AST325 Lab 2: Thermal Radiation and Statistics of Noise

Shimona Das
Student Number: 1008964943
Group N

November 2024

Abstract

This experiment investigates the characteristics of thermal noise and the calibration of a radio receiver for the interpretation of low-frequency radio signals. By using an AirSpy receiver, measurements were taken across various load temperatures to understand how thermal noise affects system response and how to convert digital signals into physical temperature units. The experiment revealed that the raw electric field data follows a Gaussian distribution, confirming thermal noise behaviour according to the Central Limit Theorem, while power data exhibit a chi-squared distribution. Calibration results showed a linear relationship between system temperature and load temperature, with approximately 4.8% of the system temperature attributable to the thermal load and 95.2% to the receiver. The receiver's internal noise level was then estimated from the y-intercept of this calibration fit. The calculated gain, dark current, and read noise values provide insight into the receiver's performance, confirming its applicability in scenarios where noise reduction and accurate signal interpretation are essential. This study emphasises the importance of noise correction techniques and also aims to establish a reliable method for radio signal calibration in radio astronomy and other fields requiring precise thermal noise analysis.

1 Introduction

Radio astronomy involves detecting and analyzing low-frequency electromagnetic waves from celestial sources and is essential for studying phenomena invisible in other parts of the electromagnetic spectrum. This lab explores the statistical behavior of thermal noise and calibrates the AirSpy receiver, a commercially available radio device, to relate digital measurements to physical temperature units.

Thermal noise or Johnson noise, is generated by the random motion of electrons in resistive materials and manifests as random variations in electric fields. When passed through the AirSpy receiver, this noise signal is then amplified, filtered, and digitized by an Analogue-to-Digital Converter (ADC) which yields a time-ordered dataset, also called a time stream. According to the Central Limit Theorem (CLT), the sum of many independent random variables tends to a Gaussian (normal) distribution. Thus, thermal noise data typically follows a Gaussian distribution, allowing us to predict its behaviour using statistical principles.

Gaussian distribution is defined by its mean and standard deviation. In radio measurements, these properties make Gaussian distributed thermal noise ideal for further statistical analysis, as the Gaussian shape remains consistent over time and is predictable. Beyond raw electric field values, the power of the signal is proportional to the square of the electric field. Squaring a Gaussian distributed variable yields a chi-squared distribution, which is an asymmetric distribution skewed towards positive values. The degrees of freedom (DoF) of this distribution depend on the sample size. With higher DoF, the distribution approaches a Gaussian shape which allow radio astronomers to analyse power statistics as an indicator of noise properties in different spectral bins.

To analyse noise across frequency, a Fourier Transform is applied to the time stream data, converting it from the time domain to the frequency domain. This transform produces a power spectrum which reveals the distribution of signal power over various frequencies. Visualizing the power spectrum in decibel (dB) scale enables easier analysis of fluctuations across a wide range of intensities, helping identify noise patterns and any potential interference in the signal.

A key aspect of this experiment is the calibration of the receiver to convert digital signal measurements to physical temperature units. This calibration is done by varying the temperature of a terminator (a resistive component) and recording the system's response. The radiometer equation guides this calibration by predicting that the uncertainty in temperature decreases with the square root of observation

time and bandwidth. This relationship allows the determination of the number of measurements needed to achieve a 1 percent uncertainty in temperature readings, thereby refining the system's accuracy for interpreting radio signals in real-world units.

This report begins with an outline of the experimental setup and the data acquisition process with the AirSpy receiver. Subsequent sections cover data processing techniques including Fourier transformations and power spectrum generation, followed by a detailed statistical analysis of Gaussian and chi-squared properties in the data. The final section addresses calibration, employing the radiometer equation to quantify the receiver's response to thermal noise, linking digital readings to physical temperatures.

2 Data and Observation

Date	Personnel	Notes
08/10/2024	S. Das, M. Pye, J. Hora	Recorded and inspected test data set
15/10/2024	S. Das, M. Pye, J. Hora	Varied temperature of the load and recorded data
22/10/2024	S. Das, M. Pye, J. Hora	Analysed spectral properties

Table 1: A summary of observations

Specification	**AirSpy R2**
Frequency Range	24 - 1800 MHz (partial coverage)
ADC Resolution	12-bit
Sampling Rate	Up to 10 MSPS (Mega Samples Per Second)
Bandwidth	Up to 10 MHz
Sensitivity	-130 dBm
Connector	SMA Female
Power Supply	USB-powered (5V, 500 mA)
Dimensions	45 x 68 x 19 mm
Operating System Compatibility	Windows, Linux, Android

Table 2: Properties of AirSpy R2

Temperature (°C)	**Mean**	**Median**	**Standard Deviation**	**Variance**
-196	-3.4041	-3.0	31.6523	1001.87
3.7	-3.4688	-3.0	34.2044	1169.94
Room Temp (25°C)	-3.4074	-3.0	34.3010	1176.56
54.5	-3.1558	-3.0	33.2098	1102.89
72.6	-3.2952	-3.0	34.2582	1173.63
90	-3.3973	-3.0	34.7592	1208.20

Table 3: Summary of ADC Sample Statistics at Different Temperatures

The experiment began with initial data collection using the AirSpy SDR device across several frequencies. High levels of Radio Frequency Interference (RFI) affected these first readings, so multiple frequencies were tested. It was found that 609 Hz produced the clearest results with the least noise, and this frequency was selected for the remainder of the experiment. Python was then used to record and calculate the variance, mean, and standard deviation of the data. To explore spectral features, the AirSpy was tuned to specific frequencies corresponding to radio stations including 88 MHz , 99 MHz, and 720 MHz, and data was collected for each. Temperature dependent measurements were also conducted. At room temperature (24.5°C), a 10 second reading was taken as the temperature gradually increased to 24.9°C. For cold temperature readings, the AirSpy reader and thermometer were enclosed in a rubber glove and then submerged in ice water, initially reading 3.9°C.

Due to questionable data quality, this reading was repeated, excluding the first 20–30 seconds to minimize interference. Due to the volume of data and the constant crashing of the computer, the sample time was reduced to 20 seconds. Further measurements were conducted with warm water. The

Figure 1: AirSpy Software-Defined Radio (SDR) device connected to a laptop through USB for data collection. This setup allows for capturing and analysing radio frequency signals, with the AirSpy serving as the receiver and the laptop handling data processing and storage. The configuration allows measuring signal power and frequency characteristics across a specified bandwidth.

initial temperature of the water was 90.0°C, decreasing slightly to 89.8°C. Additional readings were taken at 72.6°C, 53.9°C, and with liquid nitrogen. For each dataset, initial segments were excluded, and statistical analysis was performed using Python. This approach ensured reliable calibration across different temperatures and minimized RFI for accurate data acquisition.

3 Data Reduction

The data reduction process began with an initial evaluation of three preliminary datasets collected across different frequencies. Due to high levels of RFI, these datasets were unsuitable for further analysis. After testing multiple frequencies, 609 Hz was identified as the optimal frequency, providing the clearest signal with minimal noise. Within this 20 second sample, the first half of the data was found to contain excessive noise and interference so only the second half was retained for analysis. This step reduced self interference and provided a cleaner, more stable signal for processing. Statistical analyses, including mean, variance, and standard deviation, were performed on this refined dataset using Python. For temperature dependent measurements, data was collected across various thermal settings and outliers within these measurements were identified and excluded when necessary. This was done by applying a Gaussian filter which weights nearby points with a Gaussian shape and gives more importance to points closer to the centre of the window.

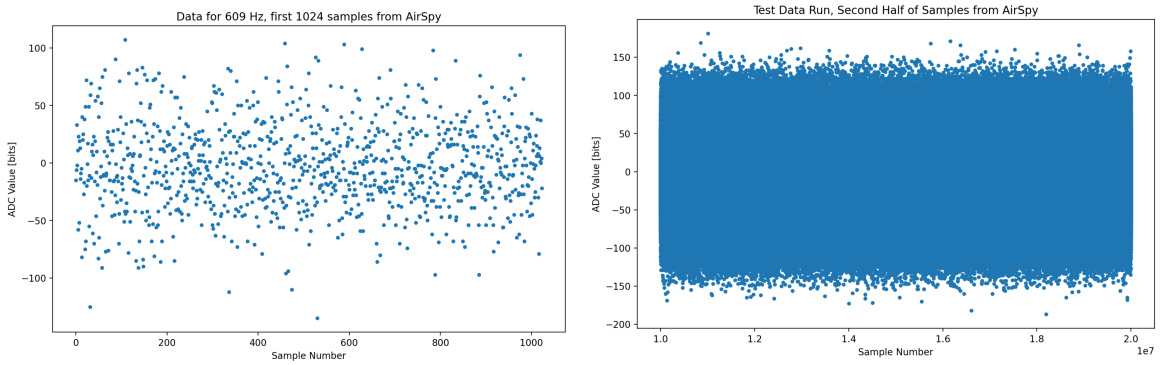
A spectral analysis was then performed by tuning the AirSpy to different frequencies (88 MHz, 99 MHz, and 720 MHz) to capture distinct spectral features. Each spectrum was processed using Fast Fourier Transform (FFT) on chunks of 1024 samples, converting the data to the frequency domain. By averaging the spectral data across segments, it was possible to reduce noise and ensure consistent signal representation. To assess temperature effects, the radiometer equation (derived in Appendix A) was used to calculate the uncertainty associated with each temperature measurement as shown in Equation 1. σ_T represents the uncertainty in the system temperature measurement, T is the system temperature, t denotes the integration time over which the signal is measured, and $\Delta\nu$ represents the bandwidth of the measurement in Hz. This allowed for the calculation of the system temperature, T_{sys}, as a function of thermal load.

$$\frac{\sigma_T}{T} = \frac{1}{\sqrt{t\Delta\nu}} \tag{1}$$

Finally, a linear fit was applied to the T_{sys} vs. T_{load} data to explore the relationship between system

temperature and load temperature. This provided insights into the thermal noise from both the receiver and the thermal load.

4 Data Analysis

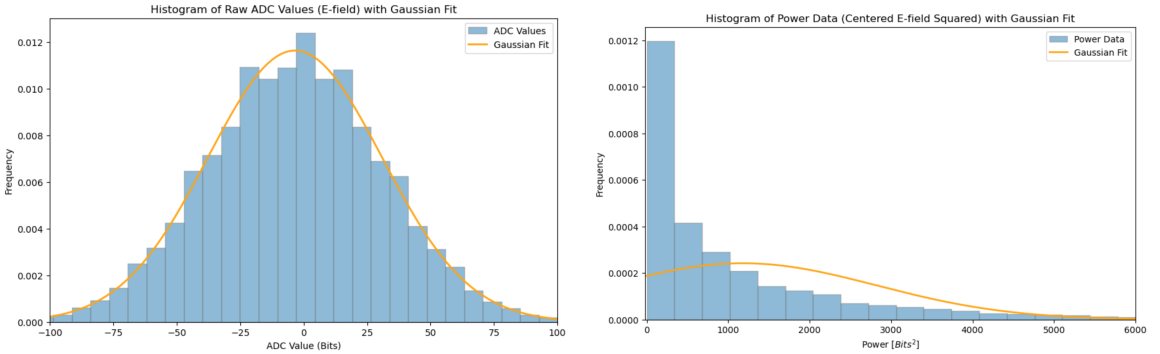


(a) Data for 609 Hz of the first 1024 samples from AirSpy, showing initial signal distribution.

(b) Data for 609 Hz, second half of the collected samples from AirSpy.

Figure 2: Comparison of ADC values for initial samples and a larger dataset at 609 Hz from AirSpy. Both graphs show the distribution of ADC values (in bits) over different sample sizes. (b) has the statistical properties: Mean = -3.41, Median = -3.0, Standard Deviation = 34.30, Variance = 1176.56 bits^2. This plot includes a larger dataset, providing a clearer view of the ADC signal's distribution across a longer time period.

Analysing Figure 2a, which displays the initial 1024 samples, it is noticeable that the data points appear relatively constrained rather than evenly distributed. This suggests that the signal may be limited, either due to external signal constraints or internal ADC limitations. The data does not span the entire ADC's allowable bit range. For accurate analysis when calculating power, we should first subtract this mean offset from the data to centre it around zero which eliminates any bias and improves the accuracy calculations. The statistics for the Figure 2b graph, reveal a mean of approximately -3.41, a median of -3.0, a standard deviation of 34.30, and a variance of 1176.56 (in bits^2). The mean and median values slightly below zero suggest that the ADC is not perfectly symmetric around zero, as an ideally centered ADC output would have a mean value of zero. This small negative offset can indicate a systematic bias in the ADC or minor interference that shifts the average value of the signal. Such a bias means that when calculating power, this offset should ideally be removed to ensure accuracy in measurements.



(a) Histogram of Raw ADC Values (E-field) with Gaussian Fit. The data displays a bell-shaped Gaussian distribution centered around zero, consistent with expected thermal noise.

(b) Histogram of Power Data (Centered E-field Squared) with Gaussian Fit. This plot shows the power distribution in $Bits^2$, revealing a right-skewed distribution characteristic of squared Gaussian noise.

Figure 3: Comparison of histograms for raw ADC values and centered power data, each with overlaid Gaussian fits. (a) shows the Gaussian distribution of raw ADC values, while (b) represents the corresponding power distribution after centering around zero.

The histogram in Figure 3a shows a single-peaked, bell-shaped Gaussian distribution which is centred around zero, typical for a normal distribution. This shape suggests that the raw ADC values predominantly consist of thermal noise, which, due to the central limit theorem, tends to follow a Gaussian distribution especially over a large number of samples. This single peak with symmetrical decay on either side is a strong indication of randomness in the signal, showing domination by thermal noise rather than any significant periodic or structured signal. In Figure 3b, the histogram of the power data displays a right-skewed, single-peaked distribution. This skewness is characteristic of squared Gaussian noise, where the power values are derived from the square of a normally distributed signal. The peak near zero further indicates that most of the power values are small, corresponding to the lower amplitude fluctuations in the original ADC signal, while occasional higher power values account for the longer tail. These distributions suggest that the signals being measured are primarily random thermal fluctuations with no significant non-random or structured components. The single peaked nature of both histograms supports the conclusion that no additional signals dominate the measured data, and the ADC is mainly picking up background noise.

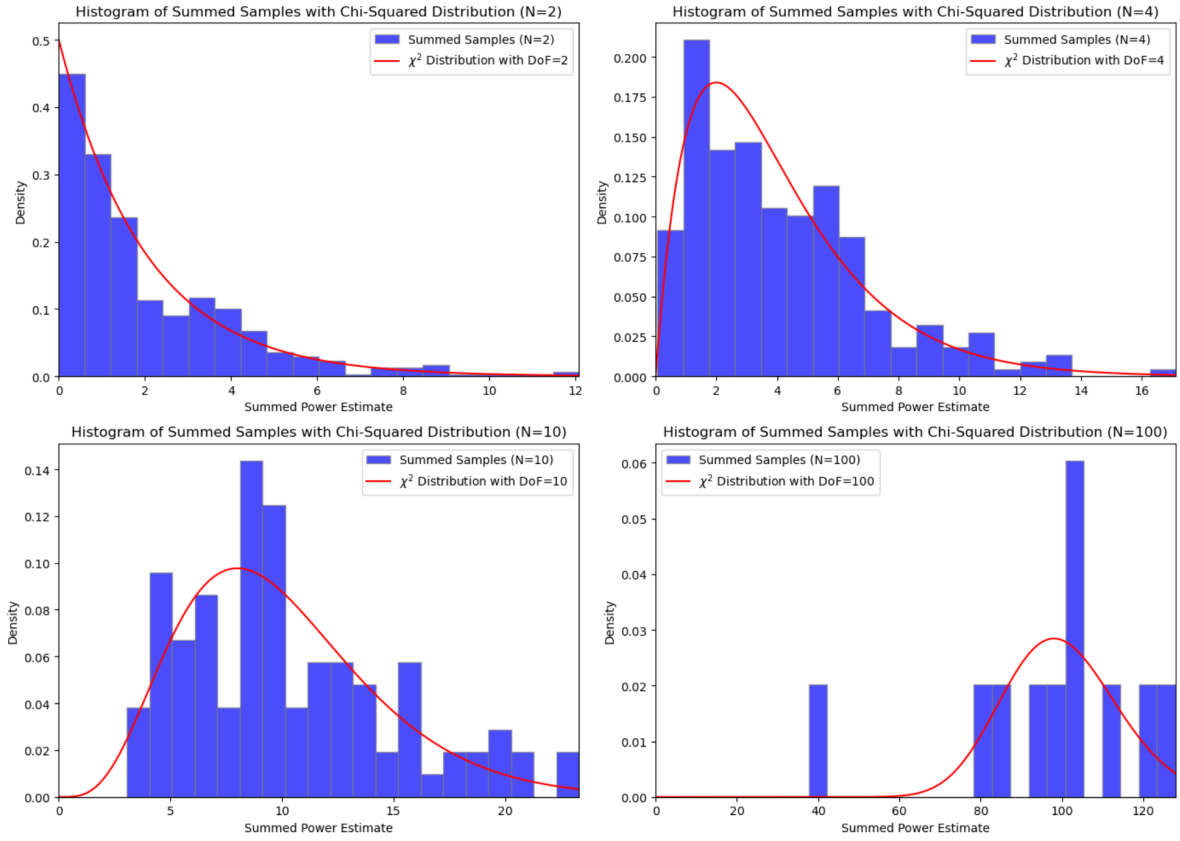


Figure 4: Histograms of summed power estimates with overlaid chi-squared distributions for different summation window sizes N. Each subplot represents a different value of N, which also corresponds to the degrees of freedom (DoF) for the chi-squared distribution. The top left is (N=2), top right (N=4), bottom left (N=10), and bottom right (N=100). Each subplot contains a histogram (blue bars) representing the density of summed squared adjacent samples and a red curve showing the theoretical chi-squared probability density function (PDF) for the corresponding degrees of freedom N.

Figure 4 shows histograms of summed power estimates from adjacent samples where each is overlaid chi-squared distribution curve, shown for various summation window sizes N. Each subplot represents a different N value, corresponding to the degrees of freedom (DoF) of the chi-squared distribution. The histogram of N=2 approximates a chi-squared distribution with DoF = 2, showing a characteristic exponential decay in density. For $N = 4$ the histogram displays a wider spread and a peak near lower values. With $N = 10$, the summed data distribution starts to take on a bell-shaped form, indicating a broader range due to higher degrees of freedom. At $N = 100$, the histogram exhibits substantial variance, and the chi-squared curve peaks near 100, illustrating the wider density spread for high DoF. This arrangement demonstrates how increasing N affects the distribution's shape and progressively

approaches a Gaussian distribution as N rises, consistent with the central limit theorem.

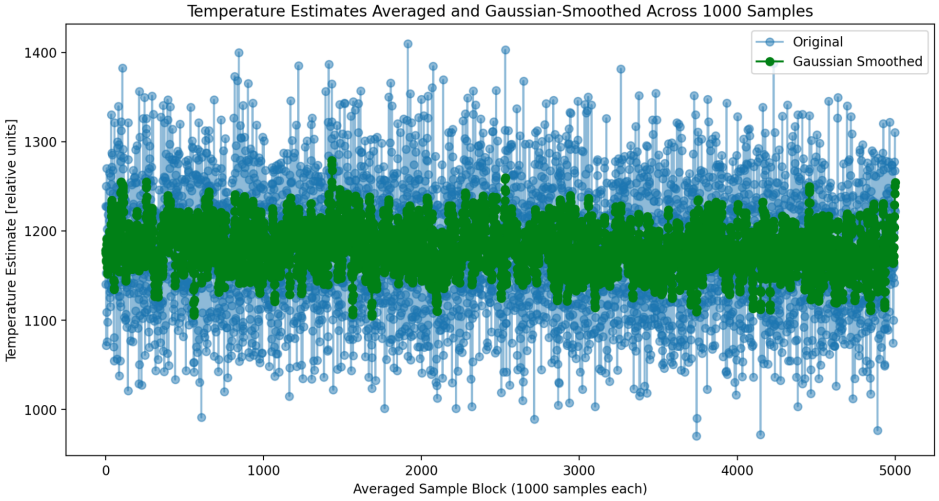


Figure 5: Temperature estimates averaged across 1000-sample blocks, with the original data (blue) and Gaussian-smoothed data (green) overlaid for comparison. The original data shows substantial variation with a mean of approximately 1180.33 $bits^2$ and a standard deviation of 65.94 $bits^2$, indicating considerable noise in temperature measurements. Applying Gaussian smoothing reduces this variation significantly with a similar mean of 1180.33 $bits^2$ but a much lower standard deviation of 25.26. This smoothed data provides a clearer and more stable view of the temperature trend.

The theoretical uncertainty in temperature based on the radiometer equation was calculated to be approximately 15.11 $bits^2$ with a mean of approximately 1180.33 $bits^2$. It is considerably lower than the measured standard deviation of the temperature estimates from the data (65.94 for the original and 25.26 after smoothing). This suggests that additional noise sources beyond thermal noise, potentially due to instrumental factors or residual interference are affecting the measurements. While Gaussian smoothing reduces the noise significantly, bringing the standard deviation closer to the theoretical expectation, there is still a gap. This indicates that despite attempts at noise reduction, the measurement system may still be subject to external noise contributions which impact precision of temperature estimates.

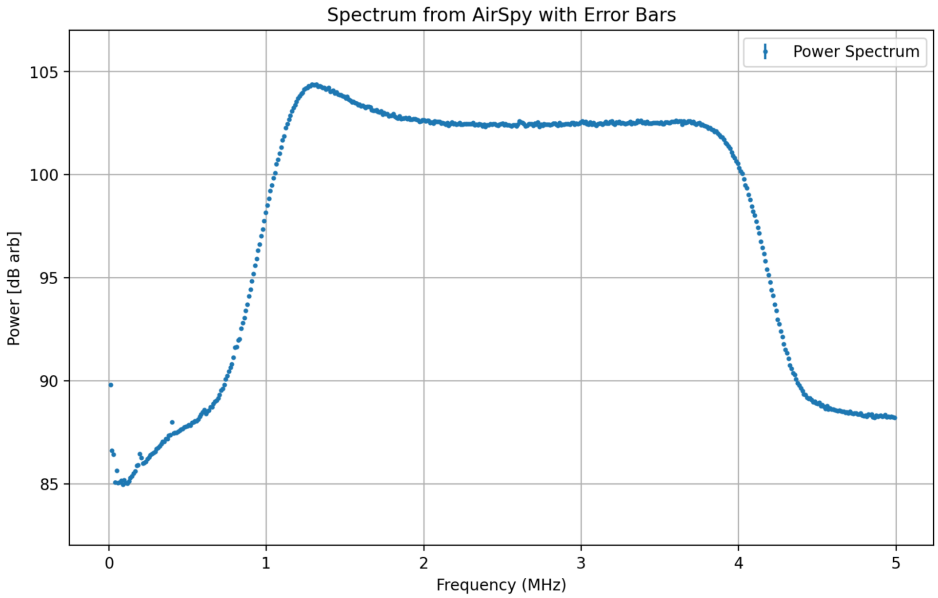


Figure 6: The plot shows the power spectrum in dB (arbitrary units) as a function of frequency (MHz) obtained from the AirSpy receiver. The spectrum reveals a characteristic response with a gradual increase in power, a plateau in the middle, and a decline at higher frequencies. This profile indicates the frequency response of the system and highlights potential filtering effects or bandwidth limitations of the receiver.

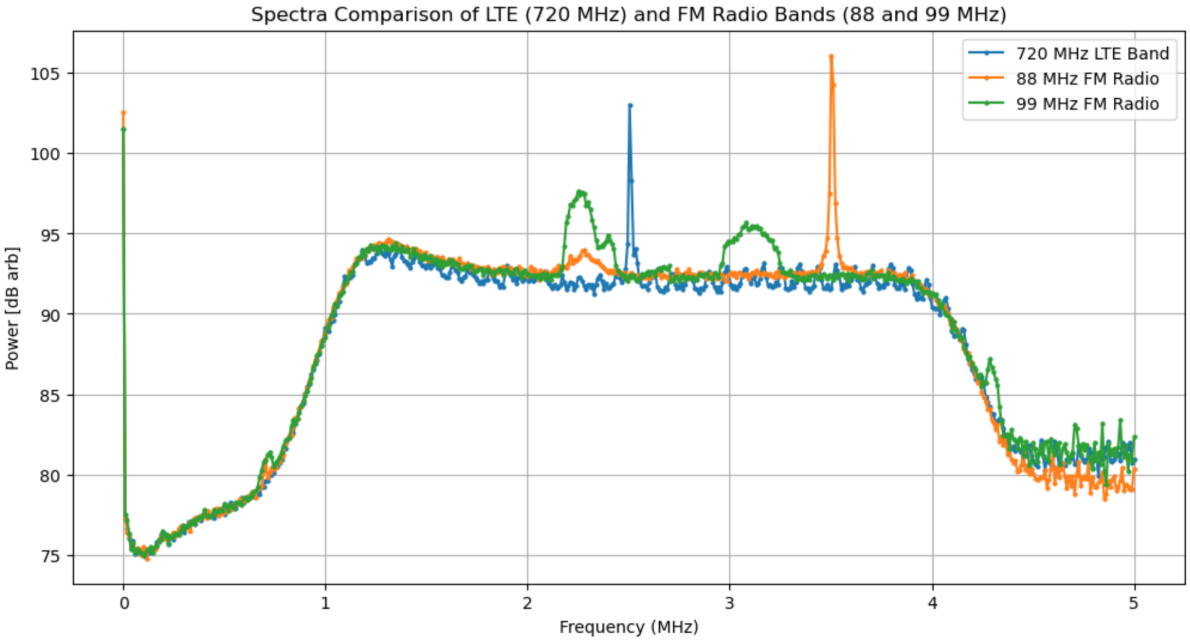


Figure 7: Comparison of power spectra for the LTE cell phone band (720 MHz) and FM radio bands (88 MHz and 99 MHz) using AirSpy. Each dataset shows the spectral response in the specified frequency range. Peaks in the spectra likely indicate unintended radio frequency interference (RFI) signals being picked up despite the AirSpy's input being terminated. This interference appears as spikes at specific frequencies in each band, particularly noticeable in the FM radio ranges around 88 MHz and 99 MHz.

The AirSpy spectrum plot shows a clear bandpass response, defined by distinct high- and low-pass characteristics on either side of the frequency range. The flat plateau region of the spectrum represents the usable band of the receiver where signal reception is strong and stable. On the left side of the spectrum, the low pass filter effect is evident as the power decreases sharply for lower frequencies, which starts from around 0 MHz and rises quickly to the passband. Similarly, the high pass filter effect on the right side is noticeable as the power decreases sharply after around 4 MHz, marking the edge of the band. These filters establish the AirSpy's effective bandwidth, approximately between 1 MHz and 4 MHz, where the signal strength is consistent.

The bandpass strength in the AirSpy receiver can be inferred by comparing the power difference between the middle and the edges. In the central region, the power reaches around 100 dB, whereas it but drops to approximately 85 dB near the band edges. This 15 dB difference indicates a relatively strong bandpass, as the edge of the band is about 15 times lower in power than the middle. This strong filtering ensures that out of band signals are effectively reduced, reducing unwanted interference. No significant RFI peaks are evident within the band, suggesting minimal external sources of interference. However, minor fluctuations and a gradual slope at the lower end of the band could indicate slight receiver variations or noise inherent to the device, but they do not significantly impact the main signal band.

Given that the AirSpy device was terminated and not connected to an antenna, ideally it should not pick up any external signals. However, the presence of distinct peaks in the power spectra in Figure 7 indicates that the device is still susceptible to leakage from strong external signals. Toronto's environment is "radio-loud" so that even extremely weak signals, down to -90 dB, can interfere with measurements. The peaks observed in this plot, particularly around 88 MHz and 99 MHz, suggest that local FM radio broadcasts are powerful enough to penetrate the device shielding, making them visible in the spectrum.

Temperature (°C)	**Variance** $[bits^2]$	**Uncertainty** $[bits^2]$
- 196.0 (Liquid Nitrogen)	1001.87	12.84
3.7	1169.94	18.47
25.0 (Room Temp)	1176.56	18.57
54.5	1102.89	17.46
72.6	1173.63	18.53
90.0	1208.20	19.06

Table 4: Summary of system temperature variance and measurement uncertainty at various load temperatures. The table presents variance in ADC readings ($bits^2$) and corresponding uncertainty across a range of temperatures, from liquid nitrogen (-196.0°C) to 90.0°C.

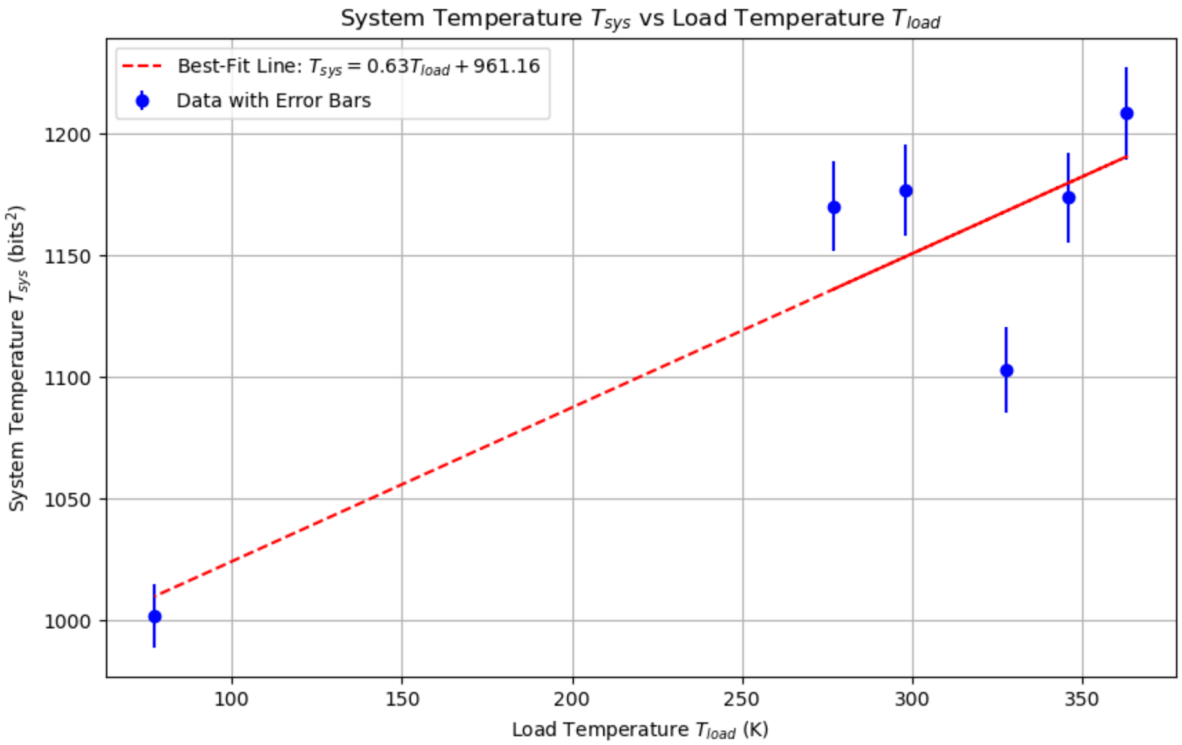
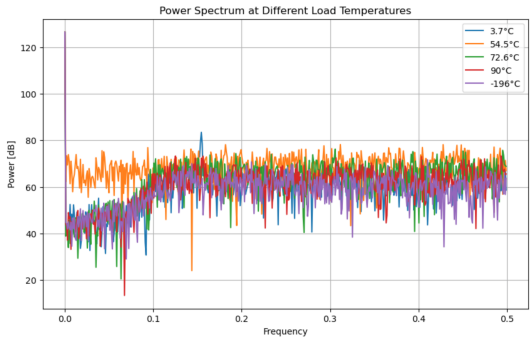
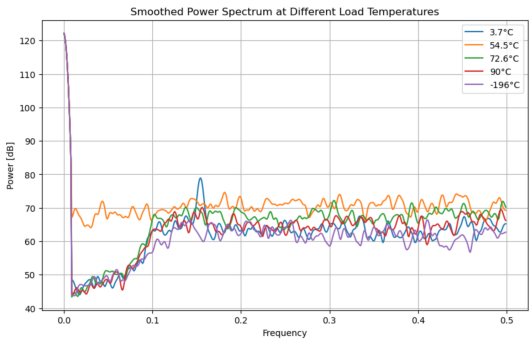


Figure 8: Plot of system temperature versus load temperature. Each data point includes error bars representing measurement uncertainty. The red dashed line shows the best-fit linear relationship between the two variables, with the equation $T_{\text{sys}} = 0.63T_{\text{load}} + 961.16\,\text{bits}^2$. Key parameters derived from this fit include a slope of $0.63\,\text{bits}^2/\text{K}$, intercept of $961.16\,\text{bits}^2$, and an x-intercept of approximately $-1522.40\,\text{K}$. The fraction of noise contribution from the terminator is 0.048 (4.8%), while the receiver contributes 0.952 (95.2%) to the total system temperature.

The slope of the line in the graph of $0.63\,\text{bits}^2/\text{K}$ represents the rate at which the system temperature T_{sys} changes with respect to the load temperature T_{load}. This indicates exactly how much of the system temperature is influenced by external thermal loads. The intercept of $961.16\,\text{bits}^2$ represents the inherent system temperature when there is no external thermal load which is essentially the internal noise of the receiver. The x-intercept, approximately $-1522.40\,\text{K}$, theoretically represents the load temperature at which the system temperature would drop to zero, which is physically unrealistic but does provide insight into the limitations and baseline noise of the system. In terms of the signal contribution it is seen that about 4.8% of the measured system temperature originates from the terminator's thermal load, while the receiver itself contributes the remaining 95.2%. The discrepancy between the fit and actual data uncertainties suggests that a more accurate fit is possible by applying a weighted linear least-squares method that accounts for measurement uncertainties.



(a) Power Spectrum at Different Load Temperatures. This plot shows the unsmoothed power spectrum for multiple load temperatures, revealing a highly variable signal with significant noise.



(b) Smoothed Power Spectrum at Different Load Temperatures. Applying smoothing reveals clearer trends across frequencies, reducing noise and allowing for better comparison between temperature spectra.

Figure 9: Comparison of power spectra at different load temperatures before and after smoothing.(a) displays the raw, unsmoothed spectra and shows the noise present in the data, while (b) presents the smoothed spectra, allowing clearer visualization of temperature-dependent variations in power.

After plotting the spectra for different load temperatures on the same axes, the thermal load trend is evident particularly in the smoothed plot. As the load temperature increases, there is a noticeable shift in the average power levels across the frequency spectrum. Higher load temperatures generally correspond with higher power readings which aligns with the expectation that thermal noise increases with temperature due to the increased motion of charge carriers. The smoothing process helps reveal this by reducing random fluctuations. Thus, while the thermal load trend is present but becomes more discernible only when the data is smoothed.

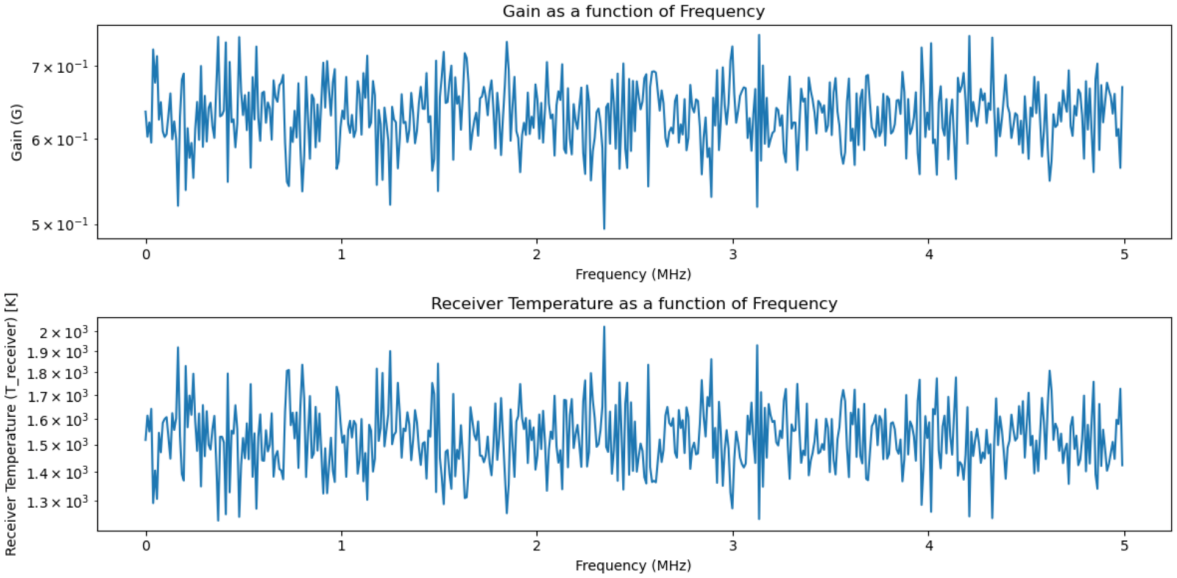


Figure 10: Comparison of receiver gain and receiver temperature as functions of frequency. The top plot illustrates the receiver gain (in dB) across a frequency range from 0 to 5 MHz and shows fluctuations in gain, with drops at specific frequencies. The bottom plot shows corresponding receiver temperature (in Kelvin), which also varies with frequency but remains relatively steady with minor peaks and valleys.

The plots in Figure 10 provide insight into the frequency dependent behaviour of both receiver gain and receiver temperature across the 0–5 MHz range. In the top plot, the receiver gain shows significant variability with dips at certain frequencies, indicating areas where the receiver's sensitivity sharply decreases. These could be caused by interference or even internal receiver filtering. A stable gain profile would typically be good to ensure consistent signal detection across the frequency range. In the bottom plot, the receiver temperature demonstrates relative stability with fluctuations that correlate

slightly with the gain dips. This suggests that variations in receiver gain can affect the apparent receiver temperature, as these fluctuations in gain might introduce or amplify noise, impacting the temperature measurement. The receiver temperature remains mostly around a baseline level, with smaller peaks that suggest minor thermal noise fluctuations across the frequency spectrum. Overall, these plots reveal exactly how receiver performance varies with frequency and also emphasizes the need for calibration to mitigate the impact of gain and temperature fluctuations on signal quality.

5 Discussion & Conclusion

This experiment examined the characteristics of thermal noise in a radio receiver setup by using the AirSpy device to quantify noise and explore the relationship between signal variance and load temperature. By varying the temperature of a terminator and analysing the resulting data, many insights were gained into the system's behaviour in response to thermal fluctuations, which is crucial for accurately interpreting radio signals in applications such as radio astronomy.

The results also confirmed the Gaussian nature of the raw electric field values, which was consistent with the Central Limit Theorem, as the thermal noise distribution closely followed a normal curve centred around zero. Squaring the electric field values to obtain power data provided with a chi-squared distribution, characteristic of thermal noise when it is sampled through detection methods. This statistical behaviour supports the appropriateness of using Gaussian and chi-squared distributions to model the noise in this system, which establishes a foundation for further calibration.

In exploring spectral properties, the FFT analysis provided insight into the bandpass characteristics of the AirSpy receiver and revealed a relatively stable usable band, minus some minor RFI influences. The presence of interference from strong FM radio signals at specific frequencies, highlighted the impact of environmental factors on radio observations. While the AirSpy is intended to operate in a "radio-silent" setting, this experiment highlighted the importance of isolating the device from external sources, particularly in "radio-loud" areas.

One critical finding from the calibration phase was the linear relationship between system temperature and load temperature, as plotted in Figure 8 graph. The slope indicated that approximately 4.8% of the system temperature originated from the thermal load, while 95.2% was the receiver itself. This high contribution from the receiver emphasizes the need for noise reduction techniques in sensitive measurements. Additionally, the y-intercept of the fit gave an estimate of the receiver's internal noise which characterises the device's baseline performance. Future improvements could focus on refining the precision of these measurements by potentially implementing a weighted least-squares approach to account for measurement uncertainties more accurately. Additionally, systematic uncertainties such as fluctuations in gain can be minimized through more controlled setup conditions or additional shielding to mitigate RFI effects.

In conclusion, this experiment successfully calibrated the AirSpy receiver, quantifying its response to thermal noise and establishing a reliable conversion from digital measurements to physical temperature units. This calibration proves crucial for accurate data interpretation with real world applications and provides a foundation for future work in thermal noise analysis and radio signal detection. By understanding and compensating for both the statistical and systematic noise in the system this study enhances the reliability of radio observations, particularly in fields such as astronomy and environmental monitoring where precise measurements are crucial.

6 Bibliography

Department of Astronomy and Astrophysics, *Thermal Radiation & the Statistics of Noise*: Lab Manual for AST325/326 (Fall 2024), University of Toronto, October 2024.

Appendix:

A Derivations

A.1 Derivation of the Radiometer Equation

The radiometer equation describes the uncertainty in a temperature measurement in terms of the integration time and bandwidth. We start by defining the proportional relationship between the temperature T and the power P in terms of the expectation of the squared electric field $\langle E^2 \rangle$:

$$T \propto P \propto \langle E^2 \rangle$$

The variance of the electric field E, denoted as $\mathrm{Var}(E)$, can be expressed as:

$$\mathrm{Var}(E) \equiv \langle (E - \langle E \rangle)^2 \rangle = \langle E^2 \rangle - \langle E \rangle^2 = \langle E^2 \rangle \propto T$$

When we measure a temperature using k samples, the estimate follows a chi-squared distribution χ^2_k, scaled by T/k. The uncertainty on such a measurement, σ_T, is given by:

$$\sigma_T = \sqrt{\mathrm{Var}\left(\frac{T}{k}\chi^2_k\right)} = \frac{T}{k} \cdot \sqrt{2k} = \frac{T}{\sqrt{k}}$$

For a timestream containing N samples, the fractional uncertainty in a temperature measurement is:

$$\frac{\sigma_T}{T} = \sqrt{\frac{2}{N}}$$

If these samples arrive with a cadence τ, then according to Nyquist's theorem, they cover a bandwidth $\Delta\nu = \frac{1}{2\tau}$ over a total time $t = N\tau$. Substituting these into the expression above, we arrive at the radiometer equation:

$$\frac{\sigma_T}{T} = \frac{1}{\sqrt{t\Delta\nu}}$$

This equation applies to any normally distributed data and is independent of amplification or other instrumental effects, making it an effective check for any non-Gaussian events or non-linear processing in the signal chain.

A.2 Derivation of the Least-Squares Fit

The Least-Squares Fit method is used to determine the best-fit line $y = mx + b$ for a set of data points (x_i, y_i). This is achieved by minimizing the sum of the squared residuals, S, where the residual for each point is the difference between the observed y_i and the predicted value from the line model.

The sum of squared residuals is given by:

$$S = \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

To find the best-fit values of m and b, we differentiate S with respect to m and b, and set these derivatives to zero.

1. Differentiating with respect to m:

$$\frac{\partial S}{\partial m} = -2\sum_{i=1}^{n} x_i (y_i - (mx_i + b)) = 0$$

2. Differentiating with respect to b:

$$\frac{\partial S}{\partial b} = -2\sum_{i=1}^{n} (y_i - (mx_i + b)) = 0$$

Expanding these equations, we obtain the normal equations:

$$\sum_{i=1}^{n} y_i = m\sum_{i=1}^{n} x_i + nb$$

$$\sum_{i=1}^{n} x_i y_i = m \sum_{i=1}^{n} x_i^2 + b \sum_{i=1}^{n} x_i$$

Solving these simultaneous equations for m and b yields the best-fit line parameters:

$$m = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - \left(\sum x_i\right)^2}$$

$$b = \frac{1}{n} \left(\sum y_i - m \sum x_i \right)$$

Thus, the slope m and intercept b are calculated to minimize the sum of squared residuals, providing the best-fit line for the given data points.

B Sample Calculations

This section includes sample calculations performed during the analysis.

B.1 Temperature Uncertainty Calculation

To calculate the temperature uncertainty using the radiometer equation, we use the following formula:

$$\frac{\sigma_T}{T} = \frac{1}{\sqrt{t \Delta \nu}}$$

where:

- $\Delta \nu$ is the bandwidth of the measurement,
- t is the integration time,
- σ_T is the uncertainty in temperature, and
- T is the system temperature.

Given:

$$\text{Frequency} = 609\,\text{Hz}$$

1. Determine the Bandwidth $\Delta \nu$: Since the frequency of measurement is 609 Hz, we assume the bandwidth $\Delta \nu = 609\,\text{Hz}$.
2. Set the Integration Time t: For this example, let's assume an integration time $t = 10\,\text{seconds}$.
3. Calculate the Temperature Uncertainty:

Substitute the values into the radiometer equation:

$$\frac{\sigma_T}{T} = \frac{1}{\sqrt{10 \times 609}}$$

Simplify the calculation:

$$\frac{\sigma_T}{T} = \frac{1}{\sqrt{6090}}$$

$$\frac{\sigma_T}{T} = \frac{1}{78.07} \approx 0.0128$$

Therefore, the fractional uncertainty in temperature is approximately 0.0128 or 1.28%.

C Code

C.0.1 Code to Plot Data and Calculate Statistics

```
# Section 4.1: plotting the graph

# import libraries
import matplotlib.pyplot as plt
import numpy as np

# plot first 1024 samples of the data
sample_size = 1024
time_axis = np.arange(sample_size)
plt.figure(figsize=(10, 6), dpi=200)
plt.scatter(time_axis, data[:sample_size], s=10)
plt.title("Data for 609 Hz, first 1024 samples from AirSpy")
plt.xlabel("Sample Number")
plt.ylabel("ADC Value [bits]")
plt.show()

# plotting all the data
time_axis2 = np.arange(len(data))
plt.figure(figsize=(10, 6), dpi=200)
plt.scatter(time_axis2, data[:], s=10)
plt.title("Data for 609 Hz")
plt.xlabel("Sample Number")
plt.ylabel("ADC Value [bits]")
plt.show()

# defining the second half of the data for plotting
split = len(data) // 2
second_half_data = data[split:]
sec_half = np.arange(split, len(data))

# plotting the second half of the data
plt.figure(figsize=(10, 6), dpi=200)
plt.scatter(sec_half, second_half_data, s=10)
plt.title("Data for 609 Hz, Second Half of Samples from AirSpy")
plt.xlabel("Sample Number")
plt.ylabel("ADC Value [bits]")
plt.show()

# calculating mean, median, standard deviation, and variance
mean_value = np.mean(second_half_data)
median_value = np.median(second_half_data)
std_dev = np.std(second_half_data)
variance = np.var(second_half_data)

mean_value, median_value, std_dev, variance
```

Listing 1: Python code for plotting data and calculating statistical properties.

C.0.2 Code to Plot Histograms of ADC Values and Power Data

```

# import libraries
import matplotlib.pyplot as plt
from scipy.stats import chi2, norm
import numpy as np

# calculating mean and standard deviation for the raw ADC data
# only use second half data from now
mean_adc = np.mean(second_half_data)
std_adc = np.std(second_half_data)

# plot the histogram of E-field without squaring
plt.figure()
plt.hist(second_half_data, bins=50, density=True, alpha=0.5, edgecolor='gray', label="
    ADC Values")
x_values = np.linspace(-100, 100, 1000)
gaussian_fit = norm.pdf(x_values, mean_adc, std_adc)
```

```
plt.plot(x_values, gaussian_fit, color='orange', linewidth=2, label="Gaussian Fit")
plt.title("Histogram of Raw ADC Values (E-field) with Gaussian Fit")
plt.xlabel("ADC Value (Bits)")
plt.ylabel("Frequency")
plt.legend()
plt.xlim(-100, 100)
plt.show()

# recentering the ADC data by subtracting the mean
# to fix the data
centered_data = second_half_data - mean_adc

# calculating power based on the centered data
power_data = centered_data ** 2
mean_power = np.mean(power_data)
std_power = np.std(power_data)

# plotting histogram of the power data
plt.figure(figsize=(10, 6))
plt.hist(power_data, bins=100, density=True, alpha=0.5, edgecolor='gray', label="Power
    Data")
x_values_power = np.linspace(0, max(power_data), 1000)
gaussian_fit_power = norm.pdf(x_values_power, mean_power, std_power)
plt.plot(x_values_power, gaussian_fit_power, color='orange', linewidth=2, label="
    Gaussian Fit")

plt.title("Histogram of Power Data (Centered E-field Squared) with Gaussian Fit")
plt.xlabel("Power [$Bits^2$]")
plt.xlim(-20, 6000)
plt.ylabel("Frequency")
plt.legend()
plt.show()
```

Listing 2: Python code for plotting histograms of raw ADC values and power data with Gaussian fits.

C.0.3 Code to Plot Histograms with Chi-Squared Distributions

```

# import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2

# create a function to sum adjacent samples and plot histograms and overlay chi-squared
    distributions

def hist_plot(data, n):
    # centering the data by subtracting the mean and making subplots
    centered_data = data - np.mean(data)
    fig, axs = plt.subplots(2, 2, figsize=(14, 10))
    axs = axs.flatten()  # flatten to index easily

    for idx, n in enumerate(n_values):
        # summing over every N adjacent samples and plotting histogram
        summed_data = np.array([np.sum(centered_data[j:j+n]**2) for j in range(0, len(
    centered_data), n)])
        axs[idx].hist(summed_data, bins=20, density=True, alpha=0.7, color='blue',
    edgecolor='gray', label=f"Summed Samples (N={n})")

        # making the chi-squared distribution
        x = np.linspace(0, max(summed_data), 1000)
        dist_chi2 = chi2.pdf(x, df=n)

        # plotting the chi-squared distribution
        axs[idx].plot(x, dist_chi2, 'r-', label=r'$\chi^2$ Distribution with DoF=' + str
    (n))
        axs[idx].set_xlabel("Summed Power Estimate")
        axs[idx].set_xlim(0, max(x))
        axs[idx].set_ylabel("Density")
        axs[idx].set_title(f"Histogram of Summed Samples with Chi-Squared Distribution (
    N={n})")
        axs[idx].legend()
```

```
    plt.tight_layout()
    plt.show()

n = [2, 4, 10, 100]
hist_plot(data, n)
```

Listing 3: Python code for plotting histograms of summed samples with overlaid chi-squared distributions.

C.0.4 Code to Calculate and Plot Temperature Estimates

```

# import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d

# define the second half of the data for temperature analysis
halfway_point = len(data) // 2
second_half_data = data[halfway_point:]

# function to calculate the averaged temperature estimates over 1000-sample intervals
def temp_estimate(data, window_size=1000):
    power_data = data ** 2
    # compute averaged temperature estimates
    averaged_temperatures = [np.mean(power_data[i:i+window_size]) for i in range(0, len(
    power_data), window_size)]
    return np.array(averaged_temperatures)

# calculate temperature estimates for second half
temp_estimate = temp_estimate(second_half_data)

# plot the temperature estimates as a time stream
plt.figure(figsize=(12, 6), dpi=200)
plt.plot(temp_estimate, marker='o', linestyle='-')
plt.title("Temperature Estimates Averaged Across 1000 Samples")
plt.xlabel("Averaged Sample Block (1000 samples each)")
plt.ylabel("Temperature Estimate [relative units]")
plt.show()

# calculate mean and standard deviation
mean_temp = np.mean(temp_estimate)
std_temp = np.std(temp_estimate)

mean_temp, std_temp

# apply Gaussian filter to the temperature estimates
smoothed_temp_estimate = gaussian_filter1d(temp_estimate, sigma=2)

# plot the original and Gaussian-smoothed temperature estimates
plt.figure()
plt.plot(temp_estimate, marker='o', linestyle='-', alpha=0.5, label='Original')
plt.plot(smoothed_temp_estimate, marker='o', linestyle='-', color='green', label='
    Gaussian Smoothed')
plt.title("Temperature Estimates Averaged and Gaussian-Smoothed Across 1000 Samples")
plt.xlabel("Averaged Sample Block (1000 samples each)")
plt.ylabel("Temperature Estimate [relative units]")
plt.legend()
plt.show()

# calculate mean and standard deviation of Gaussian-smoothed temperature estimates
mean_gaussian_temp = np.mean(smoothed_temp_estimate_gaussian)
std_gaussian_temp = np.std(smoothed_temp_estimate_gaussian)

mean_gaussian_temp, std_gaussian_temp
```

Listing 4: Python code for calculating and plotting temperature estimates with Gaussian smoothing.

C.0.5 Code to Calculate and Plot Power Spectrum

```python

# import libraries
import numpy as np
import matplotlib.pyplot as plt

chunk_size = 1024  # length of each FFT chunk
sample_rate = 10e6  # sample rate in Hz
n_samp = (len(second_half_data) // chunk_size) * chunk_size

# reshape and perform FFT
reshape_data = second_half_data[:n_samp].reshape(-1, chunk_size)
fft_result = np.fft.fft(reshape_data, axis=1)

# calculating power spectrum and mean and uncertainty
power_spec = (fft_result.real**2 + fft_result.imag**2).sum(axis=0)
mean_power = np.mean(power_spec[:512])
num_segments = reshape_data.shape[0]
uncertainty = mean_power / np.sqrt(num_segments)
freqs = np.fft.fftfreq(chunk_size, d=1/sample_rate)[:chunk_size // 2] / 1e6  # Convert
    to MHz

# plotting the spectrum in dB
plt.figure(figsize=(10, 6), dpi=200)
plt.errorbar(
    freqs, 10 * np.log10(power_spec[:chunk_size // 2]),
    yerr=10 * np.log10(1 + uncertainty / mean_power),
    fmt='.', markersize=4, label="Power Spectrum"
)
plt.title("Spectrum from AirSpy")
plt.xlabel("Frequency (MHz)")
plt.ylabel("Power [dB arb]")

plt.ylim(82, 107)
plt.legend(loc="best")
plt.grid(True)
plt.show()
```

Listing 5: Python code for calculating and plotting the power spectrum.

C.0.6 Code to Calculate and Plot Spectra of LTE and FM Radio Bands

```python

# import libraries
import numpy as np
import matplotlib.pyplot as plt

def calc_spec(data, chunk_size=1024):
    n_samples = (len(data) // chunk_size) * chunk_size
    reshaped_data = data[:n_samples].reshape(-1, chunk_size)
    # do FFT and calculate power spectrum
    fft_result = np.fft.fft(reshaped_data, axis=1)
    power_spectrum = (fft_result.real**2 + fft_result.imag**2).sum(axis=0)

    return power_spectrum[:chunk_size // 2]  # Use only positive frequencies

# get datasets
data_720 = np.fromfile('group_n_720.dat', dtype=np.int16) - 2**11
data_88 = np.fromfile('group_n_88.dat', dtype=np.int16) - 2**11
data_99 = np.fromfile('group_n_99.dat', dtype=np.int16) - 2**11

# calculating spectra
spec_720 = calc_spec(data_720)
spec_88 = calc_spec(data_88)
spec_99 = calc_spec(data_99)

# frequencies axis in MHz
sampling_rate = 10e6
frequency_axis = np.linspace(0, sampling_rate / 2, len(spec_720)) / 1e6

# plotting all spectra on the same figure
plt.figure(figsize=(12, 6))
```

```
plt.plot(frequency_axis, 10 * np.log10(spec_720), label="720 MHz LTE Band", marker='o', markersize=2)
plt.plot(frequency_axis, 10 * np.log10(spec_88), label="88 MHz FM Radio", marker='o', markersize=2)
plt.plot(frequency_axis, 10 * np.log10(spec_99), label="99 MHz FM Radio", marker='o', markersize=2)

plt.xlabel("Frequency (MHz)")
plt.ylabel("Power [dB arb]")
plt.title("Spectra Comparison of LTE (720 MHz) and FM Radio Bands (88 and 99 MHz)")
plt.legend()
plt.grid()
plt.show()
```

Listing 6: Python code for calculating and plotting spectra of LTE (720 MHz) and FM Radio bands (88 and 99 MHz).

C.0.7 Code to Calculate Statistics of Data

```

data_037 = np.fromfile('group_n_609_037.dat', dtype=np.int16) - 2**11

# calculate mean, median, standard deviation, and variance
mean_value = np.mean(data_037)
median_value = np.median(data_037)
std_dev = np.std(data_037)
variance = np.var(data_037)

mean_value, median_value, std_dev, variance
```

Listing 7: Python code for calculating mean

C.0.8 Code for System Temperature vs. Load Temperature Analysis

```

# import libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

# given data
T_load_C = [3.7, 25.0, 54.5, 72.6, 90.0, -196]  # Load temperatures in Celsius
T_sys = [1169.94, 1176.56, 1102.89, 1173.63, 1208.20, 1001.87]  # System temperatures in bits^2
uncert = [18.47, 18.57, 17.46, 18.53, 19.06, 12.84]  # Uncertainties in bits^2

# converting to Kelvin
T_load_K = [temp + 273.15 for temp in T_load_C]

# plot T_sys vs T_load with error bars
plt.figure(figsize=(10, 6))
plt.errorbar(T_load_K, T_sys, yerr=uncert, fmt='o', label="Data with Error Bars", color='blue')
plt.xlabel("Load Temperature $T_{load}$ (K)")
plt.ylabel("System Temperature $T_{sys}$ (bits$^2$)")
plt.title("System Temperature $T_{sys}$ vs Load Temperature $T_{load}$")

# doing linear regression for best-fit line
slope, intercept, r_value, p_value, std_err = linregress(T_load_K, T_sys)
best_fit_line = [slope * temp + intercept for temp in T_load_K]
plt.plot(T_load_K, best_fit_line, color='red', linestyle='--', label=f"Best-Fit Line: $T_{{sys}} = {slope:.2f} T_{{load}} + {intercept:.2f}$")
plt.legend()
plt.grid(True)
plt.show()

# calculate the x-intercept
x_intercept = -intercept / slope if slope != 0 else None

# interpret the slope and intercept
```

```
slope_inter = slope
intercept_inter = intercept
frac_terminator = slope * T_load_K[-1] / (slope * T_load_K[-1] + intercept)
frac_receiver = intercept / (slope * T_load_K[-1] + intercept)

slope, intercept, x_intercept, frac_terminator, frac_receiver
```

Listing 8: Python code for analyzing and plotting the relationship between system temperature and load temperature with error bars and linear regression.

C.0.9 Code for System Temperature vs. Load Temperature with Linear Regression

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress

# given data
T_load_C = [3.7, 25.0, 54.5, 72.6, 90.0, -196]  # temp in Celsius
uncertainties = [18.47, 18.57, 17.46, 18.53, 19.06, 12.84]  # uncertainties in bits^2
T_sys = [1169.94, 1176.56, 1102.89, 1173.63, 1208.20, 1001.87]  # system temp in bits^2

# converting T_load to Kelvin
T_load_K = [temp + 273.15 for temp in T_load_C]

# plotting T_sys vs T_load with error bars
plt.figure(figsize=(10, 6))
plt.errorbar(T_load_K, T_sys, yerr=uncertainties, fmt='o', label="Data with Error Bars",
     color='blue')
plt.title("System Temperature $T_{sys}$ vs Load Temperature $T_{load}$")
plt.xlabel("Load Temperature $T_{load}$ (K)")
plt.ylabel("System Temperature $T_{sys}$ (bits$^2$)")

# perform linear regression and plot
slope, intercept, r_value, p_value, std_err = linregress(T_load_K, T_sys)
best_fit_line = [slope * temp + intercept for temp in T_load_K]
plt.plot(T_load_K, best_fit_line, color='red', linestyle='--', label=f"Best-Fit Line:
    $T_{{sys}} = {slope:.2f} T_{{load}} + {intercept:.2f}$")
plt.legend()
plt.grid(True)
plt.show()

# find the x-intercept
x_inter = -intercept / slope if slope != 0 else None

# interpreting the slope and intercept
slope_interp = slope
intercept_interp = intercept
frac_term = slope * T_load_K[-1] / (slope * T_load_K[-1] + intercept)
frac_receiver = intercept / (slope * T_load_K[-1] + intercept)

slope, intercept, x_inter, frac_term, frac_receiver
```

Listing 9: Python code for plotting system temperature vs. load temperature with error bars and linear regression.

C.0.10 Code for Power Spectrum and Smoothed Power Spectrum at Different Load Temperatures

```

import numpy as np
import matplotlib.pyplot as plt

# loading in data files and slicing
data_3_7 = np.fromfile("group_n_609_037.dat", dtype=np.int16)[:1024]
data_54_5 = np.fromfile("group_n_609_545.dat", dtype=np.int16)[:1024]
data_72_6 = np.fromfile("group_n_609_726.dat", dtype=np.int16)[:1024]
data_90 = np.fromfile("group_n_609_900.dat", dtype=np.int16)[:1024]
data_nitro = np.fromfile("group_n_609_nitrogen.dat", dtype=np.int16)[:1024]

```

```
# list datasets and corresponding temperatures
datasets = [data_3_7, data_54_5, data_72_6, data_90, data_nitro]
temperatures = [3.7, 54.5, 72.6, 90, -196]
labels = [f"{temp} C " for temp in temperatures]

# plot spectra for each dataset
plt.figure(figsize=(10, 6))
for data, label in zip(datasets, labels):
    # compute Fourier Transform
    fft_result = np.fft.fft(data)
    # Power Spectrum calculation
    power_spec = np.abs(fft_result) ** 2
    # filter by the positive frequencies
    freqs = np.fft.fftfreq(len(data))
    positive_freqs = freqs[:len(freqs) // 2]
    pos_power_spec = power_spec[:len(power_spec) // 2]

    # plot the power spectrum in dB scale
    plt.plot(positive_freqs, 10 * np.log10(pos_power_spec), label=label)
plt.xlabel("Frequency")
plt.ylabel("Power [dB]")
plt.title("Power Spectrum at Different Load Temperatures")
plt.legend()
plt.grid()
plt.show()

from scipy.ndimage import gaussian_filter1d

# plotting the smoothed spectra for each dataset
plt.figure(figsize=(10, 6))
for data, label in zip(datasets, labels):
    # compute Fourier Transform
    fft_result = np.fft.fft(data)
    # Power Spectrum calculation
    power_spec = np.abs(fft_result) ** 2
    # filter by the positive frequencies
    freqs = np.fft.fftfreq(len(data))
    positive_freqs = freqs[:len(freqs) // 2]
    pos_power_spec = power_spec[:len(power_spec) // 2]
    # apply Gaussian smoothing to the power spectrum
    smoothed_power_spectrum = gaussian_filter1d(pos_power_spec, sigma=2)
    # plot the smoothed power spectrum in dB scale
    plt.plot(positive_freqs, 10 * np.log10(smoothed_power_spectrum), label=label)
plt.xlabel("Frequency")
plt.ylabel("Power [dB]")
plt.title("Smoothed Power Spectrum at Different Load Temperatures")
plt.legend()
plt.grid()
plt.show()
```

Listing 10: Python code for plotting power spectrum and smoothed power spectrum at different load temperatures.