Python script for Raster Scan Data
(Code provided courtesy of Tushar Agarwal)

List of tasks:

- Task #1: To read the data and plot the raster scans: Done
- Task #2: Grid scan data to 2-D map: Done
- Task #3: Grid scan data to 2-D map: Base line subtraction & Brightness temperature calibration (using quiet sun): Done
- Task #4: Subtract active regions, find quiet sun centroid, apply pointing correction
- Task #5: Iterate on brightness temperature calibration pointing corrected, active region subtracted quiet sun: TBD
- Task #6: Rotate map Sun's north pointing at top: TBD
- Task #7: Write fits maps: TBD

Data: *raster_21145.txt*
We use the raster table data for 2021 DOY 145 (May 25). In 2021 raster tables were written only for channel 6 which observed at 3.1 GHz. However, it contained several map ids and frequency bands observed in channel 6. To make it easier we have carved out only one map id for one frequency band: *raster_21145_932.txt*. Tushar used this file for his scripts.

The data header and first few rows:
raster_cfg_id rss_cfg_id year doy utc epoch source_id chan freq rate step raster_id raster_cfg_id year doy utc epoch xdecoff decoff ha dec tsrc
932 242786 2021 145 14:31:30 1621953090 66 6 3100.0 0.030 0.030 1012755 932 2021 145 14:31:34 1621953093.611000 0.5867 0.6055 282.7654 21.6725 0.000
932 242786 2021 145 14:31:30 1621953090 66 6 3100.0 0.030 0.030 1012756 932 2021 145 14:31:34 1621953094.111000 0.5554 0.6054 282.7654 21.6725 0.077

Necessary Install: python - https://www.python.org/
Required packages, install instructions can be found here:
https://packaging.python.org/en/latest/tutorials/installing-packages/

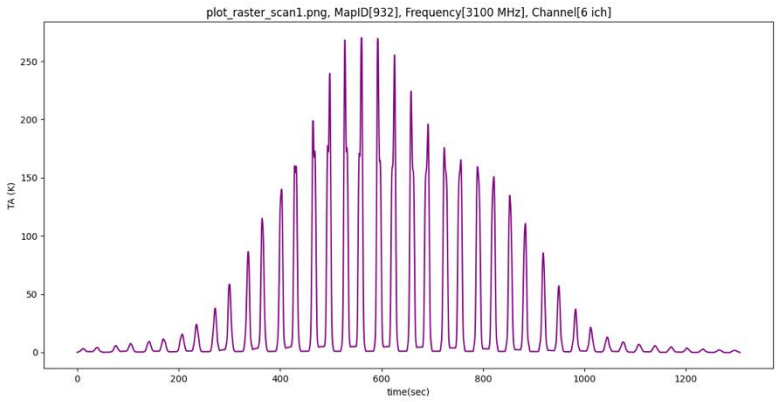
- pip
- matplotlib
- numpy
- pandas
- scipy

Packages can be installed using pip, e.g.: python -m pip install -U matplotlib

Task #1: To read the data and plot the raster scans

Python script:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# using pandas instead of numpy to read the data file
file = pd.read_csv("C:/python_solar/raster_21145_932.txt", delimiter = "  ", engine = “python”)
print(file)
epoch1 = np.array(file["epoch.1"], dtype = float)
epoch1 = epoch1-epoch1[0]
tsrc = np.array(file["tsrc"], dtype = float)

fig = plt.figure()
'''
# These commands to add the data of MapID, frequency, and channel within the plot
txt = fig.add_subplot()
txt.text(900,250,"MapID = 932")
txt.text(900,235,"Frequency(MHz) = 3100")
txt.text(900,220,"Channel(ich)= 6")
'''
plt.plot(epoch1, tsrc, color = "purple")
plt.xlabel("time(sec)")
plt.ylabel("TA (K)")
plt.title("plot_raster_scan1.png, MapID[932], Frequency[3100 MHz], Channel[6 ich]")  # here, to add those data in the title
fig.tight_layout()
plt.show()
fig.savefig("plot_raster_scan1.png", dpi = 200)
```

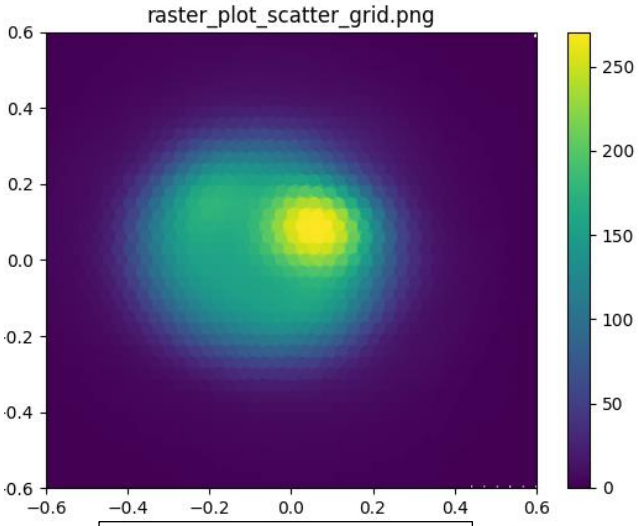


Task #2: Grid scan data to 2-D map

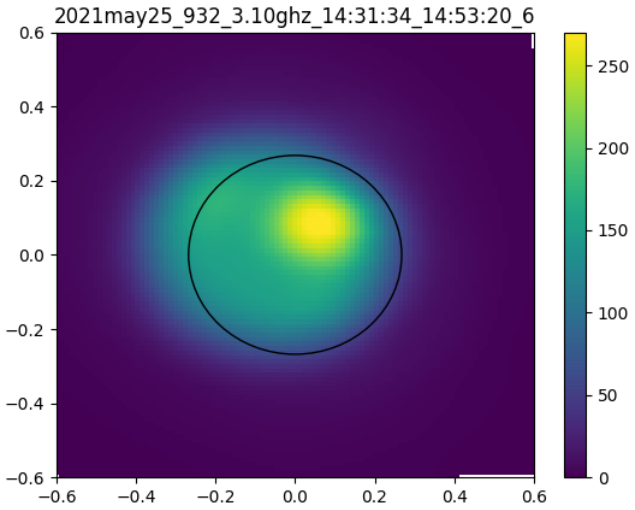
Python script:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.interpolate import griddata

# using pandas instead of numpy to read the data file
file = pd.read_csv("C:/python_solar/raster_21145_932.txt ", delimiter = "  ", engine = “python”) # extra whitespace given in between
#print(file)

gx = np.array(file["xdecoff"], dtype = float) # column 18 -- xdecoff
gy = np.array(file["decoff"], dtype = float) # column 19 -- decoff
gta = np.array(file["tsrc"], dtype = float) # column 22 -- tsrc (ta or tsys)

xmin = -0.6 # in degrees, sun disk = ±0.25 degrees
xmax = 0.6
ymin = -0.6
ymax = 0.6

nx = 101
ny = 101

xi = np.linspace(xmin, xmax, nx) # nx = 101 | xmin, xmax -- arguments of linspace
yi = np.linspace(ymin, ymax, ny) # ny = 101 | ymin, ymax -- arguments of linspace
xi, yi = np.meshgrid(xi,yi)

tamap = griddata((gx,gy), gta, (xi, yi))

fig = plt.figure()
plt.pcolormesh(xi,yi,tamap)
# plt.scatter(gx,gy, c=gta) # scattering the plot
plt.colorbar()
plt.axis([xmin, xmax, ymin, ymax])

# adding a circle
circle = plt.Circle((0,0), 0.268, color='k', fill=False)
ax = plt.gca() # gca - get current axis (uses the existing axis of the plot)
ax.add_patch(circle) # circle is a class of patch, so add_patch to call circle


# assigning variables for defining the label to be used as plot's title

mapdate = "2021may25" # date from filename
mapid = str(np.array(file["raster_cfg_id.1"][0])) # first id of raster_cfg_id.1 from the file
utstart = str(np.array(file["utc.1"][0])) # first (start) map time from the file
utend = str(np.array(file["utc.1"][1652])) # last (end) map time from the file
fmhz = str(np.array(file["freq"][0])) # first frequency (in mhz) from the file
band = "" # converting the frequency from mhz into ghz
if fmhz == "3100.0":
    band = "3.10ghz"
```

```
elif fmhz == "6000.0":
    band = "6.00ghz"
elif fmhz == "8450.0":
    band = "8.45ghz"
elif fmhz == "14000.0":
    band = "14.0ghz"
ch = str(np.array(file["chan"][0])) # first channel number from the file

# defining and assiging label
label = "_".join([mapdate, mapid, band, utstart, utend, ch])
plt.title(label)
plt.show()
```



Grid: No interpolation



Grid: With interpolation

Task #3: Grid scan data to 2-D map: Base line subtraction & Brightness temperature calibration (using quiet sun)

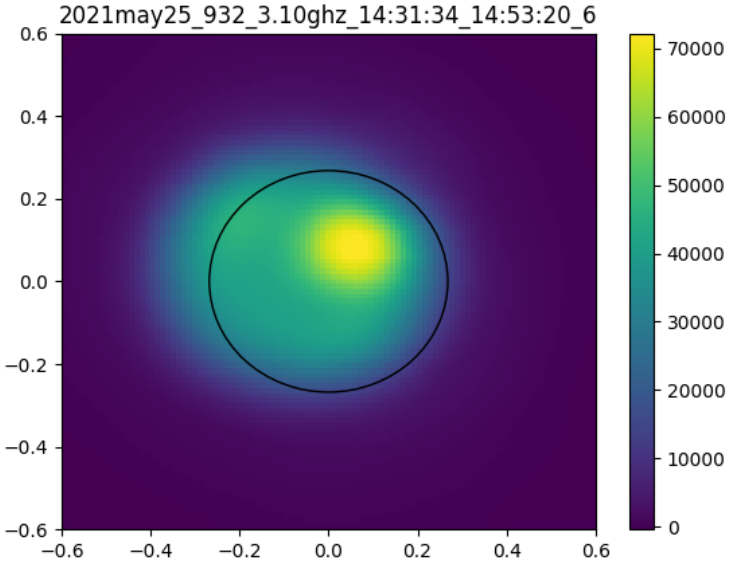
Python script:

```
from cmath import nan
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy
from scipy.interpolate import griddata

# using pandas instead of numpy to read the data file
file = pd.read_csv("C:/python_solar/raster_21145_932.txt", delimiter = "  ", engine = “python”) # extra whitespace given in between
#print(file)

gx = np.array(file["xdecoff"], dtype = float) # column 18 -- xdecoff
gy = np.array(file["decoff"], dtype = float) # column 19 -- decoff
gta = np.array(file["tsrc"], dtype = float) # column 22 -- tsrc (ta or tsys)

xmin = -0.6 # in degrees, sun disk = ±0.25 degrees
xmax = 0.6
ymin = -0.6
ymax = 0.6

nx = 101
ny = 101

xi = np.linspace(xmin, xmax, nx) # nx = 101 | xmin, xmax -- arguments of linspace
yi = np.linspace(ymin, ymax, ny) # ny = 101 | ymin, ymax -- arguments of linspace
xi, yi = np.meshgrid(xi,yi) # where xi, yi are x, y coordinate values (deg) for array pixel (element) (on the plot) (i,j)

tamap = griddata((gx,gy), gta, (xi, yi))
# print(max(tamap[np.where(tamap==tamap)])) # maximum value of tamap = +270

mapdate = "2021may25" # date from filename
mapid = str(np.array(file["raster_cfg_id.1"][0])) # first id of raster_cfg_id.1 from the file
utstart = str(np.array(file["utc.1"][0])) # first (start) map time from the file
utend = str(np.array(file["utc.1"][1652])) # last (end) map time from the file
fmhz = str(np.array(file["freq"][0])) # first frequency (in mhz) from the file
band = "" # converting the frequency from mhz into ghz
rdisk = "" # rdisk represents sun's map boundary

if fmhz == "3100.0":
    band = "3.10ghz"
    rdisk = 0.6
elif fmhz == "6000.0":
    band = "6.00ghz"
    rdisk = 0.5
elif fmhz == "8450.0":
    band = "8.45ghz"
```

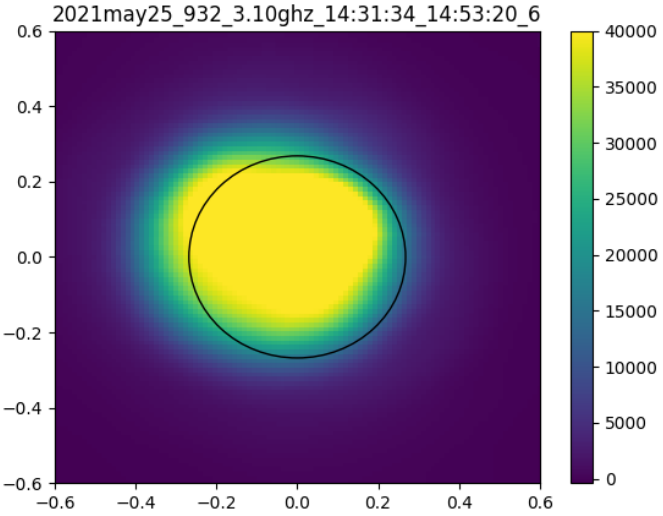
```
    rdisk = 0.45
elif fmhz == "14000.0":
    band = "14.0ghz"
    rdisk = 0.45
ch = str(np.array(file["chan"][0]))

rxy = np.sqrt(xi**2+yi**2) # rxy represnts the pixel radius -- i.e. the distance of each pixel in deg. from the map center

# for rxy>rdisk, map intensities are considered as background level "zero"
m,n = rxy.shape
i, j = np.where((rxy>rdisk) & (tamap == tamap)) # (tamap == tamap) takes on those value where the value is not nan, and it can be written as ~np.isnan(tamap) ~ converted isnan to not isnan -- to only consider not nan tamap values.

'''
# use this command to print full numpy arrray without truncation.
import sys
np.set_printoptions(threshold=sys.maxsize)
'''

arr = i*m+j
#print(arr)
narrs = (arr.size)
#print(narrs)

zero = 0
if narrs >= 1:
    zero = sum(tamap.flatten()[arr])/narrs # since tamap is 2d array and arr is 1d array, it's complicated to call tamap[arr] without first converting tamap into 1d array, so that's what I did here temporarily.
#print(zero)

# subtracting zero from map data
tamap = tamap-zero
#print(np.max(tamap))
#print(max(tamap[np.where(tamap==tamap)])) # maximum value of tamap = around +270


# converting nan data to zero in the baseline subtracted map

p,q = np.where(np.isnan(tamap))
tamap[p,q] = 0
#print(tamap)

i,j = np.where(tamap != tamap)
arr = i*m + j
#print(arr)
narr = arr.size
tamap[arr] = 0 #-- tamap2 was not defined, so I supposed it were tamap[arr]
#print("No. of nans in the map:", narr)
#print(max(tamap[np.where(tamap==tamap)])) # maximum value of tamap = +270
```

```
# matching sun disk to the map plot circle
# estimating the quiet sun level by finding the lowest value within rqsun -- quiet sun means the sun at the minimum of solar activity, occurring every 11 years
rqsun = 0.15 # inner radius of the quiet sun, in degrees to the center

i,j = np.where((rxy<rqsun) & (tamap == tamap))
arr = i*m+j
#print(arr)
narrs = arr.size
qscount = np.min(tamap.flatten()[arr]) # qscount -- quiet sun level in counts
# print(qscount)


# converting counts to brightness temperature (calibration)

qstb = ""  # quiet brightness temperature, in K (kelvin)

if fmhz == "3100.0":
    qstb = 40000
elif fmhz == "6000.0":
    qstb = 22000
elif fmhz == "8450.0":
    qstb = 18000
elif fmhz == "14000.0":
    qstb = 10000

#print(band, qstb)


count2tb = qstb/qscount # count2tb -- counts to brightness temperature
tbmap = tamap * count2tb # tbmap -- calibrated map in brightness temperature
#print(count2tb)
#print(tbmap)

# setting active regions brightness to quiet sun level
qsmap = tbmap # qsmap -- quiet sun map
i,j = np.where(tbmap>qstb)
arr = i*m+j
#print(arr)
narr = arr.size
#print(qstb)
# use this command to print full numpy arrray without truncation.

# ravel() allows you to see 2d array in 1d in same way as flatten does, but ravel affects the original array while flatten creates a copy of it, so here, ravel's appropriate
# but the problem is also that ravel affects tbmap too. So, qsmap still remails equal to tbmap. To solve that, I created a copy of qsmap, "qsmap1" and used ravel with it.
qsmap1 = qsmap.copy()
qsmap1.ravel()[arr] = qstb
```

```
#checking map intensities
#print(np.max(tbmap), np.max(qsmap1))

fig = plt.figure()
#plt.pcolormesh(xi, yi, tbmap)
plt.pcolormesh(xi, yi, qsmap1)


# plt.scatter(gx,gy, c=gta) # scattering the plot
plt.colorbar()
plt.axis([xmin, xmax, ymin, ymax])

# adding a circle
circle = plt.Circle((0,0), 0.268, color='k', fill=False)
ax = plt.gca() # gca - get current axis (uses the existing axis of the plot)
ax.add_patch(circle) # circle is a class of patch, so add_patch to call circle

# defining and assiging label
label = "_".join([mapdate, mapid, band, utstart, utend, ch])
plt.title(label)

plt.show()
```



Brightness temperature calibrated. Sun's disk overplotted. Note pointing offset



Attempt to construct quiet sun disk by setting active region brightness
(where Tb > Tb_qs Tb =Tb_qs)quiet sun value.)
➔ Need to subtract active region by fitting beam response (Task #4)

➔Need to apply pointing correction. When the active regions are subtracted we have quiet sun disk. Use the centroid of the quiet sun disk to fix the pointing offset.
Tushar is doing the script to (i) find the active region peaks; (ii) subtract Gaussin beam shapes (try subtracting successively up to three to five sources).

Task #4: Subtract active regions, find quiet sun centroid, apply pointing correction