Johnson Noise

Paul Lea and Alex Weiss Section 1

Objective:

- This investigation is intedned to determine the Boltzmann constant by measuring the fluctuations of electrons within a resistor. These fluctations will be amplified by a series of Op-Amps to increase the small initial voltage fluctuations to a readable level. This measurement will be dependent on temprature and resistance of the resistor. Additionally, the frequencies of the Johnson noise must be selected from the wider spectrum of noise using a band-pass filter. The filter and amplifier were constructed in sections 1 and 2.

Section 3.1

This section measured the response characteristics of the band-pass filter, calibrating it such that the gain integral would be usable in the upcoming Johnson Noise measurements.

Procedure

- We began by selecting several frequencies to test, in the range from 1-100 kHz.
- For each one of these frequency values, we measured the input and output voltage using the oscilloscope.
- After obtaining these voltage measurements, we were able to calculate the system gain and therefore obtained the correct calibration metrics for the Johnson noise

Data:

In [3]:
```
import matplotlib.pyplot as plt
from scipy.odr import Model, RealData, ODR
import scipy.integrate as itg
import numpy as np
import pandas as pd
import math
xs = 1000,3000,5000,25000,50000,100000 #Hz
xs = np.ravel(xs)
ys = 0.046, 0.092, 0.110, 0.051, 0.014, 0.004
```

Experimental Data: (figure 1)

Pk-Pk Voltage	1kHz	3kHz	5kHz	25kHz	50kHz	100kHz
Channel 1 (output)	$460mV$	$940mV$	$1.12V$	$512mV$	$144mV$	$44mV$
Channel 2 (inpiut)	$10.0V$	$10.2V$	$10.2V$	$10.0V$	$10.0V$	$10.0V$

Calculated Gain: (figure 2)

	1kHz	3kHz	5kHz	25kHz	50kHz	100kHz
Calculated gain	0.046	0.092	0.110	0.051	0.014	0.004

Analysis

Theoretical Gain Calculations:

High Pass Filter:

A high pass filter is designed to let higher frequencies pass through without being attenuated, while attenuating lower frequencies. The formula for the cutoff frequency on a high pass filter is as follows:

$$f_c = \frac{1}{2\pi\tau} = \frac{1}{2\pi rc}$$

We wanted a high-pass cutoff of around 5 kHz, so setting f_c to 5 kHz we can solve for required values of r while using a $0.1\mu F$ capacitor

$$5000(2\pi)(0.1 \times 10^{-6}) = \frac{1}{R}$$

$$R_{\text{theoretical}} = 318.31\ \Omega$$

Low Pass Filter

A low pass filter is the opposite of a high pass filter, where it allows the lower frequencies to pass through while attenuating higher frequencies out. The cutoff frequency is given by the same equation:

$$f_c = \frac{1}{2\pi\tau} = \frac{1}{2\pi rc}$$

We wanted a low-pass cutoff of around 20 kHz, and we used the same $0.1\mu F$ capacitor. Solving for required resistance:

$$20000(2\pi)(0.1 \times 10^{-6}) = \frac{1}{R}$$

$$R_{\text{theoretical}} = 79.5\ \Omega$$

Actual Resistances and capacitance's (figure 3)

Actual Values	R	C
Low Pass Filter	$50.9 \pm 0.5\Omega$	$0.099 \pm 0.002\mu F$
High Pass Filter	$327.1 \pm 30.0\Omega$	$0.103 \pm 0.002\mu F$

Theoretical Cutoff Values using measured components:

Low pass:

$$f_c$$

$$\Delta f_c = \sqrt{\left(\frac{\partial f_c}{\partial R}\Delta R\right)^2 + \left(\frac{\partial f_c}{\partial C}\Delta C\right)^2}$$

$= 31584.0017248904$ $= \sqrt{\left(\left(-(C*2/(2*pi*R*C)^2*pi)\right)\cdot\Delta R\right)^2 + \left(\left(-(pi*2/(2*pi*R*$

$= 709.4926405004$

$= 7.094926405 \times 10^2$

$= 7 \times 10^2$

$$f_c = (3.16 \pm 0.07) \times 10^4$$

High pass:

$$f_c$$

$$\Delta f_c = \sqrt{\left(\frac{\partial f_c}{\partial R}\Delta R\right)^2 + \left(\frac{\partial f_c}{\partial C}\Delta C\right)^2}$$

$= 4723.9181358955$ $= \sqrt{\left(\left(-(C*2/(2*pi*R*C)^2*pi)\right)\cdot\Delta R\right)^2 + \left(\left(-(pi*2/(2*pi*R*($

$= 442.8580100016$

$= 4.4285801 \times 10^2$

$= 4 \times 10^2$

$$f_c = (4.7 \pm 0.4) \times 10^3$$

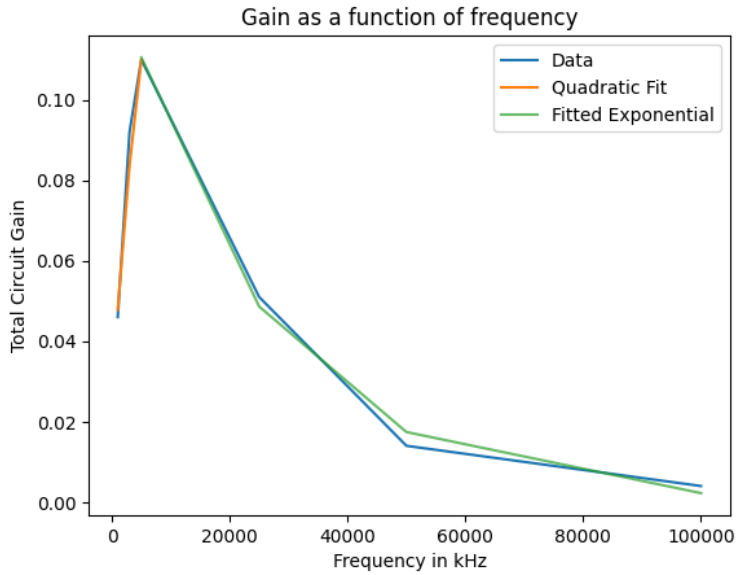
Theoretical Gain Curve:

$$g \equiv \frac{V_{out}}{V_{in}} = \frac{1}{\sqrt{1+(\omega R_1 C)^2}} \frac{\omega R_2 C}{\sqrt{1+(\omega R_2 C)^2}}$$

In [4]:

```
def exp(params, x):
    return np.multiply(params[0], np.exp(params[1]*x))

def quadShift(params, x):
    return params[0] * (x+params[3]) ** 2 + params[1] * (x + params[3]) + params [2]

#Fitting curve to second half of curve:
model = Model(exp)
data = RealData(xs[2:], ys[2:])
params = 5, -0.00003
odr = ODR(data, model, beta0 = params)
odr_result = odr.run()
params_fit = odr_result.beta
covar_fit = odr_result.sd_beta
chisquared = odr_result.res_var
V_fit = exp(params_fit, xs)
expParams = params_fit

#Fitting line to first half of curve
model = Model(quadShift)
data = RealData(xs[:3], ys[:3])
params = -0.0000000012, 0.00001, 0.113, -5300
odr = ODR(data, model, beta0 = params)
odr_result = odr.run()
params_fit = odr_result.beta
covar_fit = odr_result.sd_beta
V_fit1 = quadShift(params_fit, xs)
```

```
quadParams = params_fit

plt.plot(xs, ys, label = "Data")
plt.plot(xs[:3], V_fit1[:3], label = "Quadratic Fit")
plt.plot(xs[2:], V_fit[2:], alpha = 0.7, label = "Fitted Exponential")

plt.ylabel("Total Circuit Gain")
plt.xlabel("Frequency in kHz")
plt.title("Gain as a function of frequency")
plt.gcf().text(0.95,0.5,"Exponential Function: " + str(np.round(expParams[0],4)) + "x *
plt.gcf().text(0.95,0.45,"Quadratic Function: " + str(np.round(quadParams[0],10)) + "(x"
plt.gcf().text(0.95,0.4,"$\\chi$/NDF:  " + str(np.round(chisquared,8)))
plt.legend()
plt.show()
```



Exponential Function: 0.1357x * e ^ (-4.1e-05x)
Quadratic Function: -1.2e-09(x-5300.0)^2 + 0.0(x-5300.0) + 0.113
χ/NDF: 1.037e-05

Figure 4: Duel curve-fit models to generate frequency response curves for band-pass filter circuit.

In [5]:
```
#Defining function that returns gain at any frequency via a piecewise equation in Hz
def gainFunc(freq):
    if freq < 5:
        return quadShift(quadParams, freq)
    else:
        return exp(expParams, freq)
```

Conclusion:

In this section we successfully fit a pair of mathematical models to model the frequency response of our band-pass filter/amplifier circuit. These models qualitatively fit the data well, however, the chi squared parameter tells us that the data model is highly over-constrained. This logically makes sense, we have limited data points and several parameters to fit to. Despite this, we are able to use these models to evaluate the integral that we need for section 3.2.

Section 3.2

After we calibrated the experimental setup's band pass filter and obtained the total op-amp gain, we were ready to take measurements of the generated Johnson noise using a variety of resistors.

Procedure

- We began this portion of the lab by removing the function generator and voltage divider circuit from the experimental setup.
- After these components were removed, a selection of resistors from between 10 Ω and 10 $k\Omega$ were selected to serve as the generators of the Johnson noise.
- At this point the room's air temperature was recorded for usage during the analysis.
- Finally, we recorded the oscilloscope output of the Johnson noise for 6 different resistance values.

Figure 5.1: Experimental layout:

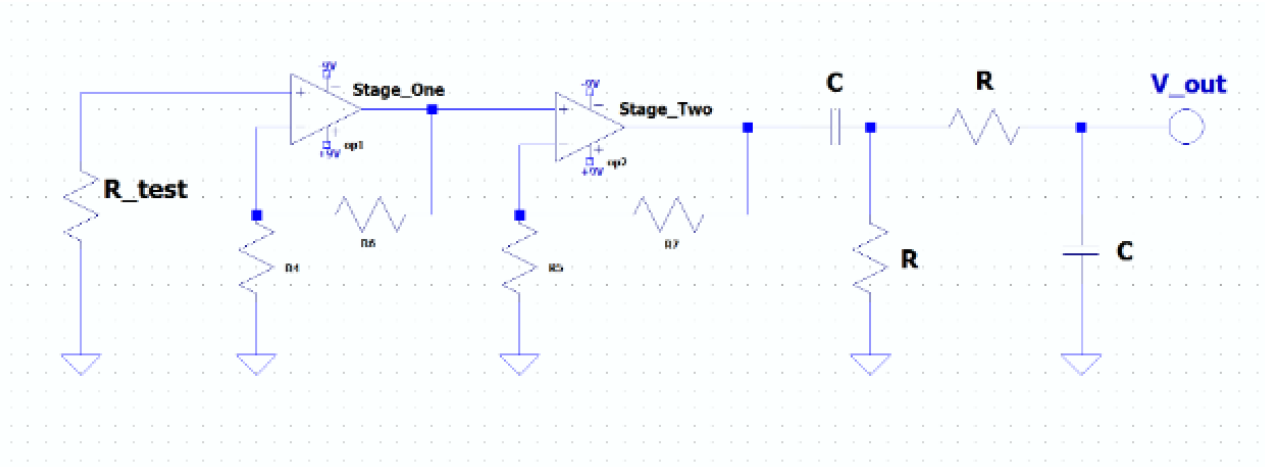
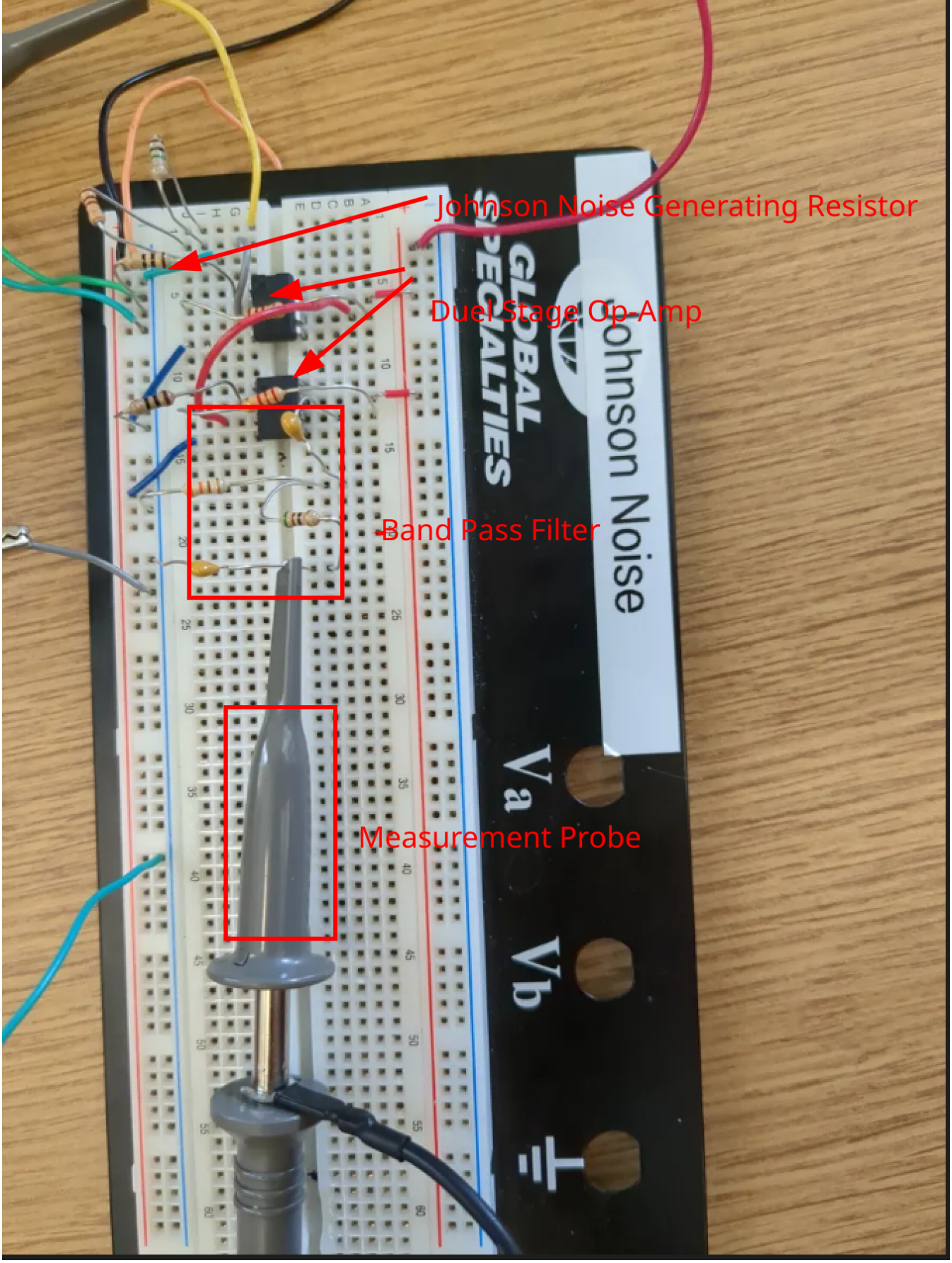


Figure 5.2: Labeled Digital Picture



Data

Room temperature: 26 degrees Celsius Op-Amp gain: 2419.946 x Resistance for each trial: (Figure 5.2)

Trial	R
1	$326.1 \pm 3\ \Omega$

2	50.8 $\pm 0.5\ \Omega$
3	815 $\pm 7\ \Omega$
4	1200 $\pm 10\ \Omega$
5	3851 $\pm 40\ \Omega$
6	9950 $\pm 90\ \Omega$

> labeled diagram here

Analysis:

We evaluate this integral to find the expectation value of $\langle V^2 \rangle$

$$\langle V^2 \rangle = 4kTR \int_0^\infty g^2(v)\, dx$$

For each value of R we will evaluate this integral.

In [6]:
```
integGsqr = itg.quad(gainFunc, 0, +np.inf)
print("Value of integral: " , integGsqr[0])
```

```
Value of integral:  3305.1656354835154
```

In [7]:
```
def read_csv(filename):
    df = pd.read_csv(filename)
    data = np.array(df)
    return data

trial1 = read_csv("Johnson Noise/Section 3.2/Trial 1/F0000CH1.CSV")
trial2 = read_csv("Johnson Noise/Section 3.2/Trial 2/F0001CH1.CSV")
trial3 = read_csv("Johnson Noise/Section 3.2/Trial 3/F0002CH1.CSV")
trial4 = read_csv("Johnson Noise/Section 3.2/Trial 4/F0003CH1.CSV")
trial5 = read_csv("Johnson Noise/Section 3.2/Trial 5/F0004CH1.CSV")
trial6 = read_csv("Johnson Noise/Section 3.2/Trial 6/F0005CH1.CSV")

volt1 = np.average(trial1[:,1] ** 2)
volt2 = np.average(trial2[:,1] ** 2)
volt3 = np.average(trial3[:,1] ** 2)
volt4 = np.average(trial4[:,1] ** 2)
volt5 = np.average(trial5[:,1] ** 2)
volt6 = np.average(trial6[:,1] ** 2)
resistances = 50.8, 326.1,  815, 1200, 3851, 9950
resistances = np.ravel(resistances)
resistUnc = 0.5, 3,  7, 10, 40, 90
volts = volt1,volt2,volt3,volt4,volt5,volt6
volts = np.ravel(volts)
volts = volts/2419.946 # Compensating for Op-Amps
voltUnc = np.multiply(volts, 0.03)
```

In [8]:
```
def boltz(params, R):
    return params[0] * R + params[1]

def constant(param0):
    return param0/(299.15 * 4 * integGsqr[0])

model = Model(boltz)
```

```
data = RealData(resistances, volts, sy=  voltUnc, sx = resistUnc)
params = 1.3E-23, 0
odr = ODR(data, model, beta0 = params)
odr_result = odr.run()
params_fit = odr_result.beta
covar_fit = odr_result.sd_beta
chisquared = odr_result.res_var
V_fit1 = boltz(params_fit, resistances)

plt.errorbar(resistances, volts, xerr = resistUnc, yerr = voltUnc , label = "Data")
plt.plot(resistances, V_fit1, label = "Fitted Line")
plt.gcf().text(0.95,0.5,"Calculated Boltzmann Constant: " + str(np.round(constant(params
plt.gcf().text(0.95,0.45,"$\\chi$/NDF:  " + str(chisquared))
plt.title("Resistances vs induced voltages")
plt.xlabel("Resistance")
plt.ylabel("Voltages")
plt.legend()
plt.show()
```

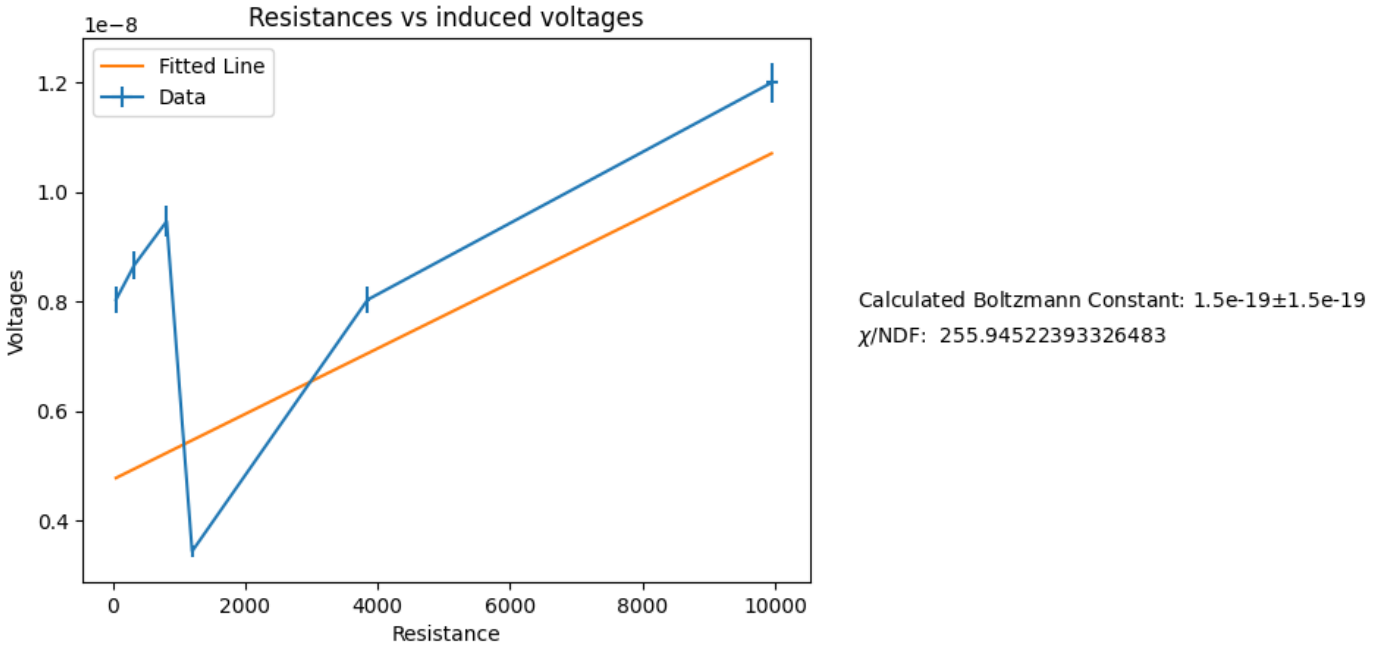
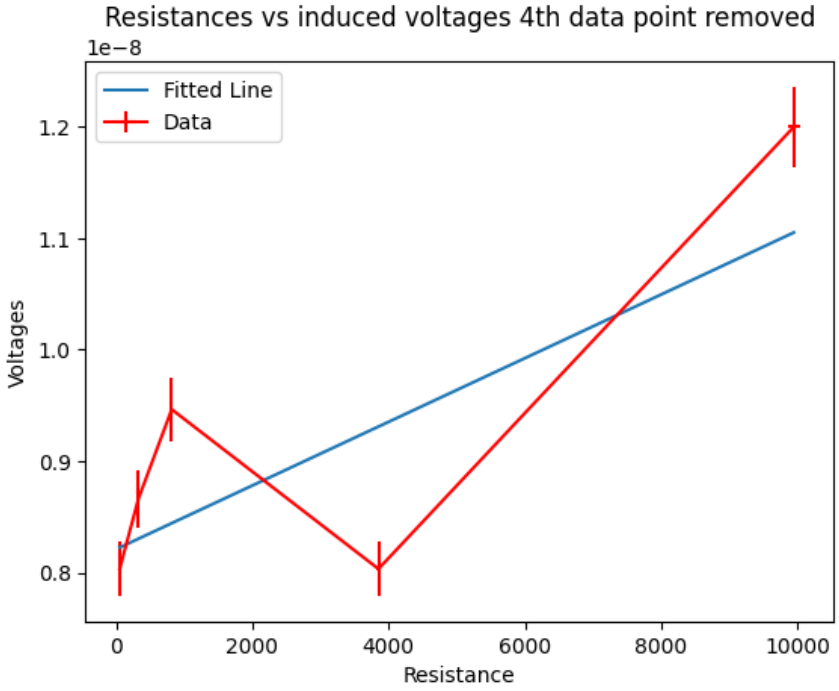


Figure 6: All collected data linearly fit to boltzmann constant model.

In [9]:

```
trimmedResistance = np.append(resistances[:3],  resistances[4:])
trimmedVoltage = np.append(volts[:3],  volts[4:])
trimmedSV = np.append(voltUnc[:3],  voltUnc[4:])
trimmedSR = np.append(resistUnc[:3],  resistUnc[4:])


data = RealData(trimmedResistance, trimmedVoltage, sy= trimmedSV )
params = 1.3E-23, 0
odr = ODR(data, model, beta0 = params)
odr_result = odr.run()
params_fit = odr_result.beta
covar_fit = odr_result.sd_beta
chisquared = odr_result.res_var
V_fit1 = boltz(params_fit, trimmedResistance)
plt.errorbar(trimmedResistance, trimmedVoltage, xerr = trimmedSR, yerr = trimmedSV, labe
plt.plot(trimmedResistance, V_fit1, label = "Fitted Line")
plt.gcf().text(0.95,0.5,"Calculated Boltzmann Constant: " + str(np.round(constant(params
plt.gcf().text(0.95,0.45,"$\\chi$/NDF:  " + str(chisquared))
plt.title("Resistances vs induced voltages 4th data point removed")
plt.xlabel("Resistance")
plt.ylabel("Voltages")
```

```
plt.legend()
plt.show()
```



Calculated Boltzmann Constant: 7e-20±4e-20
χ/NDF: 16.809625442785045

Conclusion:

This section yielded mixed results, our data acquisition had large discrepancies from the model. This is evidenced by the large chi-squared model in figure 6 of 255.9. This signals a very poor correlation to the model. This initial analysis additionally yielded a calculated Boltzmann constant that was incoherent with the literature values of the Boltzmann constant by a factor of approximately 10,000. Possible reasons for these large discrepancies may include temperature fluctuations or errant spikes in background radiation interfering with readings on the oscilloscope. The data's fit was improved by removing the 4th data point, which yielded a Chi-squared value of 16.8, an approximate tenfold decrease over the previous fit. This signaled a much better fit, and the value of the calculated Boltzmann constant was also correspondingly closer to the literature values, but still 3 orders of magnitude away from the correct value. This experiment could be improved by adding more robust temperature monitoring capabilities, as well as EM radiation shielding around the experimental setup in the form of a Faraday cage.

Section 3.3: Johnson Noise- Teachspin Approach

Procedure

- The experiment was set up by following the parameters outlined in the "Teachspin Approach" document. The band pass filter components, gain, multiplier and output were all set to specific values.
- After we had set up the experiment with the appropriate settings, we collected data for different resistance values using a high-precision digital multimeter.
- These data values along with the corresponding resistances were recorded into a table for further analysis. ### Data Pre-amp gain: 600x

Amp Gain: 400x

Total Gain: 240000x

High Pass Corner Freq: 0.3 kHz

Low Pass Corner Freq: 100kHz

Δf : 99.7 kHz | Resistance | Voltage | |-----------|---------| | 1 Ohm | 38.90mV | | 10 Ohms | 39.03mV | | 100 Ohms | 40.05mV | | 1k Ohms | 49.94mV | | 10k Ohms | 149.01mV | | 100k Ohms | 861.1 mV |

Analysis

Measured Voltage follows this formula:

$$V_{out}(t) = G[V_j(t) + V_N(t)]$$

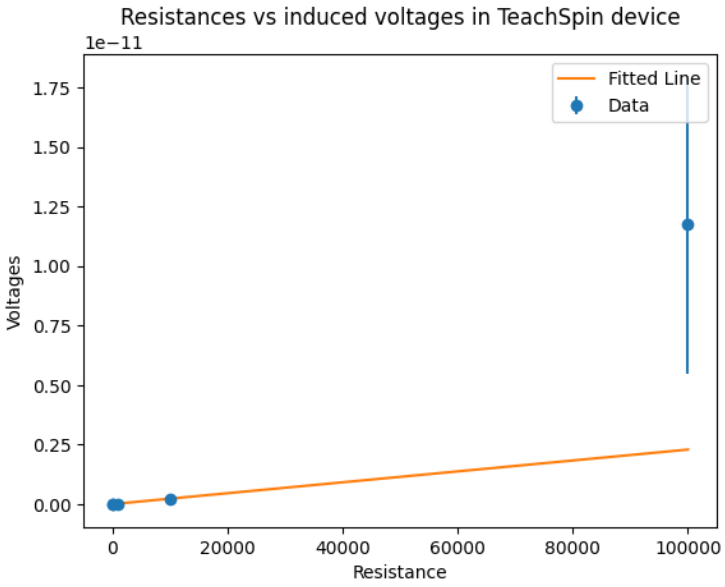
Where V_j is the Johnson noise and V_N represents the intrinsic noise of the system.

Taking the RMS of this system:

$$\langle V_{out}^2(t) \rangle = G^2\{\left\langle V_j^2(t) \right\rangle + \langle V_N^2(t) \rangle\}$$

We can estimate the first measured resistance value to be close to the intrinisc noise value and subtract it from the measurements. After this, we can compensate for the gain in the system to measure the pre-amplified voltage. Then we fit the voltage as a function of resistance measurements.

In [64]:
```
resistances = 1,10,100,1000,10000,100000
voltage = 0.03890, 0.03903, 0.04005, 0.04994, 0.14901, 0.8611
resistances = np.ravel(resistances)
voltage = np.ravel(voltage)
voltage = voltage - 0.03890
voltage = voltage[1:]
voltage = voltage / 240000
voltage = voltage ** 2
resistances = resistances[1:]
def constant(param0):
    return param0/(299.15 * 4 * 99.7)

sy =  0.01, 0.01, 0.01, 0.01, 0.1
sy = np.ravel(sy)
sy = sy /40000
sy = sy **2
data = RealData(resistances, voltage, sy = sy )
params = 1.3E-23, 0
odr = ODR(data, model, beta0 = params)
odr_result = odr.run()
params_fit = odr_result.beta
covar_fit = odr_result.sd_beta
chisquared = odr_result.res_var
V_fit1 = boltz(params_fit, resistances)
boltzC = constant(params_fit[0])
plt.errorbar(resistances, voltage, yerr = sy, label = "Data", linestyle = "", marker = "
plt.plot(resistances, V_fit1, label = "Fitted Line")
plt.gcf().text(0.95,0.5,"Calculated Boltzmann Constant: " + str(np.round(boltzC,25)) + "
plt.gcf().text(0.95,0.45,"$\\chi$/NDF:  " + str(chisquared))
plt.title("Resistances vs induced voltages in TeachSpin device")
plt.xlabel("Resistance")
plt.ylabel("Voltages")
plt.legend()
plt.show()
```



Calculated Boltzmann Constant: 1.9220000000000003e-22±5.5e-23
χ/NDF: 0.7924046207849061

Conclusions:

This section yielded much better values then the previous section. The data fit the linear model much more accurately, leading to the literature-accurate value within the calculated error for the Boltzmann constant. The last data point at 100 $k\Omega$ had a large error due to the multimeter auto-ranging to the volt range as opposed to the milli-volt range before, but the model fits all of the other points very well, with a overall chi-squared value of 0.79, signifying a well-constrained, accurate fit.