Lab Report Two: Thermal Radiation and The Statistics of Noise

Jinhao Zhang
Group Partners: Mike Gu, Epsilon Mao
Email: jinny.zhang@mail.utoronto.ca
Utorid: zha12748
Lab Group: D

November 5, 2024

ABSTRACT

In this lab, we analyzed thermal noise in a radio receiver using coherent measurement techniques, contrasting with photon detection methods based on the photoelectric effect. Key principles such as the Rayleigh-Jeans approximation, Central Limit Theorem, and the radiometer equation guided our analysis. By measuring power across various load temperatures, we calibrated the receiver's response, determining a gain of 0.435 ± 0.047 [bits^2/K] and an offset of 904.40 ± 13.68 [bits^2]. The receiver temperature was found to be 2079.32 ± 225.59 K, a value higher than expected, likely due to extended use without cooldown periods. Gaussian and chi-squared fits indicated good data quality, with minimal Radio Frequency Interference (RFI) observed in the power spectrum. These results align with theoretical predictions and demonstrate the effectiveness of coherent measurement in capturing radio-frequency noise, essential for precise radio astronomy applications.

1. Introduction

In radio astronomy, thermal noise is an inevitable and fundamental factor that affects the measurements of faint astronomical signals. Thermal noise is from the random migration of electrons caused by temperature changes Vanderlinde (2024a). For higher frequencies of light, such as visible or ultraviolet, detection methods rely on the photoelectric effect and other principles to create precise imaging and detection systems. However, as photon energy decreases with lower frequencies, these techniques become less effective. Radio waves lack the energy to be detected through the photoelectric effect. Because of this, radio astronomers use coherent detection methods, which involves measuring the oscillating electric field of the incoming radiation Vanderlinde (2024b). This approach enables the recording of time-ordered data, or "timestreams," representing the electric field amplitude over time.

In this low-energy environment, thermal noise follows the blackbody radiation model, with power output as a function of temperature. In the radio frequency range, the behaviour of thermal noise is governed by the Rayleigh-Jeans equation, which states that for radio frequency where kT >> hv, the power detected within a given band is directly proportional to the temperature of the source AST325/326 (2024). Because of this, radio astronomers can use an "equivalent temperature" to describe incoming signals, which makes measurement calibration and interpretation easier. By calibrating a receiver to measure power as a function of temperature, astronomers can directly relate digital signal values to physical units. This calibration process is essential for accurately measuring thermal noise and other signal components.

Radio detection also relies on a process known as heterodyning, where the incoming signal is mixed with a reference frequency generated by a local oscillator. This technique shifts the signal to a different frequency

range, called the intermediate frequency, where it can be more easily amplified, filtered, and digitized. In this lab, the signal is mixed, amplified, and digitized by an Analog-to-Digital Converter (ADC) within the AirSpy receiver. These time-ordered samples can then be processed, including Fourier transformation to produce a power spectrum representing the frequency components of the incoming signal.

In this lab, we use an AirSpy receiver to measure the thermal noise produced by a controlled thermal source. We also change the temperature of the source to see how it affects the detected power. The objective is to calibrate the receiver, analyze noise characteristics, and verify the radiometer equation, which models the expected uncertainty in temperature measurements based on observation time and bandwidth AST325/326 (2024). By examining how thermal noise behaves in a radio receiver, we aim to improve understanding of noise statistics and enhance signal detection in practical astronomical observations. We summarize the data and observations in Section 2, explain the theoretical basis and key equations, plots, and findings in Section 3.

2. Data and Observation

The following table gives a detailed description of the data collection process in each lab session.

Date	Personnel	Notes
3/10/2024	**M. Yiquan, Z. Jinhao, G. Yanjie**	Familiarize the AirSpy, Collected data in room temperature and warm water for inspection
10/10/2024	**M. Yiquan, Z. Jinhao, G. Yanjie**	Collected raw data for 4.1-4.4 using 609 MHz and data size of 10^8
17/10/2024	**M. Yiquan, Z. Jinhao, G. Yanjie**	Collected data up to 4.6 using same setting and started analysis
24/10/2024	**M. Yiquan, Z. Jinhao, G. Yanjie**	Changed frequency to 613 MHz and successfully collected good data. continued on data analysis

Table 1:: Group D Data Collection Layout

2.1. AirSpy Setup and Inspecting Test Data (Activity 4.1)

To setup the AirSpy, we first connect the AirSpy to the laptop and open up the Linux Terminal. Following the command given to us in Section 3.1 of the lab manual, we gathered our set of data at room temperature AST325/326 (2024). Below are the specifications and plot:

Setting	Value
Frequency	1000 MHz
Sample Rate	5 MHz
Sample Type	uint16 Real
VGA Gain	15
Mixer Gain	15
LNA Gain	14
Number of Samples	1,000,000

Table 2:: Initial Airspy Settings

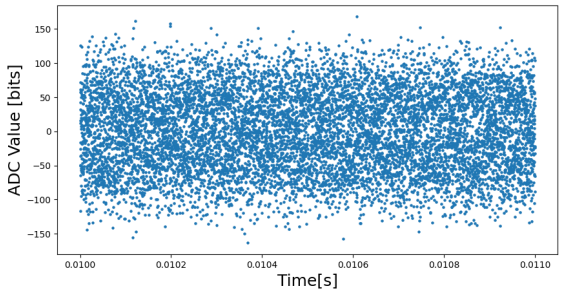


Fig. 1.—: Scatter of Section of Initial Test Data

Professor Vanderlinde mentioned that the data can be heavily influced by other sources, so we used our call phones to call eachother and placed the AirSpy in between and recorded the data to see if there are any differences Vanderlinde (2024a). However, we obtained similar scatter plots and similar mean, medican,

standard deviation, and variance from the intial data without the phone call. We think that the radiation from the phone calls does not affect the data too much at 1000 MHz.Below are the mean, median, standard deviation, and variance of the test data with and without phone calls:

Statistics	Without Phone Call	With Phone Call
Mean	-0.018 bits	- 0.018 bits
Median	0.000 bits	0.000 bits
Standard Deviation	28.535 bits	30.429 bits
Variance	814.271 bits	823.583 bits

Table 3:: Statistics of AirSpy Data

2.2. Finding the Right Frequency and Examining Distributions (Activity 4.2 and 4.3)

We need to find out whether the data collected is a good fit for a Gaussian. We first used the histogram code from lab 1 to plot out the histogram plot of the data. When we first plotted out the 1000 MHz data, we observed multiple peaks in the histogram (see Figure 2). After trying out different frequencies, we found out that the Lo frequency of 609 MHz gives us a good Gaussian Histogram plot.

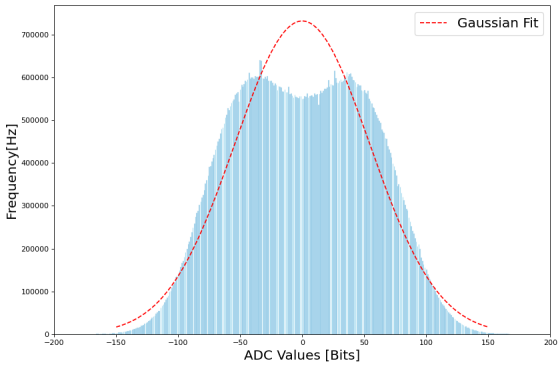


Fig. 2.—: Histogram of Double Peak Data with 1000 MHz Frequency

Since there can be random errors in our data, we decided to increase our sample size to 10^8. Later on in lab session 4, we completed the spectrum code in Activity 4.3 and started examining the spectrum for our data collection and found out that that the best frequency to use is 613 MHz. The final specifications of the AirSpy for the data analysis is given by the following table:

Setting	Value
Frequency	613MHz
Sample Rate	5MHZ
Sample Type	uint16 Real
VGA Gain	15
Mixer Gain	15
LNA Gain	14
Number of Samples	100,000,000

Table 4:: Final Airspy Settings

2.3. Varying Temperatures (Activity 4.4 to 4.6)

For this lab, we decided to measure the data in hot water, ice water, room temperature, dry ice, and liquid nitrogen. For the ice water and hot water data gathering, we first wrap a glove around the terminator of the AirSpy and put the thermometer and AirSpy into the water. Wait about 30 seconds to ensure that the terminator reaches equalibrium. Run the data collection and record the temperature at the time it was taken. Since the mediums cannot maintain their temperature in the classroom, we need to include the uncertainties in the measurements. We need to also record the final temperature after data collection is done. We would use the mean as the temperature and half of the difference as the uncertainty. Since the dry ice and liquid nitrogen were collected without a thermometer, we are going to assume they are -100 °C and -200 °C respectively with no uncertainty as suggested in the lab manualAST325/326 (2024).

Another problem we encountered was during our data collection for room temperature, we found that after a certain time, all the data values became -2048 (See Figure 3). This means that the AirSpy were not taking in any data. We realized this could be that the terminator was loose as we just put the AirSpy in liquid nitrogen before this. The terminator could have loosened as it was in the liquid nitrogen due to how cold it is We realized this and made sure that the terminator is tight on the FlyCap before every measurement. In the last lab, we were able to successfully gathered all the data for the different temperatures.

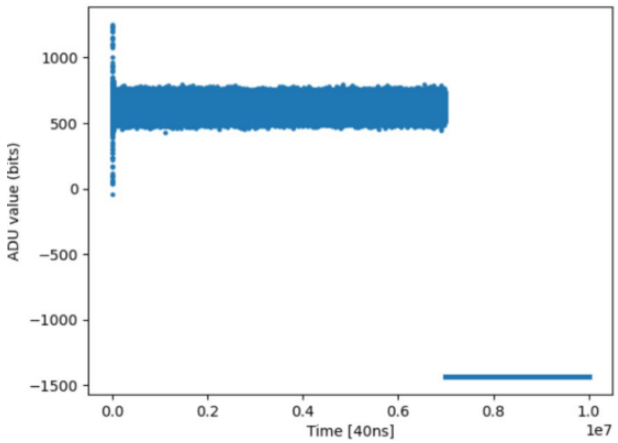


Fig. 3.—: Scatter Plot of Data With -2048 Values

Medium	Initial Temperature	Final Temperature	Load Temperature
Hot Water 1	76.0 °C	75.6 °C	75.8 ± 0.2 °C
Hot Water 2	70.0 °C	69.8 °C	69.9 ± 0.1 °C
Hot Water 3	64.2 °C	64.0 °C	64.1 ± 0.1 °C
Hot Water 4	60.0 °C	59.8 °C	59.9 ± 0.1 °C
Hot Water 5	55.0 °C	54.8 °C	54.9 ± 0.1 °C
Room Temperature	25.9 °C	25.9 °C	25.9 ± 0.0 °C
Cold Water	4.8 °C	4.8 °C	4.8 ± 0.0 °C
Dry Ice	-100.0 °C	-100.0 °C	-100.0 ± 0.0 °C
Liquid Nitrogen	-200.0 °C	-200.0 °C	-200.0 ± 0.0 °C

Table 5:: Table of Different Temperature Mediums and Uncertainty

Medium	ADC Mean [Bits^2]	ADC Variance [Bits^2]	σ [Bits^2]
Hot Water 1	3.13	1068.92	0.151
Hot Water 2	3.23	1054.90	0.149
Hot Water 3	3.20	1045.46	0.148
Hot Water 4	3.25	1040.34	0.147
Hot Water 5	3.22	1035.21	0.146
Room Temperature	3.15	1033.51	0.146
Cold Water	3.24	1032.11	0.146
Dry Ice	3.20	996.59	0.141
Liquid Nitrogen	3.20	923.75	0.131

Table 6:: Table of Statistics of Different Medium Measurements

3. Data Reduction and Analysis

3.1. Activity 4.2

The Central Limit Theorem suggest that our ADC values should converge to a Gaussian. This means that the chi-squared distribution would be able to describe the power of the data AST325/326 (2024). Note that when summing N powers and plotting the histogram, it is equal to the chi-squared with the N degree of freedom. For the plotting, I used the measurement from room temperature.

The implementation can be found in the python code in Appendix 5. First, the mean is removed from the data. Then, the data is divided by their standard deviation to standardize it. It is then grouped based on N and plot out the histogram. Afterwards, find the χ^2 values using the chi2.pdf function. Next, I need to scale the χ^2 values so that they would have unit uncertainty. This is done by multiplying the χ^2 values by the area of the histogram, where the area is calculated from summing up the frequency multiplied by the bin width in each bar. Finally, I plotted out both graphs. Below is my χ^2 plot:

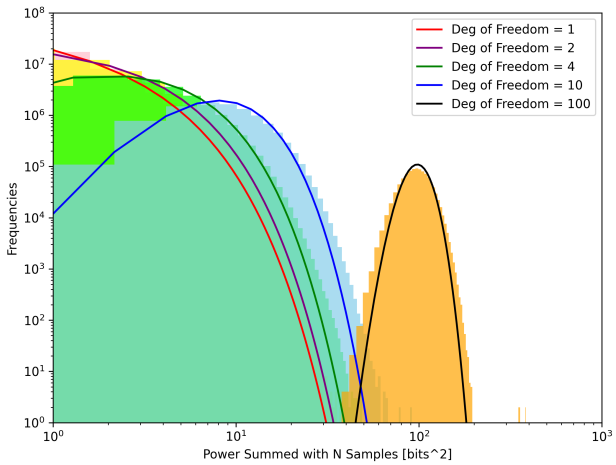


Fig. 4.—: Chi-Square Fitting of Room Temperature Data

From the graph above, we can see that with higher degrees of freedom, the histogram leads to more stable, symmetric distributions. This is because of the Central Limit Theorem, as the size of the data get

larger, the distribution would tend to a Gaussian distribution. We can see that the plot with degrees of freedom of 100 has a very nice dome shape, which is very similar to a log plot of a Gaussian.

Now, I need to write a function that averages the 1000 samples of the powers. As mentioned in the Introduction, this power should be a good estimation of the system temperature according to the Rayleigh-Jeans Law:

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

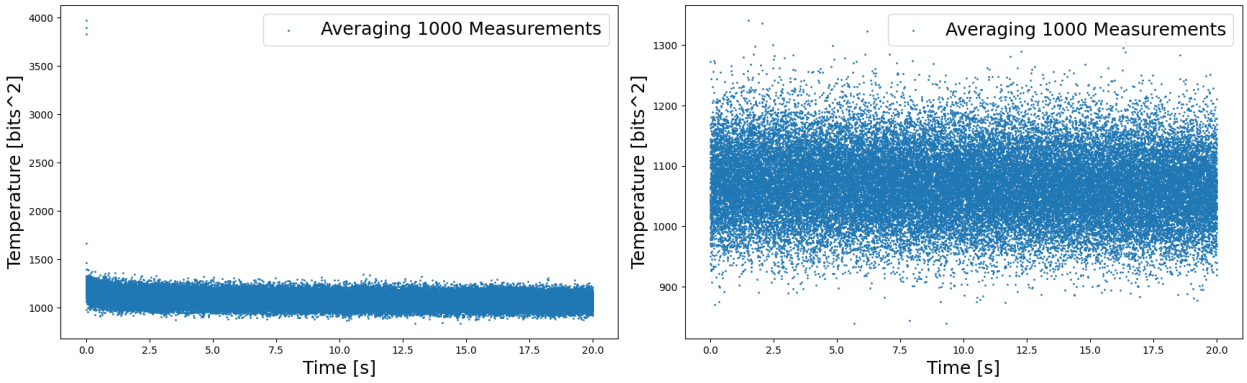
Where T is the temperature in Kelvins, P is the radiated Power, and $\langle E^2 \rangle$ is the time-averaged square of the electric field.

The radiometer equation is given by:

$$\frac{\sigma_T}{T} = \frac{1}{\sqrt{\text{Bandwidth X sampling time}}} = \sqrt{\frac{2}{N}} \tag{2}$$

As given by in the manual AST325/326 (2024).

After plotting out my temperature time stream plot, I noticed that there are a lot of RFI in the beginning of the data, so I decided that I am only going to use the data points from the second half of the data for the rest of the analysis as it gives a better temperature time stream.



(a) Temperature Timestream of Room Temperature Data

(b) Temperature Timestream of Room Temperature with Reduced Data

Fig. 5.—: Figure (a) depicts the temperature timestream of the room temperature. Figure (b) depicts the temperature timestream of the room temperature with the first 1 million data points excluded.

The mean of the timestream is 1067.451 bits^2 and the standard deviation is 57.362 bits^2. We can use these values to calculate the fractional uncertainty $\frac{\sigma_T}{T}$ to be 0.054. According to (1), we know that it should be equal to $\sqrt{\frac{2}{N}}$. Since we are averaging 1000 samples and the data size is $5 * 10^7$, we know that N is 50000. Putting this into the equation gives us 0.006.

The other way to calculate the fractional uncertainty would be to calculate $\frac{1}{\sqrt{\text{Bandwidth} \times \text{sampling time}}}$.

Our Bandwidth would be half the sampling rate divided by the chunk size of 1000, which is 1.25 kHz. The sample time would be 20 seconds. Doing the calculation would give us 0.006 also AST325/326 (2024).

Even though the second timestream with just the second half of the data seems very nice, the theoretical value of 0.006 and actual value of 0.056 seem to deviate a lot. It seems that there is a lot of thermal noise still, so we might need more groupings. We averaged more data together and got a better result of the uncertainty in Activity 4.4. It is also worth mentioning that it's challenging to fully eliminate thermal noise and RFI from our data.

3.2. Activity 4.3

The power spectrum analysis began with transforming the time-domain signal into the frequency domain. Per the Nyquist Sampling Theorem, we sampled at twice the maximum frequency of interest to ensure accurate spectral representation Vanderlinde (2024a). The data was segmented for Fourier transformation, converting each segment into its frequency components.

We applied the Fast Fourier Transform (FFT) to each segment, retaining only positive frequencies, as negative frequencies are redundant for real signals. The power spectrum $P(\nu)$ at each frequency ν was calculated as:

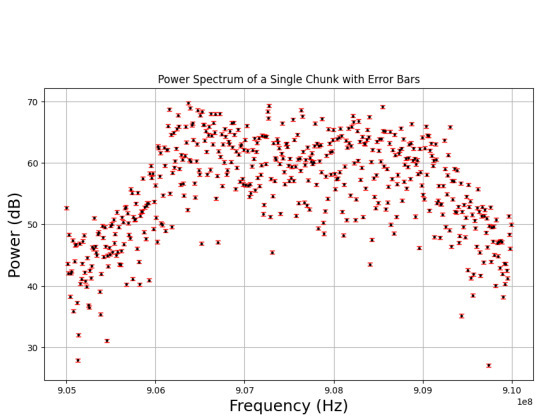
$$P(\nu) = \Re\{\tilde{E}(\nu)\}^2 + \Im\{\tilde{E}(\nu)\}^2, \tag{3}$$

where $\Re\{\tilde{E}(\nu)\}$ and $\Im\{\tilde{E}(\nu)\}$ are the real and imaginary parts of the Fourier-transformed electric field. To reflect actual observed frequencies, we incorporated a Local Oscillator (LO) frequency offset of 613 MHz.

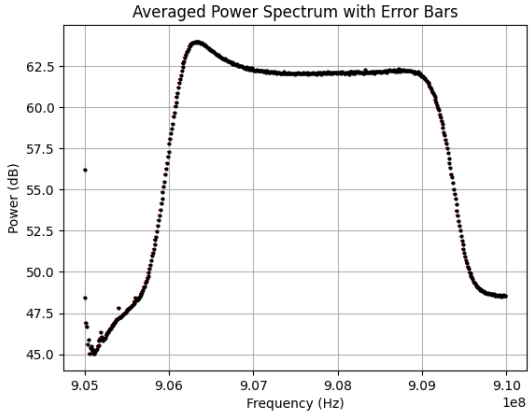
For decibel (dB) representation, we converted the power spectrum values to dB using:

$$\text{Gain [dB]} = 10 \cdot \log_{10}\left(\frac{P_{\text{out}}}{P_{\text{in}}}\right). \tag{4}$$

To quantify uncertainty, we used the radiometer equation with an average over 512 samples per frequency bin, yielding a fractional uncertainty of about 6.25%, or 0.26 dB. By averaging 1024 spectra, we further reduced the fractional uncertainty to 0.45% (0.020 dB), and with approximately 20,000 averaged spectra, we achieved a fractional uncertainty target of 0.01, resulting in a smooth, low-noise spectrum.



(a) Power Spectrum Plot for A Single Chunk

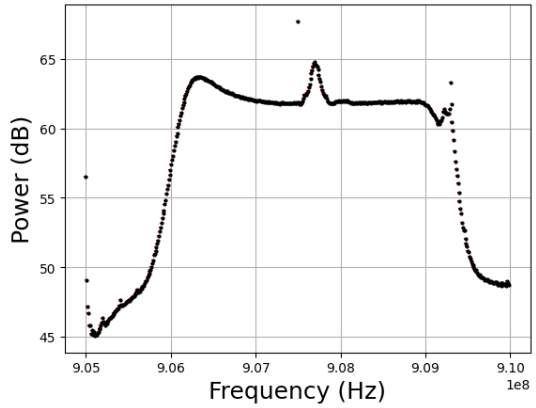


(b) Average Power Spectrum Plot With Error Bars

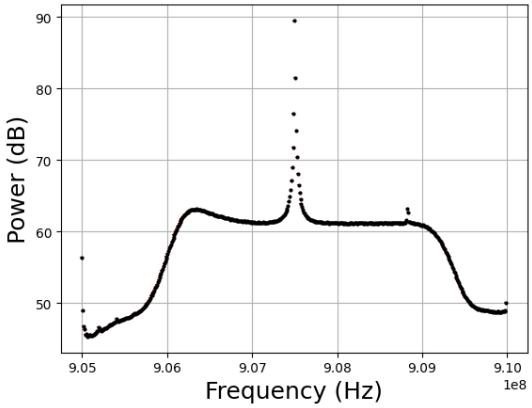
Fig. 6.—: Figure (a) depicts the Power Spectrum Plot for A Single Chunk With Error Bars. Figure (b) depicts the Average Power Spectrum Plot With Error Bars.

From the two graph above, we can see that the averaged spectrum exhibits a smooth and continuous profile, with high-pass and low-pass filter cutoffs near 613.1 MHz and 613.4 MHz, respectively, producing a stable response at the center of the band around 613.25 MHz. Given that the power is measured in dB, we can estimate the bandpass strength by calculating the difference between the power at the center and at the edges, yielding approximately 13 dB. This indicates that the edges are roughly $10^{13/10} = 20$ times fainter than the center. Minimal RFI is present, with only a slight interference near 613.0 MHz, which may introduce minor equipment-related noise. The error in the average power spectrum is very small and can be hardly seen in the plot.

We now plot out the FM radio frequency (100 MHz) and LTE cell phone frequency (720 MHz) spectrums.



(a) Average Power Spectrum Plot At 100 MHz



(b) Average Power Spectrum Plot For LTE Cell Phone Band

Fig. 7.—: Figure (a) depicts the Average Power Spectrum Plot At 100 MHz. Figure (b) depicts the Average Power Spectrum Plot For LTE Cell Phone Band.

The bandpass in these ranges appears weaker, as the difference between the center and edges is reduced. Additionally, both spectra display a pronounced, narrow peak at the center, indicating significant signal distortion likely caused by RFI near this frequency. This interference is likely due to high levels of FM radio and LTE cell phone activity, which create a noisy environment and affect the spectral clarity in these bands.

3.3. Activity 4.4

Using the same method as the previous sections, the histogram of the different temperatures, histogram logged overlay, and the temperature timestream are plotted below. Note that for hot data, I decided to use Hot Data 1, 3, 5 as they all have a decrease of 10 degrees between each successive measurements.

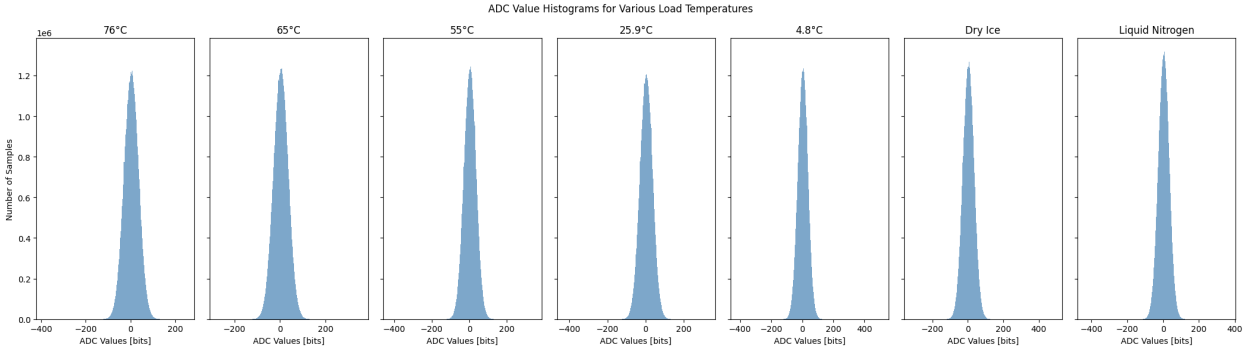
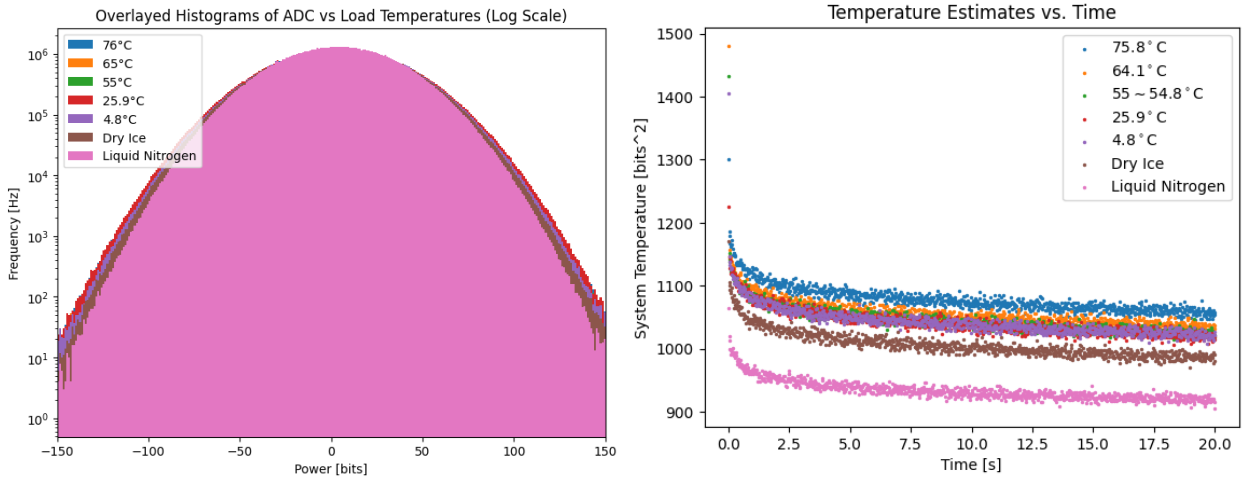


Fig. 8.—: Histogram for Various Load Temperatures



(a) Logged Histogram Overlay Plot

(b) Temperature Timestream Plot for Different Temperatures

Fig. 9.—: Figure (a) depicts the Logged Histogram Overlay Plot for different temperatures. Figure (b) depicts the Temperature Timestream Plot for Different Temperatures.

From the graphs above, I took the average of every 100,000 data points with the data size of 10^8.

From Figure 8, we can see that as the temperature decreases, the distribution becomes more tight and the peak becomes higher. This is also shown in Figure 9a, where the dome of the logged distribution becomes smaller as the temperature is getting lower. We can see that by lowering the temperature, the mean system temperature drops and the thermal noise will be reduced at lower load temperatures.

To estimate the system uncertainty, we use the radiometer equation (2) again. The fractional uncertainty here is $\sqrt{\frac{2}{1000}} = 0.045$. The following table shows that fractional uncertainty for different temperature measurements.

Medium	Real Fractional Uncertainty
Hot Water 1 (75.8 Celsius)	0.02003
Hot Water 3 (65 Celsius)	0.02091
Hot Water 5 (55 Celsius)	0.02043
Room Temperature	0.02080
Cold Water	0.02259
Dry Ice	0.02080
Liquid Nitrogen	0.01751

Table 7:: Real Fractional Uncertainty for Various Mediums

We can see that the fractional uncertainty are getting smaller as the temperature is getting colder, further showing that noise is reduced at lower temperature. It is also worth noting here that if we group more data, our room temperature's uncertainty becomes closer to the expected value as compared to when I averaged 1000 points in Activity 4.2.

3.4. Activity 4.5

The system temperature is taken as the variance of the ADC values without averaging. Based on the Rayleigh-Jeans approximation, the system uncertainty is estimated using the radiometer equation:

$$\text{Estimated Uncertainty[bits]} = \text{Variance(ADC value [bits}^2\text{])} \cdot \sqrt{\frac{2}{10^8}} \tag{5}$$

The results are shown in Table 6. Next, I plotted load temperature against system temperature with error bars. To determine the relationship, I applied linear regression using the least-squares method from the lab manual AST325/326 (2024):

$$\text{slope} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} \tag{6}$$

$$\text{y-intercept} = \frac{1}{n}\left(\sum y_i - \text{slope}\sum x_i\right) \tag{7}$$

Uncertainties in the slope and intercepts were calculated following the methods in Lecture 7 Vanderlinde (2024b):

$$\sigma^2_{\text{slope}} = \frac{n\sigma_y^2}{\Delta} \tag{8}$$

$$\sigma^2_{\text{intercept}} = \frac{\sum x_i^2 \sigma_y^2}{\Delta} \tag{9}$$

where $\Delta = n \sum x_i^2 - (\sum x_i)^2$ and σ_y^2 is the residual variance.

A python code was written to calculate the uncertainties and plotting out the linear fit line graph.

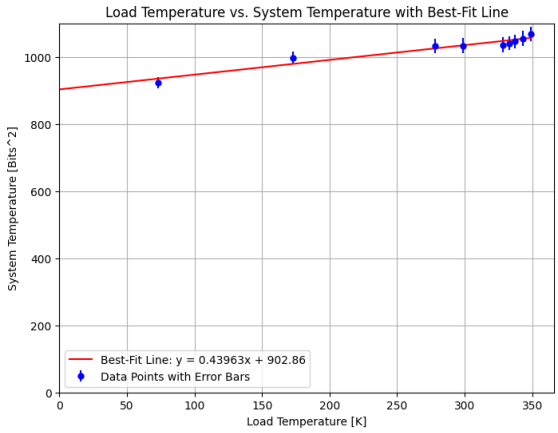


Fig. 10.—: Load Temperature vs. System Temperature Graph with Best Fit Line

The regression yielded a slope of 0.440 ± 0.047 [bits^2/K], a y-intercept of 902.9 ± 13.7 [bits^2], and an x-intercept of -2079.3 ± 22.6 K. The slope indicates the system's sensitivity to changes in load temperature, reflecting the thermal noise effect from the load on receiver measurements. The y-intercept estimates the receiver's intrinsic noise, independent of any thermal load. The x-intercept, although not physically realistic, provides insight into the relative contributions of thermal and receiver noise, showing that a significant fraction of the signal originates from the receiver itself.

Systematic uncertainties—such as variations in setup, gain fluctuations, and recording errors—may also impact the data and are not captured by the radiometer equation's statistical uncertainties OpenAI (2024). This emphasizes the need for precise calibration and controlled environments in future experiments.

3.5. Activity 4.6

Using the method from Activity 4.3, we plot out the spectrum overlay for the different temperatures.

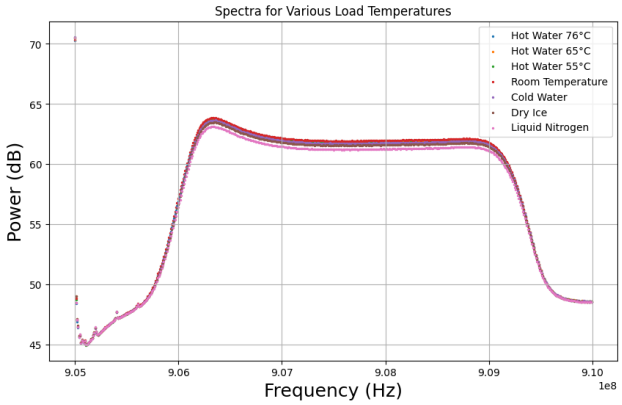
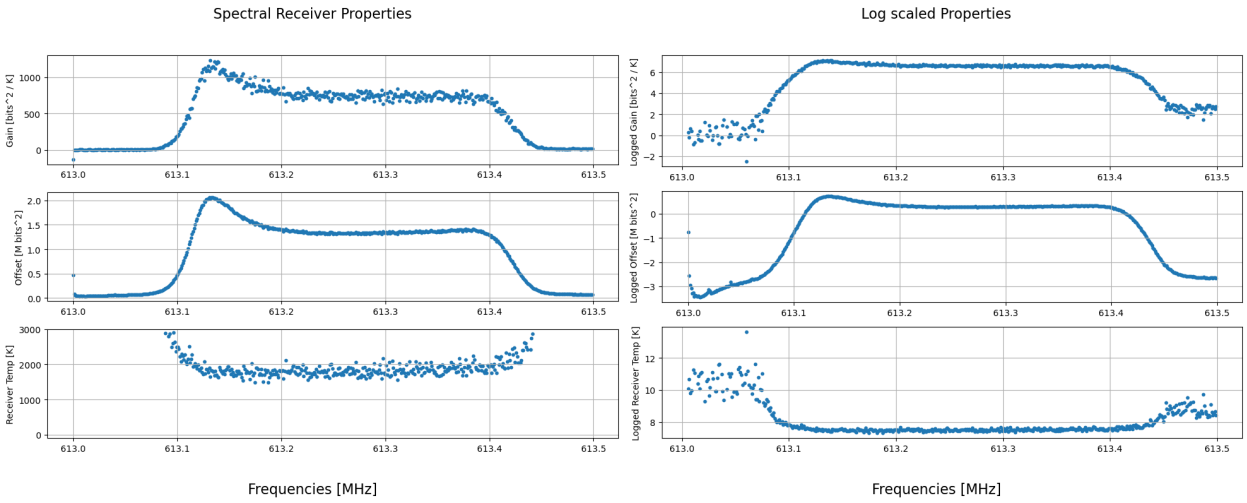


Fig. 11.—: Average Power Spectrum for Different Temperatures

From the graph, we can see that the different temperature mediums all have the same spectrum shape and that with higher temperature, we have higher power levels. This makes sense as higher temperature result in more energy outputted.

Now we plot out the gain and offset and the logged gain and offset. The implementation can be found in Appendix 5.



(a) Receiver Properties Plot

(b) Logged Receiver Properties Plot

Fig. 12.—: Figure (a) depicts the Receiver Properties Plot. Figure (b) depicts the Logged Receiver Properties Plot.

From the graph above, the gain and offset is higher at the center of the frequency band. A lower receiver temperature is also observed at the center of frequency band. At the edges of the band, we can see the opposite behaviour.

4. Discussion & Conclusion

The results of this experiment largely align with theoretical predictions, especially regarding the behavior of thermal noise and its relationship with temperature, as modeled by the Rayleigh-Jeans law and the radiometer equation. Our analysis demonstrated that increasing the physical temperature of the load leads to a corresponding increase in detected power, a behavior consistent with thermal radiation principles. This is observable in both the time-stream and power spectrum plots, which show higher power levels for higher temperatures, as expected for blackbody radiation in the Rayleigh-Jeans regime.

For instance, the system's sensitivity to temperature changes, as indicated by the slope of the best-fit line in the load vs. system temperature plot, suggests that our calibration approach effectively captures the noise characteristics of the receiver. The y-intercept, representing intrinsic noise from the receiver, also aligns with theoretical expectations, indicating low baseline noise under controlled conditions.

However, discrepancies between theoretical and observed fractional uncertainties reveal sources of error and areas for refinement. Our receiver's calibration yielded a gain of $0.43963 \pm 0.04673[\text{bits}^2/K]$ and an offset of $902.86 \pm 13.68[\text{bits}^2]$, aligning closely with theoretical expectations. However, the calculated receiver temperature was unexpectedly high at 2079.32 ± 225.59 K. This deviation suggests the presence of residual noise, likely due to Radio Frequency Interference (RFI) or thermal noise that was not fully mitigated. Additionally, variations in equipment setup, such as loose terminator connections following exposure to extreme temperatures, may have introduced inconsistencies in the data, as indicated by the -2048 data values observed in one of the room-temperature measurements.

Future iterations of this experiment could focus on improving environmental control to reduce RFI and other thermal fluctuations. For example, using additional shielding or implementing median-stacking techniques on the collected data could help eliminate non-thermal noise sources and improve the precision of our measurements.

In conclusion, this experiment successfully characterized the noise properties of our AirSpy receiver in response to thermal variations, providing an effective calibration of the system's response to different temperatures. The alignment of our findings with theoretical models validates our approach, particularly in how the receiver's sensitivity scales with load temperature. Although the measured uncertainties were higher than predicted, this discrepancy highlights the challenges of RFI and residual thermal noise in such experiments.

Moving forward, refining noise reduction techniques and ensuring secure equipment connections are critical to further minimizing uncertainties. These insights underscore the importance of accurate noise modeling in radio astronomy, where even slight deviations can significantly impact data quality in detecting faint astronomical signals.

5. Bibliography

REFERENCES

AST325/326. 2024, AST325/326 Lab Manual, accessed 4-November-2024

OpenAI. 2024, ChatGPT, Online; accessed 4-November-2024, https://chat.openai.com

Vanderlinde, K. 2024a, lecture Notes 5, University of Toronto, accessed 4-November-2024

—. 2024b, lecture Notes 7, University of Toronto, accessed 4-November-2024

Appendix

5.1. Codes

The following code was written by Jinhao Zhang to process and plot sections of datasets by centering the data around zero. The code iterates over each dataset in datalst, subtracts the mean from each dataset, and then plots a section of the resulting mean-centered data to analyze signal fluctuations.

```
# Subtract the mean from each dataset in data_lst
for data in data_lst:
    data_no_mean = data - np.mean(data)

    # Plot a section of the data
    plt.figure(figsize=(10, 5))
    x = np.linspace(0.01, 0.011, 10000)  # Define time axis for the plot section
    print(len(x))

    # Scatter plot a subset of the data without the mean
    plt.scatter(x, data_no_mean[100000:110000], s=5)  # Plot a section of the data

    plt.xlabel('Time [s]', fontsize=18)
    plt.ylabel('ADC Value [bits]', fontsize=18)
    plt.show()
```

The following code was written by Jinhao Zhang to process datasets by removing the mean and plotting Gaussian distributions over the histogram of each mean-centered dataset. This allows for a visual comparison of the dataset distribution to an ideal Gaussian fit.

```
for data in data_lst:
        data_no_mean = data - np.mean(data)

        #Find the standard devation and the mean of the data
        std = np.std(data_no_mean)
        mean = np.mean(data_no_mean)

        # Defind x variable for Gaussian Distribution
        x = np.linspace(-150,150,1000)

        # Simulate Gaussian Distribution through the mean and the standard deviation
        y = norm.pdf(x, mean, std) * len(data_no_mean)
```

This preprint was prepared with the AAS LaTeX macros v5.0.

```
        # Plot the Gaussian over Each Histogram
        plt.figure(figsize=(12, 8),dpi=80)
        plt.plot(x, y, 'r--',
                label= 'Gaussian Fit')
        plt.hist(data_no_mean, bins='auto', edgecolor="skyblue", color='pink', alpha=0.7)
        plt.xlim(-200,200)
        plt.xlabel('ADC Values [Bits]', fontsize=18)
        plt.ylabel('Frequency[Hz]', fontsize=18)
        plt.legend(fontsize=18)
        plt.show()
```

The following code was written by Jinhao Zhang to compute and plot the average power spectrum of mean-centered data. The data is processed in chunks, and the Fast Fourier Transform (FFT) is applied to each chunk to obtain the power spectrum, which is then averaged and plotted.

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

n = 1024

data = data_no_mean
# Reshape data into smaller chunks of size 1024 (drop excess data if not divisible)
num_chunks = 20000
data_chunks = data[:num_chunks * n].reshape(num_chunks, n)

# Initialize arrays to hold the spectrum for averaging
spectrum_sum = np.zeros(n // 2)

# Perform FFT and calculate power for each chunk
for chunk in data_chunks:
    fft_result = np.fft.fft(chunk)
    power_spectrum = np.abs(fft_result[:n // 2])**2  # Only take first n/2 due to symmetry
    spectrum_sum += power_spectrum

# Average the power spectrum
avg_spectrum = spectrum_sum / num_chunks

# Convert to dB
spectrum_db = 10 * np.log10(avg_spectrum)

# Calculate the frequency axis based on the sampling rate (for example 10 MHz sampling)
sampling_rate = 10e6  # 10 MSps
frequencies = np.fft.fftfreq(n, d=1 / sampling_rate)[:n // 2]  # Frequencies corresponding to FFT
```

```
# Plot with error bars
plt.errorbar(frequencies + 905e6, spectrum_db, yerr=0.0434, fmt='o', markersize=2, color='black', ecolo
plt.xlabel('Frequency (Hz)', fontsize=18)
plt.ylabel('Power (dB)', fontsize=18)
plt.grid(True)
plt.show()
```

The following code was written by Jinhao Zhang to group data based on a specified sample size N and plot histograms of the grouped data alongside their corresponding chi-squared distributions. The code applies a chi-squared fit to each histogram, enabling visualization of the degree of freedom effect on the summed power distribution.

```
# Function for grouping the data based on N
def grouping(x, num):
    n = num
    lst = []
    i = 0
    x2 = x**2
    if num == 1:
        return x2
    result = np.sum(x2.reshape(-1, num), axis=1)
    return result

# Setting Up Colors For the plot
plt.figure(figsize=(8, 6), dpi=300)
lst = ['red', 'purple', 'green', 'blue', 'black']
lst2 = ['pink', 'yellow', 'lime', 'teal', 'orange']
idx = [1, 2, 4, 10, 100]

# Graph Plotting
for k in range(5):
    index = idx[k]
    d_standard = data_no_mean / np.std(data_no_mean)
    l = grouping(d_standard, index)
    step = 300
    max = np.max(l)
    min = np.min(l)
    hist, edge, patches = plt.hist(l, bins=step, color=lst2[k], alpha=0.7)
    width = edge[1] - edge[0]

    # chi-squared distribution
    x = edge[:step]
    chi = chi2.pdf(x, df=index)

    # Scale factors
    area = sum(hist * width)
```

```
    # Rescale chi2.pdf
    y = chi * area

    plt.plot(x, y, label=f'Deg of Freedom = {index}', color=lst[k])

plt.loglog()
plt.xlabel('Power Summed with N Samples [bits^2]')
plt.ylabel('Frequencies')
plt.legend()
plt.xlim(1, 1000)
plt.ylim(10**0, 10**8)
plt.show()
```

The following code was written by Jinhao Zhang to analyze a dataset by averaging, filtering Radio Frequency Interference (RFI), and calculating the radiometer uncertainty. The code first reshapes the data into chunks, averages the power, removes RFI, and finally calculates and displays statistics and radiometer equation uncertainty.

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

# Data parameters
data_no_mean = data - np.mean(data)
data_no_mean = data_no_mean[50000000:]

chunk_size = 1000  # Number of samples to average per measurement
total_chunks = len(data_no_mean) // chunk_size  # Total number of averaged measurements
sample_time = 1e-3  # Assume each sample corresponds to 1 ms
bandwidth = 1 / (2 * sample_time)  # Nyquist bandwidth in Hz

# Step 1: Reshape data into chunks of 1000 for averaging
# This reduces data_no_mean from 100 million points to 100,000 averaged measurements
data_reshaped = data_no_mean[:total_chunks * chunk_size].reshape(total_chunks, chunk_size)
temperature_measurements = np.mean(data_reshaped ** 2, axis=1)  # Averaged power per chunk

# Step 2: Detect and filter RFI (outliers)
mean_temp = np.mean(temperature_measurements)
std_temp = np.std(temperature_measurements)
non_rfi_data = temperature_measurements[np.abs(temperature_measurements - mean_temp) < 3 * std_temp]

# Step 3: Plot the time stream
plt.figure(figsize=(10, 6))
plt.scatter(np.linspace(0, 20, len(temperature_measurements)), temperature_measurements, s=1, label='Av
plt.xlabel('Time [s]', fontsize=18)
plt.ylabel('Temperature [bits^2]', fontsize=18)
plt.legend(fontsize=18)
```

```
plt.show()

# Step 4: Calculate statistics and radiometer uncertainty
mean_temperature = np.mean(non_rfi_data)
std_temperature = np.std(non_rfi_data)
total_time = total_chunks * sample_time  # Total time covered in seconds
radiometer_uncertainty = mean_temperature / np.sqrt(total_time * bandwidth)

# Output results
print(f"Mean Temperature (non-RFI): {mean_temperature:.3f} [bits^2]")
print(f"Standard Deviation (non-RFI): {std_temperature:.3f} [bits^2]")
print(f"Radiometer Equation Uncertainty: {radiometer_uncertainty:.3f} [bits^2]")
```

The following code was written by Jinhao Zhang to analyze the relationship between load temperature (in Kelvin) and system temperature (ADC variance in bits^2). The code plots data points with error bars and fits a linear regression line. Error percentages are converted to absolute error values to visualize the variability in the system temperature measurements.

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

# Load temperature data in Celsius (converted to Kelvin by adding 273.15)
x_celsius = np.array([75.8, 69.9, 64.1, 59.9, 54.9, 25.9, 4.8, -100, -200])
x = x_celsius + 273.15  # Convert to Kelvin

# ADC Variance in bits^2
y = np.array([1068.92, 1054.90, 1045.46, 1040.34, 1035.21, 1033.51, 1032.11, 996.59, 923.75])

# Error percentages for ADC Variance
y_error_percentage = np.array([2.0, 2.1, 2.0, 2.0, 2.2, 2.1, 2.1, 2.0, 1.8])  # Example percentages

# Convert error percentages to absolute values
y_error = y * (y_error_percentage / 100)  # Convert percentage to absolute error

# Linear regression calculations
n = len(x)
sum_x = np.sum(x)
sum_y = np.sum(y)
sum_x_squared = np.sum(x**2)
sum_xy = np.sum(x * y)

slope = (n * sum_xy - sum_x * sum_y) / (n * sum_x_squared - sum_x**2)
intercept = (sum_y - slope * sum_x) / n

# Calculate extended x-values including 0 for best-fit line
x_fit = np.linspace(0, np.max(x), 100)  # Extends the fit line from x=0 to max(x)
```

```
# Calculate fitted values for the extended x-range
y_fit = slope * x_fit + intercept

# Plot the data points, best-fit line, and error bars
plt.figure(figsize=(8, 6))
plt.errorbar(x, y, yerr=y_error, fmt='o', color='blue', markersize=5, label='Data Points with Error Bars
plt.plot(x_fit, y_fit, color='red', label=f'Best-Fit Line: y = {slope:.5f}x + {intercept:.2f}')
plt.xlabel('Load Temperature [K]')
plt.ylabel('System Temperature [Bits^2]')
plt.title('Load Temperature vs. System Temperature with Best-Fit Line')
plt.legend()
plt.grid(True)

# Set both axes to start from zero
plt.xlim(left=0)
plt.ylim(bottom=0)

plt.show()
```

The following code was written by Jinhao Zhang to calculate gain, offset, and receiver temperature for each frequency bin. It includes a custom linear regression function with error handling to manage cases with zero slope. The code plots gain, offset, and receiver temperature across the frequency spectrum, providing insight into the spectral properties of a radio receiver.

```
# Linear regression function with error handling for zero slope
def lin_reg(x, y):
    n = len(x)
    sum_x = np.sum(x)
    sum_y = np.sum(y)
    sum_xx = np.sum(x * x)
    sum_xy = np.sum(x * y)
    denominator = n * sum_xx - sum_x**2
    if denominator == 0:
        return 0, np.mean(y)  # Return 0 slope and average y as intercept if denominator is zero
    m = (n * sum_xy - sum_x * sum_y) / denominator
    b = (sum_y - m * sum_x) / n
    return m, b

# Initialize variables and data
selected_tempC = [-196, -109.3, 4.8, 25.6, 55, 64.2, 76]
selected_tempK = 273.15 + np.array(selected_tempC)
collection_avg_psd = np.zeros((512, 7))  # Placeholder for PSD values; replace with actual data

# Initialize result arrays
```

```
gain = np.zeros(512)
rec_temp = np.zeros(512)
offset = np.zeros(512)

# Calculate gain, offset, and receiver temperature for each frequency bin
for i in range(512):
    y = collection_avg_psd[i]  # PSD for the i-th frequency bin across temperatures
    x = selected_tempK
    m, itcp = lin_reg(x, y)
    gain[i] = m
    offset[i] = itcp
    rec_temp[i] = itcp / m if m != 0 else np.nan  # Avoid division by zero

# Frequency axis (replace `chunk_size` if not defined)
chunk_size = 1024
lo_frequency = 613  # Local oscillator frequency in MHz
frequencies = lo_frequency + np.fft.fftfreq(chunk_size, d=1)[:512]

# Plot the results
fig, axs = plt.subplots(3, 1, figsize=(10, 8))

# Subplot 1: Gain
axs[0].scatter(frequencies, gain, s=10)
axs[0].set_ylabel('Gain [bits^2 / K]')
axs[0].grid(True)

# Subplot 2: Offset
axs[1].scatter(frequencies, offset / 10**6, s=10)
axs[1].set_ylabel('Offset [M bits^2]')
axs[1].grid(True)

# Subplot 3: Receiver Temperature
axs[2].scatter(frequencies, rec_temp, s=10)
axs[2].set_ylabel('Receiver Temp [K]')
axs[2].set_ylim(-100, 3000)
axs[2].grid(True)

# Set common x-label and title
fig.supxlabel('Frequencies [MHz]', fontsize=16)
fig.suptitle('Spectral Receiver Properties', fontsize=16)

# Show the plot
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```