

"""

*This code reads the ephemeris of the solar-disc center from the file SC.txt and for each its line it adds 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 (+2deg 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 position 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, 2deg 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 forming 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 quantities 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.")
```