university of groningen

BACHELOR THESIS

# Design of Software Routines for Automatic Calibration of Transition Edge Sensors


*Author:*
Callum BLAIR

*Supervisor:*
prof. dr. Jianting Ye


*A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Science*

*in the*


Device Physics of Complex Materials
Zernike Institute for Advanced Materials


July 5, 2019

UNIVERSITY OF GRONINGEN

# *Abstract*

Faculty of Science and Engineering

Zernike Institute for Advanced Materials

Bachelor of Science

**Design of Software Routines for Automatic Calibration of Transition Edge Sensors**

by Callum BLAIR

Software has been developed to automate the analysis of current-voltage curves from superconducting transition edge sensors to be used in the SAFARI spectrometer of the SPICA satellite. This software will allow for faster and potentially more consistent extraction of device parameters compared to the manual process, allowing further analysis to be carried out more quickly. Within the program, the data are combined with the experimental parameters to produce estimations of the critical power. These values are then collected for a range of bath temperatures and fitted to a power law model. The values for the critical temperature $T_c$ and the thermal conductance $G$ are then returned to the user. The software is now fit for purpose and meets the required specifications. However, error estimations and more robust data classification are required before the software can reach its full potential.

# Contents

# Chapter 1

# Introduction

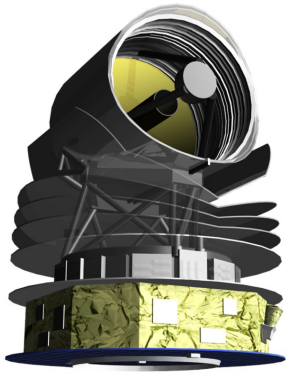


FIGURE 1.1: The future SPICA satellite [1]

The Netherlands Institute for Space Research (SRON) is currently developing an array of sensors for SAFARI, the far-infrared spectrometer of the SPICA satellite [2]. This array is to be made up of low saturation Transition-edge sensors (TES), which are a cutting edge form of cryogenic bolometer. The fact that these devices are so sensitive, means that great care must be taken in designing experiments to classify them. For this purpose, a shielded enclosure has been designed to bring the chips to the necessary milli-kelvin temperatures and protect them from external electromagnetic and vibrational interference. The aim of this particular thesis is to aid in the design of automated software routines to classify and analyse data gathered from the testing of these devices. These data are current-voltage (I-V) curves, which after careful analysis, glean vital information about the device parameters. Previous works in this area can be include [3, 4, 5].

## 1.1 Motivation and Goals

Previously, the analysis for each detector has been done manually. This process involved a lot of tedious and repetitive excel manipulation, which was manageable as only one detector at a time was treated. As the project is moving into the realm of larger number of sensors, held in an array, this method is becoming very unwieldy. Being able to identify and process data with little intervention from the user, will clearly save a large amount of valuable time. For these reasons, I have been tasked with automating as much of the process as possible, so that the backlog of data can

be quickly processed. The goal of the analysis is to classify the transition edge sensors by their attributes. These include:

- Transition temperature ($T_c$): the temperature at which the device will become fully superconducting.
- Thermal conductance $G$: how quickly heat is dissipated from the sensor, which is important for estimating the time resolution.
- Thermal conductance coefficient $\beta$: necessary for modelling the response of the sensor to incident radiation.

## 1.2 Form of Data

The aforementioned data come in a number of forms. Firstly, the raw data files are simple "ascii" files, with a .qdp extension, which will require pre-processing to ensure that they enter the program in the correct form. Secondly, there exists a subset of data which has already been manually processed which is in the form of excel spreadsheets. The two forms of data will allow for easy comparison between the automated process and the manual results already obtained. To complicate things somewhat, the readings are not consistent in their experimental approach. Differentiating one kind of experiment from another will be necessary before any analysis can be carried out.

## 1.3 Program Requirements

The specifications that shall guide the development of this software are as follows:

- The software should be able to load in raw data files in some format, preferably the raw form, to streamline the analytical process.
- The general form of the data should be identified. For example, measurements of the device while in the superconducting regime should be separated from those which are not. And/Or identify whether the curves are single or double sided.
- Once identified, appropriate analysis should be carried out on the data with minimal input from the user. This analysis should result in a power value.
- Multiple examples of these readouts should be collated for further calculation, where the results should be fitted to a model.
- Finally, the results of these calculations should then be returned to the user. These results should contain the physical parameters of the tested device.

As well as these basic functions, the program should not become prohibitively computationally expensive, so that it can easily and quickly be applied to large data sets. Moreover, errors should be easy to identify and rectify for the user. However, the end user will be assumed to be proficient in the use of the Python programming language, so that changes can be made if necessary.

# Chapter 2

# Theory & Setup

To fully understand the workings of our transition edge sensors (TES), it is necessary to draw from a daunting range of scientific and engineering disciplines. This chapter will aim to give a short introduction to a key number of these areas, so that the project can be understood in its physical and scientific context. After which, the experimental setup which generated these data treated in this thesis, will be explained.

## 2.1 Transition Edge Sensors

Traditionally, thermistors read out with a field effect transistor (FET) have been used for most calorometer applications. However, these offer restricted sensitivity and therefore lower quility measurements. In contrast, transition edge sensors are thin-film cryogenic thermal measurement devices with features resolved at lengths as small as the nano-scale. They make excellent sensors as they leverage low temperature superconductivity to detect minuscule amounts of incident radiation. Due to the sensor being held close to the critical temperature $T_c$ of the superconducting regime, the electrical properties of the sensors are very sensitive to any change in temperature. When the temperature of the device changes, due to an incident photon, the resistance across it will be altered a measurable amount, which in turn produces a change in the current, which can be measured by a superconducting quantum interference device (SQUID). This transition-edge which is used as a tool for measurement. These devices now have a number of applications in Astronomy and Physics, particularly as instruments for earth or space based telescopes. The following sections will elaborate on some of the key working mechanisms of the device. Within figure [3] we can see the differing electrical response when the sensor is in the dark and when it is illuminated by opening a cold shutter to diluted 300K black-body radiation.

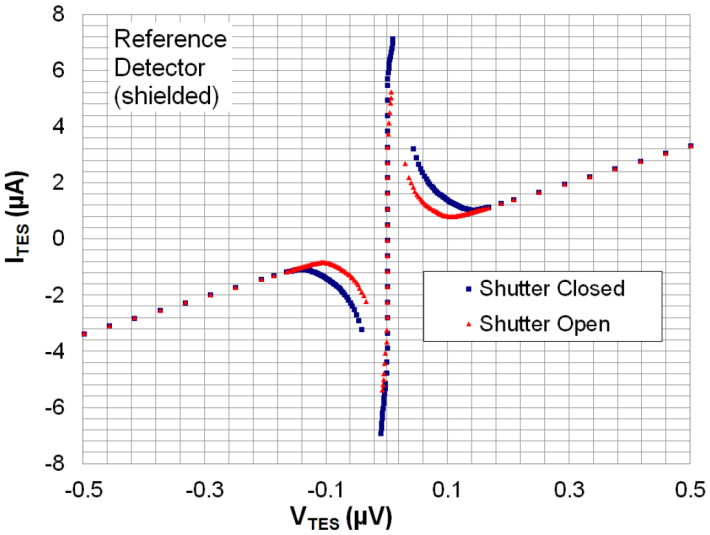



FIGURE 2.1: Response of TES with shutter open and closed [3]

### 2.1.1 Superconductivity in Materials

Superconductivity is a phenomenon which was first observed by Heike Kamerlingh Onnes in 1911 [6] when studying the behaviour of solid mercury at low temperatures. It was observed that the resistance of the material sharply reduces to zero once it passed its transition temperature of 4.2K. However, the theoretical workings of the phenomena were left unexplained until the development of BCS theory in 1957 [7], which described this behaviour of low temperature superconductors as the superfluidic flow of Cooper pairs.

Cooper pairs are electrons which have been bound together and act as one particle. When moving through the material, their binding energy prevents them from scattering, allowing them to continue travelling without resistance. The binding arises from interaction of the electrons with positive ions in the material lattice. Above the critical temperature, thermal energies in the material break these bonds preventing the pairs from being free to travel.

### 2.1.2 History of Use

The TES was not widely adopted until recently owing to a number of technical difficulties in their real world application. Due to the low resistance of the device, it was difficult to match the noise of the device to that of a FET amplifier. Modern setups use a superconducting quantum interference device (SQUID) coupled to the TES array, which can be more easily impedance matched to the device. This can then be read by room temperature electronics.

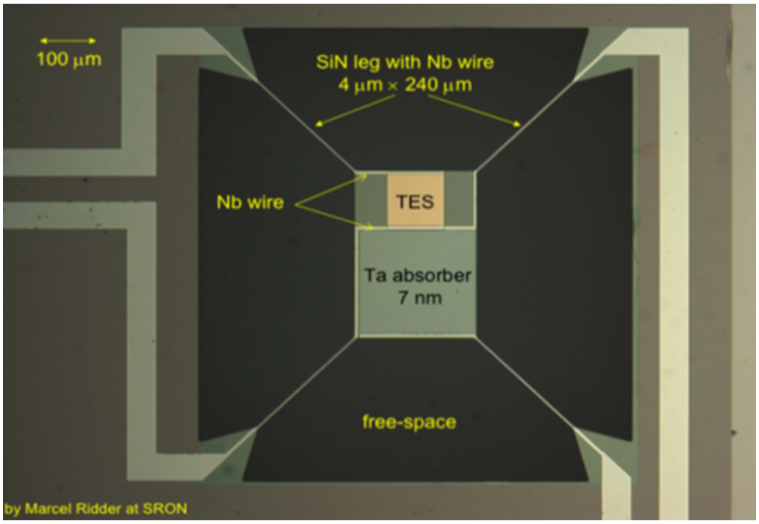



FIGURE 2.2: TES and surrounding architecture. TES and Ta absorber mounted on a 1 µm thick SiN film. This is supported by the four surrounding legs which act as a weak thermal link to the heat bath [5]

### 2.1.3 Heat Bath Power Flow

The flow of heat to bath is described by the following power law:

$$P_{bath} = K(T^n - T_{bath}^n) \tag{2.1}$$

With the coefficient $K = \frac{G}{nT^{n-1}}$, $n = \beta + 1$ and $\beta$ defined as the thermal conductance exponent [8]. These coefficients are determined by the physical structure of the device and are required for the further calculation of the response of the TES. In order for sensor to be reasonably resistant to temperature fluctuations in the bath temperature, $T_{bath}$ should be kept at a temperature below half of the critical temperature of the TES ($T_c$).

### 2.1.4 Electro-thermal Feedback

There exist two kinds of electro-thermal feedback in relation to these devices, both of which affect the stability of the electrical system. This feedback means that changes in either the electrical or thermal circuit induces a change in the other. Positive feedback can easily cause the system to become unstable, whereas negative feedback can offer a stabilising effect. As we know, a change in temperature will affect the resistance of the system. This change in resistance then alters the temperature through the change in Joule heating. In our voltage biased system, the joule heating is defined by [9]:

$$P \propto I^2 R \qquad V = IR \qquad P \propto \frac{V^2}{R} \tag{2.2}$$

In our TES, after incident radiation strikes the film: the temperature increases, this causes an increase in resistance, which leads to the Joule heating reducing. If the temperature of the film decreases for any reason: the resistance decreases, leading

to a higher amount of heating, bringing the film back to the equilibrium temperature. In addition to offering increased stability, negative electro-thermal feedback linearizes and speeds up the response, while increasing sensitivity by suppressing the Johnson noise [10].

### 2.1.5 Detector Sensitivity

The sensitivity of the device depends on the width of the transition region, which is affected by impurities in or at the edges of the thin-film as well as the shape of its geometry. The logarithmic sensitivity of a device is defined as:

$$\alpha = \frac{d \log R}{d \log T} \tag{2.3}$$

With R the resistance in ohms and T temperature in kelvin.

The differential equation governing the temperature of the transition-edge sensor is [9]:

$$C\frac{dT}{dt} = I(T)V - K(T^n - T_{bath}^n) \tag{2.4}$$

## 2.2 Experimental Set-up

### 2.2.1 Experiment Chamber

Due to the sensitivity of the devices, great care must be taken to isolate them from external sources of heat, vibration and magnetic interference. SRON Groningen has built a chamber which can hold the TES at the required temperature level while providing the necessary shielding for the experiments [4]. The chip is held in a chamber which is cooled by Leiden Cryogenics dilution refrigerator with a cooling power of 200 µW at 100 mK and a base temperature of 20 mK. The SQUID is kept at a higher, but still superconducting temperature, which is then read out by room temperature electronics.

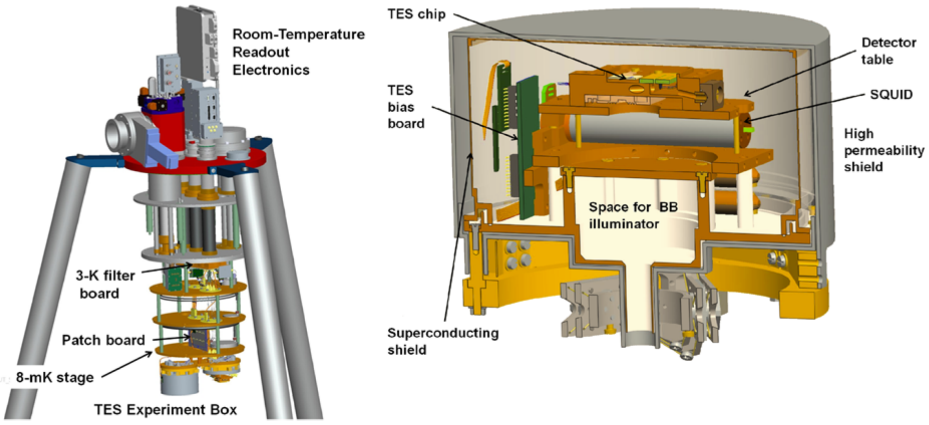



FIGURE 2.3: *Left*: Cryostat with shields removed. *Right*: Cross section of experiment box [4]

This chamber has subsequently been used to perform a large number of readings on TES chips, which will make up the data for this project.

### 2.2.2 Electrical Schematics

The experiment makes use of a voltage-biased TES held at temperatures less than the transition temperature of the device. This voltage-bias ensures that the TES will remain stably biased on its superconducting transition due to the negative electro-thermal feedback. The induced change of current is measured by the SQUID and read-out to room temperature electronics.

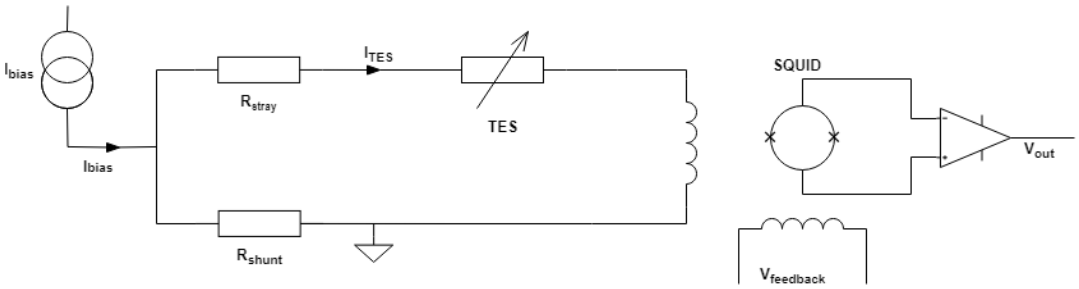



FIGURE 2.4: The electrical schematic for the experiment showing the placement of the resistors and SQUID

### 2.2.3 Important Parameters and Analysis

In figure 2.4 we see a number of resistors, currents and voltages which will be used in the analysis of the TES response. The parasitic resistance in the circuit is described by $R_{stray}$ and encompasses all miscellaneous resistances in the circuit not described by the other resistors, it can be assumed to be small. $R_{shunt}$ is a physical resistor wired in the circuit and which can be replaced by the experimenter, so changes between readings. The value is an important parameter in the analysis and must be input manually. There also exists a feedback resistance $R_{fb}$, which controls the bias that is applied to the TES, this is also set as an experimental parameter by the experimenter. The final experimental parameter is the $M_{ratio}$ which describes the ratio of mutual inductances between the coils of the input and feedback and the SQUID. This is also adjustable depending on the model of SQUID used for a certain measurement. From these parameters the associated gain can be calculated as $gain = \frac{1}{R_{fb} \cdot M_{ratio}}$.

These parameters must then be combined with the experimental data to produce $I_{TES}$ and $V_{bias}$ in the following way:

$$I_{TES} = (V_{fb} - \kappa) \cdot \left( \frac{1}{R_{fb} \cdot M_{ratio}} \right) \tag{2.5}$$

With $\kappa$ as any offset in the measurement data.

$$V_{bias} = (I_{bias} \cdot R_{shunt}) - I_{TES} \cdot (R_{shunt} + R_{stray}) \tag{2.6}$$

Once these values have been calculated, we can calculate both the power and the resistance simply:

$$P_{TES} = I_{TES} \cdot V_{bias} \tag{2.7}$$

$$R_{TES} = \frac{V_{bias}}{I_{TES}} \tag{2.8}$$

### 2.2.4 Experimental Procedure

A TES is held in the chamber at a temperature lower than $T_c$. The voltage-bias across the TES is varied and the change in feedback voltage is measured with the SQUID, producing the raw IV curves. The corresponding $I_{TES}$ and $V_{bias}$ can be calculated by analysing the raw IV curves with the experimental parameters. Once the data have been processed, the power curves can be calculated by plotting $P_{TES}$ against $V_{bias}$ or $R_{TES}$. Both curves will produce a graph which shows a plateau in the power readings. These regions correspond to the transition region where the device is held stable by the electro-thermal feedback. The power value of this plateau provides a power readout which can then be plotted against the bath temperature, seen in 2.5. The bath temperature can be changed and the experiment repeated to generate further data points.

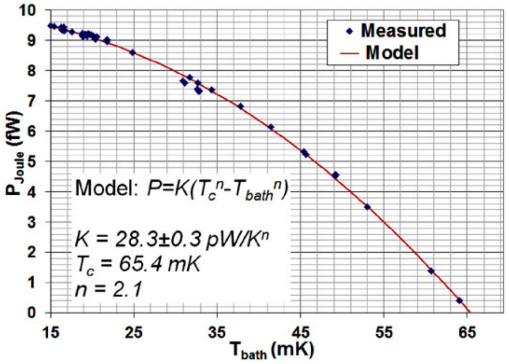



FIGURE 2.5: TES Joule Power at different bath temperatures [4]

These points can then be fitted with the power law (2.1) to find the parameters K, n and $T_c$. The leading coefficient K can then be used to calculate G, the thermal conductance of the TES.

### 2.2.5 Regions in the IV Response

Ideally, the data contains the 3 distinct regions seen in 2.1. Namely, the ohmic region (1), the transition region (2) and the superconducting region (3). The ohmic branch is the response of the device while it is far away from the transition region, when it acts similarly to a normal resistor. The transition region is the region where large changes in resistance arise from small changes in temperature, which is the phenomenon leveraged in order to detect incident photons and is therefore of most interest to us. The superconducting region is the area where the device is fully in the superconducting regime and current flows without hindrance. Unfortunately, the measurements which are to be analysed do not all take this neat form. Some of them only approach the transition from one side. Some lack a clear ohmic region or have no superconducting branch. These challenges mean that the data are difficult to analyse automatically.

# Chapter 3

# Experiments

## 3.1 Removing offset from measurement data

The first challenge in automating the analysis of the data is that there exists some offset in the measurement of the feedback voltage. Not adjusting the data accordingly will lead to the power plateaus being degraded, losing or worsening the power readout which will be the final output of this program. Therefore, a reliable method needs to be found to remove these offsets from all sorts of data.

### Assumptions and Methods

We will assume that the offset in the feedback bias (x-axis) is negligible, as it is in every series of measurements so far. For now, we will also assume that the data will have a clear and well defined region where the ohmic behaviour dominates. The simplest approach to removing this offset from the data may be to simply fit a straight line to these ohmic regions on each side of the data. The value where this straight line intersects the x-axis can be taken as the offset of the data. In order to do so, the `curve_fit` function was imported from the `scipy.optimise` module [11]. This function was then used in the `fit_ohmic` method within my `IV_curve` class.

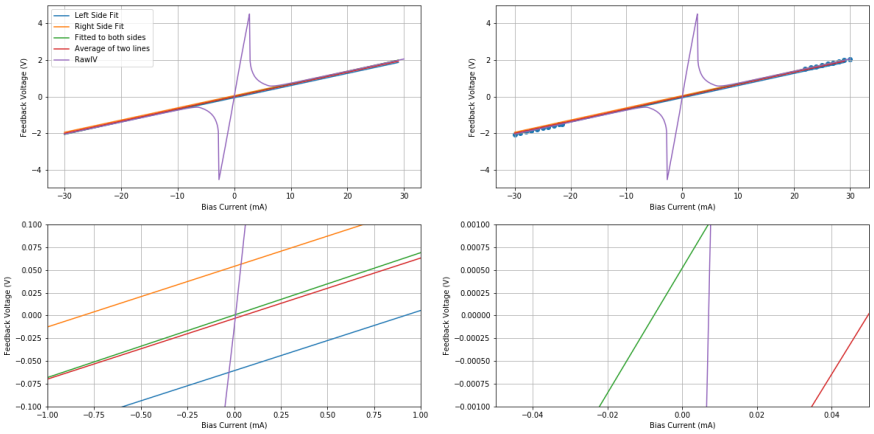
The data on which this method will be developed on was comprised of a curated set of measurements which displayed clear ohmic, transition and superconducting regions. Additionally, analysis of this data had already been carried out manually in an Excel spreadsheet. Thus, the results of my program could be easily compared with these values for accuracy.

### Results and Testing

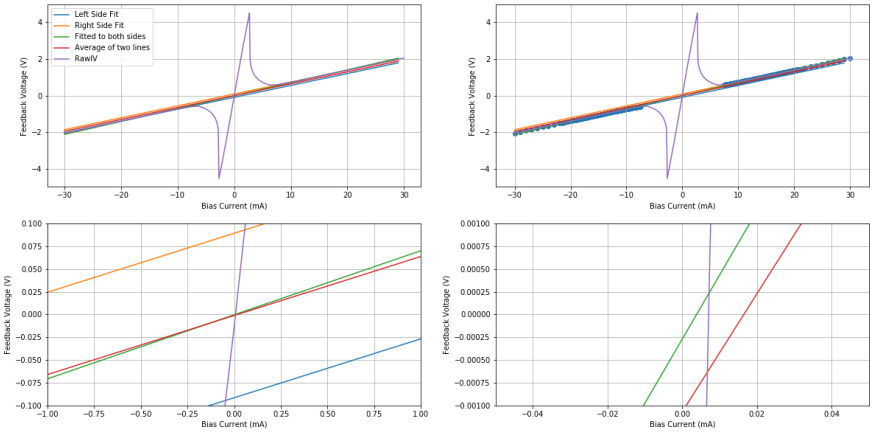
Initially, I attempted to remove the offset by fitting a line to both of the two ohmic regions that exist on either side, before the response becomes more heavily influenced by the superconductivity of the sensors. However, the number of points to be taken seemed to influence the calculated offset quite heavily. This issue arises from the fact that the influence of the superconducting behaviour is far-reaching and the device will only display true ohmic behaviour very far away from the regions that were measured.

I investigated this by fitting a number of lines, which were fitted to different parts of the data while also changing the number of points that were considered. I hoped to

find a method that would be stable and not so dependant on the number of points taken.




(A) Lines fitted to first 10 data points on each side of the range




(B) Lines fitted to first 100 points on each side of the range

FIGURE 3.1: Indication of the dependence of the calculated offset of the data on the number of points taken. The figure shows the result of fitting to both sides simultaneously and individually, then the lines being averaged.

As we can see in figure 3.1, the different methods of calculating the offset from the data are not stable and depend largely on the number of data points taken in the calculation. Unfortunately, this is further exacerbated by the fact that the data is not consistent in the number of readings or their distribution. The data seen in 3.1 represent the cleanest and most complete measurements.

## Discussion and Conclusions

The fact that the results depend so heavily on the choice of methods and sample is worrying. This indicates that this first step in the project may not be as simple as

I had initially assumed. While it is easy to fit the graph and produce an offset that is acceptable, the reliance on arbitrary magic numbers signals that this method may not generalise well to other data sets. Ideally, the offset should be invariant under such changes and not dependant on arbitrary choices of the experimenter. Further experimentation was required to find a method which would be able to accurately and consistently find the offsets in the given data. Nevertheless, rather than be stuck on this problem I decided to move on with the analysis in the hopes of solving this problem more rigorously later.

## 3.2 Calculating Power Plots

Once a value for the offset had been calculated, this could be used to generate power and resistance data. The data contained within these curves will allow for the identification of the power plateaus, corresponding to the electro-thermal feedback in the system. The value of these plateaus tell us about the behaviour of the TES under these measurement conditions. A series of data will give us a number of these values, which can be plotted against the bath temperature. These points can then be fitted to a known model to extract parameters vital for the calibration of the sensors.

### Assumptions and Methods

It is unclear to what extent the variance in the offset calculation will affect the power readouts from the program. In order to test the current method, we will compare the outputs to the values previously calculated. This data are still taken from the neat data used in the first step. As the values of the offsets are typically small, the effect should not be too large. However, any non-trivial offset will in all likelihood create reliability issues.

$$I_{TES} = (V_{raw} - V_{offset}) \cdot gain \tag{3.1}$$

The dependence of $I_{TES}$ on the $V_{offset}$ can clearly be seen in (3.1). This then carries forward into the calculation of $V_{bias}$ seen in (3.2).

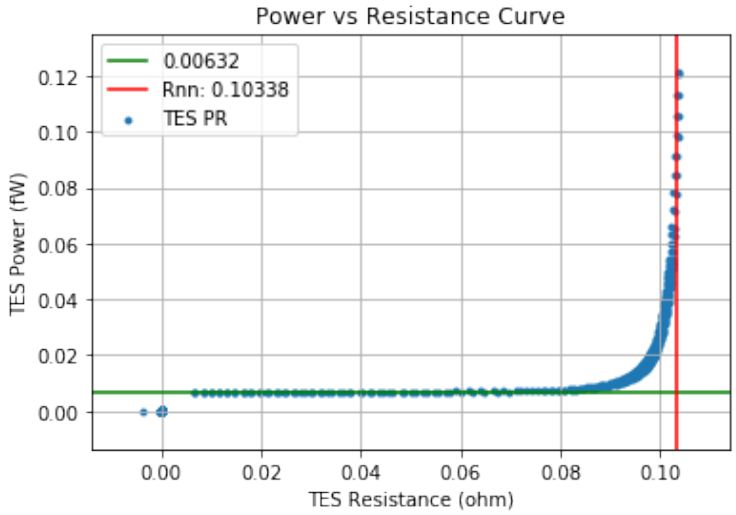
$$V_{bias} = (I_{raw} \cdot R_{shunt}) - I_{TES} \cdot (R_{shunt} + R_{stray}) \tag{3.2}$$

Finally, we can calculate the power and resistance values from the (3.1) and (3.2).

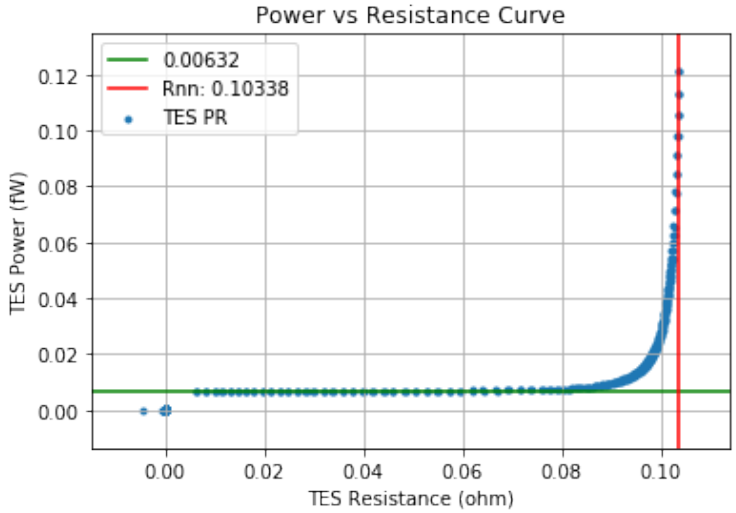
$$P_{TES} = I_{TES} \cdot V_{bias} \tag{3.3}$$

$$R_{TES} = \frac{V_{bias}}{I_{TES}} \tag{3.4}$$

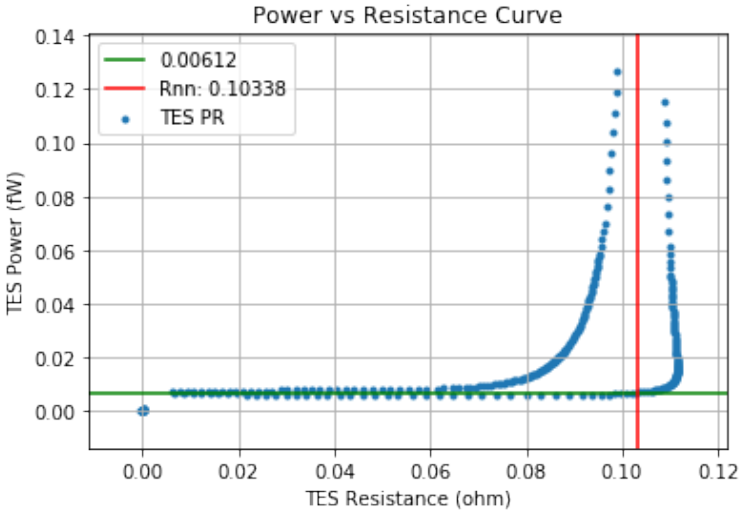
## Results and Testing




(A) Power graphs generated by fitting the ohmic section of the input




(B) Power graph using the offset in the excel sheet for comparison




(C) Power graph using a deliberately wrong offset to demonstrate change

FIGURE 3.2: Power plots generated using a variety of offset values

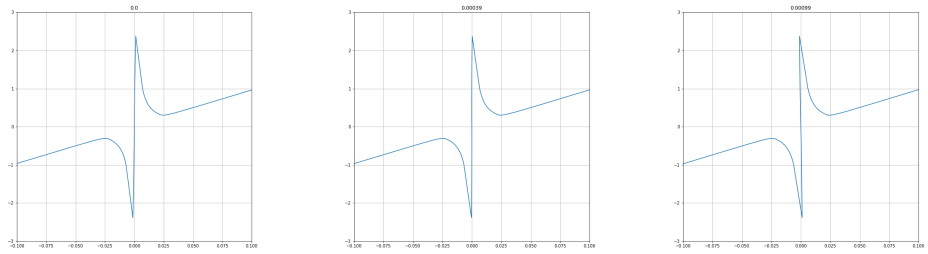
The plots which are being generated clearly show the presence of this plateau. However, it can also be seen that the data in the power resistance curves are less tidy than the graphs generated manually. This arises from the uncertainty in the offset measurement. If we manually input the offset used previously, these generate graphs with tighter variance around the ohmic resistance. Additionally, they produce more well defined power plateaus. Having said this, the values produced for the readout in 3.2 do not noticeably change apart from in the case where the offset has been made deliberately wrong. This is likely due to the clean nature of the data set we have been developing on.

### Discussion and Conclusions

This demonstrates the importance of finding accurate offset measurements if we wish to extract useful values from the analysis data with larger offsets. Furthermore, a reliable method must still be found for reading out the value of the plateau.

## 3.3 Fitting for $R_{stray}$

One of the experimental parameters is that of the stray resistance. This is the resistance that is present in the electronics outside of the device, such as the wiring to and from the measurement equipment. As the device is assumed to have zero resistance while it is operating in its superconducting region, any resistance that is measured in this part can be assumed to arise from the experimental electronics. We can safely assume that this value will be small, but it is still an important value in processing the data. The influence of this can be seen in figure 3.3. The assumption of the device having zero resistance in the superconducting region corresponds to that region of the data being fully vertical.



(A) $R_{stray} = 0$ (B) $R_{stray} = 0.00039$ (C) $R_{stray} = 0.00099$

FIGURE 3.3: Effect of variation of $R_{stray}$ on the shape of output

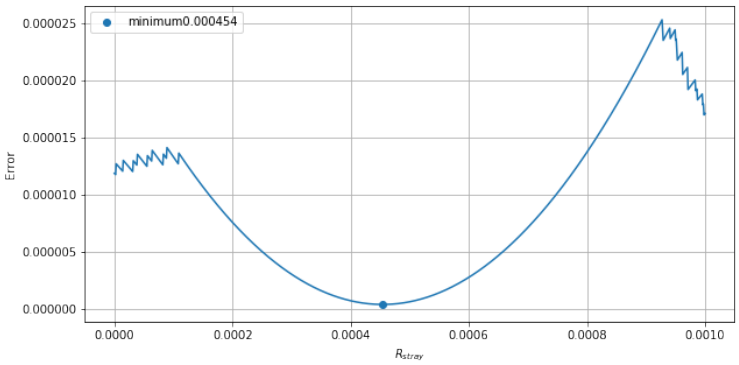
### Assumptions and Methods

We implicitly assume that the superconducting region exists in the data, as this will be necessary for any calculation. Additionally, we assume that any offset in the x-axis of the raw data is negligible. Finally, it seems reasonable to assume that the value of $R_{stray}$ will be small and non-negative. Having made these assumptions, I will be testing whether it is possible to calculate the value of this parameter without any input from the user. Previously, this value was input by hand by the person

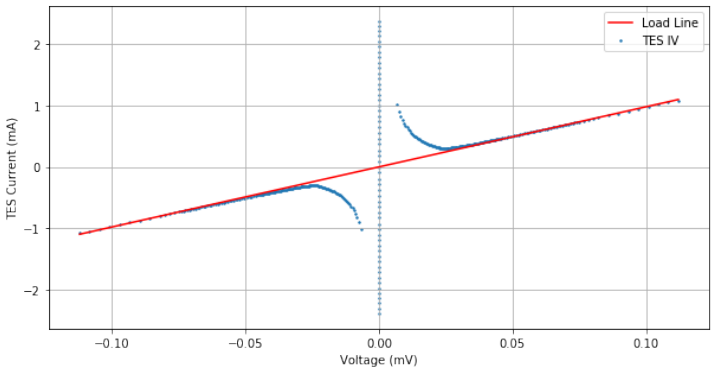
carrying out the analysis on the data. However, with these assumptions, the value of the stray resistance could actually be estimated and optimised without any input. For instance, we know that, the superconducting region in the data should be vertical in the analysed and adjusted curves. Therefore a range of values for the resistance ($R_{stray}$) can be iterated through and the resulting curves scored according to how vertical their superconducting region is, turning this unknown experimental parameter into a value which can be automatically calculated. The main challenge in doing this is finding a function which scores how well the outcomes of the trials reflect the true value of $R_{stray}$. After doing so, the score can be optimised (in this case minimised) to find a value for our parameter. This can then be saved within the structure of the program and used to analyse the data further.

### Results and Testing

Initially, I chose to optimise the absolute distance of points in the data from the x-axis; hoping that this would return the values for $R_{stray}$ which had the most vertical superconducting region. This naive approach gave poor results. Secondly, I introduced a threshold region around the x-axis which would exclude any data which was within it. Meaning that only the data close to the axis would have its distances optimised. This only created more problems as area around the axis is not known *a priori* and anything coded into the program would be arbitrary at best. In order to make this tenable, I would need to write a function which would analyse the data when it was input and identify the regions with different electrical behaviour, which is developed in the following section. Once this had been developed, this function was able to accurately find the stray resistance for any suitable set of measurements by minimising the distance of the points from the axis.




(A) Graph showing the minima in the error score corresponding to our $R_{stray}$




(B) Corresponding analysed IV curve

FIGURE 3.4: Indication of the dependence of the calculated offset of the data on the number of points taken. The figure shows the result of fitting to both sides simultaneously and individually, then the lines being averaged.

## Discussion and Conclusions

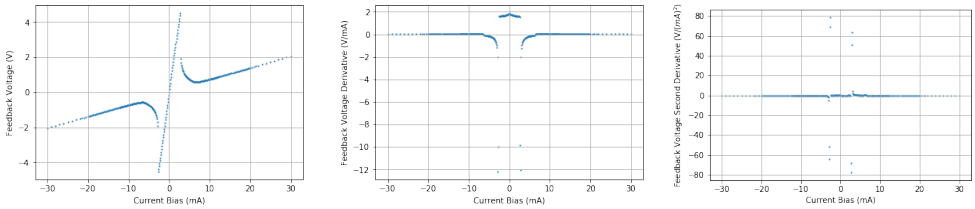
We assumed that measurements in this superconducting region exist. However, it should be noted that this part of the data does in fact not exist in every series of measurements. This method remains useful for the case that the data does exist, but cannot be used in every case. This requires that the program be able to identify whether a certain set of measurements is indeed suitable for such analysis, or whether extra input from the experimenter is required. It is not clear how we can extract the value of $R_{stray}$ without this information. One interesting result of this experiment was that the values of $R_{stray}$ that were found would usually differ somewhat between data sets within the same series. This is an interesting result as it was previously assumed that the value would remain constant between readings. However, it seems that a slightly varying value makes more sense, as the resistance would likely be affected by external temperature, internal heat generation or external magnetic fields. Allowing this value to vary may allow for better analysis and therefore more reliable calibration of the sensors. However, the performance of the algorithm leaves something to be desired. The running time is currently quite prohibitively slow, due to the structure of the algorithm just stepping through values iteratively. A more sophisticated algorithm for the optimisation is desirable to reduce computational time spent, especially as the algorithm will have to run on each set of measurements.

## 3.4 Identifying Regions

Assuming that the data will all be of the same rough form, it should be possible to split the data up into their constituent regions. Once this has been done, analysis can be performed on each part to gain insight into the experimental parameters. For example, being able to identify the ohmic region of the data may allow for a more consistent calculation of the offset in the raw data, as well as allowing the value of the stray resistance $R_{stray}$ to be calculated directly from the data.

### Assumptions and Methods

As stated, the data must clearly display each one of the regions. It is also important that these patterns are not overwhelmed by noise. In order to do this, I would need to write a function that identified these regions from some information that is already encoded into the raw data. This is an easy enough task for any human with some rudimentary instruction, however automating it would prove to be more difficult. The algorithm will be designed to iterate over the set of data and identify the different regions by the behaviour shown in their derivatives. For the definition of the ohmic region, it is defined as a deviation from the initial gradient by a margin, usually between 1 and 5 percent. This definition is vulnerable to noise in the beginning of the data.




(A) Raw Data (B) 1st Derivative (C) 2nd Derivative

FIGURE 3.5: Effect of variation of $R_{stray}$ on the shape of output

### Results and Testing

In order to find some reliable pattern in the raw data I began analysing the first and second derivatives. Luckily, the distinctive regions of the raw data also have distinct patterns in the derivatives. We can see that the gradient should start as positive in the left ohmic region, gradually decreasing until the point where the gradient passes zero. This region was split into the ohmic and the transition regions. The ohmic region was defined by the average of the gradient of the first few points of the data. Once the gradient deviated from this initial value by one percent, I defined this as the transition region. Unfortunately, this still relies on the first few data points sitting comfortably in the ohmic region, which may not hold for all data sets presented. Once the gradient becomes positive once again, this can be assumed to be the superconducting region. This process is mirrored from the other side to identify the regions on the other side of the transition.

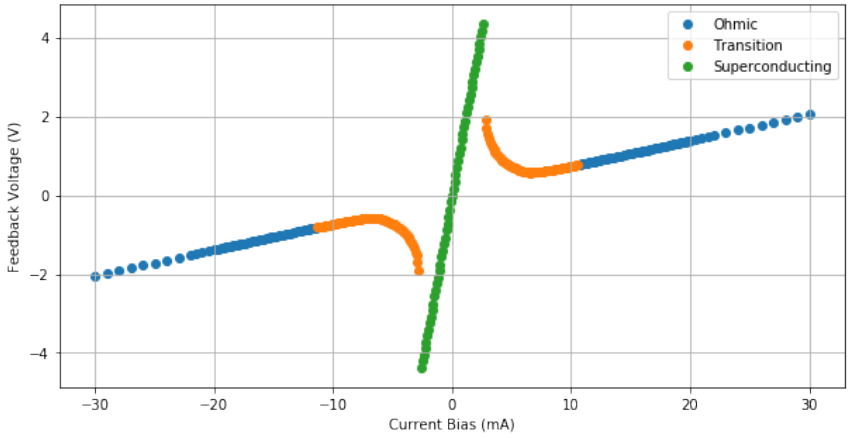



FIGURE 3.6: The raw input data split into its constituent regions

### Discussion and Conclusions

All of these methods work well for the curated data set on which I had been working so far. They could successfully identify the regions of the raw data. Now that the superconducting region could be non arbitrarily identified, the value of $R_{stray}$ could be optimised for. Additionally, I wrote a function which would identify the offset of the raw data by fitting a line to the superconducting region. This gave a much more stable result and did not rely on the ohmic region being complete.

## 3.5 Refined Power Plots

In order to generate the power readout it is necessary to identify the flat regions in the power data. Once again, we plotted the gradient of the power to identify patterns. As expected, the flat region around 0 seen in figure 3.7 corresponds to the plateau which we want to identify. In this section we will investigate the effectiveness of two different readout methods: gradient-based identification and maxima-minima identification. We wish to identify which gives the most consistent readings as the power readout is a key value in determining the device parameters.

### Assumptions and Methods

For the gradient-based approach we will assume that there exists a region in the power data where the gradient is low. This threshold is to be determined through investigation, but appears to be a potential magic-number in the analysis. The maxima-minima approach assumes that the data are clean enough to identify the transition point.

To identify this region of low gradient I wrote a naive script which checks for power values that have a gradient close to zero and averages them to give an output. This seems to work for this data set, but the introduction of another arbitrary threshold is undesirable as we want it to generalise well to other sets of measurements, which may have different gradient profiles.

The maxima-minima is simply found by iterating over the gradient of the data and identifying where the gradient first crosses zero.

These methods are to be tested on a wider range of data, including some lower quality sets. This should test the reliability of the methods and allow for comparison.

## Results and Testing

The readouts supplied by the first method appeared to be consistently lower by about 0.005 than those in the spreadsheets. The output of the second method were closer to the comparison values, but were not consistently above or below them.

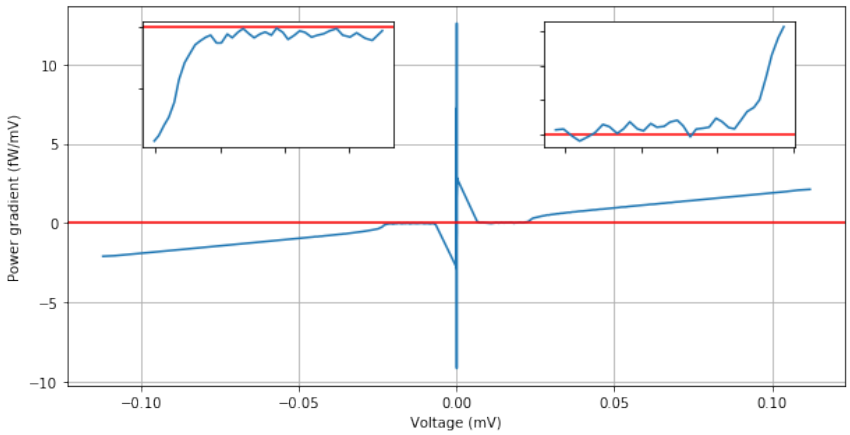
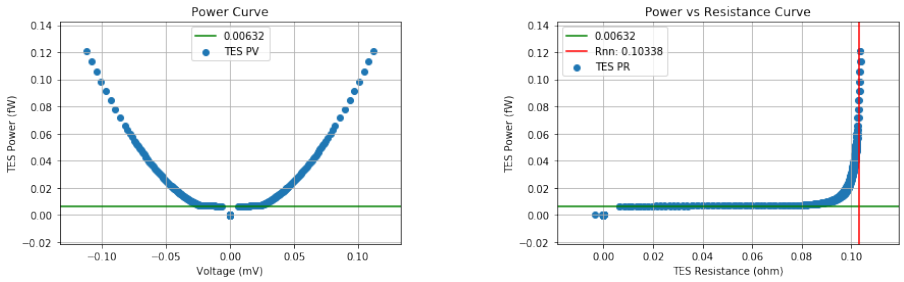



FIGURE 3.7: Graph showing gradient values around 0 correspond to the plateau regions




(A) (B)

FIGURE 3.8: The generated Power/Voltage and Power/Resistance curves with a power readout and ohmic resistance displayed

## Discussion and Conclusions

While the readout generated by the first method worked well for the curated data set, it produced unpredictable results when applied to messier data. The maxima-minima method offers a readout without any reliance on arbitrary thresholds and gives a readout even when the slope of the power plateau is larger. This slope in the plateau region arises from the finite transition width of the sensor and cannot be determined without experiment. For this reason, it would be difficult to justify introducing thresholds which have no basis in our understanding of the physics.

Additionally, the exact point taken for the power readout remains an estimation and it is important to be consistent with this selection. Noise in the data could still prove to be a problem. However, the algorithm searches for the transition point from both ends and can still produce a result if one is missing or affected by noise. While being tested, the algorithm correctly identified the minima and maxima in every case. Further refinement could include interpolating the data between the points so that the minima/maxima can be identified with even sparse data sets. However, the transition region is dense in all of the data used so far, so no problems have arisen as of yet. In conclusion, the power readouts produced by the second method should be robust enough to act as the data points to model the heatflow from the device.

## 3.6 Removing Offset with Optimisation

Having developed a number of methods to analyse the experimental data, it was now time to find a rigorous method of finding the offset. The method using the superconducting region gave acceptable results. However, many of the data sets lack this critical region, as well as lacking any true ohmic region. This severely limits the effectiveness of the so far developed functions.

The new method must be able to effectively find an offset with minimal data, only describing the transition. While also still working for the data which has well defined regions. One approach would be to classify the different sets of data as they enter program and then apply different methods depending on which regions they contain. Taking such an approach would mean that a huge program would have to be written, including a classifier as well as functions relating to each of the classes. Preferably, the program would be able to apply a simple analysis to the data which would work in the vast majority of normal use cases. Doing so would require that a minimal number of assumptions are made about the structure of the data. As all previous methods had made a number of assumptions, I decided to take a wholly different approach.

### Assumptions and Methods

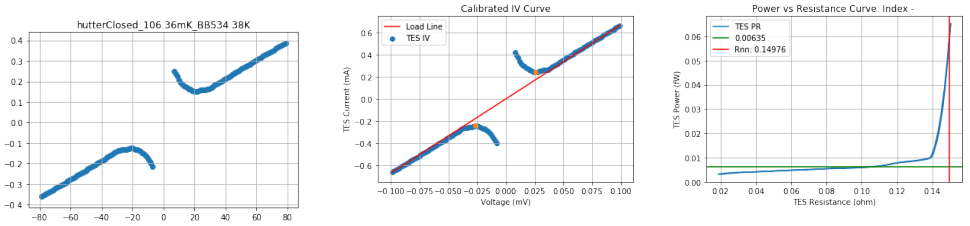
The only region that could be assumed to exist would be the transition region. If it did not, no output can be generated from the measurements in any case. We are interested in this region as it allows for the plateau of the power plots to be found and returned. We know that the correct offset will give a better defined plateau in these graphs. Therefore, the correctness of the offset can be inferred by searching for a value which gives the nicest looking power plots, turning this problem into another optimisation problem. This implicitly assumes that the data are double sided, in that the transition is approached from the positive and negative directions.

### Results and Testing

The resulting recursive algorithm implements a binary search within a reasonable range of offsets. The algorithm scores the results of each offset and then halves the region of search by selecting the half with the lowest score. This allows for the answer to be found very quickly, which is very useful considering this must be done

for each of the measurement sets within the series. The function returns the value of the offset once the precision threshold has been met.

The key component of this method is the correct choice of scoring function. Many of the first error functions gave poor and inconstant results, as the desired result would be replaced with one which had won out due to the poor definition of the function. Care was also taken to properly balance the different parts of the function, so that the resulting plot would be what was expected. After much refinement, the final version of the function produced good results after being subjected to a wide range of different data sets.

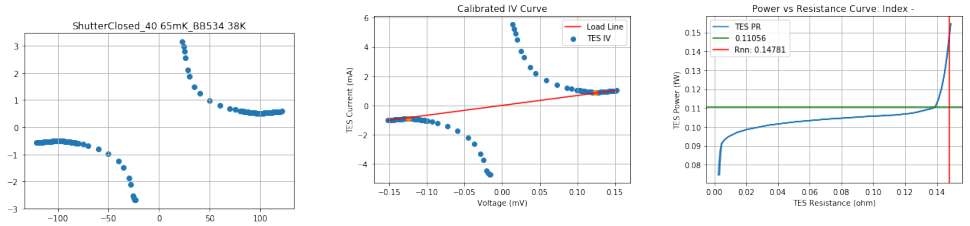



(A) Raw Data

(B) Analysed IV with readout points

(C) Readout shown on power plot

FIGURE 3.9: Testing on data with no superconducting region




(A) Raw Data

(B) Analysed IV with readout points

(C) Readout shown on power plot

FIGURE 3.10: Testing on data with only transition region

## Discussion and Conclusions

These results meant that the offset could be found accurately under almost no assumptions about the data, producing effective results for noisy or incomplete data sets. Unfortunately, this increase in generality meant that the value of $R_{stray}$ could no longer be calculated without input. Now this parameter will have to be input along with the other experimental parameters. However, if a method of identifying curves with this region is later developed, $R_{stray}$ can still be found with the old method. It could be argued that this method is an overly complicated way of calculating a simple parameter. However, the unreliability displayed with the previous methods, as well as the efficient implementation of this one, make this the clear choice for finding the offsets reliably. Having said this, the assumption that the data is double sided is a reasonably strong one. While it is true that most of the data is indeed double sided, another method will have to eventually be developed for the processing of the single sided curves. This is not catastrophic, as all but the oldest data is double sided, which means that all of the relevant data will be correctly processed under this assumption.

## 3.7 Analysing Power Readouts to Calculate Device Parameters

Now that the power readout could be calculated automatically, without prior knowledge of the form of the data, it was time to move onto the final stage of the project. Each one of these power readouts will become one data point to be used to find the parameters of the TES device. So far the program had been dealing with each raw IV curve as a single discrete object. In order to handle a number of them at the same time, it would be simpler to design a larger structure to hold a number of these objects.

### Assumptions and Methods

The only assumption that needs to be made in this case is that the data will be double sided and that it is in an excel sheet or directory containing ascii files.

### Results and Testing

This functionality is encapsulated within an IV_Series class. Data can be read into the program in a number of ways: excel sheets, a list of ascii files (.qdp) or a directory containing a number of them. Additionally, the experimental parameters (detailed in 2.2.3) must be passed into the program. Once the data is loaded in, an IV_curve object is created and stored for each measurement set. The offset of the raw data is calculated for each one of these objects, then the power analysis is carried out; finally returning a power readout. These readouts are then plotted against the bath temperature associated with that measurement. Finally, a curve is fitted to the data points which results in the device parameters K, n and $T_c$ being found. These final results are stored in a .txt file in the same directory along with the final plot. If the results are suspect in any way, the program can be run in debug mode, which will dump all intermediate graphs into the directory so the user can identify where the problem arises. Additionally, the program automatically rejects measurements which show no sign of superconductivity or show other signs of irregularity.

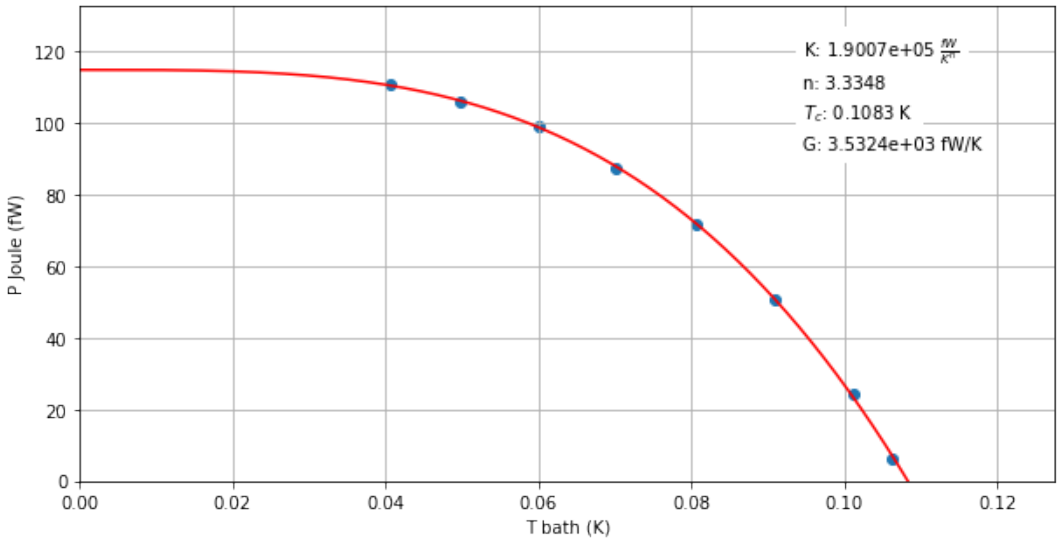



FIGURE 3.11: Example output of program after processing

### Discussion and Conclusions

Critical temperature and n are both well within expected bounds predicted from the theory, as n is typically between 2-5 and $T_c$ is expected to be approximately 100 mK. The parameters that are output for n and $T_c$ match well with the reference values. However, the value for K is around twice the value in the reference. This is due to the interdependence between the value for n and K. When the value for G is calculated we get 3525 $\mathrm{fW\,K^{-1}}$, which is relatively close to the previously calculated 3226 $\mathrm{fW\,K^{-1}}$. This discrepancy may arise from the differences in the points of reference taken to read the power. Using the fixed points of the input is actually a more reliable and robust method than that used in the reference material. This does not preclude the possibility of errors being made in other areas, however. An important refinement to the program would be the inclusion of error estimations, this may even be able to somewhat bridge the gap between this result and the historical one. This particular value is an order of magnitude higher than the TES tested in [4], which means that the sensor will more quickly lose any incident energy to the bath.

## 3.8 Discussion and Conclusions

The program in its current form is quick and reliable enough to be fit for purpose. It can produce results from less than ideal data and identifies problematic inputs. However, there remains work to be done to make further refinements to the workings and results. In this chapter we will examine the performance of the program against the initial program specifications and identify areas to be improved or new features that can be implemented.

### 3.8.1 Evaluation of Project against Requirements

The software is currently able to read the data in both excel and raw format, fully meeting the specification. This allows the user to be flexible in the use of the program and future proofs it somewhat. It would also be easy to add extra formats to the software by simply writing another function within the `IV_series` class. Currently, the `IV_series` object tests each data set with a few basic criteria to see if they are suitable for analysis with the program. Any rejected files can be looked up in the error file by the user where some basic information is stored. While this is an adequate approach, ideally the analysis should be more robust to anomalous forms of data which may pass the checks currently in place. In addition, the older data sets which are one sided would not be handled well at the moment. Therefore, the specification is met to a basic standard. The only input required from the user is the parameters of the experiment. This includes the values of the physical resistances in the system, as well as any factors to scale the input data correctly. Although, it would have been nice to keep the ability to fit for $R_{stray}$, the gain in generalisability is worth the cost. Future versions, may be able to re-implement this feature. As it stands, the majority of the analysis has been automated, which offers significant improvement over the manual process. As for the output of the program, the important parameters are saved on the plotted figure and stored in a separate text file. The figure should allow for any discrepancies to be spotted by the user, avoiding any errors or faulty calculations passing unnoticed.

Regarding the efficiency of the program; implementation of the potentially most expensive algorithm, the y-offset optimisation, still runs remarkably fast. As it stands, the creation of 100 `IV_series` from a directory containing 18 raw files takes 13.9 seconds. So around 0.139 seconds for the creation of one object. The analysis of the on of these series took an average of 42.4 seconds. This is not an insignificant amount of time and could certainly be improved, but is more than acceptable compared to the manual analysis.

The program also offers debugging modes, for easy troubleshooting by the user as well as custom errors. These help to meet the specification for usability and visibility of potential errors.

### 3.8.2 Further Refinements

With additional time, it should be possible to expand on the current program and add ample new features. The first, and potentially most important would be to add some kind of error estimation to the software. Errors estimated in the reference data were all too small to be visible in the final graphs. This does not seem truly representative of the potential experimental errors, so some attention should be paid to ensuring these can be adequately approximated. Secondly, some kind of automatic noise detection and handling would be a welcome addition. Applying a kind of Gaussian filter to the data may be necessary in some cases. More refined recognition techniques, such as k-means clustering or t-SNE algorithms, should allow for more robust classification of experimental data. This would also allow for a more complex and capable program to analyse a wide array of data, once correctly classified. Finally, further documentation is always a welcome addition to any project.

# Appendix A

# IV_curve and IV_series Classes

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy.polynomial.polynomial as poly
from scipy.interpolate import interp1d

import read_qdp
import os
import sys

class IV_curve(object):
    '''

    Methods:
    self.plotPV() - plots PV graph
    self.plotIV() - plots IV graph
    self.plotPR() - plots PR graph
    self.plotRegions(raw=1) - set raw to false to plot analysed IV regions

    #TODO - Store the temperatures in the object
    '''

    def __init__(self, rawIV, params = 0, name='NA', tweaks = [1,1,1]):
        ''' Creates a TES object from a pandas dataframe.

        Reads the first two coloums of the dataframe and saves
        them as attributes. Dataframe can consist of more coloums but they are
          ignored.
        Params can be a list or tuple of values in this order [R_fb, M_ratio,
          R_shunt]
        R_stray can be inlcuded as extra paramter. Doing so will prevent the
          calc_R_stray method being used to
        generate a value.

        Attriutes after initialisation:
        rawI (np.array[floats]), rawV (np.array[floats])
        raw_grad/raw_2grad (list[floats]) - 1st and 2nd differentials of rawIV
        superconducting (bool) - whether the data shows signs of superconductivity
        y_offset (float)
        I_TES (list[floats]), V_bias (list[floats]) - Calibrated data
        R_fb, M_ratio, R_shunt (floats)- experimental parameters
        gain (float) - calcuated from M_ratio and R_fb
        R_stray (float) - either input or calculated
        P_TES , R_TES (floats) - Power and Resistance values
        rnn (float)

        ohmic_reg, trans_reg, super_reg (ints) the demarcated regions.
        If not super conducting ohmic_reg will be the entire data range
```

```
        '''
        self.df = rawIV
        # Drop NaNs in spreadsheet to prevent errors later
        rawI = (rawIV.iloc[:,0].dropna()).values #Store as individual np.arrays for more readable code
        self.rawI = [entry * tweaks[0] for entry in rawI]
        rawV = (rawIV.iloc[:,1].dropna()).values
        self.rawV = [entry * tweaks[1] for entry in rawV]
        try:
            self.T_BB = rawIV.iloc[:,4].dropna().values
            self.T_bath = rawIV.iloc[:,3].dropna().values
        except:
            print("No supplied Temperature data")

        self.P = False
        try:
            assert len(rawIV.columns) > 5
            self.P_BB = (rawIV.iloc[:,-1]).dropna()
            for entry in self.P_BB:
                if entry != 0:
                    self.P = True
        except:
            print("Failed to parse Power Data")

        self.raw_grad = np.gradient(self.rawV, self.rawI)#Prevents repeated calls to gradient function later
        self.raw_2grad = np.gradient(self.raw_grad, self.rawI)
        self.superconducting = self.calc_superconducting() # This needs to be done first as other inits depend

        #iniditalise these values with the methods
        self.name = name
        self.y_offset = 0

        # All needed to calculate adjusted I and V etc. If params are included in initialisation, then they are
        # used. Otherwise defualts are used.
        # TODO: raise exception if len is not 3
        if params:
            try:
                self.R_fb = params[0]
                self.M_ratio = params[1]
                self.R_shunt = params[2]
            except IndexError:
                print("Error: Parameters not correct")
                return
        else:
            print("WARNING: Using defualt paramters")
            self.R_fb = 100099.6
```

```
            self.M_ratio = 18.9333333333333
            self.R_shunt = 0.00389

        self.gain = 1 / (self.R_fb * self.M_ratio)


    ############################calc_methods#################################

    def calc_I_TES(self, tweak=1):
        '''
        Generate I_tes (array) data from tweak(float), y_offset (float), gain
          (float).
        tweaks not yet used
        '''

        self.I_TES = [(entry - self.y_offset) * (10**6) * self.gain for entry in
          self.rawV]

    def calc_V_bias(self, tweak=1):
        '''
        Generates V_bias data
        '''
        self.V_bias = [((self.rawI[i]) * self.R_shunt) - (self.I_TES[i]*
          (self.R_shunt + self.R_stray)) for i in range(len(self.rawI))]

    def calc_P_TES(self):
        self.P_TES = [self.I_TES[index] * self.V_bias[index] for index in range(len
          (self.I_TES))]

    def calc_R_TES(self):
        self.R_TES = [self.V_bias[i] / self.I_TES[i] for i in range(len
          (self.V_bias))]

    def calc_superconducting(self):
        '''
        Test if the gradient deviates from the average by a significant margin. And
           save result in attribute.
        Returns the boolean value of self.superconducting
        '''
        grad = self.raw_grad
        grad_av = sum(grad)/len(grad)
        if abs(min(grad)) > 2*(grad_av) and min(grad) < 0:
            self.superconducting = True
        else:
            self.superconducting = False

        return self.superconducting

    ############################utility methods#############################
```

```
    def f(self, x, A, B):
        '''
        Function of straight line for optimisation function to find
        coefficients for.
        '''
        return A*x + B

    def fit_ohmic(self, inputdataX, inputdataY):
        '''
        Accepts two lists for X and Y vals and a tuple of indices. These are then
          used
        to fit a straight line to the straight regions.
        Returns C, D, xvals. Which are the gradient, intercept and range of the
          fitted curve.
        '''

        r1start, r1end = (0,7)
        r2start, r2end = (-7,-1)

        firstx = min(inputdataX)
        lastx = max(inputdataX)
        xvals = np.linspace(firstx, lastx)

        #Slice the data in the frame and store in an array
        yvals1 = inputdataY[r1start:r1end]
        yvals2 = inputdataY[r2start:r2end]
        lineDataY = np.append(yvals1, yvals2)

        xvals1 = inputdataX[r1start:r1end]
        xvals2 = inputdataX[r2start:r2end]
        lineDataX = np.append(xvals1, xvals2)

        #fit the sliced data to a line
        C, D = curve_fit(self.f, lineDataX, lineDataY)[0]

        return C , D , xvals

    def power_readout(self, plot = 0):
        indices = []
        count = 0
        for index in range(len(self.raw_grad)):
            if self.raw_grad[index] < 0:
                count += 1
            else:
                count = 0
            if count >= 3:
                indices.append(index-count)
                break
```

```
        count = 0
        for index in range(len(self.raw_grad)):
            if self.raw_grad[-index] < 0:
                count += 1
            else:
                count = 0
            if count >= 3:
                indices.append(-(index-count))
                break
        readout = 0
        if plot:
            if len(indices) == 2:
                readout = (self.P_TES[indices[0]] +  self.P_TES[indices[1]])/2
            elif len(indices) == 1:
                readout = self.P_TES[indices[0]]
            else:
                readout = 0

        self.readout_idx = indices
        self.readout = readout
        return readout

    ################Plotting Methods####################
    def plotRaw(self):
        plt.scatter(self.rawI, self.rawV, label='Raw IV')
        plt.grid()
        plt.title(self.name)
        plt.show()

    def plotIV(self, plot = True):
        '''
        Plots the calibrated IV curve if plot is set to true in the args. Otherwise ↵
           just calcs R_nn
        '''
        C, D , Axvals = self.fit_ohmic(self.V_bias, self.I_TES)
        self.rnn = 1/C
        if plot:
            fig, ax = plt.subplots()
            #plt.figure(figsize=(50,50))
            plt.plot(Axvals, self.f(Axvals, C , D), color='r', label='Load Line')
            plt.grid()
            plt.scatter(self.V_bias, self.I_TES, label='TES IV')
            if len(self.readout_idx) == 2:
                plt.scatter([self.V_bias[self.readout_idx[0]], self.V_bias ↵
                  [self.readout_idx[1]]], [self.I_TES[self.readout_idx[0]], ↵
                  self.I_TES[self.readout_idx[1]]])
            ax.set_xlabel("Voltage (mV)")
            ax.set_ylabel("TES Current (mA)")
```

```
            plt.title("Calibrated IV Curve")
            ax.legend()
            plt.show()

    def plotPR(self, name = ''):
        fig, ax = plt.subplots()
        plt.grid()
        plt.plot(self.R_TES, self.P_TES, label='TES PR')#, alpha=0.2)
        plt.title("Power vs Resistance Curve: Index - " + str(name))
        ax.set_xlabel("TES Resistance (ohm)")
        ax.set_ylabel("TES Power (fW)")

        if self.calc_superconducting():
            readout = self.power_readout(True)
            plt.axhline(readout, color='g',label=round(readout,5))

        plt.axvline(self.rnn, color='r',label='Rnn: ' + str(round(self.rnn,5)))
        plt.legend()

    def plotPV(self):
        fig, ax = plt.subplots()
        plt.grid()
        plt.scatter(self.V_bias, self.P_TES, label='TES PV')
        plt.title("Power Curve")
        ax.set_xlabel("Voltage (mV)")
        ax.set_ylabel("TES Power (fW)")

        if self.superconducting:
            readout = self.power_readout(True)
            plt.axhline(readout, color='g',label=round(readout,5))

        plt.legend()



    ################Methods TODO###################
    def smooth_IV(): #Remove qunatum jumps?
        #TODO
        ''' Maybe a function to smooth noisy data before we find the
        y offset.
        Inputs: rawIV (or just use self?)
        Outputs: stores smoothRawIV in self
        '''
        pass

    def reject_outliers(self, data, m = 2.):
        d = np.abs(data - np.median(data))
        mdev = np.median(d)
        s = d/mdev if mdev else 0.
```

```
        return data[s<m]

    def y_search(self, step, pivot, precision):

        ys = []
        counts = []

        if step <= precision:
            self.y_offset = pivot
            return pivot


        for y_off in np.arange(pivot-(10*step),pivot+(step*10),step/10):
            I_TES = [(entry - y_off) * (10**6) * self.gain for entry in self.rawV]
            V_bias = [((self.rawI[i]) * self.R_shunt) - (I_TES[i]*(self.R_shunt +
              self.R_stray)) for i in range(len(self.rawI))]
            P_TES = [I_TES[index] * V_bias[index] for index in range(len
              (self.I_TES))]
            P_grad = np.gradient(P_TES)
            P_grad_smooth = self.reject_outliers(P_grad)
            R_TES = [V_bias[i] / I_TES[i] for i in range(len(V_bias))]
            count =0
            max_R = max(R_TES)

            #for entry in P_grad_smooth:
                #count += abs(entry)**2
            for i in range(int(len(R_TES)/10)-1):
                count += np.sqrt((R_TES[i] - R_TES[-i])**2)
                count += (R_TES[i] - self.rnn)**2
                count += (R_TES[-i] - self.rnn)**2

            count += len(R_TES)*(self.rnn - max(R_TES))**2

            for entry in P_TES:
                if entry < 0:
                    count += 10* entry **2

            for entry in R_TES:
                if entry < 0:
                    count += 10* entry**2


            counts.append(count)
            ys.append(y_off)

        y_offset = ys[np.argmin(counts)]


        return self.y_search(step/10, y_offset, precision)
```

```
    def y_search_nonsuper(self, step, pivot, precision):

        ys = []
        counts = []

        if step <= precision:
            self.y_offset = pivot
            return pivot


        for y_off in np.arange(pivot-(10*step),pivot+(step*10),step/10):
            I_TES = [(entry - y_off) * (10**6) * self.gain for entry in self.rawV]
            V_bias = [((self.rawI[i]) * self.R_shunt) - (I_TES[i]*(self.R_shunt +
              self.R_stray)) for i in range(len(self.rawI))]
            P_TES = [I_TES[index] * V_bias[index] for index in range(len
              (self.I_TES))]
            P_grad = np.gradient(P_TES)
            P_grad_smooth = self.reject_outliers(P_grad)
            R_TES = [V_bias[i] / I_TES[i] for i in range(len(V_bias))]
            count =0
            max_R = max(R_TES)

            #for entry in P_grad_smooth:
                #count += abs(entry)**2
            for i in range(int(len(R_TES)/10)-1):
                count += np.sqrt((R_TES[i] - R_TES[-i])**2)
                count += (R_TES[i] - self.rnn)**2
                count += (R_TES[-i] - self.rnn)**2

            count += len(R_TES)*(self.rnn - max(R_TES))**2

            for entry in P_TES:
                if entry < 0:
                    count += 10* (entry **2)

            for entry in R_TES:
                if entry < 0:
                    count += 10* (entry**2)


            counts.append(count)
            ys.append(y_off)

        y_offset = ys[np.argmin(counts)]


        return self.y_search(step/10, y_offset, precision)
```

```
    def reverse_y_offset(self):
        '''Requires R_stray to have been already calculated or input.
        Attempts to calc offset through optimising the power plot.
        '''
        if self.superconducting:
            self.y_search(5,0,0.000001)
        else:
            self.y_search_nonsuper(5,0,0.000001)

    def calc_all(self):
        self.power_readout()
        self.calc_I_TES()
        self.calc_V_bias()
        self.plotIV(False)
        self.reverse_y_offset()

        self.calc_I_TES()
        self.calc_V_bias()
        self.plotIV(False)
        self.calc_P_TES()
        self.calc_R_TES()
        self.power_readout(True)


# In[89]:


class IV_series(object):

    def __init__(self, input_string,  params, tweaks = [1,1,1], mode = 'dir'):
        self.R_nn = 0
        self.R_stray = 0

        self.name = input_string

        if mode == 'dir':
            self.read_dir(input_string)
        elif mode == 'files':
            self.read_files(input_string)
        elif mode == 'excel':
            self.read_excel(input_string)
            tweaks = [1,1,1]
        else:
            print("Please select mode: dir/files/excel")
            return


        self.data = [IV_curve(series, params, filename, tweaks) for series,
          filename in self.series]
```

```
    def set_R_stray(self, val):
        self.R_stray = val

    def set_rnn(self, val):
        self.R_nn = val

    def calc_R_stray(self):
        pass # TODO impliment


    def calc_R_nn(self):
        average = 0
        for IV_curve in self.data:
            IV_curve.plotIV(False)
            average += IV_curve.rnn

        IV_series.R_nn = average/len(self.data)


    def read_dir(self, dirname):
        series = []
        for filename in os.listdir(dirname):
            if filename.endswith(".qdp"):
                print(filename)
                series.append((read_qdp.read_qdp(dirname+ '\\' +filename),
                  filename))

        self.series = series

    def read_files(self, filenames):
        series = []
        for filename in filenames:
            if filename.endswith(".qdp"):
                series.append((read_qdp.read_qdp(filename), filename))
        self.series = series

    def read_excel(self, filename):
        dataDict = pd.read_excel(filename, sheet_name=None)
        series = []
        for name, sheet in dataDict.items():
            new1 = sheet.iloc[:,0]
            new2 = sheet.iloc[:,1]
            new3 = sheet.iloc[:,2]
            try:
                new4 = sheet[['T bath', 'T BB', 'P BB']].copy()
                series.append((pd.concat([new1,new2,new3,new4], axis=1), name))
            except:
```

```
                print("Cannot find T BB and T bath or PBB: Please rename in sheet" )
                series.append((pd.concat([new1,new2], axis=1), name))

        self.series = series

    def plot_power(self, plot=True):
        #powers_off = []
        powers = []
        t_bath = []
        #t_bath_on = []

        for entry in self.data:
            if entry.superconducting and not entry.P and entry.valid:
                powers.append(entry.power_readout( True))
                t_bath.append(np.average(entry.T_bath))

            #elif entry.superconducting and entry.P:
             #   powers_on.append(entry.power_readout(True))
              #  t_bath_on.append(np.average(entry.T_bath))

        self.t_bath = t_bath
        #self.t_bath_off = t_bath_off
        self.powers = powers
        #self.powers_on = powers_on

        if plot:
            plt.scatter(t_bath, powers)
            plt.axis([0,max(t_bath)*1.2,0,max(powers)*1.2])
            plt.grid()
            plt.show()



    def fit_power(self, plot=True, save=False, xtweak=0.001, ytweak=1):
        def f2(x, K, T_c, n):
                return K*(T_c**n - x**n)

        xdata = [entry*xtweak for entry in self.t_bath]
        ydata = [entry*ytweak for entry in self.powers]

        plt.scatter(xdata, ydata)
        popt, pcov = curve_fit(f2, xdata, ydata)#, p0=(0,0,0))
        if plot:
            nxdata = np.arange(0, max(xdata)*1.2, 0.0001)
            plt.plot(nxdata, f2(nxdata, *popt), 'r-')
            plt.axis([0,max(xdata)*1.2,0,max(ydata)*1.2])
            plt.grid()
            plt.xlabel('T bath (mK)')
            plt.ylabel('P Joule (fW)')
```

```
            if save:
                plt.savefig(str(self.name) + '.png')
            plt.show()

        return popt

    def valid_entries(self):
        with open(str(self.name) + "_errors.txt", "a") as text_file:
            for idx in range(len(self.data)):
                if np.average(self.data[idx].P_TES) < 0 or self.data[idx].readout
                  <= 0:
                    self.data[idx].valid = False
                    print(f"entry: {idx} is not valid| file: {self.data[idx].name}|
                       super: {self.data[idx].superconducting} | readout:
                     {self.data[idx].readout} | powerAv: {np.average(self.data
                     [idx].P_TES)}"    , file=text_file)
                else:
                    self.data[idx].valid = True


    def analyse_IV(self, R_stray, debug=False):
        for entry in self.data:
            entry.R_stray = R_stray
            entry.calc_all()

        self.valid_entries()

        self.plot_power(plot=False)
        params = self.fit_power(save=True)

        with open(str(self.name) + ".txt", "w") as text_file:
            print(f"K: {params[0]} | n: {params[2]} | Tc: {params[1]}",
              file=text_file)

        if debug:
            for entry in self.data:
                entry.plotRaw()
                entry.plotIV()
                entry.plotPV()
                entry.plotPR()
```

# Appendix B

# read_qdp Function

```
import numpy as np
import pandas as pd
import re

def read_qdp(file_name, make_excel = False, x = 'bias', y = 'V fb (V)'):
    '''
    Accepts a raw sting literal for the filename and path.
    x and y are strings which name the first 2 columns, default to bias
    and V fb (V)
    If make_excel is set to True then excel file is created in the name
    of the input file.
    Return the a pandas dataframe with the data.
    '''
    #name first two cols after inputs
    cols = {0: x, 1: y}
    header = 0
    #precompile the regexes for use in loop
    col_nam_re = re.compile(r'la\sy(\d)\s(.+)')
    data_match_re = re.compile(r'^-?(\d.+)')

    with open(file_name) as file:
        for line in file:
            column_match = col_nam_re.match(line)
            if column_match:
                #remove any formatting from col names
                cols[int(column_match.group(1))] = re.sub(r'(\\d|\\u)',r' ',
                  column_match.group(2))
            else:
                #once data is matched, loop break and record header
                data_match = data_match_re.match(line)
                if data_match:
                    break
            header += 1
    #reads table with whitespace seperators (tab or space)


    df = pd.read_csv(file_name, sep='\s+', header=None, skiprows=header,
      index_col=False, comment='!')


    #make list of col names for df and number ones without labels
    col_names = []
    for i in range(len(df.columns)):
        try:
             col_names.append(cols[i])
        except:
            col_names.append(str(i))
        i += 1
```

```
    #set column names and sort by x vals
    df.columns = col_names
    df.sort_values(by=[x], inplace=True)

    if df.iloc[:,0][1] * df.iloc[:,1][1] < 0:
        df.iloc[:,1] *= -1

    if make_excel:
        #find name for excel sheet from end of path
        name_match = re.search(r'(\\?((?:.(?!\\))+)$)', file_name)
        name = re.sub(r'(\.qdp|\\)', r'', name_match.group(1))
        #write to file and save
        writer = pd.ExcelWriter(str(name)+'.xlsx')
        df.to_excel(writer,'Sheet1',index=False)
        writer.save()

    return df
```

# Bibliography


[1] "Spica/safari." [Online]. Available: https://www.sron.nl/missions-astrophysics/spica-safari

[2] [Online]. Available: https://spica-mission.org/instruments.html

[3] M. D. Audley, G. Lange, J. Gao, P. Khosropanah, R. Hijmering, and M. Ridder, "Optical performance of prototype horn-coupled tes bolometer arrays for safari," 07 2016, p. 991408.

[4] M. D. Audley, G. de Lange, L. Ferrari, J.-R. Gao, R. A. Hijmering, P. Khosropanah, M. Lindeman, and M. L. Ridder, "Performance of a low-noise test facility for the safari tes bolometer arrays," *Journal of Low Temperature Physics*, vol. 167, no. 3, pp. 208–213, May 2012. [Online]. Available: https://doi.org/10.1007/s10909-012-0492-z

[5] M. D. Audley, G. de Lange, J.-R. Gao, P. Khosropanah, R. Hijmering, M. Ridder, P. D. Mauskopf, D. Morozov, N. A. Trappe, and S. Doherty, "Optical performance of an ultra-sensitive horn-coupled transition-edge-sensor bolometer with hemispherical backshort in the far infrared," *Review of Scientific Instruments*, vol. 87, no. 4, p. 043103, 2016. [Online]. Available: https://doi.org/10.1063/1.4945302

[6] H. Onnes, "The resistance of pure mercury at helium temperatures," *Commun. Phys. Lab. Univ. Leiden*, vol. 12, 01 1911.

[7] J. Bardeen, L. N. Cooper, and J. R. Schrieffer, "Microscopic Theory of Superconductivity," *Physical Review*, vol. 106, pp. 162–164, Apr. 1957.

[8] K. Irwin and G. Hilton, "Transition-edge sensors." in *Enss C. (eds) Cryogenic Particle Detection. Topics in Applied Physics*. Berlin, Heidelberg: Springer, 7 2005, vol. 99.

[9] K. D. Irwin, G. C. Hilton, D. A. Wollman, and J. M. Martinis, "Thermal-response time of superconducting transition-edge microcalorimeters," *Journal of Applied Physics*, vol. 83, no. 8, pp. 3978–3985, 1998. [Online]. Available: https://doi.org/10.1063/1.367153

[10] ——, "X-ray detection using a superconducting transition-edge sensor microcalorimeter with electrothermal feedback," *Applied Physics Letters*, vol. 69, no. 13, pp. 1945–1947, 1996. [Online]. Available: https://doi.org/10.1063/1.117630

[11] "scipy.optimize.curve_fit¶." [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html