  # Natural (N-block) matrices.
  J_wb_T=np.transpose(J_wb)
  J_K=np.matmul(J_wb_T,J_wb)
  J_K_inv=np.linalg.inv(J_K)

  # Gain/P_inst (G-block) matrices.
  J_gp_T=np.transpose(J_gp)

  # ACD rows (A-block) matrices.
  # Calculate transpositions per sub-block (ambient/hot).
  # Attention!!! NumPy's transpose() makes just another view of a matrix,
  # we have to make a copy, otherwise subsequent scaling would scale the
  # original J_cdA matrix, too!
  J_cdA_T=np.transpose(J_cdA).copy()
  J_cdH_T=np.transpose(J_cdH).copy()

  # Increase importance of the ACD rows by calibration weight.
  # Only the transposed A-block matrices should be scaled up!
  J_cdA_T*=calWeight
  J_cdH_T*=calWeight
```

```
    # Compose the full block and its transposition. The transposition to be
    # scaled up - see above. Or, alternativelly, it can be built as
    # np.hstack([J_cdA_T,J_cdH_T]) from already scaled-up transposed sub-blocks.
    J_cd=np.vstack([J_cdA,J_cdH])
    J_cd_T=calWeight*np.transpose(J_cd)

    #- Base-matrix extensions.

    # Right and lower bands: Their (nEqs x 3) and (3 x nEqs) repeating blocks.
    Rb=np.matmul(J_wb_T,J_gp)
    Lb=np.transpose(Rb)

    # Lower-right corner
    LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

    #- Frequently repeating matrix products. Cf. with extMatInv() above.

    # Ainv x Rb and Lb x Ainv
    AinvRb=np.matmul(J_K_inv,Rb)
    LbAinv=np.transpose(AinvRb)

    #- Schur complement and its inversion.
    SC=LRc-nSamples*np.matmul(Lb,AinvRb)
    SC_inv=np.linalg.inv(SC)

    #- Blocks of inverted grand-system matrix.

    # Start with the lower-left block as it repeats in upper-left block, too.
    Minv21=-np.matmul(SC_inv,LbAinv)

    # Upper-left block: Off-diagonal terms. The diagonal blocks differ just
    # by added J_K_inv.
    Minv11_OD=-np.matmul(AinvRb,Minv21)

    # Upper-right block.
    Minv12=np.transpose(Minv21)

    # Lower-right block.
    Minv22=SC_inv

    #-- Apply the inverted matrix components to the RHS vector.

    # Sum of the RHS along the time axis, keeping the SPW:POL dimension.
    # Preserve the number of matrix rows, just sum their contents, i.e. make
    # the single-column matrix.
    CCsum=np.reshape(CC.sum(axis=1),(-1,1))

    #- Form the RHS vector for the normal equations (K-form minimizer),
    #  i.e, multiply the raw data by JJ_T

    # Start with the natural-variables dW and dB part.
    C_K_wb=np.matmul(J_wb_T,CC)        # 3 x nSamples matrix

    # Make the same also for its time-sum.
    C_K_wb_sum=np.matmul(J_wb_T,CCsum)    # 3 x 1

    # Follow with the instrumental part.
    C_K_gp=(np.matmul(J_gp_T,CCsum)        # 2*eqCount x 1
```

```
            +np.matmul(J_cdA_T,CCA)
            +np.matmul(J_cdH_T,CCH))

    #- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

    # Constant part of the natural-variables vector (3 x 1)
    nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)

    # Time-varying part of the natural-variables vector. It comes from the
    # addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
    nvVar=np.matmul(J_K_inv,C_K_wb)   # 3 x nSamples

    # Compose the complete natural variables vector.
    varNat=nvVar+nvConst

    # Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
    varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)

    #- Reshape the results - extract our elemenytary variables.

    # Seaparate the natural-vars components.
    dW = varNat[0]
    dBxx=varNat[1]
    dByy=varNat[2]

    # Reshape instrumental parameters to two rows (transpose): 1st is dP_inst,
    # the 2nd dG, one component per SPW:POL.
    dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

    # Extract the shifts in the P_inst and G
    dPinst=dPG[0]
    dGinst=dPG[1]

    #- Analytics: Calculate the errors for the over-determined set of eqs.

    # err=A.x-b: Sci-data part.
    errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
    # Xi square for the sci-data part.
    xiSci=np.sum(errVec*errVec)


    # err=A.x-b: ACD-data part.
    errVecACDamb=np.matmul(J_cdA,varInst)-CCA
    errVecACDhot=np.matmul(J_cdH,varInst)-CCH
    # Xi square for the acd-data part.
    xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)

    # Grand-total Xi-square normalized to the number of equations (sci/acd).
    xiSquare=xiSci/nSamples+xiACD/calWeight

    '''
#--> This is the brutal-force straightforward method for grand-system
#    assembly and solution. It works, but takes ages. It is kept here for
#    facilitation of understanding of the above efficent but bit mysterious
#    procedure, which solves the grand-system matrix inversion block-wise,
#    using the method explained in extMatInv().

#-- Loop over time samples: Grand-system assembly.
```

```
 # Grand system matrix (placeholder/restart).
JJ=np.zeros((0,2*eqCount+3*nSamples))

 # The RHS vector: We have assembled it already, just needs to be re-shaped
 # from the form of block-matrix eqBlockSize x nSamples to a 1-column matrix.
 # We need to change the order of elements: First by SPW:POL channel, only
  # after that by time (so far it is reversed) - that is the reason for
  # transposing first.

CC=np.reshape(np.transpose(CC),(-1,1))

 # Append the ambient- and hot-load powers. Scale them up by sqrt(calWeight).

 scUp=sqrt(calWeight)
CC=np.append(CC,[scUp*CCA,scUp*CCH])


 for tt in range(nSamples):

   # Compose a block-row of the grand matrix.
   JJ_row=np.hstack([np.zeros((eqBlockSize,3*tt)),J_wb,
                  np.zeros((eqBlockSize,3*(nSamples-tt-1))),J_gp])


    # Append it to the grand matrix.
   JJ=np.vstack([JJ,JJ_row])

 #--> endfor(t-samples)

 #- Append the ACD-related rows.

 # Scale-up the calibration matrices by the factor sqrt(calWeight).
J_cdA*=scUp
J_cdH*=scUp

 # Compose the full ACD block.
J_cd=np.vstack([J_cdA,J_cdH])

 # Form the ACD-equations row.
JJ_row=np.hstack([np.zeros((2*eqBlockSize,3*nSamples)),J_cd])
 JJ=np.vstack([JJ,JJ_row])

 #-- Grand system solution - brutal

 #- Make a set of normal equations (quadratic-form minimizer).

 JJ_T=np.transpose(JJ)

 J_KK=np.matmul(JJ_T,JJ)

 JJ_inv=np.linalg.inv(J_KK)

C_KK=np.matmul(JJ_T,CC)

 #- Solve the quadratic-form minimizer.
varVec=np.linalg.solve(J_KK,C_KK)
```

```
#- Decompose the solution to the instrumental and natural parts and re-form
#  them into proper shapes.

varNatur=varVec[0:3*nSamples]
varInst=varVec[3*nSamples:]

# P_inst and G for each non-flagged SPW:Pol channel.
dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

 # Extract the shifts in the P_inst and G
dPinst=dPG[0]
dGinst=dPG[1]

# dW, dBxx, and dByy variations for each time sample.
# Do transpose - make the variations time-series to be matrix rows.
varNatur=np.transpose(np.reshape(varNatur,(nSamples,3)))

# Now we can easilly decompose the time series.
dW = varNatur[0]
dBxx=varNatur[1]
dByy=varNatur[2]

#- Analytics: Calculate the errors for overdetermined set of eqs.

 # err=A.x-b
errVec=np.matmul(JJ,varVec)-CC

 # Limit the errorrs to 'natural' variables.
errVec=errVec[0:nSamples*eqBlockSize]
# Reshape the long column vector to the array of eqBlockSize vectors,
# each representing the nSamples time series.
errVec=np.transpose(np.reshape(errVec,(nSamples,eqBlockSize)))

#--> End of the brutal-force method
 '''

 #- All variations should tend to zero means.

 dW_mean=dW.mean()
 dBxx_mean=dBxx.mean()
 dByy_mean=dByy.mean()

 dB_mean=np.sqrt(dBxx_mean**2+dByy_mean**2)

 # Update stationary part of PWV and solar T_B  by means of their variations.
 w0+=dW_mean
 Tsun0xx+=dBxx_mean
 Tsun0yy+=dByy_mean

 # Update also the P_inst and G by the found differences.
 # Start with XX pol.
 for sciSPW,eqCount in spw2rowXX.items():
   # Update the P_inst and G.
   PIxx[sciSPW]+=dPinst[eqCount]
   Gxx[sciSPW]+=dGinst[eqCount]
 #--> endfor(SPWs with valid XX pol)

 # YY pol.
```

```
  for sciSPW,eqCount in spw2rowYY.items():
    # Update the P_inst and G.
    PIyy[sciSPW]+=dPinst[eqCount]
    Gyy[sciSPW]+=dGinst[eqCount]
  #--> endfor(SPWs with valid YY pol)

  #- Convergence diagnostics.

  itCount+=1

  # Updated scan-averaged PWV and Tsun after completed iteration.
  print('vvvvvvv\n After iteration no. ',itCount)
  print('\nMean PWV for the scan,',tgtScan,' w0 = ',w0,
        ", mean Tsun (XX/YY) = ",Tsun0xx,'/',Tsun0yy)

  # Mean values of dW and dB fluctuations should tend to zero.
  print("dW mean: ",dW_mean)
  print("dB mean: ",dB_mean)

  # Sky-brightness residua should tend to zero, too.
  # Convert the spwId maps to np.arrays first. We have to go via 'list'.
  # Scale back form kiloKelvins to Kelvins.
  resTxxArr=1000.0*np.array(list(resTxx.values()))
  resTyyArr=1000.0*np.array(list(resTyy.values()))

  print("\nres Txx: ", resTxxArr,"\nmean +/- RMS: ",
        resTxxArr.mean(),"+/-",resTxxArr.std())
  print("\nres Tyy: ", resTyyArr,"\nmean +/- RMS: ",
        resTyyArr.mean(),"+/-",resTyyArr.std())

  print("\nInstrumental characteristics shift [dP_inst, dG]:",dPinst,dGinst)

  print('^^^^^^^\n\n')

#--> endwhile(non-zero-means) / end of main iteration loop.

#-- Diagnostic plots.
#   Make a plot comparing the measured fluctuations, their part attributed to
#   the intrinsic brightness changes, and error in eqs-system solution.

#-Set the plotting device.

plt.rcParams["figure.figsize"] = (30,20)
# Make XX and YY plots as subplots (horizontally)
ff,sps=plt.subplots(1,2)

# X-axis label just for the bottom sub-plot.
sps[0].set_xlabel('Time [MJDsec]')
sps[1].set_xlabel('Time [MJDsec]')

# Add the antenna name as a plot title.
plt.suptitle('Antenna '+str(antID))

# Color-map for SPWs, incl. line-style.
# We have minimum 2 and maximum 12 SPWs in the spwList argument.
# If we use all 12 spws, linestyle correspondence to the spw type follows:
csline={5:'-r',7:'-g',9:'-b',11:'-k',         # AC resolved
        6:'--r',8:'--g',10:'--b',12:'--k',    # AC integrated
```

```
        0:':r',1:':g',2:':b',3:':k'}          # SQLD

#- Make the *analytical* plots - least-square residuals etc.

# XX pol

for spw,eqRow in spw2rowXX.items():
  # Panel row 1: Residuals/errors scaled to temperature in Kelvins.
  sps[0].plot(time,1000.0*errVec[eqRow]/Gxx[spw],csline[spw])

# YY pol

for spw,eqRow in spw2rowYY.items():
  # Panel row 1: Residuals/errors scaled to temperature in Kelvins.
  sps[1].plot(time,1000.0*errVec[eqRow]/Gyy[spw],csline[spw])

#- Save the plots to a file and close.
plt.savefig(plotDir+'/errs_ant_'+str(antID)+'.png')
plt.show()
plt.close()

#-- Make the full-solution plots: PWV(t) and T_b(t).

# Add variations to their time-averages. Temperatures are scaled back
# from kiloKelvins to Kelvins.
pwv=w0+dW
TsunXX=1000.0*(Tsun0xx+dBxx)
TsunYY=1000.0*(Tsun0yy+dByy)

# Set the size of the plot
plt.rcParams["figure.figsize"] = (10,5)
# Make slow-WV and fast-WV plots as subplots (horizontally)
ff,(sp1,sp2)=plt.subplots(1,2)
# Add the antenna name as a plot title.
plt.suptitle('Antenna '+str(antID))

# Left panel: Apparent target brightness per polarization.
sp1.plot(time,TsunXX,'-r')
sp1.plot(time,TsunYY,'-b')

# Right panel: PWV.
sp2.plot(time,pwv,'-g')

plt.savefig(plotDir+'/dWdB/dWdB_ant_'+str(antID)+'.png')
plt.show()
plt.close()

#-- Show, how much the instrumental variables G and P_inst have changed.

# The instrumental characteristics are maps, we cannot process them as arrays.

GxxChange={}
GyyChange={}

PIxxChange={}
PIyyChange={}

for spw in spw2rowXX.keys():
```

```
    GxxChange[spw]=(Gxx[spw]-GxxOrig[spw])/GxxOrig[spw]
    PIxxChange[spw]=(PIxx[spw]-PIxxOrig[spw])/PIxxOrig[spw]

  for spw in spw2rowYY.keys():

    GyyChange[spw]=(Gyy[spw]-GyyOrig[spw])/GyyOrig[spw]
    PIyyChange[spw]=(PIyy[spw]-PIyyOrig[spw])/PIyyOrig[spw]

  print("---")

  print("G_xx relative changes:", GxxChange)
  print("G_yy relative change:",GyyChange)

  print("---")

  print("P_inst XX relative change:",PIxxChange)
  print("P_inst YY relative change:",PIyyChange)

  print("---")

  # Print total quadratic error for the variables.
  print("\n======= Xi-squares =======     SCI: ",xiSci,
        ", ACD:",xiACD,
        ", Total:", xiSquare,"\n")


  #-- Export the results.

  # 'Compactify' the per-polarization T_B arrays.
  Tsun=np.array([TsunXX,TsunYY])

  return {'time':time,'pwv':pwv,'T_B':Tsun,'J':J_wb,'C':CC}



#------------------------------------------------------------------------------

def dWdBdG2(vis,atmScan,tgtScan,myAtm,antID,spwList,T_tgt=-1.0e6,
            flaggedData=[],myMS=None,myMsmd=None,acdLoads=None):
  """
  The dWdBdG() extended by a bond to the expected time-averaged target
  brightness temperature *T_tgt* (in Kelvins).

  Rationale: When observing the target, e.g. the Sun, we expect to obtain the
  same value of its time-averaged brightness temperature at all antennas.
  The time variations of T_B might be a bit different per antenna due to
  scintillation, but the time average over ~7minute scan should give the same
  value. If we use dWdBdG(), which calculates T_B(t) independently per
  antenna (and yet per XX/YY polarization), it is not very true: The T_Bs
  are somehow centerred around the theoretical value calculated by Alissandrikis
  formula (see below in the code) but they have per-antenna scatterring around
  that value with RMS ~200 K - see Fig. 11 in PR16. For that sake, we complete
  our grand-system of equations, described in dWdBdG(), yet by two additional
  equations (one per polarization), which enforce that the <T_B(t)_XX/YY> equals
  to the prescribed value of *T_tgt*, if it is supplied (the default value -1MK
  means "do not use it").
```

Implementation: The new equations make the grand-system yet more complicated, namely, the 'natural' part of the (J^T x J) matrix is no longer block-diagonal but still it is posssible to invert the matrix relativelly 'cheap' way. In addition to N-, G, and A- blocks, knowm from dWdBdG(), we introduce yet the M-block for the two additioonal "Mean" (over the array) equations. Similarly to the A-block equations, that enforce fulfilling of the ACD calibration equalities, and the 'relative strength', of how much we push to the ACD equations to be exactly fulfilled, is expressed by the factor 'calWeight', here we introduce a factor 'mWeight', which quantifies the relative strength of the <T_B(t)^XX/YY>=T_tgt in the grand-total system of equations.

Moreover:

1. Based on the results shown in the PR16, Fig. 11, we have introduced a bit modified antenna efficiencies, set by antenna type (CM, DA, and DV). The goal is to diminish apparent systematic (i.e by antenna type) <T_B(t)> variations - the 'steps' in Fig. 11/PR16. The relative antenna efficiencies have been studied also in the XLS table project22_25_Tb.ods.

2. SQLD spws have significantky shorter integrations and therefore also received powers and gains (because G=dP/dT). In order to compensate this, we scale all equations by the gains, i.e., we express them in temperatures.

3. A new diagnostic tool has been added in order to auto-detect corrupted data. It is based on the error-vector per SPW/pol channel. The SPW:Pol data with extreme error are now reported, in future version they will be auto-flagged.
"""

```
#--- Static parameters

#- Dirs

# Output/diagnostics plots.
plotDir='./sciPWVs_gt'
if (not os.path.exists(plotDir)): os.makedirs(plotDir)
if (not os.path.exists(plotDir+'/dWdB')): os.makedirs(plotDir+'/dWdB')


#- Tools, global data

# Flags used to check, whether we should close the tool later.
ownMS=False
ownMsmd=False

# Open own tools if 'global' not provided
if (not myMS):
  # Open the ms tool
  myMS=mstool()
  myMS.open(vis)
  ownMS=True

if (not myMsmd):
  # Open the msmd tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
```

```
  ownMsmd=True

# Create our dictionary with ACD load temperatures (ambient, hot) if needed.
if (not acdLoads):
  acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)

#- Antenna

antNames=myMsmd.antennanames()
myAntName=antNames[antID]
antType=myAntName[:2]

# Map the antenna efficiencies by antenna type. As per investigation from
# merged T_sun temperatures form 2023 and 2026 data (project_22_25_Tb.ods).
eta_l_map={'CM':0.941,'DA':0.947,'DV':0.956}
eta_F_map={'CM':0.891,'DA':0.897,'DV':0.906}

# Rear-spillover + ohmic antenna efficiency - see project_22_25_Tb.ods
eta_l=eta_l_map[antType]
# Forward-spillover antenna efficiency - see project_22_25_Tb.ods
eta_F=eta_F_map[antType]

# We have two exception that seem to have systematically lower
# antenna efficiencies: DA64 and DV08 - see project_22_25_Tb.ods
if myAntName=='DA64':
  eta_l=0.929
  eta_F=0.880
if myAntName=='DV08':
  eta_l=0.943
  eta_F=0.894

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']

# Optimized weight of the ACD calibration equations in the system.
# Search mark in dWdBdG2(): CW_1
calWeight=1.0e5

#- Target: The Sun by default but we can handle the non-solar scans, too.

# 1. What type of target do we have: The Sun or calibrator (i.e., the skyes)?

# All Sun scans.
solScans=myMsmd.scansforintent('OBSERVE_TARGET#ON_SOURCE')
solarTgt=(tgtScan in solScans)


# 2. The solar target (the default).

# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.300  # kiloKelvins!!!
```

```
# Solar radius.
Rsun=696000 # km


# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.

"""
# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for July 30 2026 Band 5 observation.
'''
Target:       RA=08 37 18.01 DEC=+18 31 23.0
Sun center:   RA=08 38 15.79 DEC=+18 28 47.2
-----------------------------------------------
diff (arcsec, incl. cos(delta)): 822   155.8
'''
x_h=822.0 # arcsec
y_h=155.8  # arcsec

# Earth-Sun distance for July 30 2026 Band 5 observation.
Dsun=1.01508611721886*149597870.700 # km

# *** End of observation-specific part.
"""


# Apparent angular solar radius in arcsec.
r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

# mu=cos(theta). Used for limb-brightenning correction below.
mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

# b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
# in Alissandrakis et al. (2020). Scaled to kiloKelvins!
b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

# Brightness temoerature corrected to limb brightenning - see
# Alissandrakis et al. (2020).
Tsun00=TsunCenter+0.5*b_2*np.log(mu)

# Weight of the array-mean equation <T_B(t)>=T_tgt_arrayMean in the system.
# If T_tgt is not explicitly set (the defaul value is -1MK), do not apply
# this equation at all. This can be achieved by setting its weight to 0.
if (T_tgt < -10000.0):
  # Do not apply T_tgt bond. If mWeight is zero, T_tgt is irrelevant.
  T_tgt=0.0
  mWeight=0.0
else:
  # We use kiloKelvins in calculations, re-scale T_tgt to kK, too.
```

```
  T_tgt*=0.001
  # Set the experimentally found optimized mWeight.
  # Search mark in dWdBdG2(): MW_1
  mWeight=5000.0

# 3. Specific handling of non-solar scans - override the solar defaults.
if (not solarTgt):

  print("dWdBdG2(): Non-solar target. Setting the target T_B to zero.")

  T_tgt=0.0
  mWeight=5000.0

  # No solar radiation is falling into antenna.
  Tsun00=0.0

#-->endif(non-solar target)

#- SPWs

# LO frequency. We take it as a reference frequency for the solar spectrum.
f_LO=1.98e11 # in Hz

# Integration limits for resolved AC SPWs.
startChan=8
stopChan=120

fChans=range(startChan,stopChan)

#- Atmosphere

# Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,' @airMass = ',airMass)
print('Estimated T_B = ',1000.0*Tsun00)
print('.......')

#- SciSPW to AtmSPW mapping.

# How science SPWs map to the SPW 0 - 3 in the ATM tool.
sciAtmMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}

#--- Read-out the sky+src powers for all SPWs in the spwList.

# Store the frequency-averaged sky+src powers in a map.
# spwId -> np.array(nTimeSamples). Do not use 2D array as we index by the SPW
# id, which is sparse in the set of integers.
Pxx={}
Pyy={}
```

```
#- Similar maps for the gains and instrumental powers P_inst=G*T_inst.
#  Here, all gains and P_inst are, in fact, the G* and P_ints* values for the
#  higher-attenuation solar regime. In order to append the ACD calibration
#  equations, whose measurements are performed in the norma-attenuation regime
#  in the preceding ATM scan, we determine also the attenuation factors
#  AF=G*/G and related power offsets AO.

# Frequency-averaged gains. The scale for the spectral-resolved norm. gains.
Gxx={}
Gyy={}

# Attenuation factors G/G*=(P_sky-P0)/(P_off-Po). Attributed to every-spw
# Gxx & Gyy, but actually calculated just in SQLD spws, the rest is mapped.
AFxx={}
AFyy={}

afMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}

# Attenuation-based power offsets AO=-Po*(P_sky-P_off)/(P_off-Po).
# They are non-zero only in SQLD spws.
AOxx={}
AOyy={}

# Spectral gain normalized to 1.0. Acual ACD-based values in spectral AC spws,
# a trivial 1.0 ndarray for integrated-AC and SQLD spws.
gXX={}
gYY={}

# Frequency-averaged instrumental powers.
PIxx={}
PIyy={}

#- Finally, the raw ACD-calibration powers.

# Ambient-load powers.
PxxAmb={}
PyyAmb={}

# Hot-load powers.
PxxHot={}
PyyHot={}

#-- Read the data and ACD calibration tables in a loop over all sciSPWs.

for sciSPW in spwList:

  #- Raw data & ACD tables

  # Read raw science powers for the target defined by targetScan.
  time,P_src=tgt_pow(vis,tgtScan,antID,sciSPW,myMS=myMS,myMsmd=myMsmd,
                     subScans='on')

  PxxS=P_src[0]
  PyyS=P_src[1]

  # Read ACD calibrations - gains,Tinst, and calibration powers.
  calDict=acd_calTabs(vis,atmScan,antID,sciSPW,tgtScan=tgtScan,
```

```
                    myMS=myMS,myMsmd=myMsmd,acdLoads=acdLoads)

Pcals=calDict['calPows']

PxxAmbS=Pcals['P_amb'][0]
PyyAmbS=Pcals['P_amb'][1]

PxxHotS=Pcals['P_hot'][0]
PyyHotS=Pcals['P_hot'][1]


# We have different handling for frequency-resolved and integrated SPWs.
# Find the number of channels in the observed SPW.
nChansO=PxxS.shape[0]

if (nChansO > 1):

  #- Resolved AC SPWs.
  #  Trim the spectral edges by posing the integration limits and average
  #  over the remaining fChans channels.

  # Science-target powers.
  PxxM=np.mean(PxxS[fChans],axis=0)
  PyyM=np.mean(PyyS[fChans],axis=0)


  # Calibration (ACD) powers.
  PxxAmbM=np.mean(PxxAmbS[fChans],axis=0)
  PyyAmbM=np.mean(PyyAmbS[fChans],axis=0)

  PxxHotM=np.mean(PxxHotS[fChans],axis=0)
  PyyHotM=np.mean(PyyHotS[fChans],axis=0)

  #- Instrumental factors.
  #  Store the normalized (to 1.0) spectral gains (arrays, constant) and
  #  their scales (numbers, to be re-fined by the solver) separately.

  # Raw G/G* gains in a full SPW. We may flip the array (below) so better
  # to do so before its limiting to fChans - wee need not to think whether
  # the fChans central range is placed symmetricaly within its SPW.

  if (solarTgt):
    # Solar gain G* has shape 2 x nPols x nChans. The 1st index 0 means we
    # pick the *leading* P_off subscan - see the note in tgt_Tb().
    GxxS=1.0/(calDict['iG*'][0,0])
    GyyS=1.0/(calDict['iG*'][0,1])
  else:
    # Non-solar standard gain G has shape nPols x nChans.
    GxxS=1.0/(calDict['iG'][0])
    GyyS=1.0/(calDict['iG'][1])

  # T_inst in full SPW. The same holds for flipped/reversed SPWs as for G*.
  TinstXX=calDict['T_inst'][0]
  TinstYY=calDict['T_inst'][1]

  # Check, whether we need flip them - this is the case of the LSB spws,
  # which have decreasing/reversed frequency channels.
  spwFreqs=calDict['freqs']
```

```
  if (spwFreqs[0] > spwFreqs[1]):
    # Reversed SPW, flip the gain and T_inst arrays.

    GxxS=np.flip(GxxS)
    GyyS=np.flip(GyyS)

    TinstXX=np.flip(TinstXX)
    TinstYY=np.flip(TinstYY)

  # Now it is safe to select the central fChans band-pass.
  GxxS=GxxS[fChans]
  GyyS=GyyS[fChans]

  TinstXX=TinstXX[fChans]
  TinstYY=TinstYY[fChans]

  # Calculate the unity-normalized gains by dividing out the mean value.
  GxxM=GxxS.mean()
  GyyM=GyyS.mean()

  gXX[sciSPW]=GxxS/GxxM
  gYY[sciSPW]=GyyS/GyyM

  # Instrumental power averaged over our fChans range.
  PIxxM=np.mean(GxxS*TinstXX)
  PIyyM=np.mean(GyyS*TinstYY)

  # Keep the time-axis of the last resolved SPW as a reference timebase,
  # to which we intrepolate the SQLD signals.
  timeBase=time

  # SQLD spws have an instrumental offset ralated to Pzero. It is calculated
  # below, in the single-channel SPWs branch. Here we set it to zero
  # because Pzero for AC-resolved spws is zero.
  AOxx[sciSPW]=0.0
  AOyy[sciSPW]=0.0


else:
  #- Integrated ACs or SQLDs.
  #  The single channel already represents a mean over the spw band-pass.

  # Single-channel science data.
  PxxM=PxxS[0]
  PyyM=PyyS[0]


  # Channel-integrated SPWs may have higher cadence. For SQLDs it is true
  # always, but it may be the case of integrated ACs, too. Let us
  # put all data to the same, slowest, timebase (as per resoved AC spws).

  # Compare the cadence (elementary time-step, a.k.a. integration).
  if ((timeBase[1]-timeBase[0]) > (time[1]-time[0])):

    # Indeed, we have higher cadence. Re-bin in time.
    # Use the timeBase defined as a time-axis for resolved SPW.
```

```
    print('---> SPW',sciSPW,
          'has higher cadence: Rebinning to time-line of AC-resolved SPW.')

    PxxM=rebin2(PxxM,time,timeBase)
    PyyM=rebin2(PyyM,time,timeBase)
    time=timeBase

  #--> endif(higher cadence)

  # The attenuation factors AF=(P_sky-Po)/(P_off-Po) and offsets AO should
  # be, moreover, extracted from the SQLDss (ACs use mapping afMap
  # defined above) for the *solar* target. In non-solar they are trivial.
if  sciSPW in  [0,1,2,3,13,14,15,16]:

    if (solarTgt):
      # Extract the caliibration powers relevant for AF calculation
      Patm=Pcals['P_sky']
      # Poff is the only calPow that has a structure 2 x nPols x nChans,
      # similarly like gain. The others are just nPols x nChans 2D arrays.
      # Here, the first dimension '2' is for leading/trailing Poff subscans
      # and we pick the leading one (see the gain extraction above).
      Poff=Pcals['P_off'][0]
      Pzero=Pcals['P_zero']

      # Calculate AFs.
      # The 1st index in powers is pol, the 2nd means (the single) channel.
      AFxx[sciSPW]=(Patm[0,0]-Pzero[0,0])/(Poff[0,0]-Pzero[0,0])
      AFyy[sciSPW]=(Patm[1,0]-Pzero[1,0])/(Poff[1,0]-Pzero[1,0])

      # Calculates AOs.
      AOxx[sciSPW]=Pzero[0,0]*(AFxx[sciSPW]-1.0)
      AOyy[sciSPW]=Pzero[1,0]*(AFyy[sciSPW]-1.0)
    else:
      # Non-solar: Trivial.
      AFxx[sciSPW]=1.0
      AFyy[sciSPW]=1.0

      AOxx[sciSPW]=0.0
      AOyy[sciSPW]=0.0

    #--> endif(solar target)

else:
    # For non-SQLD spws the offsets are zero because Pzero is zero.
    # The AFs are mapped from the corresponding SQLD spw.
    AOxx[sciSPW]=0.0
    AOyy[sciSPW]=0.0

  #--> endif(SQLD spws)

  # Calibration data.
PxxAmbM=PxxAmbS[0]
PyyAmbM=PyyAmbS[0]

PxxHotM=PxxHotS[0]
PyyHotM=PyyHotS[0]
```

```
    #- Instrumental factors.

    # Gains: Solar or non-solar scan? Anyway, the last index '0' picks the
    # single channel of the integrated SPWs as its own average.
    if (solarTgt):
      # Solar gain G* has shape 2 x nPols x nChans. The 1st index 0 means we
      # pick the *leading* P_off subscan - see the note in tgt_Tb().
      GxxM=1.0/(calDict['iG*'][0,0,0])
      GyyM=1.0/(calDict['iG*'][0,1,0])
    else:
      # Non-solar standard gain G has shape nPols x nChans.
      GxxM=1.0/(calDict['iG'][0,0])
      GyyM=1.0/(calDict['iG'][1,0])

    # Create a fake spectral normalized gain for integrated SPWs.
    # We use it in products with the ATM arrays (e.g. transmissivity), which
    # are always spectral.
    gXX[sciSPW]=np.ones(len(fChans))
    gYY[sciSPW]=np.ones(len(fChans))

    # Instrumental power: The channel 0 is the average already
    PIxxM=GxxM*calDict['T_inst'][0,0]
    PIyyM=GyyM*calDict['T_inst'][1,0]
  #--> endif(resolved SPW)


  # Append the frequency-averged time series for each SPW to the maps.
  Pxx[sciSPW]=PxxM
  Pyy[sciSPW]=PyyM

  # The calibration powers are already time-averaged.
  PxxAmb[sciSPW]=PxxAmbM
  PyyAmb[sciSPW]=PyyAmbM

  PxxHot[sciSPW]=PxxHotM
  PyyHot[sciSPW]=PyyHotM

  # Set the ACD-based gains (scales) and instrumental contributions.
  Gxx[sciSPW]=GxxM
  Gyy[sciSPW]=GyyM

  PIxx[sciSPW]=PIxxM
  PIyy[sciSPW]=PIyyM


  print('SPW ',sciSPW,': <Tsky+src_XX> = ',
        (Pxx[sciSPW].mean()-PIxx[sciSPW])/Gxx[sciSPW],'+/-',
        Pxx[sciSPW].std()/Gxx[sciSPW],
        '..... Pxx=',Pxx[sciSPW].mean(),'Gxx=',Gxx[sciSPW],
        'PIxx=',PIxx[sciSPW])
  print('          <Tsky+src_YY> = ',
        (Pyy[sciSPW].mean()-PIyy[sciSPW])/Gyy[sciSPW],'+/-',
        Pyy[sciSPW].std()/Gyy[sciSPW])

#-->endfor(all sciSPWs in the list)

# Save the original gains and instrumental powers for comparison with
# the updated values after the solution.
```

```
# Note: We have to use copy.deepcopy(), the simple assignment operator
# copies just the reference!

GxxOrig=deepcopy(Gxx)
GyyOrig=deepcopy(Gyy)

PIxxOrig=deepcopy(PIxx)
PIyyOrig=deepcopy(PIyy)

#-- Dimensions of the system.

# Count the SPW in the list.
numSpw=len(spwList)
# Count the time samples in data.
nSamples=len(time)

# How many net equations do we have per block of 1 time-sample.
# Be carefull: Do not flag the data that are *not* present in the spwList!
eqBlockSize=2*numSpw-len(flaggedData)


# The data has been read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()
if (ownMS): myMS.close()

#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).

# Pre-calculated frequently repeating factor: Inverse number of time samples.
nsInv=1.0/nSamples

# Number of the model spectral windows.
nAtmSPWs=myAtm.getNumSpectralWindows()
# Number of all channels in the model SPWs. They have all the same structure.
nChans=myAtm.getNumChan(0)
# Number of central band-pass (with edges trimmed) channels.
cChans=len(fChans)

# Frequencies of channels in our model SPWs (central band-pass only).
spw_freqs=np.zeros((nAtmSPWs,cChans))

for atmSpw in range(nAtmSPWs):

  fMin=myAtm.getMinFreq(atmSpw)['value'][0]*1.0e9 # GHz to Hz
  df=myAtm.getChanSep(atmSpw)['value'][0]*1.0e6   # MHz to Hz

  spw_freqs[atmSpw]=(fMin+df*np.arange(nChans))[fChans]

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,cChans))
dTdW=np.zeros((nAtmSPWs,cChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,cChans))
tau_1=np.zeros((nAtmSPWs,cChans))

# Target normalised spectrum
```

```
if (solarTgt):
  # Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
  SF=1.0+b_2/Tsun00*np.log(spw_freqs/f_LO)
else:
  # A flat normalized spectrum for non-solar target
  SF=np.ones(cChans)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
# We used those as weights in calculation of epsilon - the normalised
# inter-iteration change of operation-point at the very end of each iteration.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin
dG_max=0.001       # Relative scale dG/G
dTinst_max=0.001   # dPinst/G in kiloKelvins

# Iteration counter.
itCount=0
# Normalized inter-iteration change of operation point - its initial value.
# Anything bigger than 1.0
epsilon=10.0

# Set the starting values for iterations.
w0=pwv0
Tsun0xx=Tsun00
Tsun0yy=Tsun00

#-- Solution iterations: Minimize the mean value of variations.
while ((epsilon>1.0) and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample

  # N-block matrix: Jacobian part for [dW,dT_Bxx,dT_Byy] 'natural' variables.
  J_wb=np.zeros((eqBlockSize,3)) # 3 vars repeating in all spws
  # G-block matrix: Jacobian part for [dG,dP_inst] 'instrumental' variables.
  J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each SPW:Pol.
  # A-block matrix: Jacobian part for [dG,dP_inst] for ACD calibration powers.
  J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, ambient.
  J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, hot.
  # M-block matrix: Jacobian part for [dW,dT_Bxx,dT_Byy] in array-mean eqs.
  # This matrix is constant and can be defined already here.
  J_M=np.array([[0.0, nsInv, 0.0],
                [0.0, 0.0, nsInv]])


  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))
  # RHS-vector 'appendix' - amb. & hot ACD calibration powers (time-averaged).
```

```
CCA=np.zeros((eqBlockSize,1))
CCH=np.zeros((eqBlockSize,1))
# RHS-vector 'appendix 2' - array-mean values of target T_B
CCM=np.zeros((2,1))

# Rezidual mean sky temperature - after subtracted brightness of the
# atmosphere (+CMB) and (attenuated) source. We keep it for all sci SPWs and
# both polarizations and print it out at the end of every iteration cycle
# as a diagnostics of convergence and solution quality.

resTxx={}
resTyy={}

# Mapping between spw IDs and the row number in the system of equations.
# We need it for final diagnostic plots. It may be non-trivial as some
# spw:pol pairs may be flagged and ignored, causing irregularity in the map.

spw2rowXX={}
spw2rowYY={}

#- Re-calculate ATM model with updated <PWV> and <Tsun>.

# Set current estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

#- Loop over 4 ATM spws - these correspond to the channel-resolved ACs.
#  Calculate factors dependent only on the mmodel, not the data.
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Limit the spectrum to the central fChans reagion as we have trimmed
  # the edge channels in ACD-based spectral normalized gain gXX/gYY.
  # Scale to kiloKelvins.
  TbsW0[atmSPW]=0.001*Tebb[fChans]

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value'][fChans]
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value'][fChans]

  # In kiloKelvins/mm
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Transmissivity (spectral). Limitted to fChans.
  trans[atmSPW]=np.exp(-opacity[fChans])
  # Wet opacity per 1mm PWV scaled by airmass. Limitted to fChans.
  tau_1[atmSPW]=(airMass/w0)*tau_wet[fChans]
```

```
#--> endfor(modelSPWs)

#- Loop over all SCI spws - combine model-only dependent factors with
#  sciSPW-dependent data and instrumental factors and build the system.

# Count the actual number of equations - some rows may be flagged.
eqCount=0

for sciSPW in spwList:

  # Map the sci SPW to its corresponding ATM counterpart
  atmSPW=sciAtmMap[sciSPW]

  #- Model (_M) factors.

  # Tebb and its slope.
  TbsW0_M=TbsW0[atmSPW]
  dTdW_M=dTdW[atmSPW]

  # Transmissivity and unit-PWV opacity.
  trans_M=trans[atmSPW]
  tau_1M=tau_1[atmSPW]

  # Solar-flux model spectrum (normalized).
  SF_M=SF[atmSPW]

  #- Separate procedure by polarizations for combination with real data.

  #- Pol XX

  # Full identificator of data: sciSPW:pol
  sciDataID=str(sciSPW)+':XX'

  if (sciDataID not in flaggedData):

    #- Model + instrumental factors combined.

    # We do weighting of equations by inverse gain in order to compensate
    # different powers (and gains) in AC and SQLD SPWs. Consequently,
    # the equations are expressed in equivalent temperatures, which are
    # (approximatelly) the same for all SPWs.
    # Weight factor
    WF=1.0/Gxx[sciSPW]

    # Sensitivity factor for PWV.
    SWxx=(-eta_l*eta_F*Tsun0xx*np.mean(gXX[sciSPW]*trans_M*tau_1M*SF_M)
          +(1.0-eta_F+eta_F*eta_l)*np.mean(gXX[sciSPW]*dTdW_M))*Gxx[sciSPW]

    # Sensitivity factor for the (apparent) solar-brigthness temperature.
    SBxx=Gxx[sciSPW]*eta_l*eta_F*np.mean(gXX[sciSPW]*trans_M*SF_M)

    # Model T_sky as viewed by the receiver with spectral gain gXX.
    TskyXX=(eta_l*eta_F*Tsun0xx*np.mean(trans_M*SF_M*gXX[sciSPW])
            +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M*gXX[sciSPW])
            +eta_F*(1.0-eta_l)*Tamb)

    #- Build the matrix: Natural (dW & dB sensitivities) block.
```

```
        # Set the (weighted) sensitivity factors for the current SPW:pol as
        # a row in the N-block/tile for the linearized-system Jacobian matrix.
        J_wb[eqCount]=WF*np.array([SWxx,SBxx,0.0])

        #- Build the matrix: Instrumental (G & P_inst sensitivities) block.

        # We have to prepend a proper offset for the G and P_inst vars as
        # they are independendt for each SPW and polarization.
        bOffset=2*eqCount

        # G-block scaled by the inverse gain.
        J_gp[eqCount,bOffset]=WF
        J_gp[eqCount,bOffset+1]=TskyXX*WF

        #- Build the matrix: ACD powers (amb & hot) 2-block.

        # Ambient-power sub-block. Scale between the G* / P_inst* used in
        # J_wb and J_gp, and G / P_inst used in the ACD cal equations by the
        # attenuation factor AF=G*/G=(P_sky-Po)/(P_off-Po).

        # Attenuation factor (af>1) is calculated only in SQLD and mapped.
        af=AFxx[afMap[sciSPW]]

        # A-ambient sub-block Jacobian elements weighted by inverse gain.
        J_cdA[eqCount,bOffset]=af*WF
        J_cdA[eqCount,bOffset+1]=af*Tamb*WF

        # A-hot-power sub-block weighetd by inverse gain.
        J_cdH[eqCount,bOffset]=af*WF
        J_cdH[eqCount,bOffset+1]=af*Thot*WF

        #- Build the RHS vector: Target data

        # RHS: Pxx(t) with their stationary (in the working point) instrumental
        # and ATM-based parts subtracted.
        dPxx=Pxx[sciSPW]-PIxx[sciSPW]-Gxx[sciSPW]*TskyXX

        # Set it as the RHS component, weighted by inverse gain.
        CC[eqCount]=dPxx*WF

        #- Ammend the RHS vector: Calibration data.

        # Attenuation-based power offset. We have it defined for all SPWs,
        # however, non-zero it is only in SQLDs.
        aoff=AOxx[sciSPW]

        # Ambient-load power with its working-point model part subtracted.
        CCA[eqCount]=WF*(PxxAmb[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Tamb
                         +aoff)
        # Hot-load power with its working-point model part subtracted.
        CCH[eqCount]=WF*(PxxHot[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Thot
                         +aoff)
        #- Diagnostics

        # Residua of sky tempereture for mean PWV and Tsun (operation point).
        resTxx[sciSPW]=dPxx.mean()/Gxx[sciSPW]
```

```
    # Mapping between SPW:XX and equation number.
    spw2rowXX[sciSPW]=eqCount

    eqCount+=1

  else: print('Ignoring flagged ',sciDataID,' data.')
  #--> endif(not flaggged)

  #- Pol YY

  # Full identificator of data: sciSPW:pol
  sciDataID=str(sciSPW)+':YY'

  if (sciDataID not in flaggedData):

    #- Model + instrumental factors combined.

    # Weight factor - see explanation at XX pol above.
    WF=1.0/Gyy[sciSPW]

    # Sensitivity factor for PWV (weighted by 1/G).
    SWyy=(-eta_l*eta_F*Tsun0yy*np.mean(gYY[sciSPW]*trans_M*tau_1M*SF_M)
          +(1.0-eta_F+eta_F*eta_l)*np.mean(gYY[sciSPW]*dTdW_M))*Gyy[sciSPW]

    # Sensitivity factor for the (apparent) solar T_B (weighted by 1/G).
    SByy=Gyy[sciSPW]*eta_l*eta_F*np.mean(gYY[sciSPW]*trans_M*SF_M)

    # Model T_sky as viewed by the receiver with spectral gain gYY.
    TskyYY=(eta_l*eta_F*Tsun0yy*np.mean(trans_M*SF_M*gYY[sciSPW])
            +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M*gYY[sciSPW])
            +eta_F*(1.0-eta_l)*Tamb)

    #- Build the matrix: Natural (dW & dB sensitivities) block.

    # Set the (weighted) sensitivity factors for the current SPW:pol as
    # a row in the block/tile for the linearized-system Jacobian matrix.
    J_wb[eqCount]=WF*np.array([SWyy,0.0,SByy])

    #- Build the matrix: Instrumental (G & P_inst sensitivities) block.

    # We have to prepend a proper offset for the G and P_inst vars as
    # they are independendt for each SPW and polarization.
    bOffset=2*eqCount

    J_gp[eqCount,bOffset]=WF
    J_gp[eqCount,bOffset+1]=TskyYY*WF

    #- Build the matrix: ACD powers 2-block.

    # Attenuation factor G*/G. Mapped from the corresponding SQLD.
    af=AFyy[afMap[sciSPW]]

    # Ambient-power sub-block.
    J_cdA[eqCount,bOffset]=af*WF
    J_cdA[eqCount,bOffset+1]=af*Tamb*WF

    # Hot-power sub-block.
    J_cdH[eqCount,bOffset]=af*WF
```

```
      J_cdH[eqCount,bOffset+1]=af*Thot*WF

      #- Build the RHS vector: Target data.

      # RHS: Pyy(t) with their stationary (in the working point) instrumental
      # and ATM-based parts subtracted.
      dPyy=Pyy[sciSPW]-PIyy[sciSPW]-Gyy[sciSPW]*TskyYY

      # Set it as the RHS component.
      CC[eqCount]=dPyy*WF

      #- Ammend the RHS vector: Calibration data.

      # Attenuation-based power offset.
      aoff=AOyy[sciSPW]

      # Ambient-load power with its working-point part subtracted.
      CCA[eqCount]=WF*(PyyAmb[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Tamb
                       +aoff)
      # Hot-load power with its working-point part subtracted.
      CCH[eqCount]=WF*(PyyHot[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Thot
                       +aoff)

      #- Diagnostics

      # Residua of sky tempereture for mean PWV and Tsun (operation point).
      resTyy[sciSPW]=dPyy.mean()/Gyy[sciSPW]

      # Mapping between SPW:YY and equation number.
      spw2rowYY[sciSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',sciDataID,' data.')
    #--> endif(not flaggged)

  #--> endfor(all sci SPWs)

  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged SPW:pol channels.")
    return False


  #-- Grand system assembly and inversion by parts. We use the approach from
  #   extMatInv() for inversion of extended-system matrix.

  #- Ammend the RHS vector: Array-mean target brightness temperatures.
  #  Here we work already with temperatures, thus no 1/G scalling applies.

  # Target T_B with the model at work-point subtracted.
  CCM[0]=T_tgt-Tsun0xx
  CCM[1]=T_tgt-Tsun0yy


  #- Calculate the small-block matrices and their inversions needed for
```

```
#  grand-system inversion.

# 1. Natural (N-block) matrices.
J_wb_T=np.transpose(J_wb)       # N^T
J_K=np.matmul(J_wb_T,J_wb)      # (N^T x N)
J_K_inv=np.linalg.inv(J_K)      # (N^T x N)^{-1}

# 2. Gain/P_inst (G-block) matrices.
J_gp_T=np.transpose(J_gp)       # G^T

# 3. ACD rows (A-block) matrices.
# Calculate transpositions per sub-block (ambient/hot).
# Attention!!! NumPy's transpose() makes just another view of a matrix,
# we have to make a copy, otherwise subsequent scaling would scale the
# original J_cdA and J_cdH matrices, too!

J_cdA_T=np.transpose(J_cdA).copy()     # A^T
J_cdH_T=np.transpose(J_cdH).copy()     # H^T

# Increase importance of the ACD rows by calibration weight.
# Only the transposed A-block matrices should be scaled up!
J_cdA_T*=calWeight
J_cdH_T*=calWeight

# Compose the full block and its transposition. The transposition to be
# scaled up - see above. Or, alternativelly, it can be built as
# np.hstack([J_cdA_T,J_cdH_T]) from already scaled-up transposed sub-blocks.
J_cd=np.vstack([J_cdA,J_cdH]).copy()
J_cd_T=calWeight*np.transpose(J_cd)     # calWeight * (A | H )^T

# 4. Array-mean (M-block) related matrices. Incorporate mWeight into M^T, in
# line with what we have done with calWeight for the A^T and H^T matrices.
M_T=np.array([[0.0, 0.0],
              [nsInv*mWeight, 0.0],
              [0.0,nsInv* mWeight]])    # mWeight*M^T


# Pre-calculated nSamples*mWeight*(M^T x M) matrix.
# Like with A^T x A, the M^T x M product is scaled-up by mWeight just once
# (no mWeight squared!). Moreover, we scale it up here yet by nSamples,
# in order to avoid trivial product nsInv*nSamples=1 in the following calcs.
nsMTM=np.array([[0.0, 0.0, 0.0],
                [0.0, nsInv*mWeight, 0.0],
                [0.0, 0.0, nsInv*mWeight]])


# Frequently repeating blocks. NM_inv plays a role of modified J_K_inv.
NM=J_K+nsMTM                # J_K+nSamples*mWeight*M^T x M
NM_inv=np.linalg.inv(NM)    # NM^(-1)


# Off-diagonal blocks of the inverted natural sub-matrix (Ainv).
J_M_inv=nsInv*(NM_inv-J_K_inv)

#- Base-matrix extensions.

# Right and lower bands: Their (nEqs x 3) and (3 x nEqs) repeating blocks.
Rb=np.matmul(J_wb_T,J_gp)       # (N^T x G)
```

```
    Lb=np.transpose(Rb)             # (G^T x N)

    # Lower-right corner
    LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

    #- Frequently repeating matrix products. Cf. with extMatInv() above.

    # Ainv x Rb and Lb x Ainv
    AinvRb=np.matmul(NM_inv,Rb)
    LbAinv=np.transpose(AinvRb)

    #- Schur complement and its inversion.
    SC=LRc-nSamples*np.matmul(Lb,AinvRb)
    SC_inv=np.linalg.inv(SC)

    #- Blocks of inverted grand-system matrix.

    # Start with the lower-left block as it repeats in upper-left block, too.
    Minv21=-np.matmul(SC_inv,LbAinv)

    # Upper-left block: Off-diagonal terms. The diagonal blocks differ just
    # by added J_K_inv.
    Minv11_OD=J_M_inv-np.matmul(AinvRb,Minv21)

    # Upper-right block.
    Minv12=np.transpose(Minv21)

    # Lower-right block.
    Minv22=SC_inv


    #-- Apply the inverted matrix components to the RHS vector.

    # Sum of the RHS along the time axis, keeping the SPW:POL dimension.
    # Preserve the number of matrix rows, just sum their contents, i.e. make
    # the single-column matrix.
    CCsum=np.reshape(CC.sum(axis=1),(-1,1))

    #- Form the RHS vector for the normal equations (K-form minimizer),
    #  i.e, multiply the raw data by JJ_T

    # Start with the natural-variables dW and dB part.
    C_K_wb=np.matmul(J_wb_T,CC)+np.matmul(M_T,CCM)  # 3 x nSamples matrix

    # Make the same also for its time-sum.
    C_K_wb_sum=np.matmul(J_wb_T,CCsum)+nSamples*np.matmul(M_T,CCM)  # 3 x 1

    # Follow with the instrumental part.
    C_K_gp=(np.matmul(J_gp_T,CCsum)       # 2*eqCount x 1
            +np.matmul(J_cdA_T,CCA)
            +np.matmul(J_cdH_T,CCH))

    #- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

    # Constant part of the natural-variables vector (3 x 1)
    nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)

    # Time-varying part of the natural-variables vector. It comes from the
```

```
# addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
# NB: Actually, with M-blocks included, it contains a small constant part,
# too, equal to J_K_inv x M_T x CCM, but wee keep it here (hidden in
# C_K_wb).
nvVar=np.matmul(J_K_inv,C_K_wb)   # 3 x nSamples

# Compose the complete natural variables vector.
varNat=nvVar+nvConst

# Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)

#- Reshape the results - extract our elementary variables.

# Seaparate the natural-vars components.
dW = varNat[0]
dBxx=varNat[1]
dByy=varNat[2]

# Reshape instrumental parameters to two rows (transpose): 1st is dP_inst,
# the 2nd dG, one component per SPW:POL.
dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

# Extract the shifts in the P_inst and G
dPinst=dPG[0]
dGinst=dPG[1]

#- Analytics: Calculate the errors for the over-determined set of eqs.

# err=f(X)-b =approx. f(X_0)+J(X_0) x (X -X_0) -b = J x dX - [b-f(X_0)]
# In our notation, dX is the [varNat,varInst]^T vector and the
# difference [b-f(X_0)] is [ CC, (CCA | CCH), CCM ]^T
# Now we express the err vector parts for sci, acd, and array-mean eqs.

# err=f(x)-b: Sci-data part.
errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
# Xi square for the sci-data part. Sum over all SPW:Pol channels & times.
# Normalize it to the number of equations.
xiSci=np.sum(errVec*errVec)/(nSamples*eqBlockSize)

# Xi Squares per SPW:Pol channel. Sum/average only over times.
xiSciPerChan=np.sum(errVec*errVec,axis=1)/nSamples
# Its mean and RMS over the SPW:POL channels.
xscMean=xiSciPerChan.mean()
xscRMS=xiSciPerChan.std()
# Find outliers ( > 3 sigma). The badData array contains indices of
# those equations, where the agreement between model and observation is
# exceptionaly weak.
badData=np.flatnonzero(abs(xiSciPerChan-xscMean)>3.0*xscRMS)

# Show all per-channel errors as temperatures (K) and mark (we may auto-flag
# later) the SPW:Pol channels with bad data, if we have some.
# We do not have an inverse map to spw2rowXX/YY, so we have to traverse.

# XX pol
for spw,eqRow in spw2rowXX.items():
  print(str(spw)+':XX: Error:',np.sqrt(xiSciPerChan[eqRow])*1000.0,'K')
  if eqRow in badData: print('\t\t!!! Bad data in',str(spw)+':XX',
```

```
                                (xiSciPerChan[eqRow]-xscMean)/xscRMS,
                                'sigma from the mean.')
  # YY pol
  for spw,eqRow in spw2rowYY.items():
    print(str(spw)+':YY: Error:',np.sqrt(xiSciPerChan[eqRow])*1000.0,'K')
    if eqRow in badData: print('\t\t!!! Bad data in',str(spw)+':YY',
                                  (xiSciPerChan[eqRow]-xscMean)/xscRMS,
                                  'sigma from the mean.')

  # err=f(x)-b: ACD-data part.
  # Actually, this should be scaled by calWeight factor. On the other hand,
  # we are interested in Xi2 normalised per one equation, so we would
  # down-scale in the next calculation by the same factor. In order to avoid
  # useless product calWeight*1/calWeight, we omit it already now.
  errVecACDamb=np.matmul(J_cdA,varInst)-CCA
  errVecACDhot=np.matmul(J_cdH,varInst)-CCH
  # Xi square for the acd-data part.
  # Normalize it to the number of ACD equations. Scale by just single
  # block-size - we have deliberatelly ignored calWeight factor already above.
  xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)
  xiACD/=eqBlockSize

  # err=f(x)-b: Array-mean T_B part.
  # Here, the term J x dX = M x sum (CC_i) = [sum(dTxx_i), sum(dTyy_i) ]^T =
  # nS*[dBxx.mean(),dByy.mean()]^T is zero because dB.mrean() converged to 0.
  errVecMeanTb=-CCM
  # Normalized to one equation - we have mWeight*2 equations, but the same
  # holds for mWeight factor as for the calWeight above.
  xiTbM=np.sum(errVecMeanTb*errVecMeanTb)/2

  # Grand-total Xi-square normalized to the number of equations for
  # the natural/science, ACD, and mean-array sub-blocks.
  xiSquare=xiSci+xiACD+xiTbM

  #- All variations should tend to zero means.

  dW_mean=dW.mean()
  dBxx_mean=dBxx.mean()
  dByy_mean=dByy.mean()

  dB_mean=np.sqrt(dBxx_mean**2+dByy_mean**2)

  # Update stationary part of PWV and solar T_B  by means of their variations.
  w0+=dW_mean
  Tsun0xx+=dBxx_mean
  Tsun0yy+=dByy_mean

  # Update also the P_inst and G by the found differences.
  # Find the maximum inter-iteration change for convergence diagnostics.
  dT_inst_sup=-1.0
  dG_rel_sup=-1.0

  # Start with XX pol.
  for sciSPW,eqCount in spw2rowXX.items():
    # Update the P_inst and G.
    PIxx[sciSPW]+=dPinst[eqCount]
    Gxx[sciSPW]+=dGinst[eqCount]
```

```
    # Change of inst. power expressed in kiloKelvins.
    dT_inst=abs(dPinst[eqCount]/Gxx[sciSPW])
    if (dT_inst > dT_inst_sup): dT_inst_sup=dT_inst

    # Change of gain in the relative scale.
    dG_rel=abs(dGinst[eqCount]/Gxx[sciSPW])
    if (dG_rel > dG_rel_sup): dG_rel_sup=dG_rel
  #--> endfor(SPWs with valid XX pol)

  # YY pol.
  for sciSPW,eqCount in spw2rowYY.items():
    # Update the P_inst and G.
    PIyy[sciSPW]+=dPinst[eqCount]
    Gyy[sciSPW]+=dGinst[eqCount]

    # Change of inst. power expressed in kiloKelvins.
    dT_inst=abs(dPinst[eqCount]/Gyy[sciSPW])
    if (dT_inst > dT_inst_sup): dT_inst_sup=dT_inst

    # Change of gain in the relative scale.
    dG_rel=abs(dGinst[eqCount]/Gyy[sciSPW])
    if (dG_rel > dG_rel_sup): dG_rel_sup=dG_rel
  #--> endfor(SPWs with valid YY pol)

  #-- Convergence diagnostics / Iteration control.

  # Calculate the normalized inter-iteration change of operation point.
  # The factor 0.5 comes from sqrt(1/4) as we have 4 'orthogonal' variables.
  epsilon=0.5*np.sqrt((dW_mean/dwMean_max)**2+(dB_mean/dbMean_max)**2+
                      (dG_rel_sup/dG_max)**2+(dT_inst_sup/dTinst_max)**2)

  # Increment the iteration counter
  itCount+=1

  #- Diagnostic print-outs per-iteration.

  # Updated scan-averaged PWV and Tsun after completed iteration.
  print('vvvvvvv\n After iteration no. ',itCount)
  print('\nMean PWV for the scan,',tgtScan,' w0 = ',w0,'mm',
        ", mean Tsun (XX/YY) = ",1000.0*Tsun0xx,'/',1000.0*Tsun0yy,'K')

  # Mean values of dW and dB fluctuations should tend to zero.
  print("dW mean: ",dW_mean)
  print("dB mean: ",dB_mean)

  # Sky-brightness residua should tend to zero, too.
  # Convert the spwId maps to np.arrays first. We have to go via 'list'.
  # Scale back form kiloKelvins to Kelvins.
  resTxxArr=1000.0*np.array(list(resTxx.values()))
  resTyyArr=1000.0*np.array(list(resTyy.values()))

  print("\nres Txx: ", resTxxArr,"\nmean +/- RMS: ",
        resTxxArr.mean(),"+/-",resTxxArr.std())
  print("\nres Tyy: ", resTyyArr,"\nmean +/- RMS: ",
        resTyyArr.mean(),"+/-",resTyyArr.std())

  print("\nInstrumental characteristics shifts during the iteration:")
  print("--- Pol XX--- ")
```

```
  for sciSPW,eqCount in spw2rowXX.items():
    print("SPW =",sciSPW,"\tdT_inst =",dPinst[eqCount]/Gxx[sciSPW]*1000.0,"K",
          "\tdG/G =",dGinst[eqCount]/Gxx[sciSPW]*100,"%")

  print("--- Pol YY--- ")
  for sciSPW,eqCount in spw2rowYY.items():
    print("SPW =",sciSPW,"\tdT_inst =",dPinst[eqCount]/Gyy[sciSPW]*1000.0,"K",
          "\tdG/G =",dGinst[eqCount]/Gyy[sciSPW]*100,"%")

  print('^^^^^^^\n\n')

#--> endwhile(non-zero-means) / end of the main solution-iteration loop.

#-- Diagnostic plots

#   Make a plot comparing the measured fluctuations, their part attributed to
#   the intrinsic brightness changes, and error in eqs-system solution.

#-Set the plotting device.

plt.rcParams["figure.figsize"] = (30,20)
# Make XX and YY plots as subplots (horizontally)
ff,sps=plt.subplots(1,2)

# Antenna ID formated to 'NN' string comes to the PNG filename.
antIDstr='{:02d}'.format(antID)

# X-axis label just for the bottom sub-plot.
sps[0].set_xlabel('Time [MJDsec]')
sps[1].set_xlabel('Time [MJDsec]')

# Add the antenna ID/name as a plot title.
plt.suptitle('Antenna '+antIDstr+'/'+myAntName)

# Color-map for SPWs, incl. line-style.
# We have minimum 2 and maximum 12 SPWs in the spwList argument.
# If we use all 12 spws, linestyle correspondence to the spw type follows:
csline={5:'-r',7:'-g',9:'-b',11:'-k',         # AC resolved
        6:'--r',8:'--g',10:'--b',12:'--k',    # AC integrated
        0:':r',1:':g',2:':b',3:':k'}          # SQLD

#- Make the *analytical* plots - least-square residuals etc.

# XX pol

for spw,eqRow in spw2rowXX.items():
  # Panel row 1: Residuals/errors scaled to temperature in Kelvins.
  # Note: Be carefull about scaling if we use the WF weight factor above
  # other than the 'natural' WF=1/Gxx[spw]
  sps[0].plot(time,1000.0*errVec[eqRow],csline[spw])

# YY pol

for spw,eqRow in spw2rowYY.items():
  # Panel row 1: Residuals/errors scaled to temperature in Kelvins.
  # Note: Be carefull about scaling if we use the WF weight factor above
  # other than the natural WF=1/Gyy[spw]
  sps[1].plot(time,1000.0*errVec[eqRow],csline[spw])
```

```
#- Save the plots to a file and close.
plt.savefig(plotDir+'/errs_ant_'+antIDstr+'.png')
plt.show()
plt.close()

#- Make the full-solution plots: PWV(t) and T_b(t).

# Add variations to their time-averages. Temperatures are scaled back
# from kiloKelvins to Kelvins.
pwv=w0+dW
TsunXX=1000.0*(Tsun0xx+dBxx)
TsunYY=1000.0*(Tsun0yy+dByy)

# Set the size of the plot
plt.rcParams["figure.figsize"] = (10,5)
# Make slow-WV and fast-WV plots as subplots (horizontally)
ff,(sp1,sp2)=plt.subplots(1,2)
# Add the antenna ID/name as a plot title.
plt.suptitle('Antenna '+antIDstr+'/'+myAntName)

# Left panel: Apparent target brightness per polarization.
sp1.plot(time,TsunXX,'-r')
sp1.plot(time,TsunYY,'-b')

# Right panel: PWV.
sp2.plot(time,pwv,'-g')

plt.savefig(plotDir+'/dWdB/dWdB_ant_'+antIDstr+'.png')
plt.show()
plt.close()

#-- Final diagnostics print-outs.

#- Show, how much the instrumental variables G and P_inst have changed from
#  their nominal ACD-based values.

print("--- Accumulated instrumental characteristics changes ---")

for spw in spw2rowXX.keys():
  # Gain as a relative change in percents.
  dGxx=100.0*(Gxx[spw]-GxxOrig[spw])/GxxOrig[spw]
  # Instrumental power expressed as T_inst change in Kelvins.
  dPxx=1000.0*(PIxx[spw]-PIxxOrig[spw])/Gxx[spw]

  print('\t',spw,':XX: dGxx =',dGxx,'%','\tdP_Ixx =',dPxx,'K')

for spw in spw2rowYY.keys():
   # Gain as a relative change in percents.
  dGyy=100.0*(Gyy[spw]-GyyOrig[spw])/GyyOrig[spw]
  # Instrumental power expressed as T_inst change in Kelvins.
  dPyy=1000.0*(PIyy[spw]-PIyyOrig[spw])/Gyy[spw]

  print('\t',spw,':YY: dGyy =',dGyy,'%','\tdP_Iyy =',dPyy,'K')

#- Print total quadratic error for the variables.
print("\n======= Xi-squares =======     SCI: ",xiSci,
      ", ACD:",xiACD,", Array-mean: ",xiTbM,
```

```
        " | Total-per-equation:", xiSquare,
        "\n-------> Estimated total error per equation:",
        np.sqrt(xiSquare)*1000.0,"K -------\n\n")


  #-- Export the results.

  # 'Compactify' the per-polarization T_B arrays.
  Tsun=np.array([TsunXX,TsunYY])

  return {'time':time,'pwv':pwv,'T_B':Tsun,'J':J_wb,'C':CC}


#--------------------------------------------------------------------------

def dWdB_Tbs_NL(vis,atmScan,tgtScan,myAtm,antID,spwList):
  """
  Like dWdB_Tbs(), but now we do not use linearization. We apply brutal force to
  solve the full, overdetermined system of non-linear equations in least-square
  sense. Intended just for verification of the linearization method.

  Results: It has been found that the difference in PWV is max. 0.01 mm and in
  brightness temperature ~3 K - see PR16. As a result, we further develop just
  the linearized version dWdB_Tbs().
  """

  #--- Static parameters

  #- Dirs

  plotDir='./sciPWVs'

  #- Tools (to be extended by possible global tools as params)

  myMS=mstool()
  myMS.open(vis)

  myMsmd=msmdtool()
  myMsmd.open(vis)


  #- Source

  # Initial guess of the solar brightness temperature.
  Tsun0=6000.0

  # b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
  # in Allisandrakis et al. (2020).
  b_2=-1127.0 # = 2x (-563.5K) as per Table 2 in Allisandrakis et al. (2020).

  # Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
  f_LO=1.98e11

  #- Antenna

  # Rear-spillover + ohmic antenna efficiency (see Tab. ? in White et al. 2017).
  eta_l=0.97
  # Forward-spillover antenna efficiency (see Tab. ? in White et al. 2017).
```

```
eta_F=0.95


# Extract Tamb from ASDM_CADEVICE.
# NB: Now we use aU.getLoadTemperatures() for all antennas and scans and
# then pick just our antenna and scan. Do it better later.
TLoad = aU.getLoadTemperatures(vis, doplot = False,
                               warnIfNoLoadTemperatures = True,
                               mymsmd=myMsmd)
Tamb = TLoad[antID][atmScan]['amb']


#- SPWs

# Integration limits for resolved AC SPWs.
startChan=7
stopChan=120

#- Atmosphere

# Calculate limiting TbSky for infinite PWV: It is a constant close to
# the tropospheric temperature for the ALMA array (MeteoCentral).
TbsInf=myAtm.getTebbSky(spwid=0,nc=0,
                        wh2o=qa.quantity(1000000.0,'mm'))['value'][0]

# Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
w0=getPWVforAnt(vis,tgtScan,antID)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',w0,' @airMass = ',airMass)
print('.......')

#- SciSPW to AtmSPW mapping.

# How science SPWs map to the SPW 0 - 3 in the ATM tool.
sciAtmMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}


#--- Read-out the sky brightness temperatures for all SPWs in the spwList.

# Store the frequency-averaged sky temperatures in a map
# spwId -> np.array(nTimeSamples). Do not use 2D array as we index by the SPW
# id, which is sparse in the set of integers.
Txx={}
Tyy={}


for sciSPW in spwList:

  # Read ACD calibrated sky temperatures for the target defined by targetScan.
  time,TxxS,TyyS=tgt_Tb(vis,atmScan,tgtScan,antID,sciSPW,myMS,myMsmd)
```

```
  # We have different handling for frequency-resolved and integrated SPWs.
  nChans=TxxS.shape[0]

  if (nChans > 1):

    # Resolved AC SPWs.
    # Trim the spectral edges by posing the integration limits and average
    # over the remaining channels.

    TxxM=np.mean(TxxS[fChans],axis=0)
    TyyM=np.mean(TyyS[fChans],axis=0)

  else:

    # Integrated ACs or SQLDs.
    # The single channel already represents a mean over the band-pass.

    TxxM=TxxS[0]
    TyyM=TyyS[0]

    # The SQLD spws have 12x higher cadence than the ACs. Rebin the SQLD
    # data in time to have a homogeneous series. Do the same for time axis.
    if sciSPW in [0,1,2,3,13,14,15,16]:
      TxxM=rebin(TxxM,12)
      TyyM=rebin(TyyM,12)
      time=rebin(time,12)

  # Append the frequency-averged time series for each SPW to the Tsky arrays.
  Txx[sciSPW]=TxxM
  Tyy[sciSPW]=TyyM

#-->endfor(sciSpws in list)

# Count the SPW in the list.
numSpw=len(spwList)
# Count the time samples in data.
nSamples=len(time)

# The data are read-out, close the tools.
myMsmd.done()
myMS.close()


#-- Allocate arrays to be used for matrix A and the RHS-vector C calculations
#   from the ATM model. They have just nAtmSPWs components.

# Find the array dims.
nAtmSPWs=myAtm.getNumSpectralWindows()
nChans=myAtm.getNumChan(0)

# Allocate the model sky (source + atm) temperature.
Tsky=np.zeros(nAtmSPWs)

# Allocate sensitivity factors for dW and dB (Jacobian elements).
SW=np.zeros(nAtmSPWs)
SB=np.zeros(nAtmSPWs)

# Allocate and find frequencies of channels in all our ATM SPWs.
```

```
spw_freqs=np.zeros((nAtmSPWs,nChans))

for atmSpw in range(nAtmSPWs):

  fMin=myAtm.getMinFreq(atmSpw)['value'][0]*1.0e9 # GHz to Hz
  df=myAtm.getChanSep(atmSpw)['value'][0]*1.0e6   # MHz to Hz

  spw_freqs[atmSpw]=fMin+df*np.arange(nChans)

#--- Improve a guess of the mean solar brigthness temperature.

TsolEst=np.zeros(numSpw)

spwCount=0
for sciSPW in spwList:

  # Map the sci SPW to its corresponding ATM counterpart
  atmSPW=sciAtmMap[sciSPW]

  # Sky brightness in a given SPW for the current PWV estimate.
  TbsEst=myAtm.getAverageTebbSky(atmSPW,qa.quantity(w0,'mm'))['value'][0]

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Mean (spw-averaged) transmissivity
  transM=np.mean(np.exp(-opacity))
  # Frequency-dependent part of solar spectrum weighted by transmissivity.
  SFtrans=np.mean(np.exp(-opacity)*np.log(spw_freqs[atmSPW]/f_LO))

  # <Tsky> estimation from measurements, averaged over time and pols.
  TskyEst=0.5*(Txx[sciSPW].mean()+Tyy[sciSPW].mean())

  TsolEst[spwCount]=(TskyEst-(1.0-eta_F+eta_l)*TbsEst
                     -eta_l*eta_F*b_2*SFtrans
                     -(1.0-eta_l)*Tamb)/(eta_l*eta_F*transM)

  spwCount+=1

#--> endfor(sci SPWs)

# Make a new guess of Tsun as an average.
Tsun0=TsolEst.mean()
print("Tsun = ",Tsun0,"+/-",TsolEst.std())


#-- Average solar spectrum (dimension-less, normalised to 1.0 at f=f_LO)

# It has the same dimensions as spw_freqs, i.e. nAtmSPWs x nChans
SF=1.0+b_2/Tsun0*np.log(spw_freqs/f_LO)


#--- Solve the set of non-linear equations for full PWV and full T_B.
#    Use the least square minimizer, iterate until we are not close to zero.
```

```
#    Do minimisation for each time instant separatelly.

#-- Iteration static setup.

# Half dw for the estimation of d/dw0 derivatives
dws=0.05 #mm
# Half dTb for the estimation of d/dT_B derivatives
dTs=5.0 # K

# Stopper based on gradient of the least-square functional.
grad_max=0.01

#-- Time loop over all time samples.

# Initial estimation pf pww and T_B
pwv=w0
T_B=Tsun0


# Resulting time series.
pwv_TS=np.zeros(nSamples)
Tb_TS=np.zeros(nSamples)


for ts in range(nSamples):

  #-- Newton-Raphson iterations for least-square residuum minimiser.

  # Initial value of the minimizer gradient - any bigger then the stopper.
  # We calculate the gradient inside the loop.
  grad_curr=0.5
  # Iteration counter.
  itCount=0

  while((grad_curr > grad_max) and (itCount < 20)):

    # 'Operation points': We evaluate the minimizer gradient in the current
    # pwv and T_B, and the surrounding points.
    op_points=[[pwv,T_B],
               [pwv+dws,T_B],
               [pwv-dws,T_B],
               [pwv,T_B+dTs],
               [pwv,T_B-dTs]]

    GGP=np.zeros((5,2))


    opIndex=0

    for opp in op_points:

      JJ=np.array([])
      BB=np.array([])

      wC=opp[0]
      T_Bc=opp[1]

      # Set current estimation of PWV to the ATM model.
```

```
myAtm.setUserWH2O(qa.quantity(wC,'mm'))

#- Loop over 4 ATM spws - these correspond to the channel-resolved ACs.
#  Some factors depend only on the ATM mmodel, not the sciSPW data.
for atmSPW in range(nAtmSPWs):

  #- Model value of the atm brightness with current pwv.
  TbsW0=myAtm.getTebbSkySpec(atmSPW,qa.quantity(wC,'mm'))[1]['value']

  #- Slope of the atm brightness with current pwv.
  TbsW0H=myAtm.getTebbSkySpec(atmSPW,
                              qa.quantity(wC+dws,'mm'))[1]['value']
  TbsW0L=myAtm.getTebbSkySpec(atmSPW,
                              qa.quantity(wC-dws,'mm'))[1]['value']

  dTdW=(TbsW0H-TbsW0L)/(2.0*dws)

  #- Opacities & transmisivity (spectral)

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Wet opacity per 1mm PWV scaled by airmass.
  tau_1=airMass*tau_wet/wC
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)
  # Transmissivity (spectral).
  trans=np.exp(-opacity)

  #- Tsky model. Weighted averages over the SPW.
  Tsky[atmSPW]=(eta_l*eta_F*T_Bc*np.mean(trans*SF[atmSPW])
                +(1.0-eta_F+eta_l)*np.mean(TbsW0)
                +(1.0-eta_l)*Tamb)

  #- Sensitivities = Jacobian elements. Weighted averages over the SPW.

  # Sensitivity factor for PWV.
  SW[atmSPW]=(-eta_l*eta_F*T_Bc*np.mean(trans*tau_1*SF[atmSPW])
              +(1.0-eta_F+eta_l)*np.mean(dTdW))

  # Sensitivity factor for the (apparent) solar-brigthness.
  SB[atmSPW]=eta_l*eta_F*np.mean(trans*SF[atmSPW])

#-->endfor(ATM spws)

#-- Loop over all sciSPWS in the list.
for sciSPW in spwList:

  # Map the sci SPW to its corresponding ATM counterpart
  atmSPW=sciAtmMap[sciSPW]

  # Set the sensitivity factors for the current SPW as two rows per pol.
  # Here, the rows are repeating in the Jacobian matrix as the sens.
  # factors calculated from the model do not depend on XX/YY.
  JJ=np.append(JJ,[SW[atmSPW],SB[atmSPW]])
  JJ=np.append(JJ,[SW[atmSPW],SB[atmSPW]])
```

```
      # RHS: Measured Txx and Tyy, subtracted from the model Tsky
      BB=np.append(BB,Tsky[atmSPW]-Txx[sciSPW][ts])
      BB=np.append(BB,Tsky[atmSPW]-Tyy[sciSPW][ts])
    #--> endfor(sci spws in list)

    # Re-shape the Jacobian matrix to be nVars x (nPols*numSpw) array
    JJ=np.reshape(JJ,(2*numSpw,2))

    # Re-shape the RHS to be a column vector of (nPols*numSpw) dim.
    BB=np.reshape(BB,(2*numSpw,1))

    # Transposed Jacobian
    JJT=np.transpose(JJ)

    # Minimizer: Gradient of the merit function. We look for its null point.
    GGP[opIndex]=np.transpose(np.matmul(JJT,BB))

    opIndex+=1
  #-->endfor(opIndex)

  #-- Minimizer gradient and its Jacobian

  GG=GGP[0]

  GG_wPlus=GGP[1]
  GG_wMinus=GGP[2]

  GG_TbPlus=GGP[3]
  GG_TbMinus=GGP[4]

  JG11=(GG_wPlus[0]-GG_wMinus[0])/(2.0*dws)
  JG21=(GG_wPlus[1]-GG_wMinus[1])/(2.0*dws)

  JG12=(GG_TbPlus[0]-GG_TbMinus[0])/(2.0*dTs)
  JG22=(GG_TbPlus[1]-GG_TbMinus[1])/(2.0*dTs)

  # Jacobian inversion.

  iDet=1.0/(JG11*JG22-JG21*JG12)

  iG11=iDet*JG22
  iG12=-iDet*JG21
  iG21=-iDet*JG12
  iG22=iDet*JG11

  # N-R shift
  dWnr=-(iG11*GG[0]+iG12*GG[1])
  dTnr=-(iG21*GG[0]+iG22*GG[1])

  # N-R update of variables.
  pwv+=dWnr
  T_B+=dTnr

  # Gradient current magnitude
  grad_curr=np.sqrt(GG[0]*GG[0]+GG[1]*GG[1])

  itCount+=1
  print("After iteration no. ",itCount," ||grad|| = ",grad_curr)
```

```
    #-->endwhile(grad_curr > grad_max)

    print("Finished for time sample no. ",ts, ", PWV = ",pwv,", T_Sun = ",T_B)

    pwv_TS[ts]=pwv
    Tb_TS[ts]=T_B

  #--> endfor(time samples)

  return time,pwv_TS,Tb_TS

#-------------------------------------------------------------------------------


def dWdBdG2_weights(vis,atmScan,tgtScan,myAtm,antID,spwList,T_tgt=-1.0e6,
                    flaggedData=[],myMS=None,myMsmd=None,acdLoads=None):

  """
  Essentially the same as dWdBdG2(), but now we compose & solve the grand-system
  for entire range of calWeights (from 100 to 1000) and mWeights (100 - 10000)
  run in a loop.

  Why we do not make the dWdBdG2() parametrized by *cal/mWeight* and run just
  the wrapping function, which calls dWdBdG2() in a loop with varying weights?
  The reason is technical: The most time-consuming operation is reading
  and processing the sci-SPW data. In this function we have a loop just over
  the grand-system solution, which represents less then 10% of CPU time spent
  in dWdBdG2().


  The goal is to find, how much the mean parameters <PWV> and <TsunXX/YY> are
  sensitive to the calWeight/mWeight selection. Note, that the fluctuations dW,
  dTBxx, and dTbyy, are quite robust - they do not significantly change even
  between various dPWV proxies like dWdB(), dWdB_Tbs(), dWdBdG[2]() or
  dWdB_Tbs_NL() and are, to a large extent, independent of their respective
  means.

  Moreover, we are checking more in detail (with much finer cal/mWeight sampling
  over the selected interval), how the particular errors (Xi-squares) for
  the natural (pwv, T_bXX/YY),instrumental (G, P_inst), and array-mean (T_tgt)
  parts of the solution vector depend on cal/mWeight. Lastly, the number of
  iterations needed for the convergence is being watched.


  Implementation: We vary always just one weight (cal- or mWeight): the second
  is set static to its optimised value found by variation. Selection is
  performed in the code - comment/uncomment selection of *theWeight* below.
  Weight range and sampling is controlled also manually by wLow and wHigh range
  and the number of samples code in wItSize.


  Results:  It has been found that the optimum value for calWeight lies between
  100 and 300. At low values, the Xi^2 for instrumental variables is high and
  is improving with increasing calWeight. Nevertheless, around 200 the
  improvement of Xi^2 inst. slows down, while, at the same moment, the 2nd part
  of the error vector, the Xi^2 for natural variables, start rapidly increasing.
```

```
The <PWV> and <TsunXX/YY> are pretty flat in this region of calWeights, so
it is safe to select catWeght=200. This is actually the value eventually used
in production version of dWdBdG2().
"""

#--- Static parameters

#- Tools, global data

# Flags used to check, whether we should close the tool later.
ownMS=False
ownMsmd=False

# Open own tools if 'global' not provided
if (not myMS):
  # Open the ms tool
  myMS=mstool()
  myMS.open(vis)
  ownMS=True

if (not myMsmd):
  # Open the msmd tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True

# Create our dictionary with ACD load temperatures (ambient, hot) if needed.
if (not acdLoads):
  acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)

#- Antenna

antNames=myMsmd.antennanames()
myAntName=antNames[antID]
antType=myAntName[:2]

# Map the antenna efficiencies by antenna type. As per investigation from
# merged T_sun temperatures form 2023 and 2026 data (project_22_25_Tb.ods).
eta_l_map={'CM':0.941,'DA':0.947,'DV':0.956}
eta_F_map={'CM':0.891,'DA':0.897,'DV':0.906}

# Rear-spillover + ohmic antenna efficiency - see project_22_25_Tb.ods
eta_l=eta_l_map[antType]
# Forward-spillover antenna efficiency - see project_22_25_Tb.ods
eta_F=eta_F_map[antType]

# We have two exception that seem to have systematically lower
# antenna efficiencies: DA64 and DV08 - see project_22_25_Tb.ods
if  myAntName=='DA64':
  eta_l=0.929
  eta_F=0.880
if  myAntName=='DV08':
  eta_l=0.943
  eta_F=0.894

# Extract ambient & hot temperatures for the current antenna and scan.
```

```
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']

# Optimized weight of the ACD calibration equations in the system.
# Search mark in dWdBdG2_weights(): CW_1W
calWeight=100000.0

#- Target

# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.300  # kiloKelvins!!!

# Solar radius.
Rsun=696000 # km


# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.

"""
# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for July 30 2026 Band 5 observation.
'''
Target:       RA=08 37 18.01 DEC=+18 31 23.0
Sun center:   RA=08 38 15.79 DEC=+18 28 47.2
-----------------------------------------------
diff (arcsec, incl. cos(delta)): 822   155.8
'''
x_h=822.0 # arcsec
y_h=155.8  # arcsec

# Earth-Sun distance for July 30 2026 Band 5 observation.
Dsun=1.01508611721886*149597870.700 # km

# *** End of observation-specific part.
"""


# Apparent angular solar radius in arcsec.
r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

# mu=cos(theta). Used for limb-brightenning correction below.
mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

# b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
# in Alissandrakis et al. (2020). Scaled to kiloKelvins!
```

```
b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

# Brightness temoerature corrected to limb brightenning - see
# Alissandrakis et al. (2020).
Tsun00=TsunCenter+0.5*b_2*np.log(mu)

# Weight of the array-mean equation <T_B(t)>=T_tgt_arrayMean in the system.
# If T_tgt is not explicitly set (the defaul value is -1MK), do not apply
# this equation at all. This can be achieved by setting its weight to 0.
if (T_tgt < -10000.0):
  # Do not apply T_tgt bond. If mWeight is zero, T_tgt is irrelevant.
  T_tgt=0.0
  mWeight=0.0
else:
  # We use kiloKelvins in calculations, re-scale T_tgt to kK, too.
  T_tgt*=0.001
  # Set the experimentally found optimized mWeight.
  # Search mark in dWdBdG2_weights(): MW_1W
  mWeight=5000.0


#- SPWs

# LO frequency. We take it as a reference frequency for the solar spectrum.
f_LO=1.98e11 # in Hz

# Integration limits for resolved AC SPWs.
startChan=8
stopChan=120

fChans=range(startChan,stopChan)

#- Atmosphere

# Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,' @airMass = ',airMass)
print('Estimated T_B = ',1000.0*Tsun00)
print('.......')

#- SciSPW to AtmSPW mapping.

# How science SPWs map to the SPW 0 - 3 in the ATM tool.
sciAtmMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}

#--- Read-out the sky+src powers for all SPWs in the spwList.

# Store the frequency-averaged sky+src powers in a map.
# spwId -> np.array(nTimeSamples). Do not use 2D array as we index by the SPW
```

```
# id, which is sparse in the set of integers.
Pxx={}
Pyy={}

#- Similar maps for the gains and instrumental powers P_inst=G*T_inst.
#  Here, all gains and P_inst are, in fact, the G* and P_ints* values for the
#  higher-attenuation solar regime. In order to append the ACD calibration
#  equations, whose measurements are performed in the norma-attenuation regime
#  in the preceding ATM scan, we determine also the attenuation factors
#  AF=G*/G and related power offsets AO.

# Frequency-averaged gains. The scale for the spectral-resolved norm. gains.
Gxx={}
Gyy={}

# Attenuation factors G/G*=(P_sky-P0)/(P_off-Po). Attributed to every-spw
# Gxx & Gyy, but actually calculated just in SQLD spws, the rest is mapped.
AFxx={}
AFyy={}

afMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}

# Attenuation-based power offsets AO=-Po*(P_sky-P_off)/(P_off-Po).
# They are non-zero only in SQLD spws.
AOxx={}
AOyy={}

# Spectral gain normalized to 1.0. Acual ACD-based values in spectral AC spws,
# a trivial 1.0 ndarray for integrated-AC and SQLD spws.
gXX={}
gYY={}

# Frequency-averaged instrumental powers.
PIxx={}
PIyy={}

#- Finally, the raw ACD-calibration powers.

# Ambient-load powers.
PxxAmb={}
PyyAmb={}

# Hot-load powers.
PxxHot={}
PyyHot={}

#-- Read the data and ACD calibration tables in a loop over all sciSPWs.

for sciSPW in spwList:

  #- Raw data & ACD tables

  # Read raw science powers for the target defined by targetScan.
  time,P_src=tgt_pow(vis,tgtScan,antID,sciSPW,myMS=myMS,myMsmd=myMsmd,
                     subScans='on')

  PxxS=P_src[0]
  PyyS=P_src[1]
```

```
    # Read ACD calibrations - gains,Tinst, and calibration powers.
    calDict=acd_calTabs(vis,atmScan,antID,sciSPW,myMS=myMS,myMsmd=myMsmd,
                        acdLoads=acdLoads)

    Pcals=calDict['calPows']

    PxxAmbS=Pcals['P_amb'][0]
    PyyAmbS=Pcals['P_amb'][1]

    PxxHotS=Pcals['P_hot'][0]
    PyyHotS=Pcals['P_hot'][1]


    # We have different handling for frequency-resolved and integrated SPWs.
    # Find the number of channels in the observed SPW.
    nChansO=PxxS.shape[0]

    if (nChansO > 1):

      #- Resolved AC SPWs.
      #  Trim the spectral edges by posing the integration limits and average
      #  over the remaining fChans channels.

      # Science-target powers.
      PxxM=np.mean(PxxS[fChans],axis=0)
      PyyM=np.mean(PyyS[fChans],axis=0)


      # Calibration (ACD) powers.
      PxxAmbM=np.mean(PxxAmbS[fChans],axis=0)
      PyyAmbM=np.mean(PyyAmbS[fChans],axis=0)

      PxxHotM=np.mean(PxxHotS[fChans],axis=0)
      PyyHotM=np.mean(PyyHotS[fChans],axis=0)

      #- Instrumental factors.
      #  Store the normalized (to 1.0) spectral gains (arrays, constant) and
      #  their scales (numbers, to be re-fined by the solver) separately.

      # Raw G* gains in a full SPW. We may flip the array (see below) so better
      # to do so before its limiting to fChans - wee need not to think whether
      # the fChans central range is placed symmetricaly within its SPW.

      # Gain has shape 2 x nPols x nChans. The 1st index 0 means we pick the
      # *leading* P_off subscan - see the note in tgt_Tb().
      GxxS=1.0/(calDict['iG*'][0,0])
      GyyS=1.0/(calDict['iG*'][0,1])

      # T_inst in full SPW. The same holds for flipped/reversed SPWs as for G*.
      TinstXX=calDict['T_inst'][0]
      TinstYY=calDict['T_inst'][1]

      # Check, whether we need flip them - this is the case of the LSB spws,
      # which have decreasing/reversed frequency channels.
      spwFreqs=calDict['freqs']

      if (spwFreqs[0] > spwFreqs[1]):
```

```
      # Reversed SPW, flip the gain and T_inst arrays.

      GxxS=np.flip(GxxS)
      GyyS=np.flip(GyyS)

      TinstXX=np.flip(TinstXX)
      TinstYY=np.flip(TinstYY)

    # Now it is safe to select the central fChans band-pass.
    GxxS=GxxS[fChans]
    GyyS=GyyS[fChans]

    TinstXX=TinstXX[fChans]
    TinstYY=TinstYY[fChans]

    # Calculate the unity-normalized gains by dividing out the mean value.
    GxxM=GxxS.mean()
    GyyM=GyyS.mean()

    gXX[sciSPW]=GxxS/GxxM
    gYY[sciSPW]=GyyS/GyyM

    # Instrumental power averaged over our fChans range.
    PIxxM=np.mean(GxxS*TinstXX)
    PIyyM=np.mean(GyyS*TinstYY)

    # Keep the time-axis of the last resolved SPW as a reference timebase,
    # to which we intrepolate the SQLD signals.
    timeBase=time

    # SQLD spws have an instrumental offset ralated to Pzero. It is calculated
    # below, in the single-channel SPWs branch. Here we set it to zero
    # because Pzero for AC-resolved spws is zero.
    AOxx[sciSPW]=0.0
    AOyy[sciSPW]=0.0


  else:
    #- Integrated ACs or SQLDs.
    #  The single channel already represents a mean over the spw band-pass.

    # Single-channel science data.
    PxxM=PxxS[0]
    PyyM=PyyS[0]


    # Channel-integrated SPWs may have higher cadence. For SQLDs it is true
    # always, but it may be the case of integrated ACs, too. Let us
    # put all data to the same, slowest, timebase (as per resoved AC spws).

    # Compare the cadence (elementary time-step, a.k.a. integration).
    if ((timeBase[1]-timeBase[0]) > (time[1]-time[0])):

      # Indeed, we have higher cadence. Re-bin in time.
      # Use the timeBase defined as a time-axis for resolved SPW.

      print('---> SPW',sciSPW,
            'has higher cadence: Rebinning to time-line of AC-resolved SPW.')
```

```
  PxxM=rebin2(PxxM,time,timeBase)
  PyyM=rebin2(PyyM,time,timeBase)
  time=timeBase

#--> endif(higher cadence)

# The attenuation factors AF=(P_sky-Po)/(P_off-Po) and offsets AO should
# be, moreover, extracted from the SQLDss (ACs use mapping afMap
# defined above).
if sciSPW in [0,1,2,3,13,14,15,16]:

  # Extract the caliibration powers relevant for AF calculation
  Patm=Pcals['P_sky']
  # Poff is the only calPow that has a structure 2 x nPols x nChans,
  # similarly like gain. The others are just nPols x nChans 2D arrays.
  # Here, the first dimension '2' is for leading/trailing Poff subscans
  # and we pick the leading one (see the gain extraction above).
  Poff=Pcals['P_off'][0]
  Pzero=Pcals['P_zero']

  # Calculate AFs.
  # The 1st index in powers is pol, the 2nd means (the single) channel.
  AFxx[sciSPW]=(Patm[0,0]-Pzero[0,0])/(Poff[0,0]-Pzero[0,0])
  AFyy[sciSPW]=(Patm[1,0]-Pzero[1,0])/(Poff[1,0]-Pzero[1,0])

  # Calculates AOs.
  AOxx[sciSPW]=Pzero[0,0]*(AFxx[sciSPW]-1.0)
  AOyy[sciSPW]=Pzero[1,0]*(AFyy[sciSPW]-1.0)
else:
  # For non-SQLD spws the offsets are zero because Pzero is zero.
  AOxx[sciSPW]=0.0
  AOyy[sciSPW]=0.0

#--> endif(SQLD spws)

# Calibration data.
PxxAmbM=PxxAmbS[0]
PyyAmbM=PyyAmbS[0]

PxxHotM=PxxHotS[0]
PyyHotM=PyyHotS[0]


#- Instrumental factors.

# Gains (solar-attenuation): Take the leading-subscan based result.
GxxM=1.0/(calDict['iG*'][0,0,0])
GyyM=1.0/(calDict['iG*'][0,1,0])


# Create a fake spectral normalized gain for integrated SPWs.
# We use it in products with the ATM arrays (e.g. transmissivity), which
# are always spectral.
gXX[sciSPW]=np.ones(len(fChans))
gYY[sciSPW]=np.ones(len(fChans))

# Instrumental power: The channel 0 is the average already
```

```
    PIxxM=GxxM*calDict['T_inst'][0,0]
    PIyyM=GyyM*calDict['T_inst'][1,0]
  #--> endif(resolved SPW)


  # Append the frequency-averged time series for each SPW to the maps.
  Pxx[sciSPW]=PxxM
  Pyy[sciSPW]=PyyM

  # The calibration powers are already time-averaged.
  PxxAmb[sciSPW]=PxxAmbM
  PyyAmb[sciSPW]=PyyAmbM

  PxxHot[sciSPW]=PxxHotM
  PyyHot[sciSPW]=PyyHotM

  # Set the ACD-based gains (scales) and instrumental contributions.
  Gxx[sciSPW]=GxxM
  Gyy[sciSPW]=GyyM

  PIxx[sciSPW]=PIxxM
  PIyy[sciSPW]=PIyyM


  print('SPW ',sciSPW,': <Tsky+src_XX> = ',
        (Pxx[sciSPW].mean()-PIxx[sciSPW])/Gxx[sciSPW],'+/-',
        Pxx[sciSPW].std()/Gxx[sciSPW])
  print('          <Tsky+src_YY> = ',
        (Pyy[sciSPW].mean()-PIyy[sciSPW])/Gyy[sciSPW],'+/-',
        Pyy[sciSPW].std()/Gyy[sciSPW])

#-->endfor(all sciSPWs in the list)

# Save the original gains and instrumental powers for comparison with
# the updated values after the solution.
# Note: We have to use copy.deepcopy(), the simple assignment operator
# copies just the reference!

GxxOrig=deepcopy(Gxx)
GyyOrig=deepcopy(Gyy)

PIxxOrig=deepcopy(PIxx)
PIyyOrig=deepcopy(PIyy)

#-- Dimensions of the system.

# Count the SPW in the list.
numSpw=len(spwList)
# Count the time samples in data.
nSamples=len(time)

# How many net equations do we have per block of 1 time-sample.
# Be carefull: Do not flag the data that are *not* present in the spwList!
eqBlockSize=2*numSpw-len(flaggedData)


# The data has been read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()
```

```
if (ownMS): myMS.close()

#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).

# Pre-calculated frequently repeating factor: Inverse number of time samples.
nsInv=1.0/nSamples

# Number of the model spectral windows.
nAtmSPWs=myAtm.getNumSpectralWindows()
# Number of all channels in the model SPWs. They have all the same structure.
nChans=myAtm.getNumChan(0)
# Number of central band-pass (with edges trimmed) channels.
cChans=len(fChans)

# Frequencies of channels in our model SPWs (central band-pass only).
spw_freqs=np.zeros((nAtmSPWs,cChans))

for atmSpw in range(nAtmSPWs):

  fMin=myAtm.getMinFreq(atmSpw)['value'][0]*1.0e9 # GHz to Hz
  df=myAtm.getChanSep(atmSpw)['value'][0]*1.0e6   # MHz to Hz

  spw_freqs[atmSpw]=(fMin+df*np.arange(nChans))[fChans]

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,cChans))
dTdW=np.zeros((nAtmSPWs,cChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,cChans))
tau_1=np.zeros((nAtmSPWs,cChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/Tsun00*np.log(spw_freqs/f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
# We used those as weights in calculation of epsilon - the normalised
# inter-iteration change of operation-point at the very end of each iteration.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin
dG_max=0.001       # Relative scale dG/G
dTinst_max=0.001   # dPinst/G in kiloKelvins


#+++++++++++++ Loop over reasonable cal/mWeights ++++++++++

# Define the calWeight/mWeight iteration ranges.
```

```
wItSize=101

# In semi-log scale. Use plt.semilogx(xAxis,yAxis) for plot.
#exponents=0.05*(np.arange(wItSize)-(wItSize-1)/2)
exponents=0.05*(np.arange(wItSize))
weights=pow(10,exponents)


'''
wLow=100.0
wHigh=500.0
weights=wLow+(wHigh-wLow)/(wItSize-1)*np.arange(wItSize)
'''

#- We study influence of selected (iterated) calWeigth/mWeight to:

# 1. Time-averaged PWV
iPWV=np.zeros(wItSize)

# 2. Time-averaged (solar) brightness temperatures in XX and YY pols.
iTSxx=np.zeros(wItSize)
iTSyy=np.zeros(wItSize)


# 3. Number of iterations required for convergence.
iNIT=np.zeros(wItSize)

# 4. Errors separated by 'sub-systems' of equations: science (s), ACD cal (a),
#    and mean-array T_tgt (m).
iXis=np.zeros(wItSize)
iXia=np.zeros(wItSize)
iXim=np.zeros(wItSize)

# 5. Relative updates of ACD gains and P_inst. List of dictionaries as it is
#    easier than np.arrays of dictionaries.
iGG=[]
iPP=[]

#-- The main weight-iteration loop.
for wi in range(wItSize):


  theWeight=float(weights[wi])

  # Which weight do we iterate: calWeight or mWeight? The 2nd remains static.
  # Select by comment/uncomment.

  #calWeight=theWeight
  mWeight=theWeight

  # Reset the starting values for iterations.
  w0=pwv0
  Tsun0xx=Tsun00
  Tsun0yy=Tsun00

  # Reset gains/P_inst back to their pure-ACD values.
  Gxx=deepcopy(GxxOrig)
  Gyy=deepcopy(GyyOrig)
```

```
PIxx=deepcopy(PIxxOrig)
PIyy=deepcopy(PIyyOrig)

# Iteration counter.
itCount=0
# Normalized inter-iteration change of operation point - its initial value.
# Anything bigger than 1.0
epsilon=10.0


#-- Solution iterations: Minimize the mean value of variations.
while((epsilon>1.0) and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample

  # N-block matrix: Jacobian part for [dW,dT_Bxx,dT_Byy] 'natural' variables
  J_wb=np.zeros((eqBlockSize,3)) # 3 vars repeating in all spws
  # G-block matrix: Jacobian part for [dG,dP_inst] 'instrumental' variables.
  J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each SPW:Pol.
  # A-block matrix: Jacobian part for [dG,dP_inst] for ACD calibration power
  J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, ambient
  J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, hot.
  # M-block matrix: Jacobian part for [dW,dT_Bxx,dT_Byy] in array-mean eqs.
  # This matrix is constant and can be defined already here.
  J_M=np.array([[0.0, nsInv, 0.0],
                [0.0, 0.0, nsInv]])


  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))
  # RHS-vector 'appendix' - amb. & hot ACD calibration powers (time-avgd).
  CCA=np.zeros((eqBlockSize,1))
  CCH=np.zeros((eqBlockSize,1))
  # RHS-vector 'appendix 2' - array-mean values of target T_B
  CCM=np.zeros((2,1))

  # Rezidual mean sky temperature - after subtracted brightness of the
  # atmosphere (+CMB) and (attenuated) source. We keep it for all sci SPWs
  # and both pols and print it out at the end of every iteration cycle
  # as a diagnostics of convergence and solution quality.

  resTxx={}
  resTyy={}

  # Mapping between spw IDs and the row number in the system of equations.
  spw2rowXX={}
  spw2rowYY={}

  #- Re-calculate ATM model with updated <PWV> and <Tsun>.

  # Set current estimation of PWV to the ATM model.
  myAtm.setUserWH2O(qa.quantity(w0,'mm'))

  #- Loop over 4 ATM spws - these correspond to the channel-resolved ACs.
  #  Calculate factors dependent only on the mmodel, not the data.
```

```
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Limit the spectrum to the central fChans reagion as we have trimmed
  # the edge channels in ACD-based spectral normalized gain gXX/gYY.
  # Scale to kiloKelvins.
  TbsW0[atmSPW]=0.001*Tebb[fChans]

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value'][fChans]
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value'][fChans]

  # In kiloKelvins/mm
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Transmissivity (spectral). Limitted to fChans.
  trans[atmSPW]=np.exp(-opacity[fChans])
  # Wet opacity per 1mm PWV scaled by airmass. Limitted to fChans.
  tau_1[atmSPW]=(airMass/w0)*tau_wet[fChans]

#--> endfor(modelSPWs)

#- Loop over all SCI spws - combine model-only dependent factors with
#  sciSPW-dependent data and instrumental factors and build the system.

# Count the actual number of equations - some rows may be flagged.
eqCount=0

for sciSPW in spwList:

  # Map the sci SPW to its corresponding ATM counterpart
  atmSPW=sciAtmMap[sciSPW]

  #- Model (_M) factors.

  # Tebb and its slope.
  TbsW0_M=TbsW0[atmSPW]
  dTdW_M=dTdW[atmSPW]

  # Transmissivity and unit-PWV opacity.
  trans_M=trans[atmSPW]
  tau_1M=tau_1[atmSPW]

  # Solar-flux model spectrum (normalized).
  SF_M=SF[atmSPW]
```

```
    #- Separate procedure by polarizations for combination with real data.

    #- Pol XX

    # Full identificator of data: sciSPW:pol
    sciDataID=str(sciSPW)+':XX'

    if (sciDataID not in flaggedData):

      #- Model + instrumental factors combined.

      # We do weighting of equations by inverse gain in order to compensate
      # different powers (and gains) in AC and SQLD SPWs. Consequently,
      # the equations are expressed in equivalent temperatures, which are
      # (approximatelly) the same for all SPWs.
      # Weight factor
      WF=1.0/Gxx[sciSPW]

      # Sensitivity factor for PWV.
      SWxx=(-eta_l*eta_F*Tsun0xx*np.mean(gXX[sciSPW]*trans_M*tau_1M*SF_M)
            +(1.0-eta_F+eta_F*eta_l)
            *np.mean(gXX[sciSPW]*dTdW_M))*Gxx[sciSPW]

      # Sensitivity factor for the (apparent) solar-brigthness temperature.
      SBxx=Gxx[sciSPW]*eta_l*eta_F*np.mean(gXX[sciSPW]*trans_M*SF_M)

      # Model T_sky as viewed by the receiver with spectral gain gXX.
      TskyXX=(eta_l*eta_F*Tsun0xx*np.mean(trans_M*SF_M*gXX[sciSPW])
              +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M*gXX[sciSPW])
              +eta_F*(1.0-eta_l)*Tamb)

      #- Build the matrix: Natural (dW & dB sensitivities) block.

      # Set the (weighted) sensitivity factors for the current SPW:pol as
      # a row in the N-block/tile for the linearized-system Jacobian matrix.
      J_wb[eqCount]=WF*np.array([SWxx,SBxx,0.0])

      #- Build the matrix: Instrumental (G & P_inst sensitivities) block.

      # We have to prepend a proper offset for the G and P_inst vars as
      # they are independendt for each SPW and polarization.
      bOffset=2*eqCount

      # G-block scaled by the inverse gain.
      J_gp[eqCount,bOffset]=WF
      J_gp[eqCount,bOffset+1]=TskyXX*WF

      #- Build the matrix: ACD powers (amb & hot) 2-block.

      # Ambient-power sub-block. Scale between the G* / P_inst* used in
      # J_wb and J_gp, and G / P_inst used in the ACD cal equations by the
      # attenuation factor AF=G*/G=(P_sky-Po)/(P_off-Po).

      # Attenuation factor (af>1) is calculated only in SQLD and mapped.
      af=AFxx[afMap[sciSPW]]

      # A-ambient sub-block Jacobian elements weighted by inverse gain.
      J_cdA[eqCount,bOffset]=af*WF
```

```
    J_cdA[eqCount,bOffset+1]=af*Tamb*WF

    # A-hot-power sub-block weighetd by inverse gain.
    J_cdH[eqCount,bOffset]=af*WF
    J_cdH[eqCount,bOffset+1]=af*Thot*WF

    #- Build the RHS vector: Target data

    # RHS: Pxx(t) with their stationary (in the working point) instr.
    # and ATM-based parts subtracted.
    dPxx=Pxx[sciSPW]-PIxx[sciSPW]-Gxx[sciSPW]*TskyXX

    # Set it as the RHS component, weighted by inverse gain.
    CC[eqCount]=dPxx*WF

    #- Ammend the RHS vector: Calibration data.

    # Attenuation-based power offset. We have it defined for all SPWs,
    # however, non-zero it is only in SQLDs.
    aoff=AOxx[sciSPW]

    # Ambient-load power with its working-point model part subtracted.
    CCA[eqCount]=WF*(PxxAmb[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Tamb
                     +aoff)
    # Hot-load power with its working-point model part subtracted.
    CCH[eqCount]=WF*(PxxHot[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Thot
                     +aoff)
    #- Diagnostics

    # Residua of sky tempereture for mean PWV and Tsun (operation point).
    resTxx[sciSPW]=dPxx.mean()/Gxx[sciSPW]

    # Mapping between SPW:XX and equation number.
    spw2rowXX[sciSPW]=eqCount

    eqCount+=1

else: print('Ignoring flagged ',sciDataID,' data.')
#--> endif(not flaggged)

#- Pol YY

# Full identificator of data: sciSPW:pol
sciDataID=str(sciSPW)+':YY'

if (sciDataID not in flaggedData):

    #- Model + instrumental factors combined.

    # Weight factor - see explanation at XX pol above.
    WF=1.0/Gyy[sciSPW]

    # Sensitivity factor for PWV (weighted by 1/G).
    SWyy=(-eta_l*eta_F*Tsun0yy*np.mean(gYY[sciSPW]*trans_M*tau_1M*SF_M)
          +(1.0-eta_F+eta_F*eta_l)
          *np.mean(gYY[sciSPW]*dTdW_M))*Gyy[sciSPW]

    # Sensitivity factor for the (apparent) solar T_B (weighted by 1/G).
```

```
SByy=Gyy[sciSPW]*eta_l*eta_F*np.mean(gYY[sciSPW]*trans_M*SF_M)

# Model T_sky as viewed by the receiver with spectral gain gYY.
TskyYY=(eta_l*eta_F*Tsun0yy*np.mean(trans_M*SF_M*gYY[sciSPW])
        +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M*gYY[sciSPW])
        +eta_F*(1.0-eta_l)*Tamb)

#- Build the matrix: Natural (dW & dB sensitivities) block.

# Set the (weighted) sensitivity factors for the current SPW:pol as
# a row in the block/tile for the linearized-system Jacobian matrix.
J_wb[eqCount]=WF*np.array([SWyy,0.0,SByy])

#- Build the matrix: Instrumental (G & P_inst sensitivities) block.

# We have to prepend a proper offset for the G and P_inst vars as
# they are independendt for each SPW and polarization.
bOffset=2*eqCount

J_gp[eqCount,bOffset]=WF
J_gp[eqCount,bOffset+1]=TskyYY*WF

#- Build the matrix: ACD powers 2-block.

# Attenuation factor G*/G. Mapped from the corresponding SQLD.
af=AFyy[afMap[sciSPW]]

# Ambient-power sub-block.
J_cdA[eqCount,bOffset]=af*WF
J_cdA[eqCount,bOffset+1]=af*Tamb*WF

# Hot-power sub-block.
J_cdH[eqCount,bOffset]=af*WF
J_cdH[eqCount,bOffset+1]=af*Thot*WF

#- Build the RHS vector: Target data.

# RHS: Pyy(t) with their stationary (in the working point) inst.
# and ATM-based parts subtracted.
dPyy=Pyy[sciSPW]-PIyy[sciSPW]-Gyy[sciSPW]*TskyYY

# Set it as the RHS component.
CC[eqCount]=dPyy*WF

#- Ammend the RHS vector: Calibration data.

# Attenuation-based power offset.
aoff=AOyy[sciSPW]

# Ambient-load power with its working-point part subtracted.
CCA[eqCount]=WF*(PyyAmb[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Tamb
                 +aoff)
# Hot-load power with its working-point part subtracted.
CCH[eqCount]=WF*(PyyHot[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Thot
                 +aoff)

#- Diagnostics
```

```
      # Residua of sky tempereture for mean PWV and Tsun (operation point).
      resTyy[sciSPW]=dPyy.mean()/Gyy[sciSPW]

      # Mapping between SPW:YY and equation number.
      spw2rowYY[sciSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',sciDataID,' data.')
    #--> endif(not flaggged)

  #--> endfor(all sci SPWs)

  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged SPW:pol channels.")
    return False


  #-- Grand system assembly and inversion by parts. We use the approach from
  #   extMatInv() for inversion of extended-system matrix.

  #- Ammend the RHS vector: Array-mean target brightness temperatures.
  #  Here we work already with temperatures, thus no 1/G scalling applies.

  # Target T_B with the model at work-point subtracted.
  CCM[0]=T_tgt-Tsun0xx
  CCM[1]=T_tgt-Tsun0yy


  #- Calculate the small-block matrices and their inversions needed for
  #  grand-system inversion.

  # 1. Natural (N-block) matrices.
  J_wb_T=np.transpose(J_wb)       # N^T
  J_K=np.matmul(J_wb_T,J_wb)      # (N^T x N)
  J_K_inv=np.linalg.inv(J_K)      # (N^T x N)^{-1}

  # 2. Gain/P_inst (G-block) matrices.
  J_gp_T=np.transpose(J_gp)       # G^T

  # 3. ACD rows (A-block) matrices.
  # Calculate transpositions per sub-block (ambient/hot).
  # Attention!!! NumPy's transpose() makes just another view of a matrix,
  # we have to make a copy, otherwise subsequent scaling would scale the
  # original J_cdA and J_cdH matrices, too!

  J_cdA_T=np.transpose(J_cdA).copy()     # A^T
  J_cdH_T=np.transpose(J_cdH).copy()     # H^T

  # Increase importance of the ACD rows by calibration weight.
  # Only the transposed A-block matrices should be scaled up!
  J_cdA_T*=calWeight
  J_cdH_T*=calWeight

  # Compose the full block and its transposition.
```

```
J_cd=np.vstack([J_cdA,J_cdH]).copy()
J_cd_T=calWeight*np.transpose(J_cd)     # calWeight * (A | H )^T

# 4. Array-mean (M-block) related matrices. Incorporate mWeight into M^T,
# as with what we have done with calWeight for the A^T and H^T matrices.
M_T=np.array([[0.0, 0.0],
              [nsInv*mWeight, 0.0],
              [0.0,nsInv* mWeight]])    # mWeight*M^T


# Pre-calculated nSamples*mWeight*(M^T x M) matrix.
# Like with A^T x A, the M^T x M product is scaled-up by mWeight just once
# (no mWeight squared!). Moreover, we scale it up here yet by nSamples,
# in order to avoid trivial product nsInv*nSamples=1 in the following calc
nsMTM=np.array([[0.0, 0.0, 0.0],
                [0.0, nsInv*mWeight, 0.0],
                [0.0, 0.0, nsInv*mWeight]])


# Frequently repeating blocks. NM_inv plays a role of modified J_K_inv.
NM=J_K+nsMTM                # J_K+nSamples*mWeight*M^T x M
NM_inv=np.linalg.inv(NM)    # NM^(-1)


# Off-diagonal blocks of the inverted natural sub-matrix (Ainv).
J_M_inv=nsInv*(NM_inv-J_K_inv)

#- Base-matrix extensions.

# Right and lower bands: Their (nEqs x 3) and (3 x nEqs) repeating blocks.
Rb=np.matmul(J_wb_T,J_gp)       # (N^T x G)
Lb=np.transpose(Rb)             # (G^T x N)

# Lower-right corner
LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

#- Frequently repeating matrix products. Cf. with extMatInv() above.

# Ainv x Rb and Lb x Ainv
AinvRb=np.matmul(NM_inv,Rb)
LbAinv=np.transpose(AinvRb)

#- Schur complement and its inversion.
SC=LRc-nSamples*np.matmul(Lb,AinvRb)
SC_inv=np.linalg.inv(SC)

#- Blocks of inverted grand-system matrix.

# Start with the lower-left block as it repeats in upper-left block, too.
Minv21=-np.matmul(SC_inv,LbAinv)

# Upper-left block: Off-diagonal terms. The diagonal blocks differ just
# by added J_K_inv.
Minv11_OD=J_M_inv-np.matmul(AinvRb,Minv21)

# Upper-right block.
Minv12=np.transpose(Minv21)
```

```
    # Lower-right block.
    Minv22=SC_inv


    #-- Apply the inverted matrix components to the RHS vector.

    # Sum of the RHS along the time axis, keeping the SPW:POL dimension.
    # Preserve the number of matrix rows, just sum their contents, i.e. make
    # the single-column matrix.
    CCsum=np.reshape(CC.sum(axis=1),(-1,1))

    #- Form the RHS vector for the normal equations (K-form minimizer),
    #  i.e, multiply the raw data by JJ_T

    # Start with the natural-variables dW and dB part.
    C_K_wb=np.matmul(J_wb_T,CC)+np.matmul(M_T,CCM)  # 3 x nSamples matrix

    # Make the same also for its time-sum.
    C_K_wb_sum=np.matmul(J_wb_T,CCsum)+nSamples*np.matmul(M_T,CCM)  # 3 x 1

    # Follow with the instrumental part.
    C_K_gp=(np.matmul(J_gp_T,CCsum)       # 2*eqCount x 1
            +np.matmul(J_cdA_T,CCA)
            +np.matmul(J_cdH_T,CCH))

    #- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

    # Constant part of the natural-variables vector (3 x 1)
    nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)

    # Time-varying part of the natural-variables vector. It comes from the
    # addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
    # NB: Actually, with M-blocks included, it contains a small constant part,
    # too, equal to J_K_inv x M_T x CCM, but wee keep it here (hidden in
    # C_K_wb).
    nvVar=np.matmul(J_K_inv,C_K_wb)   # 3 x nSamples

    # Compose the complete natural variables vector.
    varNat=nvVar+nvConst

    # Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
    varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)

    #- Reshape the results - extract our elementary variables.

    # Seaparate the natural-vars components.
    dW = varNat[0]
    dBxx=varNat[1]
    dByy=varNat[2]

    # Reshape instrumental parameters to two rows (transpose): 1st is dP_inst,
    # the 2nd dG, one component per SPW:POL.
    dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

    # Extract the shifts in the P_inst and G
    dPinst=dPG[0]
    dGinst=dPG[1]
```

```
    #- Analytics: Calculate the errors for the over-determined set of eqs.

    # err=f(X)-b =approx. f(X_0)+J(X_0) x (X -X_0) -b = J x dX - [b-f(X_0)]
    # In our notation, dX is the [varNat,varInst]^T vector and the
    # difference [b-f(X_0)] is [ CC, (CCA | CCH), CCM ]^T
    # Now we express the err vector parts for sci, acd, and array-mean eqs.

    # err=f(x)-b: Sci-data part.
    errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
    # Xi square for the sci-data part. Sum over all SPW:Pol channels & times.
    # Normalize it to the number of equations.
    xiSci=np.sum(errVec*errVec)/(nSamples*eqBlockSize)

    # Xi Squares per SPW:Pol channel. Sum/average only over times.
    xiSciPerChan=np.sum(errVec*errVec,axis=1)/nSamples
    # Its mean and RMS over the SPW:POL channels.
    xscMean=xiSciPerChan.mean()
    xscRMS=xiSciPerChan.std()
    # Find outliers ( > 3 sigma). The badData array contains indices of
    # those equations, where the agreement between model and observation is
    # exceptionaly weak.
    badData=np.flatnonzero(abs(xiSciPerChan-xscMean)>3.0*xscRMS)

    # Show all per-channel errors as temperatures (K) and mark (auto-flag
    # later) the SPW:Pol channels with bad data, if we have some.
    # We do not have an inverse map to spw2rowXX/YY, so we have to traverse.

    # XX pol
    for spw,eqRow in spw2rowXX.items():
      print(str(spw)+':XX: Error:',np.sqrt(xiSciPerChan[eqRow])*1000.0,'K')
      if eqRow in badData: print('\t\t!!! Bad data in',str(spw)+':XX',
                                 (xiSciPerChan[eqRow]-xscMean)/xscRMS,
                                 'sigma from the mean.')
    # YY pol
    for spw,eqRow in spw2rowYY.items():
      print(str(spw)+':YY: Error:',np.sqrt(xiSciPerChan[eqRow])*1000.0,'K')
      if eqRow in badData: print('\t\t!!! Bad data in',str(spw)+':YY',
                                   (xiSciPerChan[eqRow]-xscMean)/xscRMS,
                                   'sigma from the mean.')

    # err=f(x)-b: ACD-data part.
    # Actually, this should be scaled by calWeight factor. On the other hand,
    # we are interested in Xi2 normalised per one equation, so we would
    # down-scale in the next calculation by the same factor. In order to avoid
    # useless product calWeight*1/calWeight, we omit it already now.
    errVecACDamb=np.matmul(J_cdA,varInst)-CCA
    errVecACDhot=np.matmul(J_cdH,varInst)-CCH
    # Xi square for the acd-data part.
    # Normalize it to the number of ACD equations. Scale by just single
    # block-size - we have ignored calWeight factor already above.
    xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)
    xiACD/=eqBlockSize

    # err=f(x)-b: Array-mean T_B part.
    # Here, the term J x dX = M x sum (CC_i) = [sum(dTxx_i), sum(dTyy_i) ]^T =
    # nS*[dBxx.mean(),dByy.mean()]^T is zero because dB.mrean() converged to 0
    errVecMeanTb=-CCM
    # Normalized to one equation - we have mWeight*2 equations, but the same
```

```
    # holds for mWeight factor as for the calWeight above.
    xiTbM=np.sum(errVecMeanTb*errVecMeanTb)/2

    # Grand-total Xi-square normalized to the number of equations for
    # the natural/science, ACD, and mean-array sub-blocks.
    xiSquare=xiSci+xiACD+xiTbM

    #- All variations should tend to zero means.

    dW_mean=dW.mean()
    dBxx_mean=dBxx.mean()
    dByy_mean=dByy.mean()

    dB_mean=np.sqrt(dBxx_mean**2+dByy_mean**2)

    # Update stationary part of PWV and solar T_B  by means of their vars.
    w0+=dW_mean
    Tsun0xx+=dBxx_mean
    Tsun0yy+=dByy_mean

    # Update also the P_inst and G by the found differences.
    # Find the maximum inter-iteration change for convergence diagnostics.
    dT_inst_sup=-1.0
    dG_rel_sup=-1.0

    # Start with XX pol.
    for sciSPW,eqCount in spw2rowXX.items():
      # Update the P_inst and G.
      PIxx[sciSPW]+=dPinst[eqCount]
      Gxx[sciSPW]+=dGinst[eqCount]

      # Change of inst. power expressed in kiloKelvins.
      dT_inst=abs(dPinst[eqCount]/Gxx[sciSPW])
      if (dT_inst > dT_inst_sup): dT_inst_sup=dT_inst

      # Change of gain in the relative scale.
      dG_rel=abs(dGinst[eqCount]/Gxx[sciSPW])
      if (dG_rel > dG_rel_sup): dG_rel_sup=dG_rel
    #--> endfor(SPWs with valid XX pol)

    # YY pol.
    for sciSPW,eqCount in spw2rowYY.items():
      # Update the P_inst and G.
      PIyy[sciSPW]+=dPinst[eqCount]
      Gyy[sciSPW]+=dGinst[eqCount]

      # Change of inst. power expressed in kiloKelvins.
      dT_inst=abs(dPinst[eqCount]/Gyy[sciSPW])
      if (dT_inst > dT_inst_sup): dT_inst_sup=dT_inst

      # Change of gain in the relative scale.
      dG_rel=abs(dGinst[eqCount]/Gyy[sciSPW])
      if (dG_rel > dG_rel_sup): dG_rel_sup=dG_rel
    #--> endfor(SPWs with valid YY pol)

    #-- Convergence diagnostics / Iteration control.

    # Calculate the normalized inter-iteration change of operation point.
```

```
    # The factor 0.5 comes from sqrt(1/4) as we have 4 'orthogonal' variables.
    epsilon=0.5*np.sqrt((dW_mean/dwMean_max)**2+(dB_mean/dbMean_max)**2+
                        (dG_rel_sup/dG_max)**2+(dT_inst_sup/dTinst_max)**2)

    # Increment the iteration counter
    itCount+=1

    #- Diagnostic print-outs per-iteration.

    # Updated scan-averaged PWV and Tsun after completed iteration.
    print('vvvvvvv\n After iteration no. ',itCount)
    print('\nMean PWV for the scan,',tgtScan,' w0 = ',w0,
          ", mean Tsun (XX/YY) = ",Tsun0xx,'/',Tsun0yy)

    # Mean values of dW and dB fluctuations should tend to zero.
    print("dW mean: ",dW_mean)
    print("dB mean: ",dB_mean)

    # Sky-brightness residua should tend to zero, too.
    # Convert the spwId maps to np.arrays first. We have to go via 'list'.
    # Scale back form kiloKelvins to Kelvins.
    resTxxArr=1000.0*np.array(list(resTxx.values()))
    resTyyArr=1000.0*np.array(list(resTyy.values()))

    print("\nres Txx: ", resTxxArr,"\nmean +/- RMS: ",
          resTxxArr.mean(),"+/-",resTxxArr.std())
    print("\nres Tyy: ", resTyyArr,"\nmean +/- RMS: ",
          resTyyArr.mean(),"+/-",resTyyArr.std())

    print("\nInstrumental characteristics shifts during the iteration:")
    print("--- Pol XX--- ")
    for sciSPW,eqCount in spw2rowXX.items():
      print("SPW =",sciSPW,"\tdT_inst =",dPinst[eqCount]/Gxx[sciSPW]*1000.0,
            "K","\tdG/G =",dGinst[eqCount]/Gxx[sciSPW]*100,"%")

    print("--- Pol YY--- ")
    for sciSPW,eqCount in spw2rowYY.items():
      print("SPW =",sciSPW,"\tdT_inst =",dPinst[eqCount]/Gyy[sciSPW]*1000.0,
            "K","\tdG/G =",dGinst[eqCount]/Gyy[sciSPW]*100,"%")

    print('^^^^^^^\n\n')

#--> endwhile(non-zero-means) / end of the main solution-iteration loop.


#-- Show, how much the instrumental variables G and P_inst have changed.

# The instrumental characteristics are maps, cannot process them as arrays.

GxxChange={}
GyyChange={}

PIxxChange={}
PIyyChange={}

for spw in spw2rowXX.keys():

  GxxChange[spw]=(Gxx[spw]-GxxOrig[spw])/GxxOrig[spw]
```

```
      PIxxChange[spw]=(PIxx[spw]-PIxxOrig[spw])/PIxxOrig[spw]

    for spw in spw2rowYY.keys():

      GyyChange[spw]=(Gyy[spw]-GyyOrig[spw])/GyyOrig[spw]
      PIyyChange[spw]=(PIyy[spw]-PIyyOrig[spw])/PIyyOrig[spw]


    # Store the watched quantities in the array per-weight elements.

    iPWV[wi]=w0

    # Scale the result back to Kelvins.
    iTSxx[wi]=1000.0*Tsun0xx
    iTSyy[wi]=1000.0*Tsun0yy

    iNIT[wi]=itCount
    iXis[wi]=xiSci
    iXia[wi]=xiACD
    iXim[wi]=xiTbM

    iGG.append([GxxChange,GyyChange])
    iPP.append([PIxxChange,PIyyChange])

  #--> endfor(range-of-weights)

  #-- Return the arrays of interest in depencence on calWeight.

  return {'NS':nSamples,'CW':weights,"PWV":iPWV,"Tx":iTSxx,"Ty":iTSyy,
          "XiS":iXis,"XiA":iXia,"XiM":iXim,"NIT":iNIT,"dG":iGG,"dP":iPP}



#-----------------------------------------------------------------------------

def dWdBdG_calWeights(vis,atmScan,tgtScan,myAtm,antID,spwList,flaggedData=[],
            myMS=None,myMsmd=None,acdLoads=None):

  '''
 --- OBSOLETTED---


  This is an older version of dWdBdG2_weights(), which supersedes this function.
  '''

  #--- Static parameters

  #- Dirs

  # Output/diagnostics plots.
  plotDir='./sciPWVs_GP'


  #- Tools, global data

  # Flags used to check, whether we should close the tool later.
  ownMS=False
  ownMsmd=False
```

```
# Open own tools if 'global' not provided
if (not myMS):
  # Open the ms tool
  myMS=mstool()
  myMS.open(vis)
  ownMS=True

if (not myMsmd):
  # Open the ms tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True

# Create our dictionary with ACD load temperatures (ambient, hot) if needed.
if (not acdLoads):
  acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)

#- Target

# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.3  # kiloKelvins!!!

# Solar radius.
Rsun=696000 # km

# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.

# Apparent angular solar radius in arcsec.
r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

# mu=cos(theta). Used for limb-brightenning correction below.
mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

# b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
# in Alissandrakis et al. (2020). Scaled to kiloKelvins!
b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

# Brightness temoerature corrected to limb brightenning - see
# Alissandrakis et al. (2020).
Tsun00=TsunCenter+0.5*b_2*np.log(mu)


# Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
f_LO=1.98e11
```

```
#- SPWs

# Integration limits for resolved AC SPWs.
startChan=8
stopChan=120

fChans=range(startChan,stopChan)

#- Antenna

# Rear-spillover + ohmic antenna efficiency (see Tab. 3 in White et al. 2017).
eta_l=0.95
# Forward-spillover antenna efficiency (Tab. 3 in White et al. 2017).
eta_F=0.90

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']

# Weight of the ACD calibration equations in the system. Essentially the
# number, with which appended block of the ACD equations is repeated in order
# to balance the much larger sciData-equations block. Infinity means that
# the instrumental parameters G and P_inst are essentially fixed and given
# by results of the acd_calTabs(). Low weight, on the other hand, leads to
# an ill-coinditioned system. We set in in order of nTimeSmaples for science
# data so that the target and calibration scan are balanced.
calWeight=200.0

#- Atmosphere

 # Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,' @airMass = ',airMass)
print('Estimated T_B = ',Tsun00)
print('.......')

#- SciSPW to AtmSPW mapping.

# How science SPWs map to the SPW 0 - 3 in the ATM tool.
sciAtmMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}


#--- Read-out the sky+src powers for all SPWs in the spwList.

# Store the frequency-averaged sky+src powers in a map.
# spwId -> np.array(nTimeSamples). Do not use 2D array as we index by the SPW
# id, which is sparse in the set of integers.
Pxx={}
```

```
Pyy={}

#- Similar maps for the gains and instrumental powers P_inst=G*T_inst.
#  Here, all gains and P_inst are, in fact, the G* and P_ints* values for the
#  higher-attenuation solar regime. In order to append the ACD calibration
#  equations, whose measurements are performed in the norma-attenuation regime
#  in the preceding ATM scan, we determine also the attenuation factors
#  AF=G*/G and related power offsets AO.

# Frequency-averaged gains. The sacle for the spectral-resolved norm. gains.
Gxx={}
Gyy={}

# Attenuation factors G/G*=(P_sky-P0)/(P_off-Po). Attributed to every-spw
# Gxx & Gyy, but actually calculated just in SQLD spws, the rest is mapped.
AFxx={}
AFyy={}

afMap={0:0,1:1,2:2,3:3,5:0,6:0,7:1,8:1,9:2,10:2,11:3,12:3}

# Attenuation-based power offsets AO=-Po*(P_sky-P_off)/(P_off-Po).
# The same holds as for the AFs - we use afMap.
AOxx={}
AOyy={}

# Spectral gain normalized to 1.0. Acual ACD-based values in spectral AC spws,
# a trivial 1.0 ndarray for integrated-AC and SQLD spws.
gXX={}
gYY={}

# Frequency-averaged instrumental powers.
PIxx={}
PIyy={}

#- Finally, the raw ACD-calibration powers.

# Ambient-load powers.
PxxAmb={}
PyyAmb={}

# Hot-load powers.
PxxHot={}
PyyHot={}

#-- Read the data and ACD calibration tables in a loop over all sciSPWs.

for sciSPW in spwList:

  #- Raw data & ACD tables

  # Read raw science powers for the target defined by targetScan.
  time,P_src=tgt_pow(vis,tgtScan,antID,sciSPW,myMS=myMS,myMsmd=myMsmd,
                     subScans='on')

  PxxS=P_src[0]
  PyyS=P_src[1]

  # Read ACD calibrations - gains,Tinst, and calibration powers.
```

```
calDict=acd_calTabs(vis,atmScan,antID,sciSPW,myMS=myMS,myMsmd=myMsmd,
                    acdLoads=acdLoads)

Pcals=calDict['calPows']

PxxAmbS=Pcals['P_amb'][0]
PyyAmbS=Pcals['P_amb'][1]

PxxHotS=Pcals['P_hot'][0]
PyyHotS=Pcals['P_hot'][1]


# We have different handling for frequency-resolved and integrated SPWs.
nChans=PxxS.shape[0]

if (nChans > 1):

  #- Resolved AC SPWs.
  #  Trim the spectral edges by posing the integration limits and average
  #  over the remaining fChans channels.

  # Science-target powers.
  PxxM=np.mean(PxxS[fChans],axis=0)
  PyyM=np.mean(PyyS[fChans],axis=0)



  # Calibration (ACD) powers.
  PxxAmbM=np.mean(PxxAmbS[fChans],axis=0)
  PyyAmbM=np.mean(PyyAmbS[fChans],axis=0)

  PxxHotM=np.mean(PxxHotS[fChans],axis=0)
  PyyHotM=np.mean(PyyHotS[fChans],axis=0)

  #- Instrumental factors.
  #  Store the normalized (to 1.0) spectral gains (arrays, constant) and
  #  their scales (numbers, to be re-fined by the solver) separately.

  # Raw G* gains in a full SPW. We may flip the array (see below) so better
  # to do so before its limiting to fChans - wee need not to think whether
  # the fChans central range is placed symmetrically withn its SPW.
  # The 1st index 0 means we pick the leading P_off subscan - see the note
  # in acd_calTabs2().
  GxxS=1.0/(calDict['iG*'][0,0])
  GyyS=1.0/(calDict['iG*'][0,1])

  # T_inst in full SPW. The same holds as for G*.
  TinstXX=calDict['T_inst'][0]
  TinstYY=calDict['T_inst'][1]

  # Check, whether we need flip them - this is the case of the LSB spws,
  # which have decreasing frequency channels.
  spwFreqs=calDict['freqs']

  if (spwFreqs[0] > spwFreqs[1]):
    # Reversed SPW, flip the gain and T_inst arrays.

    GxxS=np.flip(GxxS)
```

```
    GyyS=np.flip(GyyS)

    TinstXX=np.flip(TinstXX)
    TinstYY=np.flip(TinstYY)

  # Now it is safe to select the central fChans band-pass.
  GxxS=GxxS[fChans]
  GyyS=GyyS[fChans]

  TinstXX=TinstXX[fChans]
  TinstYY=TinstYY[fChans]


  # Calculate the unity-normalized gains by dividing out the mean value.
  GxxM=GxxS.mean()
  GyyM=GyyS.mean()

  gXX[sciSPW]=GxxS/GxxM
  gYY[sciSPW]=GyyS/GyyM

  # Instrumental power averaged over our fChans range.
  PIxxM=np.mean(GxxS*TinstXX)
  PIyyM=np.mean(GyyS*TinstYY)

else:
  #- Integrated ACs or SQLDs.
  #  The single channel already represents a mean over the band-pass.

  # Science data.
  PxxM=PxxS[0]
  PyyM=PyyS[0]

  # The SQLD spws have 12x higher cadence than the ACs. Rebin the SQLD
  # data in time to have a homogeneous series. Do the same for time axis.
  # Moreover, extract the attenuation factor (P_sky-Po)/(P_off-Po) and
  # offsets AO from the SQLD spws (ACs use mapping afMap defined above).
  if sciSPW in [0,1,2,3,13,14,15,16]:
    # Re-bin in time.
    PxxM=rebin(PxxM,12)
    PyyM=rebin(PyyM,12)
    time=rebin(time,12)

    # Extract the caliibration powers relevant for AF calculation
    Patm=Pcals['P_sky']
    Poff=Pcals['P_off']
    Pzero=Pcals['P_zero']

    # Calculate AFs.
    # The 1st index in powers is pol, the 2nd means (the single) channel.
    AFxx[sciSPW]=(Patm[0,0]-Pzero[0,0])/(Poff[0,0]-Pzero[0,0])
    AFyy[sciSPW]=(Patm[1,0]-Pzero[1,0])/(Poff[1,0]-Pzero[1,0])

    # Calculates AOs.
    AOxx[sciSPW]=Pzero[0,0]*(AFxx[sciSPW]-1.0)
    AOyy[sciSPW]=Pzero[1,0]*(AFyy[sciSPW]-1.0)

  #--> endfor(SQLD spws)
```

```
    # Calibration data.
    PxxAmbM=PxxAmbS[0]
    PyyAmbM=PyyAmbS[0]

    PxxHotM=PxxHotS[0]
    PyyHotM=PyyHotS[0]


    #- Instrumental factors.
    GxxM=1.0/(calDict['iG*'][0,0,0])
    GyyM=1.0/(calDict['iG*'][0,1,0])


    # Create a fake spectral normalized gain for integrated SPWs.
    # We use it in products with the ATM arrays (e.g. transmissivity), which
    # are always spectral.
    gXX[sciSPW]=np.ones(len(fChans))
    gYY[sciSPW]=np.ones(len(fChans))

    # Instrumental power: The channel 0 is the average already
    PIxxM=GxxM*calDict['T_inst'][0,0]
    PIyyM=GyyM*calDict['T_inst'][1,0]
  #--> endif(resolved SPW)


  # Append the frequency-averged time series for each SPW to the maps.
  Pxx[sciSPW]=PxxM
  Pyy[sciSPW]=PyyM

  # The calibration powers are already time-averaged.
  PxxAmb[sciSPW]=PxxAmbM
  PyyAmb[sciSPW]=PyyAmbM

  PxxHot[sciSPW]=PxxHotM
  PyyHot[sciSPW]=PyyHotM


  # Set the ACD-based gains (scales) and instrumental contributions.
  Gxx[sciSPW]=GxxM
  Gyy[sciSPW]=GyyM

  PIxx[sciSPW]=PIxxM
  PIyy[sciSPW]=PIyyM


  print('SPW ',sciSPW,': <Tsky+src_XX> = ',
        (Pxx[sciSPW].mean()-PIxx[sciSPW])/Gxx[sciSPW],'+/-',
        Pxx[sciSPW].std()/Gxx[sciSPW])
  print('          <Tsky+src_YY> = ',
        (Pyy[sciSPW].mean()-PIyy[sciSPW])/Gyy[sciSPW],'+/-',
        Pyy[sciSPW].std()/Gyy[sciSPW])

#-->endfor(all sciSPWs in the list)

# Save the original gains and instrumental powers for comparison with
# the updated values after the solution.
# Note: We have to use copy.deepcopy(), the simple assignment operator
```

```
# copies just the reference!

GxxOrig=deepcopy(Gxx)
GyyOrig=deepcopy(Gyy)

PIxxOrig=deepcopy(PIxx)
PIyyOrig=deepcopy(PIyy)


#-- Dimensions of the system.

# Count the SPW in the list.
numSpw=len(spwList)
# Count the time samples in data.
nSamples=len(time)

# How many net equations do we have per block of 1 time-sample.
# Be carefull: Do not flag the data that are *not* present in the spwList!
eqBlockSize=2*numSpw-len(flaggedData)


# The data has been read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()
if (ownMS): myMS.close()


#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).

# Number of the model spectral windows.
nAtmSPWs=myAtm.getNumSpectralWindows()
# Number of all channels in the model SPWs.
nChans=myAtm.getNumChan(0)
# Number of central band-pass (with edges trimmed) channels.
cChans=len(fChans)

# Frequencies of channels in our model SPWs (central band-pass only).
spw_freqs=np.zeros((nAtmSPWs,cChans))

for atmSpw in range(nAtmSPWs):

  fMin=myAtm.getMinFreq(atmSpw)['value'][0]*1.0e9 # GHz to Hz
  df=myAtm.getChanSep(atmSpw)['value'][0]*1.0e6   # MHz to Hz

  spw_freqs[atmSpw]=(fMin+df*np.arange(nChans))[fChans]

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,cChans))
dTdW=np.zeros((nAtmSPWs,cChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,cChans))
tau_1=np.zeros((nAtmSPWs,cChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/Tsun00*np.log(spw_freqs/f_LO)
```

```
#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin


#+++++++++++++++ Loop over reasonable calWeights - experimental +++++++++++++++


PWV=np.zeros(900)

TSxx=np.zeros(900)
TSyy=np.zeros(900)

NIT=np.zeros(900)
xis=np.zeros(900)
xia=np.zeros(900)

for cwi in range(900):


  # Set calWeight.
  calWeight=100.0+cwi


  # Reset the starting values for iterations.
  w0=pwv0
  Tsun0xx=Tsun00
  Tsun0yy=Tsun00

  Gxx=deepcopy(GxxOrig)
  Gyy=deepcopy(GyyOrig)

  PIxx=deepcopy(PIxxOrig)
  PIyy=deepcopy(PIyyOrig)


  # Initial values for the dW and dL means - anyting bigger then the stoppers.
  dW_mean=0.5
  dB_mean=0.5

  # Iteration counter.
  itCount=0


  #-- Iterations: Minimize the mean value of variations.

  while(((abs(dW_mean) > dwMean_max) or (abs(dB_mean) > dbMean_max))
```

```
    and (itCount< 20)):

#-- Prepare/reset the linearized system of equations.

#- Repeating blocks/tiles for each time-sample

# N-block matrix: Jacobian part for [dW,dT_Bxx,dT_Byy] 'natural' variables
J_wb=np.zeros((eqBlockSize,3)) # 3 vars repeating in all spws
# G-block matrix: Jacobian part for [dG,dP_inst] 'instrumental' variables.
J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each SPW:Pol.
# A-block matrix: Jacobian part for [dG,dP_inst] for ACD  powers.
J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, ambient
J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per SPW:Pol, hot.


# RHS vector for all time samples stacked as a matrix.
CC=np.zeros((eqBlockSize,nSamples))
# RHS-vector 'appendix' - amb. & ACD calibration powers (time-averaged).
CCA=np.zeros((eqBlockSize,1))
CCH=np.zeros((eqBlockSize,1))

resTxx={}
resTyy={}

spw2rowXX={}
spw2rowYY={}

#- Re-calculate ATM model with updated <PWV> and <Tsun>.
# Set current estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

#- Loop over 4 ATM spws - these correspond to the channel-resolved ACs.
#  Calculate factors dependent only on the mmodel, not the data.
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Limit the spectrum to the central fChans reagion as we have trimmed
  # the edge channels in ACD-based spectral normalized gain gXX/gYY.
  # Scale to kiloKelvins!
  TbsW0[atmSPW]=0.001*Tebb[fChans]

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value'][fChans]
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value'][fChans]

  # In kiloKelvins/mm!
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
```

```
    opacity=airMass*(tau_wet+tau_dry)


    # Transmissivity (spectral). Limitted to fChans.
    trans[atmSPW]=np.exp(-opacity[fChans])
    # Wet opacity per 1mm PWV scaled by airmass. Limitted to fChans.
    tau_1[atmSPW]=(airMass/w0)*tau_wet[fChans]

  #--> endfor(modelSPWs)

  #- Loop over all SCI spws - combine model-only dependent factors with
  #  sciSPW-dependent data and instrumental factors and build the system.

  # Count the actual number of equations - some rows may be flagged.
  eqCount=0

  for sciSPW in spwList:

    # Map the sci SPW to its corresponding ATM counterpart
    atmSPW=sciAtmMap[sciSPW]

    #- Model factors.

    # Tebb and its slope.
    TbsW0_M=TbsW0[atmSPW]
    dTdW_M=dTdW[atmSPW]

    # Transmissivity and unit-PWV opacity.
    trans_M=trans[atmSPW]
    tau_1M=tau_1[atmSPW]

    # Solar-flux model spectrum (normalized).
    SF_M=SF[atmSPW]

    #- Separate procedure by polarizations for combination with real data.

    #- Pol XX

    # Full identificator of data: sciSPW:pol
    sciDataID=str(sciSPW)+':XX'

    if (sciDataID not in flaggedData):

      #- Model + instrumental factors combined.

      # Sensitivity factor for PWV.
      SWxx=(-eta_l*eta_F*Tsun0xx*np.mean(gXX[sciSPW]*trans_M*tau_1M*SF_M)
            +(1.0-eta_F+eta_l)*np.mean(gXX[sciSPW]*dTdW_M))*Gxx[sciSPW]


      # Sensitivity factor for the (apparent) solar-brigthness temperature.
      SBxx=Gxx[sciSPW]*eta_l*eta_F*np.mean(gXX[sciSPW]*trans_M*SF_M)


      # Model T_sky as viewed by the receiver with spectral gain gXX.
      TskyXX=(eta_l*eta_F*Tsun0xx*np.mean(trans_M*SF_M*gXX[sciSPW])
              +(1.0-eta_F+eta_l)*np.mean(TbsW0_M*gXX[sciSPW])
              +(1.0-eta_l)*Tamb)
```

```
#- Build the matrix: Natural (dW & dB sensitivities) block.

# Set the sensitivity factors for the current SPW and pol as a row
# in the block/tile for the linearized-system Jacobian matrix.
J_wb[eqCount]=np.array([SWxx,SBxx,0.0])

#- Build the matrix: Instrumental (G & P_inst sensitivities) block.

# We have to prepend a proper offset for the G and P_inst vars as
# they are independendt for each SPW and polarization.
bOffset=2*eqCount

J_gp[eqCount,bOffset]=1.0
J_gp[eqCount,bOffset+1]=TskyXX

#- Build the matrix: ACD powers (amb & hot) 2-block.


af=AFxx[afMap[sciSPW]]

J_cdA[eqCount,bOffset]=af
J_cdA[eqCount,bOffset+1]=af*Tamb

# Hot-power sub-block.
J_cdH[eqCount,bOffset]=af
J_cdH[eqCount,bOffset+1]=af*Thot


#- Build the RHS vector: Target data

# RHS: Pxx(t) with their stationary (in the working point) instrumen
# and ATM-based parts subtracted.
dPxx=Pxx[sciSPW]-PIxx[sciSPW]-Gxx[sciSPW]*TskyXX

# Set it as the RHS component.
CC[eqCount]=dPxx

#- Ammend the RHS vector: Calibration data.

# Attenuation-based power offset.
aoff=AOxx[afMap[sciSPW]]

# Ambient-load power with its working-point part subtracted.
CCA[eqCount]=PxxAmb[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Tamb+aoff
# Hot-load power with its working-point part subtracted.
CCH[eqCount]=PxxHot[sciSPW]-af*PIxx[sciSPW]-af*Gxx[sciSPW]*Thot+aoff

#- Diagnostics

# Residua of sky tempereture for mean PWV and Tsun (operation point).
resTxx[sciSPW]=dPxx.mean()/Gxx[sciSPW]

# Mapping between SPW:XX and equation number.
spw2rowXX[sciSPW]=eqCount

eqCount+=1
```

```
else: print('Ignoring flagged ',sciDataID,' data.')
#--> endif(not flaggged)


#- Pol YY

# Full identificator of data: sciSPW:pol
sciDataID=str(sciSPW)+':YY'

if (sciDataID not in flaggedData):

  #- Model + instrumental factors combined.

  # Sensitivity factor for PWV.
  SWyy=(-eta_l*eta_F*Tsun0yy*np.mean(gYY[sciSPW]*trans_M*tau_1M*SF_M)
        +(1.0-eta_F+eta_l)*np.mean(gYY[sciSPW]*dTdW_M))*Gyy[sciSPW]


  # Sensitivity factor for the (apparent) solar-brigthness temperature.
  SByy=Gyy[sciSPW]*eta_l*eta_F*np.mean(gYY[sciSPW]*trans_M*SF_M)


  # Model T_sky as viewed by the receiver with spectral gain gYY.
  TskyYY=(eta_l*eta_F*Tsun0yy*np.mean(trans_M*SF_M*gYY[sciSPW])
          +(1.0-eta_F+eta_l)*np.mean(TbsW0_M*gYY[sciSPW])
          +(1.0-eta_l)*Tamb)


  #- Build the matrix: Natural (dW & dB sensitivities) block.

  # Set the sensitivity factors for the current SPW and pol as a row
  # in the block/tile for the linearized-system Jacobian matrix.
  J_wb[eqCount]=np.array([SWyy,0.0,SByy])

  #- Build the matrix: Instrumental (G & P_inst sensitivities) block.

  # We have to prepend a proper offset for the G and P_inst vars as
  # they are independendt for each SPW and polarization.
  bOffset=2*eqCount

  J_gp[eqCount,bOffset]=1.0
  J_gp[eqCount,bOffset+1]=TskyYY

  #- Build the matrix: ACD powers 2-block.

  # Attenuation factor G*/G.
  af=AFyy[afMap[sciSPW]]

  # Ambient-power sub-block.
  J_cdA[eqCount,bOffset]=af
  J_cdA[eqCount,bOffset+1]=af*Tamb

  # Hot-power sub-block.
  J_cdH[eqCount,bOffset]=af
  J_cdH[eqCount,bOffset+1]=af*Thot

  #- Build the RHS vector: Target data.
```

```
      # RHS: Pyy(t) with their sonary (in the working point) instrumental
      # and ATM-based parts subtracted.
      dPyy=Pyy[sciSPW]-PIyy[sciSPW]-Gyy[sciSPW]*TskyYY

      # Set it as the RHS component.
      CC[eqCount]=dPyy

      #- Ammend the RHS vector: Calibration data.

      # Attenuation-based power offset.
      aoff=AOyy[afMap[sciSPW]]

      # Ambient-load power with its working-point part subtracted.
      CCA[eqCount]=PyyAmb[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Tamb+aoff
      # Hot-load power with its working-point part subtracted.
      CCH[eqCount]=PyyHot[sciSPW]-af*PIyy[sciSPW]-af*Gyy[sciSPW]*Thot+aoff


      #- Diagnostics

      # Residua of sky tempereture for mean PWV and Tsun (operation point).
      resTyy[sciSPW]=dPyy.mean()/Gyy[sciSPW]

      # Mapping between SPW:YY and equation number.
      spw2rowYY[sciSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',sciDataID,' data.')
    #--> endif(not flaggged)

  #--> endfor(all sci SPWs)


  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged SPWs/polarization chans.")
    return False


  #-- Grand system assembly and inversion by parts. We use the approach from
  #   extMatInv() for inversion of extended-system matrix. The procedure
  #   performed here looks mysterious (sometimes a factor nSamples emergeor
  #   the RHS vector summed over time-samples), but all directly implies fro
  #   application of the inverted matrix of extended grand-system (preserved
  #   in the commented section below) to the RHS vector. We use advantage of
  #   the symmetry of the original grand-system matrix (JJ_K^T=JJ_K).
  #   Advantage: This block-decomposed solution is O(ns^1) in the number of
  #   time-samples, while the 'brutal-force' direct method in the commented
  #   section below scales as O(ns^3) and takes ages...

  #- Calculate the small-block matrices and their inversions needed for
  #  grand-system inversion.

  # Natural (N-block) matrices.
```

```
J_wb_T=np.transpose(J_wb)
J_K=np.matmul(J_wb_T,J_wb)
J_K_inv=np.linalg.inv(J_K)

# Gain/P_inst (G-block) matrices.
J_gp_T=np.transpose(J_gp)

# ACD rows (A-block) matrices.
# Calculate transpositions per sub-block (ambient/hot).
J_cdA_T=np.transpose(J_cdA).copy()
J_cdH_T=np.transpose(J_cdH).copy()

# Increase importance of the ACD rows by calibration weight.
# Only the transposed A-block matrices should be scaled up!

J_cdA_T*=calWeight
J_cdH_T*=calWeight

# Compose the full block and its transposition. The transposition to be
# scaled up - see above. Or, alternativelly, it can be built as
# np.hstack([J_cdA_T,J_cdH_T]) from aly scaled-up transposed sub-blocks.
J_cd=np.vstack([J_cdA,J_cdH])
J_cd_T=calWeight*np.transpose(J_cd)

#- Base-matrix extensions.

# Right and lower bands: Their (nEqs x 3) and (3 x nEqs) repeating blocks.
Rb=np.matmul(J_wb_T,J_gp)
Lb=np.transpose(Rb)

# Lower-right corner
LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

#- Frequently repeating matrix products. Cf. with extMatInv() above.

# Ainv x Rb and Lb x Ainv
AinvRb=np.matmul(J_K_inv,Rb)
LbAinv=np.transpose(AinvRb)

#- Schur complement and its inversion.
SC=LRc-nSamples*np.matmul(Lb,AinvRb)
SC_inv=np.linalg.inv(SC)

#- Blocks of inverted grand-system matrix.

# Start with the lower-left block as it repeats in upper-left block, too.
Minv21=-np.matmul(SC_inv,LbAinv)

# Upper-left block: Off-diagonal terms. The diagonal blocks differ just
# by added J_K_inv.
Minv11_OD=-np.matmul(AinvRb,Minv21)

# Upper-right block.
Minv12=np.transpose(Minv21)

# Lower-right block.
Minv22=SC_inv
```

```
    #-- Apply the inverted matrix components to the RHS vector.

    # Sum of the RHS along the time axis, keeping the SPW:POL dimension.
    # Preserve the number of matrix rows, just sum their contents, i.e. make
    # the single-column matrix.
    CCsum=np.reshape(CC.sum(axis=1),(-1,1))

    #- Form the RHS vector for the normal equations (K-form minimizer),
    #  i.e, multiply the raw data by JJ_T

    # Start with the natural-variables dW and dB part.
    C_K_wb=np.matmul(J_wb_T,CC)       # 3 x nSamples matrix

    # Make the same also for its time-sum.
    C_K_wb_sum=np.matmul(J_wb_T,CCsum)    # 3 x 1

    # Follow with the instrumental part.
    C_K_gp=(np.matmul(J_gp_T,CCsum)       # 2*eqCount x 1
            +np.matmul(J_cdA_T,CCA)
            +np.matmul(J_cdH_T,CCH))

    #- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

    # Constant part of the natural-variables vector (3 x 1)
    nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)

    # Time-varying part of the natural-variables vector. It comes from the
    # addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
    nvVar=np.matmul(J_K_inv,C_K_wb)   # 3 x nSamples

    # Compose the complete natural variables vector.
    varNat=nvVar+nvConst

    # Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
    varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)

    #- Reshape the results - extract our elemenytary variables.

    # Seaparate the natural-vars components.
    dW = varNat[0]
    dBxx=varNat[1]
    dByy=varNat[2]

    # Reshape instrumental parameters to two rows (transpose): 1st is dP_inst,
    # the 2nd dG, one component per SPW:POL.
    dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

    # Extract the shifts in the P_inst and G
    dPinst=dPG[0]
    dGinst=dPG[1]

    #- Analytics: Calculate the errors for the over-determined set of eqs.

    # err=A.x-b: Sci-data part.
    errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
    # Xi square for the sci-data part.
    xiSci=np.sum(errVec*errVec)
```

```
# err=A.x-b: ACD-data part.
errVecACDamb=np.matmul(J_cdA,varInst)-CCA
errVecACDhot=np.matmul(J_cdH,varInst)-CCH
# Xi square for the acd-data part.
xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)

# Grand-total Xi-square normalized to the number of equations and relative
# Tsky+src/Thot ratio(= approx. 3500K/350K=10)
xiSquare=xiSci/nSamples+xiACD/calWeight*10.0

#- All variations should tend to zero means.

dW_mean=dW.mean()
dBxx_mean=dBxx.mean()
dByy_mean=dByy.mean()

dB_mean=np.sqrt(dBxx_mean**2+dByy_mean**2)

# Upde stationary part of PWV and solar T_B  by means of their variations.
w0+=dW_mean
Tsun0xx+=dBxx_mean
Tsun0yy+=dByy_mean


# Update also the P_inst and G by the found differences.
# Start with XX pol.
for sciSPW,eqCount in spw2rowXX.items():
  # Update the P_inst and G.
  PIxx[sciSPW]+=dPinst[eqCount]
  Gxx[sciSPW]+=dGinst[eqCount]
#--> endfor(SPWs with valid XX pol)

# YY pol.
for sciSPW,eqCount in spw2rowYY.items():
  # Update the P_inst and G.
  PIyy[sciSPW]+=dPinst[eqCount]
  Gyy[sciSPW]+=dGinst[eqCount]
#--> endfor(SPWs with valid YY pol)

#- Convergence diagnostics.

itCount+=1

# Updated scan-averaged PWV and Tsun after completed iteration.
print('vvvvvvv\n After iteration no. ',itCount)
print('\nMean PWV for the scan,',tgtScan,' w0 = ',w0,
      ", mean Tsun (XX/YY) = ",Tsun0xx,'/',Tsun0yy)

# Mean values of dW and dB fluctuations should tend to zero.
print("dW mean: ",dW_mean)
print("dB mean: ",dB_mean)

# Sky-brightness residua should tend to zero, too.
# Convert the spwId maps to np.arrays first. We have to go via 'list'.
# Scale back form kiloKelvins to Kelvins.
resTxxArr=1000.0*np.array(list(resTxx.values()))
resTyyArr=1000.0*np.array(list(resTyy.values()))
```

```
      print("\nres Txx: ", resTxxArr,"\nmean +/- RMS: ",
            resTxxArr.mean(),"+/-",resTxxArr.std())
      print("\nres Tyy: ", resTyyArr,"\nmean +/- RMS: ",
            resTyyArr.mean(),"+/-",resTyyArr.std())

      print("\nInstrumental characteristics shift [dP_inst, dG]:",dPinst,dGinst)

      print('^^^^^^^\n\n')

    #--> endwhile(non-zero-means) / end of main iteration loop.

    PWV[cwi]=w0

    TSxx[cwi]=Tsun0xx
    TSyy[cwi]=Tsun0yy

    NIT[cwi]=itCount
    xis[cwi]=xiSci
    xia[cwi]=xiACD

  #--> endfor(range-of-calWeights)

  # Range of calWeights
  CWS=100.0+np.arange(900.0)

  #-- Return the arrays of interest in depencence on calWeight.

  return {'CW':CWS,"PWV":PWV,"Tx":TSxx,"Ty":TSyy,"XiS":xis,"XiA":xia,"NIT":NIT}


#--------------------------------------------------------------------------------

def saveSol_dWdB(vis,atmScan,tgtScan,spwList):
  """
  Create ATM model framework and then run dWdB() for all antennas in the array.
  Save the complete results (dW, dB, matrix, and RHS) into a binary NPY file.

  See np.save() and np.load().
  """

  #-- Create the atmosphere model that covers the frequency domain of all SPWs.
  #   It is common for all antennas - we vary the PWV per antenna inside dWdB().
  atmModel=setAtmosSciSPWs(vis,tgtScan)

  #-- Get antenna list.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  antIDs=myMsmd.antennaids()
  antNames=myMsmd.antennanames()
  myMsmd.done()

  #-- Loop over all antennas.
  for myAnt in antIDs:

    # Report progress.

    print('\n============================================================')
```

```
    print('Calculating antenna ',int(myAnt),'<-->',antNames[myAnt])
    print('============================================================\n')

    dwdb=dWdB(vis,atmScan,tgtScan,atmModel,int(myAnt),spwList)


    #-  For using binary np.save() we need everything as an np.ndarray.

    # Convert the antenna ID to a single-element array.
    antIDa=np.array([int(myAnt)])

    # Extract the arrays from the map.
    tt=dwdb['time']
    dW=dwdb['dW']
    dB=dwdb['dB']
    AA=dwdb['A']
    CC=dwdb['C']

    # Open our NPY-binary file in the append mode and save the arrays for
    # the current antenna.
    with open('./dWdB_DATA.npy', 'ab') as f:
      np.save(f,antIDa)
      np.save(f,tt)
      np.save(f,dW)
      np.save(f,dB)
      np.save(f,AA)
      np.save(f,CC)
    # --> end(save block)

  # --> endfor(all antennas)

  return

#-------------------------------------------------------------------------------


def saveSol_dWdB_Tbs(vis,atmScan,tgtScan,spwList):
  """
  As saveSol_dWdB(), but now for data produced by either dWdB_Tbs() or
  dWdBdG().

  The main difference is in the ATM model: While dWdB() requires static
  opacities (i.e. arrays) pre-calculated by setAtmosSciSPWs(), the dWddB_Tbs()
  and dWdBdG() routines require the full ATM tool (attool) as their parameter,
  here set and coupled to actual meteo situation via setAtmTool4SPWs().
  """

  #-- Global tools & data

  myMS=mstool()
  myMS.open(vis)

  myMsmd=msmdtool()
  myMsmd.open(vis)

  acdLoads=aU.getLoadTemperatures(vis,doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)
```

```
#-- Atmosphere model that covers the frequency domain of all SPWs.

myAtm=attool()
setAtmTool4SPWs(myAtm,vis,tgtScan)


#-- Get antenna list.
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()

#-- Redirect output to a text file.

sys.stdout=open("./1_STDOUT.txt", "w")


#-- Loop over all antennas.
for myAnt in antIDs:

  # Report progress.

  print('\n============================================================')
  print('Calculating antenna ',int(myAnt),'<-->',antNames[myAnt])
  print('============================================================\n')

  # Select by (un)commenting whether dWdB_Tbs() or dWdBdG() should be used
  # for actual per-natenna calculations.

  '''
dwdb=dWdB_Tbs(vis,atmScan,tgtScan,myAtm,int(myAnt),spwList,flaggedData=[],
        myMS=myMS,myMsmd=myMsmd,acdLoads=acdLoads)

  '''
  dwdb=dWdBdG(vis,atmScan,tgtScan,myAtm,int(myAnt),spwList,flaggedData=[],
              myMS=myMS,myMsmd=myMsmd,acdLoads=acdLoads)



  # For utilisation of binary np.save() we need everything as an np.ndarray.
  # Convert the antenna ID to a single-element array.
  antIDa=np.array([int(myAnt)])

  # Extract the arrays from the map.
  tt=dwdb['time']
  dW=dwdb['pwv']
  dB=dwdb['T_B']
  AA=dwdb['J']
  CC=dwdb['C']


  # Open our NPY-binary file in the append mode and save the arrays for
  # the current antenna.
  with open('./PWV_TB_GP_DATA.npy', 'ab') as f:
    np.save(f,antIDa)
    np.save(f,tt)
    np.save(f,dW)
    np.save(f,dB)
    np.save(f,AA)
```

```
      np.save(f,CC)
    # --> end(save block)

    # Flush the redirected stdout stream.
    sys.stdout.flush()
  # --> endfor(all antennas)

  #-- Restore & clean-up.

  # Restore stdout back to console.
  sys.stdout = sys.__stdout__

  # Close the tools.
  myAtm.done()
  myMsmd.done()
  myMS.close()

  return True


#--------------------------------------------------------------------------------

def restoreSol_dWdB(vis,filePath,runID=100000):
  """
  Load the results of dWdB() & friends for entire ALMA array from a file and
  store it as a map with antenna as its key. Restoring works as a reverse
  routine to both saveSol_dWdB() and saveSol_dWdB_Tbs()

  The *filePath* points to the NPY file with data.

  The *runID* parameter (starting from 1) controls, which of the appended
  records should be read.

  We need *vis* in order to determine the antenna array (Todo: Think about
  re-implementation of the corresponding save_() routines so that thay store
  antenna array at the beginning of the file).

  It is a solar-specific routine.

  See also saveSol_dWdB() / saveSol_dWdB_Tbs().
  """

  #-- List all antennas in the array.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  antIDs=myMsmd.antennaids()
  myMsmd.done()

  # So far empty map [antID -> dWdB_record].
  dWdB_map={}

  # Init the record counter.
  rCount=0


  # Open the binary dWdB data.
  with open(filePath, 'rb') as f:
```

```
    #-- Loop over the stack of records.
    while(rCount<runID):

      #-- Loop over all antennas.
      for myAnt in antIDs:

        # We may be aiming at the last record (the default) - check the EOF.
        # The np.load() method needs to be checked via exceptions.
        try:

          antIDa=np.load(f)

        except (EOFError,ValueError, OSError): # EOF: new/old NumPy style
          print("EOF reached beyond record no.",rCount,".\n")
          # Prevent further reading attempts.
          rCount=runID
          # Break the antenna for-loop.
          break

        #--> end try/catch block.

        # The antID has been encoded as a single-element np array in order
        # to use save()/load() infrastructure.
        antID=antIDa[0]


        # Load the arrays.
        tt=np.load(f)
        dW=np.load(f)
        dB=np.load(f)
        AA=np.load(f)
        CC=np.load(f)

        # Organise them into the record (a map) as they were returned by dWdB().
        dWdB_record={}

        dWdB_record['time']=tt
        dWdB_record['dW']=dW
        dWdB_record['dB']=dB
        dWdB_record['A']=AA
        dWdB_record['C']=CC

        # Append the record to the collection.
        dWdB_map[antID]=dWdB_record

      #--> endfor(all antennas)

      # Increment the record counter.
      rCount+=1

    #-->endwhile(recordCountReached)
  #--> end with-block

  # Return the entire collection.
  return dWdB_map


#-------------------------------------------------------------------------------
```

```
#                5.1 Consistency checks of the dPWV proxies
#-----------------------------------------------------------------------------

def dB_sum():
  """
  Sum the brightness variations as found by dWdB() for all antennas.

  Idea: If there is an intrinsic T_B variation of the Sun, we arrive to this
  'coherent' contribution in the sum. If not, and the dB changes are just
  because of atmosperic lensing, the average of dB over all antennas should
  tend to zero.
  """

  # Restore all-antenna records, get the size of the collection.
  dwdbMap=restoreSol_dWdB()
  nAnts=len(dwdbMap)

  # Extract time-axis from the 1st antenna - it should be the same for all.
  tt=dwdbMap[0]['time']


  # Summed arrays of dB for XX and YY. Init by the 1st antenna.
  dBxx=dwdbMap[0]['dB'][0]
  dByy=dwdbMap[0]['dB'][1]

  for ant in range(1,nAnts):

    dB=dwdbMap[ant]['dB']

    dBxx+=dB[0]
    dByy+=dB[1]

  #--> endfor(antennas)

  # Do the average - scale the sum by the count of antennas.
  dBxx/=nAnts
  dByy/=nAnts

  plt.plot(tt,dBxx,'b-')
  plt.plot(tt,dByy,'r-')

  return

#-----------------------------------------------------------------------------

def TbStats(resCG,resVG):
  """
  The mean brightness temperature of the Sun, as derived from dWdB_Tbs()
  and its generalization dWdBdG(), should be essentially the same for all
  antennas - we expect (like in dB_sum() above) that the 'scintillation'
  burst/drops because of turbulence focusing/de-focusing effects are in average
  zero over the scan.

  Here we make staticstics over all antennas for two sets of the results:
  One obtained by dWdB_Tbs() with constant gains determined by ACD calibration,
  and the other, resulting from dWdBdG(), where we allow for the gain correction
  via minimizing the grand-system (science + calibration measurements together).
  The parameters *resCG* and *resVG* are the maps/dicts obtained as a result of
```

```
  restoreSol_dWdB() for the constant-gain and variable-gain solutions,
  respectivelly.
  """

  # Number fo records (num. of antennas)
  nAnts=len(resCG.keys())

  # Allocate the arrays.
  Tbxx_c=np.zeros(nAnts)
  Tbyy_c=np.zeros(nAnts)
  Tbxx_v=np.zeros(nAnts)
  Tbyy_v=np.zeros(nAnts)

  ind=0

  # Loop over all antennas.
  for antID in resCG.keys():

    # Extract the data. Make the time averaging.
    Tbxx_c[ind]=resCG[antID]['dB'][0].mean()
    Tbyy_c[ind]=resCG[antID]['dB'][1].mean()

    Tbxx_v[ind]=resVG[antID]['dB'][0].mean()
    Tbyy_v[ind]=resVG[antID]['dB'][1].mean()

    ind+=1
  #--> endfor(all antennas)

  # Make and print the stats.

  print("\n------- Constant gains:\n")

  print("T_Bxx mean = ",Tbxx_c.mean(),", STD = ",Tbxx_c.std())
  print("T_Byy mean = ",Tbyy_c.mean(),", STD = ",Tbyy_c.std())


  print("\n------- Variable gains:\n")

  print("T_Bxx mean = ",Tbxx_v.mean(),", STD = ",Tbxx_v.std())
  print("T_Byy mean = ",Tbyy_v.mean(),", STD = ",Tbyy_v.std())

  return {"C": [Tbxx_c,Tbyy_c], "V": [Tbxx_v,Tbyy_v]}

#-----------------------------------------------------------------------------
#           6. Analysis: Compare the dPWV proxies vs. phase variations
#-----------------------------------------------------------------------------

def dpwvXphi_plots(doSun=False,useSQLD=False):
  """
  *** So far specific just for the solar MS 2022.1.00031.CSV, Band 5 !!! ***

  Plot the phase variations opverplotted by our dPWV proxy, scaled by
  the theoretical factor dPhi/dPWV=-25.0 for each antenna - see the PR11.
  This is just the quick-check diagnostics.

  We may use ACs or SQLD for that - see dPWV(); this is controlled by *useSQLD*.
  Whether we do this for the BP-calibrator or the Sun is controlled by *doSun*.
  """
```

```
#-- 'Static' settings.

# Our input MS.
vis='uid___A002_X110fd13_X1099e.ms'

# Find the antenna list (the IDs and corresponding names)
msmd.open(vis)
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()
msmd.done()


#-- Do we analyse the BP calibrator or the Sun? It controls the atm/tgtScan.
if (doSun):

  # The Sun's target-scan and its preceding ACD-calibration ATM scan.
  atmScan=8
  tgtScan=9

  srcName='sun'
  # Self-calibrated solar phase solution
  gTable='../calibrated/lowres-sc.phase_int'

else:

  # BP calibrator: The target and ACD-calibration scans.
  atmScan=4
  tgtScan=5

  srcName='BPcal'
  # Band-pass calibrator phase gaintable - raw, i.e. without WVR correction.
  gTable=vis+'.ap_pre_bandpass'

#-- Do we use ACs or SQLD for dPWV calculations?

if (useSQLD):
  # SPW type - part of the output-plots directory name.
  spwType='sqld'

else: # ACs
  # SPW type - part of the output-plots directory name.
  spwType='ac'

#-- Directory, where to save the output plots.
pngDir+='./dwDphi_'+spwType+'/'+srcName+'/'
if (not os.path.exists(pngDir)): os.makedirs(pngDir)

# In the MS calibration script, DV10 = id37 has been picked as the reference.
refAnt=37

#-- Calculate the dPWV proxy for the *reference* antenna.
print('Calculating reference antenna.')
mjd_ref,dpw_refXX,dpw_refYY=dPWV(vis,atmScan,tgtScan,refAnt,useSQLD)

#-- Loop over all antennas.

for myant in antIDs:
```

```
    # Report progress.
    print('Calculating antenna ',int(myant),'<-->',antNames[myant])

    #- Calculate the dPWV proxy for antenna ant.
    time_pwv,dpw_XX,dpw_YY=dPWV(vis,atmScan,tgtScan,int(myant),useSQLD)

    # Subtract the dPWV at the reference antenna.
    dwx=dpw_XX-dpw_refXX
    dwy=dpw_YY-dpw_refYY

    #- Time->Phase map/dict.
    tpDict=getPhase(gTable,myant,5) # SPW 5

    time_phi=tpDict['mjd']
    phx=tpDict['phiXX']
    phy=tpDict['phiYY']

    #- Make a plot with an overplot: Phases vs. dPWV

    # Plot phases in red: XX solid line, YY dashed.
    plt.plot(time_phi,phx,'-r')
    plt.plot(time_phi,phy,'--r')

    # Plot our dPWV-proxy, with refAnt-offset subtracted and scaled by -25.0
    # Blue color, XX solid, YY dashed
    plt.plot(time_pwv,-25.0*dwx,'-b')
    plt.plot(time_pwv,-25.0*dwy,'--b')

    # Add the antenna name as a plot title.
    plt.title(antNames[myant])

    # Save to a file and close.
    plt.savefig(pngDir+'dwDp_'+antNames[myant]+'.png')
    plt.show()
    plt.close()

  return

#-------------------------------------------------------------------------------

def dpwvXphi_corrs(doSun=False,useSQLD=False):
  """
  *** So far just for the solar MS project 2022.1.00031.CSV, Band 5!!! ***

  Calculate actual correlations between dPWV and phases. Just indicative plots
  are performed by dpwvXphi_plots() with anticipated coefficient of linear
  regression to be -25.0 (see the study in PR11, March 2025). Here, we perform
  actual correlation calculations, finding the coefficients of linear regression
  between dPWV and phases. However, the situation is not that simple: the
  preliminary analysis has shown (see PR11) that the phases tend to (perhaps
  linearly) grow also with time and time-based de-trending of the phase series
  should be performed. In order to do that, parametrized linear regression
  is computed here using paregression().

  We may use ACs or SQLD for that - see dPWV(); this is controlled by *useSQLD*.
  Whether we do this for the BP-calibrator or the Sun is controlled by
  *doSun*.
```

```
"""

#-- 'Static' settings.

# Our input MS.
vis='uid___A002_X110fd13_X1099e.ms'

# Find the antenna list (the IDs and corresponding names)
msmd.open(vis)
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()
msmd.done()

#-- Do we analyse the BP calibrator or the Sun? It controls the atm/tgtScan.
if (doSun):

  # The Sun's target-scan and its preceding ACD-calibration ATM scan.
  atmScan=8
  tgtScan=9

  srcName='sun'
  # Self-calibrated solar phase solution
  gTable='../calibrated/lowres-sc.phase_int'

else:

  # BP calibrator: The target and ACD-calibration scans.
  atmScan=4
  tgtScan=5

  srcName='BPcal'
  # Band-pass calibrator phase gaintable - raw, i.e. without WVR correction.
  gTable=vis+'.ap_pre_bandpass'

#-- Do we use ACs or SQLD for dPWV calculations? We have different time
#   resolutions: Phase solution may be integration based (for the Sun), so
#   of the same time cadence as the AC data, or 10x undersampled (for the BP
#   calibrator, where we use solint='1.92s' in gaincal()). The SQLD is 12x
#   faster than ACs.

if (useSQLD):
  # SPW type - part of the output-plots directory name.
  spwType='sqld'
else: # ACs
  # SPW type - part of the output-plots directory name.
  spwType='ac'

#-- Directory, where to save the output plots.
pngDir='./1corr_dwDphi_'+spwType+'/'+srcName+'/'
if (not os.path.exists(pngDir)): os.makedirs(pngDir)

# In the MS calibration script, DV10 = id37 has been picked as the reference.
refAnt=37

# Calculate the dPWV proxy for the reference antenna.
print('Calculating reference antenna.')
mjd_ref,dpw_refXX,dpw_refYY=dPWV(vis,atmScan,tgtScan,refAnt,useSQLD)
```

```
#-- Loop over all antennas

for myant in antIDs:

  # Report progress.
  print('Calculating antenna ',int(myant),'<-->',antNames[myant])

  # Extract phases as the time -> phase map/dictionary.
  tpDict=getPhase(gTable,int(myant),5)  # use SPW 5

  time_phi=tpDict['mjd']
  phx=tpDict['phiXX']
  phy=tpDict['phiYY']

  # Calculate the dPWV proxy for antenna ant.
  time_pwv,dpw_XX,dpw_YY=dPWV(vis,atmScan,tgtScan,int(myant),useSQLD)

  # Subtract the dPWV at the reference antenna.
  ddpwXX=dpw_XX- dpw_refXX
  ddpwYY=dpw_YY- dpw_refYY

  # We have different time resolutions/cadences: Phase solution may be
  # integration-based (for the Sun), so of the same time cadence as the AC
  # dPWV data, or 10x undersampled (for the BP calibrator, where we use
  # solint='1.92s' in gaincal()). The SQLD is yet 12x faster than ACs, so if
  # we use it, our dPWV proxy can have even 120x better resolution than the
  # phase solutions (if we use 10 integrations as in the BP calibrator case).
  # Solution: Make the resolution/cadence conforming to that of the *phases*.
  dwx=np.interp(time_phi,time_pwv,ddpwXX)
  dwy=np.interp(time_phi,time_pwv,ddpwYY)

  #-- Calculate the correlations between dPWV and phase with optimised
  #   removal of the time-trend.
  rcXX=paregression(dwx,phx)
  rcYY=paregression(dwy,phy)

  # Extract the time-trend.
  ttXX=rcXX['tTrend']
  ttYY=rcYY['tTrend']

  # Extract the coefficient of linear correlation.
  KXX=rcXX['K']
  KYY=rcYY['K']

  # Extract the offset of linear correlation.
  QXX=rcXX['Q']
  QYY=rcYY['Q']

  # Extract fit-quality measure (Xi-square) of the linear correlation.
  Xi2XX=rcXX['Xi2']
  Xi2YY=rcYY['Xi2']

  # Print the extracted values
  print('T-trend = ',ttXX,ttYY)
  print('K = ',KXX,KYY)
  print('Q = ',QXX,QYY)
  print('Xi2 = ',Xi2XX,Xi2YY)
```

```
    #- Make scatter plots: dPWW over time-detrended phase. Eeach pol. extra

    # Time vector, start with 1.
    tVec=np.arange(phiSize)+1.0

    # Detrended phases.
    pdXX=phx-ttXX*tVec
    pdYY=phy-ttYY*tVec

    # Make XX and YY plots as subplots (horizontally)
    ff,(sp1,sp2)=plt.subplots(1,2)

    # Plot the correlations as red dots.
    sp1.plot(dwx,pdXX,'.r')
    sp2.plot(dwy,pdYY,'.r')

    # Add the antenna name as a plot title.
    plt.title(antNames[myant])

    # Save to a file and close.
    plt.savefig(pngDir+'dwDp_'+antNames[myant]+'.png')
    plt.show()
    plt.close()

  return

#--------------------------------------------------------------------------------

def blindPhiFix(doSun=False,useSQLD=False):
  """
  *** So far just for the solar MS project 2022.1.00031.CSV, Band 5!!! ***

  Basically, we do the same calculations like in dpwvXphi_plots() but now
  we try to make blind corrections of the phases by dPWV with two options:

  1. We fix phase by subtracting dPWV scaled by the theoretical factor -25.0.
  2. Like in case (1.), but use the factor found by the correlation analysis.

  Phases fixed by both methods are plotted into a single chart for comparison.
  We would like to see, what is residal phase variation after the correction.

  @todo: For the Sun we do NOT have yet 'indpendent' phase solution to compare,
  we do it for the phases of the BP cal. Of course, it has no sense to compare
  the dPWV curve (scaled by -25.0) for the Sun with the phases for the BP-cal.
  The only goal is to see, whether we have similar amplitude and typical
  dynamics comparable or not.
  """

  #-- 'Static' settings.

  # Our input MS.
  vis='uid___A002_X110fd13_X1099e.ms'

  # Find the antenna list (the IDs and corresponding names)
  msmd.open(vis)
  antIDs=myMsmd.antennaids()
  antNames=myMsmd.antennanames()
  msmd.done()
```

```
#-- Do we analyse the BP calibrator or the Sun? It controls the atm/tgtScan.
if (doSun):

  # The Sun's target-scan and its preceding ACD-calibration ATM scan.
  atmScan=8
  tgtScan=9

  srcName='sun'
  # Self-calibrated solar phase solution
  gTable='../calibrated/lowres-sc.phase_int'

else:

  # BP calibrator: The target and ACD-calibration scans.
  atmScan=4
  tgtScan=5

  srcName='BPcal'
  # Band-pass calibrator phase gaintable - raw, i.e. without WVR correction.
  gTable=vis+'.ap_pre_bandpass'

#-- Do we use ACs or SQLD for dPWV calculations? We have different time
#   resolutions: Phase solution may be integration based (for the Sun), so
#   of the same time cadence as the AC data, or 10x undersampled (for the BP
#   calibrator, where we use solint='1.92s' in gaincal()). The SQLD is 12x
#   faster than ACs.

if (useSQLD):
  # SPW type - part of the output-plots directory name.
  spwType='sqld'
else: # ACs
  # SPW type - part of the output-plots directory name.
  spwType='ac'


#-- Directory, where to save the output plots.
pngDir='./blindFix_'+spwType+'/'+srcName+'/'
if (not os.path.exists(pngDir)): os.makedirs(pngDir)

# Diagnostic output to a text file in the same dir as the plots.
diagTxt=open(pngDir+'diag_outs.txt','w')


# In the MS calibration script, DV10 = id37 has been picked as the reference.
refAnt=37

# Calculate the dPWV proxy for the reference antenna.
print('Calculating reference antenna.')
mjd_ref,dpw_refXX,dpw_refYY=dPWV(vis,atmScan,tgtScan,refAnt,useSQLD)


#-- Loop over all antennas

for myant in antIDs:

  # Report progress.
```

```
print('Calculating antenna ',int(myant),'<-->',antNames[myant])

# Extract phases as the time -> phase map/dictionary.
tpDict=getPhase(gTable,int(myant),5) # use SPW 5

time_phi=tpDict['mjd']
phx=tpDict['phiXX']
phy=tpDict['phiYY']

# Calculate the dPWV proxy for antenna ant.
time_pwv,dpw_XX,dpw_YY=dPWV(vis,atmScan,tgtScan,int(myant),useSQLD)

# Subtract the dPWV at the reference antenna.
ddpwXX=dpw_XX-dpw_refXX
ddpwYY=dpw_YY-dpw_refYY

# Make the resolutions conforming to that of the phase solution.
# See explanation at the same point of dpwvXphi_corrs().
dwx=np.interp(time_phi,time_pwv,ddpwXX)
dwy=np.interp(time_phi,time_pwv,ddpwYY)

#-- Calculate the correlations between dPWV and phase with optimised
#   removal of the time-trend in phases.

rcXX=paregression(dwx,phx)
rcYY=paregression(dwy,phy)

# Extract the optimized time-trend coeffcients (dPhi=t_phi*i) for phases.
ttXX=rcXX['tTrend']
ttYY=rcYY['tTrend']

# Extract the coefficient of linear correlation between dPWV and dPhi.
KXX=rcXX['K']
KYY=rcYY['K']

# Extract the offset of linear correlation between dPWV and dPhi.

QXX=rcXX['Q']
QYY=rcYY['Q']

# Extract fit-quality measure (Xi-square) of the linear correlation.
Xi2XX=rcXX['Xi2']
Xi2YY=rcYY['Xi2']

# Print the extracted values
print('T-trend = ',ttXX,ttYY)
print('K = ',KXX,KYY)
print('Q = ',QXX,QYY)
print('Xi2 = ',Xi2XX,Xi2YY)


#-- Compare the optimized detrend in phases with just the blind detrending
#   the dPWV.

# Time measured as an integration index.
iVect=np.arange(phiSize)

# Time-trends solely in dPWV (blind = no phase knowledge)
```

```
wtx=np.polyfit(iVect,dwx,1)[0]
wty=np.polyfit(iVect,dwy,1)[0]

# How much they are different (by what factor) from the optimized detrend?
# We use scaling t_phi=-KK*t_pwv between the linear time-trend equivalently
# expressed in terms of dPWV series (instead of dPhi one).
trFactorXX=-KXX*wtx/ttXX
trFactorYY=-KYY*wty/ttYY


#-- Blindly fix the phases and plot the residuals.

# Scatter-ranges for the uncorrected phases. We compare later with corrected
# and discard those that worsen the situation from plotting

px0Range=np.max(phx)-np.min(phx)
py0Range=np.max(phy)-np.min(phy)


#- Blind correction
# Method 1: Fixed-value scale factor of -25.

bc_pxx1=phx+25.0*dwx
bc_pyy1=phy+25.0*dwy


# Scatter-ranges after correction by method 1
px1Range=np.max(bc_pxx1)-np.min(bc_pxx1)
py1Range=np.max(bc_pyy1)-np.min(bc_pyy1)


# Method 2: Use the correlation factor found by paregression()

bc_pxx2=phx-KXX*dwx
bc_pyy2=phy-KYY*dwy

# Scatter-ranges after correction by method 2
px2Range=np.max(bc_pxx2)-np.min(bc_pxx2)
py2Range=np.max(bc_pyy2)-np.min(bc_pyy2)

# For those antennas, where the blind-correction is not really bad, print
# the ratio between the optimized (using knowledge of phases) and
# and the blind time-trend coefficients for dPWV to a text file.
if (px1Range < 1.5*px0Range):
  print('De-trending factor XX for antenna: ',antNames[myant],': ',
        trFactorXX,file=diagTxt)

if (py1Range < 1.5*py0Range):
  print('De-trending factor YY for antenna: ',antNames[myant],': ',
        trFactorYY,file=diagTxt)


#- Init plots

# Set the sizes of the plot
plt.rcParams["figure.figsize"] = (10,5)
# Make XX and YY plots as subplots (horizontally)
ff,(sp1,sp2)=plt.subplots(1,2)
```

```
    # Add the antenna name as a plot title.
    plt.title(antNames[myant])


    #-  XX: left sub-plot
    # Plot the method 1 results as a red line, if the scatter is not too big.
    if (px1Range < 1.5*px0Range): sp1.plot(time_phi,bc_pxx1,'-r')
    # Overplot the method 2 results as a blue line, if scatter is not too big.
    if (px2Range < 1.5*px0Range): sp1.plot(time_phi,bc_pxx2,'-b')
    # Overplot the uncorrected phase for comparison as a green line (always).
    sp1.plot(time_phi,phx,'-g')

    #-  YY: right sub-plot

    # Plot the method 1 results as a red line, if the scatter is not too big.
    if (py1Range < 1.5*py0Range): sp2.plot(time_phi,bc_pyy1,'-r')
    # Overplot the method 2 results as a blue line, if scatter is not too big.
    if (py2Range < 1.5*py0Range): sp2.plot(time_phi,bc_pyy2,'-b')
    # Overplot the uncorrected phase for comparison as a green line (always).
    sp2.plot(time_phi,phy,'-g')

    #- Save to a file and close.
    plt.savefig(pngDir+'phi_residua_'+antNames[myant]+'.png')
    plt.show()
    plt.close()

    # Flush the diagTxt buffer.
    diagTxt.flush()

  # Close the (single) diagnostic output file.
  diagTxt.close()
  return

#-----------------------------------------------------------------------------
#  7. Compare phases by dPWV proxies with those from wvrgcal() - BP calibrator
#-----------------------------------------------------------------------------


def wvrPhase(vis,antID,spw):
  """
  Extract WVR-predicted phase delays for the PB-calibrator as a function of
  time for the WVR gain table that has been created by the call

  @code{sh}
  wvrgcal(vis=vis,caltable='wvr_gcal.W', segsource=True, sourceflag=['Sun'],
  statsource='J1256-0547',refant='DV10',maxdistm=1.0,minnumants=3)
  @endcode

  Basically, very similar to the getPhase() above, just we specify also the
  field inside the table-selection.

  The getPhase() method is called as well inside - the only purpose is to get
  easilly the time-range for the BP scan and refine the selection of the
  WVR-gcal table rows.

  The last two args tp wvrgcal() (added after PM12 in may 2025) prevents
  wvrgcal() from making caltables in antennas that do not have the WVR device
  (namely 7m antennas) or have it broken (without those switches wvrgcal()
```

```
    produces a 'spurious' table for non-working WVR by interpolation of the
    neighborhood antennas).
    """

    # Create the table tool and use it to open the WVR caltable.
    cTab=tbtool()
    cTab.open('wvr_gcal.W')

    # Construct a query string: Selection by antenna, spw, and field.
    selection="ANTENNA1=="+str(antID)+" && SPECTRAL_WINDOW_ID="+str(spw)
    selection+=" && FIELD_ID==1" # Field 1 <--> BP calibrator

    # Extract the sub-table that correspond to the selection criteria.
    # We need just the time and gain columns.
    ctSel=cTab.query(selection,sortlist="TIME",columns="TIME,CPARAM")

    # Extract all the column data for the above selection.
    times_all=ctSel.getcol("TIME")
    gains_all=ctSel.getcol("CPARAM")

    # Data are extracted, we may close the tab tool.
    cTab.done()

    # The gains array has a shape (1 x 1 x nTimeSamples), there are no separate
    # polarization values, just I-component of Stokes. Get rid of the two
    # degenarated axes.
    gI_all=gains_all[0][0]

    # FIELD_ID is not the ultimate selector - we still have the data from other
    # scans than the BP scan. Call to getPhase() is the easiest way how to get the
    # times of the BP scan. Alternativelly, msmdtool() with getTimesForScan() may
    # be used.
    gTable=vis+'.ap_pre_bandpass'
    timesBP=getPhase(gTable,antID,spw)["mjd"]

    tScanBeg=np.min(timesBP)
    tScanEnd=np.max(timesBP)

    # Sub-select just the BP-scan data.
    # Note: Pytom requires using '&' operator and not 'and' here.
    scanRange=np.where((times_all >= tScanBeg) & (times_all <= tScanEnd))

    times=times_all[scanRange]
    gI=gI_all[scanRange]

    # We are interested just in the phases.
    phase=np.angle(gI)

    # Use np.unwrap in order to get rid of the phase jumps at +/- PI.
    uPhase=np.unwrap(phase)

    # Form the returned result as a dictionary.
    # Return the phases of XX and YY (the same value) separatelly in order to have
    # the result conforming to those of getPhase() and dPWV()
    pDict={'mjd':times,'phiXX':uPhase,'phiYY':uPhase}

    return pDict
```

```
#--------------------------------------------------------------------------------

def pwvCompare():
  """
  *** So far just for the solar MS project 2022.1.00031.CSV, Band 5!!! ***

  Compare systematically (for all antennas) atmospheric phase delays as
  predicted from:

  1. Our dPWV() proxy (calculated from signal of science SPWs),
  2. wvrgcal()-produced caltable (calculated from the hi-cadence WVR data).

  We can do it just for the BP calibrator, so no free source paramater in this
  case. We are using the channel-integrated AC windows (not SQLD) in dPWV().

  Creates two-panel plot (left for XX, right for YY) for each antenna,
  having our dPWV() prediction in red and wvrgcal() calculation in blue.
  """

  # Input MS.
  vis='uid___A002_X110fd13_X1099e.ms'

  # Output directory, where to save the plots.
  pngDir='./PWV_compare_AC/'
  if (not os.path.exists(pngDir)): os.makedirs(pngDir)

  # Calculate the dPWV proxy for the reference antenna (DV10 = id 37).
  print('Calculating the reference antenna.')
  mjd_ref,dpw_refXX,dpw_refYY=dPWV(vis,4,5,37)

  #-- Loop over all antennas

  # Find the antenna list (the IDs and corresponding names)
  msmd.open(vis)
  antIDs=myMsmd.antennaids()
  antNames=myMsmd.antennanames()
  msmd.done()

  for myAnt in antIDs:

    # Report progress.
    print('Processing antenna ',int(myAnt),'<-->',antNames[myAnt])

    # Calculate our dPWV proxy for the BP-scan=5 using DL-cals from atmScan=4
    time_d,pwvXX,pwvYY=dPWV(vis,4,5,int(myAnt))

    # Subtract the reference-antenna dPWV
    pwvXX=pwvXX-dpw_refXX
    pwvYY=pwvYY-dpw_refYY

    # Extract wvrgcal()-produced phases for SPW 5.
    phiWVR=wvrPhase(vis,myAnt,5)

    # Extract times for the W-caltable.
    time_w=phiWVR['mjd']

    # Phases for XX and YY extra.
    phi_w_XX=phiWVR['phiXX']
```

```
    phi_w_YY=phiWVR['phiYY']


    #- Init plots

    # Set the sizes of the plot
    plt.rcParams["figure.figsize"] = (10,5)
    # Make XX and YY plots as subplots (horizontally)
    ff,(sp1,sp2)=plt.subplots(1,2)
    # Add the antenna name as a plot title.
    plt.title(antNames[myAnt])

    #-  XX: left sub-plot
    # Plot the dPWV() results scaled by -25.0 as a red line.
    sp1.plot(time_d,-25.0*pwvXX,'-r')
    # Overplot the wvrgcal() results as a blue line.
    sp1.plot(time_w,phi_w_XX,'-b')

    #-  YY: right sub-plot
    # Plot the dPWV() results scaled by -25.0 as a red line.
    sp2.plot(time_d,-25.0*pwvYY,'-r')
    # Overplot the wvrgcal() results as a blue line.
    sp2.plot(time_w,phi_w_YY,'-b')


    #- Save to a file and close.
    plt.savefig(pngDir+'pwv_compare_'+antNames[myAnt]+'.png')
    plt.show()
    plt.close()

  return


#------------------------------------------------------------------------------

def phiFixCompare(vis,atmScan,tgtScan,refAnt,spwID,spwRes=-1):
  """
  Compare standard WVRgcal()-based correction of *BP-cal* phases with that based
  on our dPWV() proxy.

  The dPWV() can be calculated either from SQLD or AC SPWs, the AC still
  having two sub-options: Channel-integrated or channel resolved SPW.
  In the latter case, dPWV_spwPart() is called internally, with integration
  only over the lower half of the SPW channels.

  Which data are used, it is auto-detected from the *spwID* parameter.
  In case of the channel-integrated data (namely SQLDs) we need to specify
  the SPW with channel-resolved data in addition: This is used (1) for the
  calculation of (usually small) correction factor dTsky/df that implies from
  integration over the SPW bandwidth in case that tau_1 and Tsky are function
  of frequency, (ii) for comparison with raw and wvrgcal()-corrested phases;
  these are measured always in spec-resolved SPWs. The correspsonding resolved
  SPW is given by the parameter *spwRes*. If spwID is a SQLD spw, then
  this *spwRes* MUST be set (we have no chance to deduce it). On the other hand,
  in AC-integrated data it can be inferred as spwID-1 and in case of AC-resolved
  data it is just directly the *spwID*.

  First part of the job basically replicates the code in blindPhiFix() in order
```

```
to see comaprison between ucorrected phase and its improved counterpart fixed
using our dPWV() proxy and the theoretical factor -25.0 (called "Method 1" in
blindPhiFox())

The second part lies in taking the p-cal table (pre-Bandpass) with data
corrected also by WVR calibration (the 'standard' p-cal table for the BP
calibrator is NOT corrected for WVR) as the source. The following sequence
of steps has been performed in order to obtain the WVR-corrected p-cal table
mso+'.ap_pre_bandpass-wvrFix' in CASA shell:


@code{sh}
clearcal('uid___A002_X110fd13_X1099e.ms',field='1')

wvrgcal(vis=vis,caltable='wvr_gcal.W', segsource=True, sourceflag=['Sun'],
statsource='J1256-0547',refant='DV10',maxdistm=1.0,minnumants=3)

from casarecipes.almahelpers import tsysspwmap
tsysmap = tsysspwmap(vis = mso, tsystable = mso+'.tsys', tsysChanTol = 1)

gaincal(vis=mso,caltable=mso+'.ap_pre_bandpass-wvrFix',field='1',scan='5',
solint='1.92s',refant='DV10',calmode='p',
gaintable=[mso+'.antpos',mso+'.tsys','wvr_gcal.W'],
gainfield=['','1','1'],interp=['','linear,spline','nearest'],
spwmap=[[],tsysmap,[]])

@endcode


This way, antpos, Tsys, and WVR corrections has been applied on-the-fly.

This method cannot be used - even hypothetically - for the solar scans
as we have no wvrgcaltable for the Sun to be compared with the dPWV() proxy.
"""

#-- What type of data we are using for analysis? Check the spw definition.
myMsmd=msmdtool()
myMsmd.open(vis)

freqs=myMsmd.chanfreqs(spwID)*1.0e-9

# Extract also the list of all antennas once we have the msmd tool ready.
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()

myMsmd.done()

nChans=len(freqs)

# Corresponding high-frequency SPW for subtraction of the dry component.
# To be set below in line with the spwID type.
spwHigh=-1

if (spwID in [0,1,2,3,13,14,15,16]):
  # SQLD SPW.
  spwType='sqld'
  # SQLDs follow consecutivelly in line.
  spwHigh=spwID+3
```

```
    # Must have corresponding spwRes defined, we cannot guess it.
    if (spwRes==-1):
      print('Parameter *spwRes* MUST be defined for SQLD spw.')
      return

  else:
    # ACs are interleaved (appear in pairs: resolved/integrated).
    spwHigh=spwID+6


    if (nChans>1):
      # Channel-resolved AC SPW
      spwType='ac-res'
      # We are the resolved SPW
      spwRes=spwID

    else:
      # Channel-integrated AC SPW
      spwType='ac-int'
      # An obvious guess for resolved SPW.
      if (spwRes==-1): spwRes=spwRes=spwID-1


# Output figures directory - separate for each SPW type.
pngDir='./2wvrs_'+spwType+'/'
if (not os.path.exists(pngDir)): os.makedirs(pngDir)


#-- Calculate the dPWV proxy for the reference antenna.
print('Calculating the reference antenna.')

if (spwType=='ac-res'):
  # Chan-resolved SPW: Try to integrate over ~half of it.
  mjd_ref,dpw_refXX,dpw_refYY=dPWV_spwPart(vis,atmScan,tgtScan,refAnt,spwID,
                                           startChan=10,stopChan=nChans/2)
  mjd_ref,dpw_refXX_hi,dpw_refYY_hi=dPWV_spwPart(vis,atmScan,tgtScan,refAnt,
                                                 spwHigh,startChan=10,
                                                 stopChan=nChans/2)


else:
  mjd_ref,dpw_refXX,dpw_refYY=dPWV(vis,atmScan,tgtScan,refAnt,spwID,spwRes)
  mjd_ref,dpw_refXX_hi,dpw_refYY_hi=dPWV(vis,atmScan,tgtScan,refAnt,spwHigh,
                                         spwRes)


#-- Loop over all antennas

for myant in antIDs:

  # Report progress.
  print('Calculating antenna ',int(myant),'<-->',antNames[myant])

  # Calculate the dPWV proxy for the antenna *myant*.
  if (spwType=='ac-res'):
    # Chan-resolved SPW: Try to integrate over ~half of it.
    time_dpw,dpwXX,dpwYY=dPWV_spwPart(vis,atmScan,tgtScan,int(myant),spwID,
```

```
                                                startChan=10,stopChan=nChans/2)
    time_dpw,dpwXX_hi,dpwYY_hi=dPWV_spwPart(vis,atmScan,tgtScan,int(myant),
                                                spwHigh,startChan=10,
                                                stopChan=nChans/2)

else:
    time_dpw,dpwXX,dpwYY=dPWV(vis,atmScan,tgtScan,int(myant),spwID,spwRes)
    time_dpw,dpwXX_hi,dpwYY_hi=dPWV(vis,atmScan,tgtScan,int(myant),spwHigh,
                                    spwRes)


# Subtract the dPWV at the reference antenna.
ddpwXX=dpwXX-dpw_refXX-dpwXX_hi+dpw_refXX_hi
ddpwYY=dpwYY-dpw_refYY-dpwYY_hi+dpw_refYY_hi


# Extract the *uncorrected* phases from the ap-prebandpass caltable.
gTable=vis+'.ap_pre_bandpass'
tpDict=getPhase(gTable,myant,spwRes)

time_ph=tpDict['mjd']
phx=tpDict['phiXX']
phy=tpDict['phiYY']

# Make dPWV's resolution & range conforming to that of the phase solution.
dwx=np.interp(time_ph,time_dpw,ddpwXX)
dwy=np.interp(time_ph,time_dpw,ddpwYY)


#-- Blindly fix the raw phases and plot the residuals.

# Scatter-ranges for the uncorrected phases. We compare later with corrected
# and discard those that worsen the situation from plotting

px0Range=np.max(phx)-np.min(phx)
py0Range=np.max(phy)-np.min(phy)

#- Blind correction by the fixed-value scale factor of -25.

bc_pxx=phx+25.0*dwx
bc_pyy=phy+25.0*dwy

# Scatter-ranges after correction.
px1Range=np.max(bc_pxx)-np.min(bc_pxx)
py1Range=np.max(bc_pyy)-np.min(bc_pyy)

#-- Extract *WVR-corrected phases* from the WVR-fixed ap-pre-BP caltable.
gTableWVRfix=vis+'.ap_pre_bandpass-wvrFix'
tpcDict=getPhase(gTableWVRfix,myant,spwRes)

time_phc=tpcDict['mjd']
phx_c=tpcDict['phiXX']
phy_c=tpcDict['phiYY']


#- Init plots

# Set the sizes of the plot
```

```
    plt.rcParams["figure.figsize"] = (10,5)
    # Make XX and YY plots as subplots (horizontally)
    ff,(sp1,sp2)=plt.subplots(1,2)
    # Add the antenna name as a plot title.
    plt.title(antNames[myant])


    #-  XX: left sub-plot
    # Plot the dPWV-fix results as a red line, if the scatter is not too big.
    sp1.plot(time_ph,bc_pxx,'-r')
    # Overplot the WVR-fixed phases as blue
    sp1.plot(time_phc,phx_c,'-b')
    # Overplot the uncorrected phase for comparison as a green line (always).
    sp1.plot(time_ph,phx,'-g')

    #-  YY: right sub-plot

    # Plot the method 1 results as a red line, if the scatter is not too big.
    sp2.plot(time_ph,bc_pyy,'-r')
    # Overplot the WVR-fixed phases in blue
    sp2.plot(time_phc,phy_c,'-b')
    # Overplot the uncorrected phase for comparison as a green line (always).
    sp2.plot(time_ph,phy,'-g')

    #- Save to a file and close.

    plt.savefig(pngDir+'phi_residua_'+antNames[myant]+'.png')
    plt.show()
    plt.close()

  return


#--------------------------------------------------------------------------------
def phiFixCompare3(vis,atmScan,tgtScan,refAnt,spwLow,spwRes=-1):
  """
  Like phiFixCompare(), but we use dWdP() - the method that takes into account
  the dry-component-caused power fluctuations - for determining the PWV
  variations instead of dPWV() (which takes just a single SPW for the power
  fluctuations).

  Since dWdP() works currently only for chan-integared SPWs, the spwLow may
  be just the lowest SQLD or AC-integrated SPW.

  dWdP() omits the correction for slopes of <Tsky(f)> and tau_1(f) so it does
  not need the spwRes. However, we still need it for the raw and wvrgcal()-fixed
  phases to be compared with our corrections.
  """


  #-- What type of data we are using for analysis? Check the spw definition.
  myMsmd=msmdtool()
  myMsmd.open(vis)

  freqs=myMsmd.chanfreqs(spwLow)*1.0e-9

  # Extract also the list of all antennas once we have the msmd tool ready.
  antIDs=myMsmd.antennaids()
```

```
antNames=myMsmd.antennanames()

myMsmd.done()

nChans=len(freqs)

# Corresponding high-frequency SPW for subtration of the dry component.
# To be set below in line with the spwLow type.
spwHigh=-1

if (spwLow in [0,1,2,3,13,14,15,16]):
  # SQLD SPW.
  spwType='sqld'
  # SQLDs follow consecutivelly in line.
  spwHigh=spwLow+3

  # Must have corresponding spwRes defined, we cannot guess it.
  if (spwRes==-1):
    print('Parameter *spwRes* MUST be defined for SQLD spw.')
    return

else:
  # ACs are interleaved (appear in pairs: resolved/integrated).
  spwHigh=spwLow+6


  if (nChans>1):
    # Channel-resolved AC SPW
    spwType='ac-res'
    # We are the resolved SPW
    spwRes=spwLow

    print('dWdP() inside this function is not ready yet for resolved SPWs!')
    return

  else:
    # Channel-integrated AC SPW
    spwType='ac-int'
    # An obvious guess for resolved SPW.
    if (spwRes==-1): spwRes=spwRes=spwLow-1


# Output figures directory - separate for each SPW type.
pngDir='./2wvrs_3_'+spwType+'/'
if (not os.path.exists(pngDir)): os.makedirs(pngDir)


#-- Create the atmosphere model for our SPWs (Lo/Hi):
#   Use the *resolved* swps corresponding to our integrated spwHigh/spwLow

atmLow=wopacity(vis,spwRes,tgtScan)
atmHigh=wopacity(vis,spwRes+6,tgtScan)

# Collect the sub-models for spws as a map.
atmModel={spwLow:atmLow,spwHigh:atmHigh}

#-- Calculate the dPWV proxy for the reference antenna.
print('Calculating the reference antenna.')
```

```
mjd_ref,dwXX_ref,dwYY_ref,dpXX_ref,dpYY_ref=dWdP(vis,atmScan,tgtScan,
                                                 atmModel,refAnt,spwLow)

#-- Loop over all antennas

for myant in antIDs:

  # Report progress.
  print('Calculating antenna ',int(myant),'<-->',antNames[myant])

  # Calculate the dW (wet) and dP (dry) atm. variations.
  time_dpw,dwXX,dwYY,dpXX,dpYY=dWdP(vis,atmScan,tgtScan,
                                    atmModel,int(myant),spwLow)

  # Subtract the dPWV at the reference antenna.
  ddpwXX=dwXX-dwXX_ref
  ddpwYY=dwYY-dwYY_ref

  # Extract the *uncorrected phases* from the ap-prebandpass caltable.
  gTable=vis+'.ap_pre_bandpass'
  tpDict=getPhase(gTable,myant,spwRes)

  time_ph=tpDict['mjd']
  phx=tpDict['phiXX']
  phy=tpDict['phiYY']

  # Make dPWV's resolution & range conforming to that of the phase solution.
  dwx=np.interp(time_ph,time_dpw,ddpwXX)
  dwy=np.interp(time_ph,time_dpw,ddpwYY)


  #-- Blindly fix the raw phases and plot the residuals.

  # Scatter-ranges for the uncorrected phases. We compare later with corrected
  # and discard those that worsen the situation from plotting

  px0Range=np.max(phx)-np.min(phx)
  py0Range=np.max(phy)-np.min(phy)

  #- Blind correction by the fixed-value scale factor of -25.

  bc_pxx=phx+25.0*dwx
  bc_pyy=phy+25.0*dwy

  # Scatter-ranges after correction.
  px1Range=np.max(bc_pxx)-np.min(bc_pxx)
  py1Range=np.max(bc_pyy)-np.min(bc_pyy)

  #-- Extract the *WVR-corrected phases* from WVR-fixed ap-pre-BP caltable
  gTableWVRfix=vis+'.ap_pre_bandpass-wvrFix'
  tpcDict=getPhase(gTableWVRfix,myant,spwRes)

  time_phc=tpcDict['mjd']
  phx_c=tpcDict['phiXX']
  phy_c=tpcDict['phiYY']
```

```
    #- Init plots

    # Set the sizes of the plot
    plt.rcParams["figure.figsize"] = (10,5)
    # Make XX and YY plots as subplots (horizontally)
    ff,(sp1,sp2)=plt.subplots(1,2)
    # Add the antenna name as a plot title.
    plt.title(antNames[myant])


    #-  XX: left sub-plot
    # Plot the dPWV-fix results as a red line, if the scatter is not too big.
    sp1.plot(time_ph,bc_pxx,'-r')
    # Overplot the WVR-fixed phases as blue
    sp1.plot(time_phc,phx_c,'-b')
    # Overplot the uncorrected phase for comparison as a green line (always).
    sp1.plot(time_ph,phx,'-g')

    #-  YY: right sub-plot

    # Plot the method 1 results as a red line, if the scatter is not too big.
    sp2.plot(time_ph,bc_pyy,'-r')
    # Overplot the WVR-fixed phases in blue
    sp2.plot(time_phc,phy_c,'-b')
    # Overplot the uncorrected phase for comparison as a green line (always).
    sp2.plot(time_ph,phy,'-g')

    #- Save to a file and close.

    plt.savefig(pngDir+'phi_residua_'+antNames[myant]+'.png')
    plt.show()
    plt.close()

  return


#-----------------------------------------------------------------------------

def dpwvXwvr_corrs(SQLD=False):
  """
  ***So far specific just for the solar project 2022.1.00031.CSV, Band 5 !!! ***


  Find correlation and linear trends between our dPWV()-proxy calculated phase
  correction and that found by the WVRgcal().

  Very similar approach as used in dpwvXphi_corrs(), now used for correlation
  analysis between two phase corrections cacluated by two different methods.

  This routine cannot be even theoretically used for the Sun (no wvrgcal table).
  """

  #-- 'Static' settings.

  # Our MS
  vis='uid___A002_X110fd13_X1099e.ms'

  # BP calibrator: The target and ACD-calibration scans.
```

```
atmScan=4
tgtScan=5

# In order to create the homo-located (with the same support) time series
# we use np.interp() in this case - see below. Hence, no rebin factors nor
# data trimming comes here, just the suffix of the out-plot directory.
if (useSQLD):

  # SPW type - part of the output-plots directory name.
  spwType='sqld'

else: # ACs

  # SPW type - part of the output-plots directory name.
  spwType='ac'


# In the MS calibration script, DV10 = id37 has been picked as the reference.
refAnt=37

# Calculate the dPWV proxy for the reference antenna.
print('Calculating dPWV for the reference antenna.')
#--- Try the partial spw integration over SPW 5 instead of using SPW 6
#mjd_ref,dpw_refXX,dpw_refYY=dPWV(vis,atmScan,tgtScan,refAnt,useSQLD)
spwType='spw5'
mjd_ref,dpw_refXX,dpw_refYY=dPWV_spwPart(vis,atmScan,tgtScan,refAnt,
                                         spwRes=5,startChan=10,stopChan=60)

# Directory, where to save the plots.
pngDir='/scratch/barta/LB_Solar/Project_2022.1.00031.CSV/'
pngDir+='Sun_10_a_05_INT/X1099e/2wvrs_'+spwType+'/'
if (not os.path.exists(pngDir)): os.makedirs(pngDir)

# Diagnostic output to a text file in the same dir as the plots.
diagTxt=open(pngDir+'diag_outs.txt','w')


#-- Loop over all antennas

# Find the antenna list (the IDs and corresponding names)
msmd.open(vis)
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()
msmd.done()

for myant in antIDs:

  # Report progress.
  print('Calculating antenna ',int(myant),'<-->',antNames[myant])


  # Calculate the dPWV proxy for antenna ant.
  #--- Try the partial SPW5 integration instead of SPW6.
  #mjd_ant,dpw_XX,dpw_YY=dPWV(vis,atmScan,tgtScan,int(myant),useSQLD)
  time_dpwv,dpw_XX,dpw_YY=dPWV_spwPart(vis,atmScan,tgtScan,int(myant),
                                       spwRes=5,startChan=10,stopChan=60)

  # Subtract the dPWV at the reference antenna.
```

```
ddpwXX=dpw_XX-dpw_refXX
ddpwYY=dpw_YY-dpw_refYY

#-- 1. dataseries: Re-calculate dPWV to phase corrections by the theoretical
#                 scale factor of -25.

dPhiXX=-25.0*ddpwXX
dPhiYY=-25.0*ddpwYY

#-- 2. dataseries: Extract the phase corrections from the wvrgcal() table.

# Extract wvrgcal()-produced phases for SPW 5.
phiWVR=wvrPhase(vis,myant,5)

# Extract times for the W-caltable.
time_wvr=phiWVR['mjd']

# Phases for XX and YY extra.
phi_w_XX=phiWVR['phiXX']
phi_w_YY=phiWVR['phiYY']

#-- Interpolate the phases as from dPWV to the time-points from WVR.

phi_p_XX=np.interp(time_wvr,time_dpwv,dPhiXX)
phi_p_YY=np.interp(time_wvr,time_dpwv,dPhiYY)


#-- Calculate the correlations between dPWV-based and WVR-based
#   phase-corrections with optimised removal of the time-trend.

rcXX=paregression(phi_w_XX,phi_p_XX)
rcYY=paregression(phi_w_YY,phi_p_YY)

# Extract the optimized time-trend coeffcients (dPhi=t_phi*i) for phases.
ttXX=rcXX['tTrend']
ttYY=rcYY['tTrend']

# Extract the coefficient of linear correlation between dPWV and dPhi.
KXX=rcXX['K']
KYY=rcYY['K']

# Extract the offset of linear correlation between dPWV and dPhi.

QXX=rcXX['Q']
QYY=rcYY['Q']

# Extract fit-quality measure (Xi-square) of the linear correlation.
Xi2XX=rcXX['Xi2']
Xi2YY=rcYY['Xi2']

# Print the extracted values for quick inspection.
print('T-trend = ',ttXX,ttYY)
print('K = ',KXX,KYY)
print('Q = ',QXX,QYY)
print('Xi2 = ',Xi2XX,Xi2YY)

#-- Print the results one more to the table in the file.
print(antNames[myant],'\t',ttXX,'\t',ttYY,'\t',KXX,'\t',KYY,'\t',
```

```
            QXX,'\t',QYY,'\t',Xi2XX,'\t',Xi2YY,file=diagTxt)


        #- Make scatter plots: dPhi(WVR) vs.dPhi(PWV)

        phiSize=time_wvr.shape[0]

        # Time-index vector, start with 1.
        tVec=np.arange(phiSize)+1.0

        # De-trended PWV-based phases.
        pdXX=phi_p_XX-ttXX*tVec
        pdYY=phi_p_YY-ttYY*tVec

        # Make XX and YY plots as subplots (horizontally)
        ff,(sp1,sp2)=plt.subplots(1,2)

        # Plot the correlations as red dots.
        sp1.plot(phi_w_XX,pdXX,'.r')
        sp2.plot(phi_w_YY,pdYY,'.r')

        # Add the antenna name as a plot title.
        plt.title(antNames[myant])

        # Save to a file and close.
        plt.savefig(pngDir+'dwvrDpwv_'+antNames[myant]+'.png')
        plt.show()
        plt.close()

        # --> endfor(over-all-antennas)

    # Close the (single) diagnostic output file.
    diagTxt.close()
    return

#-------------------------------------------------------------------------
#                7.1 PWV for the Sun vs. phases
#-------------------------------------------------------------------------

def sol_PhaseFix(antID,spw,pwvMap):
    """
    Compare the suggested phase-fix based on the full-PWV calculations as
    performed by dWdBdG(), with the linear interpolation of the pahse-inf solution
    obtained from the adjacent (bracketing) phase-cal scans. We would like to see
    whether the phase-inf interpolated solution has at least some significance or
    whether it is totally 'out', e.g. the phase wraps over 2PI in reality.

    Results:
    (1) The trend in PWV-based phases (Sun) roughly corresponds to the linear
    interpolated phi-Cal solution.
    (2) There is, however, an offset that may be even as large as 720 degs (i.e.
    2 lambdas in path).
    (3) A bit worrysome is the difference between XX and YY polarizations in
    phi-Cal solution.
    (4) Albeit the trend  is corresponding between both, the PWV-based phases
    exhibit variation over the scan that are easilly bigger then 360 deg.
    """
```

```
  #-- Phase-cal INF solution interpolated over a solar scan.

  # The phase-inf solutions have been obtained from the ms.split MS, where the
  # SPWs has been meanwhile renumbered. Make a map between raw and split spwIds.
  splitMap={5:0,7:1,9:2,11:3}

  # The *spw* parameter is in the original notation, find its split counterpart.
  spwSplit=splitMap[spw]

  # Extract the phases in their time dependence.
  phaseDict=getPhase('uid___A002_X110fd13_X1099e.ms.split.phase_inf',
                     antID,spwSplit)

  # Skip the very first phase-inf solution, it comes from the BP-cal scan.
  phiTime=phaseDict['mjd'][1:]
  phiXX=phaseDict['phiXX'][1:]
  phiYY=phaseDict['phiYY'][1:]


  #-- PWV-based phase delays.

  # Reference-antennna values.
  refAnt=37
  pwvRef=pwvMap[refAnt]['dW']

  # Current antID
  pwvTime=pwvMap[antID]['time']
  pwvCurr=pwvMap[antID]['dW']

  # PWV-based phase delay.
  # Approximate quick calculation for LO frequency. Use the central frequency
  # for each SPW in the next improved version!!!
  dPhi=-25.0*(pwvCurr-pwvRef)

  #-- Plot comparison.

  # Scale the pahses to degrees.
  r2d=180.0/math.pi

  plt.plot(pwvTime,r2d*dPhi,'b-')

  plt.plot(phiTime,r2d*phiXX,'r-')
  plt.plot(phiTime,r2d*phiYY,'g-')

  return

#-----------------------------------------------------------------------------

def sol_pwv2selfCal(antID,spw):
  '''
  Comparison of PWV-based phase calibration of the Sun with self-cal gaintable
  solution. Because the self-cal phases are already apriori corrected by
  the linear trend (via applycal() of phase-cal gaintable), we plot also the
  phase-cal linearly-interpolated solution.
  '''

  #-- 1. Phase-cal INF solution interpolated over a solar scan.
```

```
  # The phase-inf solutions have been obtained from the ms.split MS, where the
  # SPWs has been meanwhile renumbered. Make a map between raw and split spwIds.
  splitMap={5:0,7:1,9:2,11:3}

  # The *spw* parameter is in the original notation, find its split counterpart.
  spwSplit=splitMap[spw]

  # Extract the phases in their time dependence.
  phaseDict=getPhase('uid___A002_X110fd13_X1099e.ms.split.phase_inf',
                     antID,spwSplit)

  # Skip the very first phase-inf solution, it comes from the BP-cal scan.
  phiTime=phaseDict['mjd'][1:]
  phiXX=phaseDict['phiXX'][1:]
  phiYY=phaseDict['phiYY'][1:]

  #-- 2. Solar phases by PWV proxy.

  # Extract the phases in their time dependence.
  phaseDictPWV=getPhase('sol_pwvCalTable.W',antID,spwSplit)

  pwvTime=phaseDictPWV['mjd']
  pwvPhiXX=phaseDictPWV['phiXX']
  pwvPhiYY=phaseDictPWV['phiYY']

  #-- 3. Solar phases by self-calibration.

  # Extract the phases in their time dependence.
  phaseDictSC=getPhase('sol_selfCal-phase-int',antID,spwSplit)

  scTime=phaseDictSC['mjd']
  scPhiXX=phaseDictSC['phiXX']
  scPhiYY=phaseDictSC['phiYY']

  #-- Plot comparison.

  # Scale the phases to degrees.
  r2d=180.0/math.pi

  # Make a difference between selfcal and PWV-based solutions.
  # Wrap-back the artificial jumps - they are failed self-cal solutions.
  diffXX=r2d*wrapBack(pwvPhiXX-scPhiXX)
  diffYY=r2d*wrapBack(pwvPhiYY-scPhiYY)

  plt.plot(pwvTime,diffXX,'r-')
  plt.plot(pwvTime,diffYY,'b-')

  plt.plot(phiTime,r2d*phiXX,'r-')
  plt.plot(phiTime,r2d*phiYY,'b-')

  return

#--------------------------------------------------------------------------

def sol_pwvGcal(pwvMap,vis,refAnt,pwvCalTableName='sol_pwvCalTable.W'):
  """
  Create the phase calibration table for the Sun in the same style as
  wvrgcal() makes it for non-solar objects.
```

```
We use the full-PWV time series per antenna as found by dWdBdG() routine
and stored in *pwvMap*.



@todo: Generalize for the case we have multiple solar scans.
"""

# This is to be generalized. In case of multiple soar scans, also the *pwvMap*
# parameter shall have a bit different structure (additional scan dimension).
solScan=9

#-- Prepare the empty pwvCalTable with the structure of phase gaintable.

# Remove the old version first.
os.system('rm -rf ' + pwvCalTableName)

# Copy the structure of the phase-inf gaintable as an empty 'template'.
gCalTable=tbtool()
gCalTable.open(vis+'.split.phase_inf')
pwvCalTable=gCalTable.copy(pwvCalTableName,deep=True,norows=True)
pwvCalTable.flush()

# Close both (original & coppied template) tables.
pwvCalTable.done()
gCalTable.done()

#-- Find the per-SPW Phase Factors PF=dPhi(f_spw)/dPWV from ATM model.

myAtm=attool()
setAtmTool4SPWs(myAtm,vis,solScan,wvHeightScale=1.5,tempDropPerKm=-5.6)

# Set approximate PWV, the path length should not actually depend on that.
pwvMean=aU.getMedianPWV(vis)[0]
myAtm.setUserWH2O(qa.quantity(pwvMean,'mm'))

# Phase delays per 1mm PWV [rad/mm] for each SPW.
nSpws=myAtm.getNumSpectralWindows()
PF=np.zeros(nSpws)

for spw in range(nSpws):
  # Central channel.
  cChan=myAtm.getNumChan(spw)//2

  # Dispersive and non-dispersive delays for central channel.
  dd=myAtm.getDispersivePhaseDelay(cChan,spw)['value'][0]
  nd=myAtm.getNonDispersivePhaseDelay(cChan,spw)['value'][0]

  # Set total delay per 1mm PWV as the PF factor for the current SPW.
  # It is a *delay*, so add the minus. Express in radians/mm.
  PF[spw]=-math.pi*(nd+dd)/(pwvMean*180.0)

#--> endfor(spws)
myAtm.done()
```

```
#-- Fill the pwvCalTable with contents.

# Re-open the calTable in the rw mode.
pwvCalTable.open(pwvCalTableName,nomodify=False)
# Set the original table-write offset (start-row for write) to zero.
tabOffset=0

# PWV(t) for the reference antenna - to be subtratced.
pwvRef=pwvMap[refAnt]['dW']

#-- Loop over all antennas.

for antID in pwvMap.keys():

  time=pwvMap[antID]['time']
  pwv=pwvMap[antID]['dW']

  nSamples=len(time)


  #- Create 'static' columns. Some (e.g., interval, observation_id, etc.) are
  #  left in their defult values of zero.

  # The 'scalar columns' are easy - just nSamples-sized arrays.
  field_id=np.zeros(nSamples,dtype=int) # The Sun is field 0. Generalize!!!
  antenna_1=int(antID)*np.ones(nSamples,dtype=int)
  antenna_2=int(refAnt)*np.ones(nSamples,dtype=int)
  scan_num=solScan*np.ones(nSamples,dtype=int)

  # For the 'vector columns' we have to keep shape nPol x nChans x nSamples.
  paramErr=np.zeros((1,1,nSamples))
  flag=np.zeros((1,1,nSamples),dtype=int)
  snr=5.0*np.ones((1,1,nSamples)) # SNR better than 3.0 is fine.

  # Differential PWV between the current and reference antenna.
  pwvDiff=pwv-pwvRef

  #- Loop over SPWs. We are working with split MS, so the sci spws are 0 - 3.
  for spw in range(4):

    #- Columns that depend on SPW

    #SPW
    c_spw=spw*np.ones(nSamples,dtype=int)

    # Complex gains (just the phase) - the CPARAM column.
    cGain=np.exp(1j*PF[spw]*pwvDiff)
    # It is a vector column: Reshape it to nPols x nChans x nSamples.
    cParam=np.reshape(cGain,(1,1,nSamples))

    #- Write all the sub-columns to the calTable, starting at tabOffset.

    # Make space for writing the columns.
    pwvCalTable.addrows(nSamples)

    # Append the columns.
    pwvCalTable.putcol('TIME',time,startrow=tabOffset)
    pwvCalTable.putcol('FIELD_ID',field_id,startrow=tabOffset)
```

```
      pwvCalTable.putcol('SPECTRAL_WINDOW_ID',c_spw,startrow=tabOffset)
      pwvCalTable.putcol('ANTENNA1',antenna_1,startrow=tabOffset)
      pwvCalTable.putcol('ANTENNA2',antenna_2,startrow=tabOffset)
      # INTERVAL defaults to zeros
      pwvCalTable.putcol('SCAN_NUMBER',scan_num,startrow=tabOffset)
      # OBSERVATION_ID defalts to zero.
      pwvCalTable.putcol('CPARAM',cParam,startrow=tabOffset)
      pwvCalTable.putcol('PARAMERR',paramErr,startrow=tabOffset)
      pwvCalTable.putcol('FLAG',flag,startrow=tabOffset)
      pwvCalTable.putcol('SNR',snr,startrow=tabOffset)
      # WEIGHT defaults to an empty array.

      # Update the write offset.
      tabOffset+=nSamples
    #--> endfor(spws)

    print("Finished antenna ",antID)

  #--> endfor(antennas)

  # Sync to SDD/HDD.
  pwvCalTable.flush()

  # Close the tool.
  pwvCalTable.done()


  return



#-----------------------------------------------------------------------------
#                7.2 Direct WVR temperatures vs. dPWV
#-----------------------------------------------------------------------------


def sol_offPos(vis,tgtScan,refAnt):
  """
  Trying to find the minimum off-source separation from the Sun so that
  WVRs are not affected by direct solar illumination.

  We calculate, how far from the Sun (the disc center) we are when pointing
  OFF_SOURCE during the solar scan. This is the position taken at the first and
  last subscan of the solar scan and its purpose is to the basically sky/atm
  measurement, but now with the attenuator settings used for the solar on-source
  observation - see the  G-star calculation in  acd_calTabs()
  It seems that the psoition of the firts sub-scan is identical with those
  taken in the preceding ATM scan.
  """

  #--- Find the time-ranges for the off-source subscans.

  # Attach the msmd tool to our MS.
  myMsmd=msmdtool()
  myMsmd.open(vis)

  # Extract sub-scan information for the solar scan.
  sunSub_info=aU.computeDurationOfScan(tgtScan,mymsmd=myMsmd,
```

```
                                        returnSubscanTimes=True)

# The off-source (i.e., hi-attenuated sky/atm) sub-scans are the first
# and the last. So, we need to find the number of sub-scans. It is the 2nd
# item of the info - see computeDurationOfScan() in sun_reduction_util.py.
nSubScans=sunSub_info[1]

# The off-source scans. Note: Keys in subInfo start with 1.
offSubScan1=1
offSubScan2=nSubScans

# Extract their time-ranges - we select the data later by them.
tRange1=sunSub_info[2][offSubScan1] # Time-range for the leading subscan.
tRange2=sunSub_info[2][offSubScan2] # Time-range for the trailing subscan.

# Time extracted, close the msmd tool.
myMsmd.done();

# Take the time in the middle of the subscans as representative moment.
t1Rep=0.5*(tRange1[0]+tRange1[1])
t2Rep=0.5*(tRange2[0]+tRange2[1])


# Make the representative times as a 1-d, two-element array.
off_times=np.array([t1Rep,t2Rep])


#--- Find (reference) antenna pointings for those time-ranges.

# Open the pointing table
pTable=tbtool()
pTable.open(vis+'/POINTING')

# Select by antenna
selection="ANTENNA_ID=="+str(refAnt)

ptSel=pTable.query(selection,sortlist="TIME",columns="TIME,DIRECTION")

# Extract the column data: Time and pointing direction as Az/El
times=ptSel.getcol("TIME")
pDir=ptSel.getcol("DIRECTION")

# Data extracted, can close the table tool.
pTable.done()


# Decompose pDir to Az and El - we need 1-D array for subsequent np.interp()
az=pDir[0][0]
el=pDir[1][0]

# Now we can interppolate to the off-source times
az_off=np.interp(off_times, times, az)
el_off=np.interp(off_times, times, el)


#--- Find Az/El of the solar disc center at the same OFF times.

az_sc0,el_sc0=aU.ComputeSolarAzEl(mjdsec=t1Rep)
```

```
az_sc1,el_sc1=aU.ComputeSolarAzEl(mjdsec=t2Rep)

"""
--- This is an alternative approach if the above is too inaccurate ---

# MJD secs to MJD
smjd=qa.convert(qa.quantity(t1Rep,'s'),'d')
# This *does* use the JPL Horizon server query.
sunDict=aU.planet(body='sun',mjd=smjd['value'])

+convert solar Ra/Dec to Az/El using the same approach as in
aU.ComputeSolarAzElLatLong(), just submitting our solar Ra/Dec instead of that
found by aU.ComputeSolarRADec(mjdsec), which does NOT query JPL Horizon server
"""

# Make it a 2-element vector to conform the az/el for off positions.
# Re-scale to radioans.
az_sc=np.array([az_sc0,az_sc1])*math.pi/180.0
el_sc=np.array([el_sc0,el_sc1])*math.pi/180.0

# Calculate re-usable radians-to-degs conversion factor.
RAD2DEG=180.0/math.pi

print('OFF position:',az_off*RAD2DEG,el_off*RAD2DEG)
print('SOL position:',az_sc*RAD2DEG,el_sc*RAD2DEG)


#--- Calculate the angular separation

# Unit direction vector to the solar center at beginning and the end.
dirSc1=np.array([math.cos(el_sc[0])*math.cos(az_sc[0]),
                 math.cos(el_sc[0])*math.sin(az_sc[0]),
                 math.sin(el_sc[0])])


dirSc2=np.array([math.cos(el_sc[1])*math.cos(az_sc[1]),
                 math.cos(el_sc[1])*math.sin(az_sc[1]),
                 math.sin(el_sc[1])])


# Unit direction vector to the OFF_SOURCE position at beginning and the end.
dirOff1=np.array([math.cos(el_off[0])*math.cos(az_off[0]),
                  math.cos(el_off[0])*math.sin(az_off[0]),
                  math.sin(el_off[0])])


dirOff2=np.array([math.cos(el_off[1])*math.cos(az_off[1]),
                  math.cos(el_off[1])*math.sin(az_off[1]),
                  math.sin(el_off[1])])


# The dot product of the unit direction vectors is direclty cos(delta)
dRad0=math.acos(np.dot(dirSc1,dirOff1))
dRad1=math.acos(np.dot(dirSc2,dirOff2))

# show the result in degs.
print('Offset beg: ',dRad0*RAD2DEG)
print('Offset end: ',dRad1*RAD2DEG)
```

```
  # Return it in radians.
  return (dRad0,dRad1)

#--------------------------------------------------------------------------

def solTgtPos(vis,tgtScan,refAnt):
  """
  Similar to sol_offPos(), but now we produce a graphic output of the actual
  target-pointing relative to the solar disc center (i.e., angular separation
  of those two) as a function of time over entire *tgtScan* for a 'reference'
  antenna (the result should be the same for all) *refAnt*.
  """

  #--- Find the time-ranges for the entire tgtScan

  # Attach the msmd tool to our MS.
  myMsmd=msmdtool()
  myMsmd.open(vis)

  # Extract sub-scan information for the solar scan.
  sunSub_info=aU.computeDurationOfScan(tgtScan,mymsmd=myMsmd,
                                       returnSubscanTimes=True)

  # The off-source (i.e., hi-attenuated sky/atm) sub-scans are the first
  # and the last. So, we need to find the number of sub-scans. It is the 2nd
  # item of the info - see computeDurationOfScan() in sun_reduction_util.py.
  nSubScans=sunSub_info[1]


  # Extract their time-ranges - we select the data later by them.
  tRangeFirst=sunSub_info[2][1] # Time-range for the leading subscan.
  tRangeLast=sunSub_info[2][nSubScans] # Time-range for the trailing subscan.

  # Subscan time-ranges extracted, close the msmd tool.
  myMsmd.done();

  # The entire scan ranges from beginning of the 1st subscan to the end of
  # the last one.

  tScanBeg=tRangeFirst[0]
  tScanEnd=tRangeLast[1]


  #--- Find (reference) antenna pointings for the full-scan time-range.

  # Open the pointing table
  pTable=tbtool()
  pTable.open(vis+'/POINTING')

  # Select by antenna
  selection="ANTENNA_ID=="+str(refAnt)

  ptSel=pTable.query(selection,sortlist="TIME",columns="TIME,DIRECTION")

  # Extract the column data: Time and pointing direction as Az/El
  times=ptSel.getcol("TIME")
  pDirs=ptSel.getcol("DIRECTION")
```

```
# Data extracted, can close the table tool.
pTable.done()

# Decompose pDir to Az and El
az=pDirs[0][0]
el=pDirs[1][0]


# Now, select the time-range for the scan only.
# Note that '&' is necessary in the condition, 'and' operator does not work!
scanIdxRange=np.where((times >= tScanBeg) & (times <=tScanEnd))

tScan=times[scanIdxRange]
azTgt=az[scanIdxRange]
elTgt=el[scanIdxRange]


#--- Find Az/El of the solar disc center at the same OFF times.

# We do it more exact way than in sol_offPos(). It has been found that
# aU.ComputeSolarAzEl(mjdsec=mjds) looses precision with time if the analysis
# utils are not updated. So, here, we query the JPL server for RaDec at the
# beginning and the end of the scan and do linear interpolation in RaDec.
# Finally, we convert to AzEl for ALMA location.

#- Scan beginning
# MJD secs to MJD
smjd1=qa.convert(qa.quantity(tScanBeg,'s'),'d')
# This *does* use the JPL Horizon server query.
sunDict=aU.planet(body='sun',mjd=smjd1['value'],apparent=True)
RaDecRads1=sunDict['directionRadians']

Ra1=RaDecRads1[0]
Dec1=RaDecRads1[1]

#- Scan end
# MJD secs to MJD
smjd2=qa.convert(qa.quantity(tScanEnd,'s'),'d')
# This *does* use the JPL Horizon server query. 'Apparent' is essential here,
# otherwise we are shifted in AzEl ~10 arcmin w.r.t. the antenna pointings.
sunDict=aU.planet(body='sun',mjd=smjd2['value'],apparent=True)
RaDecRads2=sunDict['directionRadians']

Ra2=RaDecRads2[0]
Dec2=RaDecRads2[1]

#- Interpolate RaDecs

RaSol=np.interp(tScan,[tScanBeg,tScanEnd],[Ra1,Ra2])
DecSol=np.interp(tScan,[tScanBeg,tScanEnd],[Dec1,Dec2])


#- Re-calculate to AzEl for ALMA location. This part has been copied from
#  aU.ComputeSolarAzElLatLong() aU.getObservatoryLatLong()

# Conversion constants.
DEG_TO_RAD = math.pi/180.
```

```
RAD_TO_DEG = 180/ math.pi
HRS_TO_RAD = math.pi/ 12.
RAD_TO_HRS = 1.0/ HRS_TO_RAD

# ALMA location.
[latitude,longitude,obs] = aU.getObservatoryLatLong(observatory='ALMA')


# Use our JPL-based RaDecs instead of the aU.ComputeSolarRADec(mjdsec)
# Scale them to hours/degs
[RA,Dec] =[RAD_TO_HRS*RaSol,RAD_TO_DEG*DecSol]


# We need Local sidereal time (LST). It cannot be computed as an array
# (there are some conditions/comparisons in aU.ComputeLST()), so we do it
# again at the beginning and at the end of the scan and interpolate.

LST1 = aU.ComputeLST(tScanBeg,longitude)
LST2 = aU.ComputeLST(tScanEnd,longitude)
LST=np.interp(tScan,[tScanBeg,tScanEnd],[LST1,LST2])


phi = latitude*DEG_TO_RAD
hourAngle = HRS_TO_RAD*(LST - RA)


# Python apparently cannot calculate trigonometric functions for np.arrays
# at once, so we must continue in a loop :-(

# This is for angular separations between pointing and solar AzEl
angSeps=np.zeros(tScan.shape[0])



# Loop over times in the scan.
ti=0
for t in tScan:

  azSun = math.atan2(math.sin(hourAngle[ti]),
                     (math.cos(hourAngle[ti])*math.sin(phi)
                      - math.tan(Dec[ti]*DEG_TO_RAD)*math.cos(phi)))+math.pi

  arg=(math.sin(phi)*math.sin(Dec[ti]*DEG_TO_RAD)
       +math.cos(phi)*math.cos(Dec[ti]*DEG_TO_RAD)*math.cos(hourAngle[ti]))

  elSun = math.asin(arg);


  #--- Calculate the angular separation


  # Unit direction vector to the solar center
  dirSun=np.array([math.cos(elSun)*math.cos(azSun),
                   math.cos(elSun)*math.sin(azSun),
                   math.sin(elSun)])

  # Unit direction vector to the target
  dirTgt=np.array([math.cos(elTgt[ti])*math.cos(azTgt[ti]),
```

```
                    math.cos(elTgt[ti])*math.sin(azTgt[ti]),
                    math.sin(elTgt[ti])])


      # The dot product of the unit direction vectors is directly cos(delta).
      # Show it in arcmins.
      angSeps[ti]=180.0*60.0/math.pi*math.acos(np.dot(dirSun,dirTgt))

      ti+=1
    #-->endfor(all pointing times in the tgtScan)

    plt.plot(tScan,angSeps)


    return

#-----------------------------------------------------------------------------


def wvr_dataRaw(vis,tgtScan,antID,subScans=''):
    """
    Extract the raw WVR data from SPW 4 for the antenna *antID*

    The *subScans* parameter can be either an empty string (the default), or
    have one of the  "on" or "off" values. If empty, entire scan is used,
    otherwise, just on-source or off-source subscans are involved, respectivelly.

    The four WVR channels (temperatures) are extracted as a function of time.
    Return value: A tupple of two ndarrays - time (1d) as MJDsecs of dimension
    nTimeSamples, WVR (2d) of dimensions 4xnTimeSamples.
    """

    #--- Form the MS selectors: The SPW as data_description_id and the time
    #    ranges for the solar-target off-source sub-scans.

    myMsmd=msmdtool()
    myMsmd.open(vis)

    #- 1. Identify the SPW=4 data (WVR channels) by the data_desc_id
    wvrDataDescID=int(myMsmd.datadescids(spw=4)[0])

    print('Data desc ID for SWP 4 = ',wvrDataDescID)

    #- 2. Identify the on-source / off-source subscans by their time range.

    # Get sub-scan info for our tgtScan.
    sub_info=aU.computeDurationOfScan(tgtScan,mymsmd=myMsmd,
                                      returnSubscanTimes=True)
    # How many sub-scans do we have?
    nSubScans=sub_info[1]

    # The off-source subscans are just the first and last one. The on-source
    # are in the middle.
    # Note: This is a key to the map/dictionary of subscans and it starts at 1.
    offSubScans=[1,nSubScans]
    onSubScans=range(2,nSubScans)

    # Make and fill a list of time-ranges (as MJDs) for the selected sub-scans.
```

```
# We use the string representation of date-time - that is why sub_info[3] is
# extracted here. This is suitable for ms::msselect() used subsequently -
# see the comments below.
tRanges=[]

if (subScans=='on'):
  for oss in onSubScans:
    tRanges.append(sub_info[3][oss])
elif (subScans=='off'):
  for offs in offSubScans:
    tRanges.append(sub_info[3][offs])


# Make the string-list to be a single comma-separated string.
trString=','.join(tRanges)

# We are done with the time-range extraction.
myMsmd.done()


#--- Use the above identifiers/selectors for filterring the WVR data from MS.

myMs=mstool()
myMs.open(vis)

# Initialize the MS selector by SPW=4
myMs.selectinit(datadescid=wvrDataDescID)

# Select WVRs for a given antenna and the scan number.
myMs.select({'antenna1':antID,'antenna2':antID,'scan_number':tgtScan})


# On demand, refine selection to the sub-scans defined by the time-range.
# Note: We use ms::msselect() (and not ms::select()) here as it allows
# for non-contiguous union of time intervals for the sub-scans
if (trString): myMs.msselect({'time':trString})


#--- Filtration done, now pick-up and process the data.

# Extract the (complex) data and their 'coordinates' from the filtered MS.
wvrData=myMs.getdata(["data","axis_info"],ifraxis=True)

# Clean-up the tool MS tool.
myMs.reset()
myMs.close()


# Get the data - they are stored as complex amplitudes.
wvr_cAmp=wvrData["data"]

# Get their corresponding times as MJD.
wvr_mjd=wvrData["axis_info"]["time_axis"]["MJDseconds"]

# The data are complex by nature, but in fact having just the real part.
wvrPow=wvr_cAmp.real

# The 'ifraxis=True' in the ms.getdata() above causes expansion of the antenna
```

```
  # axis. In our case of WVRs (spw=4) it is degenerated to dimension=1.
  # Let us remove it (its is the 3rd axis of the array).
  wvr_pow=np.squeeze(wvrPow,axis=2)

  # The 'polarization' axis is degenerated as well for WVRs, let us get rid of
  # this dimension, too,
  wvr_pow=np.squeeze(wvr_pow,axis=0)

  print('WVR pow dims: ',wvr_pow.shape)

  # Return the time axis as MJD.
  # Return the power as a numpy array of nChans x nIntegs dims.
  return (wvr_mjd,wvr_pow)


#--------------------------------------------------------------------------

def pwvCalc(vis,atmTool,scan,antID):
  """
  Calculate PWV for the *scan* and *antID* using raw WVR data. We use
  linearized formula (13) in the PR15 in an iterative way to converge
  to dW=w-w0 to be zero - basically we use the Newton-Raphson (N-R) method. The
  atmosphere tool *atmTool* must be set beforehand via setAtmTool() for SPW 4.
  The sky brightness temperature Tbs is calculated from the ATM model using
  at.getTebbSky() method. We have also tried at.getTrjSky() as an alternative
  and compared the results with the (rare) SYSCAL PWVS via pwvCalcVerify() -
  see the comment on comparison results there.

  Returns: PWV obtained by averaging the partial results from WVR channels 1-3
  and its stddev in a tupple. We skip channel 0 (the line core) as it gives
  outlied data from the rest (frequently also NaNs). Essentially, for the
  'solar observing conditions' with PWV typically > 3.0 it is, anyway, mostly
  saturated and, hence, insensitive to smaller PWV variations.
  """

  #-- Get the raw WVR temperatures averaged over the scan.
  time,wvrs=wvr_dataRaw(vis,scan,antID)
  Twvr=np.mean(wvrs,axis=1)

  # Count of WVR cahnnels
  wvrChans=wvrs.shape[0]

  #-- Iteration setup.

  # Half dw for the slope (tau_1eff) estimation.
  dws=0.05
  # N-R iteration stopper: WVR temperature is closer to Tbs than this.
  dT_min=0.001
  # Calculate limiting TbSky for infinite PWV: It is a constant close to
  # the tropospheric temperature for ALMA array (MeteoCentral).
  TbsInf=atmTool.getTebbSky(spwid=0,nc=0,
                            wh2o=qa.quantity(10000.0,'mm'))['value'][0]

  # Array for the results.
  pwvArr=np.zeros(wvrChans)

  #-- Newton-Raphson iterations: Separetelly per each WVR channel, as they
  #   need not to converge synchronously. We skip the channel 0 (WV line core)
```

```
  #   as it gives outlaid value from the rest channels 1 - 3.

  for wC in range(wvrChans):

    # Initial estimate for PWV. 3.0 mm and more is typical for the solar obs.
    w0=3.0
    # Initial dT per channel (arbitrary number greater then dw_min).
    dT=0.5
    # Iteration count.
    itc=0

    while(abs(dT)>dT_min):
      # Calculate the TbSky for current estimate of PWV
      TbsW0=atmTool.getAverageTebbSky(wC,qa.quantity(w0,'mm'))['value'][0]


      # Calculate the slope by stepping a bit forward and backward in PWV
      TbsW0H=atmTool.getAverageTebbSky(wC,qa.quantity(w0+dws,'mm'))['value'][0]
      TbsW0L=atmTool.getAverageTebbSky(wC,qa.quantity(w0-dws,'mm'))['value'][0]


      # The slope has a meaning of effective tau_1.
      # Its values are actually close to tau_1 for correct PWV.
      tau_1eff=-np.log((TbsW0H-TbsInf)/(TbsW0L-TbsInf))/(2.0*dws)

      # How far we are to the solution?
      dT=Twvr[wC]-TbsW0

      # Calculate the shift towards the solution.
      dW=dT/(tau_1eff*(TbsInf-TbsW0))
      # Update the current PWV value
      w0+=dW

      itc+=1

    #--> endwhile(N-R iterations)

    # Insert the converged value to shrinked (skip chan 0) array of results.
    pwvArr[wC]=w0

  #--> endfor(WVR channels)

  print(pwvArr)

  # Use just WVR channels 1 - 3 for reasons described above.
  pwvMean=pwvArr[1:].mean()
  pwvStd=pwvArr[1:].std()

  print("PWV: ",pwvMean,"+/-",pwvStd)

  return (pwvMean,pwvStd)


#--------------------------------------------------------------------------

def pwvCalcVerify():
  """
  Try to 'verify' our PWV calculation in pwvCalc() against the ALMA-calculated
```

```
values in ASDM_CALWVR table, accessed by getPWVforAnt().

*** Specific for this MS! ***

We calculate the PWVs by both methods for all antennas and all non-solar scans
and make a scatter plot. We can expect some differences as our pwvCalc()
method uses WVR temperatures ove a scan, while the system-calculated PWVs
are a bit opaque - neither the method of their calculation is clear (nothing
found in literature except they use the ATM model) nor the strange times at
which we have their values in ASDM_CALWVR table are explained. The goal is
to see, whether the dicscrepancies are small and do not have systematic
trends.

Results: The correclation works basically well, but there are, indeed,
systematic differences: If at.getTebbSky() is used for sky brightness
estimation (to be interpretted as Twvr values in WVR raw data), our pwvCalc()
method gives systematically underestimated PWV values. When at.getTrjSky() is
used, then our PWVs are systematically over-estimated. Detailed
results are in ./wvr_raw/pwv-SYS_vs_ours directory.
"""

# Arrays for PWVs calculated by our method pwvCalc() and those in ASDM_CALWVR.
ourPWVs=np.array([])
sysPWVs=np.array([])

# MS definition.
vis='uid___A002_X110fd13_X1099e.ms'

# Define the scans: All non-solar
theScans=[1,2,3,4,5,6,7,8,10]

# Get antenna list.
myMsmd=msmdtool()
myMsmd.open(vis)
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()
myMsmd.done()


#-- Loop over all selected scans.
for scan in theScans:

  # Set the ATM model for the scan.
  myAtm=attool()
  setAtmTool(myAtm,'uid___A002_X110fd13_X1099e.ms',4,scan)

  #-- Loop over all antennas.
  for myAnt in antIDs:

    # Skip the CM antennas, these have no WVRs.
    if (antNames[myAnt][:2]=='CM'):
      print("Skipping antenna ",antNames[myAnt]," - no WVR data.")
      continue

    # Report progress for WVR-equiped antennas.
    print('\n============================================================')
    print('Calculating scan ',scan,' for antenna ',
          int(myAnt),'<-->',antNames[myAnt])
```

```
      print('============================================================\n')

      # Calculate PWV by our method; the 2nd item in a tupple is stddev.
      myPWV,myPWVerr=pwvCalc(vis,myAtm,scan,int(myAnt))
      # Append it to the list of all PWV values.
      ourPWVs=np.append(ourPWVs,myPWV)

      # Extract the PWV from SYS caltable ASDM_CALWWR (+ interpolation to the
      # scan mid-time)
      sysPWV=getPWVforAnt(vis,scan,int(myAnt))['pwvAvg']
      # Append it to the list of all sys-PWV values.
      sysPWVs=np.append(sysPWVs,sysPWV)

      # A quick check.
      print('--- ',myPWV,sysPWV)

    #--> endfor(antennas)

    # Close the atmosphere tool.
    myAtm.done()

  #--> endfor(scans)

  # Return the PWV collections for further analysis.
  return (ourPWVs,sysPWVs)


#-------------------------------------------------------------------------------

def dWdLs(antID,myAtm,fullPWV=False):
  """

  *** Specific for 2022.1.00031CSV project! ***


  Decompose the WVR raw variations for antenna *antID* in the solar scan into
  PWV variations and varying illumination of the WVRs by scaterred solar
  irradiance Lsun. These two should be separable because we have 4 WVR channels
  (three of them usable, one saturated) and dPWV and dLsun have different
  frequency dependence - PWV-related signal increases towards the WV line center
  while the solar illumination (thinned by atmospheric absorption) decreases.
  The atmosphere tool *myAtm* must be set beforehand using setAtmTool() for
  the scan no 9 (the solar scan) and SPW 4 (WVR data).
  The boolean switch *fullPWV* controls, whether just variations around the
  means shall come out for the extracted PWV data (the default is false, which
  is usefull for comparison with dPWV proxy) or full PWV value with the mean
  component added should come out. The Lsun contribution is always returned
  including its mean.

  Reasoning: By comparison between our dPWV data and raw WVR signal it has been
  found that the DV antennas exhibit, in general, excellent correlation between
  those two quantities (WVR raw data and dPWV from science SPWs), while the DA
  antennas are obviously much worse. It looks like the reason is to be found
  in a different antenna construction - namely supports of the sub-reflector.
  These obstacles may lead to shadowing or, vice versa, additional reflection
  of the solar illumination to the WVR. Hence, the solar illumination needs
  to be handled as variable (cannot be fully subtracted in the stationary part).
```

```
We use formula (16) in PR15 to make the decomposition. Alhorithm for numerical
solution of Eq. (16) is the following:

1. Make scan-averaged (may be decomposed to off-source and on-source subscans)
WVR channel temperatures <Twvr>. Since dW(t) and dL(t) variations should
have (by definition) zero time average, three equations

<Twvr>=T_BS(w) + <Lsun>*tr(w)

should hold for WVR channels  1 - 3. We neglect the WVR channel 0, as it is
saturated, exhibiting minimum variations (even the off-source -> on-source
transition and back is not visible in the data). We vary the PWV content w
such a way, that dispersion of three results for <Lsun> is minimized.
The sky brigtness T_BS(w) is approximated by at.getTebbSkySpec() using the
*myAtm* model of the atmosphere, the transmisivity tr(w) is calculated from
the same model. Newton-Raphson (N-R) iterations are used for minimizing the
<Lsun> dispersion over the WVR channels.

2. Once optimized <PWV> and <Lsun> are found, we calculate the matrix
coefficients and the RHS for the overdetermined system of equations
(three equations for 3 usable WVR channels and 2 variables - dW and dLsun)
and use the standard quadratic-form miminizing to find the solution
(see the same procedure in dWdB() for details).
The solutions dW(t) and dLsun(t) are checked for zero time-average: If
a non-negligible value is found, the mean PWV as found in Step 1 is
corected

<PWV>_new=<PWV>_old + <dW(t)>

and likewise the <Lsun>, and Step 2 is repeated until near-zero avarages are
reached.
"""
#-- Static parameters

#- Solar spectrum - in theory it is R-J.

# Reference frequency for the solar spectrum - central SPW 4 frequency (in Hz)
f_wvr=1.8755e11

# Spectral index for the source. Spix=2.0 corresponds to R-J law.
spix=2.0

#- Source-data definition.
vis='uid___A002_X110fd13_X1099e.ms'
solScan=9


#- Atmosphere static parameters.

# Calculate limiting TbSky for infinite PWV: It is a constant close to
# the tropospheric temperature for the ALMA array (MeteoCentral).
TbsInf=myAtm.getTebbSky(spwid=0,nc=0,
                        wh2o=qa.quantity(10000.0,'mm'))['value'][0]

# Initial guess of PWV: Scan-center interpolated ASDM_CALWVR value.
w0=getPWVforAnt(vis,solScan,antID)['pwvAvg']

print('....... Initial estimate')
```

```
print('PWV=',w0)
print('.......')


# Airmass for the solar scan.
airMass=myAtm.getAirMass()


#-- Extract WVR raw temperatures.

# Scan-averaged WVR Tbs
wTime,Twvr=wvr_dataRaw(vis,solScan,antID,'on') # on,off or all
# Skip the saturated WVR channel 0.
Twvr=Twvr[1:,:]
# Average over a scan.
TwvrMean=np.mean(Twvr,axis=1)

# How many time samples are in this scan?
nSamples=Twvr.shape[1]

# How many ATM spws  do we have?
nAtmSPWs=myAtm.getNumSpectralWindows()

#- Reconstruct the sub-channel frequencies for our ATM SPWs.
#  We need them for proper scaling of the solar-illumantion spectrum.

# We have the same number of sub-channels for all WVR-channel ATM SPWs.
# Fortunatelly, because Python is incapable of creating a vector of vectors,
# which would have different sizes, only rectangular matrix is OK :-(
nChans=myAtm.getNumChan(0)


# Allocate the per-SPW arrays.

# Prepare arrays for brightness temperature, opacities, etc.
TbsW0=np.zeros(nAtmSPWs)
tau_1=np.zeros(nAtmSPWs)
tau_1eff=np.zeros(nAtmSPWs)
trans=np.zeros(nAtmSPWs)

# Sensitivity factors for dW and dL
SW=np.zeros(nAtmSPWs)
SL=np.zeros(nAtmSPWs)

# Frequencies for sub-channels in our ATM SPWs.
wvrFreqs=np.zeros((nAtmSPWs,nChans))

for wC in range(nAtmSPWs):

    fMin=myAtm.getMinFreq(wC)['value'][0]*1.0e9 # GHz to Hz
    df=myAtm.getChanSep(wC)['value'][0]*1.0e6   # MHz to Hz

    wvrFreqs[wC]=fMin+df*np.arange(nChans)


#-- Step 1: Optimize <PWV> by finding minimum dispersion of <Lsun>

#-General N-R iteration setup for Step 1.
```

```
# Half dw for the estimation of derivatives in the N-R iterations.
dws=0.05
# N-R iteration stopper: We move in PWV space by less than this,
ddW_min=0.01
# Initial displacement in the PWV space. Anything above ddW_min.
ddW=0.5


while (abs(ddW)>ddW_min):

  # Set current estimation of PWV to the ATM model.
  myAtm.setUserWH2O(qa.quantity(w0,'mm'))

  #-- Loop over ATM spws <--> all WVR channels
  for wC in range(nAtmSPWs):

    #- Sky brightness for current PWV estimate and its logarithmic slope.
    TbsW0[wC]=myAtm.getAverageTebbSky(wC,qa.quantity(w0,'mm'))['value'][0]


    #- Tbs logarithmic slope w.r.t. changing PWV.
    TbsW0H=myAtm.getAverageTebbSky(wC,qa.quantity(w0+dws,'mm'))['value'][0]
    TbsW0L=myAtm.getAverageTebbSky(wC,qa.quantity(w0-dws,'mm'))['value'][0]
    # The slope has a meaning of effective tau_1.
    # Its values are actually close to tau_1 scaled my airmass for correct PWV
    tau_1eff[wC]=-np.log((TbsW0H-TbsInf)/(TbsW0L-TbsInf))/(2.0*dws)

    #- Transmissivity & tau_1

    # Dry zenith opacity - an array per sub-channel.
    tau_dry = np.array(myAtm.getDryOpacitySpec(wC)[1])
    # Water-vapour related 'wet' opacity - an array per sub-channel.
    tau_wet = np.array(myAtm.getWetOpacitySpec(wC)[1]['value'])


    # Wet opacity per 1mm PWV scaled by airmass - mean per SPW.
    tau_1[wC]=airMass*tau_wet.mean()/w0

    # Total opacity - an array per sub-channel.
    opacity=airMass*(tau_wet+tau_dry)

    # Mean transimissivity per SPW, scaled by the solar spectrum
    trans[wC]=np.mean(((wvrFreqs[wC]/f_wvr)**spix)*np.exp(-opacity))

  #-->endfor(ATM SPWs)

  #- Skip the WVR channel 0 - it is saturated. The _S arrays are _S_horten
  TbsW0_S=TbsW0[1:]
  tau_1_S=tau_1[1:]
  tau_1eff_S=tau_1eff[1:]
  trans_S=trans[1:]

  #- Calculate <Lsun> in each WVR channel and its dispersion (stddev).

  # Mean solar illumination <Lsun>=<r*Tsun> per WVR channel.
  Lsun=(TwvrMean-TbsW0_S)/trans_S
```

```
    # Calculate slope dLsun/dw
    dLdW=-tau_1eff_S*(TbsInf-TbsW0_S)/trans_S+Lsun*tau_1_S

    # Estimate the <Lsun> (a.k.a <r*Tsun>) in case of w0 displaced by +/- dws.
    LsunPlus=Lsun+dLdW*dws
    LsunMinus=Lsun-dLdW*dws

    # Calculate the solar-illumination temperatures dispersions over WVR
    # channels: For current w0 and w0 displaced by +/- dws.
    Td0=Lsun.std()
    TdP=LsunPlus.std()
    TdM=LsunMinus.std()

    # We expect the temperature dispersion is quadratic in w around its minimum.
    # In such case, the distance to its minimum can be calculated as -(y'/y")
    # We calculate y' as a difference, y" is thresholded by experimentally found
    # constant below. Why? The quadratic approximation is actually valid just
    # close to the minimum (PWV +/- 0.2  from its optimum value) as found
    # by an experiment(Fig. Tsun_disp_vs_w.png), in the larger distances from
    # the minimising PWV it is rather linear, with y" going to zero - see Fig.12
    # in PR15, namely the left 'shoulder'.

    # First central difference as a proxy to y'.
    Td1=(TdP-TdM)/(2.0*dws)
    # Second central difference as a proxy to y".
    Td2=(TdP+TdM-2.0*Td0)/(dws*dws)
    # Thresholding for the linear 'shoulder' of Tdisp(w)
    if (Td2 < 50.0): Td2=147.5

    # Calculate the displacement towards the optimum.
    ddW=-Td1/Td2

    # Update PWV.
    w0+=ddW

#--> endwhile(N-R iterations for Step 1)

# Static part (scan-average) of solar illumination averaged over WVR channels.
LsunMean=Lsun.mean()

print('------- After Step 1')
print('PWV=',w0,'L_sun=',LsunMean)
print('LS spec:',Lsun,'LS disp:',Td0)
print('-------')


#-- Step 2: Solve the linear set of equations for dW and dL variations
#   Iterate until they have negligible mean value (scan-average).

#- Iteration setup

# Stoppers for iterative solution: We should get less then these values.
dwMean_max=0.01
dlMean_max=0.01

# Initial values for the dW and dL means - anyting bigger then the stoppers.
dWtMean=0.5
dLtMean=1.0
```

```
#- Iterations: Minimize the mean value of variations.
while((abs(dWtMean) > dwMean_max) or (abs(dLtMean) > dlMean_max)):

  #- Recalculate ATM model with updated <PWV> and <Lsun>.

  # Set current estimation of PWV to the ATM model.
  myAtm.setUserWH2O(qa.quantity(w0,'mm'))


  #-- Loop over ATM spws <--> all WVR channels
  for wC in range(nAtmSPWs):

    #-- All quantities per WVR-channel, spectrally resolved by sub-channels.
    #   The _C values are _C_hannelised (spectral). We do not specifically
    #   indicate this in case of variables local withon the loop.

    #- Sky brightness for current PWV estimate and its logarithmic slope.
    TbsW0_C=myAtm.getTebbSkySpec(wC,qa.quantity(w0,'mm'))[1]['value']

    TbsW0H=myAtm.getTebbSkySpec(wC,qa.quantity(w0+dws,'mm'))[1]['value']
    TbsW0L=myAtm.getTebbSkySpec(wC,qa.quantity(w0-dws,'mm'))[1]['value']
    # The slope has a meaning of effective tau_1, see above.
    tau_1eff_C=-np.log((TbsW0H-TbsInf)/(TbsW0L-TbsInf))/(2.0*dws)

    #- Transmissivity & tau_1

    # Dry zenith opacity
    tau_dry = np.array(myAtm.getDryOpacitySpec(wC)[1])
    # Water-vapour related 'wet' opacity
    tau_wet = np.array(myAtm.getWetOpacitySpec(wC)[1]['value'])

    # Wet opacity per 1mm PWV scaled by airmass.
    tau_1_C=airMass*tau_wet/w0

    # Total opacity (spectral).
    opacity=airMass*(tau_wet+tau_dry)

    # Transmissivity scaled by solar spectrum
    trans_C=((wvrFreqs[wC]/f_wvr)**spix)*np.exp(-opacity)

    #-- Making the SPW averages for the ATM model quantities.

    # Mean sky-brightness temperature for the current PWV=w0.
    TbsW0[wC]=TbsW0_C.mean()
    trans[wC]=trans_C.mean()
    tau_1[wC]=tau_1_C.mean()
    tau_1eff[wC]=tau_1eff_C.mean()

    # Sensitivity factor for PWV variations.
    SW[wC]=np.mean(tau_1eff_C*(TbsInf-TbsW0_C)-LsunMean*trans_C*tau_1_C)
    # Sensitivity factor for solar illumination.
    SL[wC]=trans[wC]

  #-->endfor(wvr channel)

  #- Skip the WVR channel 0 - it is saturated.
  TbsW0_S=TbsW0[1:]
```

```
tau_1_S=tau_1[1:]
tau_1eff_S=tau_1eff[1:]
trans_S=trans[1:]
SW_S=SW[1:]
SL_S=SL[1:]


#- Calculate <Lsun> in each WVR channel and its dispersion (stddev).
#  Now just for inspection.

# Scan-averaged solar illumination <Lsun> per WVR channel.
Lsun=(TwvrMean-TbsW0_S)/trans_S
# Its dispersion over the 3 used WVR channels
Td0=Lsun.std()

#- Calculate coefficients of the system in line with Eq. (16) in PR 15.


# RHS: The Twvr with the stationary (scan-averaged) part subtracted.
# Twvr is a matrix nTimeSamples x 3 (=nChans), while the subtracted static
# part is just a 3-vector. Make both to be the same shape using tensor dot.

# A 'unity'-vector in time dimension.
uTime=np.ones(nSamples)

TwvrStatic=TbsW0_S+LsunMean*trans_S

dTwvr=Twvr-np.tensordot(TwvrStatic,uTime,0)
# An alternative setup
#dTwvr=Twvr-np.tensordot(TwvrMean,uTime,0)


#- Set of linear equations for dW and dB: Assembly the system.

# Set the 3-vector of sensitivity factors as rows in the transposed matrix.
AwlT=np.array([SW_S,SL_S])
# Transpose to get the 'direct' matrix with sensitivity factors as columns.
Awl=np.transpose(AwlT)

# Set the WVR variations directly as the RHS vector.
Cwl=dTwvr

#- The system is overdetermined, make the least-square minimizer

# Quadratic-form matrix 2x2 (we have two variables).
AK=np.matmul(AwlT,Awl)

# Modified RHS (nTimeSamples x 2)
CK=np.matmul(AwlT,Cwl)

#- Solve the nVarsx x nVars set of equations, using Cramer's rule.

A11=AK[0,0]
A12=AK[0,1]

A21=AK[1,0]
A22=AK[1,1]
```

```
    C1=CK[0]
    C2=CK[1]

    iD=1.0/(A11*A22-A12*A21) # Inverse determinant of the system.

    # Time series of the dPWV variations and dLsol variations as a result.
    dWt=iD*(A22*C1-A12*C2)
    dLt=iD*(A11*C2-A21*C1)

    # Calculate the time-averages of the above variations - these control the
    # iteration stop.
    dWtMean=dWt.mean()
    dLtMean=dLt.mean()

    print('dW mean: ', dWtMean, 'dLsun mean:', dLtMean)

    # Update the static <PWV> and <Lsun>
    w0+=dWtMean
    LsunMean+=dLtMean

  #--> endwhile(non-zero-means)

  print('======= After Step 2')
  print('PWV=',w0,'L_sun=',LsunMean)
  print('LS spec:',Lsun,'LS spec_mean +/- disp:',Lsun.mean(),' +/- ',Td0)
  print('=======')

  #-- Prepared data for export.

  # For Lsun add the mean value to the variations always
  LsunFull=LsunMean+dLt

  # For PWV add the mean just on request.
  if (fullPWV): dWt+=w0


  # Return the PWV and Lsun variations in time as a tupple.
  return (wTime,dWt,LsunFull)



#------------------------------------------------------------------------------

def dWdLs2(myAtm,antID,tgtScan,
           flaggedWVRchannels=[],ssType='',solIllumination=True,
           myMsmd=None,acdLoads=None):

  """

  *** Specific for 2022.1.00031CSV project! ***


  An updated version of dWdLs(), which is improved by the following features:

  - Included antenna efficiencies.
  - Improved solar spectrum as in dWdB_Tbs()/dWdBdG().
  - Improved solution of the system using Jacobians as in dWdB_Tbs()/dWdBdG().
```

```
Essentially, we may use the same basic equation for Tsrc+sky as we did for
sci SPWs in dWdB_Tbs() - the scattered solar radiation (both forward or rear)
contributes always by term of the form r_eff*tr(w(f))*Tsun(f), where r_eff is
the fraction of solar illumination effectivelly scatterred into WVR detector
and this factor already contains the antenna efficiencies. Since r_eff is
time dependent and unkonwn, we calculate just the product (r_eff(t)*Tsun0(t)).
Here, we absorb r_eff into the solar temperature and we designate the entire
that product simply by Tsun0(t). As in dWdBdG() we write Tsun(f,t) by
means of the normalised spectrum as T(f,t)=Tsun0(t)*SF(f). Hence, differences
to dWdBdG() are not so big and they are contained in the following:

- We do not have instrumental vars (gains/Pinst), the normalised spectral
gains are set to be uniform and equal to 1.0.

- There is no polarization separation as the WVR data are just Stokes I.

- We have only 4 WVR channels modelled by 4 ATM spws: No data->model SPW map.

- Tsun0(t), representing just the part of solar radiation scaterred to WVR,
is typically 50 - 100 K only.

As explained in dWdLs(), this procedure is bound to the specific MS, where
we had good luck and the WVRs are not saturated thanks to close-to-limb
target observation.
"""

#--- Static parameters

#- Dirs

# Source-data definition. Set static because we are MS-specific routine.
vis='uid___A002_X110fd13_X1099e.ms'
# ATM scan preceding the solar scan 9 is used always.
atmScan=8

# Output/diagnostics plots.
plotDir='./wvr_decomp'


#- Tools, global data

# Flags used to check, whether we should close the tool later.
ownMsmd=False

if (not myMsmd):
  # Open the msmd tool.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  ownMsmd=True

# Create our dictionary with ACD load temperatures (ambient, hot) if needed.
if (not acdLoads):
  acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                  warnIfNoLoadTemperatures = True,
                                  mymsmd=myMsmd)

#- Target
```

```
# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.3  # kiloKelvins!!!

# Solar radius.
Rsun=696000 # km

# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.

# Apparent angular solar radius in arcsec.
r_sol=180.0*3600.0*Rsun/ (Dsun*np.pi)

# mu=cos(theta). Used for limb-brightenning correction below.
mu=np.sqrt(1.0- (x_h**2+y_h**2)/ r_sol**2)

# b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
# in Alissandrakis et al. (2020). Scaled to kiloKelvins!
b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

# Brightness temperature corrected to limb brightenning - see
# Alissandrakis et al. (2020).
Tsun00=TsunCenter+0.5*b_2*np.log(mu) # kK

# Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
f_LO=1.98e11

#- Antenna

# Rear-spillover + ohmic antenna efficiency (see Tab. 3 in White et al. 2017).
eta_l=0.95
# Forward-spillover antenna efficiency (Tab. 3 in White et al. 2017).
# WVR has, unlike the sci-receiver, on-axis recording, may have better eta_F.
eta_F=0.9

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']

#- Atmosphere

# Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
```

```
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,'mm @airMass = ',airMass)
print('Estimated T_B(full Sun) = ',1000.0*Tsun00,'K') # scale to Kelvins.
print('.......')


# The MS metadata has been read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()


#-- Extract WVR raw temperatures.

# Raw WVR Tbs
wTime,Twvr=wvr_dataRaw(vis,tgtScan,antID,ssType)

# Average over time.
TwvrMean=np.mean(Twvr,axis=1)

# How many time samples are in this scan?
nSamples=Twvr.shape[1]

# How many ATM (i.e. in the model) spws do we have? In this case it is
# the same  as a number of WVR channels.
nAtmSPWs=myAtm.getNumSpectralWindows()

# How many equations do we actually have per block of 1 time-sample?
# Some WVR channels may be flagged (typically, the channel 0 is saturated).
eqBlockSize=nAtmSPWs-len(flaggedWVRchannels)


#- Reconstruct the sub-channel frequencies for our ATM SPWs.
#  We need them for proper scaling of the solar-illumantion spectrum.

# We have the same number of sub-channels for all WVR-channel ATM SPWs.
# Fortunatelly, because Python is incapable of creating a vector of vectors,
# which would have different sizes, only rectangular matrix is OK :-(
nChans=myAtm.getNumChan(0)

# Frequencies for sub-channels in our ATM SPWs.
wvrFreqs=np.zeros((nAtmSPWs,nChans))

for wC in range(nAtmSPWs):

    fMin=myAtm.getMinFreq(wC)['value'][0]*1.0e9 # GHz to Hz
    df=myAtm.getChanSep(wC)['value'][0]*1.0e6   # MHz to Hz

    wvrFreqs[wC]=fMin+df*np.arange(nChans)

#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).
#   Note: In this case, when we have 1-to-1 correspondence between the model
#   and data SPWs, we actually need not to store the per-spw values to arrays.
#   But we keep the structure of sci-SPW approach in dWdB_TBs() and dWdBdG().

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
```

```
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,nChans))
dTdW=np.zeros((nAtmSPWs,nChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,nChans))
tau_1=np.zeros((nAtmSPWs,nChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/Tsun00*np.log(wvrFreqs/f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin

# Initial values for the dW and dL means - anyting bigger then the stoppers.
dW_mean=0.5
dB_mean=0.5

# Iteration counter.
itCount=0

# Set the starting values for iterations.
w0=pwv0
# From dWdLs() we estimate the solar contamination ~100K = 0.1kK in the
# on-source subscans. The off-source shoud be contamination-free.
if (ssType=='on'):
  Tsun0=0.1 # kiloKelvins
else:
  Tsun0=0.0 # no illumination

#-- Iterations: Minimize the mean value of variations.

while(((abs(dW_mean) > dwMean_max) or (abs(dB_mean) > dbMean_max))
      and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample

  # Our Jacobian contains just N-blocks for [dW,dT] 'natural' variables.
  JJ=np.zeros((eqBlockSize,2)) # 2 vars repeating in non-flagged wvrChans
  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))

  # Solution residua for each used WVR channel.
  resT={}
  # Mapping the WVR channel to the index of equation in the system.
  spw2row={}
```

```
#- Re-calculate ATM model with updated <PWV>

# Set current estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

# Count the actual number of equations - some WVR channels may be flagged.
eqCount=0

#- Loop over 4 ATM spws - these directly correspond to 4 WVR channels
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Scale to kiloKelvins.
  TbsW0[atmSPW]=0.001*Tebb

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value']
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value']

  # In kiloKelvins/mm
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Transmissivity (spectral).
  trans[atmSPW]=np.exp(-opacity)
  # Wet opacity per 1mm PWV scaled by airmass.
  tau_1[atmSPW]=(airMass/w0)*tau_wet

  #- Model (_M) factors.

  # Tebb and its slope.
  TbsW0_M=TbsW0[atmSPW]
  dTdW_M=dTdW[atmSPW]

  # Transmissivity and unit-PWV opacity.
  trans_M=trans[atmSPW]
  tau_1M=tau_1[atmSPW]

  # Solar-flux model spectrum (normalized).
  SF_M=SF[atmSPW]

  #- Model + antenna efficiencies combined.

  # Sensitivity factor for PWV.
  SW=(-eta_l*eta_F*Tsun0*np.mean(trans_M*tau_1M*SF_M)
```

```
        +(1.0-eta_F+eta_F*eta_l)*np.mean(dTdW_M))

  # Sensitivity factor for the (apparent) solar illumination temperature.
  SB=eta_l*eta_F*np.mean(trans_M*SF_M)

  # Model T_sky as viewed by the receiver with uniform spectral gain.
  Tsky=(eta_l*eta_F*Tsun0*np.mean(trans_M*SF_M)
        +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M)
        +eta_F*(1.0-eta_l)*Tamb)


  #- Build the equations only for the non-flagged channels.
  if (atmSPW not in flaggedWVRchannels):

    #- Build the matrix: Only the natural (dW & dB sensitivities) block.

    # Set the sensitivity factors for the current ATM SPW as a row
    # in the block/tile for the linearized-system Jacobian matrix.
    JJ[eqCount]=np.array([SW,SB])

    #- Build the RHS vector: Target data

    # RHS: Twvr(t) with the stationary (in the working point) ATM-based part
    # subtracted. Rescale Twvr to kiloKelvin.
    dTw=0.001*Twvr[atmSPW]-Tsky

    # Set it as the RHS component.
    CC[eqCount]=dTw

    # Residua of sky temperature for mean PWV and Tsun (operation point).
    resT[atmSPW]=dTw.mean()

    # Mapping between ATM SPW and equation number.
    spw2row[atmSPW]=eqCount
    eqCount+=1

  else: print('Ignoring flagged ',atmSPW,' WVR channel.')
  #--> endif(not flaggged)

#--> endfor(ATM SPWs)

# Now, the eqCount should be the same as eqBlockSize. Just check it.
if (eqCount != eqBlockSize):
  print("******* Ooops, something is wrong !*******")
  print("   The expected number of equations does not correspond to the")
  print("   number of measured and non-flagged SPWs/polarization channels.")
  return False


#-- Solution of the system of linearised equations.

# Transposed matrices.
JJ_T=np.transpose(JJ)

# Quadratic-form matrix 2x2 (we have two variables).
J_KK=np.matmul(JJ_T,JJ)

# Modified RHS (nVars x nTimeSamples)
```

```
    C_KK=np.matmul(JJ_T,CC)


    if (solIllumination):
      # We allow for variable solar illumination and need to solve it, too.
      # Solve the nVarsx x nVars set of so called normal equations.
      # C_KK represents the nTimeSamples column vectors, the solution is
      # of the form 2 x nTimeSamples.
      varVec=np.linalg.solve(J_KK,C_KK)

    else:
      # The off-subscans are 2 degs of the Sun, the illumination should be zero.
      C_KK[1]=0.0
      varVec=C_KK/ J_KK[0,0]
    #--> endif(on/off subscan)

    # Decompose the solution vector to our components dW, dBxx, and dByy.

    dW=varVec[0]
    dB=varVec[1]

    #- All variations should tend to zero means.

    dW_mean=dW.mean()
    dB_mean=dB.mean()

    # Update stationary part of PWV and solar T_B  by means of their variations.
    w0+=dW_mean
    Tsun0+=dB_mean

    #- Convergence diagnostics.

    itCount+=1

    # Updated scan-averaged PWV and sol. illumination after completed iteration.
    print('....... After iteration no. ',itCount)
    print('Mean PWV for the scan,',tgtScan,' w0 = ',w0,"mm"
          ", mean Tsun(illumination) = ",1000.0*Tsun0,"K") # scale to Kelvins

    # Mean values of dW and dB fluctuations should tend to zero.
    print("dW mean: ",dW_mean)
    print("dB mean: ",dB_mean)

    # Sky-brightness residua should tend to zero, too.
    # Convert the spwId maps to np.arrays first. We have to go via 'list'.
    resTArr=np.array(list(resT.values()))

    print("res T: ", resT,"mean +/- RMS: ",resTArr.mean(),"+/-",resTArr.std())

    print('.......')

#--> endwhile(non-zero-means)

#-- Diagnostic plots.
#   Make a plot comparing the measured fluctuations, their part attributed to
#   the intrinsic brightness changes, and error in eqs-system solution.

#-Set the plotting device.
```

```
  plt.rcParams["figure.figsize"] = (15,15)
  # X-axis label
  plt.xlabel('Time [MJDsec]')
  # Add the antenna name as a plot title.
  plt.suptitle('Antenna '+str(antID))

  # Color-map for WVR-channels/ATM-SPWs 0 - 3.
  csline={0:'-r',1:'-g',2:'-b',3:'-k'}

  #- Make the *analytical* plots - least-square residuals etc.

  # Subtract 'theoretical' and real sky+src temperature variations.
  errVec=np.matmul(JJ,varVec)-CC

  for spw,eqRow in spw2row.items():
    # Residuals/errors as temperature in Kelvins.
    plt.plot(wTime,1000.0*errVec[eqRow],csline[spw])

  #- Save the plots to a file and close.
  plt.savefig(plotDir+'/errs_ant_'+str(antID)+'.png')
  plt.show()
  plt.close()

  #-- Make the full *solution* plots: PWV(t) and Tsun(t).

  # Add variations to the mean (scan-averaged) values.
  # Scale the solar illumination temperature form kiloKelvins to Kelvins.
  pwv=w0+dW #mm
  Tsun=1000.0*(Tsun0+dB) # K

  # Set the size of the plot
  plt.rcParams["figure.figsize"] = (10,5)
  # Make slow-WV and fast-WV plots as subplots (horizontally)
  ff,(sp1,sp2)=plt.subplots(1,2)
  # Add the antenna name as a plot title.
  plt.suptitle('Antenna '+str(antID))

  # Left panel: Fast variations - apparent target brightness
  sp1.plot(wTime,Tsun,'-r')

  # Right panel: Slow variations - total PWV (compression + humidity).
  sp2.plot(wTime,pwv,'-g')

  plt.savefig(plotDir+'/dWdB/dWdB_ant_'+str(antID)+'.png')
  plt.show()
  plt.close()

  #-- Export resulting data in 'conforming' format.


  return {'time':wTime,'pwv':pwv,'T_B':Tsun,'J':JJ,'C':CC}



#--------------------------------------------------------------------------

def dWdLs3(vis,myAtm,antID,tgtScan,
           flaggedWVRchannels=[],ssType='',solIllumination=True,
```

```
        myMsmd=None,acdLoads=None):

    """
    Not strictly specific for the project 2022.1.00023.CSV (Band 5 data),
    but for the other Band 5 datasets it gives nonsense for the solar on-source
    subscan because of the WVR saturation (only the above mentioned project
    is 'lucky' as the WVR has been pointing sligthly off-limb - see dWdLs().)

    An extended version of dWdLs2(): Similarly to dWdBdG() we now allow for
    limited corrections of the 'gains' and 'intrumental offsets' of WVR
    radiometers. The reason is similar to the sci-SPW data: The calibration
    is not exact and the gains may be wrong within a couple of percents.
    Combination with the science data can help to fix these little discrepancies.

    Since the WVR data contain already ACD-calibrated temperatures, we have
    just a relative gains (close to 1.0) and offsets (close to 0.0) to become
    new instrumental variables of the system. The 'ACD' calibration equations
    thus take the form:

    G*Tamb+P_i=Tamb
    G*Thot+P_i=Thot

    If left alone, they have solution G=1.0, P_i=0.0. But now they are used in
    combination with the science data in the grand-system like in dWdBdG().
    """

    #--- Static parameters

    #- I/O

    # Input data from the ATM scan preceding the solar scan.
    atmScan=tgtScan-1

    # Output/diagnostics plots.
    plotDir='./wvr_decomp_G'
    if (not os.path.exists(plotDir)): os.makedirs(plotDir)
    if (not os.path.exists(plotDir+'/dWdB')): os.makedirs(plotDir+'/dWdB')


    #- Tools, global data

    # Flags used to check, whether we should close the tool later.
    ownMsmd=False

    if (not myMsmd):
      # Open the msmd tool.
      myMsmd=msmdtool()
      myMsmd.open(vis)
      ownMsmd=True

    # Create our dictionary with ACD load temperatures (ambient, hot) if needed.
    if (not acdLoads):
      acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                      warnIfNoLoadTemperatures = True,
                                      mymsmd=myMsmd)

    #- Target
```

```
# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.3  # kiloKelvins!!!

# Solar radius.
Rsun=696000 # km


# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for July 30 2026Band 5 observation.
'''
Target:       RA=08 37 18.01 DEC=+18 31 23.0
Sun center:   RA=08 38 15.79 DEC=+18 28 47.2
-----------------------------------------------
diff (arcsec, incl. cos(delta)): 822   155.8
'''
x_h=822.0 # arcsec
y_h=155.8  # arcsec

# Earth-Sun distance for July 30 2026 Band 5 observation.
Dsun=1.01508611721886*149597870.700 # km

# *** End of observation-specific part.

# Apparent angular solar radius in arcsec.
r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

# mu=cos(theta). Used for limb-brightenning correction below.
mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

# b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
# in Alissandrakis et al. (2020). Scaled to kiloKelvins!
b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

# Brightness temperature corrected to limb brightenning - see
# Alissandrakis et al. (2020).
Tsun00=TsunCenter+0.5*b_2*np.log(mu) # kK

# Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
f_LO=1.98e11

#- Antenna

# Rear-spillover + ohmic antenna efficiency (see Tab. 3 in White et al. 2017).
eta_l=0.95
# Forward-spillover antenna efficiency (Tab. 3 in White et al. 2017).
# WVR has, unlike the sci-receiver, on-axis recording, may have better eta_F.
eta_F=0.9

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']

# How strongly we insist on calibration solution G=1.0, P_inst=0.0.
# See comment in dWdBdG().
```

```
# The used value has been found experimentally via dWdLs3_calWeight().
calWeight=7.0

#- Atmosphere

# How many ATM (i.e. in the model) spws do we have? In this case it is
# the same as the number of the WVR channels.
nAtmSPWs=myAtm.getNumSpectralWindows()

# Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,'mm @airMass = ',airMass)
print('Estimated T_B(full Sun) = ',1000.0*Tsun00,'K') # scale to Kelvins.
print('.......')

# We do not need the msmd tool anymore.
if (ownMsmd): myMsmd.done()


#-- Extract WVR raw temperatures.

# Raw WVR Tbs
wTime,Twvr=wvr_dataRaw(vis,tgtScan,antID,ssType)

# Average over time.
TwvrMean=np.mean(Twvr,axis=1)
# Stddevs per channel.
TwvrRMS=np.std(Twvr,axis=1)

#-- Flag and remove clear outliers.

# Find all array indices, for which data in at least one of the valid
# WVR channels are out of 3 sigma interval.
validWvrChans=np.delete(range(nAtmSPWs),flaggedWVRchannels)

for SPW in validWvrChans:
  # Outliers ( > 3 sigma) in the current SPW.
  badData=np.flatnonzero(abs(Twvr[SPW]-TwvrMean[SPW])>3.0*TwvrRMS[SPW])
  if (badData.size):
    print('Flagged in WVR channel no.',SPW,': Data in times ',wTime[badData],
          ', bad temperatures T=',Twvr[SPW,badData])

    # Delete data at bad-data indices in *all* WVR channels, so that the data
    # remain homogeneous in time. Make the time axis conformant.
    Twvr=np.delete(Twvr,badData,axis=1)
    wTime=np.delete(wTime,badData)
#--> endfor(valid WVR chans)

#-- Find dimensions of the system.
```

```
# How many time samples are in this scan (after flagging)?
nSamples=Twvr.shape[1]

# How many equations do we actually have per block of 1 time-sample?
# Some WVR channels may be flagged (typically, the channel 0 is saturated).
eqBlockSize=nAtmSPWs- len(flaggedWVRchannels)

#- Relative gains and instr. offsets - one per WVR channel, no polarization.

# Gains to be initialised to their expected value of 1.0.
Grw=np.ones(nAtmSPWs)
# Offsets should be close to 0.0.
Piw=np.zeros(nAtmSPWs)

#- Reconstruct the sub-channel frequencies for our ATM SPWs.
#  We need them for proper scaling of the solar-illumantion spectrum.

# We have the same number of sub-channels for all WVR-channel ATM SPWs.
# Fortunatelly, because Python is incapable of creating a vector of vectors,
# which would have different sizes, only rectangular matrix is OK :-(
nChans=myAtm.getNumChan(0)

# Frequencies for sub-channels in our ATM SPWs.
wvrFreqs=np.zeros((nAtmSPWs,nChans))

for wC in range(nAtmSPWs):

    fMin=myAtm.getMinFreq(wC)['value'][0]*1.0e9 # GHz to Hz
    df=myAtm.getChanSep(wC)['value'][0]*1.0e6   # MHz to Hz

    wvrFreqs[wC]=fMin+df*np.arange(nChans)

#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).
#   Note: In this case, when we have 1-to-1 correspondence between the model
#   and data SPWs, we actually need not to store the per-spw values to arrays.
#   But we keep the structure of sci-SPW approach in dWdB_TBs() and dWdBdG().

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,nChans))
dTdW=np.zeros((nAtmSPWs,nChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,nChans))
tau_1=np.zeros((nAtmSPWs,nChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/ Tsun00*np.log(wvrFreqs/ f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
```

```
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin

# Initial values for the dW and dL means - anyting bigger then the stoppers.
dW_mean=0.5
dB_mean=0.5

# Iteration counter.
itCount=0

# Set the starting values for iterations.
w0=pwv0
# From dWdLs() we estimate the solar contamination ~100K = 0.1kK in the
# on-source subscans. The off-source shoud be contamination-free.
if (ssType=='on'):
  Tsun0=0.1 # kiloKelvins
else:
  Tsun0=0.0 # no illumination

#-- Iterations: Minimize the mean value of variations.

while(((abs(dW_mean) > dwMean_max) or (abs(dB_mean) > dbMean_max))
      and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample


  # N-blocks: Jacobian-part for the [dW,dT] 'natural' variables.
  if (solIllumination):
    # Solve for both dW and dB - we allow for varying solar illumination.
    J_wb=np.zeros((eqBlockSize,2)) # 2 vars repeating in non-flagged wvrChans
  else:
    # Solar illumination should be zero, solve for dW solely.
    J_wb=np.zeros((eqBlockSize,1)) # 1 var repeating in non-flagged wvrChans

  # G-block matrix: J-part for [dG,dP_inst] 'instrumental' variables.
  J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each WVR channel.

  # A-block matrix: J-part for [dG,dP_inst] for WVR "calibration" equations.
  J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR channel, amb.
  J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR channel, hot.


  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))

  # RHS-vector 'appendix' - amb. & hot 'calibration powers' (time-averaged).
  CCA=np.zeros((eqBlockSize,1))
  CCH=np.zeros((eqBlockSize,1))

  # Solution residua for each used WVR channel.
  resT={}
  # Mapping the WVR channel to the index of equation in the system.
```

```
spw2row={}


#- Re-calculate ATM model with updated <PWV>

# Set current estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

# Count the actual number of equations - some WVR channels may be flagged.
eqCount=0

#- Loop over 4 ATM spws - these directly correspond to 4 WVR channels
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Scale to kiloKelvins.
  TbsW0[atmSPW]=0.001*Tebb

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value']
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value']

  # In kiloKelvins/mm
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Transmissivity (spectral).
  trans[atmSPW]=np.exp(-opacity)
  # Wet opacity per 1mm PWV scaled by airmass.
  tau_1[atmSPW]=(airMass/w0)*tau_wet

  #- Model (_M) factors.

  # Tebb and its slope.
  TbsW0_M=TbsW0[atmSPW]
  dTdW_M=dTdW[atmSPW]

  # Transmissivity and unit-PWV opacity.
  trans_M=trans[atmSPW]
  tau_1M=tau_1[atmSPW]

  # Solar-flux model spectrum (normalized).
  SF_M=SF[atmSPW]

  #- Model + antenna efficiencies combined.

  # Sensitivity factor for PWV.
```

```
SW=(-eta_l*eta_F*Tsun0*np.mean(trans_M*tau_1M*SF_M)
    +(1.0-eta_F+eta_F*eta_l)*np.mean(dTdW_M))*Grw[atmSPW]

# Sensitivity factor for the (apparent) solar illumination temperature.
SB=eta_l*eta_F*np.mean(trans_M*SF_M)*Grw[atmSPW]

# Model T_sky as viewed by the receiver with uniform spectral gain.
Tsky=(eta_l*eta_F*Tsun0*np.mean(trans_M*SF_M)
      +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M)
      +eta_F*(1.0-eta_l)*Tamb)



#- Build the equations only for the non-flagged channels.
if (atmSPW not in flaggedWVRchannels):

  #- Build the matrix: Natural (dW & dB sensitivities) block.

  # Set the sensitivity factor(s) for the current ATM SPW as a row
  # in the block/tile for the linearized-system Jacobian matrix.
  if (solIllumination):
    # Solve for both dW & dB independently.
    J_wb[eqCount]=np.array([SW,SB])
  else:
    # Only dW is allowed to vary.
    J_wb[eqCount]=np.array([SW])


  #- Build the matrix: Instrumental (Grw & Piw sensitivities) block.

  # We have to prepend a proper offset for the Grw and Piw vars as
  # they are independendt for each SPW and polarization.
  bOffset=2*eqCount

  J_gp[eqCount,bOffset]=1.0
  J_gp[eqCount,bOffset+1]=Tsky

  #- Build the matrix: ACD powers (amb & hot) 2-block.

  J_cdA[eqCount,bOffset]=1.0
  J_cdA[eqCount,bOffset+1]=Tamb

  # Hot-power sub-block.
  J_cdH[eqCount,bOffset]=1.0
  J_cdH[eqCount,bOffset+1]=Thot

  #- Build the RHS vector: WVR data

  # RHS: Twvr(t) with the stationary (in the working point) ATM-based part
  # subtracted. Rescale Twvr to kiloKelvin.
  dTw=0.001*Twvr[atmSPW]-Tsky

  # Set it as the RHS component.
  CC[eqCount]=dTw

  #- Ammend the RHS vector: Calibration data.

  # Ambient-load power with its working-point part subtracted.
```

```
      CCA[eqCount]=Tamb-Piw[atmSPW]-Grw[atmSPW]*Tamb
      # Hot-load power with its working-point part subtracted.
      CCH[eqCount]=Thot-Piw[atmSPW]-Grw[atmSPW]*Thot


      #- Diagnostics

      # Residua of sky temperature for mean PWV and Tsun (operation point).
      resT[atmSPW]=dTw.mean()

      # Mapping between ATM SPW and equation number.
      spw2row[atmSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',atmSPW,' WVR channel.')
    #--> endif(not flaggged)

  #--> endfor(ATM SPWs)

  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged SPWs/polarization channels.")
    return False

  #-- Solution of the system of linearised equations. See dWdBdG() for
  #   explanation.

  #- Calculate the small-block matrices and their inversions needed for
  #  grand-system inversion.

  # Natural (N-block) matrices.
  J_wb_T=np.transpose(J_wb)
  J_K=np.matmul(J_wb_T,J_wb)
  J_K_inv=np.linalg.inv(J_K)

  # Gain/P_inst (G-block) matrices.
  J_gp_T=np.transpose(J_gp)

  # ACD rows (A-block) matrices.
  # Calculate transpositions per sub-block (ambient/hot).
  J_cdA_T=np.transpose(J_cdA).copy()
  J_cdH_T=np.transpose(J_cdH).copy()

  # Increase importance of the ACD rows by calibration weight.
  # Only the transposed A-block matrices should be scaled up!
  J_cdA_T*=calWeight
  J_cdH_T*=calWeight

  # Compose the full block and its transposition. The transposition to be
  # scaled up - see above. Or, alternativelly, it can be built as
  # np.hstack([J_cdA_T,J_cdH_T]) from already scaled-up transposed sub-blocks.
  J_cd=np.vstack([J_cdA,J_cdH]).copy()
  J_cd_T=calWeight*np.transpose(J_cd)

  #- Base-matrix extensions.
```

```
    # Right and lower bands: Their (nEqs x 3) and (3 x nEqs) repeating blocks.
    Rb=np.matmul(J_wb_T,J_gp)
    Lb=np.transpose(Rb)

    # Lower-right corner
    LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

    #- Frequently repeating matrix products. Cf. with extMatInv() above.

    # Ainv x Rb and Lb x Ainv
    AinvRb=np.matmul(J_K_inv,Rb)
    LbAinv=np.transpose(AinvRb)

    #- Schur complement and its inversion.
    SC=LRc-nSamples*np.matmul(Lb,AinvRb)
    SC_inv=np.linalg.inv(SC)

    #- Blocks of inverted grand-system matrix.

    # Start with the lower-left block as it repeats in upper-left block, too.
    Minv21=-np.matmul(SC_inv,LbAinv)

    # Upper-left block: Off-diagonal terms. The diagonal blocks differ just
    # by added J_K_inv.
    Minv11_OD=-np.matmul(AinvRb,Minv21)

    # Upper-right block.
    Minv12=np.transpose(Minv21)

    # Lower-right block.
    Minv22=SC_inv

    #-- Apply the inverted matrix components to the RHS vector.

    # Sum of the RHS along the time axis, keeping the SPW:POL dimension.
    # Preserve the number of matrix rows, just sum their contents, i.e. make
    # the single-column matrix.
    CCsum=np.reshape(CC.sum(axis=1),(-1,1))

    #- Form the RHS vector for the normal equations (K-form minimizer),
    #  i.e, multiply the raw data by JJ_T

    # Start with the natural-variables dW and dB part.
    C_K_wb=np.matmul(J_wb_T,CC)       # 2 x nSamples matrix

    # Make the same also for its time-sum.
    C_K_wb_sum=np.matmul(J_wb_T,CCsum)    # 2 x 1

    # Follow with the instrumental part.
    C_K_gp=(np.matmul(J_gp_T,CCsum)       # 2*eqCount x 1
            +np.matmul(J_cdA_T,CCA)
            +np.matmul(J_cdH_T,CCH))

    #- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

    # Constant part of the natural-variables vector (2 x 1)
    nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)
```

```
    # Time-varying part of the natural-variables vector. It comes from the
    # addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
    nvVar=np.matmul(J_K_inv,C_K_wb)   # 2 x nSamples

    # Compose the complete natural variables vector.
    varNat=nvVar+nvConst

    # Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
    varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)

    #- Reshape the results - extract our elemenytary variables.

    # Seaparate the natural-vars components.
    # PWV variations present always.
    dW=varNat[0]

    # dB variations only in case of solar illumination, otherwise they are zeros
    if (solIllumination):
      dB=varNat[1]
    else:
      dB=np.zeros(nSamples)


    # Reshape instrumental parameters to two rows (transpose): 1st is dP_inst,
    # the 2nd dG, one component per WVR channel.
    dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

    # Extract the shifts in the P_inst and G
    dPinst=dPG[0]
    dGinst=dPG[1]

    #- Analytics: Calculate the errors for the over-determined set of eqs.

    # err=A.x-b: Sci-data part.
    errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
    # Xi square for the sci-data part.
    xiSci=np.sum(errVec*errVec)


    # err=A.x-b: Calibration-data part.
    errVecACDamb=np.matmul(J_cdA,varInst)-CCA
    errVecACDhot=np.matmul(J_cdH,varInst)-CCH
    # Xi square for the cal-data part.
    xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)

    # Grand-total Xi-square normalized to the number of equations (sci/acd).
    xiSquare=xiSci/nSamples+xiACD/calWeight


    #- All variations should tend to zero means.

    dW_mean=dW.mean()
    dB_mean=dB.mean()

    # Update stationary part of PWV and solar T_B  by means of their variations.
    w0+=dW_mean
    Tsun0+=dB_mean
```

```
  # Update also the Piw and Grw by the found differences.
  for atmSPW,eqCount in spw2row.items():
    # Update the P_inst and G.
    Piw[atmSPW]+=dPinst[eqCount]
    Grw[atmSPW]+=dGinst[eqCount]
  #--> endfor(SPWs with valid XX pol)


  #- Convergence diagnostics.

  itCount+=1

  # Updated scan-averaged PWV and sol. illumination after completed iteration.
  print('....... After iteration no. ',itCount)
  print('Mean PWV for the scan,',tgtScan,' w0 = ',w0,"mm"
        ", mean Tsun(illumination) = ",1000.0*Tsun0,"K") # scale to Kelvins

  # Mean values of dW and dB fluctuations should tend to zero.
  print("dW mean: ",dW_mean)
  print("dB mean: ",dB_mean)

  # Sky-brightness residua should tend to zero, too.
  # Convert the spwId maps to np.arrays first. We have to go via 'list'.
  resTArr=np.array(list(resT.values()))

  print("res T: ", resT,"mean +/- RMS: ",resTArr.mean(),"+/-",resTArr.std())

  print('.......')

#--> endwhile(non-zero-means)

#-- Diagnostic plots.
#   Make a plot comparing the measured fluctuations, their part attributed to
#   the intrinsic brightness changes, and error in eqs-system solution.

# We may have a set of per-scan figures: Distinguish by scan ID in filename.
scID='_{:02d}'.format(tgtScan)

#-Set the plotting device.
plt.rcParams["figure.figsize"] = (15,15)
# X-axis label
plt.xlabel('Time [MJDsec]')
# Add the antenna name as a plot title.
plt.suptitle('Antenna '+str(antID)+', scan '+str(tgtScan))

# Color-map for WVR-channels/ATM-SPWs 0 - 3.
csline={0:'-r',1:'-g',2:'-b',3:'-k'}

#- Make the *analytical* plots - least-square residuals etc.

for spw,eqRow in spw2row.items():
  # Residuals/errors as temperature in Kelvins.
  plt.plot(wTime,1000.0*errVec[eqRow],csline[spw])

#- Save the plots to a file and close.

plt.savefig(plotDir+'/errs_ant_'+str(antID)+'_scan'+scID+'.png')
```

```
  plt.show()
  plt.close()


  #-- Make the full *solution* plots: PWV(t) and Tsun(t).

  # Add variations to the mean (scan-averaged) values.
  # Scale the solar illumination temperature from kiloKelvins back to Kelvins.
  pwv=w0+dW #mm
  Tsun=1000.0*(Tsun0+dB) # K

  # Set the size of the plot
  plt.rcParams["figure.figsize"] = (10,5)
  # Make slow-WV and fast-WV plots as subplots (horizontally)
  ff,(sp1,sp2)=plt.subplots(1,2)
  # Add the antenna name as a plot title.
  plt.suptitle('Antenna '+str(antID)+', scan '+str(tgtScan))

  # Left panel: Fast variations - apparent target brightness
  sp1.plot(wTime,Tsun,'-r')

  # Right panel: Slow variations - total PWV (compression + humidity).
  sp2.plot(wTime,pwv,'-g')

  plt.savefig(plotDir+'/dWdB/ant_'+str(antID)+'_scan_'+scID+'.png')
  plt.show()
  plt.close()


  #-- Show, how much the instrumental variables G and P_inst have changed.

  print("Corrected relative gains [dimension-less]: ", Grw)
  print("Corrected instrumental offsets [K]:", 1000.0*Piw)

  # Print total quadratic error for the variables.
  print("\n======= Xi-squares =======     SCI: ",xiSci,
        ", ACD:",xiACD,
        ", Total:", xiSquare,"\n")


  #-- Export the resulting data in the 'conforming' format.

  return {'time':wTime,'pwv':pwv,'T_B':Tsun,'J':J_wb,'C':CC}


#--------------------------------------------------------------------------------
def dWdLs4(vis,myAtm,antID,tgtScan,
           flaggedWVRchannels=[],ssType='',solIllumination=True,
           myMsmd=None,acdLoads=None):

  """
  Improved version of dWdLs3().

  Inspired by dWdBdG2(), we now handle the situation of enforced zero solar
  illumination (setting *solIllumination=False*) more appropriatelly.
  The insofar procedure in dWdLs3() sets Tsun=0.0 hard and 'degrades' the T_B
  variations dB from a variable to a constant (we narrow the J_wb matrix by
  cutting the dB column) and it is set as a 'hard-wired solution' to 0.0.
```

```
    In this new implementation we keep the T_B to be a variable, but we set its
    time-averaged value Tsun to 0 by implicit setting of parameter, corresponding
    to *T_tgt* in dWdBdG2(), to zero. In other words, we add the equations that
    force the time-averaged solar-illumination Tsun to zero, still keeping its
    small variations dB. As a result, the zero-illumination condition is not
    super-seded to other equations, but it has its relative strength in the
    entire system, expresssed by internal (experimentaly found) parameter mWeight.

    As in dWdBdG2(), we have specific antenna-efficiencies setup per antenna type:
    This is another 'upgrade' to dWdLs3(). Unlike the dWdBdG2() we need not
    to introduce a 'natural' weight factor WF=1/G, because our 'raw' measured WVR
    signal is already expressed as temperature and all 4 WVR channels have the
    same weight (this is not true fro sci SPWs, handled in dWdBdG2(), where SQLD
    spws have different integration time and, consequenctly, powers and gains).
    """

    #--- Static parameters

    #- I/O

    # Input data from the ATM scan preceding the target scan.
    # Note: Here it has a little sense sofar - we use calibrated Twvr in Kelvins
    # and the nominal gain Gwr thus Gwr=1.0. We use it only for finding Tamb
    # and Thot temperatures in order to keep the structure of equations conformant
    # the that in sWdBdG2(). However, we may later try expolarion of the real
    # ATM scans, which have also CALIBRATE_WVR#AMBIENT/HOT intents. In those
    # cases, we have to be more carefull and identify the actual ATM scan
    # relevant to the *tgtScan*
    atmScan=tgtScan-1

    # Output/diagnostics plots.
    plotDir='./wvr_decomp_G'
    if (not os.path.exists(plotDir)): os.makedirs(plotDir)
    if (not os.path.exists(plotDir+'/dWdB')): os.makedirs(plotDir+'/dWdB')


    #- Tools, global data

    # Flags used to check, whether we should close the tool later.
    ownMsmd=False

    # Open own tools if 'global' not provided
    if (not myMsmd):
      # Open the msmd tool.
      myMsmd=msmdtool()
      myMsmd.open(vis)
      ownMsmd=True

    # Create our dictionary with ACD load temperatures (ambient, hot) if needed.
    if (not acdLoads):
      acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                      warnIfNoLoadTemperatures = True,
                                      mymsmd=myMsmd)

    #- Antenna

    antNames=myMsmd.antennanames()
```

```
myAntName=antNames[antID]
antType=myAntName[:2]

# Map the antenna efficiencies by antenna type. As per investigation from
# merged T_sun temperatures form 2023 and 2026 data (project_22_25_Tb.ods).
eta_l_map={'CM':0.941,'DA':0.947,'DV':0.956}
eta_F_map={'CM':0.891,'DA':0.897,'DV':0.906}

# Rear-spillover + ohmic antenna efficiency - see project_22_25_Tb.ods
eta_l=eta_l_map[antType]
# Forward-spillover antenna efficiency - see project_22_25_Tb.ods
eta_F=eta_F_map[antType]

# We have two exception that seem to have systematically lower
# antenna efficiencies: DA64 and DV08 - see project_22_25_Tb.ods
if myAntName=='DA64':
  eta_l=0.929
  eta_F=0.880
if myAntName=='DV08':
  eta_l=0.943
  eta_F=0.894

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']

# How strongly we insist on calibration solution G=1.0, P_inst=0.0.
# See comment in dWdBdG2().
# The used value has been found experimentally via dWdLs4_weights().
# Search mark in dWdLs4(): CW_2
calWeight=20.0

#- Target

# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.300  # kiloKelvins!!!

# Solar radius.
Rsun=696000 # km


# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.


"""
# *** This part is observation specific - need to read ephemeris !!!
```

```
# Heliocentric coordinates for July 30 2026Band 5 observation.
 '''
Target:       RA=08 37 18.01 DEC=+18 31 23.0
Sun center:    RA=08 38 15.79 DEC=+18 28 47.2
 -----------------------------------------------
diff (arcsec, incl. cos(delta)): 822    155.8
 '''
x_h=822.0 # arcsec
y_h=155.8  # arcsec

# Earth-Sun distance for July 30 2026 Band 5 observation.
Dsun=1.01508611721886*149597870.700 # km

# *** End of observation-specific part.
"""

 # Apparent angular solar radius in arcsec.
 r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

 # mu=cos(theta). Used for limb-brightenning correction below.
 mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

 # b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
 # in Alissandrakis et al. (2020). Scaled to kiloKelvins!
 b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

 # Brightness temperature corrected to limb brightenning - see
 # Alissandrakis et al. (2020).
 Tsun00=TsunCenter+0.5*b_2*np.log(mu) # kK

 # Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
 f_LO=1.98e11

 # Solar illumination. May be either zero or unbound, depending on the
 # solIllumination parameter setting (T/F).
 T_tgt=0.0

 if (solIllumination):
   # Do not apply Tsun=0 condition, we allow for unbound illumination.
   # If mWeight is set to zero, the T_tgt value itself is irrelevant.
   mWeight=0.0
 else:
   # We are pointing far off the Sun. Enforce zero illumination by adding
   # the T_tgt=0.0 equation with the appropriate weight.
   # Search mark in dWdLs4(): MW_2
   mWeight=1.0

 #- Atmosphere

 # How many ATM (i.e. in the model) spws do we have? In this case it is
 # the same as the number of the WVR channels.
 nAtmSPWs=myAtm.getNumSpectralWindows()

 # Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
 pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

 # Set initial estimation of PWV to the ATM model.
```

```
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,'mm @airMass = ',airMass)
print('Estimated T_B(full Sun) = ',1000.0*Tsun00,'K') # scale to Kelvins.
print('.......')

# The MS metadata has been read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()

#-- Extract WVR raw temperatures.

# Raw WVR Tbs
wTime,Twvr=wvr_dataRaw(vis,tgtScan,antID,ssType)

# Average over time.
TwvrMean=np.mean(Twvr,axis=1)
# Stddevs per channel.
TwvrRMS=np.std(Twvr,axis=1)

#-- Flag and remove clear outliers.

# Find all array indices, for which data in at least one of the valid
# WVR channels are out of 3 sigma interval.
validWvrChans=np.delete(range(nAtmSPWs),flaggedWVRchannels)

for SPW in validWvrChans:
  # Outliers ( > 3 sigma) in the current SPW.
  badData=np.flatnonzero(abs(Twvr[SPW]-TwvrMean[SPW])>3.0*TwvrRMS[SPW])
  if (badData.size):
    print('Flagged in WVR channel no.',SPW,': Data in times ',wTime[badData],
          ', bad temperatures T=',Twvr[SPW,badData])

    # Delete data at bad-data indices in *all* WVR channels, so that the data
    # remain homogeneous in time. Make the time axis conformant.
    Twvr=np.delete(Twvr,badData,axis=1)
    wTime=np.delete(wTime,badData)
#--> endfor(valid WVR chans)

#-- Find dimensions of the system.

# How many time samples are in this scan (after flagging)?
nSamples=Twvr.shape[1]

# How many equations do we actually have per block of 1 time-sample?
# Some WVR channels may be flagged (typically, the channel 0 is saturated).
eqBlockSize=nAtmSPWs-len(flaggedWVRchannels)

# Pre-calculated frequently repeating factor: Inverse number of time samples.
nsInv=1.0/nSamples

#- Relative gains and instr. offsets - one per WVR channel, no polarization.

# Gains to be initialised to their expected value of 1.0.
```

```
Grw=np.ones(nAtmSPWs)
# Offsets should be close to 0.0.
Piw=np.zeros(nAtmSPWs)

#- Reconstruct the sub-channel frequencies for our ATM SPWs.
#  We need them for proper scaling of the solar-illumantion spectrum.

# We have the same number of sub-channels for all WVR-channel ATM SPWs.
# Fortunatelly, because Python is incapable of creating a vector of vectors,
# which would have different sizes, only rectangular matrix is OK :-(
nChans=myAtm.getNumChan(0)

# Frequencies for sub-channels in our ATM SPWs.
wvrFreqs=np.zeros((nAtmSPWs,nChans))

for wC in range(nAtmSPWs):

    fMin=myAtm.getMinFreq(wC)['value'][0]*1.0e9 # GHz to Hz
    df=myAtm.getChanSep(wC)['value'][0]*1.0e6   # MHz to Hz

    wvrFreqs[wC]=fMin+df*np.arange(nChans)

#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).
#   Note: In this case, when we have 1-to-1 correspondence between the model
#   and data SPWs, we actually need not to store the per-spw values to arrays.
#   But we keep the structure of sci-SPW approach in dWdB_TBs() and dWdBdG().

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,nChans))
dTdW=np.zeros((nAtmSPWs,nChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,nChans))
tau_1=np.zeros((nAtmSPWs,nChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/Tsun00*np.log(wvrFreqs/f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
# We used those as weights in calculation of epsilon - the normalised
# inter-iteration change of operation-point at the very end of each iteration.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin
dG_max=0.001       # Relative scale dG/G
dTinst_max=0.001   # dPinst/G in kiloKelvins

# Iteration counter.
```

```
itCount=0

# Normalized inter-iteration change of operation point - its initial value.
# Anything bigger than 1.0
epsilon=10.0

# Set the starting values / an initial guess for iterations.
w0=pwv0
# From dWdLs() we estimate the solar contamination ~100K = 0.1kK in the
# on-source subscans. The off-source should be contamination-free.
if (ssType=='on'):
  Tsun0=0.1 # kiloKelvins
else:
  Tsun0=0.0 # no illumination

#-- Iterations: Minimize the mean value of variations.

while( (epsilon>1.0) and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample


  # N-blocks: Jacobian-part for the [dW,dT] 'natural' variables.
  J_wb=np.zeros((eqBlockSize,2)) # 2 vars repeating in non-flagged wvrChans
  # G-block matrix: J-part for [dG,dP_inst] 'instrumental' variables.
  J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each WVR channel.

  # A-block matrix: J-part for [dG,dP_inst] for WVR "calibration" equations.
  J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR channel, amb.
  J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR channel, hot.
  # M-block matrix: Jacobian part for [dW,dT] in array-mean eqs.
  # This matrix is constant and can be defined already here.
  J_M=np.array([[0.0, nsInv]])


  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))

  # RHS-vector 'appendix 1' - amb. & hot 'calibration powers' (time-averaged).
  CCA=np.zeros((eqBlockSize,1))
  CCH=np.zeros((eqBlockSize,1))

  # RHS-vector 'appendix 2' - array-mean values of the target T_B
  CCM=np.zeros((1,1))

  # Solution residua for each used WVR channel.
  resT={}
  # Mapping the WVR channel to the index of equation in the system.
  spw2row={}


  #- Re-calculate ATM model with updated <PWV> and solar illumination.

  # Set current estimation of PWV to the ATM model.
  myAtm.setUserWH2O(qa.quantity(w0,'mm'))
```

```
    # Count the actual number of equations - some WVR channels may be flagged.
    eqCount=0

    #- Loop over 4 ATM spws - these directly correspond to 4 WVR channels
    for atmSPW in range(nAtmSPWs):

      #- Model value of the sky/atm brightness with current pwv.
      Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
      # Scale to kiloKelvins.
      TbsW0[atmSPW]=0.001*Tebb

      #- Slope (centered difference) of the atm brightness with current pwv.
      TebbH=myAtm.getTebbSkySpec(atmSPW,
                                 qa.quantity(w0+dws,'mm'))[1]['value']
      TebbL=myAtm.getTebbSkySpec(atmSPW,
                                 qa.quantity(w0-dws,'mm'))[1]['value']

      # In kiloKelvins/mm
      dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

      #- Opacities & transmisivity.

      # Dry zenith opacity
      tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
      # Water-vapour related 'wet' opacity
      tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
      # Total opacity (spectral).
      opacity=airMass*(tau_wet+tau_dry)

      # Transmissivity (spectral).
      trans[atmSPW]=np.exp(-opacity)
      # Wet opacity per 1mm PWV scaled by airmass.
      tau_1[atmSPW]=(airMass/w0)*tau_wet

      #- Model (_M) factors.

      # Tebb and its slope.
      TbsW0_M=TbsW0[atmSPW]
      dTdW_M=dTdW[atmSPW]

      # Transmissivity and unit-PWV opacity.
      trans_M=trans[atmSPW]
      tau_1M=tau_1[atmSPW]

      # Solar-flux model spectrum (normalized).
      SF_M=SF[atmSPW]

      #- Model + antenna efficiencies combined.

      # Sensitivity factor for PWV.
      SW=(-eta_l*eta_F*Tsun0*np.mean(trans_M*tau_1M*SF_M)
          +(1.0-eta_F+eta_F*eta_l)*np.mean(dTdW_M))*Grw[atmSPW]

      # Sensitivity factor for the (apparent) solar illumination temperature.
      SB=eta_l*eta_F*np.mean(trans_M*SF_M)*Grw[atmSPW]

      # Model T_sky as viewed by the receiver with uniform spectral gain.
      Tsky=(eta_l*eta_F*Tsun0*np.mean(trans_M*SF_M)
```

```
        +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M)
        +eta_F*(1.0-eta_l)*Tamb)


# Build the equations only for the non-flagged channels.
if (atmSPW not in flaggedWVRchannels):

  #- Build the matrix: Natural (dW & dB sensitivities) block.

  # Set the sensitivity factor(s) for the current ATM SPW as a row
  # in the block/tile for the linearized-system Jacobian matrix.
  J_wb[eqCount]=np.array([SW,SB])


  #- Build the matrix: Instrumental (Grw & Piw sensitivities) block.

  # We have to prepend a proper offset for the Grw and Piw vars as
  # they are independendt for each SPW and polarization.
  bOffset=2*eqCount

  J_gp[eqCount,bOffset]=1.0
  J_gp[eqCount,bOffset+1]=Tsky

  #- Build the matrix: ACD powers (amb & hot) 2-block.

  J_cdA[eqCount,bOffset]=1.0
  J_cdA[eqCount,bOffset+1]=Tamb

  # Hot-power sub-block.
  J_cdH[eqCount,bOffset]=1.0
  J_cdH[eqCount,bOffset+1]=Thot

  #- Build the RHS vector: WVR data

  # RHS: Twvr(t) with the stationary (in the working point) ATM-based part
  # subtracted. Rescale measured WVR temperatures  to kiloKelvin.
  dTw=0.001*Twvr[atmSPW]-Piw[atmSPW]-Grw[atmSPW]*Tsky

  # Set it as the RHS component.
  CC[eqCount]=dTw

  #- Ammend the RHS vector: Calibration data.

  #  This is a sort of trick. It would be natural to add equations
  #  Grw[atmSPW]=1.0, Piw[atmSPW]=0.0. But we like to keep the structure
  #  of dWdBdG2() equations (with all the lin-algebra apparatus for its
  #  'pain-less' solution), so we encoded the trivial eqs. above into
  #  form Tamb=Grw*Tamb+Piw, Thot=Grw*Thot+Piw, which has exactly the
  #  same solution Grw=1.0, Piw=0.0. This form is co-variant with that
  #  used in PWV proxy calculation in dWdBdG2().


  # Ambient-load power with its working-point part subtracted.
  # Note: Unlike Twvr, the Tamb and Thot are in kiloKelvins already.
  CCA[eqCount]=Tamb-Piw[atmSPW]-Grw[atmSPW]*Tamb
  # Hot-load power with its working-point part subtracted.
  CCH[eqCount]=Thot-Piw[atmSPW]-Grw[atmSPW]*Thot
```

```
      #- Diagnostics

      # Residua of sky temperature for mean PWV and Tsun (operation point).
      resT[atmSPW]=dTw.mean()

      # Mapping between ATM SPW and equation number.
      spw2row[atmSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',atmSPW,' WVR channel.')
    #--> endif(not flagged)

  #--> endfor(ATM SPWs)

  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged SPWs/polarization channels.")
    return False


  #-- Solution of the grand system of linearised equations. See dWdBdG2() for
  #   explanation.

  #- Ammend the RHS vector: Array-mean target brightness temperatures.

  # Effective expected target T_B with the model at work-point subtracted.
  # In our case, effective T_B is just the part of 'parasitic' illumination.
  # For far-off-Sun pointings T_tgt is set to zero. For near-Sun pointings
  # we use solIllumantion=True and Tsun0 is unbound because mWeight=0.
  CCM[0]=T_tgt-Tsun0

  #- Calculate the small-block matrices and their inversions needed for
  #  grand-system inversion.

  # 1. Natural (N-block) matrices.
  J_wb_T=np.transpose(J_wb)
  J_K=np.matmul(J_wb_T,J_wb)
  J_K_inv=np.linalg.inv(J_K)

  # 2. Gain/P_inst (G-block) matrices.
  J_gp_T=np.transpose(J_gp)

  # 3. ACD rows (A-block) matrices.
  # Calculate transpositions per sub-block (ambient/hot).
  # We need to copy() here - see the same place in dWdBdG2() for explanation!
  J_cdA_T=np.transpose(J_cdA).copy()
  J_cdH_T=np.transpose(J_cdH).copy()

  # Increase importance of the ACD rows by calibration weight.
  # Only the transposed A-block matrices should be scaled up!
  J_cdA_T*=calWeight
  J_cdH_T*=calWeight
```

```
    # Compose the full block and its transposition. The transposition to be
    # scaled up - see above. Or, alternativelly, it can be built as
    # np.hstack([J_cdA_T,J_cdH_T]) from already scaled-up transposed sub-blocks.
    J_cd=np.vstack([J_cdA,J_cdH]).copy()
    J_cd_T=calWeight*np.transpose(J_cd)

    # 4. Array-mean (M-block) related matrices. Incorporate mWeight into M^T, in
    # line with what we have done with calWeight for the A^T and H^T matrices.
    M_T=np.array([[0.0],
                  [nsInv*mWeight]])    # mWeight*M^T

    # Pre-calculated nSamples*mWeight*(M^T x M) matrix.
    # Like with A^T x A, the M^T x M product is scaled-up by mWeight just once
    # (no mWeight squared!). Moreover, we scale it up here yet by nSamples,
    # in order to avoid trivial product nsInv*nSamples=1 in the following calcs.
    nsMTM=np.array([[0.0, 0.0],
                    [0.0, nsInv*mWeight]])

    # Frequently repeating blocks. NM_inv plays a role of modified J_K_inv.
    NM=J_K+nsMTM                # J_K+nSamples*mWeight*M^T x M
    NM_inv=np.linalg.inv(NM)    # NM^(-1)

    # Off-diagonal blocks of the inverted natural sub-matrix (Ainv).
    J_M_inv=nsInv*(NM_inv-J_K_inv)


    #- Base-matrix extensions.

    # Right and lower bands: Their (nEqs x 2) and (2 x nEqs) repeating blocks.
    Rb=np.matmul(J_wb_T,J_gp)
    Lb=np.transpose(Rb)

    # Lower-right corner
    LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

    #- Frequently repeating matrix products. Cf. with extMatInv() above.

    # Ainv x Rb and Lb x Ainv
    AinvRb=np.matmul(NM_inv,Rb)
    LbAinv=np.transpose(AinvRb)

    #- Schur complement and its inversion.
    SC=LRc-nSamples*np.matmul(Lb,AinvRb)
    SC_inv=np.linalg.inv(SC)

    #- Blocks of inverted grand-system matrix.

    # Start with the lower-left block as it repeats in upper-left block, too.
    Minv21=-np.matmul(SC_inv,LbAinv)

    # Upper-left block: Off-diagonal terms. The diagonal blocks differ just
    # by added J_K_inv.
    Minv11_OD=J_M_inv-np.matmul(AinvRb,Minv21)

    # Upper-right block.
    Minv12=np.transpose(Minv21)

    # Lower-right block.
```

```
Minv22=SC_inv

#-- Apply the inverted matrix components to the RHS vector.

# Sum of the RHS along the time axis, keeping the SPW:POL dimension.
# Preserve the number of matrix rows, just sum their contents, i.e. make
# the single-column matrix.
CCsum=np.reshape(CC.sum(axis=1),(-1,1))

#- Form the RHS vector for the normal equations (K-form minimizer),
#  i.e, multiply the raw data by JJ_T

# Start with the natural-variables dW and dB part.
C_K_wb=np.matmul(J_wb_T,CC)+np.matmul(M_T,CCM)   # 2 x nSamples matrix

# Make the same also for its time-sum.
C_K_wb_sum=np.matmul(J_wb_T,CCsum)+nSamples*np.matmul(M_T,CCM)    # 2 x 1

# Follow with the instrumental part.
C_K_gp=(np.matmul(J_gp_T,CCsum)       # 2*eqCount x 1
        +np.matmul(J_cdA_T,CCA)
        +np.matmul(J_cdH_T,CCH))

#- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

# Constant part of the natural-variables vector (2 x 1)
nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)

# Time-varying part of the natural-variables vector. It comes from the
# addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
nvVar=np.matmul(J_K_inv,C_K_wb)   # 2 x nSamples

# Compose the complete natural variables vector.
varNat=nvVar+nvConst

# Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)

#- Reshape the results - extract our elementary variables.

# Seaparate the natural-vars components.
# PWV variations.
dW=varNat[0]
# Illumination/brightness variations: In this approach we may have them, too
dB=varNat[1]


# Reshape instrumental parameters to two rows (transpose): 1st is dP_inst,
# the 2nd dG, one component per WVR channel.
dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

# Extract the shifts in the P_inst and G
dPinst=dPG[0]
dGinst=dPG[1]

#- Analytics: Calculate the errors for the over-determined set of eqs.
#  See corresponding part of dWdBdG2() for explanation of error estimation.
```

```
    # err=f(x)-b: Sci-data part.
    errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
    # Xi square for the sci-data part, normalized per one equation.
    xiSci=np.sum(errVec*errVec)/(nSamples*eqBlockSize)

    # Xi Squares per SPW:Pol channel. Sum/average only over times.
    xiSciPerChan=np.sum(errVec*errVec,axis=1)/nSamples
    # Its mean and RMS.
    xscMean=xiSciPerChan.mean()
    xscRMS=xiSciPerChan.std()
    # Find outliers ( > 3 sigma). The badData array contains indices of
    # those equations, where the agreement between model and observation is
    # exceptionaly weak.
    badData=np.flatnonzero(abs(xiSciPerChan-xscMean)>3.0*xscRMS)

    # Show solution errors per WVR channel and mark (later possibly auto-flag)
    # the WVR channels with bad data, if we have some.
    # We do not have an inverse map to spw2row, so we have to traverse.
    for wvrChan,eqRow in spw2row.items():
      print('WVR chan '+str(wvrChan)+': Error: ',
            np.sqrt(xiSciPerChan[eqRow])*1000.0,'K')
      if eqRow in badData: print('\t\t!!! Bad data in WVR chan',str(wvrChan),
                                 ':',(xiSciPerChan[eqRow]-xscMean)/xscRMS,
                                 'sigma from the mean.')

    # err=f(x)-b: Calibration-data part.
    errVecACDamb=np.matmul(J_cdA,varInst)-CCA
    errVecACDhot=np.matmul(J_cdH,varInst)-CCH
    # Xi square for the cal-data part.
    xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)
    # Normalized per equation - see corresponding-place comment in dWdBdG2().
    xiACD/=eqBlockSize

    # err=f(x)-b: Array-mean T_B (illumination) part.
    errVecMeanTb=-CCM
    # Xi square for the averaged illumination part.
    # We have just a single equation, thus no normalization is required.
    xiTbM=np.sum(errVecMeanTb*errVecMeanTb)

    # Grand-total Xi-square normalized to the number of equations for
    # the natural/science, ACD, and mean-array sub-blocks.
    xiSquare=xiSci+xiACD+xiTbM

    #- All variations should tend to zero means.

    dW_mean=dW.mean()
    dB_mean=dB.mean()

    # Update stationary part of PWV and illumination by avgs of their variations
    w0+=dW_mean
    Tsun0+=dB_mean


    # Update also the Piw and Grw by the found differences.
    # Find the maximum inter-iteration change for convergence diagnostics.
    dT_inst_sup=-1.0
    dG_rel_sup=-1.0
```

```
  for atmSPW,eqCount in spw2row.items():
    # Update the P_inst and G.
    Piw[atmSPW]+=dPinst[eqCount]
    Grw[atmSPW]+=dGinst[eqCount]

    # Change of inst. power expressed in kiloKelvins.
    dT_inst=abs(dPinst[eqCount]/Grw[wvrChan])
    if (dT_inst > dT_inst_sup): dT_inst_sup=dT_inst

    # Change of gain in the relative scale.
    dG_rel=abs(dGinst[eqCount]/Grw[wvrChan])
    if (dG_rel > dG_rel_sup): dG_rel_sup=dG_rel
  #--> endfor(non-flagged WVRchannels/atmSPWs)

  #-- Convergence diagnostics / Iteration control.

  # Calculate the normalized inter-iteration change of operation point.
  # The factor 0.5 comes from sqrt(1/4) as we have 4 'orthogonal' variables.
  epsilon=0.5*np.sqrt((dW_mean/dwMean_max)**2+(dB_mean/dbMean_max)**2+
                      (dG_rel_sup/dG_max)**2+(dT_inst_sup/dTinst_max)**2)

  # Increment the iteration counter.
  itCount+=1


  #- Diagnostic print-outs.

  # Updated scan-averaged PWV and sol. illumination after completed iteration.
  print('vvvvvvv\n After iteration no. ',itCount)
  print('Mean PWV for the scan,',tgtScan,' w0 = ',w0,"mm"
        ", mean Tsun(illumination) = ",1000.0*Tsun0,"K") # scale to Kelvins

  # Mean values of dW and dB fluctuations should tend to zero.
  print("dW mean: ",dW_mean)
  print("dB mean: ",dB_mean)

  # Sky-brightness residua should tend to zero, too.
  # Convert the spwId maps to np.arrays first. We have to go via 'list'.
  # Scale-up back to Kelvins (we calculate in kiloKelvins).
  resTArr=1000.0*np.array(list(resT.values()))

  print("res T: ", resT,"mean +/- RMS: ",resTArr.mean(),"+/-",resTArr.std())

  # Note: Here we measure the Pinst change directly in Kelvin(!), as nominal
  # Piw is zero. For gain, we need not to divide by Grw as it is nominally 1.
  print("\nInstrumental characteristics shifts during the iteration:")
  for atmSPW,eqCount in spw2row.items():
    print("WVR chanNum = ",atmSPW,"\tdP_I =",1000.0*dPinst[eqCount],"K",
          "\tdG/G =",dGinst[eqCount]*100,"%")

  print('^^^^^^^\n\n')

#--> endwhile(non-zero-means) / End of solution iteration loop.

#-- Diagnostic plots.
#   Make a plot comparing the measured fluctuations, their part attributed to
#   the intrinsic brightness changes, and error in eqs-system solution.
```

```
# We may have a set of per-scan figures: Distinguish by scan ID in filename.
scID='_{:02d}'.format(tgtScan)
# Antenna ID as a 2-digit string.
antIDstr='{:02d}'.format(int(antID))

#-Set the plotting device.
plt.rcParams["figure.figsize"] = (15,15)
# X-axis label
plt.xlabel('Time [MJDsec]')
# Add the antenna name as a plot title.
plt.suptitle('Antenna '+str(antID)+', scan '+str(tgtScan))

# Color-map for WVR-channels/ATM-SPWs 0 - 3.
csline={0:'-r',1:'-g',2:'-b',3:'-k'}

#- Make the *analytical* plots - least-square residuals etc.

for spw,eqRow in spw2row.items():
  # Residuals/errors as temperature in Kelvins.
  plt.plot(wTime,1000.0*errVec[eqRow],csline[spw])

#- Save the plots to a file and close.

plt.savefig(plotDir+'/errs-scan_'+scID+'_ant_'+antIDstr+'.png')
plt.show()
plt.close()


#-- Make the full *solution* plots: PWV(t) and Tsun_illumination(t).

# Add variations to the mean (scan-averaged) values.
# Scale the solar illumination temperature from kiloKelvins back to Kelvins.
pwv=w0+dW #mm
Tsun=1000.0*(Tsun0+dB) # K

# Set the size of the plot
plt.rcParams["figure.figsize"] = (10,5)
# Make slow-WV and fast-WV plots as subplots (horizontally)
ff,(sp1,sp2)=plt.subplots(1,2)
# Add the antenna name as a plot title.
plt.suptitle('Antenna '+str(antID)+', scan '+str(tgtScan))

# Left panel: Fast variations - apparent target brightness
sp1.plot(wTime,Tsun,'-r')

# Right panel: Slow variations - total PWV (compression + humidity).
sp2.plot(wTime,pwv,'-g')

plt.savefig(plotDir+'/dWdB/scan_'+scID+'_ant_'+antIDstr+'.png')
plt.show()
plt.close()


#-- Show, how much the instrumental variables G and P_inst have changed.

print("Corrected relative gains [dimension-less]: ", Grw)
print("Corrected instrumental offsets [K]:", 1000.0*Piw)
```

```
  # Print total quadratic error for the variables.
  print("\n======= Xi-squares =======     SCI: ",xiSci,
        ", ACD:",xiACD,", Array-mean: ",xiTbM,
        " | Total-per-equation:", xiSquare,
        "\n-------> Estimated total error per equation:",
        np.sqrt(xiSquare)*1000.0,"K -------\n\n")


  #-- Export the resulting data in the 'conforming' format.

  return {'time':wTime,'pwv':pwv,'T_B':Tsun,'J':J_wb,'C':CC}


#--------------------------------------------------------------------------------
def dWdLs4_weights(vis,myAtm,antID,tgtScan,
                   flaggedWVRchannels=[],ssType='',solIllumination=False,
                   myMsmd=None,acdLoads=None):


  """
  Similar to dBdWdG2_weights(). We are varying the *calWeight* and *mWeight*
  parameters (one at a time, selected by commenting/ucommenting the line
  with *theWeight* below in the code) in order to find their optimized values,
  which somehow minimize the combined error (residuum) of natural, instrumental,
  and mean-array parts of our grand-system of equations.

  Implementation: Basically, we use the dWdLs4(), where the main solution-finder
  iteration loop is nested in an outer loop of varying calWeight/mWeight. Here,
  some explaining comments are abbreviated - see dWdLs4() for full detail.
  Moreover, the plots of all time-dependent quantties are omitted.
  """

  #--- Static parameters

  #- I/O

  # Input data from the ATM scan preceding the target scan.
  atmScan=tgtScan-1

  #- Tools, global data

  # Flags used to check, whether we should close the tool later.
  ownMsmd=False

  # Open own tools if 'global' not provided
  if (not myMsmd):
    # Open the msmd tool.
    myMsmd=msmdtool()
    myMsmd.open(vis)
    ownMsmd=True

  # Create our dictionary with ACD load temperatures (ambient, hot) if needed.
  if (not acdLoads):
    acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                    warnIfNoLoadTemperatures = True,
                                    mymsmd=myMsmd)

  #- Antenna
```

```
antNames=myMsmd.antennanames()
myAntName=antNames[antID]
antType=myAntName[:2]

# Map the antenna efficiencies by antenna type. As per investigation from
# merged T_sun temperatures form 2023 and 2026 data (project_22_25_Tb.ods).
eta_l_map={'CM':0.941,'DA':0.947,'DV':0.956}
eta_F_map={'CM':0.891,'DA':0.897,'DV':0.906}

# Rear-spillover + ohmic antenna efficiency - see project_22_25_Tb.ods
eta_l=eta_l_map[antType]
# Forward-spillover antenna efficiency - see project_22_25_Tb.ods
eta_F=eta_F_map[antType]

# We have two exception that seem to have systematically lower
# antenna efficiencies: DA64 and DV08 - see project_22_25_Tb.ods
if myAntName=='DA64':
  eta_l=0.929
  eta_F=0.880
if myAntName=='DV08':
  eta_l=0.943
  eta_F=0.894

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
Thot = 0.001*acdLoads[antID][atmScan]['hot']

# How strongly we insist on calibration solution G=1.0, P_inst=0.0.
# See comment in dWdBdG2().
# The used value has been found experimentally via dWdLs4_weights().
# Search mark in dWdLs4_weights(): CW_2W
calWeight=20.0

#- Target

# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.300  # kiloKelvins!!!

# Solar radius.
Rsun=696000 # km


# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.
```

```
"""
# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for July 30 2026Band 5 observation.
'''
Target:       RA=08 37 18.01 DEC=+18 31 23.0
Sun center:   RA=08 38 15.79 DEC=+18 28 47.2
-----------------------------------------------
diff (arcsec, incl. cos(delta)): 822   155.8
'''
x_h=822.0 # arcsec
y_h=155.8  # arcsec

# Earth-Sun distance for July 30 2026 Band 5 observation.
Dsun=1.01508611721886*149597870.700 # km

# *** End of observation-specific part.
"""

 # Apparent angular solar radius in arcsec.
 r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

 # mu=cos(theta). Used for limb-brightenning correction below.
 mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

 # b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
 # in Alissandrakis et al. (2020). Scaled to kiloKelvins!
 b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

 # Brightness temperature corrected to limb brightenning - see
 # Alissandrakis et al. (2020).
 Tsun00=TsunCenter+0.5*b_2*np.log(mu) # kK

 # Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
 f_LO=1.98e11

 # Solar illumination. May be either zero or unbound, depending on the
 # solIllumination parameter setting (T/F).
 T_tgt=0.0

 if (solIllumination):
   # Do not apply Tsun=0 condition, we allow for unbound illumination.
   # If mWeight is set to zero, the T_tgt value itself is irrelevant.
   mWeight=0.0
 else:
   # We are pointing far off the Sun. Enforce zero illumination by adding
   # the T_tgt=0.0 equation with the appropriate weight.
   # Search mark in dWdLs4_weights(): MW_2W
   mWeight=1.0

 #- Atmosphere

 # How many ATM (i.e. in the model) spws do we have? In this case it is
 # the same as the number of the WVR channels.
 nAtmSPWs=myAtm.getNumSpectralWindows()

 # Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
 pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']
```

```
# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,'mm @airMass = ',airMass)
print('Estimated T_B(full Sun) = ',1000.0*Tsun00,'K') # scale to Kelvins.
print('.......')

# The MS metadata has been read-out, we may close the (local) tools.
if (ownMsmd): myMsmd.done()

#-- Extract WVR raw temperatures.

# Raw WVR Tbs
wTime,Twvr=wvr_dataRaw(vis,tgtScan,antID,ssType)

# Average over time.
TwvrMean=np.mean(Twvr,axis=1)
# Stddevs per channel.
TwvrRMS=np.std(Twvr,axis=1)

#-- Flag and remove clear outliers.

# Find all array indices, for which data in at least one of the valid
# WVR channels are out of 3 sigma interval.
validWvrChans=np.delete(range(nAtmSPWs),flaggedWVRchannels)

for SPW in validWvrChans:
  # Outliers ( > 3 sigma) in the current SPW.
  badData=np.flatnonzero(abs(Twvr[SPW]-TwvrMean[SPW])>3.0*TwvrRMS[SPW])
  if (badData.size):
    print('Flagged in WVR channel no.',SPW,': Data in times ',wTime[badData],
          ', bad temperatures T=',Twvr[SPW,badData])

    # Delete data at bad-data indices in *all* WVR channels, so that the data
    # remain homogeneous in time. Make the time axis conformant.
    Twvr=np.delete(Twvr,badData,axis=1)
    wTime=np.delete(wTime,badData)
#--> endfor(valid WVR chans)

#-- Find dimensions of the system.

# How many time samples are in this scan (after flagging)?
nSamples=Twvr.shape[1]

# How many equations do we actually have per block of 1 time-sample?
# Some WVR channels may be flagged (typically, the channel 0 is saturated).
eqBlockSize=nAtmSPWs-len(flaggedWVRchannels)

# Pre-calculated frequently repeating factor: Inverse number of time samples.
nsInv=1.0/nSamples

#- Relative gains and instr. offsets - one per WVR channel, no polarization.
```

```
# Gains to be initialised to their expected value of 1.0.
Grw=np.ones(nAtmSPWs)
# Offsets should be close to 0.0.
Piw=np.zeros(nAtmSPWs)

#- Reconstruct the sub-channel frequencies for our ATM SPWs.
#  We need them for proper scaling of the solar-illumantion spectrum.

# We have the same number of sub-channels for all WVR-channel ATM SPWs.
# Fortunatelly, because Python is incapable of creating a vector of vectors,
# which would have different sizes, only rectangular matrix is OK :-(
nChans=myAtm.getNumChan(0)

# Frequencies for sub-channels in our ATM SPWs.
wvrFreqs=np.zeros((nAtmSPWs,nChans))

for wC in range(nAtmSPWs):

    fMin=myAtm.getMinFreq(wC)['value'][0]*1.0e9 # GHz to Hz
    df=myAtm.getChanSep(wC)['value'][0]*1.0e6   # MHz to Hz

    wvrFreqs[wC]=fMin+df*np.arange(nChans)

#-- Arrays for calculations of the matrix and the RHS-vector components
#   of the linearized system that depend just on the models (ATM + quiet Sun).
#   Note: In this case, when we have 1-to-1 correspondence between the model
#   and data SPWs, we actually need not to store the per-spw values to arrays.
#   But we keep the structure of sci-SPW approach in dWdB_TBs() and dWdBdG().

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,nChans))
dTdW=np.zeros((nAtmSPWs,nChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,nChans))
tau_1=np.zeros((nAtmSPWs,nChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/Tsun00*np.log(wvrFreqs/f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
# We used those as weights in calculation of epsilon - the normalised
# inter-iteration change of operation-point at the very end of each iteration.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin
dG_max=0.001       # Relative scale dG/G
dTinst_max=0.001   # dPinst/G in kiloKelvins
```

```
#+++++++++++++  Loop over reasonable cal/mWeights ++++++++++


# Define the calWeight/mWeight iteration ranges.
wItSize=101


# In semi-log scale. Use plt.semilogx(xAxis,yAxis) for plot.
#exponents=0.05*(np.arange(wItSize)-(wItSize-1)/2)
exponents=0.05*(np.arange(wItSize))-2
weights=pow(10,exponents)


'''
wLow=3.0
wHigh=33.0
weights=wLow+(wHigh-wLow)/(wItSize-1)*np.arange(wItSize)
'''



#- We study influence of selected (iterated) calWeigth/mWeight to:

# 1. Time-averaged PWV
iPWV=np.zeros(wItSize)

# 2. Time-averaged solar-illumination brightness temperature.
iTSun=np.zeros(wItSize)

# 3. Number of iterations required for convergence.
iNIT=np.zeros(wItSize)

# 4. Errors separated by 'sub-systems' of equations: science (s), ACD cal (a),
#    and mean-array T_tgt (m).
iXis=np.zeros(wItSize)
iXia=np.zeros(wItSize)
iXim=np.zeros(wItSize)

# 5. Relative updates of gains and P_inst.
iGG=np.zeros((wItSize,nAtmSPWs))
iPP=np.zeros((wItSize,nAtmSPWs))


#-- The main weight-iteration loop.
for wi in range(wItSize):

  theWeight=float(weights[wi])

  # Which weight do we iterate: calWeight or mWeight? The 2nd remains static.
  # Select by comment/uncomment.

  #calWeight=theWeight
  mWeight=theWeight

  # Reset the starting values for iterations.
  w0=pwv0
```

```
# From dWdLs4() we estimate the solar contamination ~100K = 0.1kK in the
# on-source subscans. The off-source should be contamination-free.
if (ssType=='on'):
  Tsun0=0.1 # kiloKelvins
else:
  Tsun0=0.0 # no illumination


#- Reset gains/P_inst back to their pure-ACD values.

# Gains to be initialised to their expected value of 1.0.
Grw=np.ones(nAtmSPWs)
# Offsets should be close to 0.0.
Piw=np.zeros(nAtmSPWs)


# Iteration counter.
itCount=0
# Normalized inter-iteration change of operation point - its initial value.
# Anything bigger than 1.0
epsilon=10.0

#-- Solution-finding iterations: Minimize the mean value of variations.

while( (epsilon>1.0) and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample

  # N-blocks: Jacobian-part for the [dW,dT] 'natural' variables.
  J_wb=np.zeros((eqBlockSize,2)) # 2 vars repeating in non-flagged wvrChans
  # G-block matrix: J-part for [dG,dP_inst] 'instrumental' variables.
  J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each WVR channel.

  # A-block matrix: J-part for [dG,dP_inst] for WVR "calibration" equations.
  J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR channel, amb.
  J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR channel, hot.
  # M-block matrix: Jacobian part for [dW,dT] in array-mean eqs.
  # This matrix is constant and can be defined already here.
  J_M=np.array([[0.0, nsInv]])


  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))

  # RHS-vector 'appendix 1' - amb. & hot 'calibration powers' (time-avgd).
  CCA=np.zeros((eqBlockSize,1))
  CCH=np.zeros((eqBlockSize,1))

  # RHS-vector 'appendix 2' - array-mean values of the target T_B
  CCM=np.zeros((1,1))

  # Solution residua for each used WVR channel.
  resT={}
  # Mapping the WVR channel to the index of equation in the system.
  spw2row={}
```

```
#- Re-calculate ATM model with updated <PWV> and solar illumination.

# Set current estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

# Count the actual number of equations - some WVR channels may be flagged.
eqCount=0

#- Loop over 4 ATM spws - these directly correspond to 4 WVR channels
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Scale to kiloKelvins.
  TbsW0[atmSPW]=0.001*Tebb

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value']
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value']

  # In kiloKelvins/mm
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Transmissivity (spectral).
  trans[atmSPW]=np.exp(-opacity)
  # Wet opacity per 1mm PWV scaled by airmass.
  tau_1[atmSPW]=(airMass/w0)*tau_wet

  #- Model (_M) factors.

  # Tebb and its slope.
  TbsW0_M=TbsW0[atmSPW]
  dTdW_M=dTdW[atmSPW]

  # Transmissivity and unit-PWV opacity.
  trans_M=trans[atmSPW]
  tau_1M=tau_1[atmSPW]

  # Solar-flux model spectrum (normalized).
  SF_M=SF[atmSPW]

  #- Model + antenna efficiencies combined.

  # Sensitivity factor for PWV.
  SW=(-eta_l*eta_F*Tsun0*np.mean(trans_M*tau_1M*SF_M)
```

```
      +(1.0-eta_F+eta_F*eta_l)*np.mean(dTdW_M))*Grw[atmSPW]

  # Sensitivity factor for the (apparent) solar illumination temperature.
  SB=eta_l*eta_F*np.mean(trans_M*SF_M)*Grw[atmSPW]

  # Model T_sky as viewed by the receiver with uniform spectral gain.
  Tsky=(eta_l*eta_F*Tsun0*np.mean(trans_M*SF_M)
        +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M)
        +eta_F*(1.0-eta_l)*Tamb)



  # Build the equations only for the non-flagged channels.
  if (atmSPW not in flaggedWVRchannels):

    #- Build the matrix: Natural (dW & dB sensitivities) block.

    # Set the sensitivity factor(s) for the current ATM SPW as a row
    # in the block/tile for the linearized-system Jacobian matrix.
    J_wb[eqCount]=np.array([SW,SB])


    #- Build the matrix: Instrumental (Grw & Piw sensitivities) block.

    # We have to prepend a proper offset for the Grw and Piw vars as
    # they are independendt for each SPW and polarization.
    bOffset=2*eqCount

    J_gp[eqCount,bOffset]=1.0
    J_gp[eqCount,bOffset+1]=Tsky

    #- Build the matrix: ACD powers (amb & hot) 2-block.

    J_cdA[eqCount,bOffset]=1.0
    J_cdA[eqCount,bOffset+1]=Tamb

    # Hot-power sub-block.
    J_cdH[eqCount,bOffset]=1.0
    J_cdH[eqCount,bOffset+1]=Thot

    #- Build the RHS vector: WVR data

    # RHS: Twvr(t) with the stationary ATM-based part
    # subtracted. Rescale measured WVR temperatures  to kiloKelvin.
    dTw=0.001*Twvr[atmSPW]-Piw[atmSPW]-Grw[atmSPW]*Tsky

    # Set it as the RHS component.
    CC[eqCount]=dTw

    #- Ammend the RHS vector: Calibration data.

    # Ambient-load power with its working-point part subtracted.
    # Note: Unlike Twvr, the Tamb and Thot are in kiloKelvins already.
    CCA[eqCount]=Tamb-Piw[atmSPW]-Grw[atmSPW]*Tamb
    # Hot-load power with its working-point part subtracted.
    CCH[eqCount]=Thot-Piw[atmSPW]-Grw[atmSPW]*Thot
```

```
      #- Diagnostics

      # Residua of sky temperature for mean PWV and Tsun (operation point).
      resT[atmSPW]=dTw.mean()

      # Mapping between ATM SPW and equation number.
      spw2row[atmSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',atmSPW,' WVR channel.')
    #--> endif(not flagged)

  #--> endfor(ATM SPWs)

  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged SPW:pol channels.")
    return False


  #-- Solution of the grand system of linearised equations. See dWdBdG2()
  #   for explanation.

  #- Ammend the RHS vector: Array-mean target brightness temperatures.

  # Effective expected target T_B with the model at work-point subtracted.
  # In our case, effective T_B is just the part of 'parasitic' illumination.
  # For far-off-Sun pointings T_tgt is set to zero. For near-Sun pointings
  # we use solIllumantion=True and Tsun0 is unbound because mWeight=0.
  CCM[0]=T_tgt-Tsun0

  #- Calculate the small-block matrices and their inversions needed for
  #  grand-system inversion.

  # 1. Natural (N-block) matrices.
  J_wb_T=np.transpose(J_wb)
  J_K=np.matmul(J_wb_T,J_wb)
  J_K_inv=np.linalg.inv(J_K)

  # 2. Gain/P_inst (G-block) matrices.
  J_gp_T=np.transpose(J_gp)

  # 3. ACD rows (A-block) matrices.
  # Calculate transpositions per sub-block (ambient/hot).
  # Need copy() here - see the same place in dWdBdG2().
  J_cdA_T=np.transpose(J_cdA).copy()
  J_cdH_T=np.transpose(J_cdH).copy()

  # Increase importance of the ACD rows by calibration weight.
  # Only the transposed A-block matrices should be scaled up!
  J_cdA_T*=calWeight
  J_cdH_T*=calWeight

  # Compose the full block and its transposition. The transposition to be
  # scaled up - see above. Or, alternativelly, it can be built as
```

```
    # np.hstack([J_cdA_T,J_cdH_T]) from already scaled-up transp. sub-blocks.
    J_cd=np.vstack([J_cdA,J_cdH]).copy()
    J_cd_T=calWeight*np.transpose(J_cd)

    # 4. Array-mean (M-block) related matrices. Incorporate mWeight into M^T,
    # like what we have done with calWeight for the A^T and H^T matrices.
    M_T=np.array([[0.0],
                  [nsInv*mWeight]])    # mWeight*M^T

    # Pre-calculated nSamples*mWeight*(M^T x M) matrix.
    # Like with A^T x A, the M^T x M product is scaled-up by mWeight just once
    # (no mWeight squared!). Moreover, we scale it up here yet by nSamples,
    # in order to avoid trivial nsInv*nSamples=1 in the following calcs.
    nsMTM=np.array([[0.0, 0.0],
                    [0.0, nsInv*mWeight]])

    # Frequently repeating blocks. NM_inv plays a role of modified J_K_inv.
    NM=J_K+nsMTM                # J_K+nSamples*mWeight*M^T x M
    NM_inv=np.linalg.inv(NM)    # NM^(-1)

    # Off-diagonal blocks of the inverted natural sub-matrix (Ainv).
    J_M_inv=nsInv*(NM_inv-J_K_inv)


    #- Base-matrix extensions.

    # Right and lower bands: Their (nEqs x 2) and (2 x nEqs) repeating blocks.
    Rb=np.matmul(J_wb_T,J_gp)
    Lb=np.transpose(Rb)

    # Lower-right corner
    LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

    #- Frequently repeating matrix products. Cf. with extMatInv() above.

    # Ainv x Rb and Lb x Ainv
    AinvRb=np.matmul(NM_inv,Rb)
    LbAinv=np.transpose(AinvRb)

    #- Schur complement and its inversion.
    SC=LRc-nSamples*np.matmul(Lb,AinvRb)
    SC_inv=np.linalg.inv(SC)

    #- Blocks of inverted grand-system matrix.

    # Start with the lower-left block as it repeats in upper-left block, too.
    Minv21=-np.matmul(SC_inv,LbAinv)

    # Upper-left block: Off-diagonal terms. The diagonal blocks differ just
    # by added J_K_inv.
    Minv11_OD=J_M_inv-np.matmul(AinvRb,Minv21)

    # Upper-right block.
    Minv12=np.transpose(Minv21)

    # Lower-right block.
    Minv22=SC_inv
```

```
    #-- Apply the inverted matrix components to the RHS vector.

    # Sum of the RHS along the time axis, keeping the SPW:POL dimension.
    # Preserve the number of matrix rows, just sum their contents, i.e. make
    # the single-column matrix.
    CCsum=np.reshape(CC.sum(axis=1),(-1,1))

    #- Form the RHS vector for the normal equations (K-form minimizer),
    #  i.e, multiply the raw data by JJ_T

    # Start with the natural-variables dW and dB part.
    C_K_wb=np.matmul(J_wb_T,CC)+np.matmul(M_T,CCM)   # 2 x nSamples matrix

    # Make the same also for its time-sum.
    C_K_wb_sum=np.matmul(J_wb_T,CCsum)+nSamples*np.matmul(M_T,CCM)    # 2 x 1

    # Follow with the instrumental part.
    C_K_gp=(np.matmul(J_gp_T,CCsum)       # 2*eqCount x 1
            +np.matmul(J_cdA_T,CCA)
            +np.matmul(J_cdH_T,CCH))

    #- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

    # Constant part of the natural-variables vector (2 x 1)
    nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)

    # Time-varying part of the natural-variables vector. It comes from the
    # addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
    nvVar=np.matmul(J_K_inv,C_K_wb)   # 2 x nSamples

    # Compose the complete natural variables vector.
    varNat=nvVar+nvConst

    # Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
    varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)

    #- Reshape the results - extract our elementary variables.

    # Seaparate the natural-vars components.
    # PWV variations.
    dW=varNat[0]
    # Illumination/brightness vars: In this approach we may have them, too.
    dB=varNat[1]


    # Reshape instrumental parameters to two rows (transpose): 1st is dP_inst,
    # the 2nd dG, one component per WVR channel.
    dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

    # Extract the shifts in the P_inst and G
    dPinst=dPG[0]
    dGinst=dPG[1]

    #- Analytics: Calculate the errs for the over-determined set of eqs.
    #  See corresponding part of dWdBdG2() for explanation of err estimation.

    # err=f(x)-b: Sci-data part.
    errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
```

```
    # Xi square for the sci-data part, normalized per one equation.
    xiSci=np.sum(errVec*errVec)/(nSamples*eqBlockSize)

    # Xi Squares per SPW:Pol channel. Sum only over times.
    xiSciPerChan=np.sum(errVec*errVec,axis=1)
    # Its mean and RMS.
    xscMean=xiSciPerChan.mean()
    xscRMS=xiSciPerChan.std()
    # Find outliers ( > 3 sigma). The badData array contains indices of
    # those equations, where the agreement between model and observation is
    # exceptionaly weak.
    badData=np.flatnonzero(abs(xiSciPerChan-xscMean)>3.0*xscRMS)

    # Extract the WVR channels with bad data, if we have some.
    # We do not have an inverse map to spw2row, so we have to traverse.
    for wvrChan,eqRow in spw2row.items():
      print('WVR chan '+str(wvrChan)+': Error:',xiSciPerChan[eqRow])
      if eqRow in badData: print('\t\t!!! Bad data in WVR chan',str(wvrChan),
                                 ':',(xiSciPerChan[eqRow]-xscMean)/xscRMS,
                                 'sigma from the mean.')

    # err=f(x)-b: Calibration-data part.
    errVecACDamb=np.matmul(J_cdA,varInst)-CCA
    errVecACDhot=np.matmul(J_cdH,varInst)-CCH
    # Xi square for the cal-data part.
    xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)
    # Normalized per one equation.
    xiACD/=eqBlockSize

    # err=f(x)-b: Array-mean T_B (illumination) part.
    errVecMeanTb=-CCM
    # Xi square for the averaged illumination part.
    xiTbM=np.sum(errVecMeanTb*errVecMeanTb)

    # Grand-total Xi-square normalized to the number of equations for
    # the natural/science, ACD, and mean-array sub-blocks.
    xiSquare=xiSci+xiACD+xiTbM

    #- All variations should tend to zero means.

    dW_mean=dW.mean()
    dB_mean=dB.mean()

    # Update stat. part of PWV and illumination by avgs of their variations.
    w0+=dW_mean
    Tsun0+=dB_mean

    # Update also the Piw and Grw by the found differences.
    # Find the maximum inter-iteration change for convergence diagnostics.
    dT_inst_sup=-1.0
    dG_rel_sup=-1.0

    for wvrChan,eqCount in spw2row.items():
      # Update the P_inst and G.
      Piw[wvrChan]+=dPinst[eqCount]
      Grw[wvrChan]+=dGinst[eqCount]

      # Change of inst. power expressed in kiloKelvins.
```

```
      dT_inst=abs(dPinst[eqCount]/Grw[wvrChan])
      if (dT_inst > dT_inst_sup): dT_inst_sup=dT_inst

      # Change of gain in the relative scale.
      dG_rel=abs(dGinst[eqCount]/Grw[wvrChan])
      if (dG_rel > dG_rel_sup): dG_rel_sup=dG_rel
    #--> endfor(non-flagged WVRchannels/atmSPWs)


    #-- Convergence diagnostics / Iteration control.

    # Calculate the normalized inter-iteration change of operation point.
    # The factor 0.5 comes from sqrt(1/4) as we have 4 'orthogonal' variables.
    epsilon=0.5*np.sqrt((dW_mean/dwMean_max)**2+(dB_mean/dbMean_max)**2+
                        (dG_rel_sup/dG_max)**2+(dT_inst_sup/dTinst_max)**2)

    # Increment the iteration counter
    itCount+=1


    #- Diagnostic print-outs.

    # Updated scan-avgd PWV and sol. illumination after completed iteration.
    print('vvvvvvv\n After iteration no. ',itCount)
    print('Mean PWV for the scan,',tgtScan,' w0 = ',w0,"mm"
          ", mean Tsun(illumination) = ",1000.0*Tsun0,"K") # scale to Kelvins

    # Mean values of dW and dB fluctuations should tend to zero.
    print("dW mean: ",dW_mean)
    print("dB mean: ",dB_mean)

    # Sky-brightness residua should tend to zero, too.
    # Convert the spwId maps to np.arrays first. We have to go via 'list'.
    # Scale-up back to Kelvins (we calculate in kiloKelvins).
    resTArr=1000.0*np.array(list(resT.values()))

    print("res T: ", resT,"mean +/- RMS: ",resTArr.mean(),"+/-",resTArr.std())

    # Note: Here we measure the Pinst change directly in Kelvin(!), as nominal
    # Piw is 0. For gain, we need not to divide by Grw as it is nominally 1.
    print("\nInstrumental characteristics shifts during the iteration:")
    for wvrChan,eqCount in spw2row.items():
      print("WVR chanNum = ",wvrChan,"\tdP_I =",1000.0*dPinst[eqCount],"K",
            "\tdG/G =",dGinst[eqCount]*100,"%")

    print("Epsilon:",epsilon)

    print('^^^^^^^\n\n')

  #--> endwhile(non-zero-means) / End of solution-iterations loop.


  # Store the watched quantities in the array per-weight elements.

  iPWV[wi]=w0
  iTSun[wi]=1000.0*Tsun0 # Scale the result back to Kelvins.

  iNIT[wi]=itCount
```

```
    iXis[wi]=xiSci
    iXia[wi]=xiACD
    iXim[wi]=xiTbM

    iGG[wi]=100.0*(Grw-1.0)
    iPP[wi]=1000.0*Piw

  #--> endfor(range-of-weights) / End of the main loop.


  #-- Return the arrays of interest in depencence on calWeight.

  return {'NS':nSamples,'CW':weights,"PWV":iPWV,"Tsun":iTSun,
          "XiS":iXis,"XiA":iXia,"XiM":iXim,"NIT":iNIT,"dG":iGG,"dP":iPP}

#--------------------------------------------------------------------------------

def dWdLs3_calWeight(myAtm,antID,tgtScan,
                     flaggedWVRchannels=[],ssType='',solIllumination=True,
                     myMS=None,myMsmd=None,acdLoads=None):

  """
 --- OBSOLETTED ---


 Replaced/superseded by a new dWdLs4_weights() function.

  """

  #--- Static parameters

  #- Dirs

  # Source-data definition. Set static because we are MS-specific routine.
  vis='uid___A002_X110fd13_X1099e.ms'
  # ATM scan preceding the solar scan 9 is used always.
  atmScan=8

  # Output/diagnostics plots.
  plotDir='./wvr_decomp_G'


  #- Tools, global data

  # Flags used to check, whether we should close the tool later.
  ownMS=False
  ownMsmd=False

  # Open own tools if 'global' not provided
  if (not myMS):
    # Open the ms tool
    myMS=mstool()
    myMS.open(vis)
    ownMS=True

  if (not myMsmd):
    # Open the ms tool.
    myMsmd=msmdtool()
```

```
    myMsmd.open(vis)
    ownMsmd=True

# Create our dictionary with ACD load temperatures (ambient, hot) if needed.
if (not acdLoads):
    acdLoads=aU.getLoadTemperatures(vis, doplot = False,
                                    warnIfNoLoadTemperatures = True,
                                    mymsmd=myMsmd)

#- Target

# Initial guess of the solar brightness temperature.
# SD calibration script: Reference T_b for solar disc center @198GHz
TsunCenter=6.3  # kiloKelvins!!!

# Solar radius.
Rsun=696000 # km

# *** This part is observation specific - need to read ephemeris !!!

# Heliocentric coordinates for Dec 2023 Band 5 observation.
x_h=928.0 # arcsec
y_h=87.0  # arcsec

# Earth-Sun distance for Dec 2023 Band 5 observation.
Dsun=1.4723338439337283e8 # km

# *** End of observation-specific part.

# Apparent angular solar radius in arcsec.
r_sol=180.0*3600.0*Rsun/(Dsun*np.pi)

# mu=cos(theta). Used for limb-brightenning correction below.
mu=np.sqrt(1.0-(x_h**2+y_h**2)/r_sol**2)

# b_2=2.0*a_2, where a_2 is the temperature drop with frequency in Eq. (4)
# in Alissandrakis et al. (2020). Scaled to kiloKelvins!
b_2=-1.127 # = 2x (-563.5K) as per Table 2 in Alissandrakis et al. (2020).

# Brightness temperature corrected to limb brightenning - see
# Alissandrakis et al. (2020).
Tsun00=TsunCenter+0.5*b_2*np.log(mu) # kK

# Reference frequency for the solar spectrum: We take the LO frequency (in Hz)
f_LO=1.98e11

#- Antenna

# Rear-spillover + ohmic antenna efficiency (see Tab. 3 in White et al. 2017).
eta_l=0.95
# Forward-spillover antenna efficiency (Tab. 3 in White et al. 2017).
# WVR has, unlike the sci-receiver, on-axis recording, may have better eta_F.
eta_F=0.9

# Extract ambient & hot temperatures for the current antenna and scan.
# It determines the rear-spillover contribution to noise.
# Scaled to kiloKelvins!!!
Tamb = 0.001*acdLoads[antID][atmScan]['amb']
```

```
Thot = 0.001*acdLoads[antID][atmScan]['hot']

#- Atmosphere

# Initial guess of mean PWV: Scan-center interpolated ASDM_CALWVR value.
pwv0=getPWVforAnt(vis,tgtScan,antID,myMsmd)['pwvAvg']

# Set initial estimation of PWV to the ATM model.
myAtm.setUserWH2O(qa.quantity(pwv0,'mm'))

# Airmass for the solar scan.
airMass=myAtm.getAirMass()

# Print the initial estimates. Temperature to be scaled from kiloKelvins to K.
print('....... Initial estimate')
print('Mean PWV for scan,',tgtScan,' w0 = ',pwv0,'mm @airMass = ',airMass)
print('Estimated T_B(full Sun) = ',1000.0*Tsun00,'K') # scale to Kelvins.
print('.......')

#-- Extract WVR raw temperatures.

# Raw WVR Tbs
wTime,Twvr=wvr_dataRaw(vis,tgtScan,antID,ssType)

# Average over time.
TwvrMean=np.mean(Twvr,axis=1)

# How many time samples are in this scan?
nSamples=Twvr.shape[1]

# How many ATM (i.e. in the model) spws do we have? In this case it is
# the same as the number of the WVR channels.
nAtmSPWs=myAtm.getNumSpectralWindows()

# How many equations do we actually have per block of 1 time-sample?
# Some WVR channels may be flagged (typically, the channel 0 is saturated).
eqBlockSize=nAtmSPWs-len(flaggedWVRchannels)


#- Reconstruct the sub-channel frequencies for our ATM SPWs.
#  We need them for proper scaling of the solar-illumantion spectrum.

# We have the same number of sub-channels for all WVR-channel ATM SPWs.
# Fortunatelly, because Python is incapable of creating a vector of vectors,
# which would have different sizes, only rectangular matrix is OK :-(
nChans=myAtm.getNumChan(0)

# Frequencies for sub-channels in our ATM SPWs.
wvrFreqs=np.zeros((nAtmSPWs,nChans))

for wC in range(nAtmSPWs):

    fMin=myAtm.getMinFreq(wC)['value'][0]*1.0e9 # GHz to Hz
    df=myAtm.getChanSep(wC)['value'][0]*1.0e6   # MHz to Hz

    wvrFreqs[wC]=fMin+df*np.arange(nChans)

#-- Arrays for calculations of the matrix and the RHS-vector components
```

```
#   of the linearized system that depend just on the models (ATM + quiet Sun).
#   Note: In this case, when we have 1-to-1 correspondence between the model
#   and data SPWs, we actually need not to store the per-spw values to arrays.
#   But we keep the structure of sci-SPW approach in dWdB_TBs() and dWdBdG().

# Atmosphere (+CMB) brightness temperature for PWV=w0 and its slope.
# Place-holders, to be calculated inside iterations.
TbsW0=np.zeros((nAtmSPWs,nChans))
dTdW=np.zeros((nAtmSPWs,nChans))

# Transmissivity and unit-PWV opacity. Place-holders.
trans=np.zeros((nAtmSPWs,nChans))
tau_1=np.zeros((nAtmSPWs,nChans))

# Average Solar-Flux spectrum: QSR model normalised to 1.0 at f=f_LO.
SF=1.0+b_2/Tsun00*np.log(wvrFreqs/f_LO)


#--- Solve the linearised set of equations for dW and dB variations.
#    Iterate until they have negligible mean value (scan-average).

#-- Iteration setup

# Half dw for the estimation of the dTsky/dw by centerred difference.
dws=0.05

# Stoppers for iterative solution: We should get less then these values.
dwMean_max=0.01    #mm
dbMean_max=0.001   # kiloKelvin



#+++++++++++++++ Loop over reasonable calWeights - experimental +++++++++++++++


# Scan-averaged PWV for a given calWeight
PWV=np.zeros(900)
# Scan-averaged solar illumination T_B for a given calWeight
TSil=np.zeros(900)

# Number of iterations required for convergence
NIT=np.zeros(900)
# Xi-square: Natural part
xis=np.zeros(900)
# Xi-square: Instrumental part
xia=np.zeros(900)

for cwi in range(900):
  # Set calWeight.
  calWeight=1.0+0.05*cwi
  #- Relative gains and instr. offsets - one per WVR channel, no polarization.

  # Gains to be initialised to their expected value of 1.0.
  Grw=np.ones(nAtmSPWs)
  # Offsets should be close to 0.0.
  Piw=np.zeros(nAtmSPWs)

  # Initial values for the dW and dL means - anyting bigger then the stoppers.
```

```
dW_mean=0.5
dB_mean=0.5

# Iteration counter.
itCount=0


# Set the starting values for iterations.
w0=pwv0
# From dWdLs() we estimate the solar contamination ~100K = 0.1kK in the
# on-source subscans. The off-source shoud be contamination-free.
if (ssType=='on'):
  Tsun0=0.1 # kiloKelvins
else:
  Tsun0=0.0 # no illumination

#-- Iterations: Minimize the mean value of variations.

while(((abs(dW_mean) > dwMean_max) or (abs(dB_mean) > dbMean_max))
      and (itCount< 20)):

  #-- Prepare/reset the linearized system of equations.

  #- Repeating blocks/tiles for each time-sample


  # N-blocks: Jacobian-part for the [dW,dT] 'natural' variables.
  if (solIllumination):
    # Solve for both dW and dB - we allow for varying solar illumination.
    J_wb=np.zeros((eqBlockSize,2)) # 2 vars repeating in non-flagged wvrc
  else:
    # Solar illumination should be zero, solve for dW solely.
    J_wb=np.zeros((eqBlockSize,1)) # 1 var repeating in non-flagged wvrc

  # G-block matrix: J-part for [dG,dP_inst] 'instrumental' variables.
  J_gp=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per each WVR chan

  # A-block matrix: J-part for [dG,dP_inst] for WVR "calibration" eqs
  J_cdA=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR c, amb.
  J_cdH=np.zeros((eqBlockSize,2*eqBlockSize)) # 2 vars per WVR c, hot.


  # RHS vector for all time samples stacked as a matrix.
  CC=np.zeros((eqBlockSize,nSamples))

  # RHS-vector 'appendix' - amb. & hot 'calibration powers' (time-avg)
  CCA=np.zeros((eqBlockSize,1))
  CCH=np.zeros((eqBlockSize,1))

  # Solution residua for each used WVR channel.
  resT={}
  # Mapping the WVR channel to the index of equation in the system.
  spw2row={}


  #- Re-calculate ATM model with updated <PWV>

  # Set current estimation of PWV to the ATM model.
```

```
myAtm.setUserWH2O(qa.quantity(w0,'mm'))

# Count the actual number of equations - some WVR channels may be flagged.
eqCount=0

#- Loop over 4 ATM spws - these directly correspond to 4 WVR channels
for atmSPW in range(nAtmSPWs):

  #- Model value of the sky/atm brightness with current pwv.
  Tebb=myAtm.getTebbSkySpec(atmSPW, qa.quantity(w0,'mm'))[1]['value']
  # Scale to kiloKelvins.
  TbsW0[atmSPW]=0.001*Tebb

  #- Slope (centered difference) of the atm brightness with current pwv.
  TebbH=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0+dws,'mm'))[1]['value']
  TebbL=myAtm.getTebbSkySpec(atmSPW,
                             qa.quantity(w0-dws,'mm'))[1]['value']

  # In kiloKelvins/mm
  dTdW[atmSPW]=0.001*(TebbH-TebbL)/(2.0*dws)

  #- Opacities & transmisivity.

  # Dry zenith opacity
  tau_dry = np.array(myAtm.getDryOpacitySpec(atmSPW)[1])
  # Water-vapour related 'wet' opacity
  tau_wet = np.array(myAtm.getWetOpacitySpec(atmSPW)[1]['value'])
  # Total opacity (spectral).
  opacity=airMass*(tau_wet+tau_dry)

  # Transmissivity (spectral).
  trans[atmSPW]=np.exp(-opacity)
  # Wet opacity per 1mm PWV scaled by airmass.
  tau_1[atmSPW]=(airMass/w0)*tau_wet


  #- Model (_M) factors.

  # Tebb and its slope.
  TbsW0_M=TbsW0[atmSPW]
  dTdW_M=dTdW[atmSPW]

  # Transmissivity and unit-PWV opacity.
  trans_M=trans[atmSPW]
  tau_1M=tau_1[atmSPW]

  # Solar-flux model spectrum (normalized).
  SF_M=SF[atmSPW]

  #- Model + antenna efficiencies combined.

  # Sensitivity factor for PWV.
  SW=(-eta_l*eta_F*Tsun0*np.mean(trans_M*tau_1M*SF_M)
      +(1.0-eta_F+eta_F*eta_l)*np.mean(dTdW_M))*Grw[atmSPW]

  # Sensitivity factor for the (apparent) solar illumination temperature.
  SB=eta_l*eta_F*np.mean(trans_M*SF_M)*Grw[atmSPW]
```

```
    # Model T_sky as viewed by the receiver with uniform spectral gain.
    Tsky=(eta_l*eta_F*Tsun0*np.mean(trans_M*SF_M)
          +(1.0-eta_F+eta_F*eta_l)*np.mean(TbsW0_M)
          +eta_F*(1.0-eta_l)*Tamb)



    #- Build the equations only for the non-flagged channels.
    if (atmSPW not in flaggedWVRchannels):

      #- Build the matrix: Natural (dW & dB sensitivities) block.

      # Set the sensitivity factor(s) for the current ATM SPW as a row
      # in the block/tile for the linearized-system Jacobian matrix.
      if (solIllumination):
        # Solve for both dW & dB independently.
        J_wb[eqCount]=np.array([SW,SB])
      else:
        # Only dW is allowed to vary.
        J_wb[eqCount]=np.array([SW])


      #- Build the matrix: Instrumental (Grw & Piw sensitivities) block.

      # We have to prepend a proper offset for the Grw and Piw vars as
      # they are independendt for each SPW and polarization.
      bOffset=2*eqCount

      J_gp[eqCount,bOffset]=1.0
      J_gp[eqCount,bOffset+1]=Tsky

      #- Build the matrix: ACD powers (amb & hot) 2-block.

      J_cdA[eqCount,bOffset]=1.0
      J_cdA[eqCount,bOffset+1]=Tamb

      # Hot-power sub-block.
      J_cdH[eqCount,bOffset]=1.0
      J_cdH[eqCount,bOffset+1]=Thot


      #- Build the RHS vector: WVR data

      # RHS: Twvr(t) with the stationary  ATM-based part
      # subtracted. Rescale Twvr to kiloKelvin.
      dTw=0.001*Twvr[atmSPW]-Tsky

      # Set it as the RHS component.
      CC[eqCount]=dTw

      #- Ammend the RHS vector: Calibration data.

      # Ambient-load power with its working-point part subtracted.
      CCA[eqCount]=Tamb-Piw[atmSPW]-Grw[atmSPW]*Tamb
      # Hot-load power with its working-point part subtracted.
      CCH[eqCount]=Thot-Piw[atmSPW]-Grw[atmSPW]*Thot
```

```
      #- Diagnostics

      # Residua of sky temperature for mean PWV and Tsun (operation point).
      resT[atmSPW]=dTw.mean()

      # Mapping between ATM SPW and equation number.
      spw2row[atmSPW]=eqCount

      eqCount+=1

    else: print('Ignoring flagged ',atmSPW,' WVR channel.')
    #--> endif(not flaggged)

  #--> endfor(ATM SPWs)

  # Now, the eqCount should be the same as eqBlockSize. Just check it.
  if (eqCount != eqBlockSize):
    print("******* Ooops, something is wrong !*******")
    print("   The expected number of equations does not correspond to the")
    print("   number of measured and non-flagged WVR channels.")
    return False



  #-- Solution of the system of linearised equations. See dWdBdG() for
  #   explanation.

  #- Calculate the small-block matrices and their inversions needed for
  #  grand-system inversion.

  # Natural (N-block) matrices.
  J_wb_T=np.transpose(J_wb)
  J_K=np.matmul(J_wb_T,J_wb)
  J_K_inv=np.linalg.inv(J_K)

  # Gain/P_inst (G-block) matrices.
  J_gp_T=np.transpose(J_gp)

  # ACD rows (A-block) matrices.
  # Calculate transpositions per sub-block (ambient/hot).
  J_cdA_T=np.transpose(J_cdA)
  J_cdH_T=np.transpose(J_cdH)

  # Increase importance of the ACD rows by calibration weight.
  # Only the transposed A-block matrices should be scaled up!
  J_cdA_T*=calWeight
  J_cdH_T*=calWeight

  # Compose the full block and its transposition. The transposition to be
  # scaled up - see above. Or, alternativelly, it can be built as
  # np.hstack([J_cdA_T,J_cdH_T]) from already scaled-up transposed sub-blks.
  J_cd=np.vstack([J_cdA,J_cdH])
  J_cd_T=calWeight*np.transpose(J_cd)

  #- Base-matrix extensions.

  # Right and lower bands: Their (nEqs x 3) and (3 x nEqs) repeating blocks.
```

```
    Rb=np.matmul(J_wb_T,J_gp)
    Lb=np.transpose(Rb)

    # Lower-right corner
    LRc=nSamples*np.matmul(J_gp_T,J_gp)+np.matmul(J_cd_T,J_cd)

    #- Frequently repeating matrix products. Cf. with extMatInv() above.

    # Ainv x Rb and Lb x Ainv
    AinvRb=np.matmul(J_K_inv,Rb)
    LbAinv=np.transpose(AinvRb)

    #- Schur complement and its inversion.
    SC=LRc-nSamples*np.matmul(Lb,AinvRb)
    SC_inv=np.linalg.inv(SC)

    #- Blocks of inverted grand-system matrix.

    # Start with the lower-left block as it repeats in upper-left block, too.
    Minv21=-np.matmul(SC_inv,LbAinv)

    # Upper-left block: Off-diagonal terms. The diagonal blocks differ just
    # by added J_K_inv.
    Minv11_OD=-np.matmul(AinvRb,Minv21)

    # Upper-right block.
    Minv12=np.transpose(Minv21)

    # Lower-right block.
    Minv22=SC_inv

    #-- Apply the inverted matrix components to the RHS vector.

    # Sum of the RHS along the time axis, keeping the SPW:POL dimension.
    # Preserve the number of matrix rows, just sum their contents, i.e. make
    # the single-column matrix.
    CCsum=np.reshape(CC.sum(axis=1),(-1,1))

    #- Form the RHS vector for the normal equations (K-form minimizer),
    #  i.e, multiply the raw data by JJ_T

    # Start with the natural-variables dW and dB part.
    C_K_wb=np.matmul(J_wb_T,CC)        # 2 x nSamples matrix

    # Make the same also for its time-sum.
    C_K_wb_sum=np.matmul(J_wb_T,CCsum)    # 2 x 1

    # Follow with the instrumental part.
    C_K_gp=(np.matmul(J_gp_T,CCsum)        # 2*eqCount x 1
            +np.matmul(J_cdA_T,CCA)
            +np.matmul(J_cdH_T,CCH))

    #- Apply blocks of the inverted JJ_K^(-1) matrix to the C_K RHS vector.

    # Constant part of the natural-variables vector (2 x 1)
    nvConst=np.matmul(Minv11_OD,C_K_wb_sum)+np.matmul(Minv12,C_K_gp)

    # Time-varying part of the natural-variables vector. It comes from the
```

```
    # addition to the Minv11 diagonal-terms (see comment above at Minv11_OD).
    nvVar=np.matmul(J_K_inv,C_K_wb)   # 2 x nSamples

    # Compose the complete natural variables vector.
    varNat=nvVar+nvConst

    # Instrumental variables. The result is of 2*eqBlockSize x 1 shape.
    varInst=np.matmul(Minv21,C_K_wb_sum)+np.matmul(Minv22,C_K_gp)


    #- Reshape the results - extract our elemenytary variables.

    # Seaparate the natural-vars components.
    # PWV variations present always.
    dW=varNat[0]

    # dB variations only in case of solar illum., otherwise they are zeros
    if (solIllumination):
      dB=varNat[1]
    else:
      dB=np.zeros(nSamples)


    # Reshape instrumental parameters to two rows (transpose): 1st is dP_i,
    # the 2nd dG, one component per WVR channel.
    dPG=np.transpose(np.reshape(varInst,(eqBlockSize,2)))

    # Extract the shifts in the P_inst and G
    dPinst=dPG[0]
    dGinst=dPG[1]

    #- Analytics: Calculate the errors for the over-determined set of eqs.

    # err=A.x-b: Sci-data part.
    errVec=np.matmul(J_wb,varNat)+np.matmul(J_gp,varInst)-CC
    # Xi square for the sci-data part.
    xiSci=np.sum(errVec*errVec)


    # err=A.x-b: Calibration-data part.
    errVecACDamb=np.matmul(J_cdA,varInst)-CCA
    errVecACDhot=np.matmul(J_cdH,varInst)-CCH
    # Xi square for the cal-data part.
    xiACD=np.sum(errVecACDamb*errVecACDamb)+np.sum(errVecACDhot*errVecACDhot)

    # Grand-total Xi-square normalized to the number of equations (sci/acd).
    xiSquare=xiSci/nSamples+xiACD/(calWeight**2)


    #- All variations should tend to zero means.

    dW_mean=dW.mean()
    dB_mean=dB.mean()

    # Update stationary part of PWV and sol T_B  by means of their variations.
    w0+=dW_mean
    Tsun0+=dB_mean
```

```
      # Update also the Piw and Grw by the found differences.
      for atmSPW,eqCount in spw2row.items():
        # Update the P_inst and G.
        Piw[atmSPW]+=dPinst[eqCount]
        Grw[atmSPW]+=dGinst[eqCount]
      #--> endfor(valid WVR chans)


      #- Convergence diagnostics.

      itCount+=1

      # Updated scan-averaged PWV and sol. illum. after completed iteration.
      print('....... After iteration no. ',itCount)
      print('Mean PWV for the scan,',tgtScan,' w0 = ',w0,"mm"
            ", mean Tsun(illumination) = ",1000.0*Tsun0,"K") # scale to Kelvins

      # Mean values of dW and dB fluctuations should tend to zero.
      print("dW mean: ",dW_mean)
      print("dB mean: ",dB_mean)

      # Sky-brightness residua should tend to zero, too.
      # Convert the spwId maps to np.arrays first. We have to go via 'list'.
      resTArr=np.array(list(resT.values()))

      print("res T: ", resT,"mean +/- RMS: ",resTArr.mean(),"+/-",resTArr.std())

      print('.......')

    #--> endwhile(non-zero-means) / end of main iteration loop.

    PWV[cwi]=w0

    TSil[cwi]=Tsun0

    NIT[cwi]=itCount
    xis[cwi]=xiSci
    xia[cwi]=xiACD

  #--> endfor(range-of-calWeights)

  # Range of calWeights
  CWS=1.0+0.05*np.arange(900.0)

  #-- Return the arrays of interest in depencence on calWeight.

  return {'CW':CWS,"PWV":PWV,"Ts":TSil,"XiS":xis,"XiA":xia,"NIT":NIT}


#-----------------------------------------------------------------------------

def pwv4scans(vis,solWVR=False):
  '''
  Calculate PWV(t) from the raw WVR data using dWdLs2()/dWdLs3() in four
  consecutive scans around the solar scan - start with leading phase-scan and
  finish with the trailing phase-cal scan.

  For comparison, calculate the sciSPW-based PWV(t) proxy by dWdBdG() for the
```

```
solar on-source subscans, too.

WVRs in the solar on-source subscan are disabled by default ( *solWVR* = F),
because, in general, they are saturated. May be set to T for the specific
MS: Band 5 data of 2022.1.00023.CSV (WVR are ~10 arcmin off-limb)

Save WVR + sciSPW data to './pwv4scans.npy' for 12m antennas and sciSPW data
extra for all antennas to './PWV_TB_GP_DATA-antEff.npy'. Both files are open
in 'append mode, the recent-most results are the last (the older may be used
for comparison of different approaches, e.g.dWdLs2() vs. dWdLs3(), solar
contamination in off-the-Sun scans allowed or not, etc.)
'''

#-- I/O

# Output-plots directory.
pngDir='./4scanPWVs/'
if (not os.path.exists(pngDir)): os.makedirs(pngDir)


#-- Global tools & data

myMS=mstool()
myMS.open(vis)

myMsmd=msmdtool()
myMsmd.open(vis)

acdLoads=aU.getLoadTemperatures(vis,doplot = False,
                                warnIfNoLoadTemperatures = True,
                                mymsmd=myMsmd)


#-- Prepare ATM models

#- ATMs for WVR

# Leading P-cal scan (WVR).
pCal1ATM=attool()
setAtmTool(pCal1ATM,vis,4,7,myMsmd=myMsmd)

# Pre-solar ATM scan (WVR).
sysCalATM=attool()
setAtmTool(sysCalATM,vis,4,8,myMsmd=myMsmd)

# Solar scan
solATM=attool()
setAtmTool(solATM,vis,4,9,myMsmd=myMsmd)

# Trailing P-cal scan (WVR).
pCal2ATM=attool()
setAtmTool(pCal2ATM,vis,4,10,myMsmd=myMsmd)


#- ATMs for SCI SPWs.

# Sci SPW for leading-phase scan (sci).
p1SciATM=attool()
```

```
setAtmTool4SPWs(p1SciATM,vis,7,myMsmd=myMsmd)

# Sci SPW for solar scan (sci).
solSciATM=attool()
setAtmTool4SPWs(solSciATM,vis,9,myMsmd=myMsmd)


# Sci SPW for leading-phase scan (sci).
p2SciATM=attool()
setAtmTool4SPWs(p2SciATM,vis,10,myMsmd=myMsmd)


#-- Get antenna list.
antIDs=myMsmd.antennaids()
antNames=myMsmd.antennanames()

#-- Redirect output to a text file.

sys.stdout=open("./STDOUT.txt", "w")

#-- Loop over all antennas.
for myAnt in antIDs:

  # Ant ID as int.
  antID=int(myAnt)


  # Report progress.

  print('\n============================================================')
  print('Calculating antenna ',antID,'<-->',antNames[myAnt])
  print('============================================================\n')

  #-- Run sci-spw PWV extraction via dWdBdG2() for all antennas.

  sciPWV_S=dWdBdG2(vis,8,9,solSciATM,antID,
                   spwList=[5,7,9,11,6,8,10,12,0,1,2,3],
                   T_tgt=6905.0,flaggedData=[],myMS=myMS,myMsmd=myMsmd,
                   acdLoads=acdLoads)


  '''
 An attempt (failed) to extract sci-SPW PWV proxy from the phase-calibrator
 scans. For some reason - most likely that PC-scan is performed in the non-MD
 mode - the powers in the p-cal scan are much higher than expected and
 using the gains found by the ATM/ACD scan lead to brightness temperatures
 comparable to the Sun, i.e. around 4000K!

 sciPWV_P1=dWdBdG2(vis,8,7,p1SciATM,antID,
                   spwList=[5,7,9,11,6,8,10,12,0,1,2,3],
                T_tgt=0.0,flaggedData=[],myMS=myMS,myMsmd=myMsmd,
                   acdLoads=acdLoads)

 sciPWV_P2=dWdBdG2(vis,8,10,p2SciATM,antID,
                   spwList=[5,7,9,11,6,8,10,12,0,1,2,3],
                T_tgt=0.0,flaggedData=[],myMS=myMS,myMsmd=myMsmd,
                   acdLoads=acdLoads)
```

```
    '''
    #-- Run the dWdLs4() calculations for each scan. Only for 12m antennas.
    if (antNames[myAnt][:2]!='CM'): # Skip CM antennas, no WVR.

      pCal1PWV=dWdLs4(vis,pCal1ATM,antID,7,
                      flaggedWVRchannels=[0],ssType='',solIllumination=False,
                      myMsmd=myMsmd,acdLoads=acdLoads)

      sysCalPWV=dWdLs4(vis,sysCalATM,antID,8,
                       flaggedWVRchannels=[0],ssType='',solIllumination=False,
                       myMsmd=myMsmd,acdLoads=acdLoads)

      solOffPWV=dWdLs4(vis,sysCalATM,antID,9,
                       flaggedWVRchannels=[0],ssType='off',
                       solIllumination=False,
                       myMsmd=myMsmd,acdLoads=acdLoads)


      if (solWVR):
        # WVR in solar on-source subscan.
        solOnPWV=dWdLs4(vis,solATM,antID,9,
                        flaggedWVRchannels=[0],ssType='on',solIllumination=True,
                        myMsmd=myMsmd,acdLoads=acdLoads)



      pCal2PWV=dWdLs4(vis,pCal2ATM,antID,10,
                      flaggedWVRchannels=[0],ssType='',solIllumination=False,
                      myMsmd=myMsmd,acdLoads=acdLoads)


      #- Comparison plots only for 12m antennas.

      # Init plot for antenna.
      plt.rcParams["figure.figsize"] = (20,15)
      plt.suptitle(antNames[myAnt])

      # Plot the WVR-based PWV(t).
      t1=pCal1PWV['time']
      p1=pCal1PWV['pwv']
      plt.plot(t1,p1,'r.')

      t2=sysCalPWV['time']
      p2=sysCalPWV['pwv']
      plt.plot(t2,p2,'g.')


      t3=solOffPWV['time']
      p3=solOffPWV['pwv']
      plt.plot(t3,p3,'b.')

      if (solWVR):
        # WVR in solar on-source subscan.
        t4=solOnPWV['time']
        p4=solOnPWV['pwv']
        plt.plot(t4,p4,'k.')
```

```
    t5=pCal2PWV['time']
    p5=pCal2PWV['pwv']
    plt.plot(t5,p5,'m.')


    # Add the sciSPW-based PWV(t).
    t6=sciPWV_S['time']
    p6=sciPWV_S['pwv']
    plt.plot(t6,p6,'c.')

    '''
  t7=sciPWV_P1['time']
  p7=sciPWV_P1['pwv']
   plt.plot(t7,p7,'c.')

  t8=sciPWV_P2['time']
  p8=sciPWV_P2['pwv']
   plt.plot(t8,p7,'c.')
    '''

    # Add the SYSCAL-produced PWVs. Select just last three values
    # which are at times around our 4 scans.
    sTimes,sPWVs=getPWVforAnt(vis,9,antID,myMsmd=myMsmd)['allPWVs']
    tLast=sTimes[-3:]
    pLast=1000.0*sPWVs[-3:] # PWV in m -> mm

    plt.plot(tLast,pLast,'ro')


    # Save to a file and close.
    antIndex='_{:02d}'.format(antID)


    plt.savefig(pngDir+'ant'+antIndex+'.png')
    plt.show()
    plt.close()


    #- Save WVR/PWV data to binary NPY file.

    # Convert the antenna ID to a single-element array.
    antIDa=np.array([antID])

    # Open our NPY-binary file in the append mode and save the arrays for
    # the current antenna.
    with open('./pwv4scansW.npy', 'ab') as f:
      np.save(f,antIDa)


      # WVR-based PWVs
      np.save(f,t1)
      np.save(f,p1)
      np.save(f,t2)
      np.save(f,p2)
      np.save(f,t3)
      np.save(f,p3)

      if (solWVR):
```

```
          # WVR in solar on-source subscan.
          np.save(f,t4)
          np.save(f,p4)

        np.save(f,t5)
        np.save(f,p5)

        # Sci SPW-based PWVs
        np.save(f,t6)
        np.save(f,p6)

        '''
       np.save(f,t7)
       np.save(f,p7)
       np.save(f,t8)
       np.save(f,p8)
        '''
      # --> end(save block)

    #--> endif(not-a--CM-antenna)

    #-- Save the *full* dWdBdG() output for *all* antennas.

    antIDa=np.array([antID])

    # Extract the arrays from the map.
    tt=sciPWV_S['time']
    dW=sciPWV_S['pwv']
    dB=sciPWV_S['T_B']
    AA=sciPWV_S['J']
    CC=sciPWV_S['C']


    # Open our NPY-binary file in the append mode and save the arrays for
    # the current antenna. Save the matrix AA and the RHS vector CC, too, so
    # that the results can be extracted by our restoreSol_dWdB() method.
    with open('./PWV.npy', 'ab') as f:
      np.save(f,antIDa)
      np.save(f,tt)
      np.save(f,dW)
      np.save(f,dB)
      np.save(f,AA)
      np.save(f,CC)
    # --> end(save block)

    # Flush the redirected stdout stream.
    sys.stdout.flush()
# --> endfor(all antennas)

#--- Clean-up

# Restore stdout back to console.
sys.stdout = sys.__stdout__


solSciATM.done()
pCal2ATM.done()
solATM.done()
```

```
  sysCalATM.done()
  pCal1ATM.done()

  myMsmd.done()
  myMS.done()

  return


#--------------------------------------------------------------------------------


def pwv4scansRestore(vis,hasSolWVR=True,runID=100000):
  """
  Extract data saved by pwv4scans(). The parameter *runID* controls, which run
  of pwv4scans() should be extracted (pwv4scans() appends the new results to
  the end of existing files). The default value of 100000 means to take the
  recent-most data. The *runID* starts counting from 1.

  Parameter *vis* is here for determining the antenna array.

  The flag *hasSolWVR* (default = T) indicates, whether the data contain also
  the on-source subscan WVR data.
  """

  #-- Static settings

  filePath='./pwv4scansW.npy'


  #-- List all antennas in the array.
  myMsmd=msmdtool()
  myMsmd.open(vis)
  antIDs=myMsmd.antennaids()
  antNames=myMsmd.antennanames()

  myMsmd.done()

  # So far empty map [antID -> pwv_rec].
  pwv_map={}


  # Init the record counter.
  rCount=0


  # Open the binary dWdB data.
  with open(filePath, mode='rb') as f:

    #-- Loop over the stack of records.
    while(rCount<runID):

      #-- Loop over all antennas.
      for myAnt in antIDs:

        # Skip the CM antennas, they have no WVR.
        if(antNames[myAnt][:2]=='CM'): continue
```

```
    # We may be aiming at the last record (the default) - check the EOF.
    # The np.load() method needs to be checked via exceptions.
    try:

      antIDa=np.load(f)

    except (EOFError,ValueError, OSError): # EOF: new/old NumPy style
      print("EOF reached beyond record no. ",rCount,".\n")
      # Prevent further reading attempts.
      rCount=runID
      # Break the antenna for-loop.
      break

    #--> end try/catch block.

    # Convert the antenna ID from (single-element) int-array to int.
    antID=antIDa[0]

    # Read the times and pwvs per scans/sub-scans
    t1=np.load(f)
    p1=np.load(f)
    t2=np.load(f)
    p2=np.load(f)
    t3=np.load(f)
    p3=np.load(f)


    if (hasSolWVR):
      # Read solar on-source subscan data only if they are present.
      t4=np.load(f)
      p4=np.load(f)


    t5=np.load(f)
    p5=np.load(f)
    t6=np.load(f)
    p6=np.load(f)


    #- Organise the data depending whether we have or not the on-source WVRs

    if (hasSolWVR):
      # All times as a list of arrays
      times=[t1,t2,t3,t4,t5,t6]
      # All corresponding PWSs as a list of arrays
      pwvs=[p1,p2,p3,p4,p5,p6]
    else:
      # Times except the on-source subscan as a list of arrays
      times=[t1,t2,t3,t5,t6]
      # Corresponding PWSs except the on-source subscanas a list of arrays
      pwvs=[p1,p2,p3,p5,p6]


    # One record per antenna
    pwv_rec={'times':times,'pwvs':pwvs}

    # Append the record to the collection. If multiple records/runs are
```

```
        # read, the map item is overwritten until the last one.
        pwv_map[antID]=pwv_rec


      #--> endfor(all antennas)

      # Increment the record counter.
      rCount+=1

    #-->endwhile(recordCountReached)
  #--> end with-block

  # Return the entire collection.
  return pwv_map


#-----------------------------------------------------------------------------

def dpwvXsolWVR_raw():
  """
  A simple 'by-eye' comparison our dPWV proxy for the *solar* scan with
  the *raw* WVR data in the solar scan.

  dPWV proxy is calculated from *science SPWs* using dWdB(). We utilize here the
  pre-calculated output saved in dWdB_data.npy

  WVR is calculated from raw WVR data (SPW 4) via dWdLs().

  We plot both time profiles as lef and right panel of a single figure to
  see the differences.

  *** Specific for 2022.1.00031CSV project! ***

  This method cannot be used outside of the project - it was just a good luck
  that the WVR were ponting out of the Sun because the Band 5 receiver was
  pointed close to the limb - see fixSolWVRs() for details.
  """

  #-- Static settings for this specific project/MS.

  # Input data selection
  vis='uid___A002_X110fd13_X1099e.ms'
  tgtScan=9
  atmScan=8

  # Output figures directory - separate for each SPW type.
  pngDir='./sol_WVR+dPWV/'
  if (not os.path.exists(pngDir)): os.makedirs(pngDir)

  # Set the ATM model for the WVR data (SPW 4) and the solar scan
  myAtm=attool()
  setAtmTool(myAtm,vis,4,tgtScan)

  #-- Extract the list of antennas.
  myMsmd=msmdtool()
  myMsmd.open(vis)

  antIDs=myMsmd.antennaids()
```

```
antNames=myMsmd.antennanames()

myMsmd.done()

#-- dPWVs: To be extracted from the pre-calculated file.

dwdbMap=restoreSol_dWdB()

#-- Loop over all antennas

for myAnt in antIDs:

  # Skip all CM antennas, these have no WVRs.
  if (antNames[myAnt][:2]=='CM'):
    print("Skipping antenna ",antNames[myAnt]," - no WVR data.")
    continue

  # Report progress.
  print('Calculating antenna ',int(myAnt),'<-->',antNames[myAnt])

  #- Science SPW: dPWV proxy

  # Time axis for science SPWs.
  timeSci=dwdbMap[int(myAnt)]['time']
  # dPWV proxy from science SPW
  dPWV=dwdbMap[int(myAnt)]['dW']

  #- WVR data from SPW 4

  timeWVR,dW,dL=dWdLs(int(myAnt),myAtm)

  #- Init plots

  # Set the sizes of the plot
  plt.rcParams["figure.figsize"] = (10,5)
  # Make XX and YY plots as subplots (horizontally)
  ff,(sp1,sp2)=plt.subplots(1,2)
  # Add the antenna name as a plot title.
  plt.title(antNames[myAnt])


  #-  XX: left sub-plot
  # Plot the dPWV results as a red line.
  sp1.plot(timeSci,dPWV[0],'-r')
  # Overplot the WVR variations as blue.
  sp1.plot(timeWVR,dW,'-b')

  #-  YY: right sub-plot
  # Plot the dPWV results as a red line.
  sp2.plot(timeSci,dPWV[1],'-r')
  # Overplot the WVR variations as blue.
  sp2.plot(timeWVR,dW,'-b')

  #- Save to a file and close.

  plt.savefig(pngDir+'wvrXdpwv_'+antNames[myAnt]+'.png')
  plt.show()
  plt.close()
```

```
  #--> endfor(all antennas)

  # Close the ATM tool.
  myAtm.done()

  return


#--------------------------------------------------------------------------------

def pwv4ScansCompare(vis):
  """
  More sofisticated comarison between PWV as determined from sci SPWs
  via dWdBdG() method with the value calculated from the raw WVR temperatures
  via dWdLs3().

  We compare for all 12m antennas (the 9m CM antennas do not have WVRs) and
  for sake of efficiency we use pre-calculated PWVs stored in './pwv4scans.npy'
  via pwv4scans() and re-read by pwv4scansRestore().
  """

  # Where to store the correlation plots.
  pngDir='./pwvCorrs/'

  # Load the PWV time profiles mapped by antenna ID.
  pwvMap=pwv4scansRestore(vis)

  for antID,pwvProfile in pwvMap.items():

    # Extract times and pwvs for all scans/sub-scans.
    times=pwvProfile['times']
    pwvs=pwvProfile['pwvs']

    # The solar on-source subscan WVR-base PWV is listed at index 3.
    tW=times[3]
    pW=pwvs[3]

    # Its proxy counterpart calculated from sci SPW is at index 5.
    tS=times[5]
    pS=pwvs[5]

    # We need to have collocated data in order to calculate linear regression.
    # Interpolate to times of sci SPW proxy data as they have higher cadence.
    pWi=np.interp(tS,tW,pW)

    # Calculate coefficients of linear correlation between WVR- and sciSPW-based
    # PWV values interpolated to the sci-SPW times.
    k,q=np.polyfit(pWi,pS,1)

    #-- Print & plot the correlation results.
    print(antID,'\t',k,'\t',q)

    # Antenna index with two digits (e.g., '07') as the fig title.
    antIndex='_{:02d}'.format(antID)
    plt.suptitle(antIndex)

    # Scatter plot
```

```
    plt.plot(pWi,pS,'b.')

    # Find the range of WVR-based PWVs
    pwmin=np.min(pW)
    pwmax=np.max(pW)

    # Overplot the scatter-plot with best linfit/
    x=np.array([pwmin,pwmax])
    y=k*x+q
    plt.plot(x,y,'r-')

    # Add the identity line for comparison.
    plt.plot(x,x,'g-')

    # Save the fig to a file and close.
    plt.savefig(pngDir+'pwv_'+antIndex+'.png')
    plt.show()
    plt.close()
  #-->endfor(all recorded antennas)

  return
```

```
"""
This code reads the ephemeris of the solar-disc center from the file SC.txt and
for each its line it ads a predefined offset (default is +2deg) in *Azimuth* !
Shifted positions (transformed back to RA/Dec) are then written to the modified
ephemeris file off_pos.txt. The original solar-center ephemeris is obtained
by Solar Ephemeris Generator.

How does it work? For each line of the original ephemeris we
1. Convert RA/Dec to Az/El.
2. Add the desired offset (+2degs is the default) to Az (scaled by 1/cos(EL))
3. Convert back to Ra/Dec.

Rationale: For the sake of test of compatibility between our PWV proxy and WVR
data, we observe the Sun with two sub-arrays, Array 1 (~37 antennas) observes
defined solar target, Array 2 (2 -3 antennas) measures co-temporally WVRs in
the postion off-the-Sun. We, however, need that the offset from the Sun is
constant in Azimuth in order to preserve the same airmass for both arrays.

Two alternative approaches have been tried for the AZ-shift calculations:
(i) edit_coordinates(), which uses the aU utils for transforming to AzEl frame
and back, and (ii), edit_coordinates2(), which uses the *measure* CASA tool for
the same purpose. The results are similar (differences within 30 arcsecs), we
use the latter one as it is more accurate.
"""

import math

#-- Static parameters

#- Define static column positions in an ephemeris table.
TIME_START, TIME_END = 1, 18 # "2026-Jun-24 07:00"
RA_START, RA_END = 23, 34  # "06 10 46.41"
DEC_START, DEC_END = 35, 46  # "+23 24 46.6"

#- Conversion constants.
DEG_TO_RAD = math.pi/180.
RAD_TO_DEG = 180/math.pi
HRS_TO_RAD = math.pi/12.
RAD_TO_HRS = 1.0/HRS_TO_RAD

#- ALMA location.
[alma_lat,alma_long,obs] = aU.getObservatoryLatLong(observatory='ALMA')
# Latitude in radians.
phi = alma_lat*DEG_TO_RAD

#- I/O
input_file = "./SC.txt"       # Solar disc center.
output_file = "./off_pos.txt" # Off-limb position for WVR, 2degs W in Azimuth.

#-------------------------------------------------------------------------------


def edit_coordinates(time_str,ra_str,dec_str,azShift=2.0):
    """
    Convert to AzEl and shift by *azShift* degs (positive means westward at
    north hemisphere). Then convert back to RaDec.
    We use methods from Analysis Utils for doing so.
```

```
The shift means the actual angular separation from the solar center in the
direction of AZ, so it needs to be scale by 1/cos(EL)!
"""

#-- Convert strings to values using CASA quanta.

# Time works in its 'native' format.
time_q=qa.quantity(time_str)
# For RA/Dec we need to replace blanks with adequate separators.
ra_q=qa.quantity(ra_str.replace(" ", ":"))
dec_q=qa.quantity(dec_str.replace(" ", "."))

#-- Convert the quantities to expected units and extract the values.

# Time as MJDSecs.
time=qa.convert(time_q,'s')['value']
# RA as hours.
RA=qa.convert(ra_q,'h')['value']
# Dec as degs (already in original qa, just extract the value).
Dec=dec_q['value']

#-- Convert to AzEl frame using aU methods.

# Local Sidereal Time at ALMA
LST = aU.ComputeLST(time,alma_long)
# Hour Angle in radians.
HA = HRS_TO_RAD* (LST- RA)

# Azimuth.
AZ = math.atan2(math.sin(HA),
                (math.cos(HA)*math.sin(phi)
                 - math.tan(Dec*DEG_TO_RAD)*math.cos(phi)))+math.pi

# Elevation.
arg=(math.sin(phi)*math.sin(Dec*DEG_TO_RAD)
     +math.cos(phi)*math.cos(Dec*DEG_TO_RAD)*math.cos(HA))
EL = math.asin(arg);

#-- Add the AZ offset (default is +2 degs) expressed in radians.
#   We need to scale it by 1/cos(EL)! Otherwise, the actual separation
#   from the Sun's center is not 2degs.

AZ+=azShift*DEG_TO_RAD/ math.cos(EL)


#-- Convert back to Ra/Dec.

sin_dec = (math.sin(EL)*math.sin(phi) +
           math.cos(EL)*math.cos(phi)*math.cos(AZ))

# New Dec in degs.
nDec=RAD_TO_DEG*math.asin(sin_dec)


y = -math.sin(AZ)*math.cos(EL)*math.cos(phi)
x = math.sin(EL)- sin_dec*math.sin(phi)
```

```
    # New hour angle converted to hours.
    nHA = RAD_TO_HRS*math.atan2(y,x)

    # Wrap it to 24h.
    nHA=nHA % 24.0

    # New RA in hours.
    nRA = LST- nHA

    # Wrap it to 24h.
    nRA=nRA % 24.0


    #-- Check the difference in degs.

    distSC=math.sqrt(((nRA- RA)*15.0*math.cos(DEG_TO_RAD*Dec))**2 +(nDec- Dec)**2)

    print("Off-pos to SC: ",qa.time(qa.quantity(time,'s'),prec=8,form="ymd"),
          (nRA- RA)*15.0,nDec- Dec,distSC)

    #-- Use quanta (+ a little adjustment) for formating the new RA/Dec strings.

    #- RA

    nRA_q=qa.quantity(nRA,'h')
    nRA_str=qa.angle(nRA_q,prec=8,form='time')[0]
    # Make the string JPL Horizon ephemeris conformant.
    nRA_str=nRA_str.replace(":"," ")

    #- Dec

    nDec_q=qa.quantity(nDec,'deg')
    nDec_str=qa.angle(nDec_q,prec=7,form='dig2')[0]
    # Make the string JPL Horizon ephemeris conformant.
    nDec_str=nDec_str.replace("."," ",2)


    #-- Ensure the new strings maintain the exact column width
    new_ra = nRA_str.ljust(RA_END - RA_START)
    new_dec = nDec_str.ljust(DEC_END - DEC_START)

    return new_ra, new_dec


#------------------------------------------------------------------------------


def edit_coordinates2(time_str,ra_str,dec_str,azShift=2.0):
    """
   A more accurate alternative to edit_coordinates(). Now we use
   methods of *measure* tool from the CASA toolkit.

   Convert to AzEl and shift by *azShift* degs (positive means westward at
   north hemisphere). Then convert back to RaDec.

   The shift means the actual angular separation from the solar center in the
   direction of AZ! The measure.shift() method takes care about it.
    """
```

```
    #-- Convert strings to values using CASA quanta.

    # Time works in its 'native' format.
    time=qa.quantity(time_str)
    # For RA/Dec we need to replace blanks with adequate separators.
    ra=qa.quantity(ra_str.replace(" ", ":"))
    dec=qa.quantity(dec_str.replace(" ", "."))

    azOffset=qa.quantity(azShift,'deg')

    #-- Reform qunatities to measures.

    myMe=metool()

    # Sun center direction.
    SunDir=myMe.direction('J2000',ra,dec)

    # Reference frame
    tEpoch=myMe.epoch('utc',time)
    obsPoint=myMe.observatory('ALMA')

    myMe.doframe(tEpoch)
    myMe.doframe(obsPoint)

    #-- Convert to AZEL and apply the offset.

    SunAZEL=myMe.measure(SunDir,'azel')

    # The me.shift() performs actual displacement, taking into account the
    # spherical scaling by 1/cos(EL). The 3rd argument, the position angle (PA),
    # is measured from the positive y-axis (!) (i.e. elevation) to the positive
    # x-axis (i.e. azimuth). That is why it is set to 90 deg - see CASA docs
    # for measures toolkit.
    SunOffset=myMe.shift(SunAZEL,azOffset,'90.0deg')


    """
    #-- An alternative implementation to the AZ-shift. Results are similar.

    # Current elevation.
    el=SunAZEL['m1']['value']

    # Make a copy and manipulate the Azimuth.
    SunOffset=SunAZEL
    # We need to scale by 1/cos(el) !
    SunOffSet['m0']['value']+=(azShift*DEG_TO_RAD/math.cos(el))
    """


    #-- Convert back and extract the new RA/Dec.

    SunDirOff=myMe.measure(SunOffset,'J2000')


    # New RA/Dec as quantities.
    nRA=SunDirOff['m0']
    nDec=SunDirOff['m1']
```

```
    # Just a check.
    SunDir2=myMe.direction('J2000',nRA,nDec)

    print(time_str,myMe.separation(SunDir,SunDir2)['value'])


    #-- Get new RA/Dec as strings.

    #- RA

    nRA_str=qa.angle(nRA,prec=8,form='time')[0]
    # Make the string JPL Horizon ephemeris conformant.
    nRA_str=nRA_str.replace(":"," ")

    #- Dec

    nDec_str=qa.angle(nDec,prec=7,form='dig2')[0]
    # Make the string JPL Horizon ephemeris conformant.
    nDec_str=nDec_str.replace("."," ",2)


    #-- Ensure the new strings maintain the exact column width
    new_ra = nRA_str.ljust(RA_END - RA_START)
    new_dec = nDec_str.ljust(DEC_END - DEC_START)

    myMe.done()

    return new_ra, new_dec



#-----------------------------------------------------------------------------

# This flag tracks whether we are currently inside the data block
inside_ephemeris = False


"""
The main() code for the routine. It reads the input ephemeris file line by line
and calls edit_coordinates2() intrenally. The shifted RA/Decs are then written
into the output file.
"""

with open(input_file, "r") as infile, open(output_file, "w") as outfile:
    for line in infile:
        stripped_line = line.rstrip("\n")

        # Check for the start marker
        if "$$SOE" in stripped_line:
            inside_ephemeris = True
            outfile.write(line)  # Write the marker line as-is
            continue

        # Check for the end marker (optional, but good practice)
        if "$$EOE" in stripped_line:
            inside_ephemeris = False
            outfile.write(line)  # Write the marker line as-is
```

```
            continue

        # If we are not inside the SOE/EOE block, copy the header/footer
        # untouched
        if not inside_ephemeris:
            outfile.write(line)
            continue

        # --- Process Data Lines ---
        if not stripped_line:
            outfile.write(line)
            continue

        # Extract columns using absolute positions
        time_val=stripped_line[TIME_START:TIME_END]
        ra_val = stripped_line[RA_START:RA_END]
        dec_val = stripped_line[DEC_START:DEC_END]

        # Modify the values
        new_ra_val, new_dec_val = edit_coordinates2(time_val,ra_val, dec_val)

        # Reconstruct the line
        modified_line = (stripped_line[:RA_START]
                         + new_ra_val
                         + stripped_line[RA_END:DEC_START]
                         + new_dec_val
                         + stripped_line[DEC_END:])

        # Write out with original trailing newline character
        outfile.write(modified_line + "\n")

print("File processing complete! Headers preserved.")
```

Annex 5

Online Project Repository
http://wave.asu.cas.cz/shared/ESO/final_report

Annex 6

Phasor analysis for quality assessment of self-calibration solution

Study of the Difference Between Accumulated Phase Corrections from Self-Cal Tables and Measurement Set Phase Differentials Using Directional Phasor Analysis

Abstract

Determining the exact total phase corrections applied during solar interferometric observations is vital for validating self-calibration convergence and imaging dynamic structures. In this study, we analyze the correspondence between two complementary diagnostic routes: (1) compounding phase corrections across successive self-calibration gain tables, and (2) computing visibility phase differentials between the pre-self-calibration data and the final self-calibrated Measurement Set. We detail the physical origins of the discrepancies that emerge when using simple scalar averaging over channel spectra, examine the empirical heuristic routine developed to correct branch-cut phase jumps across spectral windows, and present a rigorous, parameter-free directional phasor methodology based on circular statistics on the complex unit circle S^1. Verification across antenna baselines confirms complete phase closure down to numerical limits.

1. Introduction and Problem Context

Validating self-calibration performance requires confirming that the phase corrections applied to the visibilities match the underlying gains derived during each calibration cycle. This can be evaluated via two independent diagnostic approaches:

1. **Accumulated Gain Table Solutions:** Compiling antenna phase corrections across consecutive gaincal() execution cycles to determine the net cumulative phase rotation applied to each antenna.
2. **Measurement Set Visibility Differentials:** Calculating the channel-averaged phase difference between the original pre-selfcal data and the final post-selfcal split Measurement Set:

$$\Delta\phi_MS(t) = \phi_orig(t) - \phi_selfcal(t)$$

In an ideal system, the phase difference $\Delta\phi_MS(t)$ on any baseline to the reference antenna must strictly track the cumulative phase rotation $\phi_accum(t)$. However, direct numerical comparisons often reveal large, unphysical discrepancies across extended time ranges. Detailed investigation of the channel-by-channel phase spectra demonstrates that these errors arise from systematic calculation artifacts when simply calculating linear arithmetic averages of phase angles across frequency channels and spectral windows (SPWs).

2. Failure Modes of Linear Phase Averaging

Screening visibility phase data across multiple SPWs reveals four operational failure regimes that degrade linear phase averaging:

- **Case 1 (Branch Cut Bimodal Splitting):** The majority of data points are clustered in a narrow phase range, but channel noise pushes points across the $\pm\pi$ ($\pm 180°$) branch cut. An arithmetic mean of $+179°$ and $-179°$ evaluates to $0°$ (a $180°$ phase reversal) and inflates the standard deviation to ~$179°$.

- **Case 2 (SPW-Wide 2π Phase Shifts):** Similar to Case 1, but most points in one entire SPW are shifted by 2π relative to adjacent SPWs, resulting in an artificially high standard deviation for that SPW and corrupting the multi-SPW combined mean.
- **Case 3 (Mixed SPW Coherence):** Only one or two SPWs exhibit good channel alignment (low standard deviation), while the remaining SPWs are scattered across the whole phase range.
- **Case 4 (Full Spectral Decoherence):** All SPW data are degraded simultaneously, exhibiting wide dispersion across the whole channels.

3. Heuristic Correction: Dynamic Phase Standard and Variance Rollback

To correct the phase data under the physical constraint that the phase jump between adjacent timestamps should not be large, an empirical correction workflow was established:

3.1 Phase Standard Establishment Algorithm

For each dataset and at every timestamp, edge channels are discarded (trimming the first and last 10 channels of each SPW to avoid bandpass roll-off). A dynamic reference value—the Phase Standard (ϕ_std)—is determined through three decision rules:

- **Rule A (Dominant Coherent SPW):** Calculate the standard deviation from the trimmed phase data of all SPWs. For the lowest standard deviation, if it is below a preset threshold ($\leq \pi/3 \approx 1.047$ rad), the corresponding SPW is considered relatively good, and the mean value of its trimmed phase data is set as the phase standard.
- **Rule B (Coherence Failure & Temporal Continuity):** If none of the SPWs are good (all $\sigma > \pi/3$), calculate the standard deviation from the phase data within the middle 50 percentile (25th–75th) of all trimmed channels across all SPWs. If this standard deviation $\geq \pi/2 \approx 1.571$ rad, the entire dataset at this timestamp is deemed unsuited for calculating a phase average. Assuming the phase jump between adjacent timestamps is small, the phase standard from the previous timestamp is retained: $\phi_std(t_i) = \phi_std(t_\{i-1\})$. (For $t = 0$, Rule A is applied ignoring the phase threshold).
- **Rule C (Moderate Multi-SPW Coherence):** For the remaining cases where no single SPW meets Rule A but the standard deviation of the 50 percentile group remains below $\pi/2$, the mean value of this 50-percentile group is set as the phase standard.

3.2 Point-by-Point 2π Modification and Rollback Verification

Once ϕ_std is established, channel corrections are executed:

3. **Phase Modification:** Apply phase correction for each data point according to the phase standard by checking the difference $|\theta_k - \phi_std|$. If the absolute value is $\leq \pi$, no change is applied. If the absolute difference is $> \pi$, apply a $\pm 2\pi$ modification accordingly.
4. **Variance Rollback Check:** Recalculate the mean value and standard deviation for each SPW. If the newly modified data has lower or equal standard deviation, the phase change is kept. Otherwise, roll back to the unchanged phase data to avoid over-correction.

4. The Directional Phasor Formulation (Circular Statistics)

While the heuristic correction routine resolves major branch jumps, its dependence on fixed empirical thresholds ($\pi/3$, $\pi/2$) and temporal state carryover prompted the implementation of a mathematically unified, parameter-free approach using directional statistics on the complex unit circle S^1.

Because phase-only self-calibration maintains $|g| \approx 1.0$, visibility phases θ_k are mapped directly to unit complex phasors:

$$z_k(t) = e^{i\,\theta_k(t)} = \cos(\theta_k(t)) + i \cdot \sin(\theta_k(t)) \quad (|z_k| = 1)$$

4.1 Resultant Vector and Directional Mean

Averaging over all trimmed channels across all SPWs yields the mean resultant vector:

$$\bar{R}(t) = (1 / N) \sum z_k(t) = \bar{C}(t) + i \cdot \bar{S}(t), \quad \bar{\theta}(t) = \operatorname{atan2}(\bar{S}(t), \bar{C}(t))$$

Because vector addition operates directly on Cartesian coordinates, phase wrapping across $\pm\pi$ is resolved naturally:

- **Cases 1 & 2 (Boundary Wrapping):** Phasors at +179° and −179° add constructively along the negative real axis ($\bar{C} \approx -0.9998$, $\bar{S} = 0$), immediately yielding $\bar{\theta} = 180°$ (π rad) without unwrapping thresholds.
- **Case 3 (Mixed SPW Coherence):** Incoherent SPWs scatter uniformly over S^1 and cancel vectorially ($\sum e^{i\theta} \approx 0$), allowing the coherent SPWs to dominate $\bar{R}(t)$ automatically.
- **Case 4 (Full Decoherence):** The resultant magnitude approaches zero ($|\bar{R}| \to 0$), correctly signaling total dispersion while introducing zero state contamination into subsequent integrations.

4.2 Circular Dispersion and Multi-Cadence Table Accumulation

The circular standard deviation is defined analytically from the coherence length $|\bar{R}| \in [0, 1]$:

$$\sigma_{circ}(t) = \sqrt{(-2 \ln |\bar{R}(t)|)} \quad \text{(rad)}$$

For self-calibration sequences with decreasing solution intervals (e.g., ['inf', '60s', 'int', ...]), tables are accumulated multiplicatively using nearest-centroid timestamp matching:

$$G_{accum}(t) = \prod_{m=0}^{M-1} e^{i\,\phi_m(t)} \implies \phi_{accum}(t) = \arg(G_{accum}(t)) \in [-\pi, \pi]$$

5. Results and Residual Phase Closure Verification

Evaluating the modified phase data across antenna baselines confirms that both approaches share an identical physical result: the total phase correction during solar observation. Fig 1 shows the

comparison using the ALMA test data taken on 2026/07/30. The residual phase closure metric: $\delta\phi(t) = \text{wrap}_\pi(\Delta\phi_{MS}(t) - \phi_{accum}(t))$ is almost zero which verifies that the accumulated gain tables accurately compile the exact cumulative phase rotation applied to the calibrated Measurement Set.

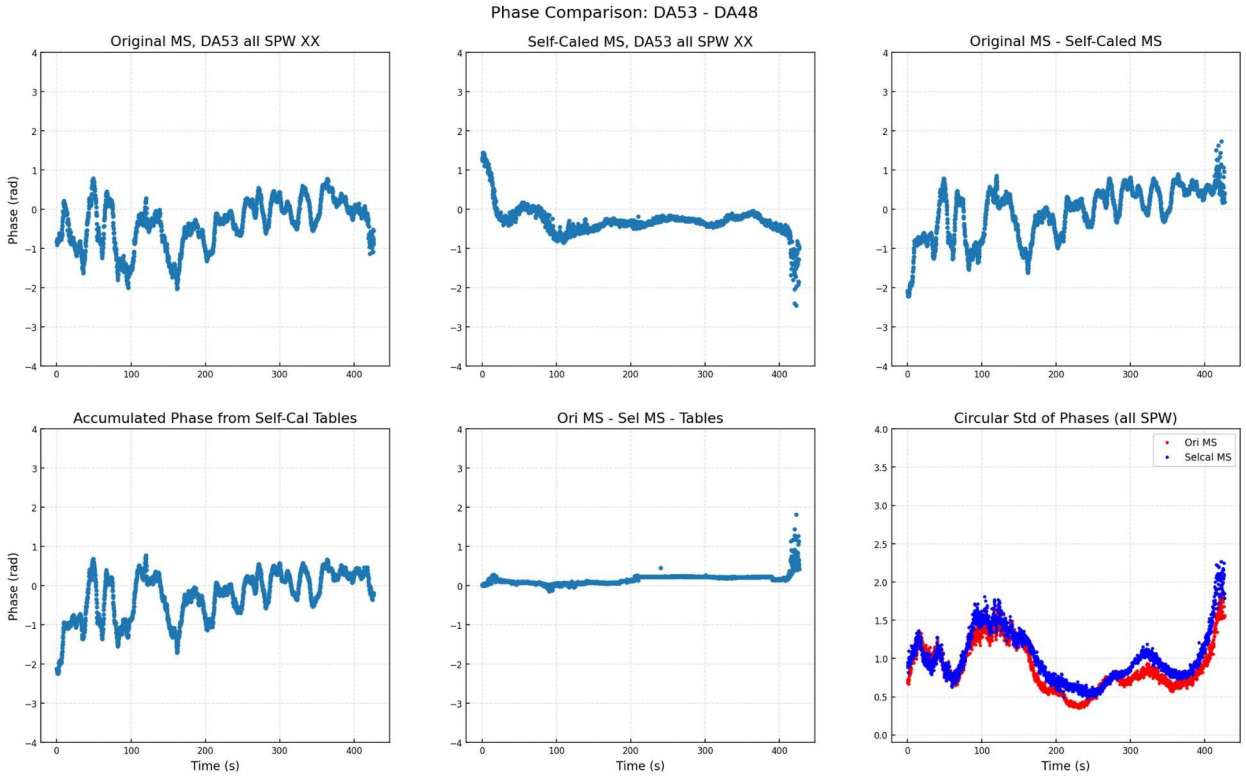


Fig 1. Phase comparison between accumulated phase gain table from self-calibration and phase differential between the unself-calibrated data and the final self-calibrated measurement set. The circular standard deviations of phases at all spectrum windows from both measurement sets are shown by the blue and red curve in the last panel.

6. Conclusion

The phase discrepancies observed between gain tables and visibility differentials were caused by linear phase averaging artifacts over branch cuts. The heuristic phase standard selection, 2π modification, and rollback verification confirmed that the visibility phase difference and the accumulated calibration solutions represent the exact same total phase correction. The directional phasor formulation provides a robust, parameter-free foundation that automates this analysis across all antennas and datasets without heuristic tuning.