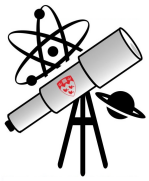
McGill Radio Lab



Introductory Beam Analysis : A Relatively - Comprehensive Review

Report by: Mattias Lazda

McGill University Department of Physics

August 28, 2020

Purpose

The purpose of this report is to provide insight into the methods used to carry out general beam analysis relevant to projects related to the McGill Radio Lab (i.e. PRIzM, ALBATROS, etc.). This report will guide you through the code required to understand and carry out basic beam analysis techniques and will be broken down into the following sections: 1. Introduction; 2. PyGSM and GlobalSkyModel; 3. Principle Component Analysis; 4. Beams; 5. Sky Brightness; 6. Loading in Simulated PRIzM Beams; 7. Interferometry, ALBATROS & Phase Plots and 8. GSM Believability. Each section will clearly mention the python file required to produce the results available here on Github. **Note that the goal of this report is to not delve into the theory behind the code, but to instead focus on getting the code up and running.** Lastly, this report will not contain any of the actual code: however, the code will be available on github and will be referenced appropriately. Figures will be attached within this report to assure that the reader can produce the correct results before moving on to the following sections. For any clarifications, feel free to contact me via email at mattias.lazda@mail.mcgill.ca.

---

1 Introduction

The first thing you'll want to do is make sure you have Python3 installed and are familiar with either the Python Shell and/or a Python IDE (I used Jupyter Notebook). Next, you'll want to make sure you have installed all the required Python packages required to run the code. This can be done simply by typing **'pip install name_of_package'** in either your virtual environment or a Jupyter cell. The following packages will be required: healpy, h5py, numpy, astropy, scipy, ephem, datetime and matplotlib.pyplot. Note that healpy only runs on MAC OS X and Linux, not Windows.

Once you have installed these packages, you will then want to download the zip file called "**Required Files**" available on github. This will contain the two following files: pygsm.py, pygsm2016.py. You will need to download gsm_components.h5 and gsm2016_components.h5 from HERE. The first two files contain the code that generates the map of the sky at a specific frequency using the data of the last two files. **Note that I did not write this code. I made slight adjustments to the original code (available HERE) in order to have it fit my needs.** It came to my attention that the original publishers have been working on similar scripts but using results from the Dowell et. al. (2017) paper. I did not get the chance to work with it: however, it should be similar to that of the GlobalSkyModel() and GlobalSkyModel2016() classes present in the pygsm scripts.

Once all of this has been completed, you should now be equipped with the software to carry out the beam analysis.

2 PyGSM and GlobalSkyModel

Overview:

This section will introduce you to the main class you'll be using. The GlobalSkyModel class (using results from Oliveira-Costa et al., (2008) [1])can be found within the pygsm.py file and is used to generate maps of the sky at varying frequencies (see 1). Likewise, the GlobalSkyModel2016 class can be found within the pygsm2016.py file. The only difference between the two is that the 2016 is based off of Zheng et. al.'s, (2016) [2] paper and has more accurate data. The downside is that the files contain more data and therefore take longer to run/manipulate. When learning a new task, it is more efficient to use the 2008 version. Once you have it down, move to the 2016 version to get the more accurate results. Keep an eye out for any updated maps to use the most up-to-date results (i.e. the LFSS methods mentioned in Section 1). See 'intro_to_gsm.py' for examples. Here are some sample plots of what you should see:

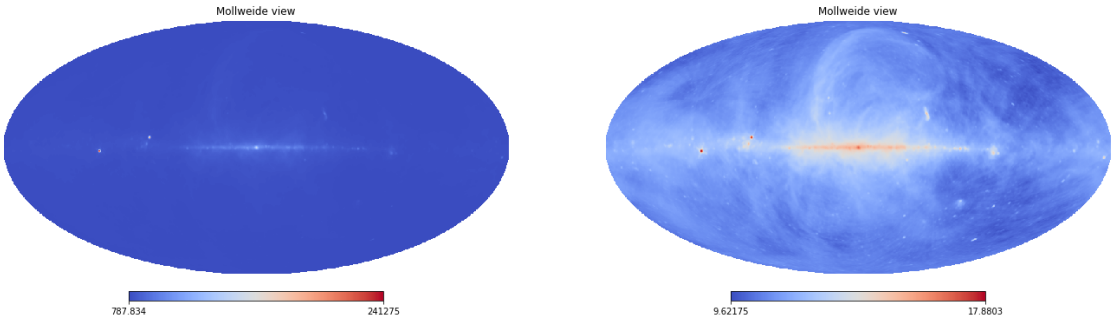


Figure 1: Map of sky at 70MHz produced using GlobalSkyModel(2008). Left has had no alteration to temperature. Right has taken the $\log_2$ of the temperature.

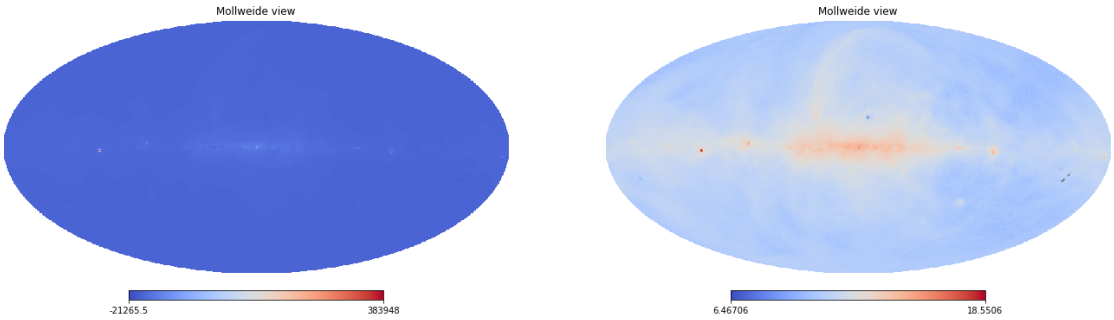


Figure 2: Map of sky at 70MHz produced using GlobalSkyModel(2016). Left has had no alteration to temperature. Right has taken the $\log_2$ of the temperature.

3 Principle Component Analysis

Overview:

The goal of this section is to develop the ability to perform a Singular Value Decomposition (SVD) on the mxn matrix containing m map data points for n frequencies. It will give you a better understanding of how the Global Sky Model **generate** function works. The resulting principle components give information about how the sky acts as a function of frequency and can be used to approximate the sky. See 'PrincipleComponentAnalysis.py' for example. Plots should resemble the following:

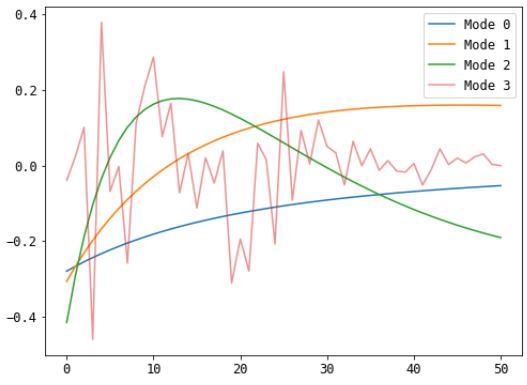


Figure 3: Sample plot of the 4 first principle components of a mx50 matrix for a frequency range of 50MHz - 100MHz.

4 Beams

Overview:

In this section we will discuss a lot about the basics of the beams and how to apply them. We will break it down into multiple sub-sections, adding more details as we go along.

4.1 Healpy Methods

Healpy (Healpix but for python) is a python package used to simplify working with spheres and is necessary to create simulated beams. Here, we will discuss all the healpy methods I used as well as key definitions related to healpix, so feel free to refer to this if ever the healpy documentation isn't enough.

1. nside:

Nside is probably the most important healpix property. Essentially, nside determines the resolution of your map. The lower the value of nside, the lower the resolution of the map and vice versa. The value of nside must be a positive power of 2, i.e:

$$\text{nside} = 2^x, x \epsilon \mathbb{N}^+. \tag{1}$$

Once you've successfully generated a map (see subsection **??**), change the nside and notice the difference in the resolution of the image. If you are using GlobalSkyModel to generate your maps, the default nside is 2^9, whereas it is 2^{10} for the GlobalSkyModel2016 maps. Be careful though! Since nside increases exponentially, the higher you push the nside, you start to trade increased resolution for longer runtimes. Anything above 2^{10} becomes very long to work with. In section 8.1 we discuss when it's best to use a certain nside.

2. npix:

npix is the number of pixels in your image. It is dependent on the value of nside and can be computed using the **healpy.nside2npix(nside)** function.

3. healpy.pix2ang(nside, pixarray):

The **pix2ang** function converts the position of each pixel into its (θ, ϕ) position, where $\theta\epsilon[0, \pi]$ represents the latitude and $\phi\epsilon[0, 2\pi]$ represents the longitude.

4. healpy.mollview(array):

Mollweide view's the array. Used to plot the maps/beams etc.

5. healpy.map2alm(map):

Converts a healpix map into its spherical harmonic coefficients (alm's). Only includes m $\geq$ 0 coefficients. Working in alm-space is crucial to speeding up and making your code more efficient.

6. healpy.alm2map(alms, nside):

Converts alm coefficients back to map space. Useful once you've finished working in alm space and want to see the resulting map of the sky or beam.

7. $\mathbf{l_{max}}$:

l_{max} is the largest value of l for a given set of alm's. Recall that for l = 0, m = 0; l = 1, m = -1, 0, 1; l = 2, m = -2, -1, 0, 1, 2, etc. The value of l_{max} can be determined using the following equation:

$$l_{max} \approx \sqrt{2 \times N} - 0.5 \tag{2}$$

where N is the number of alm coefficients making up the map.

4.2 Creating your first beam

When it comes to coding your first beam, it's probably a good thing if you know what you are working towards. A beam in this scenario is an area that covers the sky (shape depends on the type of beam that you use) that will be used to calculate the average

temperature of the sky in that area. For now, we will focus on using a Gaussian beam as it is the most similar to the actual beams that the McGill Radio Lab (MRL) uses while also being easy to manipulate. Refer to **creating_your_first_beam.py** for a detailed walk-through.

Your beam should look like this (though maybe slightly different depending on which function you use and/or your full width half max (fwhm)):

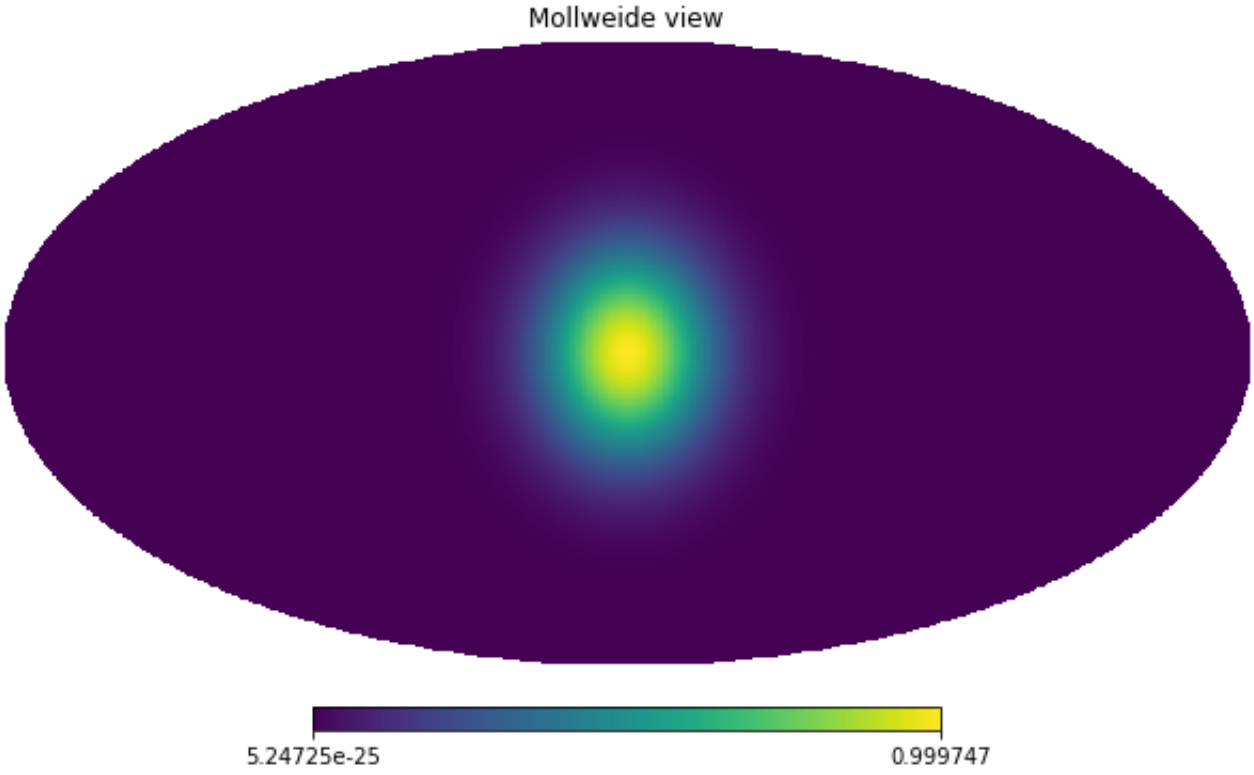


Figure 4: Example of what your beam should look like.

4.3 Rotating Your Beam

In this section, we will discuss how to rotate your beam longitudinally (i.e. changing ϕ). When you are ready to use your beam, set the latitude you would like to work at (Marion island is at $\theta = 137\,\text{deg}$) and the longitude to 0. If you would like to change the longitude, **do not** change the center of the beam by changing **beam_cent** in the code from section 4.2. Instead, use the method discussed here.

We will be working in spherical harmonic space and will therefore be utilizing alm-

related methods discussed in section 4.1. Refer to **rotating_beam.py** for a walk through. Here is the beam from the previous section (see Figure 4) after applying a ϕ rotation of $\frac{\pi}{2}$:

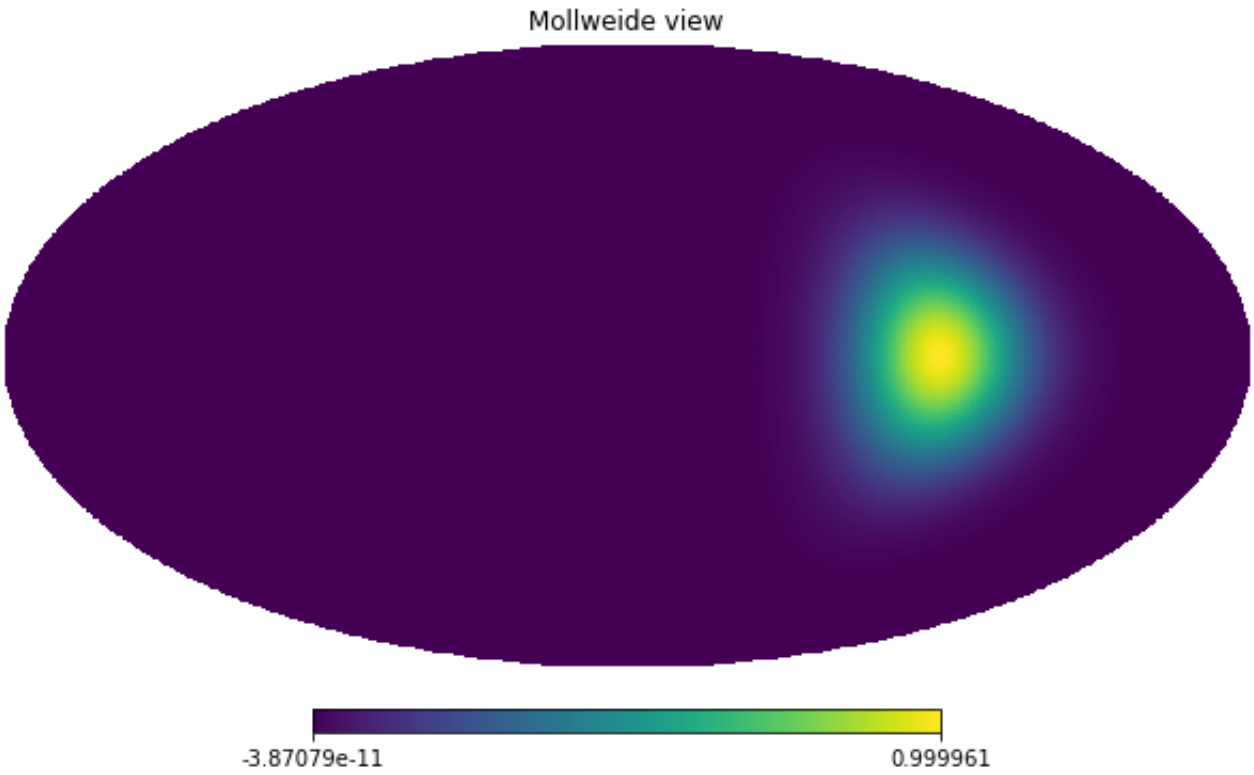


Figure 5: Beam from Figure 4 rotated by $\frac{\pi}{2}$

5 Sky Brightness

Overview:

In this section, we will look into three methods of determining the average brightness of the sky. The first method revolves working solely in map space, whereas the second is done in alm space. The first method is much slower than the second method, so I would recommend using the second method as often as possible. Note that the base equation for calculating the average temperature of the sky is:

$$T = \frac{\int \text{beam} \times \text{map}}{\int \text{beam}} \tag{3}$$

This is equation is manipulated in such a way to simplify working in alm-space. For details, contact Prof. Sievers. See **sky_brightness.py** for the two methods used. Here is a sample of what you should see:

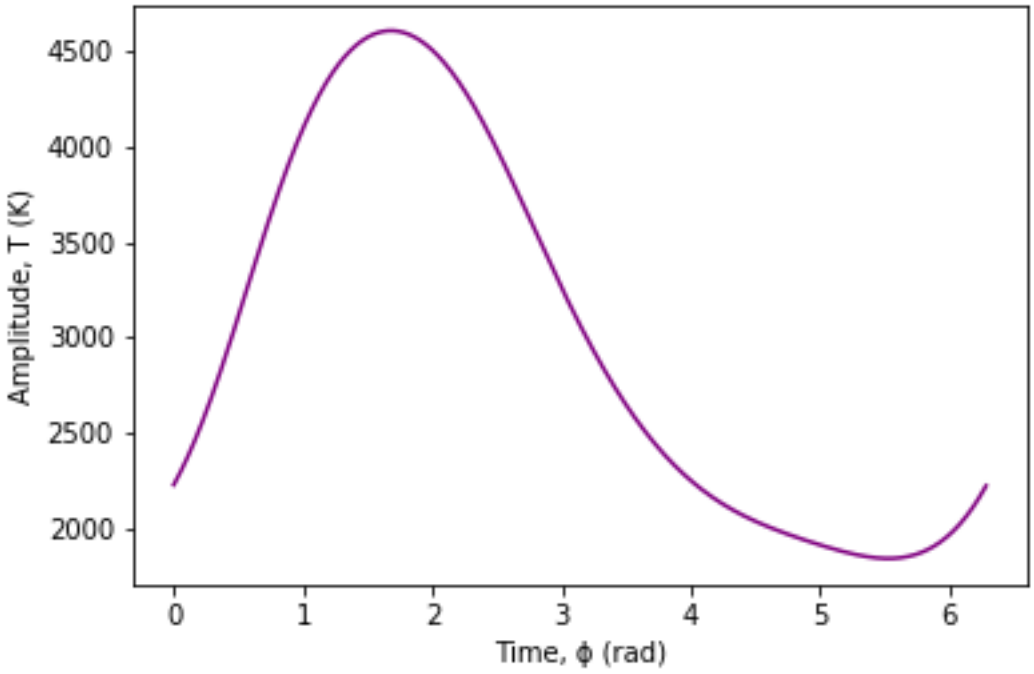


Figure 6: Average Sky Brightness over one day at $\theta = 137deg, \nu = 70MHz$ using a Gaussian beam and GlobalSkyModel maps.

Note: To create a waterfall plot, create a matrix where each column is the result of the Fast Fourier Transform for a specific frequency. You should get a plot that resembles the following:

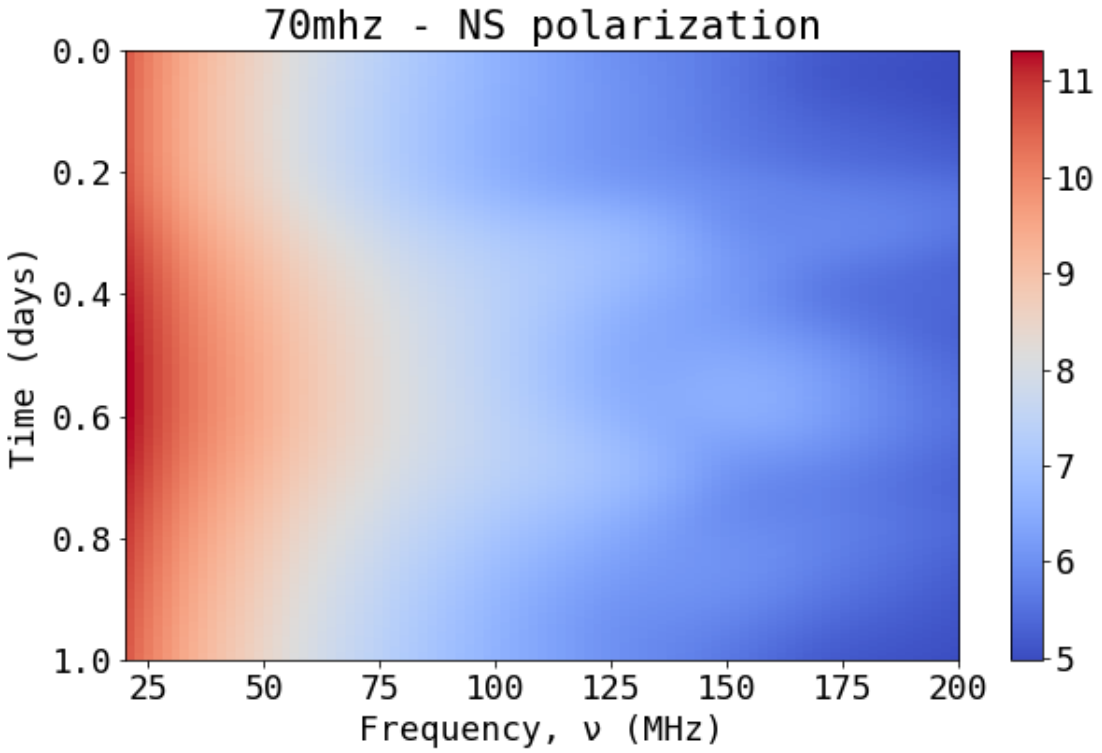


Figure 7: Sample Waterfall Plot. The title is in reference to the polarization of the simulated PRIzM beam which we will discuss later on. Make sure to use plt.imshow() as well as plt.axis('auto') on the mxn matrix.

6 Loading in Simulated PRIzM Beams

Overview:

In this section, we will discuss how to load in the simulated beams. Note that by the time you are reading this, the ALBATROS simulated beams will probably be out, so this may not be as relevant. These beams give a better approximation of the real beams used and often produce more realistic results than when using the Gaussian beam. The code was originally written by Prof. Sievers (as with a lot of the code in this report!) but I will explain how to use it. The main edit I made to the original code was adding a 'pol' parameter signifying the polarization of the beam. 'EW' stands for the East-West polarization and 'NS' stands for North-South polarization. See **'prizm_beam.py'** for details. You will need to download the actual beam files: Ask Prof. Sievers for details.

7 Interferometry, ALBATROS & Phase Plots

Overview:

In this section, we will look into how to generate amplitude and phase plots related to ALBATROS. This section will probably be most useful to you if you are working with interferometry. The code I will provide is using GlobalSkyModel 2016 maps and PRIzM simulated beams: however, other beams and sky maps may be used without having the need to change too much.

Note: It is extremely important that you center your beam correctly. If your beam is not centered properly, your plots will struggle to resemble anything useful and will lead to frustration.

7.1 Centering Your Beam

To center your beam, you are looking for the beam rotation such that

$$\text{beam} \cdot \text{baseline} = 0, \tag{4}$$

where the baseline is the vector between the two LWA antennas, and your 'beam' is actually just a small dot. See part 1 of **interferometry.py** for an example.

7.2 Amplitude and Phase Plots

Once you've found the center of your beam, rotate all your simulated beams (in my case, all the PRIzM beams) to match the proper beam center. You're now safe to run the code and should expect 'believable' plots. See part 2 of **interferometry.py** for a walk through of the functions used. Here is a sample waterfall plot produced using the code:

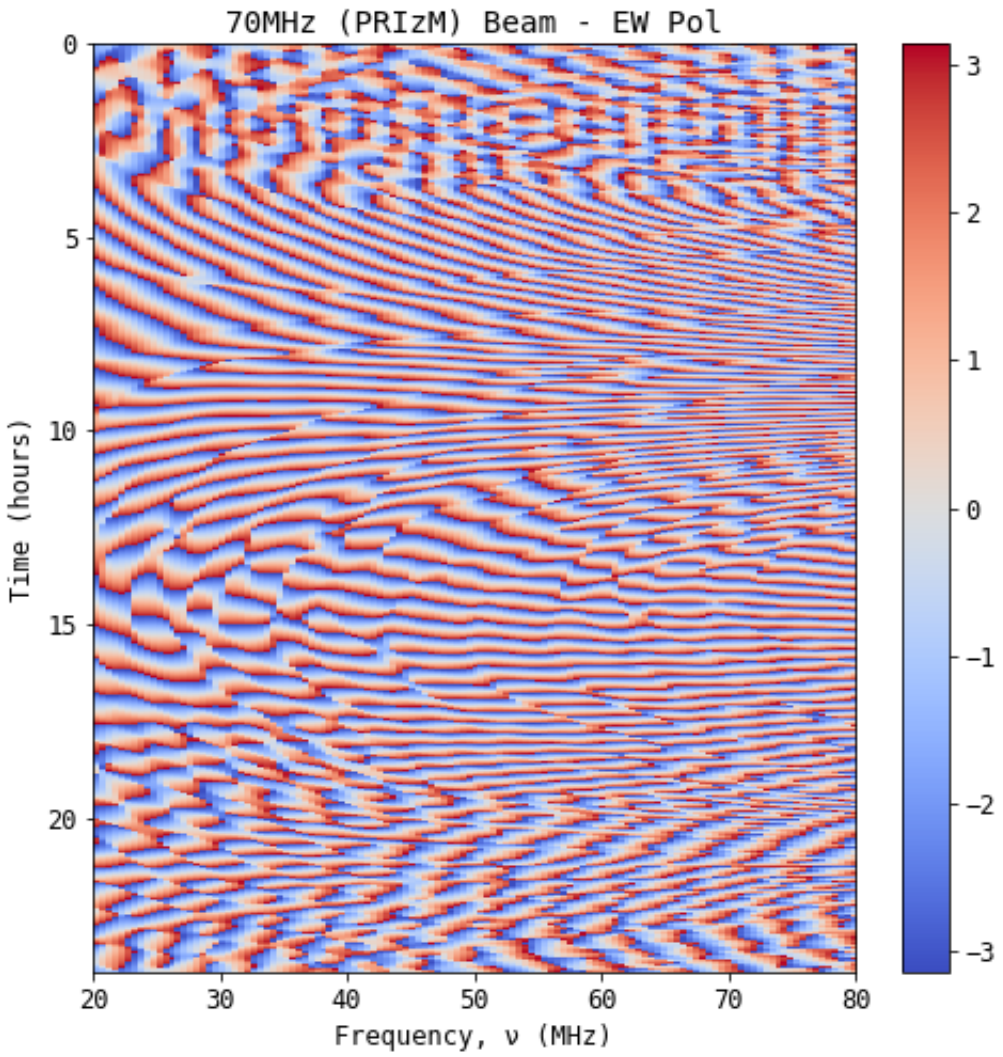


Figure 8: Sample phase plot produced using GlobalSkyModel 2016 and PRIzM simulated beams.

8 GSM Believability

Overview:

In this section, we will discuss how I began to look into how much of the GlobalSkyModel maps we should actually believe. This section will be broken down into three

subsections: Determining which nside to use, comparing GSM to other low frequency maps and analyzing the power spectrum of the beams. I'd like to note that I was unable to complete this section and thus it may be a good spot to pick up off from.

8.1 Determining which nside to use

Often, when time is limited, we want to be able to decrease the runtime of the code. For that reason, I set out to determine what nside you need to use depending on the frequency (this applies more for the interferometer). Recall that the baseline is proportional to the frequency. In fact, as you increase the frequency, the baseline increases and the narrower the fringes become. Consequently, the higher you push the frequency, the higher nside you'll need in order to make out the details in the interferometer. Here is a plot summarizing the results of what I found:

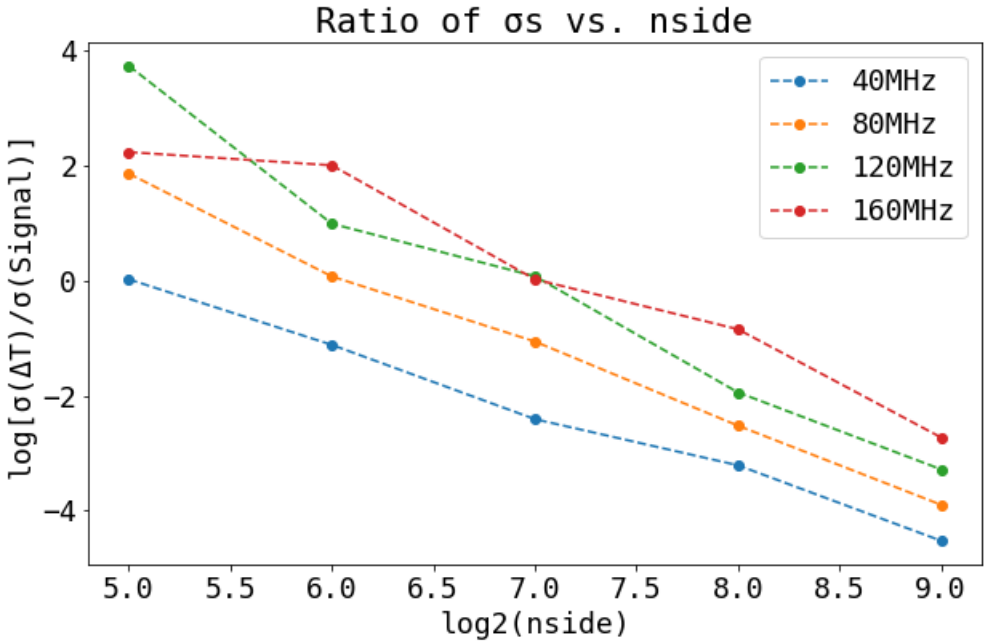


Figure 9: Plot of log of the ratio of standard deviations as a function of nside. Nside = 2^{10} was used as the reference nside. It is suggested that for low frequencies (up until 80MHz) the difference is low enough to use nside = 2^9. However, if you are working with frequencies greater than 80MHz, more tests are warranted and it is possible that nside = 2^{10} or 2^{11} may be needed when working with the interferometer.

I've included the code: however, it isn't going to be as detailed as the others. It could however be used as reference in case one would like to see what I did. See **nsidevsfreq.py**.

8.2 Comparing GSM to OVRO LWA Maps

I've included this section to provide a comparison between GlobalSkyModel 2016 and the OVRO LWA maps available at https://lambda.gsfc.nasa.gov/product/foreground/fg_ovrolwa_radio_maps_get.cfm. This is simply to give an idea of how well GSM compares with more recent data.

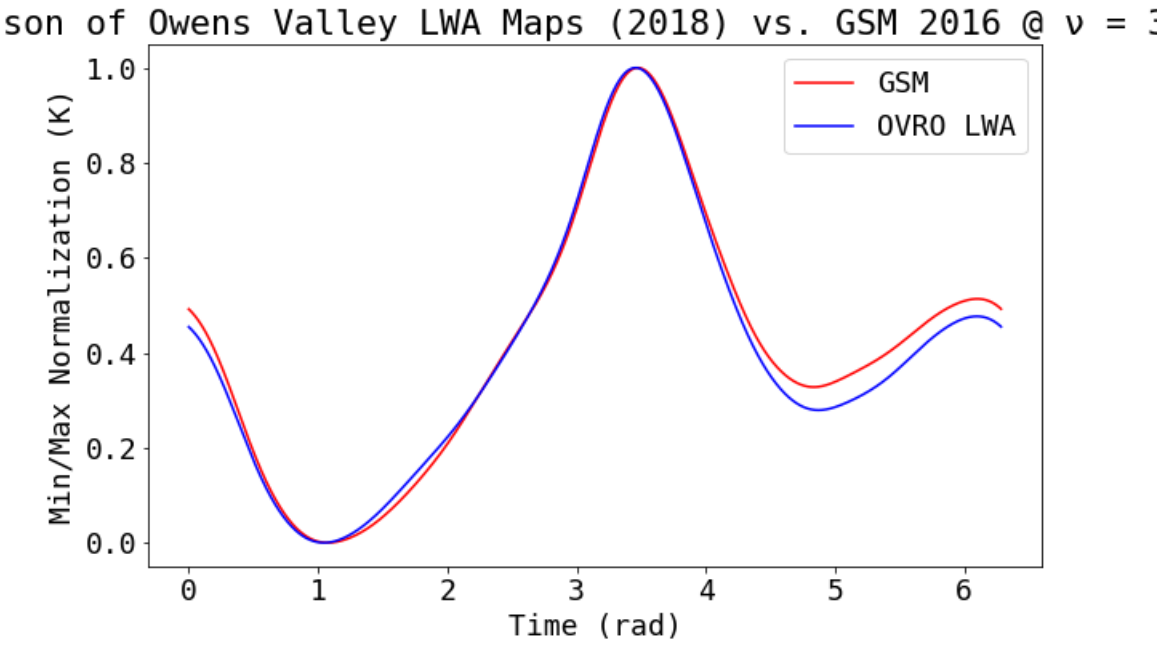


Figure 10: Comparison of OVRO LWA Maps Vs. GlobalSkyModel 2016 Maps at 36MHz. A min/max normalization has been applied to the amplitudes.

8.3 Power Spectrum

In this final section, we will look into the power spectrum of the PRIzM beam and in the future, compare it to the power spectrum of the OVRO beam. I was able to produce these plots using the PRIzM beam and code from **powerspec.py**. You will want to use this as reference when trying to find the power spectrum of the OVRO beam.

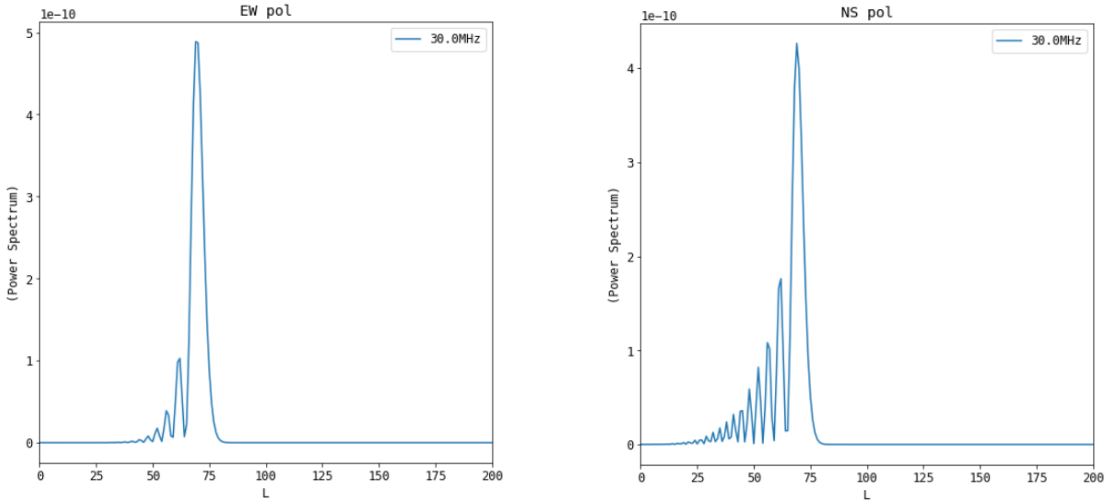


Figure 11: Power Spectrum of PRIzM beams at 30.0MHz. EW and NS polarizations both included.

References

[1] Oliveira-Costa et al. (2008). A model of diffuse Galactic Radio Emission from 10 MHz to 100 GHz. https://doi.org/10.1111/j.1365-2966.2008.13376.x

[2] Zheng, H. et al. (2017). An Improved Model of Diffuse Galactic Radio Emission from 10 MHz to 5 THz. Monthly Notices of the Royal Astronomical Society, 464(3), 3486–3497. https://doi.org/10.1093/mnras/stw2525 2

2