Lab #2 – Thermal Radiation & the Statistics of Noise

AST326 – November 14, 2018

Ayush Pandhi (ayush.pandhi@mail.utoronto.ca)

Group E: Ayush Pandhi, Hansen Jiang and Keslen Murdock

1 Abstract

Raw electric field data is gathered from an AirSpy device at various load temperatures is analyzed as both a temperature time-stream and a spectrum of power corresponding to different frequency bins. The electric field itself follows a normal distribution as predicted by the central limit theorem. The spectra are found to be quite noisy, especially at FM and LTE bands, and in general the power in the spectra increases with load temperature. A similar trend is found for temperature time-streams where in general the system temperature increases with load temperature. This follows the expected linear relationship between power (and system temperature) and the variance of the electric field. Additionally, the gain and receiver temperature are determined from a linear fit to the data and are found to be: p_0 = 0.10 and p_1 = 932.3, suggesting that the vast majority of signal is produced by the receiver rather than the load.

2 Introduction

Radio astronomy specifically deals with observations of celestial objects at low frequencies such as radio galaxies, quasars, pulsars and more. The origins of this young subfield in astronomy is rooted in the communications industry; beginning with Karl Jansky at Bell Telephone Laboratories who became the first person to detect radio waves from an astronomical source in 1932 when he observed radiation from the Milky Way. Over the past century, radio astronomy has gained popularity as it can abuse the radio atmospheric window to allow precise measurements from the Earth and as a result radio antennae across the globe have been conglomerated into large arrays to optimize the quality of observations. However signals at long wavelengths, particularly in the radio, cannot use the photoelectric effect for detection as individual photons do not have sufficient energy to photo-ionize materials. Therefore in radio astronomy, sufficiently strong signals are measured coherently by directly observing the oscillating electric fields within incident light. As such, coherent detectors are useful for analyzing electric fields, power and spectra of these observations at low frequencies.

The following experiment uses coherent observations of the electric field gathered from an AirSpy receiver placed in a number of different settings: room temperature, boiling water, ice, dry ice and liquid nitrogen. Theoretically, this data should provide a basis from which the temperature of the system and power can be determined through the relation that:

$$T \propto P \propto \langle E^2 \rangle \tag{1}$$

Additionally, Fourier Transforms provide an insight into the spectrum of power in relation to frequency bins through Fourier amplitudes of the data:

$$P(v) = \Re\{\tilde{\mathrm{E}}(\mathrm{v})\}^2 + \Im\{\tilde{\mathrm{E}}(\mathrm{v})\}^2 \tag{2}$$

Thus the data is analyzed as a spectrum and as temperature time-streams of the data sets to study this theorized relationship as well as the statistics by which noise is characterized in the system. The change in these distributions over the different settings is also discussed in detail.

3 Observation and Data

A commercially available AirSpy device was used to obtain the electric field data sets. This device has a tunable heterodyne receiver between 24-1800 MHz, which can record up to 10^7 samples per second. Additionally, it also contains: a tunable Local Oscillator, a mixer, an Intermediate Frequency filter, a Rafael Tuner chip with three gain adjustments, a Low Noise Amplifier for increasing the strength of the input signal, a mixer amplifier to boost the mixed signal, and the Intermediate Frequency amplifier which boosts the resulting signal in the desired frequency band. The AirSpy device is connected via USB to feed an input signal into an SMA connector and output raw ADC samples. The *airspy_rx* program controls various parameters for this data recording process, which are given in the instructions as:

```
-r <filename>: Receive data into file
-w Receive data into file with WAV header and automatic name
 This is for SDR# compatibility and may not work with other software
[-s serial_number_64bits]: Open device with specified 64bits serial
number
[-p packing]: Set packing for samples,
 1=enabled(12bits packed), 0=disabled(default 16bits not packed)
[-f frequency_MHz]: Set frequency in MHz between [24, 1900] (default
900MHz)
[-a sample_rate]: Set sample rate
[-t sample_type]: Set sample type,
 0=FLOAT32_IQ, 1=FLOAT32_REAL, 2=INT16_IQ(default), 3=INT16_REAL,
4=U16_REAL
[-b biast]: Set Bias Tee, 1=enabled, 0=disabled(default)
[-v vga_gain]: Set VGA/IF gain, 0-15 (default 5)
[-m mixer_gain]: Set Mixer gain, 0-15 (default 5)
[-l lna_gain]: Set LNA gain, 0-14 (default 1)
[-g linearity_gain]: Set linearity simplified gain, 0-21
[-h sensivity_gain]: Set sensitivity simplified gain, 0-21
[-n num_samples]: Number of samples to transfer (default is unlimited)
[-d]: Verbose mode
```

The data sets for this experiment recorded 100,000,000 samples of RAW data at 5 million samples per second with the Local Oscillator frequency set to 1GHz and all of the gains maxed out. The following table provides a log of the observations:

Date	Time [EST]	Setting	Duration [s]	Load Temp [^{o}C]
2018/11/09	11:05 am	Room Temp	20.0	87.2 ± 0.1
2018/11/09	11:13 am	Boiling Water	20.0	21.9 ± 0.1
2018/11/09	11:25 am	Ice	20.0	0.8 ± 0.1
2018/11/09	11:39 am	Dry Ice	20.0	-78.5 ± 0.1
2018/11/09	11:48 am	Liquid Nitrogen	20.0	-195.8 ± 0.1

Table 1: *AirSpy observation summary for various settings, uncertainty in load temp is the reading error for a thermometer.*

There is an initial drift in temperature due to the ADC warming up as it records samples; to counteract this issue, the majority of analysis was done on the second half (50 million samples) of the data sets, with the exception being the temperature time-streams which were done will the full data sets.

4 Data Reduction and Methods

A focal point of this experiment involves analyzing the power in the coherent detector as a time-stream. The power in this case is given by the equation:

$$P = \frac{\epsilon_o c E_o^2}{2} \propto \langle |E|^2 \rangle \tag{3}$$

Where ϵ_o and c are the permittivity of free space and speed of light constants respectively. E_o is the raw electric field measurement. Thus, it can be said that the power in a coherent detector is determined as the time-average of the squared magnitude of the electric field. Furthermore, measuring the spectrum of the data sets requires using a Fourier transform to divide the time-stream into components at different frequencies. The amplitude and phase of the sinusoid at a given frequency is determined as:

$$|\tilde{E}| = \sqrt{\Re\{\tilde{\mathrm{E}}(\mathrm{v})\}^2 + \Im\{\tilde{\mathrm{E}}(\mathrm{v})\}^2} \tag{4}$$

$$\tilde{E}_\phi = \operatorname{atan}\left(\frac{\Im\{\tilde{E}(v)\}}{\Re\{\tilde{E}(v)\}}\right) \tag{5}$$

Trivially, this also leads to the spectral power equation mentioned earlier (Equation 2). This spectral power describes all the incident power across a certain range of frequencies, or spectral band. The spectral resolution, or width of the band, is given by:

$$\Delta v = \frac{1}{2N\tau} \tag{6}$$

Where v is the frequency, N is the number of samples contained in the band and τ is the cadence. There is an upper limit to the frequencies for which the spectrum can be measured which is defined by the Nyquist Sampling Theorem. It follows that data must be sampled at twice the rate of the fastest varying sinusoid; signals at higher frequencies will create an aliasing effect in the spectra.

When dealing with radio frequencies, the flux within a given band is described by the Rayleigh-Jeans approximation:

$$\mathrm{B_v}(T) = \left(\frac{2v^2 k_B}{c^2}\right) T \tag{7}$$

Here k_B is the Boltzmann constant. Furthermore, the power per unit frequency or power across some bandwidth, Δv, can be described as:

$$\mathrm{P_v} = k_B T \tag{8}$$

$$P = k_B T \Delta v \tag{9}$$

It should be noted however that the total system temperature is the sum of multiple sources of signals and noise and can be loosely defined as:

$$T_{sys} = T_{CMB} + \Delta T_{source} + T_{atm} + T_{spill} + T_{RFI} + T_{receiver} + \cdots \tag{10}$$

Statistically speaking, the probability of measuring the electric field tends towards a normal distribution where $\langle E \rangle$ is zero because static electric fields do not persist for long in free space are filtered out in the receiver. Thus the relationship from earlier (Equation 1) can be expanded upon to relate the system temperature to the variance in the electric field:

$$T \propto var(E) \tag{11}$$

The uncertainty in the aforementioned system temperature is derived from the χ^2 distribution. The temperature measurement effectively measures the variance of the electric field using k samples (k describes the degrees of freedom in the system) and thus the uncertainty of the measurement is:

$$\sigma_T = \sqrt{var\left(\frac{T\chi_k^2}{k}\right)} = \sqrt{\left(\frac{T}{k}\right)^2 2k} = T\sqrt{2/k} \tag{12}$$

When analyzing time-streams however, the fractional uncertainty for N samples is described as:

$$\frac{\sigma_T}{T} = \sqrt{2/N} = 1/\sqrt{t\Delta v} \tag{13}$$

Where the final step used Equation 6 and gives the radiometer equation which holds for any normally distributed data, independent of amplifications, filters and other effects.

5 Data Analysis and Modeling

The detailed python code for generating all figures and data analysis in this section can be found in the Appendix. To start, the raw room temperature data was analyzed and the following results were obtained:

$$Mean = -5.8\,bits,\ \ Median = -6.0\,bits,\ \ Stddev = 30.5,\ \ Var = 931.4$$

It was observed that the ADC is not perfectly symmetric around zero and the data set was adjusted by subtracting the mean from each point to center it on zero. A small section (1024 points) were organized into a scatter plot (Figure 1), however it is difficult to see exactly what is going in with this visualization and as such the data was then represented as a histogram on both a regular and logarithmic y-axis (Figure 2).

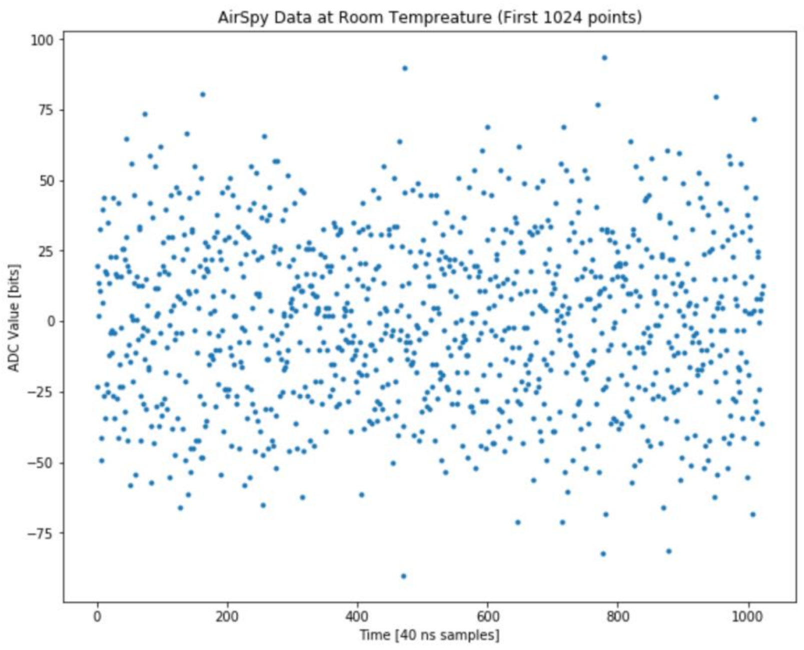

Figure 1: *Scatter plot of first 1024 points of the room temperature data set.*

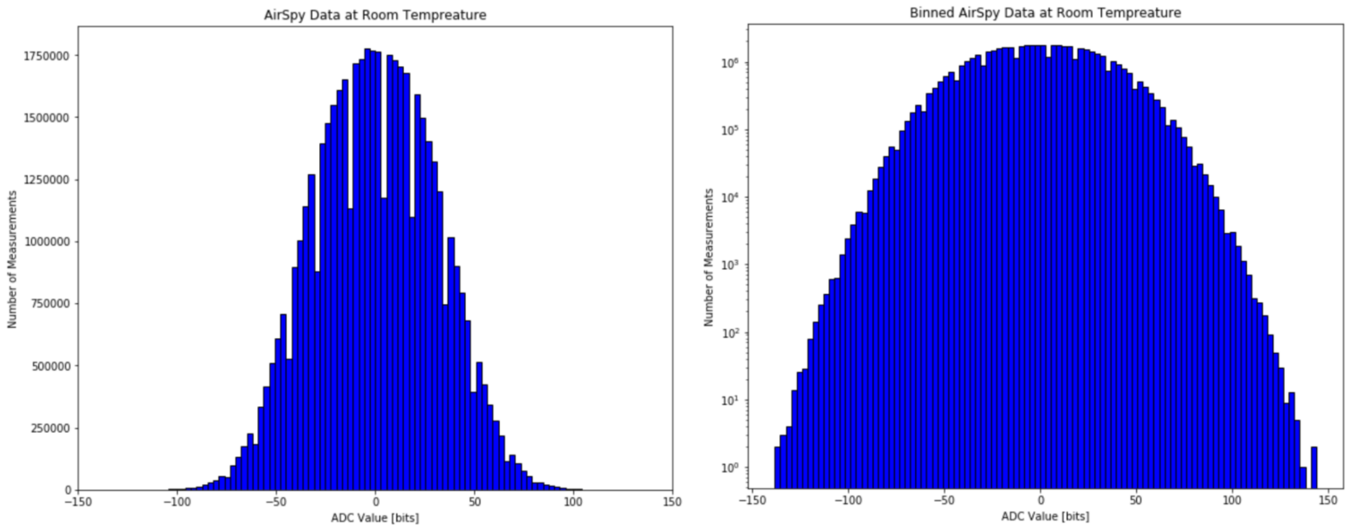

Figure 2: *a) (Left) Histogram of room temperature data (100 bins), (Right) Identical histogram as a) but on a log-y axis.*

The power for individual time samples can be computed based on Equation 3 and since this involves drawing individual samples from a normal distribution (Figure 2), its histogram should be similar to a χ^2 distribution function with one degree of freedom.

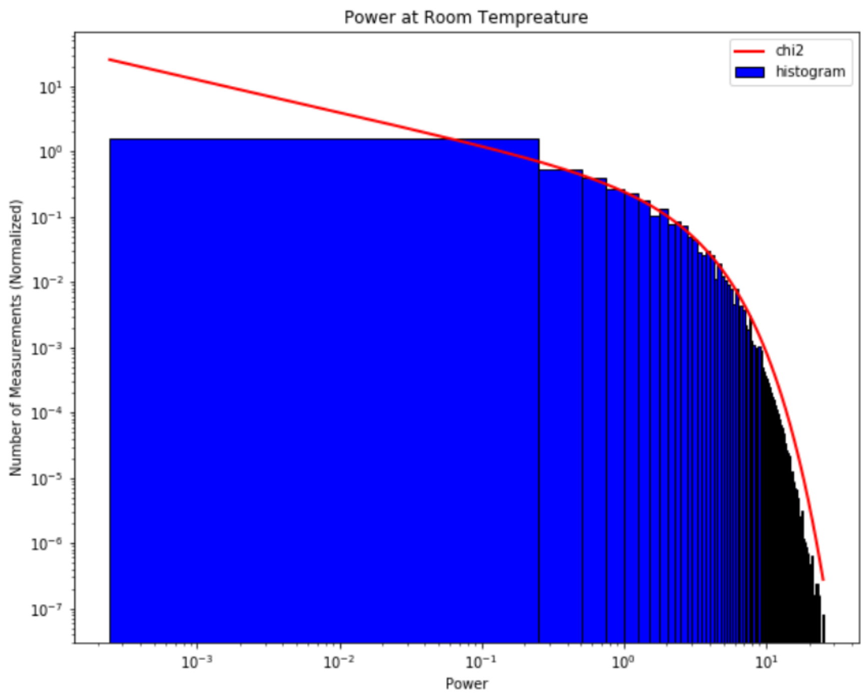

Figure 3: *Histogram of power at room temperature over-plotted with a χ^2 distribution with DoF=1 on log axes (100 bins, normalized to 1).*

A similar analysis is done by summing N adjacent samples rather than on individual samples for N = 2, 4, 10 and 100. Each distribution is also compared to a corresponding χ_N^2 function, where N represents the degrees of freedom in this case. They are plotted together to additionally see the progression of the distributions as N becomes large (Figure 4), however the plots of each N distribution alone can be found in the Appendix.

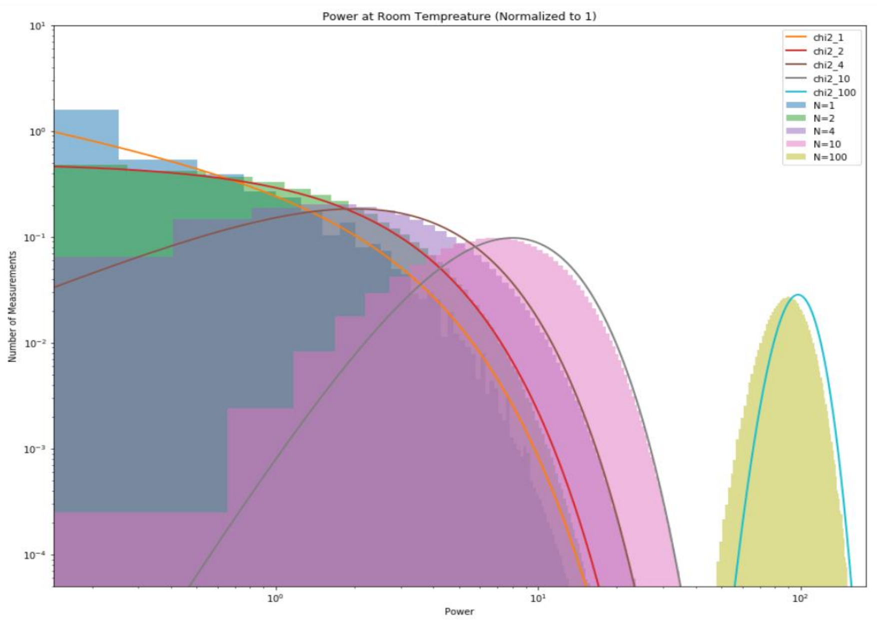

Figure 4: *Over-plot of all power histograms and corresponding χ^2 distributions for N = 2, 4, 10 and 100 on log axes (100 bins, normalized to 1).*

Moving forward, a realistic measurement of temperature as a time-stream is generated by averaging across a 1000 samples (Figure 5). The sudden jump near the beginning of the time-stream is thought to be caused by RFI and not thermal noise and as such should not be considered during analysis of the data. Here it is found that the mean and standard deviation are: 1002.1 ± 70.6 K. Additionally, since the sample rate for this experiment was set to 5 million samples per second, when averaging over 1000 samples, each point in the temperature time-stream corresponds to 2 x 10^{-4} seconds. Thus, from Equation 6, this implies that each point also corresponds to Δv = 2500 Hz.

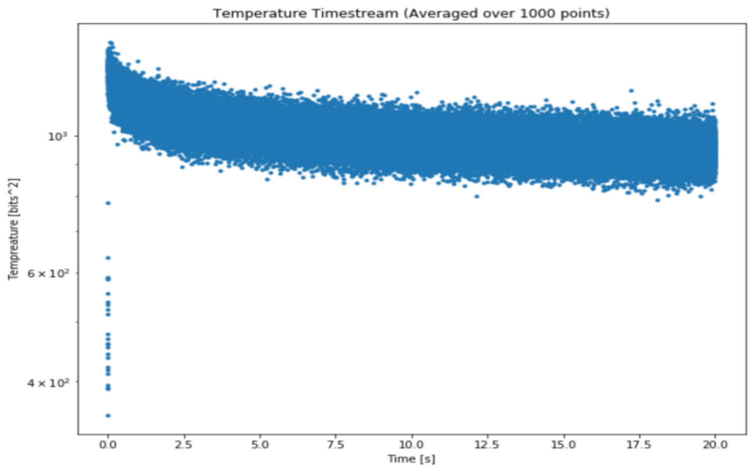

Figure 5: *Temperature (averaged over 1000 samples) of room temperature data plotted as a time-stream over 20.0 seconds.*

Next, spectral power is plotted (over 1024 samples) with corresponding frequency bins (Figure 6) which have been adjusted back to their original frequencies by re-adding the Local Oscillator frequency that was filtered initially. The plots are on a logarithmic dB (decibel) axis to more clearly see the details in the spectrum. Additionally, errors are computed for each point from the radiometer equation (Equation 13) and the over-plotted with the spectrum. The percent error in this case is 4.4%.

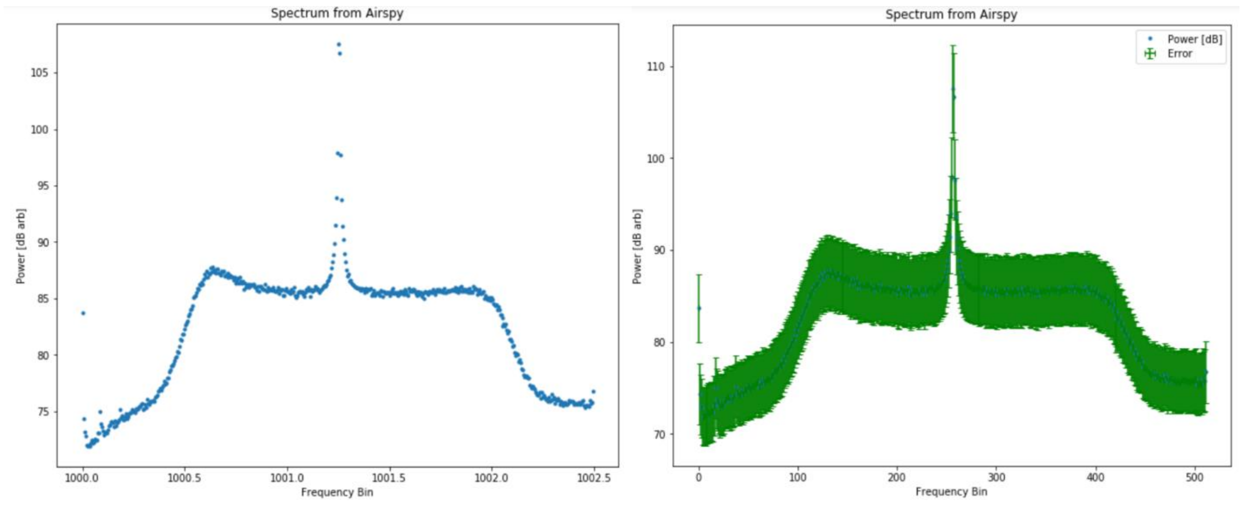

Figure 6: *a) (Left) Spectral Power (dB) versus Frequency over 1024 points, b) (Right) Identical plot as a) but with errors.*

The spectra from Figure 6 proves to be very noisy and to get a more precise measurement of the spectrum the spectra are instead averaged over 20,000 samples, which corresponds to a 1% uncertainty according to the radiometer equation. Using these conditions, a new spectrum is generated (Figure 7).

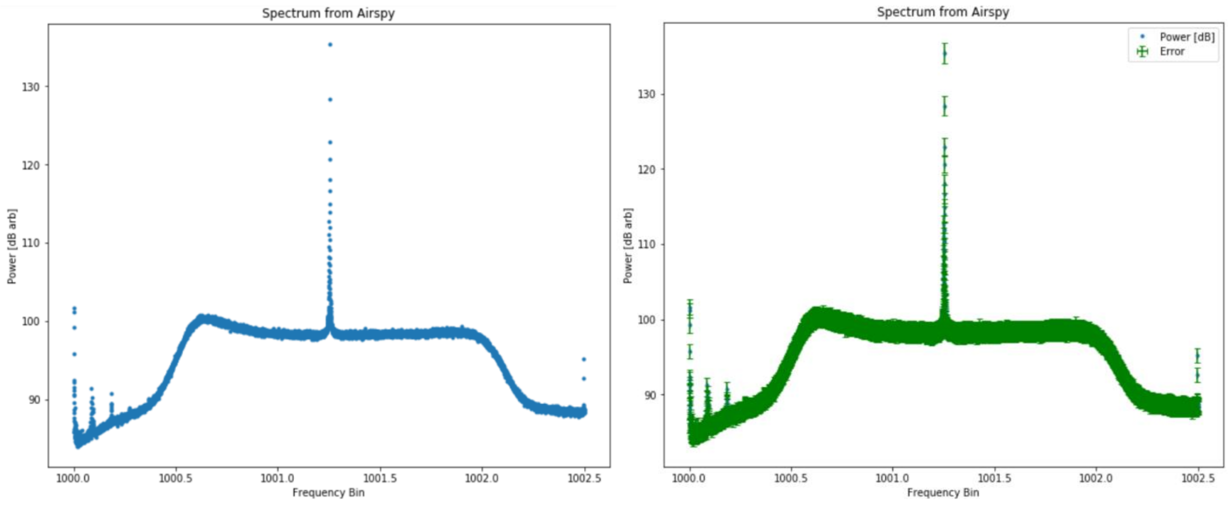

Figure 7: *a) (Left) Spectral Power (dB) versus Frequency over 20,000 points, b) (Right) Identical plot as a) but with errors.*

The AirSpy device was also re-tuned to FM radio (87.5-108 MHz) and LTE (720 MHz) bands to analyze spectral features in those ranges. The corresponding plots for these bands can be found in the Appendix and they are further discussed in section 6.

This same analysis for power time-streams and spectral power was then conducted for various physical temperatures of the load (see Table 1). The raw electric field histograms along with a table of numerical results (mean, standard deviation and variance) for all temperature conditions can be found in the Appendix. From the resulting power time-stream plot and spectra it can be determined that the dry ice data set acts as an outlier; this is discussed further in section 6.

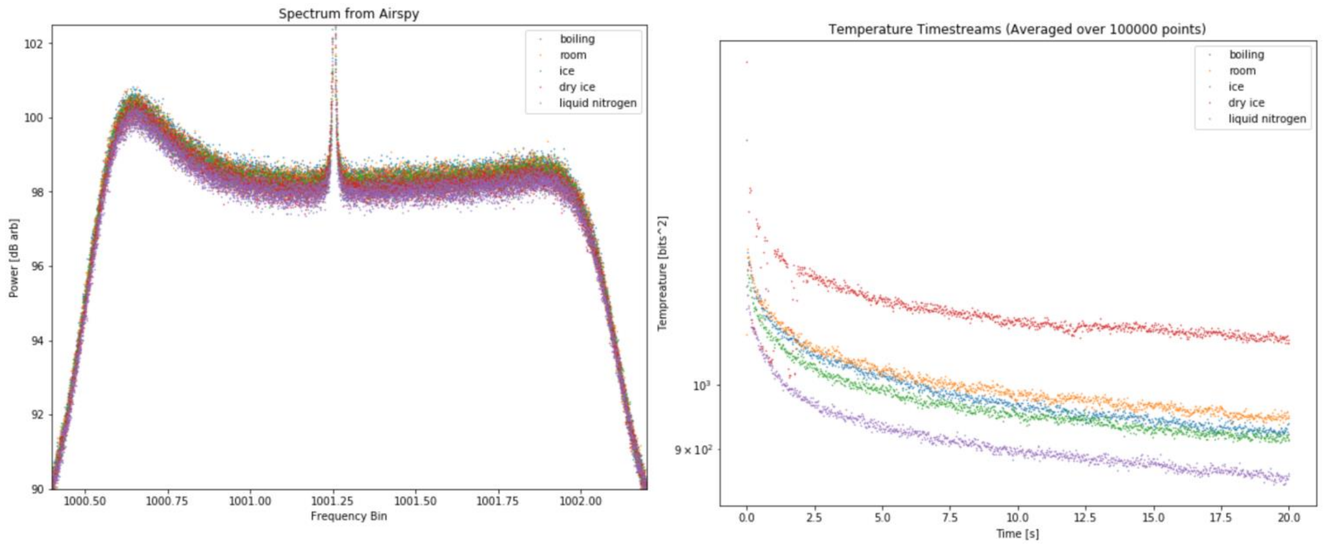

Figure 8: *a) (Left) Spectral power for all data sets (dB), b) (Right) Temperature time-streams for all data sets on log-y axis.*

Next, a relationship between variance of the electric field (which linearly relates to system temperature) is plotted with a linear fit to determine the slope and y-intercept parameters. The uncertainty is derived from the radiometer equation. It should be noted that, again, the dry ice data seems to be a heavy outlier and while liquid nitrogen does not exactly match the linear regression, it is reasonably close to being linear. This could be due to the fact that measurement uncertainties have not been considered in the linear fit. The estimated parameters from this fit are: p_0 = 0.10 (gain) and p_1 = 932.3 (offset/receiver temperature) for the linear relationship, $T_{sys} \propto var(E) = p_0 T_{load} + p_1$.

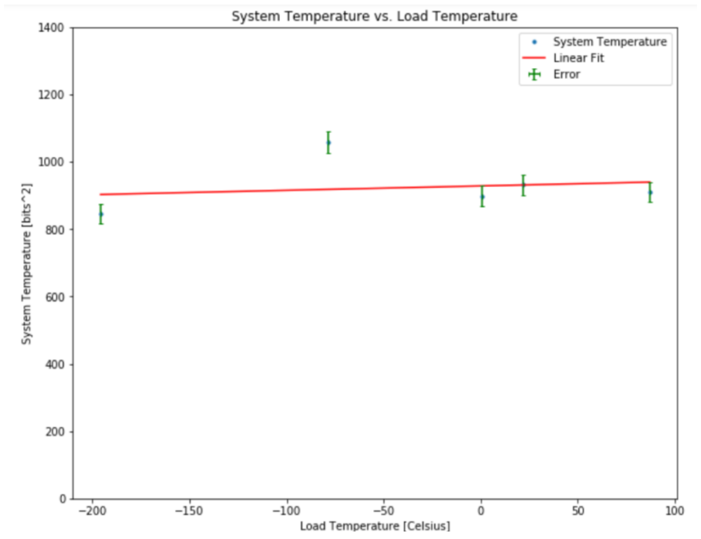

Figure 9: *System Temperature versus Load Temperature over-plotted with a linear fit to the data.*

Similarly, the gain and offset can be plotted as a function of frequency for the given data sets (Figure 10). They are additionally plotted on logarithmic axes to see more detail in their evolution across the band. The uncertainty here also follows the radiometer equation.

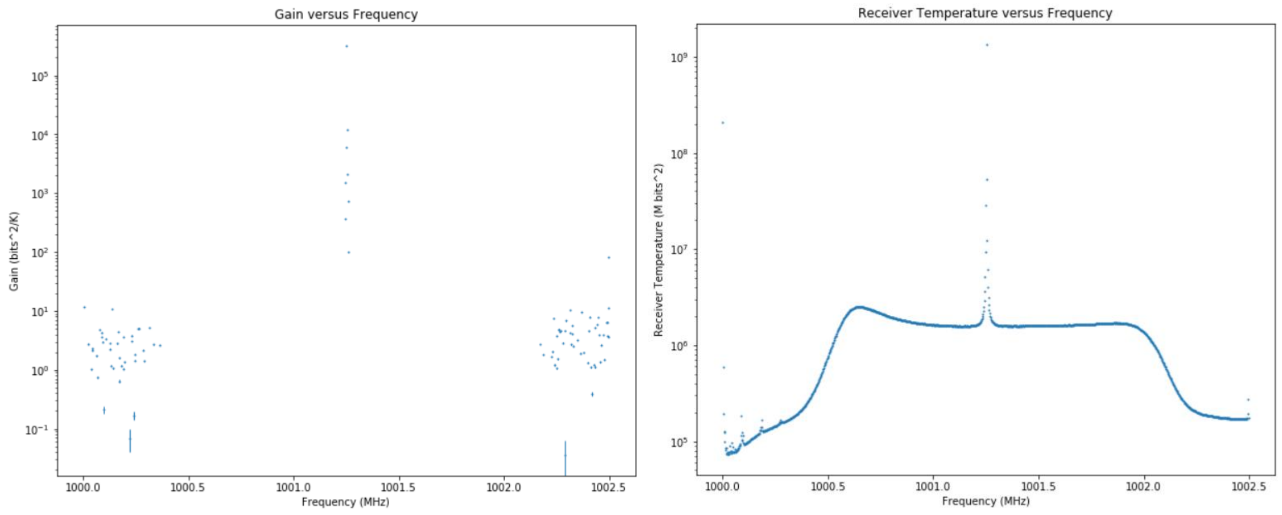

Figure 10: *a) (Left) Gain plotted against frequency, b) (Right) Receiver Temperature plotted against Frequency.*

6 Discussion

In Figure 2 the raw electric field at room temperature is binned into a histogram and, as expected by the central limit theorem, the distribution resembles a single-peaked Gaussian function. Thus it can be concluded that the electric field samples follow a normal distribution for a large number of samples. Furthermore, when plotting the power distribution, which scales as the average of the squared electric field, for various values of N, it is found that the distributions follow the corresponding χ_N^2 functions with N degrees of freedom. In addition, for larger values of N, the distributions being to resemble a Gaussian which is an expected result.

Using this idea of summing over N adjacent samples, Figure 5 is depicts the temperature (averaged over 1000 samples) as a time-stream over 20.0 seconds. There is clearly a sudden jump at the beginning which is characterized as RFI and not an actual feature of the observations. The numerical results were: 1002.1 ± 70.6 K (mean T and σ_T), with each point corresponding to 2 x 10^{-4} seconds and thus Δv = 2500 Hz. In addition it is found that computing $\frac{\sigma_T}{T}$ directly using the stated mean T and σ_T gives approximately 0.007 (0.7%). Using the radiometer directly implies the expected value is 0.045 (4.5%). Thus it is concluded that the measured value is fairly similar to the expected.

Figure 6 and 7 depict the spectral power at room temperature for 4.4% (N = 1024) and 1.0% (N = 20,000) errors respectively. From visual analysis it can be determined that the low and high-pass filters are roughly 1000.5 Hz and 1002.25 Hz respectively. Additionally, there is significant RFI between 1000-1000.25 Hz, in the center at 1001.25 Hz and a small amount towards the end at 10002.5 Hz. It should also be noted that there is a slight drop-off in power when moving from the low-pass to high-pass filter. The edge of the bandpass appears to be roughly 10 dB fainter than the center, thus its power about 10 times less than in the center of the band.

When the AirSpy is re-tuned to either the FM or LTE bands, the spectra becomes a lot more chaotic. The FM spectra is fairly similar to the pervious with the exception that the RFI in the center is across a much wider range and there is another very significant RFI spike just past 1002.0 Hz. For LTE, it is hard to discern any features as most of the band seems to be RFI, although on average it seems to keep the general shape of the first spectra. This increase in RFI is expected as there are a lot of common signals in these bands (radios and phones).

Generally, from Figure 8 it is clear that as load temperature increases, the power (and temperature) goes up as well. This is an expected outcome of Equation 1; the variance in the electric field decreases as the load temperature is decreased and thus it will decrease power. Therefore, the results agree with the theory for the most part. However, there is a clear outlier in the dry ice data set which can be seen as it depicts the highest temperature; this data set was deemed to be an anomaly, likely as a result of faulty data collection, and thus it is ignored when discussing the over-arching trends between the data sets.

In Figure 9, a linear fit is applied to the relationship between system temperature and load temperature; the gain (slope) and offset (y-intercept) are estimated from this to be p_0 = 0.10 and p_1 = 932.3. This suggests that the vast majority of signal (approximately 90% or more) is actually coming from the receiver rather than the load. A large source of systemic error that could affect this is that the system may not have reached equilibrium when the measurements were taken, thus giving less ideal data than expected.

9 Appendix

9.1 Individual Power Histograms for N = 2, 4, 10 and 100

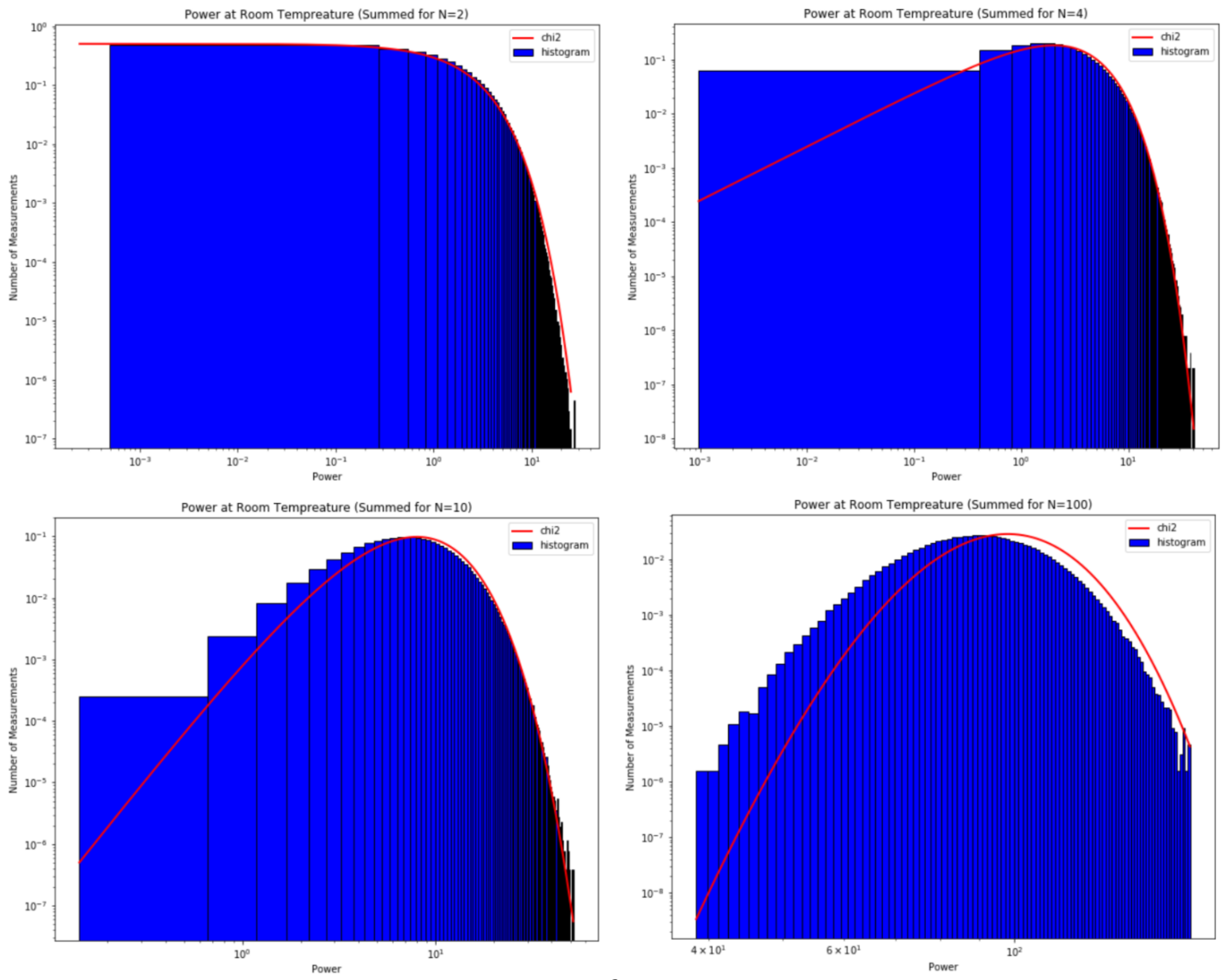

Figure 11: *Individual power histograms and corresponding χ^2 distributions for N = 2, 4, 10 and 100 on log axes (100 bins, normalized to 1).*

9.2 Spectral Power Plots for FM and LTE Bands

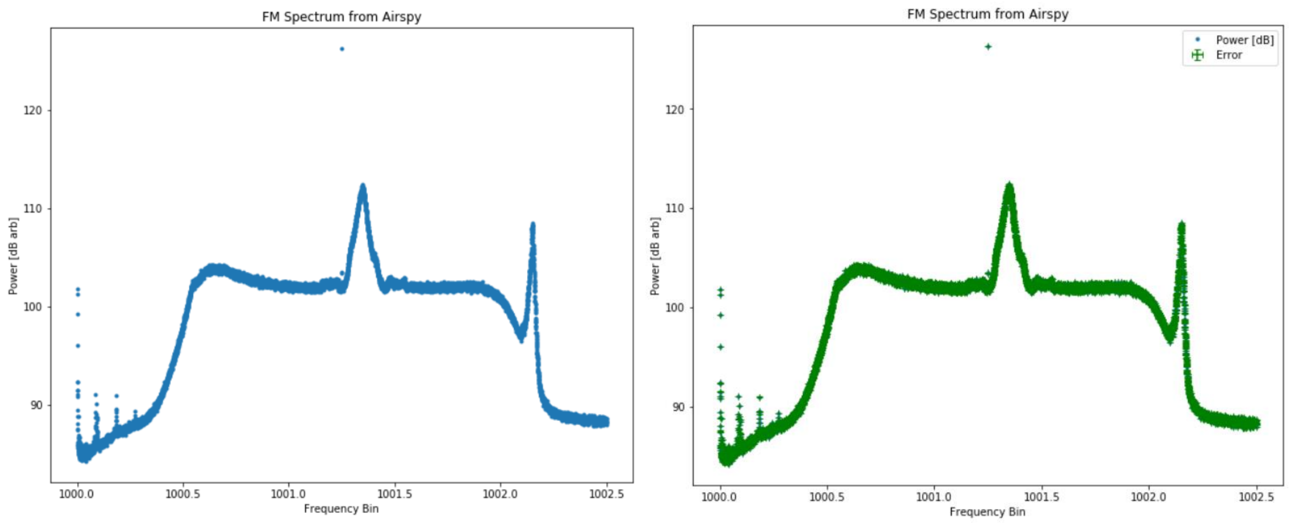

Figure 12: *a) (Left) Spectral Power at FM band (dB) versus Frequency over 20,000 points, b) (Right) Identical plot as a) but with errors.*

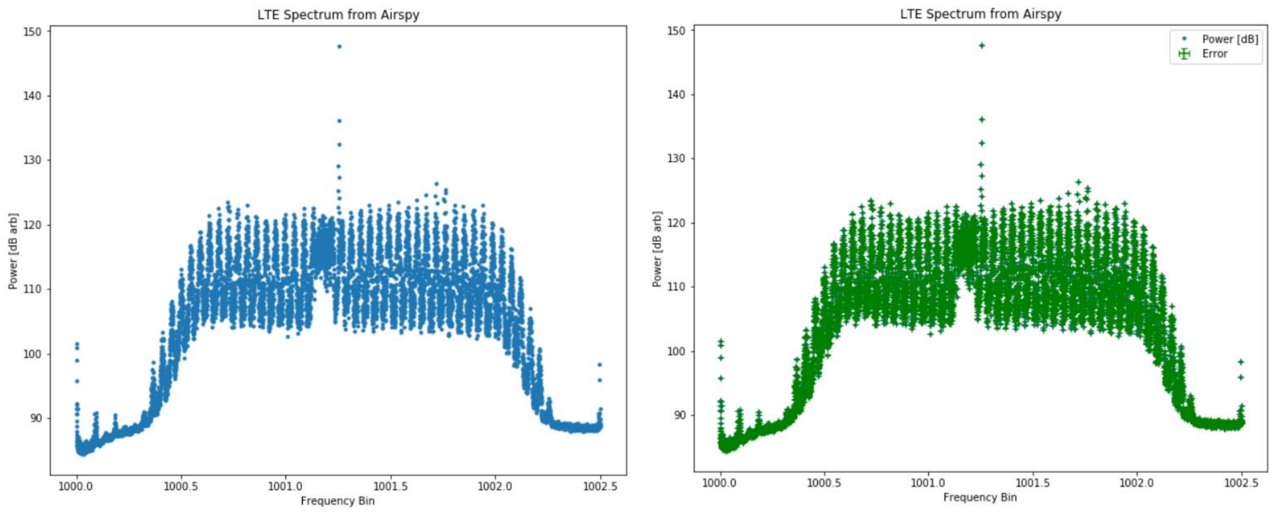

Figure 13: *a) (Left) Spectral Power at LTE band (dB) versus Frequency over 20,000 points, b) (Right) Identical plot as a) but with errors.*

9.3 Raw Electric Field Histograms for Various Load Temperatures

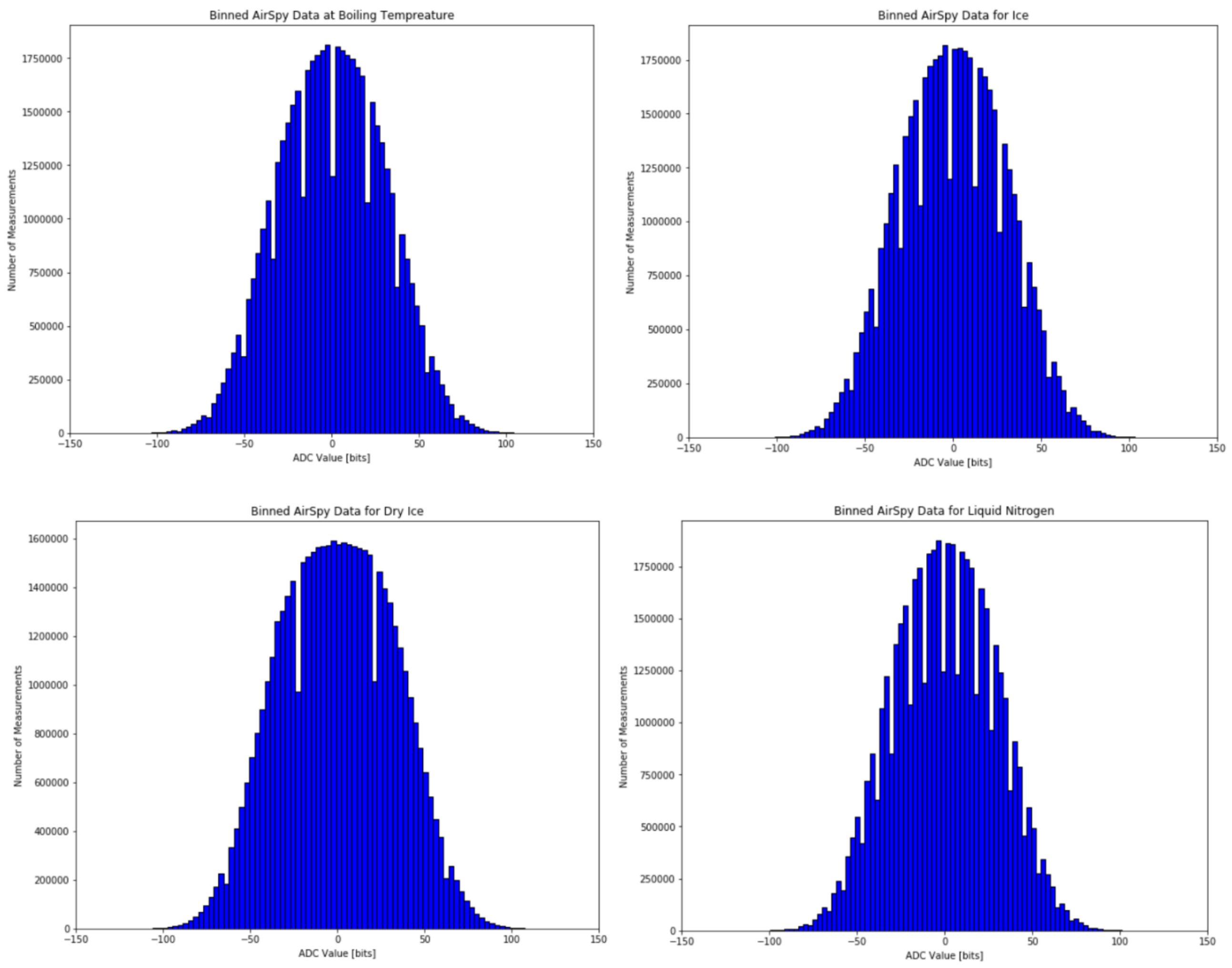

Figure 14: *Histogram of various load temperature data sets (100 bins).*

9.4 Table of Statistical Results for Various Load Temperatures

Load Temp [^{o}C]	Mean	Standard Deviation	Variance
87.2 ± 0.1	-5.8	30.2	910.6
21.9 ± 0.1	-5.8	30.5	931.3
0.8 ± 0.1	-5.7	30.0	899.1
-78.5 ± 0.1	-5.8	32.5	1058.7
-195.8 ± 0.1	-5.7	29.1	845.9

Table 2: *Computed mean, standard deviation and variance for each of the various load temperatures*

9.5 Python Code for Figures 1 and 2

```
#Importing required modules
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import norm
from scipy.stats import chi2

#Loading data from all different sources
room_data_full = np.fromfile('room_UHF_100m.dat',dtype='int16')-2.**11

#Removing the first 50M data points which are warm up time
room_data = room_data_full[50000000:]

#Basic analysis on data
room_mean = np.mean(room_data)
room_median = np.median(room_data)
room_std = np.std(room_data)
room_var = np.var(room_data)
print('room_data properties: ', room_mean, room_median, room_std, room_var)


#Adjusting data to be centered around 0

room_data_fix = room_data - room_mean


#Same analysis on adjusted data

room_fix_mean = np.mean(room_data_fix)

room_fix_median = np.median(room_data_fix)

room_fix_std = np.std(room_data_fix)

room_fix_var = np.var(room_data_fix)

print(room_fix_mean, room_fix_median, room_fix_std, room_fix_var)


#Scatter plot of a small section of 1000 points

plt.figure(figsize = (10, 8))

plt.plot(room_data_fix[:1024], '.')

plt.title('AirSpy Data at Room Tempreature (First 1024 points)')
plt.xlabel('Time [40 ns samples]')

plt.ylabel('ADC Value [bits]')

plt.show()


#Plotting the data as a histogram

plt.figure(figsize = (10, 8))

plt.hist(room_data_fix, bins = 100, color='b', edgecolor='k')

plt.title('AirSpy Data at Room Tempreature')

plt.xlabel('ADC Value [bits]')

plt.ylabel('Number of Measurements')

plt.xlim(-150, 150)

plt.show()


#Same plot with a log-y axis

plt.figure(figsize = (10, 8))

plt.hist(room_data_fix, bins = 100, color='b', edgecolor='k')

plt.title('Binned AirSpy Data at Room Tempreature')

plt.xlabel('ADC Value [bits]')

plt.ylabel('Number of Measurements')

plt.yscale('log')

plt.show()
```

9.6 Python Code for Figures 3, 4 and 11

```
#Computing power per time measurement
power_room = (1/2)*((8.85418782*(10**(-12)))*(3*(10**8))*(room_data_fix**2))

#Setting up chi2
xlin = np.linspace(min(power_room), max(power_room), 1000)
chi2_room = chi2.pdf(xlin, 1)

#Plotting the data as a histogram
plt.figure(figsize = (10, 8))
plt.hist(power_room, bins = 100, color='b', edgecolor='k', label='histogram')
plt.title('Power at Room Tempreature')
plt.xlabel('Power')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.xscale('log')
plt.show()

#Overplot with chi2 to show it fits well
plt.figure(figsize = (10, 8))
plt.hist(power_room, bins = 100, color='b', edgecolor='k', normed=1, label='histogram')
plt.plot(xlin, chi2_room, 'r-', linewidth=2, label='chi2')
plt.title('Power at Room Tempreature')
plt.xlabel('Power')
plt.ylabel('Number of Measurements (Normalized)')
plt.yscale('log')
plt.xscale('log')
plt.show()

#Defining a function to sum N adjacent samples
def sum_adj_samples(N, data):
    new_data = np.empty(int(data.shape[0]/N))
    for i in range(int(data.shape[0]/N)):
        new_data[i] = np.sum(data[i*N:(i*N)+N])
    return new_data

#Summing 2 adjacent samples
power_room2 = sum_adj_samples(2, power_room)

#Setting up chi2
xlin2 = np.linspace(min(power_room2), max(power_room2), 1000)
chi2_room2 = chi2.pdf(xlin2, 2)

#Overplot with chi2 to show it fits well
plt.figure(figsize = (10, 8))
plt.hist(power_room2, bins = 100, color='b', normed=1, edgecolor='k', label='histogram')
plt.plot(xlin, chi2_room2, 'r-', linewidth=2, label='chi2')
plt.title('Power at Room Tempreature (Summed for N=2)')
plt.xlabel('Power')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.xscale('log')
plt.show()

#Summing 4 adjacent samples
power_room4 = sum_adj_samples(4, power_room)

#Setting up chi2
xlin4 = np.linspace(min(power_room4), max(power_room4), 1000)
chi2_room4 = chi2.pdf(xlin4, 4)

#Overplot with chi2 to show it fits well
plt.figure(figsize = (10, 8))
plt.hist(power_room4, bins = 100, color='b', normed=1, edgecolor='k', label='histogram')
plt.plot(xlin4, chi2_room4, 'r-', linewidth=2, label='chi2')
plt.title('Power at Room Tempreature (Summed for N=4)')
plt.xlabel('Power')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.xscale('log')
plt.show()

#Summing 10 adjacent samples
power_room10 = sum_adj_samples(10, power_room)

#Setting up chi2
xlin10 = np.linspace(min(power_room10), max(power_room10), 1000)
chi2_room10 = chi2.pdf(xlin10, 10)

#Overplot with chi2 to show it fits well
plt.figure(figsize = (10, 8))
plt.hist(power_room10, bins = 100, color='b', normed=1, edgecolor='k', label='histogram')
plt.plot(xlin10, chi2_room10, 'r-', linewidth=2, label='chi2')
plt.title('Power at Room Tempreature (Summed for N=10)')
plt.xlabel('Power')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.xscale('log')
plt.show()

#Summing 100 adjacent samples
power_room100 = sum_adj_samples(100, power_room)

#Setting up chi2
xlin100 = np.linspace(min(power_room100), max(power_room100), 1000)
chi2_room100 = chi2.pdf(xlin100, 100)

#Overplot with chi2 to show it fits well
plt.figure(figsize = (10, 8))
plt.hist(power_room100, bins = 100, color='b', normed=1, edgecolor='k', label='histogram')
plt.plot(xlin100, chi2_room100, 'r-', linewidth=2, label='chi2')
plt.title('Power at Room Tempreature (Summed for N=100)')
plt.xlabel('Power')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.xscale('log')
plt.show()

#Overplot with chi2 to show it fits well
plt.figure(figsize = (15, 12))
plt.hist(power_room, bins = 100, normed=1, label='N=1', alpha=0.5)
plt.plot(xlin, chi2_room, linewidth=2, label='chi2_1')
plt.hist(power_room2, bins = 100, normed=1, label='N=2', alpha=0.5)
plt.plot(xlin, chi2_room2, linewidth=2, label='chi2_2')
plt.hist(power_room4, bins = 100, normed=1, label='N=4', alpha=0.5)
plt.plot(xlin4, chi2_room4, linewidth=2, label='chi2_4')
plt.hist(power_room10, bins = 100, normed=1, label='N=10', alpha=0.5)
plt.plot(xlin10, chi2_room10, linewidth=2, label='chi2_10')
plt.hist(power_room100, bins = 100, normed=1, label='N=100', alpha=0.5)
plt.plot(xlin100, chi2_room100, linewidth=2, label='chi2_100')
plt.title('Power at Room Tempreature')
plt.xlabel('Power')
plt.ylabel('Number of Measurements')
plt.xlim(min(power_room10),)
plt.ylim(0.00005, 10)
plt.yscale('log')
plt.xscale('log')
plt.legend()
plt.show()
```

9.7 Python Code for Figures 5, 6, 7, 12 and 13

```
#Getting tempreature as average over 1000 samples of E-filed squared
tlin = np.linspace(0, 20, 100000)
temp_room1000 = (sum_adj_samples(1000, room_data_full**2))/1000

#Scatter plot of a small section of 1000 points
plt.figure(figsize = (10, 8))
plt.plot(tlin, temp_room1000, '.')
plt.title('Temperature Timestream (Averaged over 1000 points)')
plt.xlabel('Time [s]')
plt.ylabel('Tempreature [bits^2]')
plt.yscale('log')
plt.show()

#Fourier transform on the room data
f = np.fft.fft(room_data_fix[0:2**19].reshape(-1, 1024), axis=1)
s = (f.real**2 + f.imag**2).sum(axis=0)
freq = 1000 + np.arange(0, 2.5, 2.5/512) #Adding the LO frequency back in to calibrate the frequency
axis

#Plotting Spectrum from Airspy
plt.figure(figsize=(10,8))
plt.plot(freq, s[0:512], '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power')
plt.title('Spectrum from Airspy')
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq, 10*np.log10(s[0:512]), '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('Spectrum from Airspy')
plt.show()

#Uncertainity using Radiometer equation
N = 1024
sigma = (2/N)**0.5
print('Percent Error: ', sigma*100, '%')

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(10*np.log10(s[0:512]), '.', label='Power [dB]')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('Spectrum from Airspy')
plt.errorbar(np.arange(0, 512), 10*np.log10(s[0:512]), xerr=0, yerr=sigma*10*np.log10(s[0:512]),
linestyle='none', ecolor='g', label='Error', capsize=3)
plt.legend()
plt.show()

#Need average over 20 000 samples to get 1% uncertainty
#Fourier transform on the room data
f2 = np.fft.fft(room_data_fix[0:10240000].reshape(-1, 20000), axis=1)
```

```
s2 = (f2.real**2 + f2.imag**2).sum(axis=0)

freq2 = 1000 + np.arange(0, 2.5, 2.5/10000) #Adding the LO frequency back in to calibrate the
frequency axis


#Uncertainity using Radiometer equation

N2 = 20000

sigma2 = (2/N2)**0.5

print('Percent Error: ', sigma2*100, '%')


#Plotting Spectrum from Airspy in dB

plt.figure(figsize=(10,8))

plt.plot(freq2, 10*np.log10(s2[0:10000]), '.')

plt.xlabel('Frequency Bin')

plt.ylabel('Power [dB arb]')

plt.title('Spectrum from Airspy')

plt.show()


#Plotting Spectrum from Airspy in dB

plt.figure(figsize=(10,8))

plt.plot(freq2, 10*np.log10(s2[0:10000]), '.', label='Power [dB]')

plt.xlabel('Frequency Bin')

plt.ylabel('Power [dB arb]')

plt.title('Spectrum from Airspy')

plt.errorbar(freq2, 10*np.log10(s2[0:10000]), xerr=0, yerr=sigma2*10*np.log10(s2[0:10000]),
linestyle='none', ecolor='g', label='Error', capsize=3)

plt.legend()

plt.show()


#Same analysis for FM room data

roomFM_data_full = np.fromfile('room_FM_100m.dat',dtype='int16')-2.**11
```

```
roomFM_data = roomFM_data_full[50000000:]

#Basic analysis on data
roomFM_mean = np.mean(roomFM_data)
roomFM_median = np.median(roomFM_data)
roomFM_std = np.std(roomFM_data)
roomFM_var = np.var(roomFM_data)
print('roomFM_data properties: ', roomFM_mean, roomFM_median, roomFM_std, roomFM_var)

#Adjusting data to be centered around 0
roomFM_data_fix = roomFM_data - roomFM_mean

#Same analysis on adjusted data
roomFM_fix_mean = np.mean(roomFM_data_fix)
roomFM_fix_median = np.median(roomFM_data_fix)
roomFM_fix_std = np.std(roomFM_data_fix)
roomFM_fix_var = np.var(roomFM_data_fix)
print(roomFM_fix_mean, roomFM_fix_median, roomFM_fix_std, roomFM_fix_var)

#Computing power per time measurement
powerFM_room = (1/2)*((8.85418782*(10**(-12)))*(3*(10**8))*(roomFM_data_fix**2))

#Need average over 20 000 samples to get 1% uncertainty
#Fourier transform on the room data
fFM = np.fft.fft(roomFM_data_fix[0:2**19].reshape(-1, 1024), axis=1)
sFM = (fFM.real**2 + fFM.imag**2).sum(axis=0)
fFM2 = np.fft.fft(roomFM_data_fix[0:10240000].reshape(-1, 20000), axis=1)
sFM2 = (fFM2.real**2 + fFM2.imag**2).sum(axis=0)
```

```
#Plotting Spectrum from Airspy
plt.figure(figsize=(10,8))
plt.plot(freq, sFM[0:512], '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power')
plt.title('FM Spectrum from Airspy')
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq, 10*np.log10(sFM[0:512]), '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('FM Spectrum from Airspy')
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq2, 10*np.log10(sFM2[0:10000]), '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('FM Spectrum from Airspy')
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq2, 10*np.log10(sFM2[0:10000]), '.', label='Power [dB]')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('FM Spectrum from Airspy')
plt.errorbar(freq2, 10*np.log10(sFM2[0:10000]), xerr=0, yerr=sigma2, linestyle='none', ecolor='g',
label='Error', capsize=3)
plt.legend()
plt.show()


#Same analysis for FM room data
roomLTE_data_full = np.fromfile('room_LTE_100m.dat',dtype='int16')-2.**11
roomLTE_data = roomLTE_data_full[50000000:]


#Basic analysis on data
roomLTE_mean = np.mean(roomLTE_data)
roomLTE_median = np.median(roomLTE_data)
roomLTE_std = np.std(roomLTE_data)
roomLTE_var = np.var(roomLTE_data)
print('roomLTE_data properties: ', roomLTE_mean, roomLTE_median, roomLTE_std, roomLTE_var)


#Adjusting data to be centered around 0
roomLTE_data_fix = roomLTE_data - roomLTE_mean


#Same analysis on adjusted data
roomLTE_fix_mean = np.mean(roomLTE_data_fix)
roomLTE_fix_median = np.median(roomLTE_data_fix)
roomLTE_fix_std = np.std(roomLTE_data_fix)
roomLTE_fix_var = np.var(roomLTE_data_fix)
print(roomLTE_fix_mean, roomLTE_fix_median, roomLTE_fix_std, roomLTE_fix_var)


#Computing power per time measurement
powerLTE_room = (1/2)*((8.85418782*(10**(-12)))*(3*(10**8))*(roomLTE_data_fix**2))
```

```
#Need average over 20 000 samples to get 1% uncertainty
#Fourier transform on the room data
fLTE = np.fft.fft(roomLTE_data_fix[0:2**19].reshape(-1, 1024), axis=1)
sLTE = (fLTE.real**2 + fLTE.imag**2).sum(axis=0)
fLTE2 = np.fft.fft(roomLTE_data_fix[0:10240000].reshape(-1, 20000), axis=1)
sLTE2 = (fLTE2.real**2 + fLTE2.imag**2).sum(axis=0)

#Plotting Spectrum from Airspy
plt.figure(figsize=(10,8))
plt.plot(freq, sLTE[0:512], '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power')
plt.title('LTE Spectrum from Airspy')
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq, 10*np.log10(sLTE[0:512]), '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('LTE Spectrum from Airspy')
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq2, 10*np.log10(sLTE2[0:10000]), '.')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('LTE Spectrum from Airspy')
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq2, 10*np.log10(sLTE2[0:10000]), '.', label='Power [dB]')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('LTE Spectrum from Airspy')
plt.errorbar(freq2, 10*np.log10(sLTE2[0:10000]), xerr=0, yerr=sigma2, linestyle='none', ecolor='g',
label='Error', capsize=3)
plt.legend()
plt.show()
```

9.8 Python Code for Figures 8, 9, 10 and 14

```
#Loading boiling data
boiling_data_full = np.fromfile('boiling.dat',dtype='int16')-2.**11
boiling_data = boiling_data_full[50000000:]

#Basic analysis on data
boiling_mean = np.mean(boiling_data)
boiling_median = np.median(boiling_data)
boiling_std = np.std(boiling_data)
boiling_var = np.var(boiling_data)
print('boiling_data properties: ', boiling_mean, boiling_median, boiling_std, boiling_var)

#Adjusting data to be centered around 0
boiling_data_fix = boiling_data - boiling_mean

#Same analysis on adjusted data
```

```
boiling_fix_mean = np.mean(boiling_data_fix)

boiling_fix_median = np.median(boiling_data_fix)

boiling_fix_std = np.std(boiling_data_fix)

boiling_fix_var = np.var(boiling_data_fix)

print(boiling_fix_mean, boiling_fix_median, boiling_fix_std, boiling_fix_var)


#Plotting the data as a histogram

plt.figure(figsize = (10, 8))

plt.hist(boiling_data_fix, bins = 100, color='b', edgecolor='k')

plt.title('Binned AirSpy Data at Boiling Tempreature')

plt.xlabel('ADC Value [bits]')

plt.ylabel('Number of Measurements')

plt.xlim(-150, 150)

plt.show()


#Same plot with a log-y axis

plt.figure(figsize = (10, 8))

plt.hist(boiling_data_fix, bins = 100, color='b', edgecolor='k')

plt.title('Binned AirSpy Data at Boiling Tempreature')

plt.xlabel('ADC Value [bits]')

plt.ylabel('Number of Measurements')

plt.yscale('log')

plt.show()


#Loading ice data

ice_data_full = np.fromfile('ice.dat',dtype='int16')-2.**11

ice_data = ice_data_full[50000000:]


#Basic analysis on data
```

```
ice_mean = np.mean(ice_data)
ice_median = np.median(ice_data)
ice_std = np.std(ice_data)
ice_var = np.var(ice_data)
print('ice_data properties: ', ice_mean, ice_median, ice_std, ice_var)

#Adjusting data to be centered around 0
ice_data_fix = ice_data - ice_mean

#Same analysis on adjusted data
ice_fix_mean = np.mean(ice_data_fix)
ice_fix_median = np.median(ice_data_fix)
ice_fix_std = np.std(ice_data_fix)
ice_fix_var = np.var(ice_data_fix)
print(ice_fix_mean, ice_fix_median, ice_fix_std, ice_fix_var)

#Plotting the data as a histogram
plt.figure(figsize = (10, 8))
plt.hist(ice_data_fix, bins = 100, color='b', edgecolor='k')
plt.title('Binned AirSpy Data for Ice')
plt.xlabel('ADC Value [bits]')
plt.ylabel('Number of Measurements')
plt.xlim(-150, 150)
plt.show()

#Same plot with a log-y axis
plt.figure(figsize = (10, 8))
plt.hist(ice_data_fix, bins = 100, color='b', edgecolor='k')
plt.title('Binned AirSpy Data for Ice')
plt.xlabel('ADC Value [bits]')

plt.ylabel('Number of Measurements')

plt.yscale('log')

plt.show()


#Loading dry ice data

dryice_data_full = np.fromfile('dry_ice.dat',dtype='int16')-2.**11

dryice_data = dryice_data_full[50000000:]


#Basic analysis on data

dryice_mean = np.mean(dryice_data)

dryice_median = np.median(dryice_data)

dryice_std = np.std(dryice_data)

dryice_var = np.var(dryice_data)

print('dryice_data properties: ', dryice_mean, dryice_median, dryice_std, dryice_var)


#Adjusting data to be centered around 0

dryice_data_fix = dryice_data - dryice_mean


#Same analysis on adjusted data

dryice_fix_mean = np.mean(dryice_data_fix)

dryice_fix_median = np.median(dryice_data_fix)

dryice_fix_std = np.std(dryice_data_fix)

dryice_fix_var = np.var(dryice_data_fix)

print(dryice_fix_mean, dryice_fix_median, dryice_fix_std, dryice_fix_var)


#Plotting the data as a histogram

plt.figure(figsize = (10, 8))

plt.hist(dryice_data_fix, bins = 100, color='b', edgecolor='k')
plt.title('Binned AirSpy Data for Dry Ice')
plt.xlabel('ADC Value [bits]')
plt.ylabel('Number of Measurements')
plt.xlim(-150, 150)
plt.show()

#Same plot with a log-y axis
plt.figure(figsize = (10, 8))
plt.hist(dryice_data_fix, bins = 100, color='b', edgecolor='k')
plt.title('Binned AirSpy Data for Dry Ice')
plt.xlabel('ADC Value [bits]')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.show()

#Loading liquid nitrogen data
liqnit_data_full = np.fromfile('liquid_nitrogen.dat',dtype='int16')-2.**11
liqnit_data = liqnit_data_full[50000000:]

#Basic analysis on data
liqnit_mean = np.mean(liqnit_data)
liqnit_median = np.median(liqnit_data)
liqnit_std = np.std(liqnit_data)
liqnit_var = np.var(liqnit_data)
print('liqnit_data properties: ', liqnit_mean, liqnit_median, liqnit_std, liqnit_var)

#Adjusting data to be centered around 0
liqnit_data_fix = liqnit_data - liqnit_mean
```

```
#Same analysis on adjusted data
liqnit_fix_mean = np.mean(liqnit_data_fix)
liqnit_fix_median = np.median(liqnit_data_fix)
liqnit_fix_std = np.std(liqnit_data_fix)
liqnit_fix_var = np.var(liqnit_data_fix)
print(liqnit_fix_mean, liqnit_fix_median, liqnit_fix_std, liqnit_fix_var)

#Plotting the data as a histogram
plt.figure(figsize = (10, 8))
plt.hist(liqnit_data_fix, bins = 100, color='b', edgecolor='k')
plt.title('Binned AirSpy Data for Liquid Nitrogen')
plt.xlabel('ADC Value [bits]')
plt.ylabel('Number of Measurements')
plt.xlim(-150, 150)
plt.show()

#Same plot with a log-y axis
plt.figure(figsize = (10, 8))
plt.hist(liqnit_data_fix, bins = 100, color='b', edgecolor='k')
plt.title('Binned AirSpy Data for Liquid Nitrogen')
plt.xlabel('ADC Value [bits]')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.show()

#Plotting the data as a histogram
plt.figure(figsize = (10, 8))
plt.hist(boiling_data_fix, bins = 100, label='Boiling')
plt.hist(room_data_fix, bins = 100, label='Room Temp')
plt.hist(ice_data_fix, bins = 100, label='Ice')
plt.hist(dryice_data_fix, bins = 100, label='Dry Ice')
plt.hist(liqnit_data_fix, bins = 100, label='Liquid Nitrogen')
plt.title('Binned AirSpy Data')
plt.xlabel('ADC Value [bits]')
plt.ylabel('Number of Measurements')
plt.xlim(-150, 150)
plt.legend()
plt.show()

#Same plot with a log-y axis
plt.figure(figsize = (10, 8))
plt.hist(boiling_data_fix, bins = 100, label='Boiling')
plt.hist(room_data_fix, bins = 100, label='Room Temp')
plt.hist(ice_data_fix, bins = 100, label='Ice')
plt.hist(dryice_data_fix, bins = 100, label='Dry Ice')
plt.hist(liqnit_data_fix, bins = 100, label='Liquid Nitrogen')
plt.title('Binned AirSpy Data')
plt.xlabel('ADC Value [bits]')
plt.ylabel('Number of Measurements')
plt.yscale('log')
plt.legend()
plt.show()

#Plotting Spectrum from Airspy in dB
plt.figure(figsize=(10,8))
plt.plot(freq2, 10*np.log10(s2[0:10000]), '.', markersize=1, label='boiling')
plt.plot(freq2, 10*np.log10(s[0:10000]), '.', markersize=1, label='room')
plt.plot(freq2, 10*np.log10(s3[0:10000]), '.', markersize=1, label='ice')
plt.plot(freq2, 10*np.log10(s4[0:10000]), '.', markersize=1, label='dry ice')
plt.plot(freq2, 10*np.log10(s5[0:10000]), '.', markersize=1, label='liquid nitrogen')
plt.xlabel('Frequency Bin')
plt.ylabel('Power [dB arb]')
plt.title('Spectrum from Airspy')
plt.ylim(90, 102.5)
plt.xlim(1000.4, 1002.2)
plt.legend()
plt.show()

#Getting tempreature as average over 1000 samples of E-filed squared
tlin = np.linspace(0, 20, 1000)
temp_room100000 = (sum_adj_samples(100000, room_data_full**2))/100000
temp_boiling100000 = (sum_adj_samples(100000, boiling_data_full**2))/100000
temp_ice100000 = (sum_adj_samples(100000, ice_data_full**2))/100000
temp_dryice100000 = (sum_adj_samples(100000, dryice_data_full**2))/100000
temp_liqnit100000 = (sum_adj_samples(100000, liqnit_data_full**2))/100000

#Scatter plot of a small section of 1000 points
plt.figure(figsize = (10, 8))
plt.plot(tlin, temp_boiling100000, '.', markersize=1, label='boiling')
plt.plot(tlin, temp_room100000, '.', markersize=1, label='room')
plt.plot(tlin, temp_ice100000, '.', markersize=1, label='ice')
plt.plot(tlin, temp_dryice100000, '.', markersize=1, label='dry ice')
plt.plot(tlin, temp_liqnit100000, '.', markersize=1, label='liquid nitrogen')
plt.title('Temperature Timestreams (Averaged over 100000 points)')
plt.xlabel('Time [s]')
plt.ylabel('Tempreature [bits^2]')
plt.yscale('log')
plt.legend()
plt.show()


def f(x, a, b):
    return a*x + b


T_load = np.array((87.2, 21.9, 0.8, -78.5, -195.8))
T_load_error = np.array((0.1, 0.1, 0.1, 0.1, 0.1))
T_sys = np.array((910.5787401751303, 931.3771725763115, 899.0944174949105,
1058.7341928277358, 845.8891468787122))
T_sys_error = np.array((30.1757972583183, 30.518472644880372, 29.984903159672044,
32.538195906161356, 29.08417347766156))


#Define functions for initial conditions
#Function for gain
def p0(x,y,n):
    p0 = ((n*np.sum(x*y)) - (np.sum(x)*np.sum(y)))/((n*np.sum(x**2)) - ((np.sum(x))**2))
    return p0


#Function for offset
def p1(x,y,n,p):
    p1 = (1/n)*(np.sum(y) - p*np.sum(x))
    return p1


#Calculate gain and offset for initial plot
p0_1 = p0(T_load, T_sys, 5)
p1_1 = p1(T_load, T_sys, 5, p0_1)
print(p0_1, p1_1)


#Function for averaging over 20 000 spectra
```

```
def avg(x):
    data_reshape = x.reshape(20000,-1) #Reshape the data into 20 000 chunks
    avg_fft = np.fft.fft(data_reshape,axis=1) #Run the FFT
    avg_s = (avg_fft.real**2 + avg_fft.imag**2) #Calculate the power using equation 5
    avg_s_mean = np.mean(avg_s, axis=0) #Average over the chunks
    return avg_s_mean

#All averaged spectra
rt_avg = avg(room_data)
ice_avg = avg(ice_data)
boiling_avg = avg(boiling_data)
dryice_avg = avg(dryice_data)
liqnit_avg = avg(liqnit_data)

#Create a new array made up of all of the averaged spectra
all_variances = np.vstack((liqnit_avg, dryice_avg, ice_avg, rt_avg, boiling_avg))

#Calculate all values of p0 and p1 and put them into arrays
p0_array = np.zeros(2500)
p1_array = np.zeros(2500)
for i in range(0, 2500):
    p0_array[i] = p0(T_load, all_variances[:,i], 5)
    p1_array[i] = p1(T_load, all_variances[:,i], 5, p0_array[i])

#Plots of gain and offset on logarithmic axes
#Plot of gain on logarithmic axes
plt.figure(figsize=(10,8))
plt.errorbar(freq, p0_array[:1250], xerr=None, yerr=error_p, ls='', marker='o',markersize=1, lw=1)
plt.title('Gain versus Frequency')
plt.ylabel('Gain (bits^2/K)')
plt.xlabel('Frequency (MHz)')
plt.yscale('log')
plt.show()
#Plot of offset on logarithmic axes
plt.figure(figsize=(10,8))
plt.errorbar(freq, p1_array[:1250], xerr=None, yerr=error_p, ls='', marker='o', markersize=1, lw=1)
plt.title('Receiver Temperature versus Frequency')
plt.ylabel('Receiver Temperature (M bits^2)')
plt.xlabel('Frequency (MHz)')
plt.yscale('log')
plt.show()
```