HartRAO Absolute Flux Calibration

DARA Dift Scan For VIRGO A

Mini-Project

Technical Report For The Assignment On How To Do The Absolute Flux Calibration Using Data Of The 26m Antenna From HartRAO.

Student:

Dionísio Cândido Nhadelo

Tutors:

Job Vorster & James Chibueze

December 13, 2022

1 Introduction

In this mini-project we are going to perform flux density calibration for the HartRAO 26m antenna observations. We do flux density calibration to translate the gain scaling produced during the observation to absolute flux density scale, for that we must use a calibrator with a known flux density. The data that we are going to use are stored in FITS file, FITS stands for **Flexible Image Transport System**, they practically do whatever we want it to do. It is mostly but not only used for images, it can be used as a container for anything you want, like image data and binary tables.

There are some advantages of using FITS files because they come as raw and uncompressed, unlike .gif, .jpeg or .png files. The FITS files contains information about how and where the images were obtained, they are very informative by their headers. FITS files are divided in chunks, it's basic structure is composed by **Header Data Units** (HDU), so FITS files are composed by the primary HDUs and some extensions.

1.0.1 Aim

The main aim of this mini-project is to find the Point Source Sensitivity (PSS) using one of the three flux calibrators that are generally observed at HartRAO, they are namely Virgo A, Hydra A and 3C123. In order to compute the absolute flux calibration we are supposed to perform four main steps. Firstly we have to perform the noise diode calibration, where we calculate the counts per Kelvin conversion factor, what will allow us to get the antenna temperature (in K) which will be used to perform the baseline calibration.

Secondly we have to perform the drift scan fitting, where we fit the Gaussian to the centre part, which corresponds to the on data in order to get the peak. As third step we compute the pointing correction, where we fit a Gaussian function to the amplitudes obtained from the three scans to get the pointing correction factor, and finally we calculate the flux density by using the equation and parameters from the table 5 of the Ott et al., paper.

For a better understanding of the subject we are going to work on, let us consider some basic concepts of the radio astronomy observations with a single dish:

1.0.2 Antenna Beam Pattern

The antenna beam pattern is a measure of the sensitivity of the telescope to the incoming radio signals as a function of angle on the sky. The term antenna beam pattern derives from the idea of a beam of radio waves leaving a transmitting antenna, once the sensitivity pattern is the same, whether the antenna receives or transmits. In other words we can describe the antenna beam pattern as a graphical representation of elements of the radiation characteristics of an antenna.

1.0.3 Full Width at Half Maximum

The Full Width at Half Maximum (FWHM) is the angular resolution of a single dish telescope. This angle is defined by the width of central peak of the telescope sensitivity pattern, the angular width of this peak is measured between the two points where the received power falls to one-half of the on-axis value.

1.0.4 Sidelobes

In radio astronomy observations with a telescope antenna we can see the main beam (main lobe) and sidelobes. The main beam is the central peak of the sensitivity pattern, while sidelobes are the off-axis responses. Sidelobes are not desirable in radio astronomy observations because they can make us mistake a off-axis with a on-axis source. For example, we could detect as much amount of power from a bright source that is located in one of the sidelobes as we can detect from a faint source on-axis.

1.0.5 Antenna Temperature Of a Single Dish Telescope

In radio astronomy, the term antenna temperature does not mean the physical temperature of the antenna (the primary mirror). Antenna temperature indicates how much energy the telescope has measured. The formula of the antenna temperature is represented by the following equation:

$$T_A = \frac{1}{\Omega_A} \int_{4\pi} T_B(\theta,\phi) G_n(\theta,\phi) d\Omega K$$

Where:

T_A - is the antenna temperature, it is obtained by converting the raw counts to antenna temperature in Kelvin. This does not have anything with physical temperature.

Ω_A - is the beam solid angle. This angle is useful to estimate the antenna temperature that is produced by a compact source covering a solid angle Ω_s and a brightness temperature T_B.

$T_B(\theta,\phi)$ - is the brightness temperature, this is the temperature needed for a black body (thermal) radiator to produce the same specific intensity as the observed source.

$G_n(\theta,\phi)$ - is the generic power gain of the transmitting antenna, in other words is a generic beam pattern. It's unit is expressed in decibels (dB) as a log-scale.

1.0.6 Single Dish Drift Scan

In a simple way, a single dish drift scan can be described as a usage of a one telescope antenna to scan the sky, while that, the data are collected. To perform drift scan with a single dish, basically the telescope is parked at a given elevation and angle, letting the source pass through the antenna beam-width as the Earth rotates.

1.0.7 Absolute Flux Calibration

Absolute flux calibration the process of finding a relationship between instrumentall counts or antenna temperature and absolute flux density.

1.1 Accessing the data

It is a good practice in Data Science to inspect the data that we are going to work with, so that we get to know with what kind of data are we dealing with, that is not different in Astronomy, there is a lot of Data Science in astronomy, so, let us access the data and explore it. As first steps to explore the data, we are going to import some Python libraries that will allow us to open the files, to manipulate and visualize the data when needed. We do that with the following Python commands:

```python
"""
First of all we should import the needed libraries,for this project
we just need the following:
"""
import numpy as np # for mathematical calculations
import matplotlib.pyplot as plt # for data visualization
import astropy.io.fits as fits # for fits files opening
from scipy import stats # for error propagation
```

After importing all the basic libraries we are going to use in this coding project, we can now open the FITS file with the following Python code:

```python
"""
To make things less complicated, lets have the fit file at the same
folder as the Jupyter notebook
"""
file_name = "Report Data Sets/Data12.fits" # Reading the FITS file
hdu = fits.open(file_name) # Opens the FITS and assigns to the hdu.
hdu.info() # This will show details about the opened file
```

Now that we have opened the FITS file, we can play around with it. From hdu[0] to hdu[2] we can find basic information about the observation. Let us now look for the following: Date of observation, Telescope name, Source of interest, System temperature and the Observing frequency. In order to make it look good, lets summarize the outputs in a table. Our Python code for this is as it follow:

```python
print('Date of creation of the file was {}.'.format(hdu[0].header['DATE']))
print('The Telescope name  is {}.' .format(hdu[0].header["TELESCOP"]))
print('The source of interest was {}.'.format(hdu[0].header["OBJECT"]))
print("The system temperature is about {}K" .format(hdu[1].header["NOMTSYS"]))
print("The observing frequency is : {}MHz" .format(hdu[2].header["CENTFREQ"]))
```

Table 1: Observation Parameters.

Parameter	Value
Date of Observation	2018-02-24T23:10:09
Telescope Name	HartRAO 26m Antenna
Source of Interest	VIRGO A
System Temperature	132.0 [K]
Observing Frequency	12178.593 [MHz]

1.2 Noise diode calibration

Now that we have opened the file and play with it by seeing some basic information on it (Table 1), we can now do a visualization of the data by plotting the antenna counts as function of time with the following Python code:

```python
""" We are going to plot the noise diode against the MJD
"""

# Time of observation as Modified Julian Date
noise_MJD = hdu[2].data['MJD']

######################### CHANNEL-1 #######################
plt.figure(figsize=(16,5))
plt.subplot(121)
noise_diode_count1 = hdu[2].data['Count1']
plt.plot(noise_MJD,noise_diode_count1)
plt.title("Channel-1")
plt.xlabel('MJD',fontsize=14)
plt.ylabel('Antenna Counts1',fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

######################### CHANNEL-2 #######################
plt.subplot(122)
noise_diode_count2 = hdu[2].data['Count2']
plt.plot(noise_MJD,noise_diode_count2)
plt.title("Channel-2")
plt.xlabel('MJD',fontsize=14)
plt.ylabel('Antenna Counts2',fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
```

The above code produce the following output:

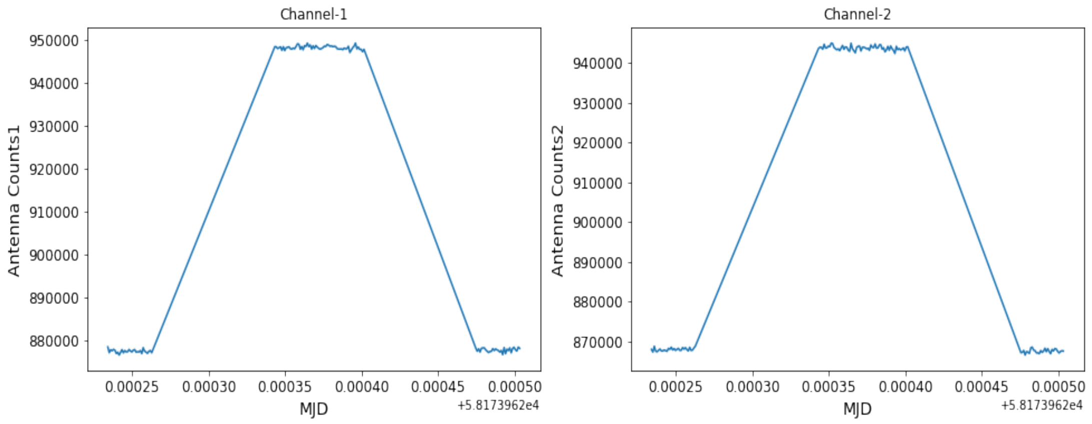

Figure 1: Antenna counts plotted against MJD.

From both plots in Figure 1, we can see that there is a part with a higher number of counts, it means the noise diode is on, the lowers is when the diode noise is turned off. We can now calculate the average on counts when the diode is ”off” and ”on”. With *numpy.where* and *numpy.mean* we are going to separate the counts in two parts, the counts that will be above the mean, we will consider as *on counts* and the counts that will be bellow the mean, we will consider as *off counts*. The results generated by the following python code can be seen in Table 2:

```
"""This part of the code will calculate the mean background for
'off' and 'on' counts for both polarizations"""

########################### CHANNEL-1 ###########################
on_1 = np.where(noise_diode_count1 > np.mean(noise_diode_count1))[0]
off_1 = np.where(noise_diode_count1 < np.mean(noise_diode_count1))[0]

on_1_mean = np.mean(noise_diode_count1[on_1]) #Mean for "on" counts1
off_1_mean = np.mean(noise_diode_count1[off_1]) #Mean for "off" counts1
on_1_sem = sem(on_1) #Standard error on mean, "on" for counts1
off_1_sem = sem(off_1) #Standard error on mean, "off" for counts1
err_1 = np.sqrt(on_1_sem**2 + off_1_sem**2) #Uncertainties propagation1

print("Off counts for channel-1 : {:.2f} ± {:.2f}".format(off_1_mean, err_1))
print("On  counts for channel-1 : {:.2f} ± {:.2f}".format(on_1_mean, err_1))

########################### CHANNEL-2 ###########################
on_2 = np.where(noise_diode_count2 > np.mean(noise_diode_count2))[0]
off_2 = np.where(noise_diode_count2 < np.mean(noise_diode_count2))[0]

on_2_mean = np.mean(noise_diode_count2[on_2]) #Mean for "on" counts2
off_2_mean = np.mean(noise_diode_count2[off_2]) #Mean for "off" counts2
on_2_sem = sem(on_2) #Standard error on mean for "off" counts2
off_2_sem = sem(off_2) #Standard error on mean for "on" counts2
err_2 = np.sqrt(on_2_sem**2 + off_2_sem**2) #Uncertainties propagation2
```

```
print("Off counts for channel-2 : {:.2f} ± {:.2f}".format(off_2_mean, err_2))
print("On  counts for channel-2 : {:.2f} ± {:.2f}".format(on_2_mean, err_2))
```

At this point we are going to compute the counts per conversion factor for Left Circular Polarizatio (LCP) and Right Circular Polarization (RCP), for that we are going to use *TCAL1* and *TCAL2* respectively from *hdu[2].header*, such as their respective error bars, *TCALSIG1* and *TCALSIG2*. Conv1 represents the conversion factor for LCP and conv2 for RCP, their respective values are in Table 2. All the process is done with the following Python code:

```
"""At this step we are going to calculate the counts per
conversion factor."""

############################ CHANNEL-1 ############################
noise_temp1 = hdu[2].header['TCAL1']
conv1 = (on_1_mean - off_1_mean)/noise_temp1

on_1_sem = stats.sem(on_1) #Uncertainty "off" counts1
off_1_sem = stats.sem(off_1) #Uncertainty"on" counts1
err_11 = np.sqrt(on_1_sem**2 + off_1_sem**2 + hdu[2].header["TCALSIG1"]**2)
print("The conversion factor for channel-1 is {:.2f} ± {:.2f} Counts/K" \
.format(conv1, err_11))


############################ CHANNEL-2 ############################
noise_temp2 = hdu[2].header['TCAL2']
conv2 = (on_2_mean - off_2_mean)/noise_temp2

on_2_sem = stats.sem(on_2) #Uncertainty "off" counts2
off_2_sem = stats.sem(off_2) #Uncertainty "on" counts2
err_12 = np.sqrt(on_2_sem**2 + off_2_sem**2 + hdu[2].header["TCALSIG2"]**2)
print("The conversion factor for channel-2 is {:.2f} ± {:.2f} Counts/K" \
.format(conv2, err_12))
```

All the results related to the noise diode calibration are presented in Table 2. The table contains outputs for the noise diode temperature, ”off” and ”on” counts and for the conversion factor. From this table, we can see that all values of the LCP are higher than the RCP apart from the noise diode temperature, from which we can see that the RCP has the higher value. If we do a general overview of all the outputs in the table we can notice that there is a slightly difference between the values of the LCP and RCP, but even that they are still close.

Table 2: Noise Diode Calibration Results.

Parameter	Pol 1 Value	Pol 2 Value
Noise Diode Temperature	11.67K ± 0.50K	12.68K ± 0.50K
Off Counts	877575.64 ± 6.58	867679.69 ± 6.58
On Counts	948253.52 ± 6.58	943795.29 ± 6.58
Counts/K Factor	6056.37 ± 6.60	6002.81 ± 6.60

1.3 Drift scan fitting

Now we are going to work with drift scan data. First we need to convert from counts to antenna temperature in Kelvins, for that we have to divide the counts by the relevant counts per Kelvin conversion factor that we calculated above as conv1 and conv2 for both LCR and RCP.

The above procedure must be done at least for three points, for this project we are going to work with the north, centre and south pointing, their respective data are stored in the FITS files in index 3, 4 and 5 respectively, as well as the Modified Julian Date (MJD) that we have to convert to seconds, once the unit is in days. Lets consider the below Python code:

```python
"""This will convert the counts to Antenna Temperature
for both polarizations of North, Centre and South."""
################################## NORTH  #############################
#Antenna temperature FOR COUNTS 1 & COUNTS 2
north_T1 = hdu[3].data['Count1']/conv1  # From counts to antenna temperature
north_T2 = hdu[3].data['Count2']/conv2  # From counts to antenna temperature
north_mjd = (hdu[3].data['MJD']- min(hdu[3].data['MJD']))*24*3600 #Time

################################ CENTRE  ##############################
#Antenna temperature FOR COUNTS 1 & COUNTS 2
centre_T1 = hdu[4].data['Count1']/conv1 # From counts to antenna temperature
centre_T2 = hdu[4].data['Count2']/conv2 # From counts to antenna temperature
centre_mjd = (hdu[4].data['MJD']- min(hdu[4].data['MJD']))*24*3600 #Time

################################## SOUTH  #############################
#Antenna temperature FOR COUNTS 1 & COUNTS 2
south_T1 = hdu[5].data['Count1']/conv1 # From counts to antenna temperature
south_T2 = hdu[5].data['Count2']/conv2 # From counts to antenna temperature
south_mjd = (hdu[5].data['MJD']- min(hdu[5].data['MJD']))*24*3600 #Time
```

At this point we are going to perform the baseline correction, to help with that we are going to plot the converted antenna temperature against the converted time to seconds in order to identify the lower_mjd and the upper_mjd.

The lower_mjd and upper_mjd are both time indices that will be use to isolate the "off source" from the "on source" data. This will be done with help of the numpy functions such as np.where and np.logical_or to isolate the "off source" and np.logical_and to isolate the "on source" data.

The steps above will allow us to perform the baseline correction,to fit a 3^{rd} order polynomial to the "off source" data, and the subtract the polynomial from our data. All these steps will be done for north, centre and south pointing. Due that the following Python codes will be long, in this report we are only going to show the code for the north pointing, the code will be the same for the centre and south pointing, it will just be necessary to replace some parts of it, it will be more clear where in the next steps.

1.3.1 NORTH

Lets plot the north drift scan data for LCP and RCP so that we can idntify what the lower_mjd and upper_mjd values are. For that we need north_mjd (time), nort_T1, and nort_T2 that we have already defined in previous code. The Figure 2 is generated by the following Python code:

```python
########## LCP PLOT #############
plt.figure(figsize=(16,5))
plt.subplot(121)
plt.plot(north_mjd, north_T1)
plt.title("North_Channel-1 (LCP)")
plt.ylabel("Antenna Temperature (K)")
plt.xlabel("Time (s)")

######### RCP PLOT ############
plt.subplot(122)
plt.plot(north_mjd, north_T2)
plt.title("North_Channel-2 (RCP)")
plt.ylabel("Antenna Temperature (K)")
plt.xlabel("Time (s)")
plt.savefig("north1.png")
```

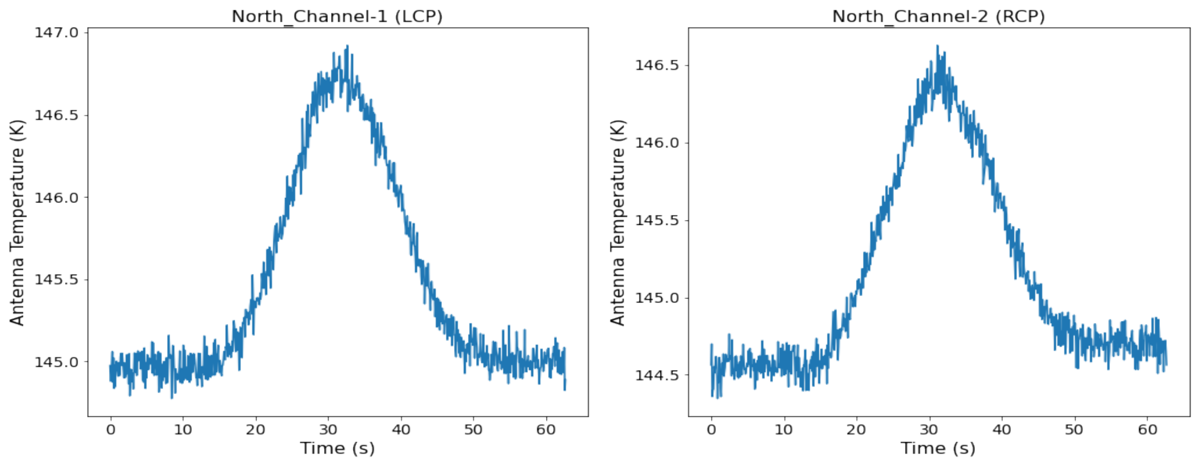

Figure 2: North LCP and RCP data.

From Figure 2 we can define by estimating the values of lower_mjd and upper_mjd for LCP and RCP, these values will allow us to isolate the ”off sorce” data from the ”on source” data and fit a polynomial to the ”off source” north data and then subtract the fitted polynomial from our north data. See the following Python code:

```python
"""At this point we want to perform a Baseline correction
/subtraction."""

# Defining the lower and upper mjd
upper_mjd = 48
lower_mjd = 15

# 1. Isolate the "off source" data from the "on source" data.
inds_off = np.where(np.logical_or(north_mjd > upper_mjd, \
north_mjd < lower_mjd))[0]
inds_on = np.where(np.logical_and(north_mjd < upper_mjd, \
north_mjd > lower_mjd))[0]


plt.figure(figsize=(16,5))# Figure size
plt.subplot(121) # LCP plot

#Plot the "on" and "off counts"
plt.plot(north_mjd[inds_on],north_T1[inds_on],color='red') #On
plt.plot(north_mjd[inds_off],north_T1[inds_off],color='blue') #Off
plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Antenna Temperature (K)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("North_Channel-1 (LCP)")

#2. Fit a polynomial to the "off source" data.
#np.polyfit(x_data,y_data,order of polynomial)
p = np.polyfit(north_mjd[inds_off],north_T1[inds_off],3)
p1 = np.poly1d(p)
plt.plot(north_mjd[inds_off],p1(north_mjd[inds_off]), \
color='black',linestyle='dashed')

#3. Subtract the polynomial.
north_T1_corrected = north_T1 - p1(north_mjd) #polynomial_subtracted.
plt.subplot(122) #
plt.plot(north_mjd[inds_on],north_T1_corrected[inds_on],color='red')
plt.plot(north_mjd[inds_off],north_T1_corrected[inds_off],color='blue')
plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Antenna Temperature (K)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("North_Channel-1_corrected (LCP)")
plt.show()
```

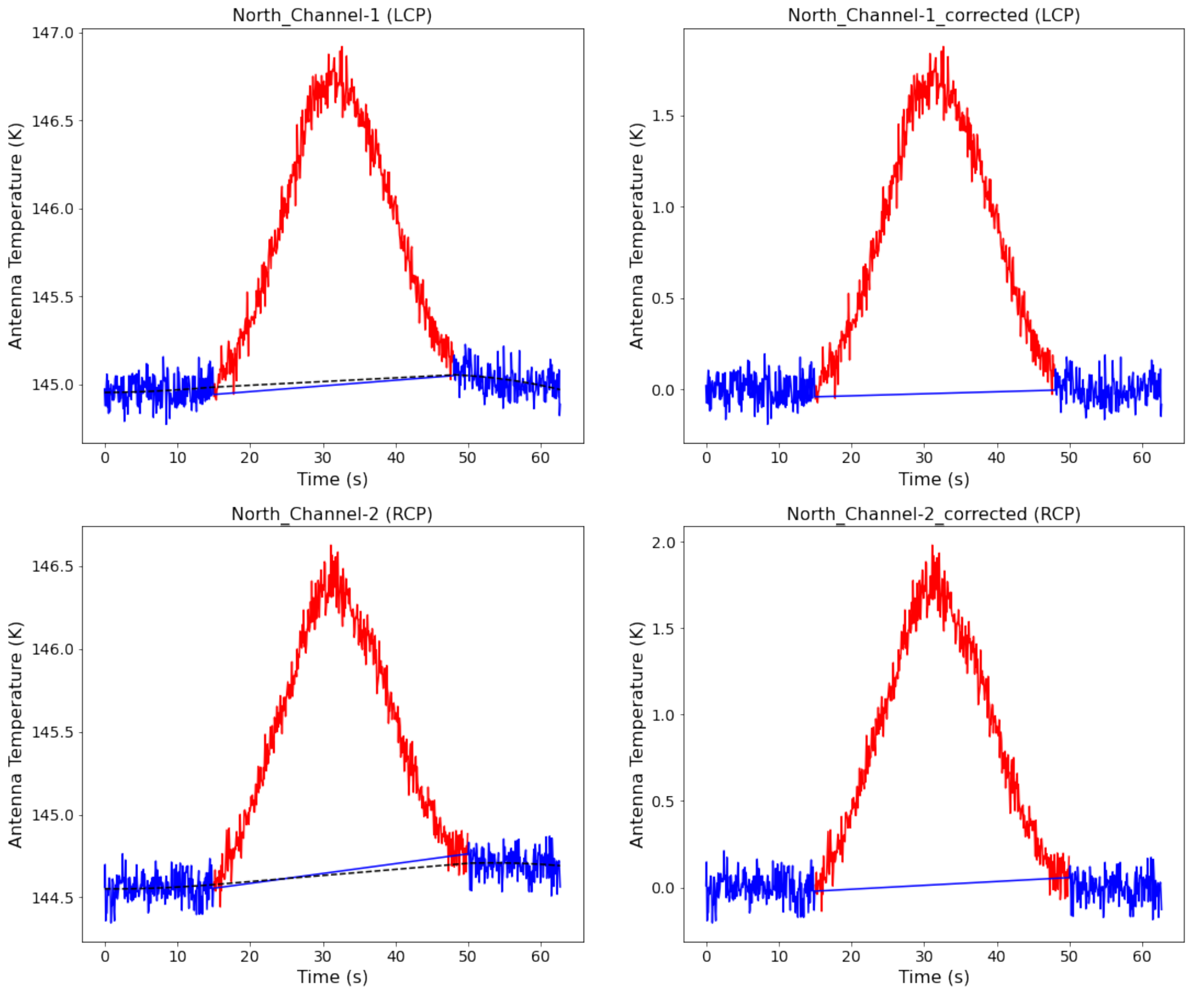

Figure 3: North LCP and RCP baseline correction.

The above Python code generated the outputs in Figure 3, from where we can see that the first row shows the LCP and the corrected LCP plots, the same in the second row but this time for RCP. The next step is to fit the Gaussian to the LCP and RCP corrected data for baseline in order to get the peak. Consider the code below:

```
#Fit a gaussian to get the peak.
import scipy.optimize as optim
def gaussian(x,A,mu,sigma):
    return A*np.exp(-(x-mu)**2/(2*sigma**2))
#Use a gaussian by using the function "gaussian(x,A,mu,sigma)"
popt_north,pcov_north = optim.curve_fit(gaussian,\
                                        north_mjd[inds_on],\
                                        north_T1_corrected[inds_on],\
                                        p0=[2.0,33,17])
```

```
"""popt_north is an array with the fitted parameters.
In this case the A, mu and sigma.
pcov_north is a 3x3 matrix which gives the errors of the fit.
To get the errors use np.sqrt(np.diag(pcov_north))
"""
x = np.arange(min(north_mjd),max(north_mjd),0.01)

plt.figure(figsize=(17,7))
plt.subplot(121)

plt.plot(north_mjd,north_T1_corrected)
plt.plot(x,gaussian(x,*popt_north))
plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Antenna Temperature (K)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("North_Channel-1_corrected (LCP)", fontsize=16)
print('north peak is {:.2f} ± {:.2f}'.format(popt_north[0],\
                                np.sqrt(np.diag(pcov_north))[0]))

popt_north2,pcov_north2 = optim.curve_fit(gaussian,\
                                        north_mjd[inds_on],\
                                        north_T2_corrected[inds_on],\
                                        p0=[2.3,30,17])

plt.subplot(122)

plt.plot(north_mjd,north_T2_corrected)
plt.plot(x,gaussian(x,*popt_north2))
plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Antenna Temperature (K)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("North_Channel-2_corrected (RCP)", fontsize=16)
plt.savefig("north3.png")

print('The amplitude of the drift scan is %.2f ± %.2f K'%(popt_north2[0],\
                            np.sqrt(np.diag(pcov_north2))[0]))
```

Let us now fit the Gaussian to the corrected data so that we can estimate the peak values. The north amplitude peak for LCP and RCP values can be seen in Table 3.

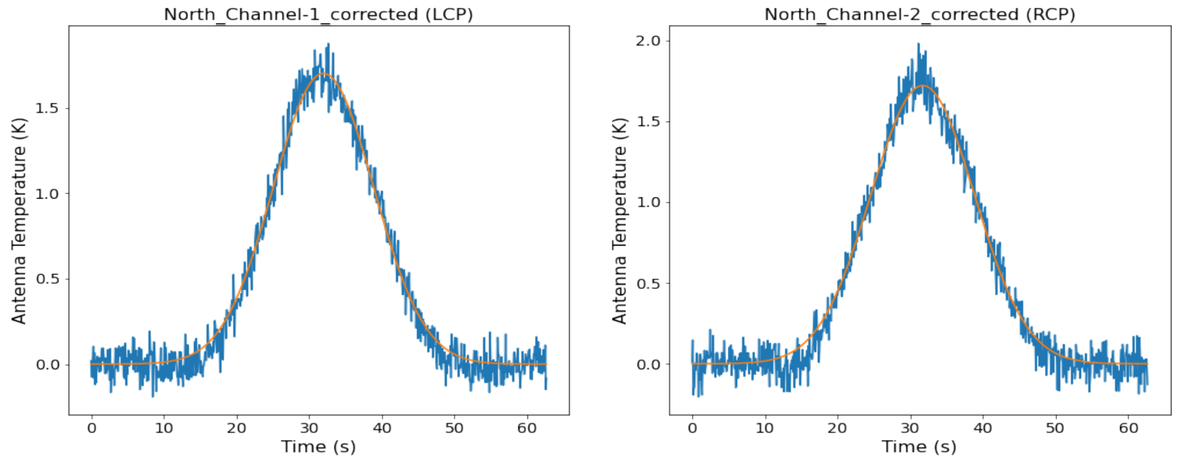

Figure 4: North LCP and RCP Gaussian fit.

We are done with the north pointing. Now that we have explained all the code steps for the north pointing, we are going to use the same code for the centre and south pointing, we just have to change the values of the lower_mjd and upper_mjd with new values according to the plot of the north and south data. We have to change the variable names as well to suit the centre and south pointing. Let us now show the plots for the centre and south:

1.3.2 CENTRE

Plotting the north data so that we can read the lower and uppwer mjd:

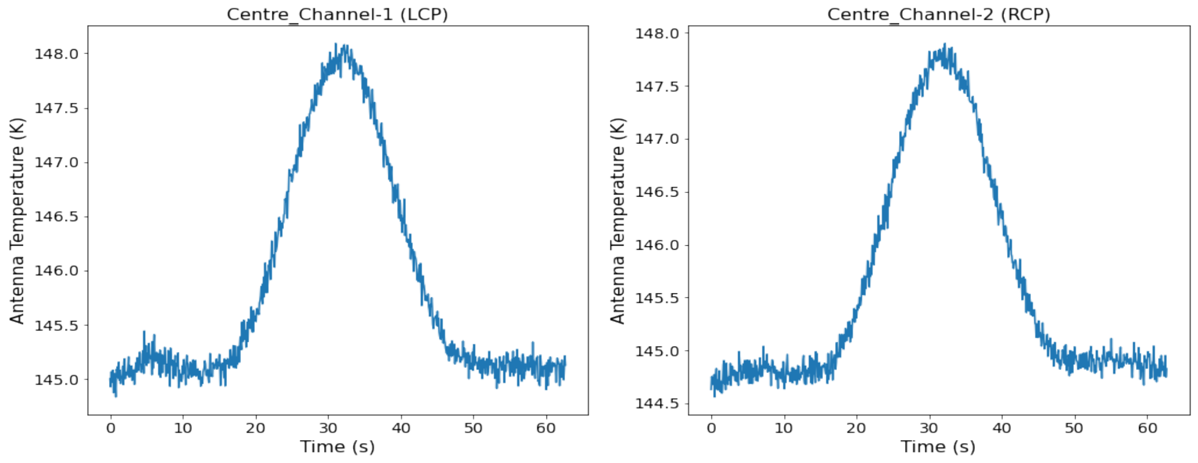

Figure 5: Centre LCP and RCP data.

Performing the baseline correction:

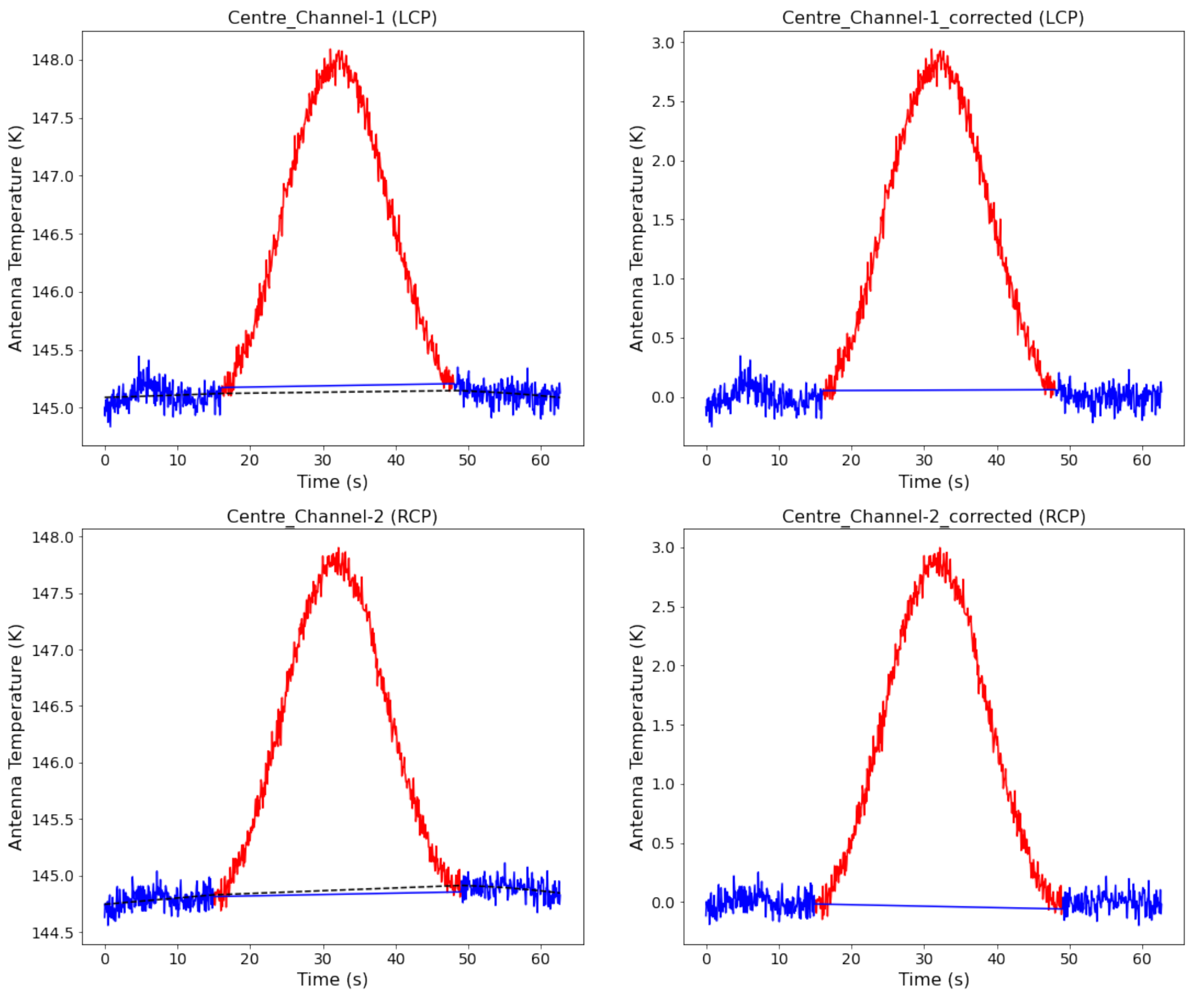

Figure 6: Centre LCP and RCP baseline correction.

Let us now fit the Gaussian to the corrected data so that we can estimate the peak values. The centre amplitude peak for LCP and RCP values can be seen in Table 3.

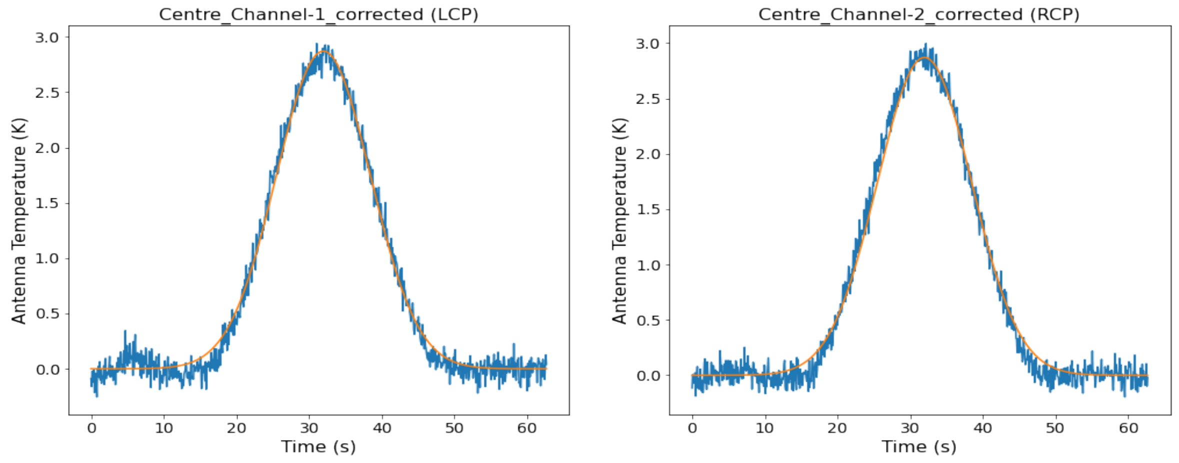

Figure 7: Centre LCP and RCP Gaussian fit.

1.3.3 SOUTH

Plotting the south data so that we can read the lower and uppwer mjd:

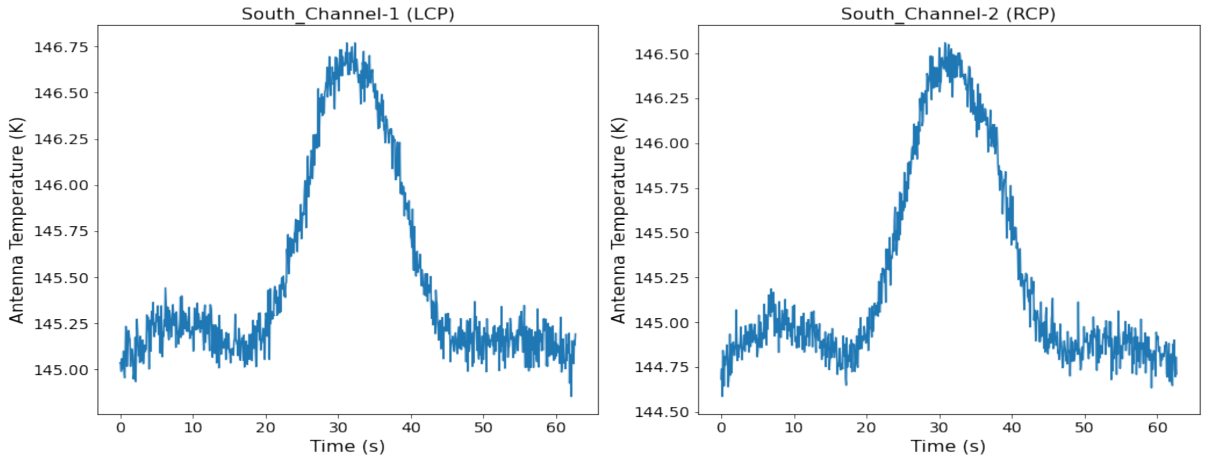

Figure 8: South LCP and RCP data.

Performing the baseline correction:

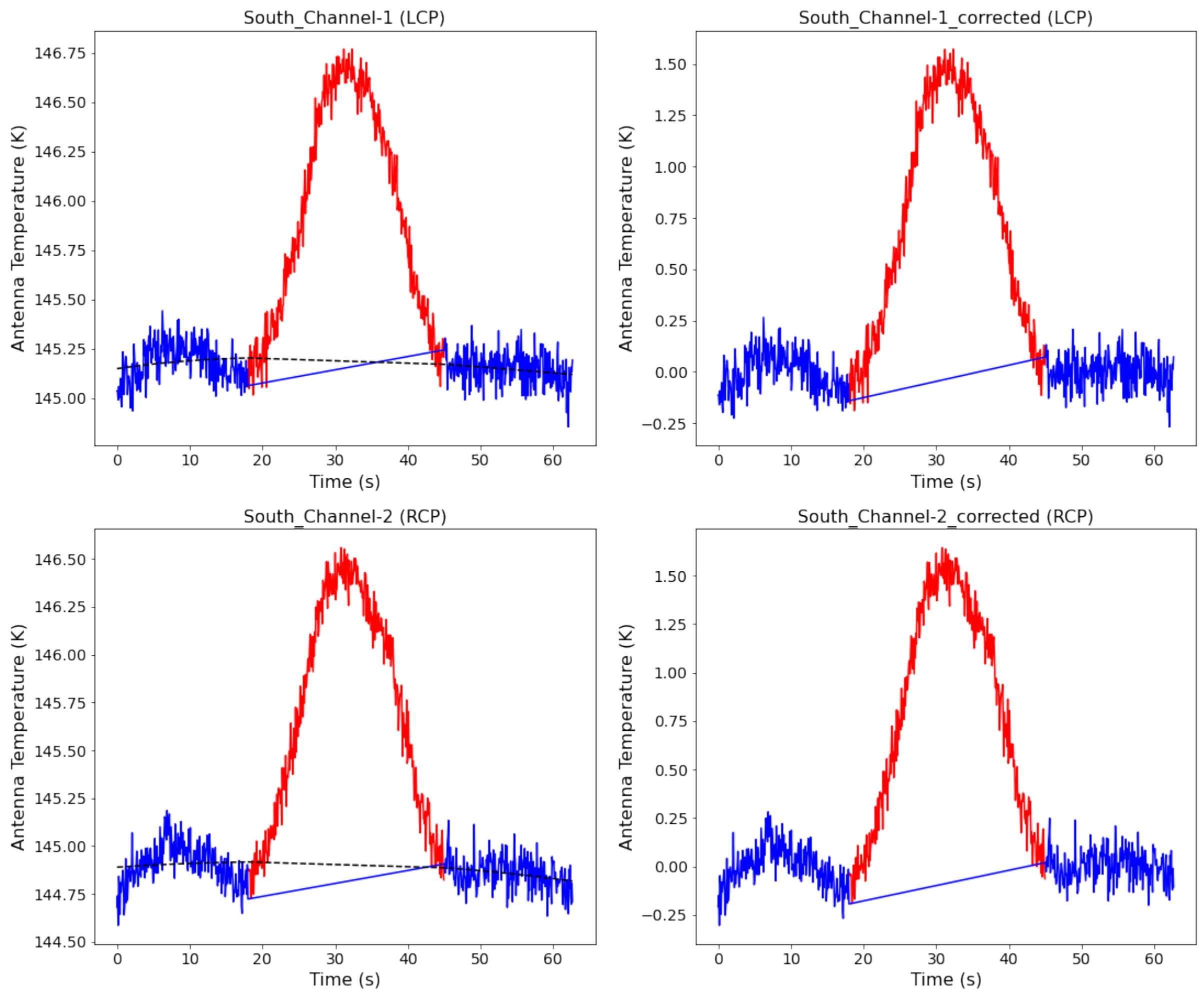

Figure 9: South LCP and RCP baseline correction.

Let us now fit the Gaussian to the corrected data so that we can estimate the peak values. The centre amplitude peak for LCP and RCP values can be seen in Table 3.

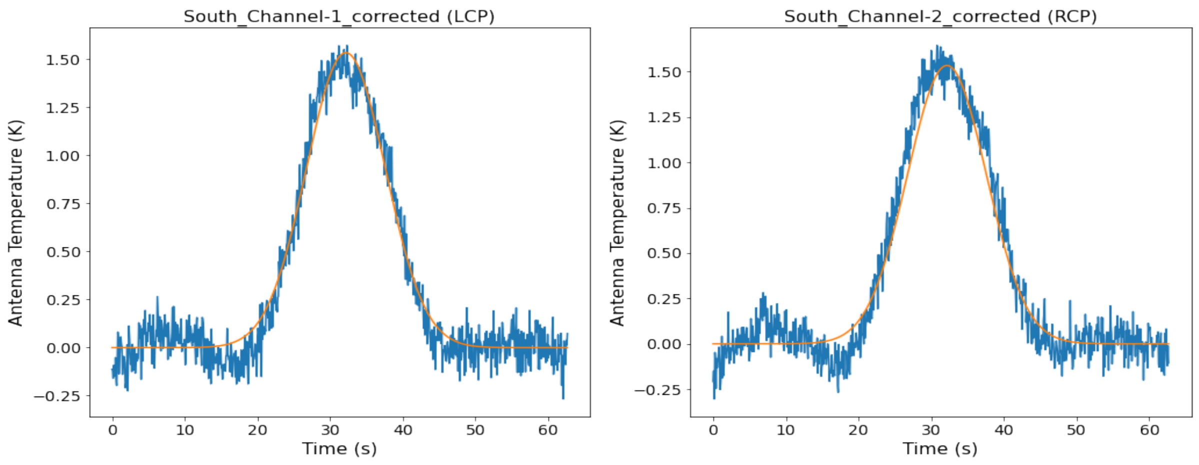

Figure 10: South LCP and RCP Gaussian fit.

The Table 3 summarizes all the peak values of the corrected values for LCP and RCP for north, centre and south pointing. It can be seen that the highest are in the center, in both polarizations.

Table 3: Drift Scan fitting Results.

Parameter	Pol 1 Value	Pol 2 Value
North Peak	1.71 ± 0.01 K	1.73 ± 0.01 K
Centre Peak	2.87 ± 0.01 K	2.93 ± 0.01 K
South Peak	1.53 ± 0.01 K	1.60 ± 0.01 K

1.4 Pointing correction

At this stage we are ready to plot the pointing correction for both polarizations. For this we plot the amplitudes of the north, centre and south, and then we fit a Gaussian. We do this process for the two polarizations. From the Figure 11 we can see that for polarization 2 the north and centre amplitudes are slightly off the Gaussian fit. Consider the Python code bellow:

```
#Now we have fitted the North, South and Centre scans.
x = [0,1,2]
scan_amplitudes = [popt_south[0],popt_centre[0],popt_north[0]]
plt.figure(figsize=(16,5))
plt.subplot(121)
plt.scatter(x,scan_amplitudes)

plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Antenna Temperature (K)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Pointing_Correction_1", fontsize=16)
#Pointing correction:
popt_pointing, pcov_pointing  = optim.curve_fit(gaussian,x, \
                                    scan_amplitudes,p0=[2.7,0.5,1])
x = np.arange(-2,2,0.01)
plt.plot(x,gaussian(x,*popt_pointing))
#Pointing correction factor:
print('Pointing correction factor: %.3f'%(popt_pointing[0]/popt_centre[0]))
pointing_correction1 = popt_pointing[0]/popt_centre[0]

###########################
plt.subplot(122)

x2 = [0,1,2]
scan_amplitudes2 = [popt_south2[0],popt_centre2[0],popt_north2[0]]
plt.scatter(x2,scan_amplitudes2)

plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Antenna Temperature (K)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Pointing_Correction_2", fontsize=16)
#Pointing correction:
popt_pointing2, pcov_pointing  = optim.curve_fit(gaussian,x2, \
                                    scan_amplitudes2,p0=[2.7,0.5,1])
x = np.arange(-2,2,0.01)
plt.plot(x,gaussian(x,*popt_pointing))
#Pointing correction factor:
print('Pointing correction factor: %.3f'%(popt_pointing2[0]/popt_centre2[0]))
pointing_correction2 = popt_pointing[0]/popt_centre2[0]
plt.show()
plt.savefig("point.png")
```

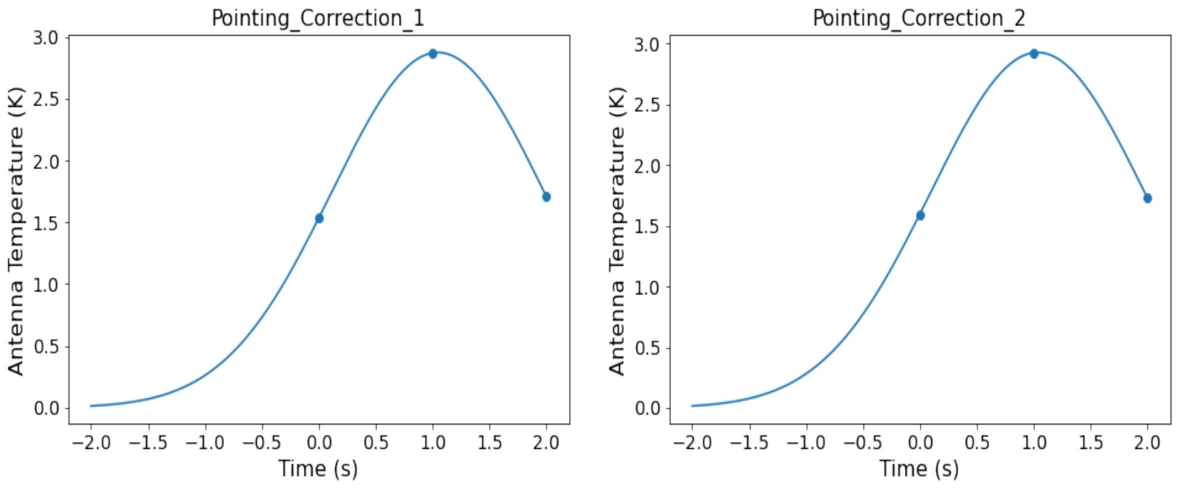

Figure 11: Pointing correction plots.

It happens that I had the same values of pointing correction factor for LCP and RCP, it corresponds to 1.001.

1.5 Flux density calculation

Now we are going to calculate the PSS for each polarization, but first we need to calculate the flux, and this is obtained by applying the equation The above code gives us the below outputs:

```
"""Calculate the PSS. The PSS is a factor that you
multiply to your antenna temperature to get flux density."""
#PSS has units of Jy/K.
nu = hdu[2].header["CENTFREQ"] # frequency of observation
def ott(nu,a,b,c):
    return 10**(a+b*np.log10(nu)+c*np.log10(nu)**2)

print("The flux density at {} MHz for Virgo A is {:.2f} Jy" \
      .format(nu,(ott(nu,4.484,-0.603,-0.0280))))
#The final answer.
VirA_flux  = ott(nu,4.484,-0.603,-0.0280)

PSS_1 = VirA_flux / (popt_centre[0]*pointing_correction1)
PSS_2 = VirA_flux / (popt_centre2[0]*pointing_correction2)


#The final answer.
print("PSS for polarization (1) for this observation is {:.2f} ± {} Jy/K" \
      .format(PSS_1, np.sqrt(np.diag(pcov_centre))[0]))
print("PSS for polarizatio (2) for this observation is {:.2f} ± {} Jy/K" \
      .format(PSS_2, np.sqrt(np.diag(pcov_centre2))[0]))"
```

The above code gives us the below outputs:
The flux density at 12178.593 MHz for Virgo A is 35.73 Jy

The PSS for this polarizatio (1) for this observation is 12.43 ± 0.01 Jy/K
The PSS for this polarizatio (2) for this observation is 12.20 ± 0.01 Jy/K

```
plt.figure(figsize=(16,7))
plt.subplot(121)
plt.plot(centre_mjd,centre_T1_corrected*PSS_1)

plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Flux Density (Jy)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Centre_T1_Flux_Density",fontsize=16)

plt.subplot(122)
plt.plot(centre_mjd,centre_T2_corrected*PSS_2)

plt.xlabel('Time (s)',fontsize=16)
plt.ylabel('Flux Density (Jy)',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.title("Centre_T2_Flux_Density",fontsize=16)
plt.show()

plt.savefig("flux.png")
```

The figure 12 shows us the plots for the LCP and RCP flux density. These are plotted using the centre_mjd against the respective corrected data.

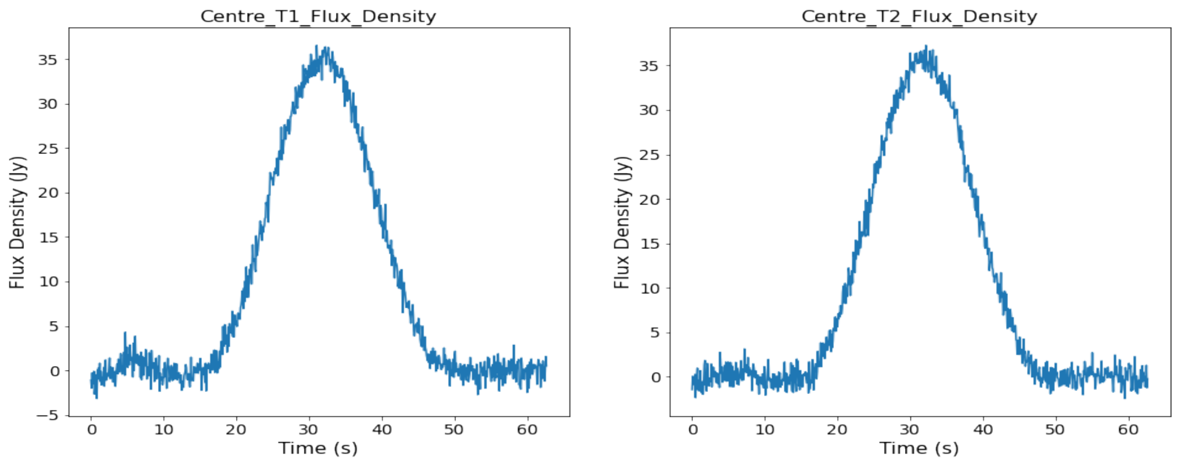

Figure 12: Flux density for center LCP and RCP.

The equation bellow is for the absolute flux density, from the Ott et al., paper. Ott et al., have used the equation to perform absolute flux calibration for many sources, they were updating the table of the sources that can be use as calibrators. For that they measured the BGPW source to reduce the internal inconcistencies. They used the wavelength in range of $0.9cm \leq \lambda \leq 21cm$.

$$\log S[Jy] = a + b * log\nu[MHZ] + c * log^2\nu[MHZ]$$

Where:
S is the flux density;
ν is the frequency of observation;
a, b, c are the constants of the source for calibration.

2 Conclusion

We followed all the steps needed to do an absolute flux density calibration of the HartRAO drift scan for the 26m antenna. We could see from the outputs that the 26m is slightly off pointing. Two things did call my attention, first, the fact that the values of the pointing correction for both polarization are the same, the second one is the value of the PSS for LCP and RCP that are close to each other, but that makes sense once the value of pointing correction factor is exactly the same for LCP and RCP. My question now would be, "in what situations we have the same value for pointing correction factor"? We calculated as well as their respective error bar. I think the baseline correction looks good and we have nice Gaussian fits in general.