Brigham Young University
BYU ScholarsArchive

Theses and Dissertations

2019-05-01

Thermal Gradient Characterization and Control in Micro-Fabricated Gas Chromatography Systems

Austin Richard Foster
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

BYU ScholarsArchive Citation

Foster, Austin Richard, "Thermal Gradient Characterization and Control in Micro-Fabricated Gas Chromatography Systems" (2019). *Theses and Dissertations*. 7481.
https://scholarsarchive.byu.edu/etd/7481

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Thermal Gradient Characterization and Control in Micro-Fabricated Gas Chromatography Systems

Austin Richard Foster

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Brian D. Iverson, Chair
H. Dennis Tolley
Brian D. Jensen

Department of Mechanical Engineering

Brigham Young University

Copyright © 2019 Austin Richard Foster

All Rights Reserved

ABSTRACT

Thermal Gradient Characterization and Control in Micro-Fabricated Gas Chromatography Systems

Austin Richard Foster
Department of Mechanical Engineering, BYU
Master of Science

In order to make gas chromatography (GC) more widely accessible, considerable effort has been made in developing miniaturized GC systems. Thermal gradient gas chromatograpy (TGGC), one of the heating methods used in GC, has recieved attention over the years due to it's ability to enhance analyte focusing. The present work seeks to develop high performance miniaturized GC systems by combining miniaturized GC technology with thermal gradient control methods, creating miniaturized thermal gradient gas chromatography (µTGGC) systems. To aid in this development a thermal control system was developed and shown to successfully control various µTGGC systems. DAQ functionality was also included which allowed for the recording of temperature and power data for use in modeling applications. Thermal models of the various µTGGC systems were developed and validated against the recorded experiemental data. Thermal models were also used to aid in decisions required for the development of new µTGGC system designs. The results from the thermal models were then used to calibrate and validate a stochastic GC transport model. This transport model was then used to evaluate the effect of thermal gradient shape on GC separation performance.

Keywords: gas chromatography, micro-gas chromatography, thermal gradient gas chromatography, thermal control systems, CFD, thermal modeling, chromatographic modeling

ACKNOWLEDGMENTS

Although many individuals have helped me in completing the work reported in this thesis, I will only mention a few to be brief. First, I would like to thank Dr. Brian Iverson for all of the time he invested in meeting with me and helping me work through the numerous problems that were encountered throughout this project, as well as providing me with opportunities and encouragement to stretch my abilities and grow. I would also like to thank Dr. Dennis Tolley for the time he invested in helping me to refine the GC transport model. I would also like to express appreciation for the funding provided by Perkin Elmer which made this work possible. I am grateful to Dr. Abhijit Ghosh for teaching me how to run GC separations and for always having something interesting to discuss as we waited for separations to finish. I am also grateful for so many of my fellow students who have helped me in countless ways as I have worked on this project. Lastly I would like to thank my wife Chelsea for her patience and support as I spent long hours on campus trying to get all of this to work. Her friendship and positivity helped carry me through to the end.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

A	Empirical coefficient to the Chapman-Enskog theory (atm $\mathring{A}^2$cm$^2\sqrt{g/mol}$/K$^{3/2}$s)
$bits$	Bit data read in by the Arduino (bits)
C_f	Coefficient of friction
c_p	Spiral column aluminum specific heat (J/kgK)
D	Mass diffusivity (m^2/s)
D_h	Hydraulic diameter (m)
$Data_I$	Bitwise current data from Arduino (bits)
F	Air thermal conductivity slope (W/mK^2)
G	Air thermal conductivity intercept (W/mK)
h	Convection coefficient (W/m^{2}K)
h_b	Spiral system bottom surface convection coefficient (W/m^{2}K)
h_t	Spiral system top surface convection coefficient (W/m^{2}K)
I	Heater current (A)
ΔH	Enthalpy of evaporation (J)
k	Retention factor
k_{air}	Air thermal conductivity (W/mK)
k_{Al}	Aluminum thermal conductivity (W/mK)
k_S	Solimide thermal conductivity (W/mK)
L	Column length (m)
L_{air}	Air gap distance between column and insulation (m)
M_1	Molar mass of the first gas in a solution (kg/mol)
M_2	Molar mass of the second gas in a solution (kg/mol)
m_V	Slope of voltage decay (V/s)
p	Heater power (W)
P	Column absolute gas pressure (Pa)
P_i	Column inlet absolute gas pressure (Pa)
P_o	Column outlet absolute gas pressure (Pa)
P_x	Column absolute gas pressure at position x (Pa)
ΔP	Column pressure drop (Pa)
$\%P$	Percentage of source power consumed by heater
PWM	PWM control value ($0 \leq PWM \leq 255$)
PWM_c	PWM cutoff value where voltage decay begins to intersect next rise
q''_{cond}	Spiral column heat flux into insulation (W/m^2)
q''_{in}	Spiral column heater input flux (W/m^2)
q''_S	Heat flux entering Solimide insulation (W/m^2)
r	Spiral column radial coordinate (m)
R	Gas constant (J/molK)
R''_{air}	Thermal contact resistance from air gap between column and insulation (m^{2}K/W)
R_H	Heater resistance (Ω)
$R_{i,j}$	Resolution between analytes i and j
R_S	Solimide insulation radius (m)
Re_D	Diameter dependent Reynolds number
ΔS	Entropy of evaporation (J/K)

t	Time coordinate (s)
T	Temperature (K)
T_{air}	Air temperature in gap between column and insulation (K)
$T_{Experiment}$	Silicon column experimental temperature (°C)
T_{ext}	Extrapolated temperature at Solimide-copper interface (K)
T_i	Thermocouple temperature at position i (K)
$T_{Simulation}$	Silicon column simulated temperature (°C)
T_∞	Ambient temperature (K)
t_{base}	Time for voltage decay to reach baseline voltage (s)
t_{fall}	Time PWM voltage begins to fall (s)
t_R	Analyte retention time (s)
t_{rise}	Time of PWM voltage rise (s)
t_T	Total PWM cycle time
Δt	Incremental time step (s)
ΔT	Temperature difference (K)
$\bar{u}$	Column average mobile phase velocity (m/s)
u_x	Column mobile phase velocity at position x (m/s)
V_0	Baseline heater voltage (V)
V_f	Full voltage value (V)
$w_{h,i}$	Width at half max of analyte i (s)
x	Column lengthwise position (m)
x_1	Air gap starting length (m)
Δy	Thermocouple spacing (m)
z	Spiral column thickness (m)
β	Phase volume ratio
β_i	Regression parameters to heater and current resistance models ($i = 0,1,2,3$)
γ	Regression fit parameter for mass diffusivity model
μ	Helium mobile phase viscosity (Pa·s)
ρ	Spiral column aluminum density (kg/m^3)
σ_i	Standard deviation of the peak of analyte i (s)
σ_{12}^2	Average collision diameter (Å)
$\Delta\sigma^2$	Incremental change in analyte zone variance (m^2)
Ω	Chapman-Enskog collision integral

CHAPTER 1. INTRODUCTION

1.1 Gas Chromatography

Whether or not we are aware of it, our lives are significantly influenced by chemicals we come in contact with that we do not detect with our natural senses. Without the ability to detect these chemicals it would not be possible for us to control or understand the effects they can have on our lives. The field of analytical chemistry includes the development and use of methods and instruments to facilitate the detection and quantification of chemicals. One common method used by analytical chemists to perform these tasks is chromatography. Chromatography, as both a method and a technology, allows for the separation of mixed chemicals for purification and detection purposes [1]. As the capability to detect the presence of chemicals is desirable in numerous fields, chromatography is utilized in a variety of industries. Examples of common applications include food testing, pharmaceutical testing, defense applications, drug testing, forensics, and pollutant analysis in environmental sciences, to name just a few.

Numerous techniques exist that apply the principles of chromatography to separate chemical mixtures. One of these techniques, which uses gas flow to mobilize and separate chemicals, is appropriately termed gas chromatography (GC). The function and objective of GC is the separation of volatile and semi-volatile, primarily organic, compounds for detection. Many substances of interest in our day-to-day lives fall within this class of chemicals. One example that garners both positive and negative attention in our modern world is the fossil fuels that we use to power our cars and light our homes. GC is used by petroleum companies to analyze the purity of fossil fuels along with their suitability for use by the public. Another important example of GC's influence on our everyday lives is its use in checking for the presence of high levels of pesticides on produce sold in grocery stores. Clearly the capabilities made possible by GC serve to significantly improve our lives and are of great value.

Despite the advanced state of the field of chromatography, its inception is surprisingly recent. The principles of chromatography were first discovered in 1906 by the Russian botanist Mikhail Tsvett as a result of his efforts to separate the various pigments found in leaves. As chromatography was first used to separate various plant pigments, the method was given the name chromatography, meaning "writing color". Tsvett's early chromatographic devices consisted of a tube, referred to as a column, packed with an adsorbent material that was selected for its interactive nature with the plant pigments. A liquid was moved through this packed column to mobilize the pigments and allow for their separation. Following the publication of Tsvett's work in 1906, the field of chromatography experienced gradual but steady growth. Similar to the work of Tsvett, other early work in chromatography used packed column liquid chromatography to aid in the separation of compounds found in plant and animal tissues [2].

The concept of GC, which is distinguished by the use of a gas instead of a liquid as the carrier fluid, was first considered in the early 1950's. The first work in GC was performed by Archer J. P. Martin *et al.* in 1952 and was used for separating volatile fatty acids [3]. Following its initial development by Martin *et al.*, other researchers began quickly adopting the new technology for their own use [2]. Due to its quick adoption by other researchers, numerous advances were made to GC devices to improve the technology's capabilities. One of the more noteworthy advances in GC was the creation of the open tubular, or capillary, column by Marcel Golay in 1956, which came as a result of information gathered from his theoretical model of packed column behavior. The introduction of this advance in GC column technology led to improved separations compared to the packed columns invented by Tsvett 50 years earlier for liquid chromatography [4]. Due to the broad range of application, GC has advanced rapidly over the past seven decades and has developed into a large field of study within analytical chemistry.

Before continuing, a brief introduction to the components and mechanisms of operation of a GC device are in order to aid the reader in better understanding the research that follows. The gas carrier fluid used in GC is also referred to as the mobile phase and is most often helium, although in some cases hydrogen or nitrogen can be used. This carrier gas is passed through a long thin tube, referred to as the column, that is typically 10 - 60 m long with an inner diameter of 100 - 530 µm. Columns used in GC are primarily made of fused silica and, less commonly, stainless-steel. The inner surface of the column is coated with a thin layer of an absorbent polymer or liquid, which is

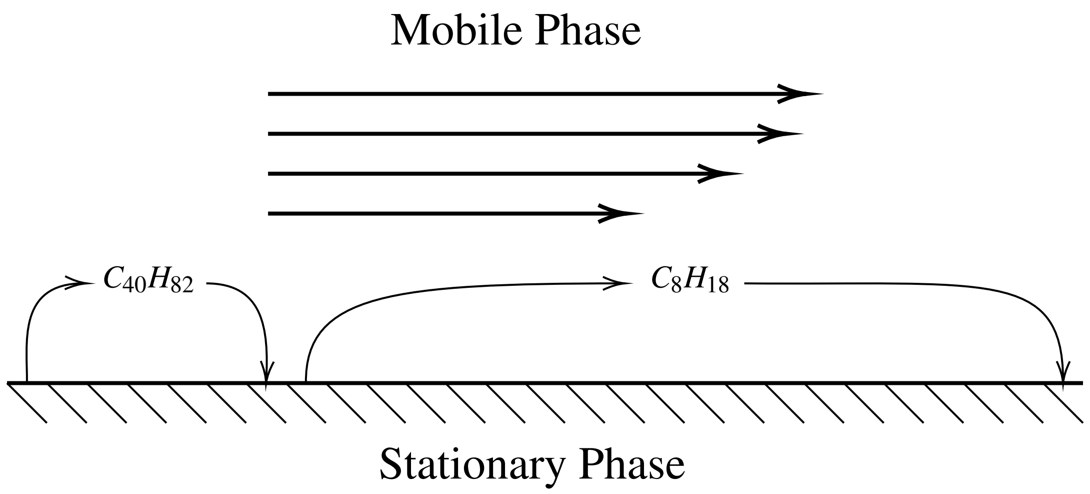


Figure 1.1: Diagram demonstrating how lighter, more volatile compounds like C8 move more quickly down the column due to less interactions with the stationary phase whereas heavier, less volatile compounds like C40 move more slowly down the column due to more interactions with the stationary phase.

referred to as the stationary phase. The driving mechanism behind GC separation is the successive sorption and desorption of a given chemical (or analyte) into the stationary phase and then back into the mobile phase. When a mixture of analytes is injected into the column, each analyte will absorb into the stationary phase to a different degree than other analytes. The analytes that spend more time in the mobile phase will move down the column more quickly, causing them to separate from the other, less volatile, analytes (Figure 1.1).

After an analyte has moved through the column, it is eluted (or exits) out the end of the column into a device that allows the analyte to be detected. A number of detector types exist in GC that provide different capabilities. The most common detection device is a flame ionization detector (FID) due to its reliability and simplicity. The basic principle behind an FID is that the separated chemical sample is mixed with hydrogen and an oxidizer (usually air) after elution and is then burned. The combusted gases then pass through a sensitive ammeter that is able to detect charge variations that result from ionized molecules passing through. As the chemicals separated via GC are required to be volatile or semi-volatile, and since FID detectors use combustion to measure the presence of analytes, conventional GC methods only allow for the detection of organic compounds. The resultant data from a GC separation is a chromatogram with time on the horizontal axis and signal intensity on the vertical axis as shown in Figure 1.2. In the chromatogram, each analyte

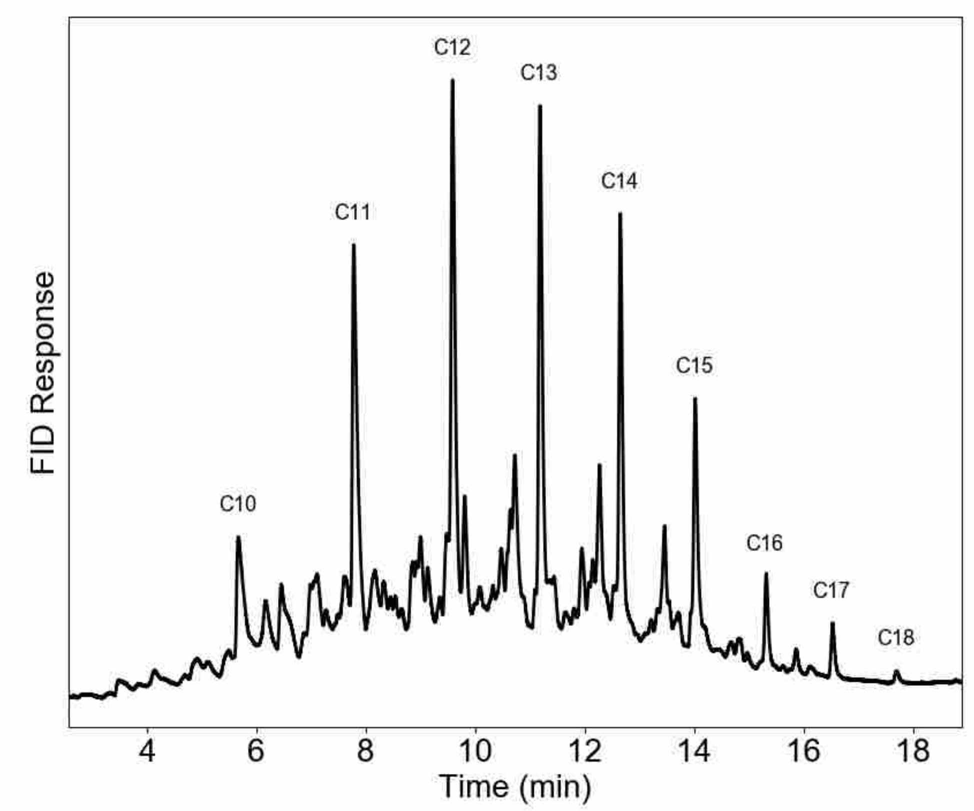


Figure 1.2: Chromatogram of kerosene separated in a stainless-steel micro-column using a temperature programmed gas chromatography control method.

exhibits an individual peak with defining features including elution time, peak width (measured in time units), and symmetry or shape of the peak.

The interaction rate between a given analyte and the stationary phase is a function of the chemical passing through the column and the material used for the stationary phase; it is also influenced by the temperature of the column. Just as a liquid will evaporate more quickly when placed on a hot surface, each chemical becomes more volatile and, therefore, interacts less with the stationary phase as the column temperature increases. This increase in volatility leads to an increase in the analyte's rate of motion along the column length. Temperature is the primary variable used to control GC separations as it strongly affects the analyte velocity. In order to achieve accurate and precise control of the column temperature, large, high performance convection ovens were developed and are now standard for performing GC separations (Figure 1.3).

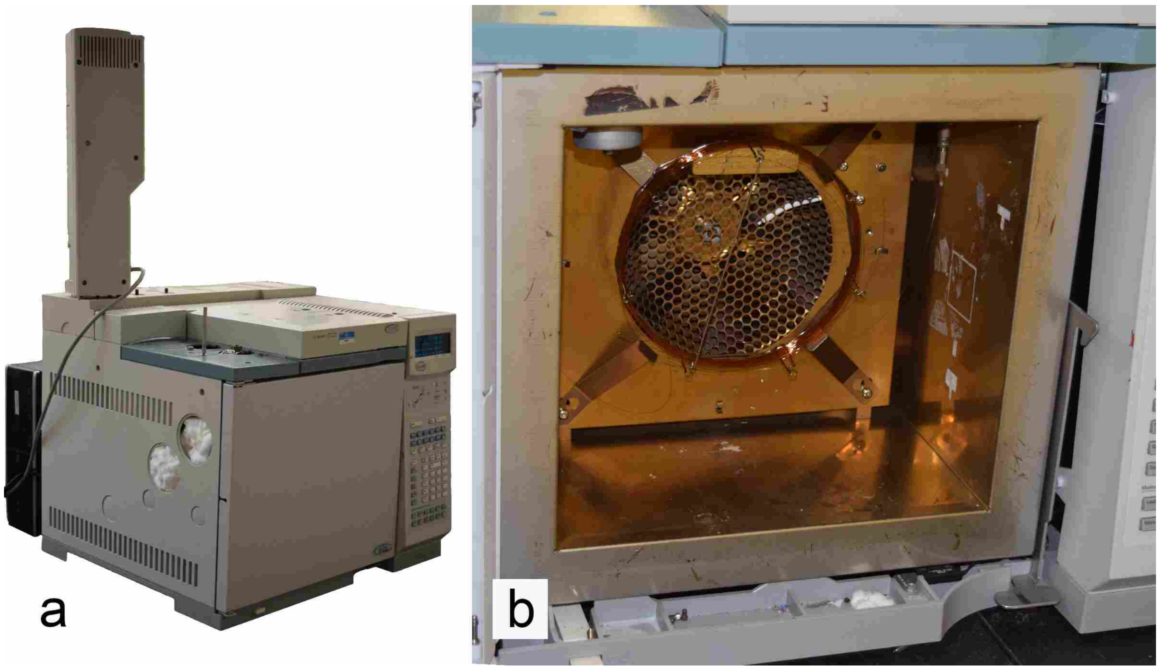


Figure 1.3: (a) The Agilent GC oven used for the work presented in this thesis. Note the computer tower behind the oven as a size reference. (b) A fused silica capillary column mounted inside the GC oven.

Initially, GC separations employed a constant uniform temperature along the length of the column for the entire analysis, a method referred to as isothermal gas chromatography (ITGC). The temperatures used in early GC runs were relatively low, often below 100 °C, due to the high volatility of the chemicals of interest [3]. However, ITGC presented a significant problem for chromatographers. The first analytes to elute form sharp peaks, but each subsequent peak would elute with an increasingly broader width that is spaced increasingly further from the preceding peak. This limitation made it difficult to separate and analyze mixtures with chemicals that span a broad range of volatilities. In order to address this issue, a new temperature control method was developed in which the column's temperature was increased throughout the separation run. This new method, termed temperature programmed gas chromatography (TPGC), offered the advantage of maintaining sharp peaks and repeatable elution times a wide range of volatilities.

Apart from the advances described, the nature of GC devices has undergone relatively little change. Despite the remarkable capabilities of modern GC ovens, one of their significant drawbacks is their lack of portability. This lack of portability is a result of both the size and weight of GC ovens as well as the power required to heat the large space inside. Since the 1970's an active field of research has been to develop methods that would allow for a reduction in size and increased portability. It is often desirable that chromatographic measurements be made on-site due to rapid sample degradation and the desire for immediate results. Two noteworthy approaches have emerged to address this problem of miniaturizing GC: (1) the application of microfabrication techniques to develop miniaturized GC columns and (2) the use of thermal gradients along the length of the GC column to aid in focusing analytes and increasing speed of analysis.

1.2 Column Miniaturization

Research into the development of miniaturized GC columns, referred to as micro gas chromatography (μGC), became an active field of research following the development of microfabrication techniques for integrated circuit production. The first work to create a μGC column was published in 1979 by Terry *et al.* where the column was etched in a silicon wafer [5]. Since that time, a majority of the work in μGC has focused on the use of silicon wafers as the column substrate [6–9]. Throughout the years, several other materials have emerged as substrate alternatives, including glass [10], ceramic [11], metal [12], and organic polymer [13]. One inherent limitation in the development of μGC columns in alternate substrate materials is that progress is dependent on available microfabrication processes. For this reason, silicon has remained the dominant substrate in the field of μGC, as significant advances have occurred in silicon microfabrication techniques for the field of microfluidics and the growing number of computational devices [14].

Although much progress has been made within the field of μGC, one difficulty it presents is its inherent limitations in column length. In traditional GC, devices with capillary columns that provide good separation performance are rarely shorter than 15 m. In μGC, it is common practice for columns to be in a serpentine, winding pattern in order to decrease its footprint. However, one limitation in μGC column fabrication is the tradeoff between column footprint size and column length. In using microfabrication techniques to produce μGC columns, an increase in the length of the column results in a corresponding increase in the column footprint. This limitation makes it impossible to place a column of a typical length ($\geq$15 m) on a substrate with an area less than 32 cm^2. Despite this limitation, research in the field of μGC has shown that although microfabricated columns cannot perform as well as conventional GC columns, they are able to successfully separate a wide array of volatile and semi-volatile compounds on columns with very small footprints.

Despite the non-ideal separation characteristics of μGC columns, repeated demonstration of μGC's success in miniaturizing GC columns indicates it could be a promising method for allowing for the miniaturization of GC devices.

1.3 Thermal Gradient Gas Chromatography

In order to capitalize on the miniaturized nature of μGC columns while avoiding the drawback of limited column lengths, a method for improving chromatographic separation over short distances is required. One such method is to apply a decreasing thermal gradient along the length of the column. This temperature control method, referred to as thermal gradient gas chromatography (TGGC), is intended to focus peaks and improve separation performance, especially when chromatographic conditions are not ideal, such as with shortened columns, poor injections, and stationary phase inconsistencies, to name a few.

The fundamental principles behind TGGC are the same as those for conventional GC. As mentioned, the various chemicals in a sample have different volatilities and absorptivites, and each compound's volatility increases with temperature. In other words, at higher temperatures the molecules of a chemical tend to interact less with the stationary phase causing them to spend more time in the flowing mobile phase and move through the column more quickly. In TGGC, a negative thermal gradient is applied along the column such that the column temperature decreases in the direction of flow. Consider the behavior of a single analyte injected into a column with a decreasing temperature gradient. Due to the high temperature at the inlet, the analyte tends to remain in the mobile phase causing the chemical to initially move rapidly along the column. However, as the analyte moves through the column, the temperature decreases due to the decreasing gradient along the column. This decrease in temperature causes a corresponding reduction in velocity as the molecules of the analyte begin to interact more frequently with the stationary phase. Assuming the temperature is continuously decreasing along the column, the temperature at the back of the analyte separation band will always be higher than the temperature near the front. The result of this temperature difference between the front and back of the analyte band is that the analytes near the back will move more quickly than those near the front leading to a focusing of the analyte band as illustrated in Figure 1.4. As each chemical has a distinct volatility and level of interaction with the stationary phase, the thermal gradient not only serves to focus the sample but also allows for

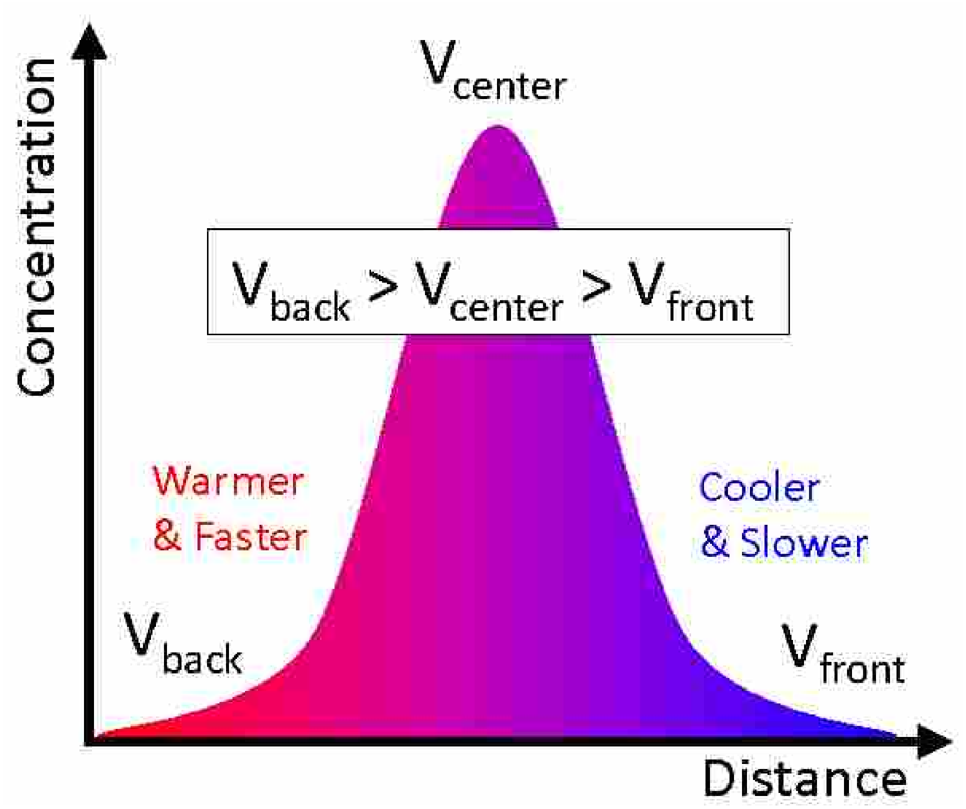


Figure 1.4: A graphical representation of the TGGC focusing effect.

separation of the various analytes in the mixture. Since these benefits are a function of the gradient and not the length of the column, TGGC provides the potential of improved chromatographic performance in shorter columns.

The use of thermal gradients for focusing analyte bands was first reported by Zhukhovitskii *et al.* in 1951 [15]. These initial experiments used a packed column and a mobile heater that could be moved along the column length. After a sample was injected into the column, the heater was moved back and forth repeatedly along the column with the heater increasing in temperature with each pass in order to elute individual chemicals in the mixture. For nearly two decades following these initial experiments other published TGGC work used a similar mobile heating method as it was believed that only moving gradients were valuable in separating chemical mixtures [16]. This remained the case until Fatscher and Vergnaud published the first work demonstrating that a stationary gradient could successfully separate analytes [17]. Throughout the 1970's the focus in TGGC research shifted from mobile heaters on packed columns to stationary gradients on capillary columns [18,19]. In 1997, a controversy was published by Blumberg refuting the conclusions of Jain and Phillips and claiming that his theoretical model of TGGC separation proves that the resolution of TGGC cannot outperform that of ITGC [20–23]. Despite this prediction work has since been published that demonstrates better performance from TGGC than both ITGC and TPGC [24]. The reason for this apparent discrepancy is that Blumberg's model assumes idealized conditions, namely a perfectly focused injection, and a stationary phase with constant properties which are impossible to achieve. Despite this published controversy, the work of many researchers has demonstrated that TGGC provides adequate focusing to correct for non-ideal GC conditions. Of particular note for the present work are studies that have demonstrated TGGC's capability to successfully separate mixtures on columns with an average length of 2.1 m [20–22, 24–28].

TGGC provides the capability of improving separation performance for short columns. Since one of the primary drawbacks for µGC is column length limitations, TGGC presents itself as a viable option for improving the performance of µGC columns. Over the past five years, work at Brigham Young University (BYU) has been conducted to develop µGC devices using thermal gradients to optimize their separation performance. In 2014, Wang *et al.* published work where a 1.4 m serpentine column was run under both TPGC and TGGC conditions. A mixture of non-polar and polar analytes was analyzed on the column (polar compounds have a tendency to elute with poor peak shape). Results demonstrated that TGGC is able to improve peak shape for both polar and non-polar compounds, with the more dramatic improvement being seen for the polar compounds. This work is significant in that it provides evidence that TGGC helps to improve separations using µGC columns [27]. In 2017, Ghosh *et al.* published a report on the use of TGGC with a 5.9 m long column etched in silicon and mounted to a heater/clamp assembly. One of the primary goals of the work was developing new methods of interfacing with µGC columns to allow higher maximum temperatures for the elution of less volatile compounds. In this report, *n*-alkanes up to tetracontane (C40) were eluted, an impressive achievement in a microfabricated thermal gradient gas chromatograpy (µTGGC) system [29]. This work also supports the conclusion of Wang *et al.* that TGGC serves to improve peak shape when compared to TPGC.

1.3.1 Theoretical Treatments of TGGC

An important supporting area within GC is the development of theoretical models to predict the behavior of GC devices. As discussed previously, the development of a theoretical model for GC in packed columns led Golay to recognize the potential value of using an open tubular column [30]. Thus, a correct theoretical model can not only serve to aid in understanding the performance of an existing system, but also guide future developments. For both ITGC and TPGC methods, theoretical models are well understood [31,32], however, the work that has been done to model TGGC performance has been limited at best, as will be shown.

Essentially, the driving mechanisms for TGGC separations are a particular case of those for ITGC, thus much of the foundation for modeling TGGC was laid by the work done in developing ITGC theoretical models [33–35]. Of particular note, the work of Giddings explored the effects of minor deviations from uniform ITGC conditions [36]. The first two theoretical treatments of TGGC reported in the literature were by Rowan in 1979, and Duarte and McCoy in 1982. Results of these works deal primarily with the influence of positive instead of negative thermal gradients [37,38]. It was not until 1992 that the first theoretical comparison between ITGC and TGGC using negative thermal gradients was performed [39–42]. In their work, Blumberg *et al.* developed a model of analyte behavior based on the mass diffusion equation and concluded that TGGC could never exceed ITGC in performance. However, it is noteworthy that the model developed by Blumberg *et al.* required that the conditions for separation be ideal; i.e. the injected sample was required to have zero width and a finite area (a Dirac delta function), a linear gradient along the column, and constant stationary phase parameters, which is unrealistic in all cases. In 2014, Tolley *et al.* published an approach to solving for the position, separation and concentration of GC mixtures which allows for non-idealized conditions not capable in the Blumberg model [43]. Tolley *et al.*'s approach, referred to in this work as the transport model, is unique because it employs stochastic methods computing simple equations for a large number of molecules instead of solving complex and limiting differential equations. Tolley *et al.*'s 2014 publication provides preliminary results of the transport model along with qualitative comparisons to experimental ITGC separation results obtained by Wang *et al.* [27]. In order to validate the transport model, however, direct, quantitative comparisons must be made between the transport model and GC separation data from experiments. Once validated, a transport model of this kind would serve as a useful tool in studying various TGGC systems and thermal conditions that could be used to enhance TGGC performance.

1.4 Thesis Objectives

Considering the potential advantages of µGC and TGGC, the combination of these two fields offers promising capabilities in creating a portable GC system. µGC research has shown that microfabricated GC columns can serve as functional replacements for conventional capillary columns and require a significantly smaller footprint. Work in the field of TGGC has repeatedly demonstrated that thermal gradients are able to significantly improve chromatographic performance for short columns as well as correct for other non-ideal conditions [20–22, 24–28]. A thorough investigation of approaches that combine both of these fields is absent in the literature. The present work seeks to understand and explore the capabilities achievable by combining these two technologies. This work is also intended to validate the stochastic transport model developed by Tolley *et al.* using experimental results and then use this transport model to study the potential of µTGGC devices.

In light of the problem under consideration and the insight gained from preceding work, the objectives for the current research are as follows.

1. Physical System Development: Develop an adaptable thermal control and data acquisition system capable of recording temperature and power consumption data for various µTGGC systems.
2. Thermal Model Development: Develop and validate computer simulations of thermal conditions for experiments performed on µTGGC systems using experimental data from the first objective.
3. Transport Model Validation and Application: Use chromatographic and thermal model data to calibrate and validate a GC transport model and use the validated GC transport model to compare performance between various modes of TGGC.

This work represents the first fully validated model of a µTGGC system. It is a significant advancement in the fields of µGC and TGGC specifically as well as in the field of GC generally. It both sets the stage for accurately analyzing GC performance in non-ideal GC systems as well as permits the exploration of potential advances in GC system design via computational means.

CHAPTER 2. EXPERIMENTAL SYSTEMS

2.1 Preceding Work

Previous iterations of an experimental thermal gradient system at BYU involved the use of a micro-machined column in silicon that was interfaced with a custom heating and clamping mechanism (Figure 2.1). A basic control system had been developed to establish and ramp the thermal gradient along the chip. Although the system was shown to provide successful separation of chemical mixtures, it was difficult to control and did not permit automated data collection of temperature and power for post-run analysis.

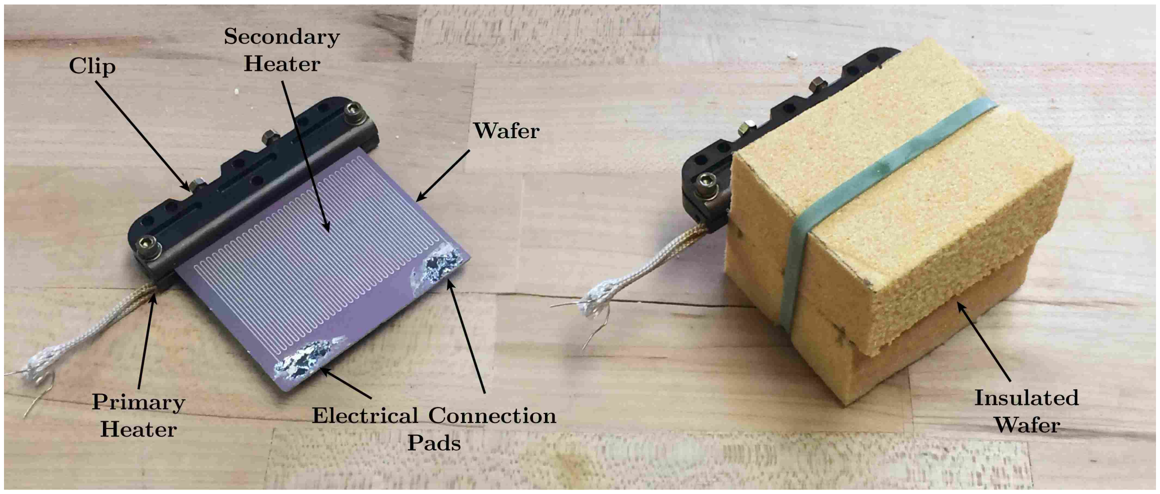


Figure 2.1: Photo of previous iteration of the μTGGC system both with and without insulation.

In light of the shortcomings inherent in the control system of this previous iteration, and given the apparent needs for this work, it was determined that a new thermal and power control system with data acquisition (DAQ) functionality would be developed. As it was understood that the column shape and conditions used in experiments would be subject to change, it was clear that a versatile control system that could be applied to various designs would need to be developed. To allow for reliable control of the thermal gradient, the controller would need to accurately control multiple heaters simultaneously while reliably rejecting input disturbances. Accurate temperature and power consumption data acquisition would need to be included in the control system to enable thermal and transport modeling of the experimental conditions. Finally, to ensure that this system would be user friendly and the data would be accessible, a PC user interface would need to be developed to integrate these functions.

All chromatographic data was recorded by interfacing the thermal gradient column with an Agilent 6890 GC oven (Figure 1.3). Helium was selected for the carrier gas in all GC runs and analytes were detected using a flame ionization detector (FID). Although parameters were occasionally changed to suit the needs of a specific run, the following parameters were generally used for the GC separations. The carrier gas flow rate was set to 1 mL/min; the GC oven would automatically adjust the inlet pressure to provide the desired column flow. The injection port where the chemical mixtures are introduced into the column and detector on the oven were both set to 250 °C for most runs. The split ratio employed was usually between 5:1 and 20:1 depending on the concentration of analytes in the sample being used, and in all cases, 1 µL of the sample mixture was injected for each run.

As the overarching goal of the work by the TGGC group at BYU is to improve the performance of a µTGGC system, multiple designs were considered throughout the span of the project. In improving a µTGGC system design, there are multiple competing requirements that must be balanced in order to optimize GC separation performance. Some of these competing requirements include column/analyte interaction behavior, ease of manufacture, oven interfacing methods, and thermal transport properties of column material, to name a few. The interplay of these requirements led to the development of several µGC column designs as outlined in the following section.

2.2 µTGGC System Design and Components

2.2.1 Serpentine Column in Silicon

Prior to the start of the present work a µTGGC system etched in a silicon wafer had been developed by students in the BYU Electrical and Computer Engineering Department. The column is fabricated by etching a serpentine column into a 500 µm thick Si wafer following a layout similar to that shown in Figure 2.2. Diffusion bonding was then used to attach another 500 μm thick wafer on top of the etched wafer, forming an approximately 5.8 m long channel with a rectangular cross section (158 x 80 μm). The completed chip was baked in an oven to allow the surface of the silicon to oxidize making it more chemically inert and electrically insulated.

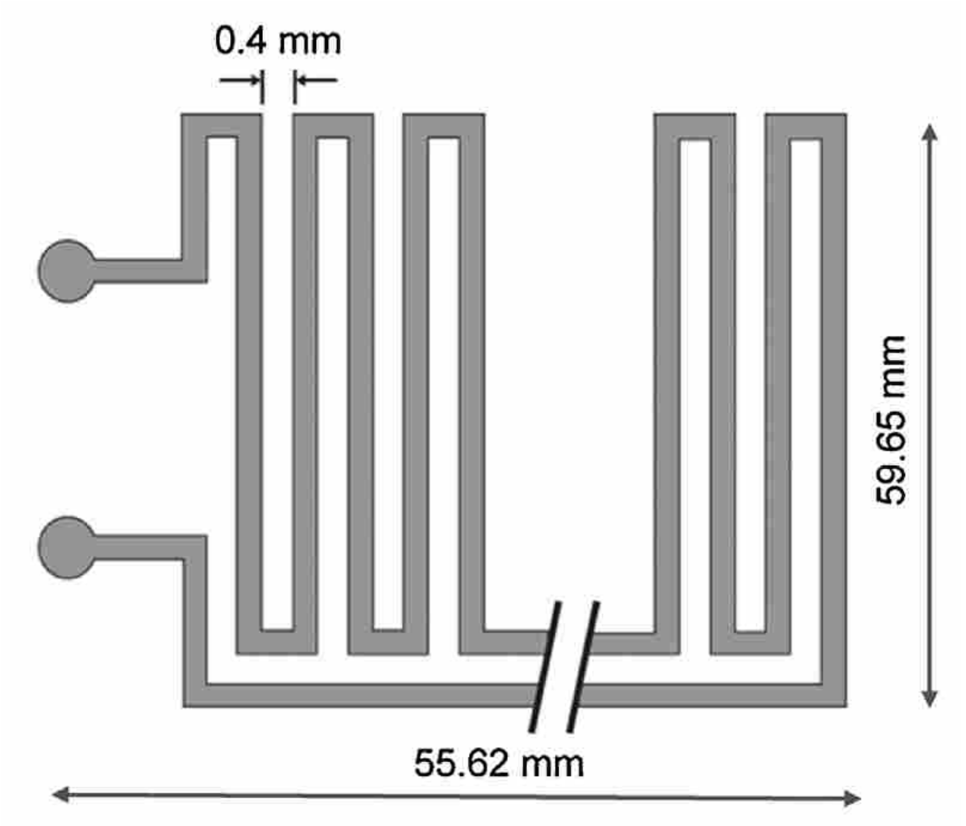


Figure 2.2: Dimensions of serpentine channel etched into the silicon wafer.

Initial designs for heating employed a clamping mechanism with a cartridge heater that relied on mechanical pressure and a polyimide gasket to interface the column and transfer lines. Subsequent designs have used a polyimide resin to connect the transfer lines directly into the column from the edge of the chip. Connection to the column from the side in this way enabled a flat, open surface for heater attachment. Silk-screened, serpentine heaters using an electrically conductive silver paste (ESL 599-E) that was cured using a temperature ramp method between 125 °C and 450 °C over a span of 25.5 minutes was used to achieve thin-film, resistive heating directly on the wafer (Figure 2.3).

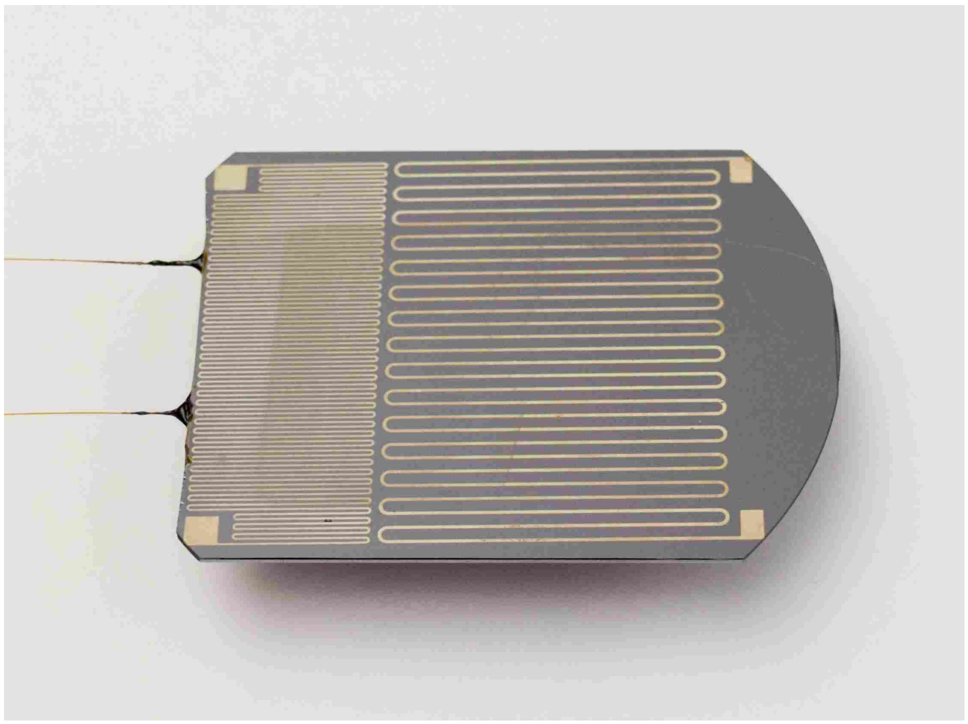


Figure 2.3: Photo of a finished silicon chip column with side transfer line connections.

A static coating method was developed to prepare columns with complex geometries. The method consists of connecting the inlet transfer line to a syringe pump that has been filled with a stationary phase solution containing 1% vinyl, 5% phenyl, 94% methylpolysiloxane (SE-54) and 1% dicumyl peroxide (as a crosslinking agent) in n-pentane (weight for weight) that has been degassed using sonication. The outlet end of the chip is left open to ambient air pressure while the column is slowly filled with the mixture. Once the stationary phase solution reached the end of the outlet transfer line, the outlet transfer line was closed with an RTV silicone sealant and capped with a silicone septum. To prevent any gas bubbles from damaging the coating, the filled column was pressurized using He at 689 kPa for 1 hour. The inlet of the column was then attached to a vacuum pump to evaporate the solvent and allow the static coating to remain. The stationary phase was cured by placing the Si column in a GC oven at 250 °C overnight. Following stationary phase curing, the column was conditioned by baking the chip at 350 °C for several hours. This treatment helped to lower the baseline signal from the FID and improve peak shape [29].

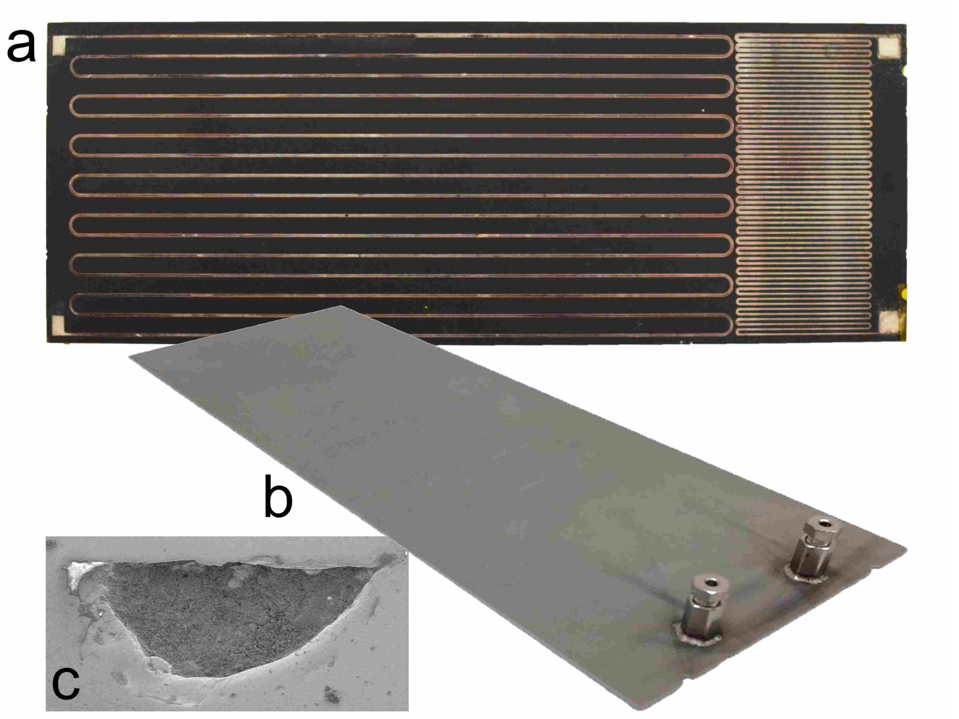


Figure 2.4: (a) The back side of the column with the ceramic coating and surface heaters added, (b) the stainless-steel column with the fitting attachments added before being coated with ceramic, (c) an SEM image of the column cross section.

2.2.2 Serpentine Column in Stainless-Steel

A second µGC column employed in the research was a stainless-steel column with an etched channel in a serpentine arrangement. The stainless-steel columns were produced by VACCO Industries using a proprietary method of stacking numerous thin layers of stainless-steel and fusing them together via diffusion bonding. Several of the layers included in the stack were bonded together and a semicircular serpentine trench with a radius of 57 µm (See Figure 2.4) was etched into them, forming a 9.6 m long channel. Inlet and outlet holes were also etched into the stainless-steel stack; fittings were brazed coincident with these holes for attachment of transfer lines. The addition of fittings allowed for a simple interface using a ferrule and nut so the transfer line could be removed and replaced as needed. The exterior of the bonded stainless-steel stack was coated in an electrically insulating ceramic coating (Cerakote C-7600 Glacier Black at Full Blown Coatings in Sandy, UT) to allow thin-film heaters to be silk-screened to the back of the column using the same method described above for heating serpentine columns in Si. The columns were then deactivated by Silcotek using a proprietary silicon coating called SilcoNert 2000. After deactivation, the column was coated with a stationary phase using the same method described above for silicon columns.

2.2.3 Spiral Column in Aluminum

A third μGC column employed a spiral column design in which the column spiraled radially outward in order to capitalize on radial conduction to achieve a smooth column gradient. The system consisted of a 0.51 mm thick aluminum disk with a diameter of 152.4 mm (6 in). A spiral groove (0.44 mm wide and 0.38 mm deep) was machined by the BYU Precision Machining Laboratory into the aluminum plate starting at a radius of 5.8 mm and ending at a radius of 40.5 mm. The channel spiraled out 1 mm every 2π radians, resulting in a column length of approximately 4.8 m. After machining the spiral, the chip was coated on its back side with the same electrically insulating ceramic coating used on the stainless-steel column described above. A circular heater was then silk-screened onto the back of the column using the same methods as described for the serpentine columns in silicon and stainless-steel. A stainless-steel, open tubular column (pre-coated with a stationary phase) from Restek (4.8 m x 100 m) was then placed in the spiral groove and held in place using Kapton tape (Figure 2.5).

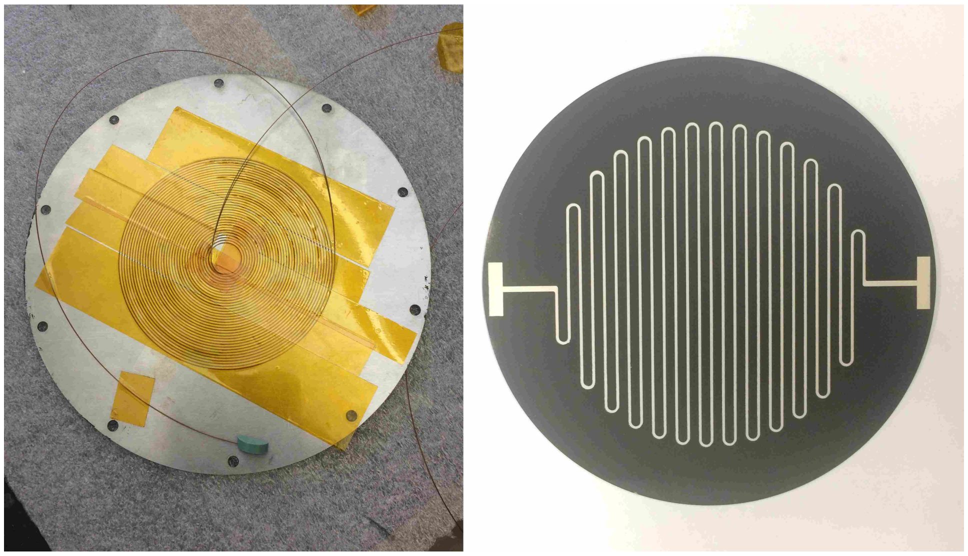


Figure 2.5: Images of the stainless steel capillary column in the spiral groove of the Aluminum disk and the ceramic coated backside of the disk with a printed surface heater.

2.2.4 Transfer Lines and Insulation

For each microfabricated system discussed above, the transfer lines between the system and the GC injector and detector on a commercial GC system should be held isothermal to ensure that the column separation characteristics are preserved. Isothermal transfer lines were achieved using lengths of coiled nichrome wire coated in an electrically insulative e-glass (electrical grade glass) coating. In some cases, the transfer line was held within a thin copper tube which was subsequently coiled in the same nichrome resistive wire (Figure 2.6). The addition of the copper tube decreased temperature variations along the length of the transfer lines. In order for each system to operate at elevated temperatures (up to 300 °C) for extended periods of time, an insulating polyimide foam (Solimide HT-340) was selected.

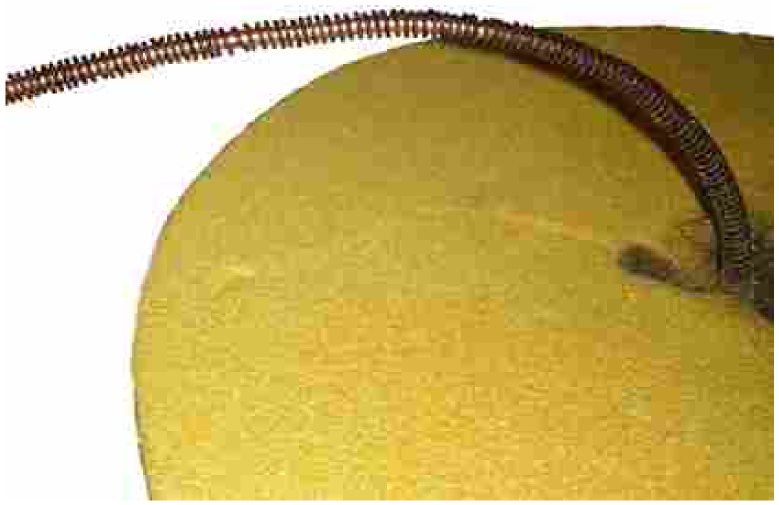


Figure 2.6: A photo of the nichrome wire wrapped around copper tubing. Note that for many of the experiments only nichrome wire was used to heat the transfer lines with no copper in between.

2.3 DAQ and Controller

2.3.1 Requirements

Given that several TGGC physical systems would be tested, control and data acquisition systems needed to be highly adaptable to ensure they could operate with the different μGC systems.

For this reason, the system needed to allow for modular attachment of the components that were controlled (cartridge heaters, resistive wire heaters, surface heaters, and cooling fan). The design of the control system code also had to be such that it could accommodate control of various µGC system arrangements.

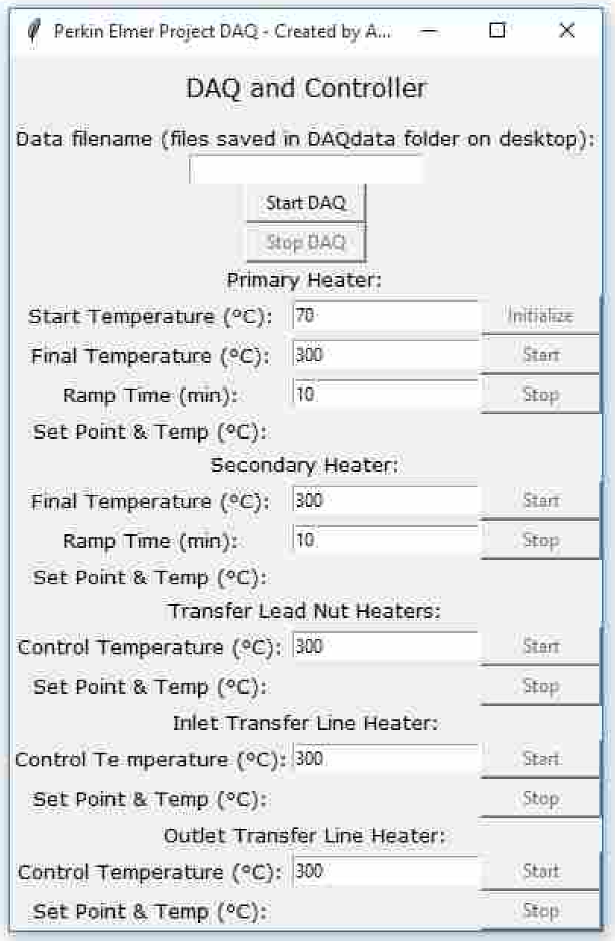
The control system also needed to control various components simultaneously, which requires that the system read data from multiple feedback thermocouples and implement control algorithms at a sufficient speed for accurate control. The user would also need the capability of changing set points for the various components without having to interrupt the control algorithm, and allow rapid, live communication with the hardware.

In addition to the feedback thermocouples, data on three thermocouples and heater power dissipation was to be recorded. This required that the system be able to read and record data from more thermocouples than just those used for the feedback control of the various heating and cooling elements. Similarly, the system needed to be capable of gathering data on the power consumed by each heating/cooling element in the system. All of the temperature data collected from the hardware had to then be communicated back to the user interface software and saved in a data file so they could be accessed for post GC run analysis.

2.3.2 DAQ and Controller System

The system was controlled by an Arduino Mega microcontroller as it allowed for analog data collection from up to 16 inputs simultaneously. Python was used for developing the graphical user interface (GUI) that allowed for control of the various heating and cooling elements (Figure 2.7). Serial communication was used to send user control values to the Arduino and gather data sent by the Arduino. Data was saved as a comma separated value file under a user specified name and location. The Arduino code consisted of functions allowing for reliable communication with the python script, proportional integral (PI) controllers, and data acquisition and processing routines (See Appendix A for code).

In order to gather temperature data, K-type thermocouples were connected to Adafruit AD8495 thermocouple amplifiers. To minimize wire clutter, a board was produced with an array of plugins for up to 16 thermocouples and connected to the Arduino via a ribbon cable as shown in Figure 2.9 (Appendix B).


Perkin Elmer Project DAQ - Created by A...

DAQ and Controller

Data filename (files saved in DAQdata folder on desktop):

Primary Heater:

Start Temperature (°C): 70

Final Temperature (°C): 300

Ramp Time (min): 10

Set Point & Temp (°C):

Secondary Heater:

Final Temperature (°C): 300

Ramp Time (min): 10

Set Point & Temp (°C):

Transfer Lead Nut Heaters:

Control Temperature (°C): 300

Set Point & Temp (°C):

Inlet Transfer Line Heater:

Control Te mperature (°C): 300

Set Point & Temp (°C):

Outlet Transfer Line Heater:

Control Temperature (°C): 300

Set Point & Temp (°C):

Figure 2.7: Screenshot of the Python GUI used to operate the thermal control system.

The component control circuitry was more complex as it needed to both control the connected components as well as gather data on power consumption for each of the components. The control circuitry consisted of a DC/DC solid state relay (SSR) connected to a power source and controlled via a pulse width modulated (PWM) signal running at 120 Hz from the Arduino. The PI controller implemented on the Arduino was shown to match given set point temperatures within ± 3 °C (Figure 2.8). It was determined that the best method for gathering power data for each component was measuring the current and voltage drop for each component. These voltage and current data could then be used in conjunction with the control PWM inputs to determine the average power consumed by each component as a function of time. The voltage data circuitry consisted of a voltage divider with resistors selected such that the voltage fed to the Arduino never exceeded 5 V (the Arduino's analog-to-digital converter's maximum capability). The current data circuitry consisted of a shunt resistor with voltage dividers on each end of the resistor. The voltages from these dividers are fed to a differential amplifier that was tuned to amplify the difference between the voltages so that sufficient resolution could be achieved over the Arduino's 5 V analog-to-digital range (Figure 2.9). In post processing, this difference could then be calibrated to the current flowing through the component, or could be computed directly using the resistance of the shunt resistor (Appendix B). As the signal through each component's circuitry was affected by the PWM signal opening and closing the SSR (which at times experienced aliasing with the DAQ reading frequency), an algorithm was developed and implemented on the Arduino to select the true voltage or current value from data with high levels of noise (Appendix C).

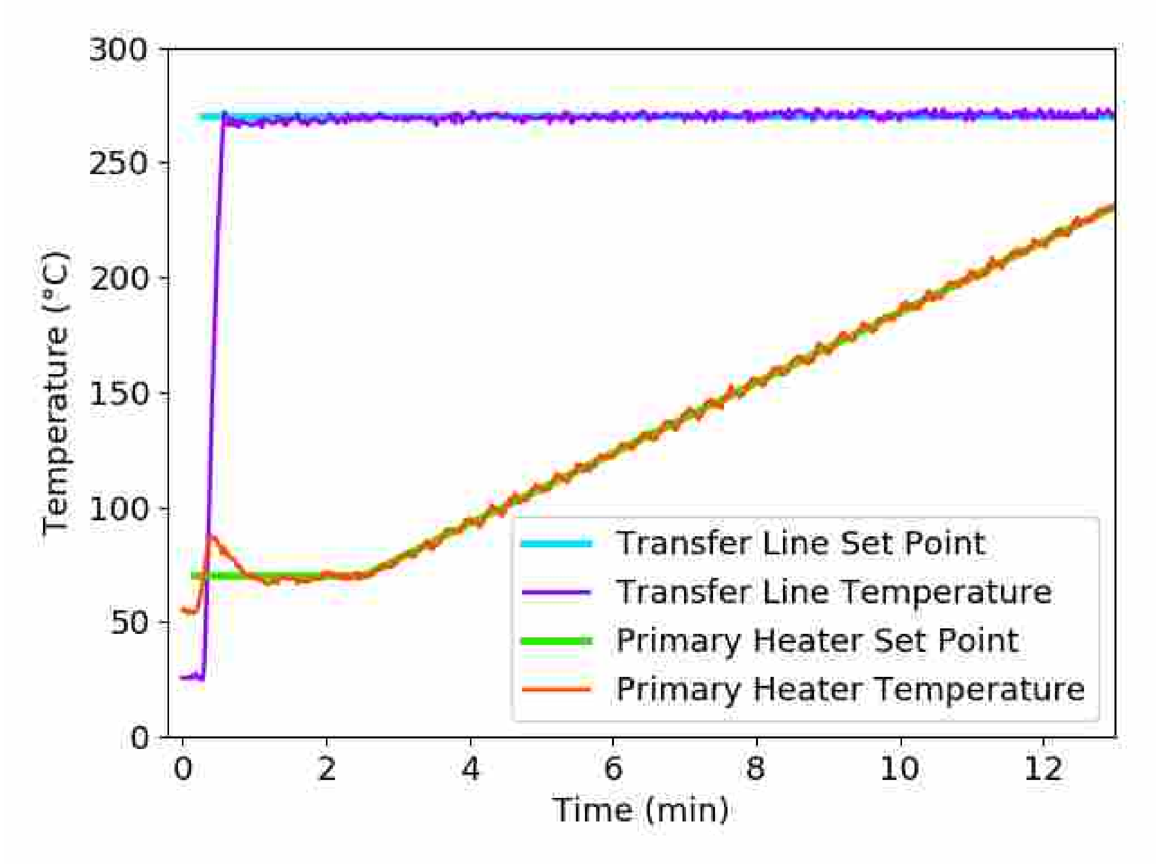


Figure 2.8: Demonstration of controller following two input set points simultaneously. Plotted data corresponds to the chromatogram shown in Figure D.1.

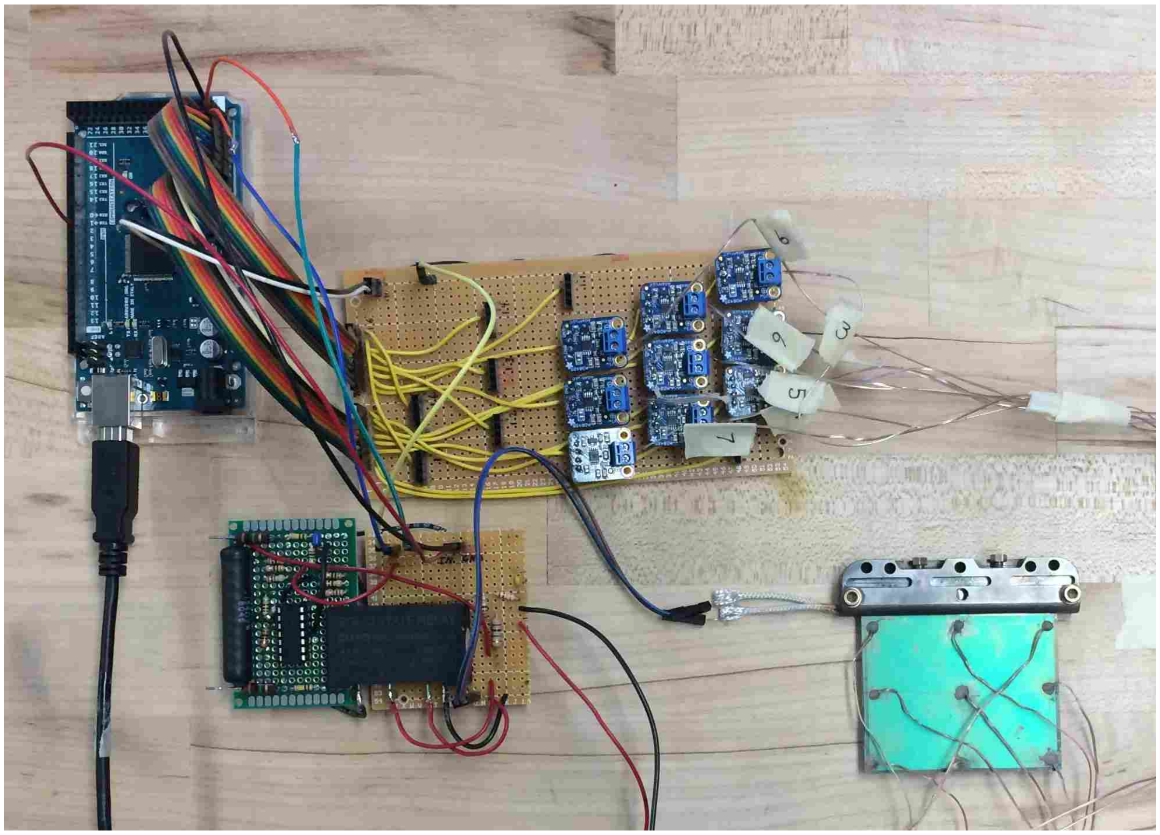


Figure 2.9: The colteroller/DAQ system connected to an early TGGC system. The large protoboard in the center is the thermocouple connection circuitry and the board on the bottom left is the relay and power data acquisition circuitry.

For the spiral column system (Section 2.2.3), it was determined that the ability to cool the edges of the plate would be helpful in establishing a steeper thermal gradient. A custom air duct was designed and 3D printed to direct air flow from a 120 mm square 24 volt computer fan to the edges of the aluminum disk (Figure 2.10). A circuit was also implemented between the power control board and the fan that stepped the voltage down from the power supply input voltage (approximately 35 V) down to 24 V which was the fan's maximum voltage rating.

2.3.3 Data Collection

The thermal control and DAQ systems gathered data and communicated them back to the user interface to be saved in a CSV file. In addition to time, the DAQ system reported the following data for each of the components in the system: feedback temperature (bit), temperature set point (°C), control PWM value (unitless, out of 255), voltage (bit), and current (bit). Functionality was

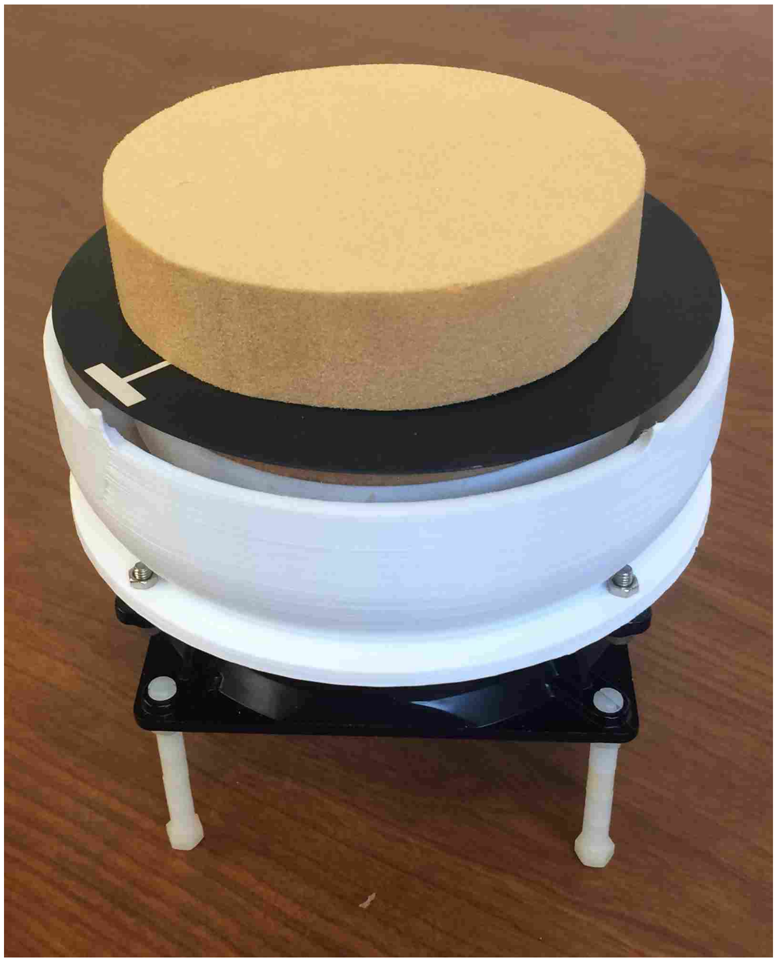


Figure 2.10: Assembled spiral system atop the fan duct and cooling fan.

added to the Arduino and Python code to allow for the addition of more thermocouples that were not associated with a controller. Data for these additional thermocouples was also reported in units of bits. As seen from the units above, all but the temperature set point had to be converted to new units to be usable in post processing. Collection of raw data in this form did not affect the Arduino's performance.

The data from the feedback thermocouples and additional thermocouples were reported in bit values between 0 and 1023. These values corresponded to the maximum and minimum voltages the Arduino analog-to-digital converter was able to process (5 V and 0 V, respectively). In order to convert these data to a useable temperature scale, the data were transformed into their voltage values and then converted using a function provided by the Adafruit thermocouple amplifiers

(Equation 2.1) [44]. Benchmark tests were run in an ice bath and boiling water to confirm that the conversion was accurate.

$$T(^{\circ}C) = \frac{5\left(\frac{bits}{1023}\right) - 1.25}{0.005} \tag{2.1}$$

Data for voltage and current were recorded in Arduino bit values from the µTGGC system and converted using a linear regression to the voltage and current consumed by each of the heaters. Once a calibration for these values was found, the total maximum power consumed by the heater was computed and then scaled using PWM control values. Using an oscilloscope, it was found that the voltage response of the SSRs experienced linear decay when the control signal dropped to zero. To find the average power as a function of the PWM input, the energy was found for one period of the PWM signal by integrating power over one PWM signal. That energy value was then divided by the period time of the PWM to find average power as a function of PWM value, voltage, and current. By using the data on heater power consumption in conjunction with the computed PWM power percentage a 95% confidence interval for the estimate of the mean power consumption of the µTGGC system was computed and used to check the power used in the thermal simulations (See Appendix E for notes on heater power consumption).

Along with the data provided by the DAQ, the chromatographic separation was recorded using software for the Agilent GC oven. The software recorded time (min) and GC signal strength (picoamps) as a CSV file for use in post processing.

2.4 Experimental Operation

For each of the µTGGC systems described above, numerous separations were run in order to refine the thermal system and improve performance. Sample results from each of the systems are shown below to demonstrate their capabilities.

The first µTGGC device on which the controller/DAQ was used was a replica of the µTGGC used by Ghosh *et al.* but with no column (Figure 2.11) [29]. The purpose of this system was to be a benchmark for developing the thermal modeling capabilities required for future systems. The system was also important as it was a valuable intermediate stage in the development and refinement of the controller/DAQ system implemented on future µTGGC systems. In this simplified experi-

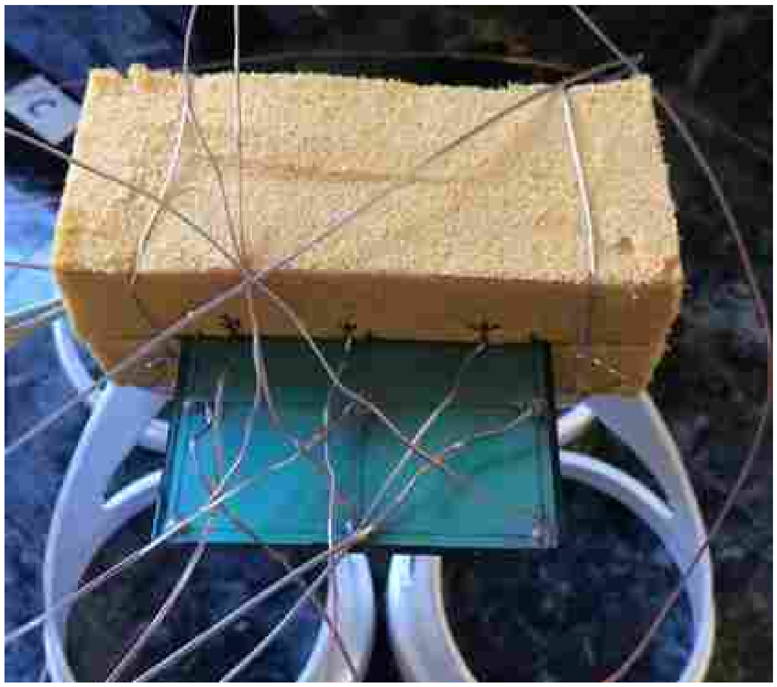


Figure 2.11: Replica experimental system partially insulated with thermocouples attached to the silicon surface.

mental system, a silicon wafer with no internal column was placed in the clamp housing and nine thermocouples were bonded to the chip's surface. As the wafer did not have an internal column, no chromatographic separations were run on this system. However, the thermal data extracted from the system were useful in advancing the controller/DAQ and thermal modeling.

The best separation performance for columns etched in Si was obtained using the previous heater/clamp design with a cartridge heater to establish the gradient (Figure 2.1). The interfacing method was via side connected transfer lines as described in section 2.2.1. The system was found to be unable to separate compounds heavier than C30 and the peak quality deteriorated with increasing analyte mass. This limitation in separating heavier analytes was likely due to the presence of cold spots along the transfer lines which had a significant negative influence on peak shape and separation performance. An additional chromatogram for this configuration is found in Appendix D.

Serpentine columns created in stainless-steel were pursued to alleviate challenges in interfacing with transfer lines. As will be discussed in Chapter 4, experiments performed with stainless-steel columns were used as calibration data for the GC transport model. These data were gathered

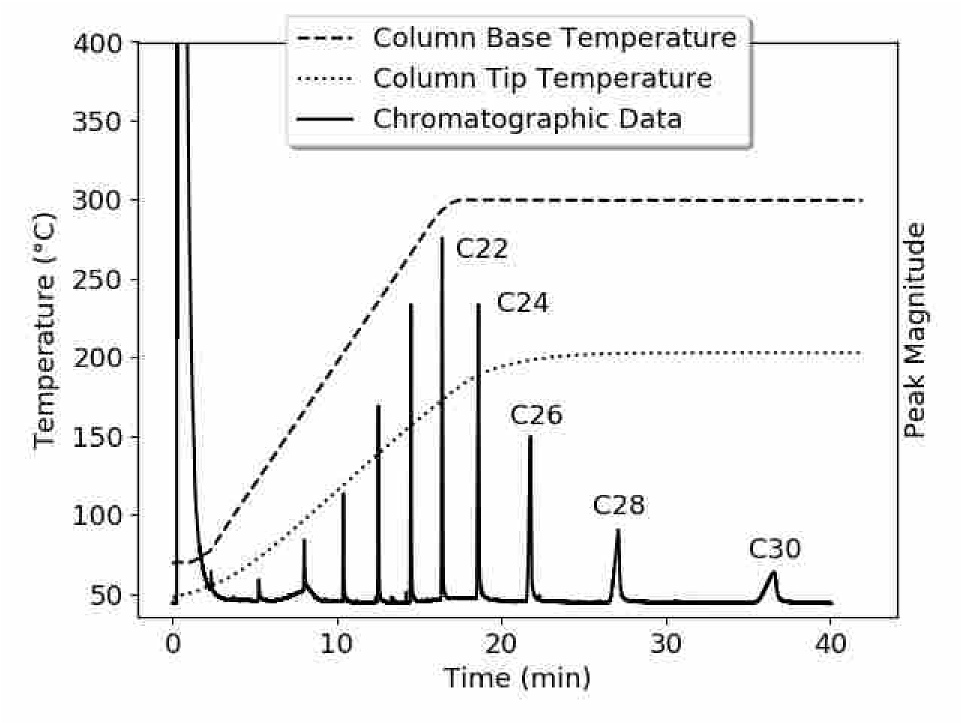


Figure 2.12: Chromatogram of separation of a C10 - C40 mixture on the silicon column via a TGGC separation method. Thermocouple positions corresponding to the base and tip temperatures are indicated in Figure 3.11a

by injecting C12-C14 at numerous pressure and isothermal temperature settings as well as under TGGC conditions. The elution times and peak widths at half max were then extracted from the separation data for calibrating the transport model. A chromatogram from this separation is provided in Appendix I.

The serpentine stainless-steel column was also run under TGGC conditions using screen-printed surface heaters. This system was able to separate samples of C8-C20 when both primary and secondary heaters were used (Figure 2.13). Other separations were also performed on the serpentine stainless-steel column under temperature programmed gas chromatography (TPGC) conditions demonstrating the quality of the coating within the column. It was found, however, that the low thermal conductivity of stainless-steel made it difficult to establish a gradient with a temperature difference near 50 °C along the column length, which is thought to be the ideal condition for good TGGC separations. Another shortcoming of the stainless-steel was that the nature of the serpentine configuration gave rise to temperature oscillations along the length of the channel. As a result, a spiral column configuration in aluminum was pursued with the goal of removing the

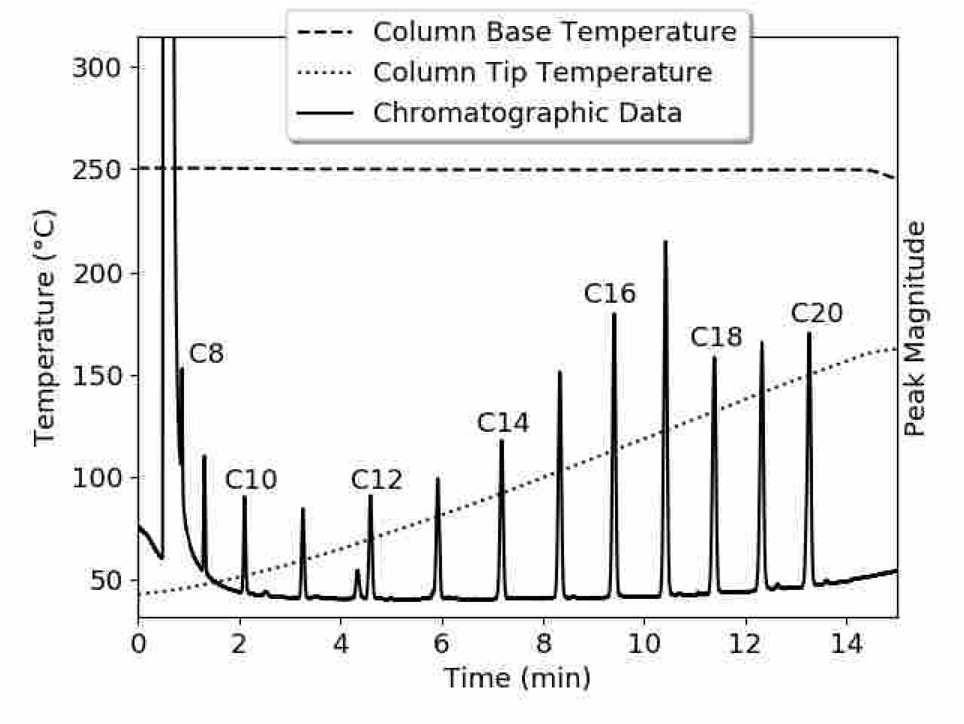


Figure 2.13: Chromatogram of separation of a C8 - C20 mixture on the stainless-steel column via a TGGC separation method. Thermocouple positions corresponding to the base and tip temperatures are indicated in Figure 3.12a

oscillatory behavior observed in serpentine column configurations via axisymmetric conduction in a radial system, as well as achieve temperature differences closer to the ideal condition (see additional chromatogram, Appendix D).

Finally, numerous separations were performed using the aluminum spiral column with and without forced convective cooling at the edges. The fan cooling system showed promise in making steeper gradients possible at lower temperatures, but the non-linear nature of the fan under PI control made smooth control at low speeds difficult, resulting in erratic peak elution times over portions of the chromatographic run. The system also exhibited poor performance due to cold spots in the transfer line. However, the addition of a copper sheath and resistively heated transfer line significantly reduced cold spots and allowed for the separation of C10-C40 (Figure 2.14) (see additional chromatograms, Appendix D).

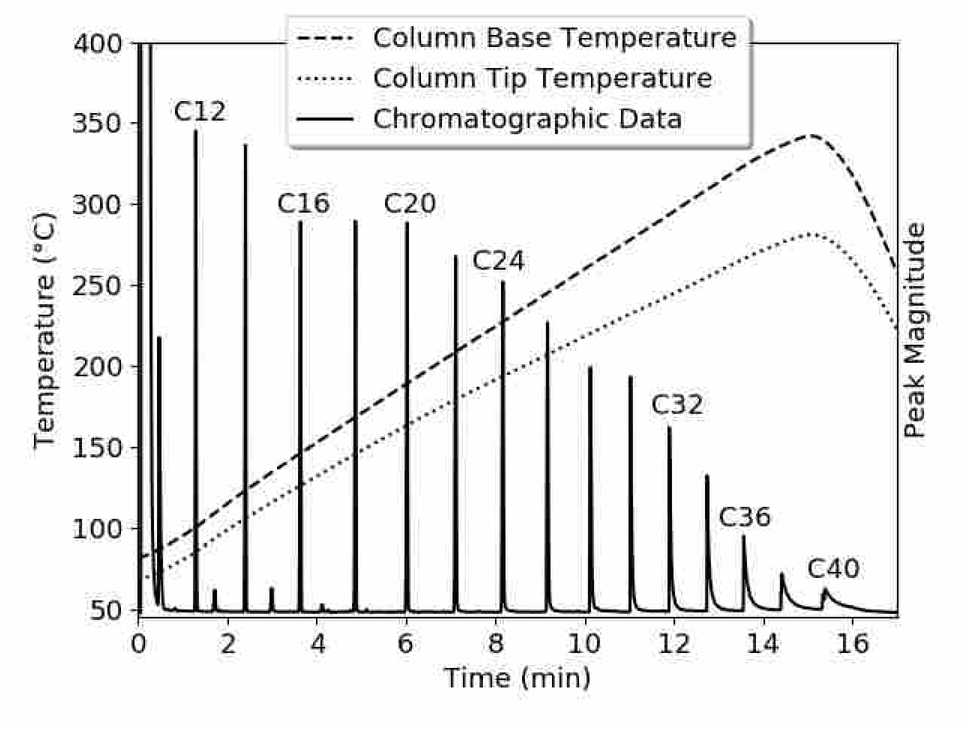


Figure 2.14: Chromatogram of separation of a C10 - C40 mixture on the spiraled column in aluminum via a TGGC separation method. The column base temperature corresponds to the center of the aluminum disk and the column tip temperature corresponds to where the spiraled column breaks contact with the disk at the outer edge of the spiraled portion (see Figure 2.5)

2.5 Conclusion

In order to successfully control and gather data on various μTGGC systems a combined controller and DAQ system was created. It was shown to successfully control complex systems and data was recorded for future use. When μTGGC systems were attached to a conventional GC, chromatographic separation data were recorded. These data, along with the temperature data could then be used to benchmark and validate the thermal and GC transport models.

CHAPTER 3. THERMAL MODELING

3.1 Motivation

Although useful, the discrete temperature measurements from experimentation were insufficient to fully characterize the temperature gradient. A higher spatial fidelity of temperature data on the µGC column would improve decision making in future iterations of the system and enable GC transport modeling of actual experimental conditions. To make this possible a computational thermal model of each system was developed to provide sufficient spatial temperature resolution for the aforementioned goals. An example of the high temperature data resolution that is made possible through the use of computational modeling is shown in Figure 3.1.

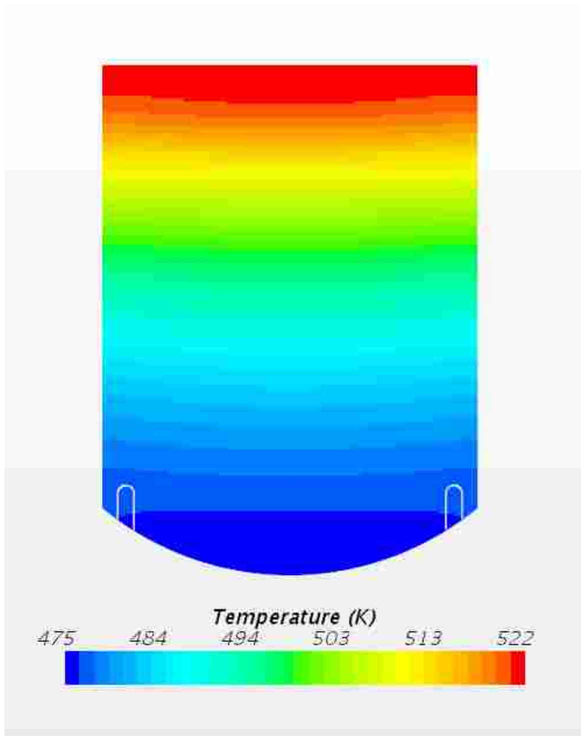


Figure 3.1: An example of the high resolution ($\leq$ 0.5 mm) temperature data that is acheivable through the use of a computational thermal modeling package.

3.2 Modeling Methods

Given the complexity of the μTGGC system and the changing system design, a versatile thermal modeling tool using the CFD software package STAR-CCM+ was selected. This approach offered physics-based thermal modeling and the ability to accommodate complex geometries. Geometries for thermal models of the TGGC systems were developed using Solidworks and imported into STAR-CCM+ for mesh computation and the development of the thermal model.

After loading a CAD model of a system of interest into STAR-CCM+, time (steady v. unsteady) and energy (segregated v. coupled) solving methods were selected and properties for the various materials used in each simulation were specified. A license to the Material Properties Database (MPDB) was purchased, providing access to a wide array of material properties which had been compiled from numerous peer reviewed sources. Aside from the Solimide foam properties, this database was able to provide all of the material property information used in the simulations.

In determining a proper mesh size, multiple factors were taken into consideration. The first consideration was the time required for simulations. Cutting the grid size in half results in an approximately four-fold increase in cells within the mesh, leading to an approximately four-fold increase in required computational time and it was observed that a mesh size finer than 0.25 mm was too small. However, it was found that for the convective boundary condition, when the grid size got too high the simulation became unstable, providing an upper bound for the mesh of approximately 4 mm that could be used in the simulation. These two limitations indicated the bounds for the mesh size that could feasibly be used in the simulations.

The next step in the model development process was the determination of proper boundary conditions for the model. Convective boundary conditions combined with radiative heat loss (assuming diffuse grey behavior) was sufficient to correctly model heat loss from external surfaces, without the added time and complexity of modeling the convective flows. The convective coefficient (h) was computed using an average of empirical correlations for natural convection from flat surfaces in various orientations. In order to simplify the relation for h used in STAR-CCM+ a second order polynomial fit was made on the average convection coefficient as shown in Figure 3.2 and Equation 3.1. To maintain accuracy in the simulations, the convection coefficient was kept temperature dependent. A second order fit was used instead of a higher order fit to minimize the computational load introduced by the convective boundaries, thus reducing the time required to complete simulations. Despite the apparent discrepancy between the empirical convection coefficient curve and the second order fit, it was found that the fitted convection coefficient values provided accurate results. This was due to the convective boundaries occurring on the outer surfaces of the insulation, which was far enough from the column to smooth out any error introduced by an imprecise convection coefficient. Radiation was not lumped into the convection coefficient, which allowed for more accurate simulation conditions as well.

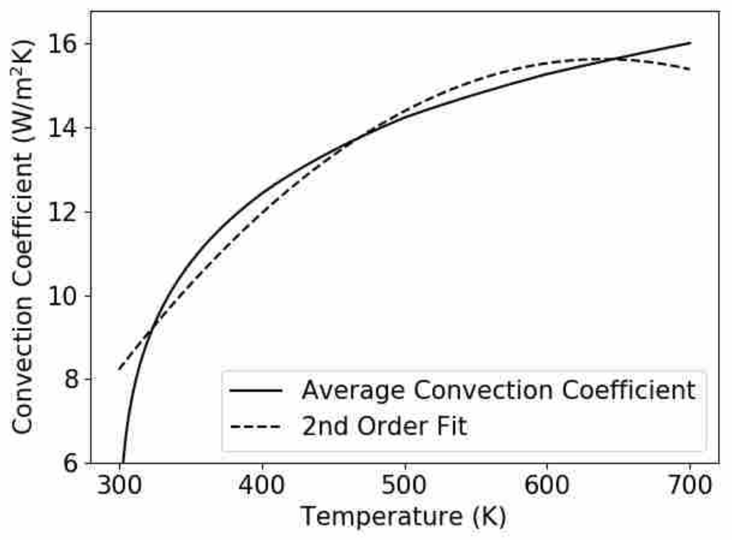


Figure 3.2: The average convective coefficient for natural convection from empirical correlations and a second order fit.

$$h = -6.4464 \times 10^{-5}(T^2) + 0.082357(T) - 10.6804 \tag{3.1}$$

Another aspect of the thermal transport that was found to play a significant role in the results given by the models was contact resistance between the column substrate and the Solimide insulation. Experiments were performed to determine the appropriate contact resistance for these interfaces as discussed below.

As all heat inputs to the experimental system were achieved using resistive heating, the primary method for heat input in the simulations was to specify the heat flux at heating boundaries. Power data reported by the controller/DAQ provided close but not sufficiently close temperature agreement with experimental results. Thus, an alternate method was developed in which the temperatures recorded from the experimental run were read into STAR-CCM+ and a virtual PI controller was implemented in the simulation such that the power output of the heater was dependent on the desired temperature of the heater at any point in time. The power data taken from the experimental system were then used to ensure that the power used in the simulation was within statistically significant bounds (Appendix E). For simulation of the serpentine silicon column in the heater-clamp assembly, a uniform temperature boundary was used. The experimental heat source for this system was a resistive cartridge heater that was placed in the Kovar clamp such that a uniform temperature boundary (where the clamp interfaced with the silicon) provided adequate agreement between the model and experimental system.

In the case of the stainless-steel simulation, an additional adjustment to the contact resistance between the column and insulation was required. In order to make electrical contact with the secondary (or larger) heater (Figure 2.4) alligator clips had to be connected to a portion of the column that was under insulation. The presence of these clips created an air gap between the column and insulation. In order to account for this air gap in the simulation a contact resistance for the air gap was created based on the model for thermal resistance for conduction through a one-dimensional plane wall [45]. The air gap resistance factored in the temperature of the air in the gap (T_{air}) to determine the thermal conductivity of air (k_{air}) as well as the height of the air gap (L_{air}) as a function of position along the column (x) where the air gap started halfway down the length of the column (x_1) (Equation 3.2). The height of the air gap was determined to be dependent on x due to the sloped lower surface of the insulation sitting on the heater contact alligator clips. Note in Equation 3.2 that $F = 6.515 \times 10^{-5}$ W/mK^2 and $G = 0.00663$ W/mK.

$$R''_{air} = \frac{L_{air}}{k_{air}} = \frac{0.0019(x-x_1)/(0.171-x_1)}{FT_{air}+G} \tag{3.2}$$

An alternate approach to thermal modeling of the aluminum spiral column system was also employed. Due to the simplicity of this system's geometry, a combination of finite difference methods and analytical methods was used to solve the heat diffusion equation. The temperature of the column was modeled using a transient finite difference approach and a Python solver was developed to compute the gradient. The energy loss from conduction into the insulation was modeled using an analytical solution to the heat diffusion equation. Although not used for transport modeling, this thermal model demonstrated that a radial system could significantly reduce temperature oscillations along the length of the column and it indicated the shape of the resultant gradient along a spiral path milled into the aluminum disk (Figure 3.9).

3.3 Validation Methods

For each of the thermal models, a process for validation against experimental data was performed to ensure that the spatial and temporal data provided by the thermal models were representative of the μTGGC system. Although the particulars of validation for each of the systems were unique, the general approach was similar. After specifying geometry and an initial simulation, the data reported by the simulation would be compared to the associated experimental data. The factor of interest for validating the simulations was the difference in temperature between the experiment and model, as measured at the locations where the thermocouples had been placed in the experiments (Figure 3.3). Observing the difference between the thermocouple and simulated

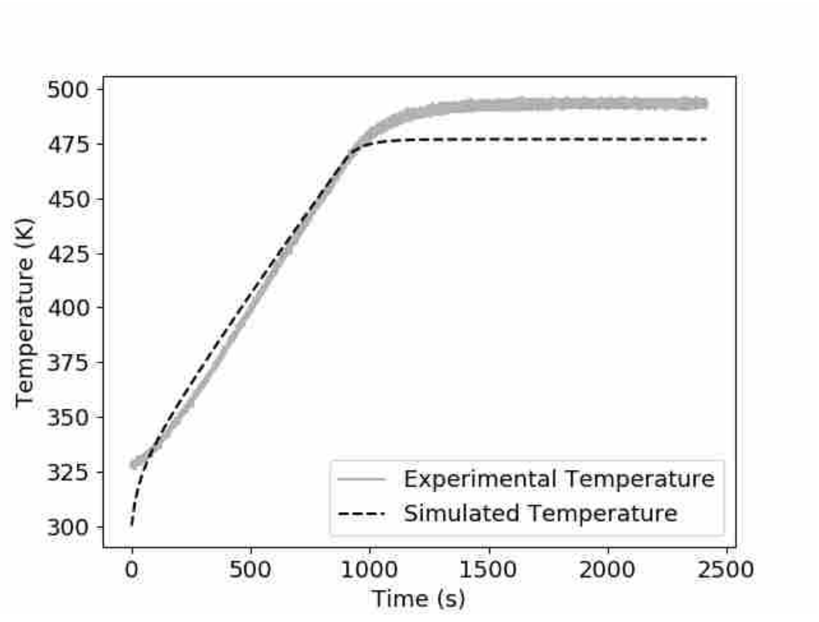


Figure 3.3: Comparison of experimental and modeled temperatures for the silicon column. Only data for the temperature of the primary heater ramping from near room temparature to 225 °C are shown.

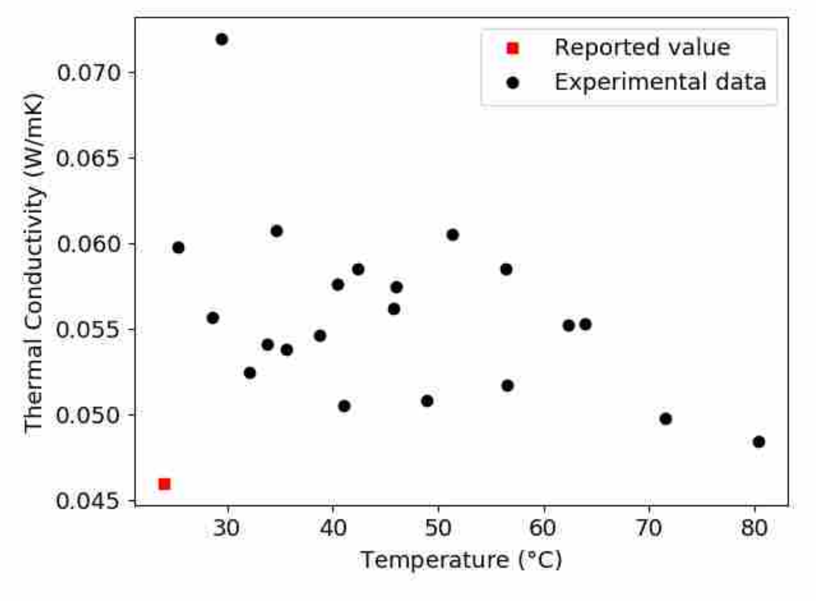
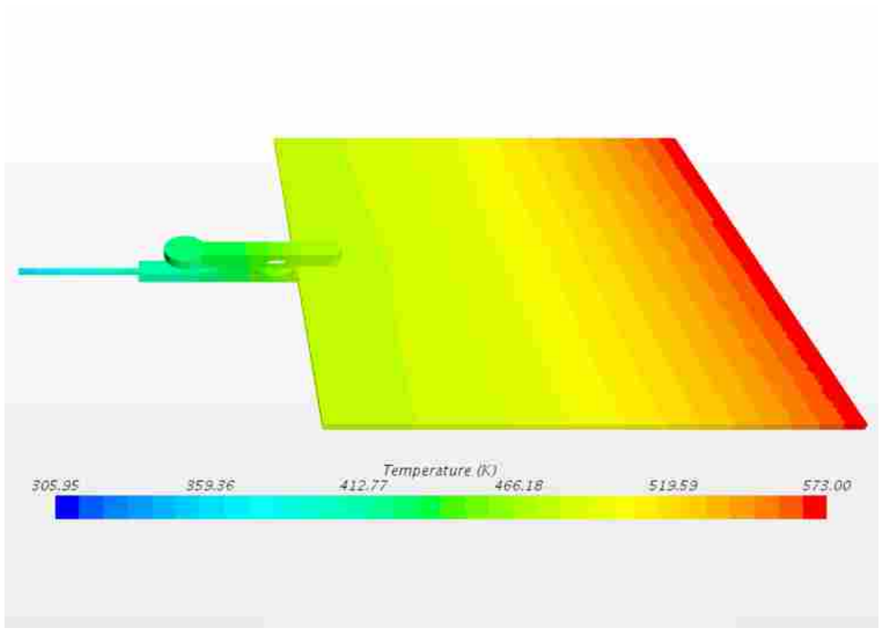


Figure 3.4: Reported value [46] and experimental data for Solimide thermal conductivity.

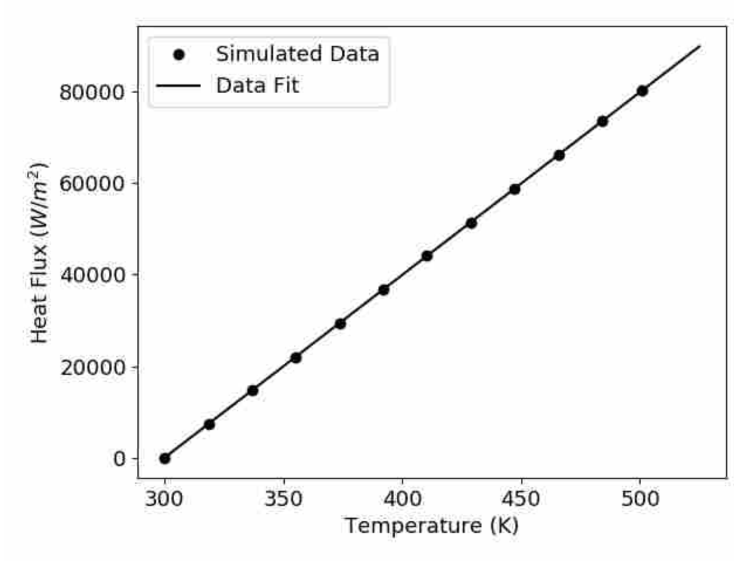
temperatures over time allowed for effective identification of weaknesses in the model. After each run conditions would be altered as needed (e.g. contact resistance, mesh size, boundary conditions, heat input, etc.) and the simulation would be run again. Numerous iterations were run in this manner for each system to minimize model/experiment mismatch until the difference between the two was within $\pm$ 5 °C.

One of the significant factors that was found to influence accuracy of the simulation was the thermal conductivity of Solimide and contact resistance between the Solimide insulation and the column. In order to determine values for both of these parameters an experiment was developed in which a heat flux sensor and segment of Solimide foam were placed on top of a heated copper block. The heat flux sensor was attached to the copper with a sheet of thermal contact tape to eliminate variability in the heat flux passing from the copper to the sensor. The Solimide foam was placed on top of the heat flux sensor with a series of thermocouples placed in the foam near the center of the heat flux sensor to approximate one dimensional heat conduction through the foam. The Solimide was weighed down using another copper block similar to the µTGGC experiments. The copper block on the bottom was then heated to various temperatures and data were gathered and Fourier's Law of conduction was used to compute thermal conductivity for the foam (Figure

3.4) and the definition of contact resistance was used to compute the contact resistance between the heat flux sensor and the Solimide (Appendix G).



(a)



(b)

Figure 3.5: Results from the thermal model for alligator clip cooling: (a) Temperature distribution resulting from the presence of clip cooling; (b) heat loss into an alligator clip fit as a function of surface temperature.

In the case of models that involved surface heaters, it was important to know the effects an alligator clip could have on the heat loss from the surface of interest. To this end a simulation study of an alligator clip attached to a surface was modelled in STAR-CCM+ (Figure 3.5a). The far edge of the chip was set to various temperatures and the heat flux into the alligator clip was measured. It was found that the relationship between chip temperature and alligator clip heat loss was linear (Figure 3.5b). In order to avoid including the complexity of alligator clips in full simulations the linear fit for alligator heat loss could be included as a surface boundary heat loss, thus retaining the effects of the alligator clip without including the complex geometry it required.

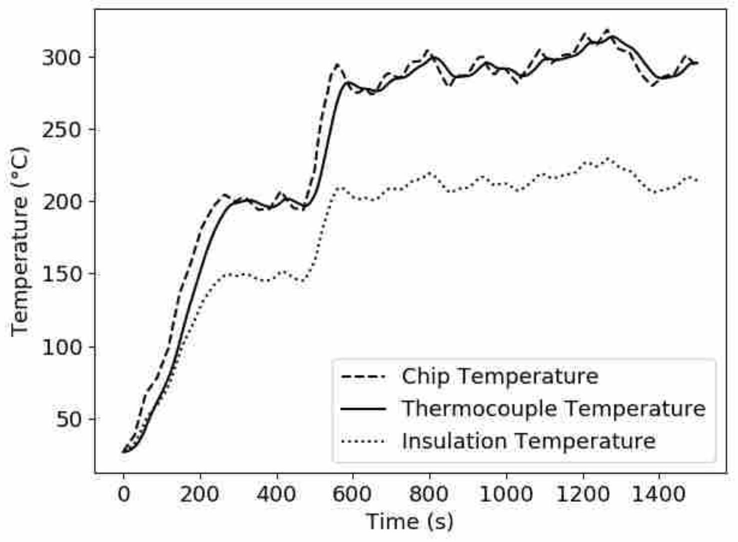


Figure 3.6: Thermocouple computational model temperature data.

Another aspect of validating the thermal model that required particular attention was the modeling of thermocouple temperatures. The thermocouples used in the experiments were primarily attached to surfaces using strips of polyimide (Kapton) tape, and after initial comparisons between experimental data and simulated results it was clear that the transient nature of the two did not agree. However, it would not be feasible to include a thermocouple and Kapton tape in the thermal model as it would significantly increase simulation complexity and time. In order to determine how a surface's temperature relates to the temperature of a thermocouple taped to the surface a STAR-CCM+ model of a thermocouple taped to a heated surface was developed. This simulation illustrated that although a thermocouple doesn't match the temperature of the surface it is taped to under transient conditions, when the surface reaches steady state the thermocouple settles at the same temperature as the surface (Figure 3.6). Knowing this made it possible to identify whether or not a simulation and the associated experiment matched correctly even when comparing experimental thermocouple data to the simulated surface temperature.

3.4 Design Study Models

As discussed, although the principal goal in developing thermal models was to obtain high resolution temperature data on the µTGGC experimental systems, another important purpose of the thermal models was to aid in making informed design choices for the development of future µTGGC system designs. To this end, several models were developed to answer specific design questions. One of these design exploration simulations was a series of simulations run to determine the proper size and resistance values of the heaters that would be printed on the stainless-steel column. The models developed helped in avoiding printing poorly designed heaters on the expensive columns that were fabricated. Results from the model developed are shown in Figure 3.7.

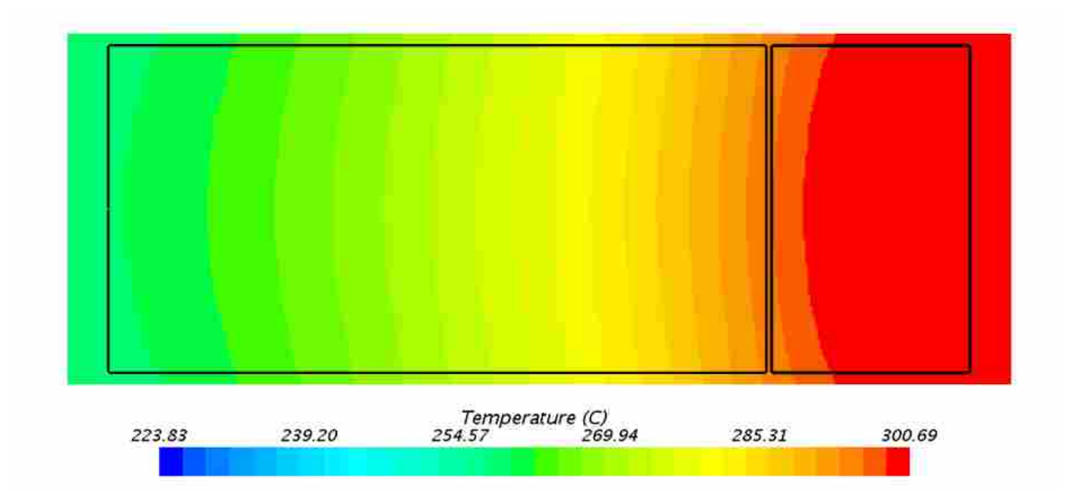


Figure 3.7: Heater sizes (black lines) and resultant temperature profile on the stainless-steel column.

Following the completion of preliminary experimental runs on the stainless-steel column, it was found that the separation quality was less than ideal and it was suspected that cold spots might be the cause. One of the suspected causes for the cold spots was the heat loss into the alligator

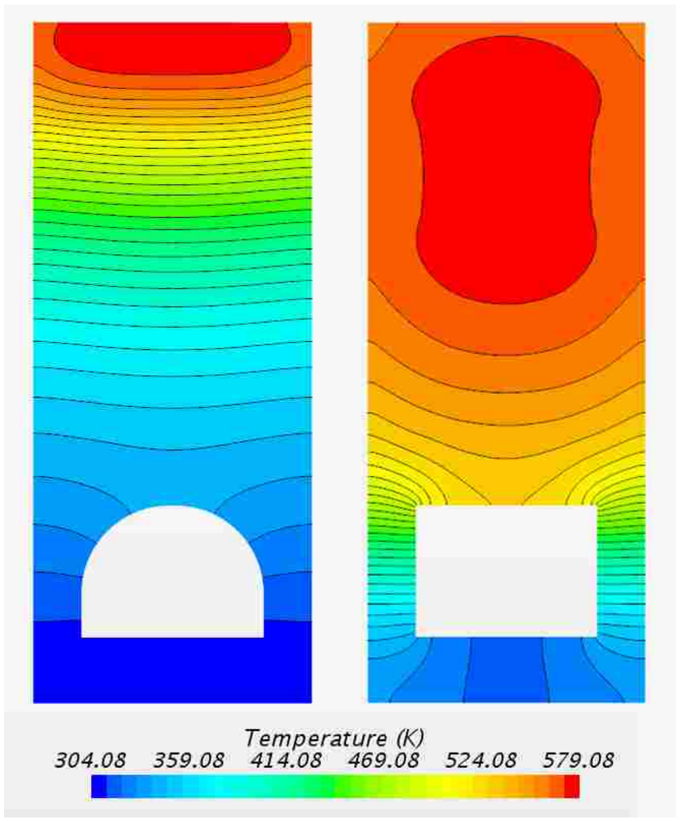
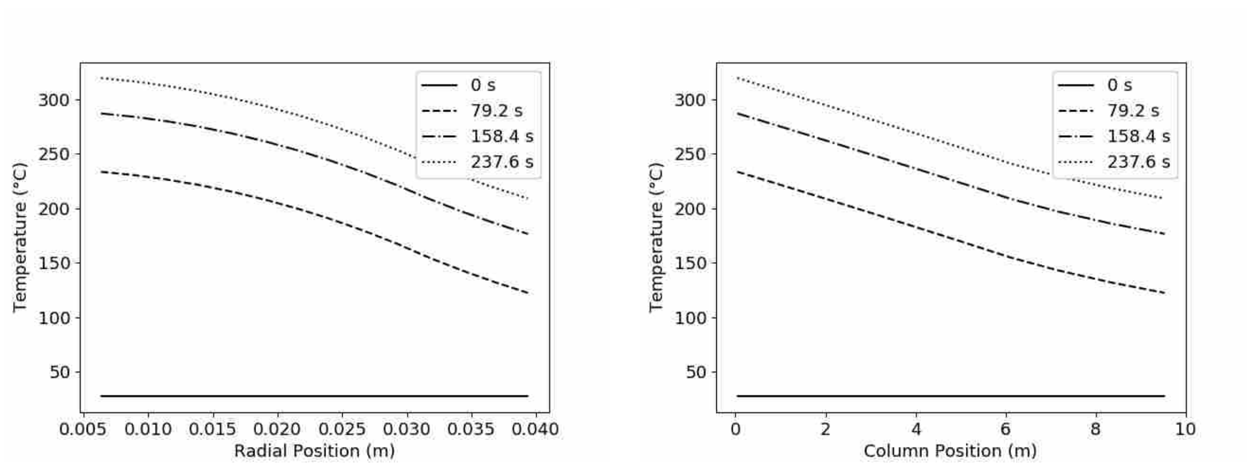


Figure 3.8: Two stainless-steel column footprint changes explored as part of the design study.

clips used on the corners of the column to connect to the surface heaters might be causing the cold spots. To this end, a series of design study simulations were developed in an effort to minimize the effect of the alligator clips on the column gradient. Two of the shapes explored in the design study are shown in Figure 3.8. Ultimately it was determined that no change of alligator clip location could improve the nature of the gradient.

The third round of design study thermal modeling was to help in determining a new direction for the µTGGC project after tests on the stainless-steel column didn't provide the desired results. In the case of these models, a blended analytical and one-dimensional numerical approach was employed to find the resultant column gradient along a spiral path. As is shown in Figure 3.9 the radial gradient along the disk was non-linear, however when coupled with the spiral column path along the disk, the resultant column gradient was approximately linear. The result of this design study was the development of the spiral column system that has been discussed in the previous chapter. Details regarding the creation of the model are given in Appendix F.



(a) (b)

Figure 3.9: Results from the spiral system model with respect to both (a) the radial direction and (b) along the spiral column path. In the case of the results shown above a single heater with a diameter of 6 cm was run at a constant 35 W of power input. Note that in plot (a) only the gradient over the range of the spiral column is plotted.

3.5 µTGGC System Modeling Results

Initial attempts at validating the STAR-CCM+ thermal models involved using data from a system that had been rigged with nine thermocouples along with a thermocouple for heater control. Comparison between the experimental data gathered from the thermocouples and the results from the simulation showed good match-up ($< \pm 5$ °C error) (Figure 3.10). Although this approach was helpful in developing the methods for system validation, as well as showing the shape of the thermal gradient along a square chip, it was not representative of a system on which GC separations were performed and thus there were no chromatograms to go with the validated thermal model.

The next system that was thoroughly validated was the silicon chip mounted in the heater/-clamp assembly. The heater/clamp assembly was used for experiments because of delays in being able to produce more silicon chips due to damaged equipment in the BYU clean room. Thus a clamp that had been used in prior experiments was milled out to allow for transfer line attachment at the side of the chip. The simulation geometry was simplified slightly from the system used in experiments to improve simulation time. In the process of validating this model it was found that the match-up of the simulation to the experimental data was highly dependent on the contact resistance and thermal conductivity of the Solimide. Thus it was for this system that the experiments

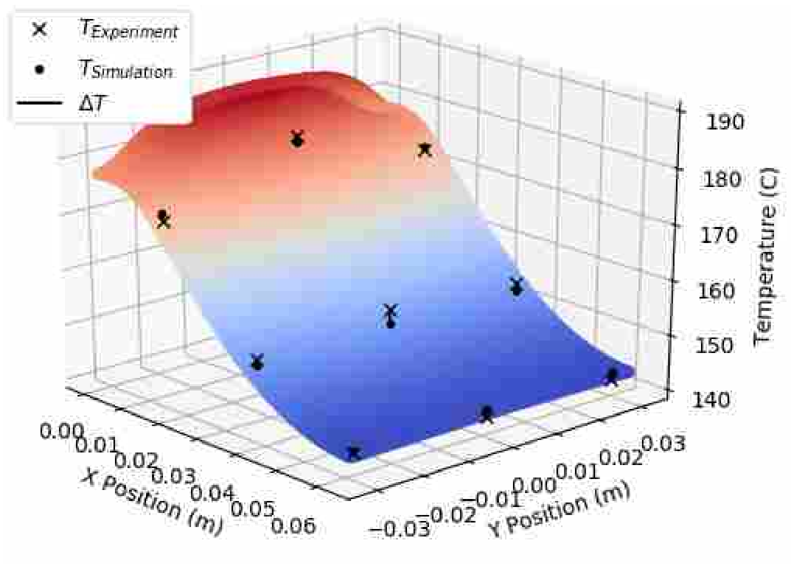
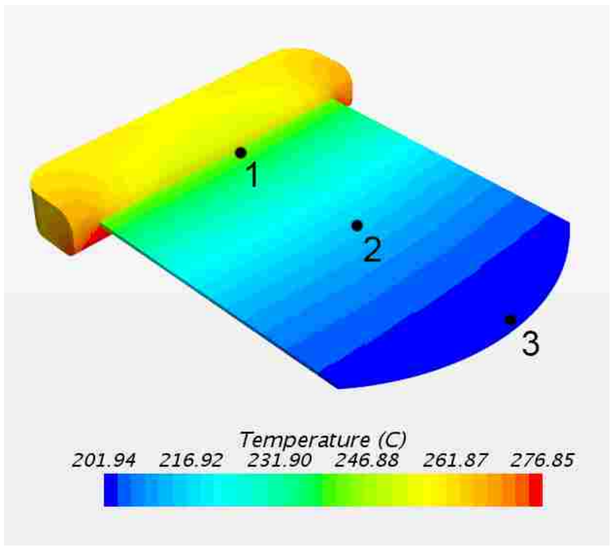
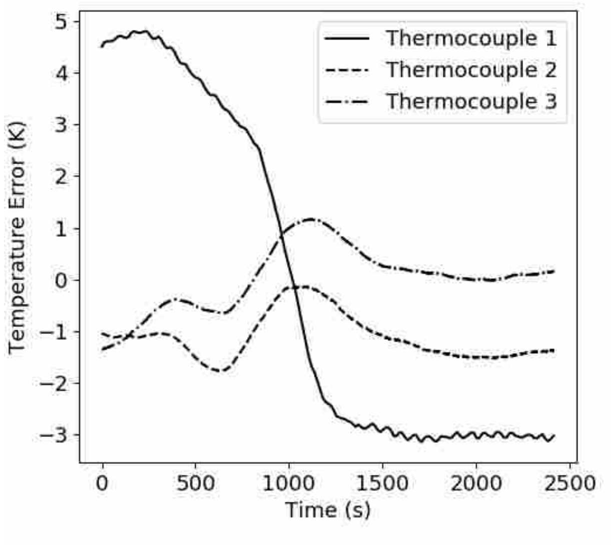


Figure 3.10: Comparison between simulation and experimental temperatures for the nine thermocouple replica system.

on Solimide properties were performed. It was found that these experimentally obtained properties greatly improved the match-up between simulated and experimental temperatures on the chip (Figure 3.11).

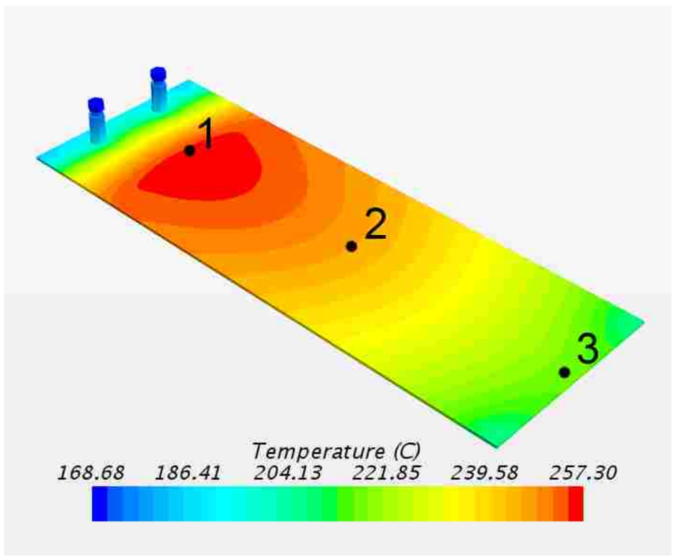
The last µTGGC system for which simulations were run were those done on the stainless-steel chip. The experiments on the stainless-steel chip were run with the intent to validate the GC transport model. It was in the process of validation for the stainless-steel chip model that it was recognized that a sub-model for the temperature of a thermocouple would need to be developed. Another noteworthy aspect of the simulation method was the refinement of power treatment in the system. As all the heaters in the system were screen-printed on the chip they were much more sensitive to errors in the power calibration. A more robust method for power calibration was developed to make the modeling of this system more accurate. The comparison between experimental and simulated temperatures for this µTGGC system is shown in Figure 3.12. Upon completion of this model the temperature data was extracted and used for experiments on the GC transport model.

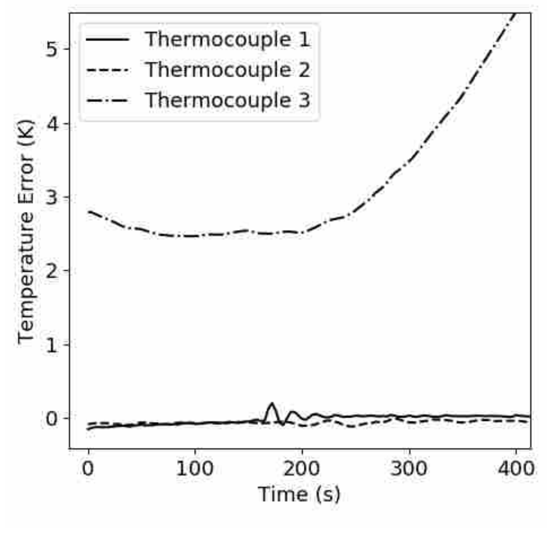




(a) (b)

Figure 3.11: Results from the thermal model for the silicon column: (a) Temperature distribution along the silicon column and Kovar clamp with the thermocouple positions indicated; (b) plot showing the difference between the simulation and experimental data.





(a) (b)

Figure 3.12: Results from the thermal model for the stainless-steel column: (a) Temperature distribution along the stainless-steel column with the thermocouple positions indicated; (b) plot showing difference between the simulation and experimental data.

3.6 Gradient Preparation

The final step in preparing the thermal gradient data for the transport model was the extraction of the temperature distribution along the length of the column. During the computation of the thermal simulation in STAR-CCM+, data tables containing the temperature at each node were exported at regular simulated time steps. As the layout of the mesh used in the simulations did not align with the serpentine path followed by the column a method for interpolating the temperatures along the column path was developed. The first step of this process was to determine the path that the column followed along the substrate (e.g., Si or stainless-steel). Design drawings were consulted to determine the dimensions of the path along the substrate and a Python script was written that computed and recorded the positions of a stepwise progression along the column path. This stepwise data of the column path was then used in conjunction with the tabulated temperature data provided by the STAR-CCM+ simulations, and 3D temperature interpolation between the simulation data and the column position was taken for each point along the column (Appendix H). The

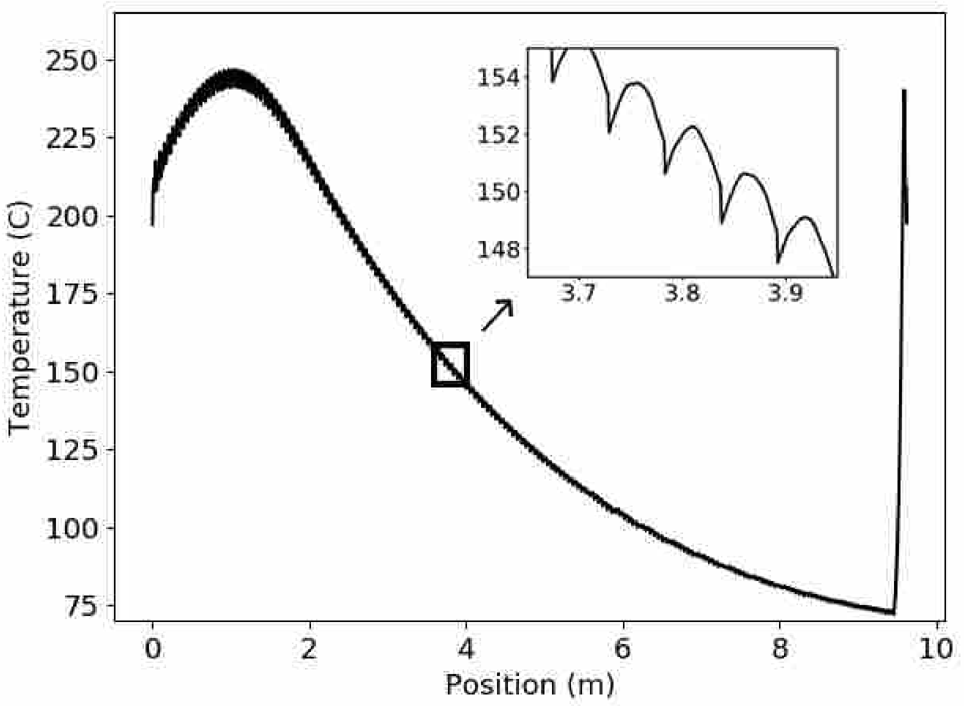


Figure 3.13: Plot of the extracted gradient along the serpentine channel path in the stainless-steel column under initial conditions with the primary heater set to 250 °C.

output from these column gradient extraction methods showed oscillatory behavior (Figure 3.13) not immediately apparent from looking at the gradient along the entire substrate. The effects of these oscillations were studied using the transport model, and will be discussed in the next chapter.

3.7 Conclusion

In order to make simulation of GC transport in thermal gradient columns possible high resolution thermal data had to be obtained. To this end computational thermal models were developed using STAR-CCM+. Thermal models for several systems were created and were verified against experimental temperature data. The simulated thermal models were shown to accurately represent the behavior of the various systems, thus indicating that the temperature data are suitable for use in the transport model. Thermal simulations were also developed to aid in design decisions for future TGGC work.

CHAPTER 4. GC TRANSPORT MODEL

4.1 Motivation

Since the inception of GC, researchers have sought to develop useful models to predict chromatographic separation. One of the primary purposes of such models is to better understand the fundamental driving forces behind GC separation, with the intent to improve performance. As in the development of open tubular column technology, a correct model can provide the information necessary to make changes to GC systems leading to significant improvements in performance.

Many of the models developed for GC separation are based on the limited information that a GC system can provide. For a given analyte, the only information a GC system can give are the retention time and width of a peak along with the conditions that gave these results (column temperature, inlet pressure, etc.). Using this limited information, the most widely used model for chromatographic retention behavior, the retention factor (k), was developed [31]. As this parameter reliably indicates GC retention under ITGC and TPGC conditions, its use in the field of GC has persisted to the present. However, one of its limitations has been its inability to predict GC retention on columns with a variable temperature along their length, making modeling of TGGC separation an elusive task.

The first full treatment of a model for TGGC was developed in the early 1990's by Blumberg, approximately 40 years after the application of thermal gradients on GC columns was first developed by Zhukoviskii [15, 39]. In order to develop his model, Blumberg applied the mass diffusion equation to model the gas and analyte movement along the column. Starting with this equation he was able to derive a simplified form by modeling each chromatographic zone as a normally distributed concentration in the column. This allowed for the mass diffusion equation to solve for peak position and standard deviation as a function of time. However the approach included significant assumptions; in particular, the analyte injection was idealized as a normal distribution with a standard deviation of zero along with assuming a monotonic linear thermal gradient. Although Blumberg's model was a significant advance in modeling TGGC separations, its limitations are motivation to revisit the approach.

In 2015, a new approach to modeling non-ideal GC separations was developed by Tolley *et al.* by applying the widely used retention factor along with principles of gas mass diffusion to a stochastic modeling approach [43]. Due to the stochastic nature of Tolley's treatment of GC, spatial variations in temperature along with non-ideal injections and variable coating thickness could be incorporated without making the model extremely difficult to solve. This capability of accommodating variable and non-ideal conditions makes Tolley's model a significant advance in GC modeling capabilities. This model by Tolley *et al.* was selected as the starting point for this work in predicting chromatographic separation behavior based on numerically predicted column temperatures (Chapter 3). The stochastic modeling approach by Tolley *et al.* will henceforth be referred to as the transport model. This work offers a full calibration of the transport model using TGGC separations and explores its use as a diagnostic tool in TGGC.

4.2 Separation Performance Measures

Before describing the methods used to calibrate and validate the transport model it is helpful to describe a few of the common metrics used to quantify GC separation quality. The first of these measures, the retention or elution time (t_R), is a measure of the time required for a peak of interest to move down the length of the column and through the detector. The second measure used is the peak width at half its maximum (w_h), which provides a measure of how broad a peak has become as it has moved down the column. The width at half max is the preferred measure of peak spread as it is not as sensitive to variation due to peak tailing and fronting as the peak base width. A visual representation of retention time and peak width at half max is given in Figure 4.1. The third measure of separation performance that will be discussed in the following sections is resolution. Resolution is a parameter that measures how well two peaks are resolved from each other, and roughly represents how many peaks apart two analytes are from one another. Thus, a higher resolution means a better separation. The formula for computing resolution between two peaks is given in Equation 4.1.

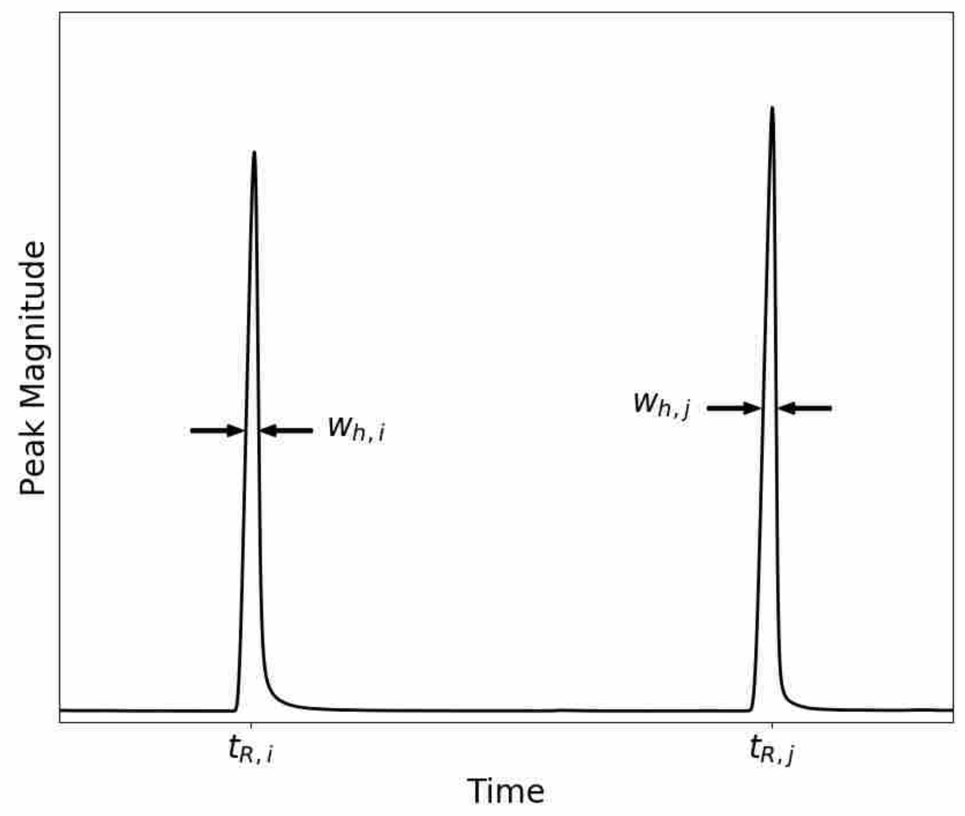


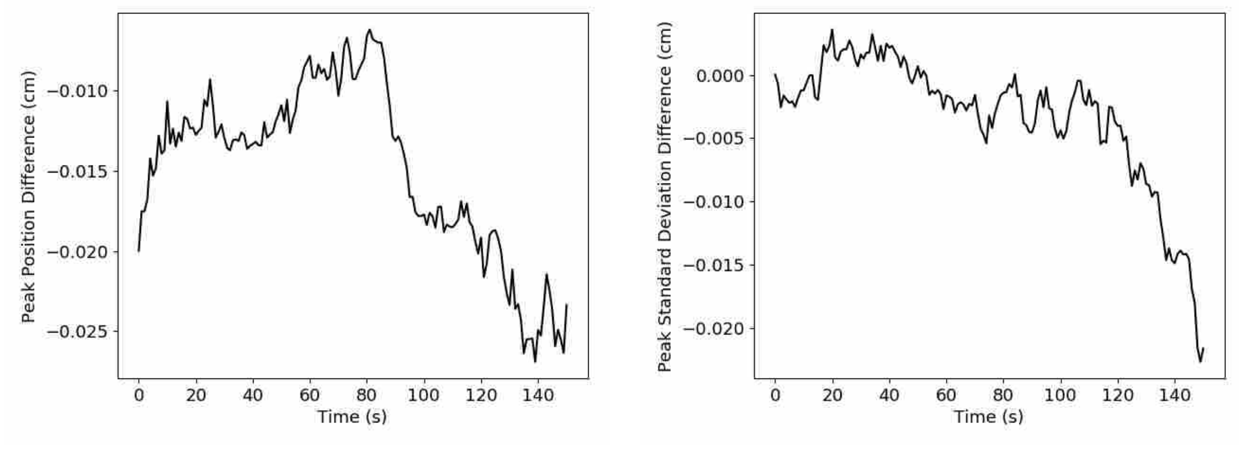
Figure 4.1: Schematic of retention time and peak width at half max for two neighboring peaks.

$$R_{i,j} = \frac{t_{R,j} - t_{R,i}}{4[(\sigma_i + \sigma_j)/2]} = 1.17741 \frac{t_{R,j} - t_{R,i}}{w_{h,i} + w_{h,j}} \tag{4.1}$$

4.3 Code Adaptation

The first step in allowing for calibration and validation of the transport model was to modify the transport model to accept thermal gradient data obtained from computer simulations. Early iterations of the transport model were written in the statistical coding language R. In order to decrease run time and permit interfacing with large sets of thermal data from simulations the transport model was translated into Python. To ensure that the code was translated correctly both the R code and Python translation were run five times each under the same ITGC input conditions. The average error between the two was then computed to ensure correct function of the Python version.

As can be seen in Figure 4.2 minor differences exist in the behavior of the peaks; however, due to the stochastic nature of the model, minor differences are expected. Following completion of the code translation, functionality that allowed for reading in of thermal gradient data was included and averages of the results from five runs of both the Python and R forms of the transport model under ITGC conditions were again compared and shown to perform equivalently, yielding results similar to what is seen in Figure 4.2.



(a) (b)

Figure 4.2: (a) Peak position and (b) standard deviation differences between the original R code and Python transport models.

4.4 Transport Model Calibration and Validation

The next step in preparing the transport model for use on simulated TGGC data was to calibrate the parameters used to determine the retention factor and mass diffusivity of the compounds separated in the transport model. Following the model reported by Blumberg [32], the retention factor could be computed as a function of the enthalpy (ΔH) and entropy (ΔS) of analyte evaporation and the phase ration (β) (Equation 4.2). Since the enthalpy and entropy parameters dictate the nature of interactions between the analyte and the stationary phase they were dependent not only on the compound under consideration but were also influenced by the nature of the stationary phase in the column used for separation.

$$k = \exp\left(\frac{\Delta H}{RT} - \frac{\Delta S}{R} - \log(\beta)\right) \tag{4.2}$$

In order to accurately model the dispersion of the analytes as they moved through the column the analyte zone variance as a function of time had to be determined. Starting with the one dimensional form of the mass diffusion equation and assuming each analyte to be normally distributed within the column, the relationship between the change in an analyte's variance as a function of time was determined as given in Equation 4.3. In order to compute the variance of an analyte in the mobile phase a proper model for mass diffusivity in a gas is required. The Chapman-Enskog model for mass diffusivity was selected due to its relatively low reported error (<8%) as well as its dependence on temperature and pressure, lending itself well to TGGC applications (Equation 4.4, variables defined in nomenclature section) [47].

$$\Delta\sigma^2 = 2D\Delta t \tag{4.3}$$

$$D = \frac{AT^{3/2}\sqrt{1/M_1 + 1/M_2}}{P\sigma_{12}^2\Omega} = \gamma\frac{T^{3/2}}{P} \tag{4.4}$$

In order to facilitate calibration of the transport model, data were taken by performing ITGC runs on the serpentine stainless-steel column at various temperatures and inlet pressures to determine the elution time and width at half max for each peak. Details on these runs are provided in Appendix I. These data could then be used together with linear regression techniques on log transformed data to determine enthalpy and entropy parameters for each compound in the column. In the case of the experiments performed here, a mixture of n-dodecane (C12), n-tridecane (C13), and n-tetradecane (C14) dissolved in a solution of n-pentane (C5) was used. The calibration separations were performed in an Agilent 6890 GC oven with the serpentine stainless-steel column mounted in front of the convective fan (see Figure 1.3) so that the entire column would remain isothermal throughout the run. In order to minimize effects of human error and ensure that the data were statistically sound, the pressures for each run were randomized and after each change in temperature the GC oven was allowed to sit for 15 minutes to give the oven sufficient time to reach the temperature reported by the oven's thermocouple. The data taken for the calibration were shown to have minimal noise and a clearly visible trend giving confidence in performing the parameter calibration computations (Figure 4.3). Details regarding the method used to calibrate the model parameters from the ITGC data are given in Appendix I.

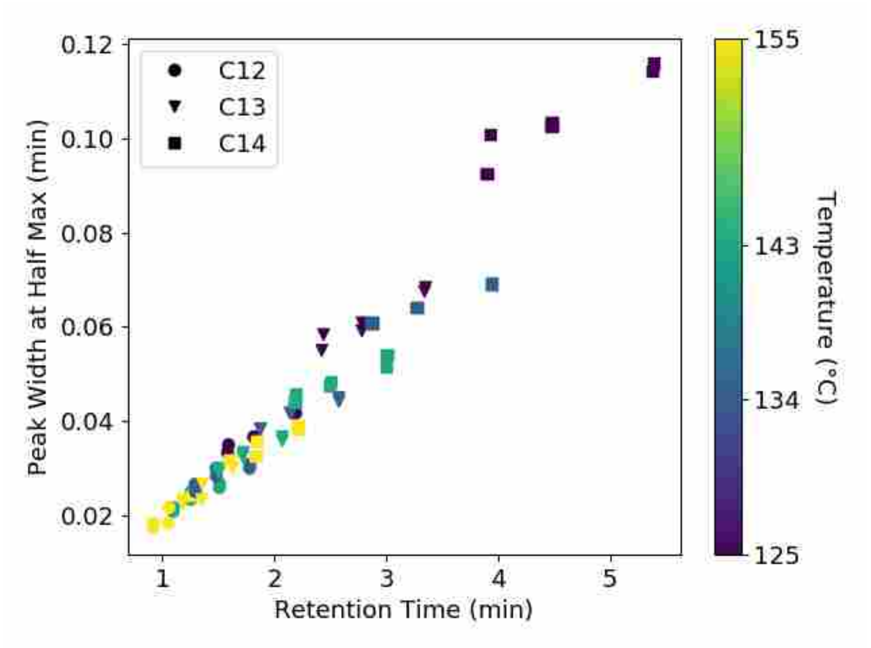


Figure 4.3: Data gathered from ITGC runs on the stainless-steel chip for use in calibrating the transport model. For tabulated conditions see Appendix I

Following the ITGC separations the stainless-steel column was run under TGGC conditions as shown in Figure 4.4 with varying inlet pressures and secondary heater temperature ramp rates. In each of these runs the primary heater was set to 250 °C and the secondary heater was turned off to establish the gradient. Under this initial condition the column tip temperature was at about 70 °C. At the time of injection the secondary heater was turned on and its temperature was controlled from its starting point up to 250 °C over time spans from 8 to 15 minutes. Details regarding each of these runs are privided in Appendix I. The purpose for these data was to permit validation of the parameters derived from the ITGC data. After determining the enthalpy and entropy parameters for the retention factor model (Equation 4.2) using the ITGC data it was found that the parameters obtained in this manner did not give adequately close results between the experiment and model for the TGGC runs (23% error in elution time). Therefore, a new method of computing correct retention and dispersion model parameters was developed that involved the use of data taken from the TGGC runs.

In order to determine the correct retention parameters a set of six guesses were made for both parameters ($\Delta H/R$, and $\Delta S/R - \log(\beta)$) over a wide range. The transport model was run for all six guesses of each parameter and the squared error for elution time was computed. A quadratic function was then fit to the resultant squared error values and a gradient method was used to hone in on the parameters that would minimize the retention time squared error. This process of guessing and solving for the optimum was repeated until the process converged on the best possible retention parameters. A similar method was used for the dispersion parameter, that only differed in that only one parameter was computed decreasing the dimensionality of the function space. Details on these calibration methods and their resultant parameters are discussed in more detail in Appendix I.

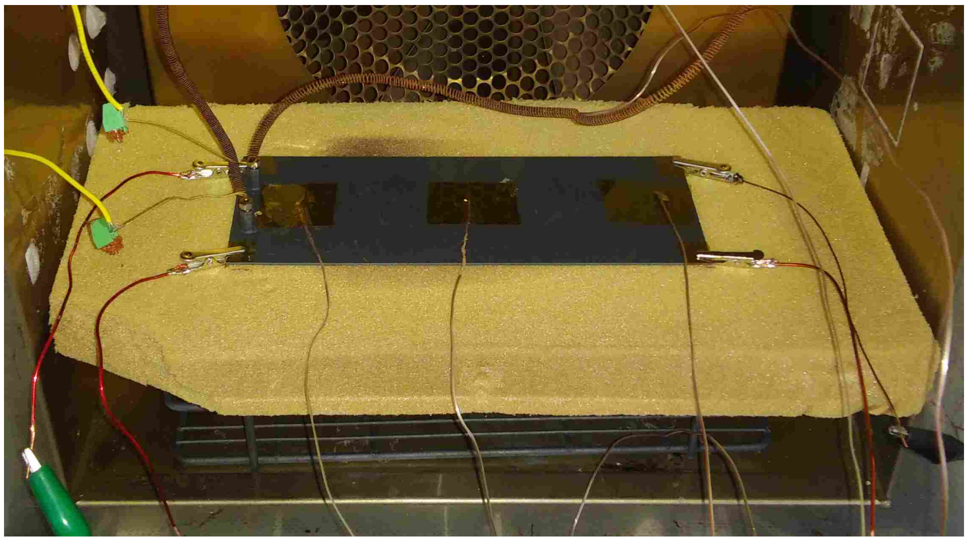


Figure 4.4: The stainless-steel column connected to the DAQ/controller for running TGGC separation conditions.

4.5 Transport Model Conditions

Before describing the various tests that were run using the transport model a note should be made regarding the conditions that were common to all transport model runs. In order to minimize variation in the results between runs 1000 molecules were modeled for each analyte. The time was discretized to a resolution of 5×10^{-5} s to ensure that non-linear temperatures, pressures, and mobile phase velocities would not have a significant influence on the behavior of the molecules. In all cases the column inlet pressure was set to 449.6 kPa and the outlet pressure was set to the local atmospheric pressure of 83.6 kPa. According to measurements taken from cross-sectional SEM images of the serpentine stainless-steel column (Figure 2.4) a hydraulic diameter of 57 μm was used to model the fluid flow through the column. The product of the coefficient of friction and the Reynolds number ($C_f Re_D$) was set to 15.697 to model the effects of the semicircular cross-section of the column channel. As all of the associated experiments were performed on the stainless-steel column, the stainless-steel channel length of 9.6 m was used in all transport model runs. In all cases, the injection width was set to 2 cm with the molecules initially uniformly distributed throughout the injection width. Although information was not known regarding the experimental injection widths, a non-ideal, wide injection was used to demonstrate TGGC's ability to correct for poor conditions. Temperature conditions varied between runs and are described on a case-by-case basis in the sections below.

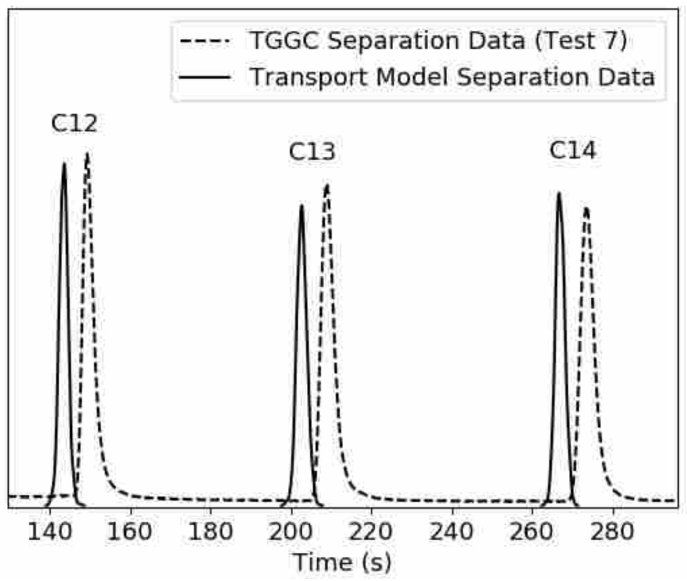


Figure 4.5: Comparison of peak shape and elution time between the transport model and experimental results for a TGGC run on the stainless-steel column. For data on Test 7 see Appendix I

4.6 Transport Model TGGC Performance

Following the determination of parameters for both the retention factor and dispersion using data from two TGGC runs, the parameters were tested against the ITGC data as well as a third TGGC run to test the calibrated model's ability to predict the separation performance under varying conditions. A description of the heating conditions for the TGGC runs is given in Section 4.4, and tabulated data are given in Appendix I. An example of the transport model TGGC separation results compared to corresponding experimental data is shown in Figure 4.5 (see also Table 4.1).

It was found that both the retention factor and dispersion parameters required fine tuning on a run-by-run basis in order to obtain exact agreement between simulation and experimental results. Thus in order for the transport model to serve as a predictive tool for future runs a degree of error

Table 4.1: Table containing the retention time and width at half max values along with their associated errors for the TGGC transport model validation runs. Six runs are tabulated with three analytes per run. Retention error $\lesssim 4\%$ and width at half max average of 23.2%.

Run	Analyte	Model Retention Time (s)	Retention Time Error Relative to Experiment (%)	Model Width at Half Max (s)	Width at Half Max Error Relative to Experiment (%)
1	C12	143.56	4.04	2.41	19.51
	C13	202.82	2.91	2.73	23.14
	C14	267.06	2.35	2.84	25.36
2	C12	143.57	4.03	2.51	16.29
	C13	202.79	2.92	2.87	19.11
	C14	267.00	2.38	2.66	30.02
3	C12	143.54	4.05	2.40	20.07
	C13	202.72	2.96	2.74	22.90
	C14	267.06	2.35	2.63	30.72
4	C12	143.57	4.02	2.43	18.91
	C13	202.86	2.89	2.74	22.89
	C14	267.06	2.36	2.78	26.83
5	C12	143.49	4.08	2.37	21.06
	C13	202.81	2.92	2.78	21.62
	C14	267.03	2.37	2.77	27.22
6	C12	143.54	4.05	2.36	21.43
	C13	202.81	2.92	2.78	21.80
	C14	266.96	2.39	2.74	27.82

would have to be tolerated. However, considering the highly non-ideal nature of the TGGC experimental setup (i.e., oscillatory gradient, uneven stationary phase thickness along column length, mobile phase mixing at serpentine bends), the transport model was able to obtain results remarkably close to the experimental results with retention time errors of less than approximately 4% and peak width errors averaging 23.2% (Table 4.1). It should be noted that the high error values for the width at half max are partially a result of the small scales of the widths, thus these error values are not necessarily indicative of a poor calibration for analyte dispersion.

Following calibration and validation of the model using TGGC experimental data, the stainless-steel column was tested under TPGC conditions to allow for further validation of the transport model. Data from these runs are included in Appendix I along with a comparison of transport model output for a TPGC run with its corresponding experimental separation data. These TPGC data will serve to further refine the capabilities of the transport model.

4.7 TGGC Gradient Quality Studies

4.7.1 GC Heating Mode Comparison

Of primary interest is an understanding of how the separation performance of ITGC, TPGC, and TGGC heating methods compare. Due to the considerable transient and spatial temperature differences between the three methods, it is difficult to find a way of standardizing the conditions for the runs in order to draw conclusions regarding method efficacy. Due to this, it was decided that instead of standardizing conditions, the results were standardized using the non-dimensional GC resolution parameter between two analytes (i and j in Equation 4.1). For ITGC, the conditions from one of the ITGC calibration runs was used in which the entire column was set to 155 °C. For TPGC, the column was programmed to increase 30 °C every minute starting from 50 °C following experimental conditions used on the chip. The conditions for TGGC were the same as those discussed in Section 4.6.

Due to the disparity between elution times under the three heating methods, a comparative chromatogram is not included here. However, the resolution for all the runs are tabulated in Table 4.2. The results from the transport model show that, apart from the resolution between C12 and C13 under TPGC conditions, TGGC provided the best resolution performance. Another notewor-

Table 4.2: Table containing the resolution values for three runs in the transport model under ITGC, TPGC, and TGGC heating conditions

	ITGC		TPGC		TGGC	
Run	$R_{C12,C13}$	$R_{C13,C14}$	$R_{C12,C13}$	$R_{C13,C14}$	$R_{C12,C13}$	$R_{C13,C14}$
1	10.802	13.178	14.360	12.757	13.564	13.593
2	10.460	13.227	14.316	12.509	12.954	13.668
3	10.267	12.878	14.466	12.823	13.570	14.107
Average	10.510	13.094	14.381	12.696	13.363	13.789

thy conclusion provided by the data in Table 4.2 is that TGGC helps to normalize the resolution between the peaks. For both ITGC and TPGC there is a significant difference between $R_{C12,C13}$ and $R_{C13,C14}$, however for TGGC the two values are nearly identical.

It should also be noted that the results presented in Table 4.2 appear to be in conflict with results reported by Blumberg. According to his model of TGGC separation, it is not possible for TGGC to ever surpass ITGC in resolution [42]. It should be noted, however, that Blumberg arrived at this conclusion by assuming that the injection into the column was infinitesimally narrow. Injections presented here were set at 2 cm in width which may explain the disagreement with Blumberg's results. However, more work would be needed to determine the exact cause of the discrepancy between the results presented here and those published by Blumberg. Another qualification that must be noted with respect to these results is the run time associated with each separation. For the results presented here C12 eluted at 76.8 s, 214.7 s, and 143.6 s for ITGC, TPGC, and TGGC respectively. As shorter run times are preferred, TGGC appears to exhibit better performance over TPGC, however a wider set of test conditions would be needed to make broader statements regarding the relative performance of the three heating modes.

4.7.2 Oscillation Effect

Another question of interest is how the presence of oscillations along the gradient in a serpentine column affected the column's ability to focus peaks. The transport model was used to examine the effect of oscillations by comparing TGGC transport model predictions for separation using various temperature profiles along the column, all other conditions remaining the same. Simulated thermal gradient data with oscillations from the run used to validate the model were

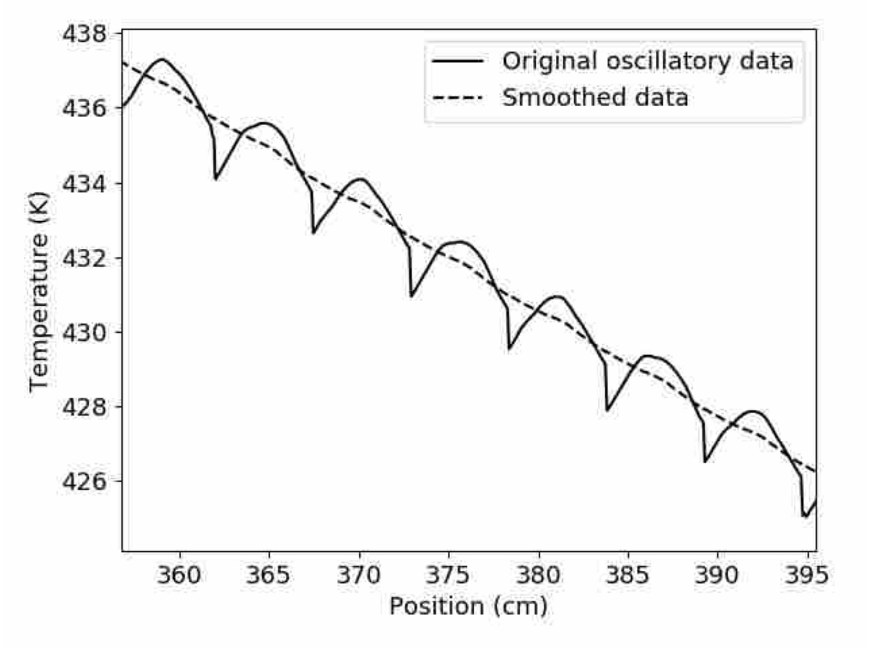


Figure 4.6: Zoomed in plot of the thermal gradient before and after smoothing. The magnitude of the oscillations is approximately 2 K. Compare to Figure 3.13.

smoothed into an oscillation-free temperature profile. The simulated thermal data from the TGGC run was post-processed with a moving window averaging filter using a window sufficiently large so as to smooth the oscillations without deviating from the broader gradient behavior. A segment of the results for this moving window averaging is shown in Figure 4.6.

Once smoothed, the thermal gradient data was run in the transport model multiple times and each result was compared to each result for the oscillatory thermal gradient data. As a visual comparison of the separation differences between the two gradient shapes, a sample chromatogram for each gradient was plotted (Figure 4.7). Note that apart from a slight decrease in elution time, there is no notable difference between the performance of a smooth gradient in comparison to an oscillating gradient. This conclusion is further supported by the values reported in Tables 4.3 and 4.4. Note that in the case of the peak widths at half max, the ranges of values for oscillating and smooth gradient runs for a given analyte overlap. This overlap for each analyte indicates that there is not a statistically significant difference between the peaks widths for the two gradient shapes. From these results it can be concluded that under the conditions experienced by the µTGGC systems used in this project, oscillations of approximately 2 K should have no appreciable effect on the focusing capabilities of the system.

Table 4.3: Table containing the retention time and width at half max values for three runs in the transport model using both oscillating and smooth gradients. Heating conditions for the run match those of Test 7 as reported in Table I.1 in Appendix I

	Retention Time (s)						Width at Half Max (s)					
	Oscillating			Smooth			Oscillating			Smooth		
Run	C12	C13	C14	C12	C13	C14	C12	C13	C14	C12	C13	C14
1	143.56	202.82	267.06	142.90	201.99	266.22	2.41	2.73	2.84	2.39	2.74	2.82
2	143.57	202.79	267.00	142.80	202.01	266.20	2.51	2.87	2.66	2.40	2.79	2.75
3	143.54	202.72	267.06	142.84	201.96	266.25	2.40	2.74	2.63	2.34	2.88	2.81

4.7.3 Gradient Shape Effect

As discussed in Chapter 2 the new μTGGC system that is being developed is a radial system with a spiral column. This new method for establishing a thermal gradient along the column was selected initially because it allowed for a monotonic gradient unlike the serpentine channels (as in the silicon and stainless-steel rectangular columns). However, thermal modeling has suggested that these spiral column systems provide a linear thermal gradient along the column length due to the heat transfer behavior inherent to radial systems (See Figure 3.9). One question from preliminary

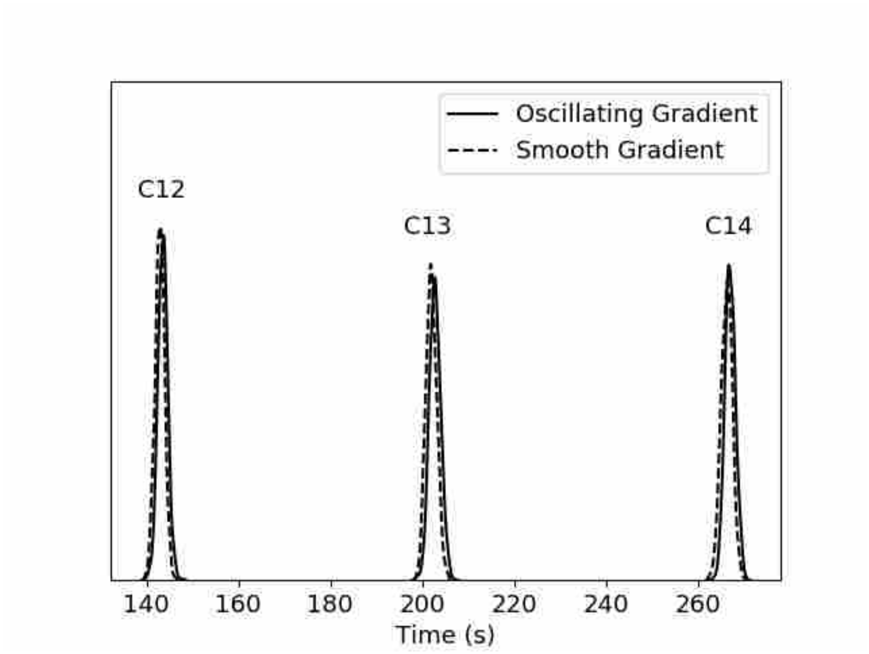


Figure 4.7: Comparison of peak shape and elution time between the the oscillating and smooth gradients as computed by the transport model. Heating conditions for the run match those of Test 7 as reported in Appendix I

Table 4.4: Table containing the resolution values for three runs in the transport model using both an oscillating and a smooth gradient

	Oscillating		Smooth	
Run	$R_{C12,C13}$	$R_{C13,C14}$	$R_{C12,C13}$	$R_{C13,C14}$
1	13.575	13.579	13.562	13.602
2	12.960	13.671	13.432	13.642
3	13.556	14.107	13.335	13.303
Average	13.364	13.786	13.443	13.516

thermal studies is how the separation characteristics along a linear gradient will compare with an exponentially decaying gradient. In order to explore this effect a fixed temperature drop of 50 °C along the length of the column was selected to define a linear gradient along the column length. A corresponding exponentially decaying gradient was developed by setting the start temperature equal to the linear gradient with a starting slope 3 times as steep as the linear gradient. The shape was then constrained by making the average values of the two gradients equal. Both of these gradient shapes were then ramped at the same constant rate so that the inlet and outlet temperatures of both gradients would always be the same. The transient ramping that was selected was to

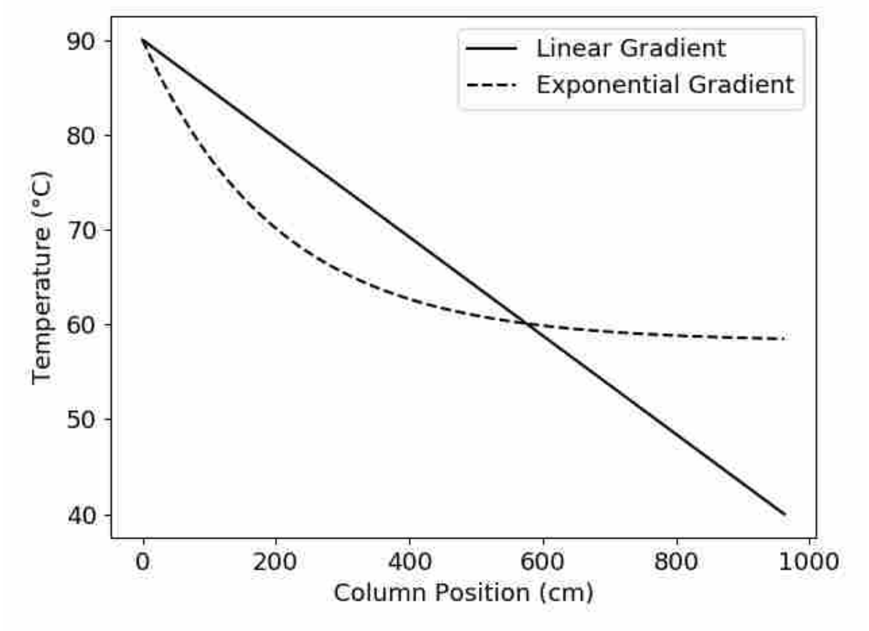


Figure 4.8: Comparison of linear and exponential gradient shapes at their initial condition. Both gradients were moved upward linearly at the same rate.

ramp the outlet temperature from 40 °C to 250 °C in 12 minutes. The two temperature profiles were standardized in this way so as to minimize variation between the two runs and allow direct comparison between linear and exponential gradients (See Figure 4.8). The two gradient shapes were each imported into the transport model and run three times. Their results were then compared to determine the effect that the shape of the gradient has on separation characteristics.

As can be seen in Figure 4.9 there is an slight difference between the retention times and peak widths for exponential versus linear gradients. The retention time and peak width at half max values are given in Table 4.5, and the resolution values are reported in Table 4.6. The resultant resolution values show a resolution increase of approximately 1 for the linear gradient compared to the exponential gradient. It should be noted, however, that this result is specific to the gradients used in the runs reported here and cannot be extended to all linear and exponential gradients. However, for the case considered here, the differences can be understood by considering the two gradients. The longer elution times for the exponential gradient are a result of the fact that over the majority of the column the exponential gradient's temperature is lower than the linear gradient.

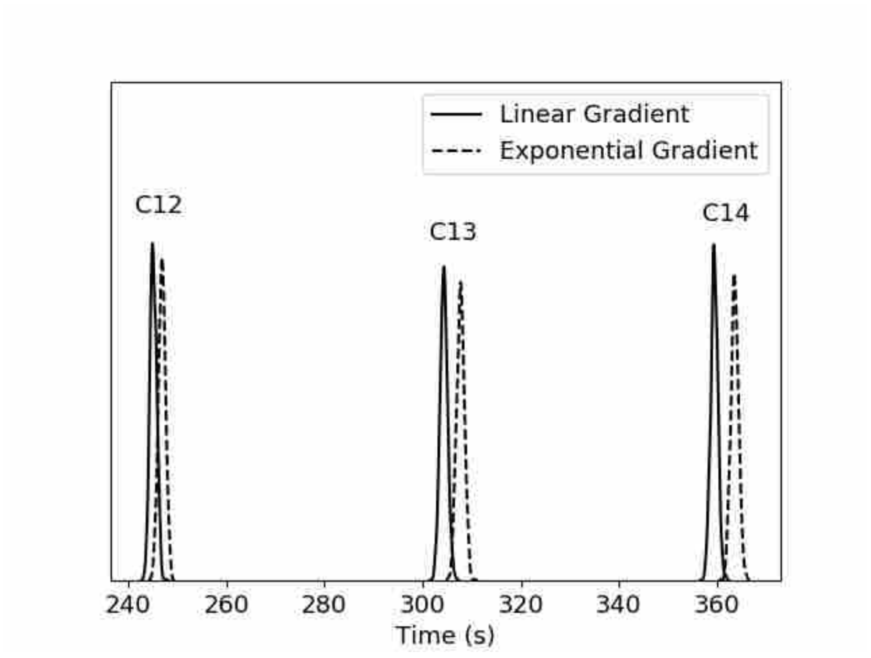


Figure 4.9: Comparison of peak shape and elution time between the the linear and exponential gradients as computed by the transport model.

The broader peaks for the exponential gradient is a result of the shallower temperature gradient at the column outlet compared to the linear gradient.

Table 4.5: Table containing the retention time and width at half max values for three runs in the transport model using both linear and exponential gradients.

	Retention Time (s)						Width at Half Max (s)					
	Linear			Exponential			Linear			Exponential		
Run	C12	C13	C14	C12	C13	C14	C12	C13	C14	C12	C13	C14
1	245.15	304.33	359.45	246.92	307.74	363.61	1.59	1.77	1.70	1.69	1.87	1.85
2	245.19	304.31	359.45	246.93	307.74	363.56	1.61	1.76	1.74	1.81	1.87	1.87
3	245.13	304.32	359.42	246.92	307.74	363.55	1.64	1.72	1.81	1.69	1.92	1.80

Table 4.6: Table containing the resolution values for three runs in the transport model using both linear and exponential gradients.

	Linear		Exponential	
Run	$R_{C12,C13}$	$R_{C13,C14}$	$R_{C12,C13}$	$R_{C13,C14}$
1	20.738	18.703	20.115	17.683
2	20.655	18.549	19.456	17.573
3	20.741	18.378	19.837	17.664
Average	20.711	18.543	19.803	17.640

4.8 Conclusion

The transport model previously developed by Dr. Tolley was translated into Python and then tested to ensure proper functionality. Data were taken in both ITGC and TGGC heating methods for use in calibrating and validating the model. Models for retention factor and mass diffusion in gases were employed and shown to adequately match the results observed experimentally. Following validation of the transport model it was applied to understand the effects of gradient oscillations and linearity on TGGC separation performance. The results from the transport model indicated that the presence of oscillations had a negligible effect on separation performance. Alternatively it was found that for the exponential gradient considered slower separations and broader the peaks are produced when compared to separations on a corresponding linear gradient.

CHAPTER 5. CONCLUSION

Based on previous work in the fields of µGC and TGGC, it is apparent that the synthesis of these two fields is a promising direction for developing new miniaturized GC devices. The present work has focused efforts within this sphere of µTGGC in order to help advance understanding in the field. The three primary objectives of this work were to (1) develop a TGGC experimental system to allow for µTGGC experimentation and data acquisition, (2) develop computational thermal models to provide better insight into the thermal characteristics of µTGGC systems, and (3) use experimental and simulated thermal data to validate a versatile GC transport model and explore the effects the temperature gradient has on separation.

5.1 Thermal Control and Data Acquisition System Development (Chapter 2)

The first objective of the preceding work was to develop an experimental system that would allow for implementation of TGGC heating and control on various µGC columns while providing temperature and power consumption feedback for use in post-processing. The experimental system circuitry and control code were developed in-house and tested on various column layouts. The system was shown to be successful in completing its function as well as being an adaptable system in that it was successfully used to control TGGC heating on three µGC columns, allowing for separation up to C40 on the spiral column system. The system was also shown to be a functional data acquisition tool in that the data recorded by the DAQ was successfully used in both the development of the computational thermal models as well as in calibrating and validating the GC transport model.

5.2 Computational Thermal Model Development (Chapter 3)

The second objective in the reported work was to develop a highly accurate computational thermal model of experiments performed using the data recorded by the controller/DAQ system.

Various periphery experiments were performed to ensure that the conditions used in the models were valid. The methods used in modeling the various μGC systems were shown to be successful through the validation of models against experimental data for three experimental systems. In all cases the error in the simulation was kept below 5 °C. The thermal modeling methods also proved effective in the development of various models used to explore potential new μGC column layouts.

5.3 GC Transport Model Validation and Analysis (Chapter 4)

The third objective of the preceding work was to use the data from ITGC and μTGGC runs along with the data from the thermal simulations to both calibrate and validate a GC transport model. Following calibration and validation this model was to be used to explore the effects of various heating methods on separation performance. TGGC was found to improve resolution by approximately 1.15 times compared to ITGC for the chosen injection conditions and although TPGC offered slightly better resolution, TGGC equalizes resolution across various analytes. Tests were also run to determine how alterations to the gradient quality would affect TGGC separation performance. The transport model indicated that gradient oscillations had no effect on the GC separation performance for the stainless-steel column used in experiments, and the exponential gradient considered resulted in a decrease in resolution of approximately 1 when compared to separations on linear gradients.

5.4 Future Work

In light of the successes achieved in the present work, there are several areas that have been identified as possible areas for future work. These possibilities can be broadly separated into three categories: advancements in the experimental system, computational thermal modeling, and GC transport modeling.

5.4.1 Experimental System

Although the TGGC control/DAQ system was shown to successfully control a variety of μTGGC systems and gather sufficient data for thermal model development, there are a few aspects of the system design that could be improved upon. As was shown earlier, the voltage and current readings provided by the circuitry were subject to significant noise. Part of this is a result of aliasing produced by the PWM signal in addition to noise in the data collection circuitry. In order to reduce the noise in the data the circuit design could be improved upon and the method for filtering the data could be adapted to better address the aliasing from the PWM signal. The ability to read voltage and current data with less noise would be useful both in developing thermal models for future systems as well in characterizing and minimizing the power consumption in future µTGGC systems.

Another area that could be improved in the experimental system is the fan control capabilities. In the present setup, the fan does a poor job of offering fine-tuned control of tip cooling for low control values. This is a result of the fan's non-linear behavior at low voltage and current values. In order to improve this, an electronic speed controller could be used or a fan with smoother dynamics could be replaced in the system. Such an approach would allow for finer control of the tip temperature on the spiral system, which would in turn help to improve separation performance in this system.

5.4.2 Computational Thermal Model

As explained previously, a significant portion of the work required to make thermal models of various systems match correctly has already been performed for the previous µTGGC systems. Moving forward, the computational methods developed for the systems that have been modeled here can be applied to the spiral column design to develop an associated high fidelity model for that system. As was the case with each of the models developed for this work, the development of a spiral column thermal model will likely present unanticipated challenges that will help to further refine the thermal modeling method developed here. Following the completion of the spiral column model, the same methods will be useful in developing future system models as the µTGGC project at BYU moves on to new µGC column designs.

5.4.3 GC Transport Model

The GC transport model is largely successful in modeling complex µTGGC separations and may be used as a powerful tool in the design and study of µTGGC systems. However, before applying the transport model to its various uses, the TPGC data that was collected on the stainless-steel column will be incorporated in the transport model calibration methods to improve its results. Once the model calibration has been refined in this way, the transport model will be used as a design tool in developing new µTGGC systems. Previously, no useful tool has existed to accelerate iterations through various µTGGC system designs. Moving forward, the transport model can be used as a tool for predicting chromatographic performance making it possible to eliminate non-optimal designs without the cost of actually producing a system. This should allow for much faster more efficient convergence on an optimal µTGGC system design.

A second area in which the transport model may be useful is as a diagnostic tool. Presently when running TGGC separations if poor separation quality is achieved the chromatographer is only able to guess what the cause of the problem might be, based on past experience. In order to aid in system troubleshooting the transport model can be run in non-ideal conditions (e.g., presence of cold spots, uneven coating thickness, and poor gradient shape), and the effect of these conditions can be studied and cataloged. Results can then be used to help chromatographers determine the cause of issues they are seeing in their chromatographic system, which could significantly speed up the troubleshooting process when running TGGC experiments.

REFERENCES

[1] McMurry, J. E., 2014. *Organic Chemistry with Biological Applications*. Cengage Learning, Boston, MA. 1

[2] Gehrke, C. W., Wixom, R. L., and Bayer, E., 2001. *Chromatography-A Century of Discovery 1900-2000. The Bridge to The Sciences/Technology*., Vol. 64 Elsevier, Amsterdam, Netherlands. 2

[3] James, A. T., and Martin, u. A., 1952. "Gas-liquid partition chromatography: the separation and micro-estimation of volatile fatty acids from formic acid to dodecanoic acid." *Biochemical Journal,* **50**(5), p. 679. 2, 5

[4] Ettre, L. S., and Zlatkis, A., 2011. *75 years of chromatography: a historical dialogue*., Vol. 17 Elsevier, Amsterdam, Netherlands. 2

[5] Terry, S. C., Jerman, J. H., and Angell, J. B., 1979. "A gas chromatographic air analyzer fabricated on a silicon wafer." *IEEE Transactions on Electron Devices,* **26**(12), pp. 1880–1886. 6

[6] Reston, R. R., and Kolesar, E., 1994. "Silicon-micromachined gas chromatography system used to separate and detect ammonia and nitrogen dioxide. i. design, fabrication, and integration of the gas chromatography system." *Journal of Microelectromechanical Systems,* **3**(4), pp. 134–146. 6

[7] Yu, C., Lucas, M., Koo, J., Stratton, P., DeLima, T., Behmeyer, E., and Gas, A. H.-p. H.-h., 1998. "Chromatograph." *Proceedings of the ASME, Micro-Electro-Mechanical Systems*, pp. 481–486. 6

[8] Radadia, A. D., Morgan, R. D., Masel, R. I., and Shannon, M. A., 2009. "Partially buried microcolumns for micro gas analyzers." *Analytical Chemistry,* **81**(9), pp. 3471–3477. 6

[9] Yuan, H., Du, X., Tai, H., Li, Y., Zhao, X., Guo, P., Yang, X., Su, Y., Xiong, Z., and Xu, M., 2017. "The effect of the channel curve on the performance of micromachined gas chromatography column." *Sensors and Actuators B: Chemical,* **239**, pp. 304–310. 6

[10] Lewis, A. C., Hamilton, J. F., Rhodes, C. N., Halliday, J., Bartle, K. D., Homewood, P., Grenfell, R. J., Goody, B., Harling, A. M., Brewer, P., et al., 2010. "Microfabricated planar glass gas chromatography with photoionization detection." *Journal of Chromatography A,* **1217**(5), pp. 768–774. 6

[11] Briscoe, C. G., Yu, H., Grodzinski, P., Huang, R.-F., and Burdon, J. W., 2003. Multilayered ceramic micro-gas chromatograph and method for making the same, Mar. 4 US Patent 6,527,890. 6

[12] Lewis, P. R., and Wheeler, D. R., 2007. Non-planar microfabricated gas chromatography column, Sept. 25 US Patent 7,273,517. 6

[13] Malainou, A., Vlachopoulou, M., Triantafyllopoulou, R., Tserepi, A., and Chatzandroulis, S., 2008. "The fabrication of a microcolumn for gas separation using poly (dimethylsiloxane) as the structural and functional material." *Journal of Micromechanics and Microengineering,* **18**(10), p. 105007. 6

[14] Plummer, J. D., and Griffin, P. B., 2001. "Material and process limits in silicon vlsi technology." *Proceedings of the IEEE,* **89**(3), pp. 240–258. 6

[15] Zhukhovitskii, A., Zolotareva, O., Sokolov, V., and Turkeltaub, N., 1951. "New method of chromatographic analysis." In *Dokl. Akad. Nauk SSSR*, Vol. 77, pp. 435–438. 8, 44

[16] Tudge, A., 1962. "Studies in chromatographic transport: Iii. chromathermography." *Canadian Journal of Physics,* **40**(5), pp. 557–572. 8

[17] Fatscher, M., and Vergnaud, J., 1969. "Gas phase chromatography with longitudinal temperature gradient." *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences Serie C,* **269**(3), p. 219. 8

[18] Fenimore, D. C., 1975. "Gradient temperature programming of short capillary columns." *Journal of Chromatography A,* **112**, pp. 219–227. 8

[19] Zizin, V. G., and Makov, N. I., 1979. "New method of thermochromatographic analysis." *Industrial Laboratory,* **45**(12), pp. 1328–1331. 8

[20] Jain, V., and Phillips, J. B., 1995. "Fast temperature programming on fused-silica open-tubular capillary columns by direct resistive heating." *Journal of chromatographic science,* **33**(1), pp. 55–59. 9, 11

[21] Jain, V., and Phillips, J. B., 1995. "High-speed gas chromatography using simultaneous temperature gradients in both time and distance along narrow-bore capillary columns." *Journal of chromatographic science,* **33**(11), pp. 601–605. 9, 11

[22] Phillips, J. B., and Jain, V., 1995. "On-column temperature programming in gas chromatography using temperature gradients along the capillary column." *Journal of chromatographic science,* **33**(10), pp. 541–550. 9, 11

[23] Blumberg, L., 1997. "Focusing cannot enhance resolution or speed limit of a GC column." *Journal of Chromatographic Science,* **35**(9), pp. 451–454. 9

[24] Contreras, J. A., Rockwood, A. L., Tolley, H. D., and Lee, M. L., 2013. "Peak sweeping and gating using thermal gradient gas chromatography." *Journal of Chromatography A,* **1278**, pp. 160–165. 9, 11

[25] Zhao, H., Yu, L., Zhang, J., and Guan, Y., 2002. "Characteristics of TGPGC on short micro packed capillary column." *Analytical Sciences,* **18**(1), pp. 93–95. 9, 11

[26] Contreras, J. A., Wang, A., Rockwood, A. L., Tolley, H. D., and Lee, M. L., 2013. "Dynamic thermal gradient gas chromatography." *Journal of Chromatography A,* **1302**, pp. 143–151. 9, 11

[27] Wang, A., Hynynen, S., Hawkins, A. R., Tolley, S. E., Tolley, H. D., and Lee, M. L., 2014. "Axial thermal gradients in microchip gas chromatography." *Journal of Chromatography A,* **1374**, pp. 216–223. 9, 10, 11

[28] Boeker, P., and Leppert, J., 2015. "Flow field thermal gradient gas chromatography." *Analytical chemistry,* **87**(17), pp. 9033–9041. 9, 11

[29] Ghosh, A., Johnson, J. E., Nuss, J. G., Stark, B. A., Hawkins, A. R., Tolley, L. T., Iverson, B. D., Tolley, H. D., and Lee, M. L., 2017. "Extending the upper temperature range of gas chromatography with all-silicon microchip columns using a heater/clamp assembly." *Journal of Chromatography A,* **1517**, pp. 134–141. 9, 15, 24

[30] Golay, M., 1956. "Progress report of gas chromatographic experimental work for september and october 1956." *The Perkin-Elmer Corp.* 10

[31] Guiochon, G., and Guillemin, C. L., 1988. *Quantitative gas chromatography for laboratory analyses and on-line process control.*, Vol. 42 Elsevier. 10, 44

[32] Blumberg, L. M., 2010. *Temperature-programmed gas chromatography*. John Wiley & Sons, Hoboken, NJ. 10, 47

[33] Van Deemter, J., Zuiderweg, F., and Klinkenberg, A. v., 1956. "Longitudinal diffusion and resistance to mass transfer as causes of nonideality in chromatography." *Chemical Engineering Science,* **5**(6), pp. 271–289. 10

[34] Golay, M. J., et al., 1958. "Theory of chromatography in open and coated tubular columns with round and rectangular cross-sections." *Gas Chromatography,* **2**, pp. 36–55. 10

[35] Giddings, J. C., 1960. "Theoretical basis for kinetic effects in gas-solid chromatography." *Nature,* **188**(4753), p. 847. 10

[36] Giddings, J. C., 1963. "Advances in the theory of plate height in gas chromatography.." *Analytical Chemistry,* **35**(4), pp. 439–449. 10

[37] Rowan, R., 1979. "Differential programmed temperature gas chromatography." *Analytical Chemistry,* **51**(9), pp. 1540–1545. 10

[38] Duarte, P., and McCoy, B., 1982. "Stationary spatial temperature gradient in gas chromatography." *Separation Science and Technology,* **17**(7), pp. 879–896. 10

[39] Blumberg, L. M., and Berger, T. A., 1992. "Variance of a zone migrating in a non-uniform time-invariant linear medium." *Journal of Chromatography A,* **596**(1), pp. 1–13. 10, 44

[40] Blumberg, L. M., 1992. "Outline of a theory of focusing in linear chromatography." *Analytical Chemistry,* **64**(20), pp. 2459–2460. 10

[41] Blumberg, L. M., 1993. “Variance of a zone migrating in a linear medium: Ii. time-varying non-uniform medium.” *Journal of Chromatography A,* **637**(2), pp. 119–128. 10

[42] Blumberg, L., 1994. “Limits of resolution and speed of analysis in linear chromatography with and without focusing.” *Chromatographia,* **39**(11-12), pp. 719–728. 10, 54

[43] Tolley, H. D., Tolley, S. E., Wang, A., and Lee, M. L., 2014. “Moving thermal gradients in gas chromatography.” *Journal of Chromatography A,* **1374**, pp. 189–198. 10, 45

[44] Adafruit, 2019. Analog output k-type thermocouple amplifier - AD8495 breakout, https://www.adafruit.com/product/1778?gclid=cj0kcqiatvpjbrdparisajfzz0r5wkpzs8tmvcb9a1epybzaq22oerhnot7583zhveg7r-m5o8sr5oqaaljlealw_wcb, Apr. 24

[45] Bergman, T. L., Incropera, F. P., DeWitt, D. P., and Lavine, A. S., 2011. *Fundamentals of heat and mass transfer*. John Wiley & Sons, Hoboken, NJ. 32, 115

[46] Boyd Corporation, 2019. Solimide HT-340 data sheet, https://www.boydcorp.com/datasheets/solimide-ht-340-data-sheet.pdf, May. 34

[47] Chapman, S., Cowling, T. G., and Burnett, D., 1990. *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*. Cambridge University Press, Cambridge, England. 48

[48] White, F. M., and Corfield, I., 2006. *Viscous fluid flow.*, Vol. 3 McGraw-Hill New York, New York, NY. 137, 139

APPENDIX A.   CONTROLLER CODES

As described in chapter 2, the thermal control and DAQ system used an Arduino Mega microcontroller to interface with the system hardware. The code for controlling the Arduino was written in C and C++. In order to make user interaction with the controller possible, a GUI was written in python. One of the significant difficulties in developing the necessary control and GUI code was determining a reliable method for live communication between the GUI and the Arduino. In order to achieve this a serial communication routine was developed to maintain order between the two codes as data was being sent both ways on the serial communication line between the Arduino and computer.

The code for the Arduino was responsible for controlling the power input to the μTGGC system. A proportional integral control method was implemented as the dynamics of the thermal systems involved were slow enough that derivative control was not necessary (gain values reported in lines 27-56 in Section A.1). Along with controlling the heaters, the Arduino recorded temperature data and performed a filtering routine on the voltage and current data that was collected. These three data were then sent to the computer so it could be saved in a comma separated value (csv) file.

The Python GUI served the purpose of allowing the user to interface with the Arduino controller as it was running. This was important as it allowed the user to control the moment at which each heater would be turned on or off, and alter set points of the various heaters mid run. Another important aspect of the Python code was that on the backend it was collecting the data sent by the Arduino and storing it in a csv file that could be accessed following completion of the run.

The Arduino code consists of a main script along with a header file, and the Python code is a single file. All three have been included below.

A.1 Arduino Main Script

```
#include "ControlDAQ.h"

General g;
Control prmCtrl = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control scndCtrl = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control lnCtrl = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control inTLCtrl = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control outTLCtrl = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control T1 = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control T2 = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control T3 = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control T4 = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};
Control T5 = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1};

void setup() {

  // Serial setup
  Serial.begin(9600);

  // Debugging LED pin
  pinMode(LEDpin,OUTPUT);

  // General parameters
  g.numChars = 10;

  // Controller parameters
  prmCtrl.Kp = 15;
  prmCtrl.Ki = 0.25;
  prmCtrl.ctrlPin = 2;
  prmCtrl.voltPin = 15;
  prmCtrl.currentPin = 14;
  prmCtrl.feedbackPin = 1;

  scndCtrl.Kp = 15;
  scndCtrl.Ki = 0.25;
  scndCtrl.ctrlPin = 2;
  scndCtrl.voltPin = 13;
  scndCtrl.currentPin = 12;
  scndCtrl.feedbackPin = 3;

  lnCtrl.Kp = 15;
  lnCtrl.Ki = 0.25;
  lnCtrl.ctrlPin = 2;
  lnCtrl.voltPin = 10;
  lnCtrl.currentPin = 11;
  lnCtrl.feedbackPin = 2;

  inTLCtrl.Kp = 20;
  inTLCtrl.Ki = 0.1;
  inTLCtrl.ctrlPin = 3;
  inTLCtrl.voltPin = 9;
  inTLCtrl.currentPin = 8;
  inTLCtrl.feedbackPin = 4;

  outTLCtrl.Kp = 20;
  outTLCtrl.Ki = 0.1;
  outTLCtrl.ctrlPin = 5;
  outTLCtrl.voltPin = 7;
  outTLCtrl.currentPin = 6;
  outTLCtrl.feedbackPin = 5;

  T1.feedbackPin = 5;
  T2.feedbackPin = 6;
  T3.feedbackPin = 1;
  T4.feedbackPin = 8;
  T5.feedbackPin = 9;
}

void loop() {

  // Collect time data
  g.timeLast = g.timeNow;
  g.timeNow = millis()/1000.0;

  // Check for data sent
  dataChainReader(g, prmCtrl, scndCtrl, lnCtrl, inTLCtrl, outTLCtrl);

  // Record and Report data
  Serial.println('<');
  Serial.println(g.timeNow,4);
  sendData(prmCtrl);
  sendData(scndCtrl);
  sendData(lnCtrl);
  sendData(inTLCtrl);
  sendData(outTLCtrl);
  sendTemp(T1);
  sendTemp(T2);
  sendTemp(T3);
  sendTemp(T4);
  sendTemp(T5);
  Serial.println('>');

  // Implement control
  tempCtrl(g.timeNow,g.timeNow-g.timeLast, prmCtrl);
  tempCtrl(g.timeNow,g.timeNow-g.timeLast, scndCtrl);
  tempCtrl(g.timeNow,g.timeNow-g.timeLast, lnCtrl);
  tempCtrl(g.timeNow,g.timeNow-g.timeLast, inTLCtrl);
  tempCtrl(g.timeNow,g.timeNow-g.timeLast, outTLCtrl);
}
```

A.2 Arduino Header File

```
  // ---- Data structure ---- //
  // General data
  struct General {
    boolean newData;
    int dataCount;
    int numData;
    int controllerNum;
    float timeNow;
    float timeLast;
    byte numChars;
    char receivedChars[10];
  };
  // Controller data
  struct Control {
    bool on;
    float setPoint;
    float tempFinal;
    float rampTime;
    float m;
    float b;
    float error;
    float errorSum;
    float Kp;
    float Ki;
    float PWMfloat;
    int PWM;
    int ctrlPin;
    bool overshoot;
    float helpFactor;
    int feedbackPin;
    int voltPin;
    int currentPin;
    float initialTemp;
    float initialTime;
    bool initial;
  };

  // ---- Function Definitions ---- //

  // Print data
```

```
void sendData(Control& c){
  Serial.println(analogRead(c.feedbackPin));
  Serial.println(c.setPoint);
  Serial.println(c.PWM);
  Serial.println(analogRead(c.voltPin));
  Serial.println(analogRead(c.currentPin));
}

// Print temperature
void sendTemp(Control& c){
  Serial.println(analogRead(c.feedbackPin));
}

// Temperature controller
void tempCtrl(float t, float deltaT, Control& c){

  if(c.on){
    // Compute set point
    if(c.initialTemp == c.tempFinal){
      c.setPoint = c.initialTemp;
    }
    else if (c.initial){
      c.setPoint = c.initialTemp;
      c.initial = 0;
    }
    else if(t < c.initialTime+c.rampTime*60.0){
      c.setPoint = c.m*t+c.b;
    }
    else{
      c.setPoint = c.tempFinal;
    }

    // Implement control
    float temp = ((5.0*analogRead(c.feedbackPin)/1023.0)-1.25)/0.005;
    c.error = c.setPoint - temp;
    c.errorSum = c.errorSum + c.error;
    c.PWMfloat = c.Kp*c.error + c.Ki*deltaT*c.errorSum;
    // Prevent overshoot
    if (c.PWMfloat > 255){
      c.PWMfloat = 255;
      c.errorSum = c.errorSum - c.error;
      c.overshoot = 1;
    }
    else if (c.PWMfloat < 0){
      c.PWMfloat = 0;
      c.errorSum = c.errorSum;
    }
    else{
      c.overshoot = 0;
    }
    // Implement control
    c.PWM = int(c.PWMfloat);
    analogWrite(c.ctrlPin,c.PWM);
  }
}

// Data converter
float convertData(General& g){
  float val;
  if(g.newData){
    val = atof(g.receivedChars);
    g.newData = false;
  }
  return val;
}

// Data reciever
void recvWithStartEndMarkers(General& g) {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

  if (Serial.available() > 0) {
    while (Serial.available() > 0 && g.newData == false) {
        rc = Serial.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                g.receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= g.numChars) {
                    ndx = g.numChars - 1;
                }
            }
            else {
                g.receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
                g.newData = true;
            }
        }

        else if (rc == startMarker) {
            recvInProgress = true;
        }
    }
}
}

// Data chain reader
void dataChainReader(General& g, Control& a, Control& B, Control& c,
    Control& d, Control& e){

  // Read in data
  recvWithStartEndMarkers(g);

```

```
// Check for data start
if(g.receivedChars[0] == 's'){
  g.dataCount++;
  g.newData = false;
}
else if(g.dataCount==1){
  g.numData = convertData(g);
  g.newData = false;
  g.dataCount++;
}
else if(g.dataCount==2){
  g.controllerNum = convertData(g);
  g.newData = false;
  g.dataCount++;
}
else if(g.dataCount==3){

  if(g.controllerNum == 1){
    a.on = 1;
    a.initialTemp = convertData(g);
    a.tempFinal = a.initialTemp;
  }
  else if(g.controllerNum == 2){
    a.on = 1;
    a.tempFinal = convertData(g);
    a.initialTemp = a.setPoint;
    a.initialTime = millis()/1000.0;
  }
  else if(g.controllerNum == 3){
    B.on = 1;
    B.tempFinal = convertData(g);
    B.initialTemp = (5.0*analogRead(B.feedbackPin)/1023.0-1.25)/0.005;
    B.initialTime = millis()/1000.0;
  }
  else if(g.controllerNum == 4){
    c.on = 1;
```

```
      c.initialTemp = convertData(g);
      c.tempFinal = c.initialTemp;
    }
    else if(g.controllerNum == 5){
      d.on = 1;
      d.initialTemp = convertData(g);
      d.tempFinal = d.initialTemp;
    }
    else if(g.controllerNum == 6){
      e.on = 1;
      e.initialTemp = convertData(g);
      e.tempFinal = e.initialTemp;
    }
    else if(g.controllerNum == 7){
      a.on = 0;
      analogWrite(a.ctrlPin,0);
      a.setPoint = 0;
    }
    else if(g.controllerNum == 8){
      B.on = 0;
      analogWrite(B.ctrlPin,0);
      B.setPoint = 0;
    }
    else if(g.controllerNum == 9){
      c.on = 0;
      analogWrite(c.ctrlPin,0);
      c.setPoint = 0;
    }
    else if(g.controllerNum == 10){
      d.on = 0;
      analogWrite(d.ctrlPin,0);
      d.setPoint = 0;
    }
    else if(g.controllerNum == 11){
      e.on = 0;
      analogWrite(e.ctrlPin,0);
      e.setPoint = 0;
    }
    g.newData = false;
    g.dataCount++;
  }
  else if(g.dataCount==4&&g.numData!=4){
    if(g.controllerNum == 2){
      a.rampTime = convertData(g);
      a.m = (a.tempFinal-a.initialTemp)/(a.rampTime*60.0);
      a.b = a.initialTemp - a.m*a.initialTime;
    }
    else if(g.controllerNum == 3){
      B.rampTime = convertData(g);
      B.m = (B.tempFinal-B.initialTemp)/(B.rampTime*60.0);
      B.b = B.initialTemp - B.m*B.initialTime;
    }
    g.newData = false;
    g.dataCount++;
  }
  else if(g.dataCount>g.numData){
    g.controllerNum = 0;
    g.newData = false;
    g.dataCount = 0;
  }
}
```

A.3 Python GUI Script

```
#-------------------------------------------------------
# Title: Complete Controller - Python Script
# Author: Austin Foster
# Date: January 17, 2018
# Description:  This is a first attempt at the
#               completed controller
#-------------------------------------------------------

```

```python
# Import necessary modules
import numpy as np
import tkinter as tk
from tkinter import ttk
import serial
import csv
from time import sleep
import sys

# Define constants
global app
LARGE_FONT = ("Verdana", 12)
NORMAL_FONT = ("Verdana", 10)
SMALL_FONT = ("Verdana", 8)
global params
params = np.array([70, 300, 5, 10, 5, 0])
global time, tempPrim, liveTemp, DAQ
time = 0
temp1 = 0
DAQ = 0

# Connect serial port
global ser
ser = serial.Serial('COM4', 9600, timeout=1) # Establish the connection
    on a specific port

# Define classes
class PerkinElmerProjectDAQ(tk.Tk):

    # Define methods
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, string="Perkin Elmer Project DAQ - Created
    by Austin Foster")
        # Create a container. You always need a container
        container = tk.Frame(self)      # Frame is the edge of the window
        container.pack(side="top",fill="both",expand=True)

        # Do dome configureation
        container.grid_rowconfigure(0,weight=1)
        container.grid_columnconfigure(0,weight=1)

        # Specify a dictionary
        self.frames = {}

        frame = DAQdisplay(container, self)            # StartPage is defined later

        self.frames[DAQdisplay] = frame

        frame.grid(row=0, column=0, sticky="nsew")      # Sticky is basically alignment and stretch. nsew = north, south, east west

        self.show_frame(DAQdisplay)

    def show_frame(self, cont):             # cont is the controller

        frame = self.frames[cont]
        frame.tkraise()             # This command raises to the front

    def toggleLED(self):
        ser.write(b'g')
        return 1

    def updateData(self,data,handle):
        handle.config(text = str(data))

    def theInfiniteLoop(self,bottonHandles,startHandle,liveTempHandles,fileHandle):

        # Create data file
        fileLead = r'C:\Users\PEGC\Desktop\DAQdata'
        fileLead = fileLead + '\\'
        filename = fileHandle.get()
        filenameFull = fileLead + filename + '.csv'
        outfile = open(filenameFull,'w',newline='')
        writer = csv.writer(outfile)
        writer.writerow(('Time (s)','Tip Temp (ArdV)','Tip Set Point (C)'
    ,'Tip PWM (/255)','Tip Voltage (ArdV)','Tip Current (ArdV)','
    Secondary Temp (ArdV)','Secondary Set Point (C)','Secondary PWM
    (/255)','Secondary Voltage (ArdV)','Secondary Current (ArdV)','
    Primary Temp (ArdV)','Primary Set Point (C)','Primary PWM (/255)','
    Primary Voltage (ArdV)','Primary Current (ArdV)','TL1 Temp (ArdV)','
    TL1 Set Point (C)','TL1 PWM (/255)','TL1 Voltage (ArdV)','TL1 Current
     (ArdV)','TL2 Temp (ArdV)','TL2 Set Point (C)','TL2 PWM (/255)','TL2
    Voltage (ArdV)','TL2 Current (ArdV)','BetweenTipAnd2ndary (ArdV)','
    Between2ndaryAnd1maryClose22ndary (ArdV)','
    Between2ndaryAnd1maryClose21mary (ArdV)','InletNut (ArdV)','Outletnut
     (ArdV)'))

        # Make buttons active
        for i in range(len(bottonHandles)):
            bottonHandles[i].config(state='active')
        startHandle.config(state='disabled')

        dataRun = 0
        dataArr = []

        running = 1

        while running:

            if filename == '':
                print('Enter a filename before starting run')
                for i in range(len(bottonHandles)):
                    bottonHandles[i].config(state='disabled')
                startHandle.config(state='active')
                running = 0

            try:

                # Record data
                data = ser.readline().decode().split(’\r\n’)    # Read in data line
                if data[0]==’>’ and len(dataArr)>0:     # Check for data line end character and that the data array is not empty
                    for i in range(len(liveTempHandles)):
                        if float(dataArr[5*i+2])>0.5:
                            sp = str(dataArr[5*i+2])
                        else:
                            sp = ’-’
                        liveTempHandles[i].config(text = sp + ’  ;  ’ + str(round((5.*float(dataArr[5*i+1])/1023.-1.25)/0.005,1)))
                    writer.writerow(dataArr)    # Write data to file
                    dataArr = []                # Clear data array
                    dataRun = 0                 # Reset data acquisiton state
                if dataRun:                             # Check data acquisition state
                    dataArr.append(data[0])     # Record data to array
                if data[0]==’<’:                        # Check for data line start character
                    dataRun = 1                 # Commence data acquisition

                app.update_idletasks()
                app.update()

            except:
                print(’DAQ Connection Compromised’)
                for i in range(len(bottonHandles)):
                    bottonHandles[i].config(state=’disabled’)
                startHandle.config(state=’active’)
                running = 0
```

```

    def stopDAQ(self):
        quit()
        #outfile.close()

    def buttonFunc(self,onBtnHndl,offBtnHndl,lblNum,entryHndl,
rwknEntryHndl):

        # Turn on necessary bottons
        for i in range(len(onBtnHndl)):
            onBtnHndl[i].config(state='active')

        # Turn off necessary buttons
        for i in range(len(offBtnHndl)):
            offBtnHndl[i].config(state='disable')

        # Turn on entries
        for i in range(len(rwknEntryHndl)):
            rwknEntryHndl[i].config(state='normal')

        # Build parameter list
        params = [b's',len(entryHndl)+1,lblNum]
        for i in range(len(entryHndl)):
            params.append(float(entryHndl[i].get()))
            entryHndl[i].config(state='disable')

        # Send parameters
        for i in range(len(params)):
            ser.write(b'<')
            if i==0:
                sendData = params[i]
            else:
                sendData = str.encode(str(params[i]))
            ser.write(sendData)
            ser.write(b'>')

```

```python

class DAQdisplay(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)
        label = tk.Label(self, text="DAQ and Controller", font=LARGE_FONT)
        label.grid(row=0,column=0,pady=10,padx=10,columnspan=3)

        # Filename entry
        fileNameLabel = tk.Label(self, text="Data filename (files saved in DAQdata folder on desktop):",font=NORMAL_FONT)
        fileNameLabel.grid(row=1,column=0,columnspan=3)
        fileNameEntry = tk.Entry(self, width=25)
        fileNameEntry.grid(row=2,column=0,padx=5,columnspan=3)

        # Start and Stop DAQ buttons
        daqStrt = tk.Button(self, text="Start DAQ",height=1,width=10, command=lambda: controller.theInfiniteLoop(buttonHandles,daqStrt,[prmLvTmp,scndLvTmp,lnLvTmp,inTLLvTmp,outTLLvTmp],fileNameEntry))
        daqStrt.grid(row=3,column=0,columnspan=3)
        daqStp = tk.Button(self, text="Stop DAQ",height=1,width=10,state='disabled',command=lambda: controller.stopDAQ())
        daqStp.grid(row=4,column=0,columnspan=3)

        # Display Primary heater settings
        prmTitle = tk.Label(self, text="Primary Heater:", font=NORMAL_FONT)
        prmTitle.grid(row=5,column=0,padx=5,columnspan=3)
        prmStrtTmpLab = tk.Label(self, text="Start Temperature (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        prmStrtTmpLab.grid(row=6,column=0)
        prmStrtTmp = tk.Entry(self)
        prmStrtTmp.grid(row=6,column=1)
        prmStrtTmp.insert(0,params[0])
        prmInitBtn = tk.Button(self, text="Initialize",height=1,width=10,state='disabled',command=lambda: controller.buttonFunc([prmStrtBtn,scndStrtBtn,prmStpBtn],[prmInitBtn],1,[prmStrtTmp],[]))
        prmInitBtn.grid(row=6,column=2)
        prmFnlTmpLab = tk.Label(self, text="Final Temperature (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        prmFnlTmpLab.grid(row=7,column=0)
        prmFnlTmp = tk.Entry(self)
        prmFnlTmp.grid(row=7,column=1)
        prmFnlTmp.insert(0,params[1])
        prmRmpTmLab = tk.Label(self, text="Ramp Time (min):", font=NORMAL_FONT)
        prmRmpTmLab.grid(row=8,column=0)
        prmRmpTm = tk.Entry(self)
        prmRmpTm.grid(row=8,column=1)
        prmRmpTm.insert(0,params[3])
        prmLvTmpLab = tk.Label(self, text="Set Point & Temp (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        prmLvTmpLab.grid(row=9,column=0)
        prmLvTmp = tk.Label(self, text="", font=NORMAL_FONT)
        prmLvTmp.grid(row=9,column=1)
        prmStrtBtn = tk.Button(self, text="Start",height=1,width=10,state='disabled',command=lambda: controller.buttonFunc([prmStpBtn],[prmStrtBtn],2,[prmFnlTmp,prmRmpTm],[]))
        prmStrtBtn.grid(row=7,column=2)
        prmStpBtn = tk.Button(self, text="Stop",height=1,width=10,state='disabled',command=lambda: controller.buttonFunc([prmInitBtn,prmStrtBtn],[prmStpBtn],7,[],[prmStrtTmp,prmFnlTmp,prmRmpTm]))
        prmStpBtn.grid(row=8,column=2)

        # Display Secondary heater settings
        scndTitle = tk.Label(self, text="Secondary Heater:", font=NORMAL_FONT)
        scndTitle.grid(row=10,column=0,padx=5,columnspan=3)
        scndFnlTmpLab = tk.Label(self, text="Final Temperature (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        scndFnlTmpLab.grid(row=11,column=0)
        scndFnlTmp = tk.Entry(self)
        scndFnlTmp.grid(row=11,column=1)
        scndFnlTmp.insert(0,params[1])
        scndRmpTmLab = tk.Label(self, text="Ramp Time (min):", font=NORMAL_FONT)
        scndRmpTmLab.grid(row=12,column=0)
        scndRmpTm = tk.Entry(self)
        scndRmpTm.grid(row=12,column=1)
        scndRmpTm.insert(0,params[3])
        scndLvTmpLab = tk.Label(self, text="Set Point & Temp (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        scndLvTmpLab.grid(row=13,column=0)
        scndLvTmp = tk.Label(self, text="", font=NORMAL_FONT)
        scndLvTmp.grid(row=13,column=1)
        scndStrtBtn = tk.Button(self, text="Start",height=1,width=10, state='disabled',command=lambda: controller.buttonFunc([scndStpBtn],[scndStrtBtn],3,[scndFnlTmp,scndRmpTm],[]))
        scndStrtBtn.grid(row=11,column=2)
        scndStpBtn = tk.Button(self, text="Stop",height=1,width=10,state='disabled',command=lambda: controller.buttonFunc([scndStrtBtn],[scndStpBtn],8,[],[scndFnlTmp,scndRmpTm]))
        scndStpBtn.grid(row=12,column=2)

        # Lead Nuts
        lnTitle = tk.Label(self, text="Transfer Lead Nut Heaters:", font=NORMAL_FONT)
        lnTitle.grid(row=14,column=0,padx=5,columnspan=3)
        lnFnlTmpLab = tk.Label(self, text="Control Temperature (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        lnFnlTmpLab.grid(row=15,column=0)
        lnFnlTmp = tk.Entry(self)
        lnFnlTmp.grid(row=15,column=1)
        lnFnlTmp.insert(0,params[1])
        lnLvTmpLab = tk.Label(self, text="Set Point & Temp (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        lnLvTmpLab.grid(row=16,column=0)
        lnLvTmp = tk.Label(self, text="", font=NORMAL_FONT)
        lnLvTmp.grid(row=16,column=1)
        lnStrtBtn = tk.Button(self, text="Start",height=1,width=10,state=’disabled’,command=lambda: controller.buttonFunc([lnStpBtn],[lnStrtBtn],4,[lnFnlTmp],[]))
        lnStrtBtn.grid(row=15,column=2)
        lnStpBtn = tk.Button(self, text="Stop",height=1,width=10,state=’disabled’,command=lambda: controller.buttonFunc([lnStrtBtn],[lnStpBtn],9,[],[lnFnlTmp]))
        lnStpBtn.grid(row=16,column=2)

        # Inlet Transfer Line
        inTLTitle = tk.Label(self, text="Inlet Transfer Line Heater:", font=NORMAL_FONT)
        inTLTitle.grid(row=17,column=0,padx=5,columnspan=3)
        inTLFnlTmpLab = tk.Label(self, text="Control Te mperature (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        inTLFnlTmpLab.grid(row=18,column=0)
        inTLFnlTmp = tk.Entry(self)
        inTLFnlTmp.grid(row=18,column=1)
        inTLFnlTmp.insert(0,params[1])
        inTLLvTmpLab = tk.Label(self, text="Set Point & Temp (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        inTLLvTmpLab.grid(row=19,column=0)
        inTLLvTmp = tk.Label(self, text="", font=NORMAL_FONT)
        inTLLvTmp.grid(row=19,column=1)
        inTLStrtBtn = tk.Button(self, text="Start",height=1,width=10,state=’disabled’,command=lambda: controller.buttonFunc([inTLStpBtn],[inTLStrtBtn],5,[inTLFnlTmp],[]))
        inTLStrtBtn.grid(row=18,column=2)
        inTLStpBtn = tk.Button(self, text="Stop",height=1,width=10,state=’disabled’,command=lambda: controller.buttonFunc([inTLStrtBtn],[inTLStpBtn],10,[],[inTLFnlTmp]))
        inTLStpBtn.grid(row=19,column=2)

        # Outlet Transfer Line
        outTLTitle = tk.Label(self, text="Outlet Transfer Line Heater:", font=NORMAL_FONT)
        outTLTitle.grid(row=20,column=0,padx=5,columnspan=3)
        outTLFnlTmpLab = tk.Label(self, text="Control Temperature (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        outTLFnlTmpLab.grid(row=21,column=0)
        outTLFnlTmp = tk.Entry(self)
        outTLFnlTmp.grid(row=21,column=1)
        outTLFnlTmp.insert(0,params[1])
        outTLLvTmpLab = tk.Label(self, text="Set Point & Temp (\N{DEGREE SIGN}C):", font=NORMAL_FONT)
        outTLLvTmpLab.grid(row=22,column=0)
        outTLLvTmp = tk.Label(self, text="", font=NORMAL_FONT)
        outTLLvTmp.grid(row=22,column=1)
        outTLStrtBtn = tk.Button(self, text="Start",height=1,width=10, state='disabled',command=lambda: controller.buttonFunc([outTLStpBtn],[outTLStrtBtn],6,[outTLFnlTmp],[]))
        outTLStrtBtn.grid(row=21,column=2)
        outTLStpBtn = tk.Button(self, text="Stop",height=1,width=10,state='disabled',command=lambda: controller.buttonFunc([outTLStrtBtn],[outTLStpBtn],11,[],[outTLFnlTmp]))
        outTLStpBtn.grid(row=22,column=2)

        buttonHandles = np.array([daqStp,prmInitBtn,scndStrtBtn,lnStrtBtn,inTLStrtBtn,outTLStrtBtn])

# Call classes and stuff in "real life"
app = PerkinElmerProjectDAQ()

# Open infinite loop
dataRun = 0
dataArr = []

while True:

    # Update GUI
    app.update_idletasks()
    app.update()
```

APPENDIX B. CONTROLLER CIRCUIT DIAGRAM

As discussed in chapter 2, the control system involved the use of various custom circuit boards. These boards were a fan voltage step down circuit, the thermocouple interfacing circuit board, and the power and relay circuit board. The circuit for the fan voltage was a simple voltage regulator with two capacitors, and the thermocouple board was an array of female connection ports for 16 thermocouple amplifiers. Due the simplicity of these circuits diagrams have not been included below. The circuit diagram for power and relay control was significantly more complex and has thus been included below (Figure B.1).

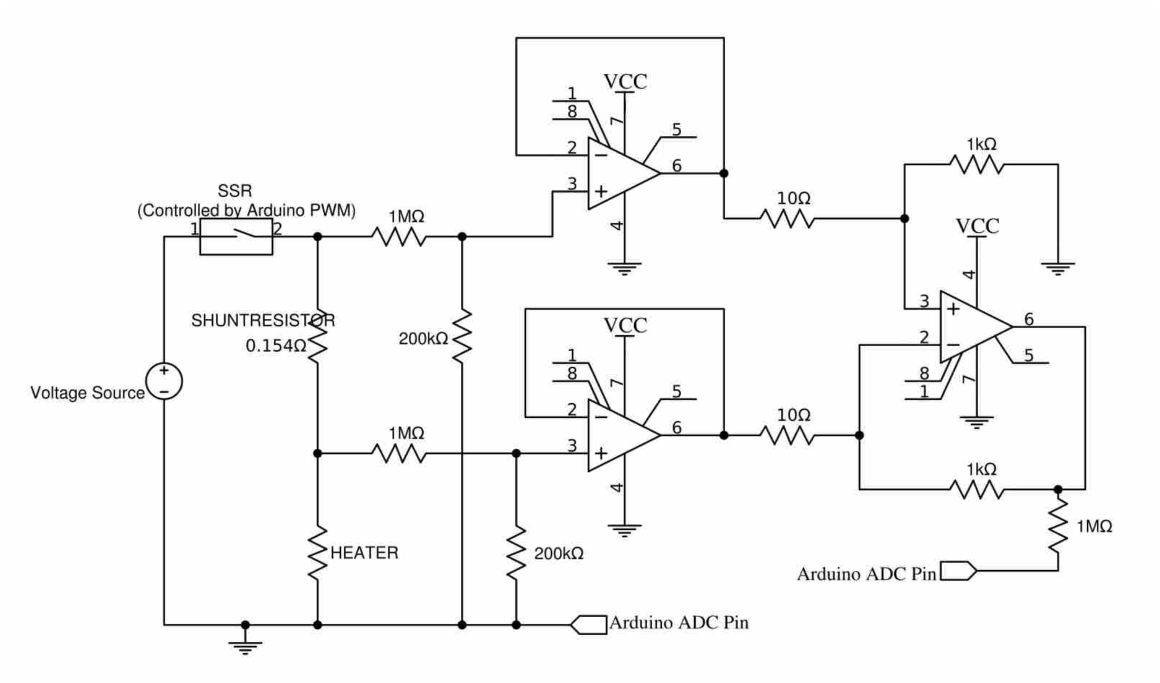


Figure B.1: Diagram of a portion of the circuitry used for the TGGC control system.

APPENDIX C. VOLTAGE AND CURRENT DATA SMOOTHING FUNCTION

One of the difficulties encountered in gathering accurate voltage and current data from the experimental system was the aliasing experienced as a result of the PWM control on the relays that regulated the power supplied to the heaters. At low PWM values the majority of the data the Arduino read in were zero voltage and current values, which was not useful in determining the power consumed by the heater in question. In order to address this problem a multifaceted filtering method was developed to determine which data points were the correct values.

The data communication between the Arduino and Python codes occurred at regular intervals that were about 0.155 seconds apart. In between the sending of each of these data parcels, the Arduino would take numerous readings of voltage and current from each of the heaters. These data readings were filtered to obtain the data values that were representative of the power being consumed by the heater.

The filtering method consisted of a multi-stage data cleaning process. The data communication occurred at regular intervals, and in between these intervals data were collected and filtered to obtain good values for sending to the Arduino. The first step in data filtering was to compute a running average of all the data points that the Arduino read in. This running average value would carry through between data sending events to the Python code. As many of the data points read in by the Arduino were at zero (because the PWM control value was below 100%), the Arduino would take a second running average of only the input data that was above the first overall running average (see Figure C.1 for demonstration of running averages). Doing this served to remove the zero values from the data that would be used. This non-zero data was then used by the Arduino to determine the voltage or current data value for each data send-off event.

Another functionality included in the code was a method for detecting sudden shifts in voltage or current. A bounding threshold function was included that was a decaying inverse function that depended on the time span for the running averages. These threshold values served to further bound the final average that would be computed for the final data that was sent off to the Python code. If at any point the recorded average value for a data send-off event was outside threshold bounds, the Arduino would take that as a sign that the voltage value had shifted and it would reset all the running averages along with the threshold decay. For visual clarification on this filtering method see Figure C.1.

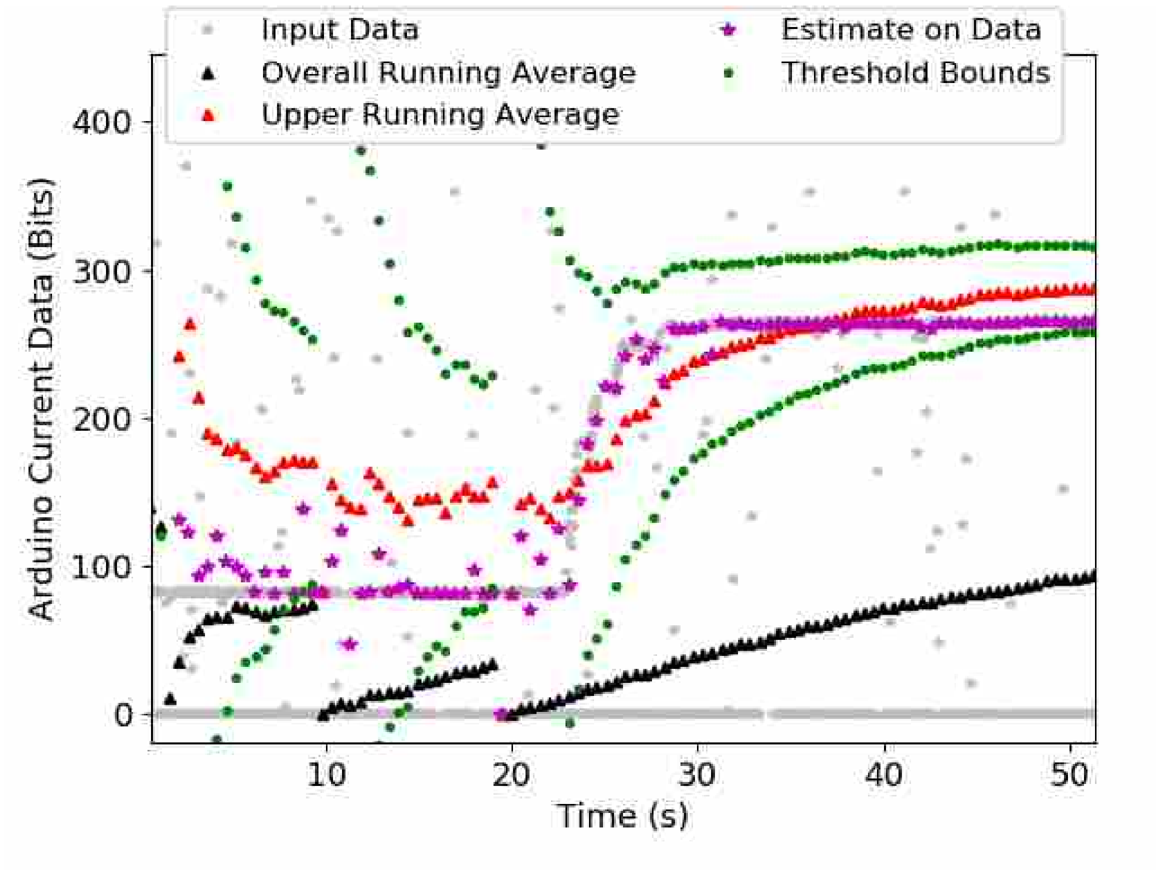


Figure C.1: Demonstration of data smoothing functionality.

APPENDIX D. ADDITIONAL CHROMATOGRAMS

Included below are a selection of additional chromatograms from the various experimental systems described in chapter 2.

D.1 Silicon Column

During the series of experiments run on the silicon column, one of the experiments performed involved the use of both a base heater and secondary heater. The chromatogram shown below in Figure D.1 demonstrates the separation performance of the silicon column with the secondary heater in use. It was found that although the secondary heater was able to increase the column's tip temperature considerably, it actually led to a decrease in the separation performance of the column. It is believed that this was a result of the poor shape of the gradient that resulted

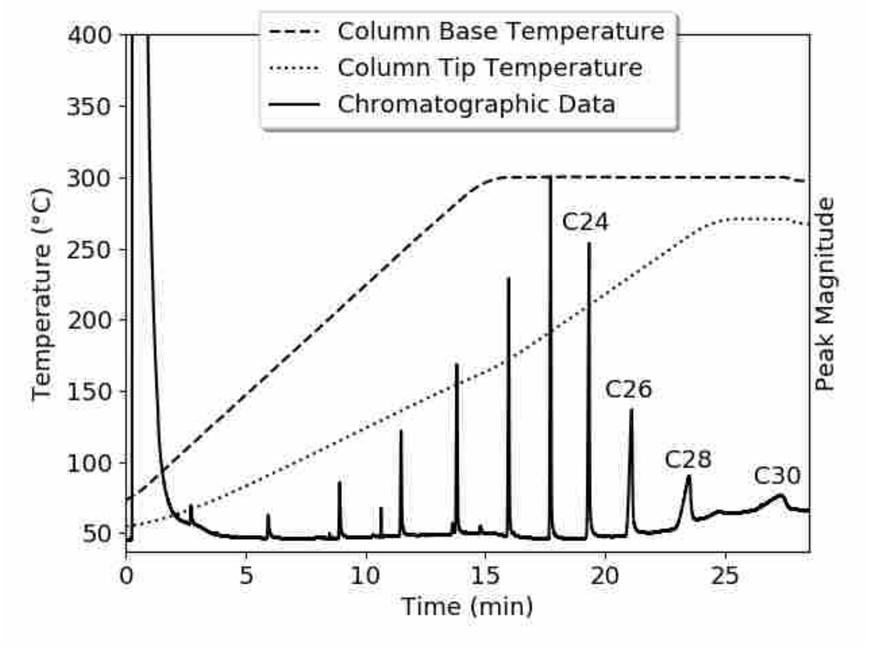


Figure D.1: Separation of C10-C30 on the silicon column using the secondary heater. Column flow set at 1 mL/min, split ratio of 20:1, transfer lines and injector and detector at 250 °C.

from using the secondary heater. As can be seen in Figure 2.3, the secondary heater covered a larger portion of the column, and IR imaging showed that the heat input of the heater was most concentrated at its center. This would seem to indicate that starting the secondary heater would lead to a hump in the gradient near the center of the heater, causing peak broadening.

D.2 Stainless-Steel Column

Before performing TGGC separations on the stainless-steel column, the column was run under TPGC conditions to test the quality of the coating. In one of these tests an ASTM D2887 standard mixture was successfully separated (Figure D.2). This test indicated that the column coating quality was high enough to separate heavy compounds.

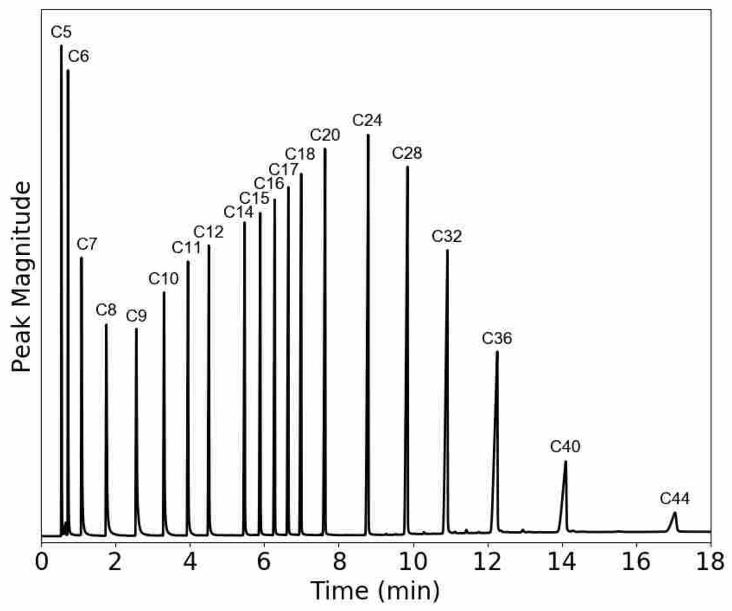


Figure D.2: Separation of ASTM D2887 mixture on the stainless-steel column demonstrating column's ability to separate heavy compounds. TPGC conditions from 40 to 300 °C at approximately 25 °C/min.

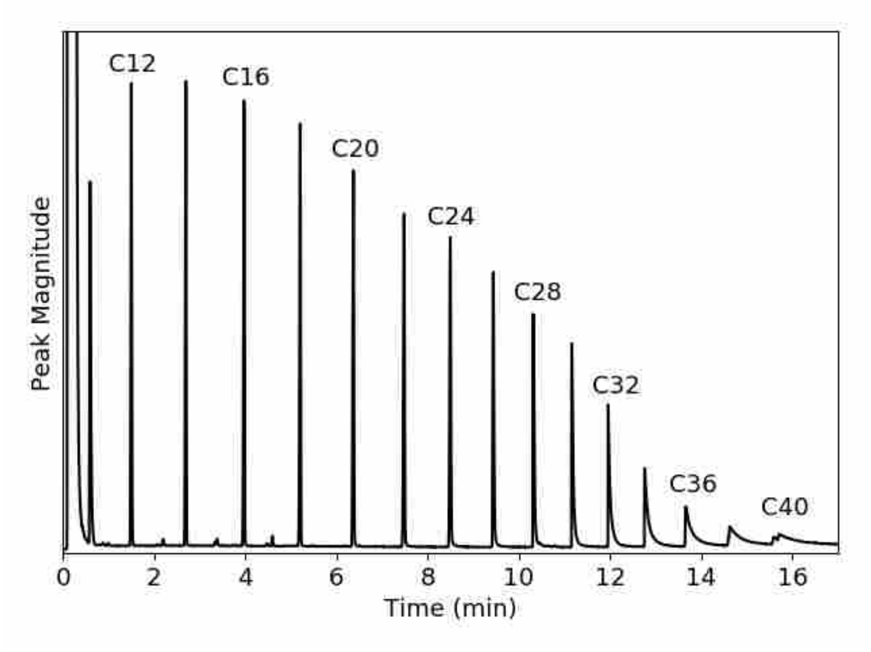


Figure D.3: Separation of C10-C40 mixture on the spiral column without copper sheaths on transfer lines, showing an increase in peak tailing for heavier compounds. Column heated from 70 to 300 °C in approximately 15 minutes.

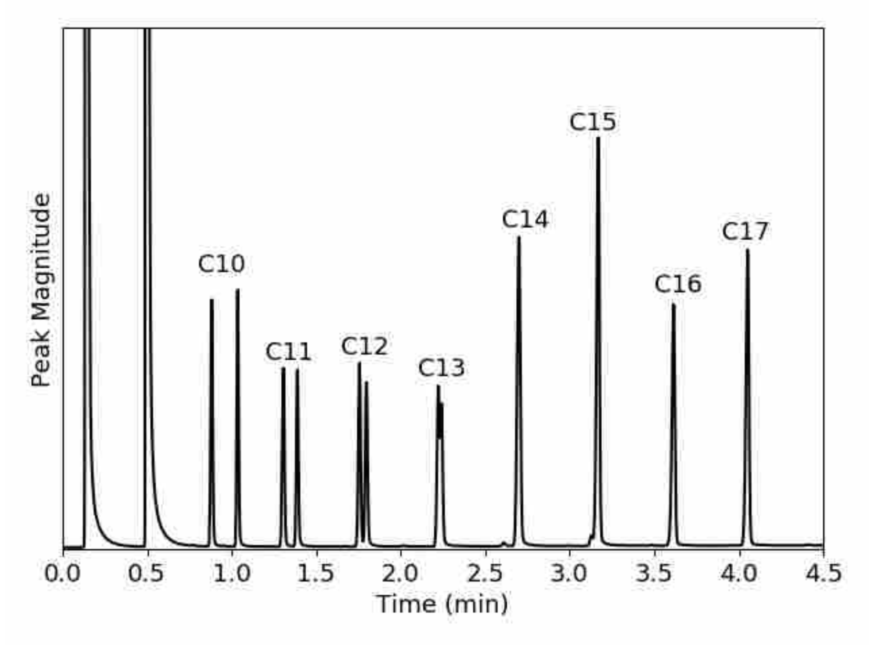


Figure D.4: Separation of double injected C10-C17 mixture on the spiral column demonstrating TGGC's ability to correct for poor injections.

D.3 Spiral Column

As discussed in chapter 2, one of the improvements discovered in working with the spiral column system was that using a copper tube between the heating coil and transfer line helped to improve separation performance by eliminating cold spots. Before testing this improvement, the spiral column was tested without the copper transfer lines. For comparison, an example of the results from this run are provided in Figure D.3. It should be noted that the addition of the copper tubing on the transfer line served to sharpen the heavier peaks that show considerable tailing in the figure below.

Another test that was run on the spiral column was a test to see how the thermal gradient was able to correct for poor injections. To test this a sample was injected into the column twice with an approximately 25 s gap between the injections. The results from this test are shown in Figure D.4. Note that as each subsequent peak elutes the gap between the peaks decreases. By the time C14 elutes the TGGC system has successfully merged the severely split peaks.

APPENDIX E. POWER INPUT VERIFICATION METHODS FOR STAR-CCM+

E.1 Power Circuitry Calibration

The data read in by the Arduino from the power control circuit was converted into bits, where the data scaled between 0 and 1023 bits which correlated to 0 and 5 V respectively. In order to make the data useful in thermal simulation development. Similarly, after consideration of the voltage data read in by the Arduino it was determined that it would be more reliable to use current and resistance data to compute the power consumed by the heater. In order to make these calibrations possible data was taken on the current passing through the heater and the associated Arduino bit readings, along with resistance values across the heater along with temperatures of the heater.

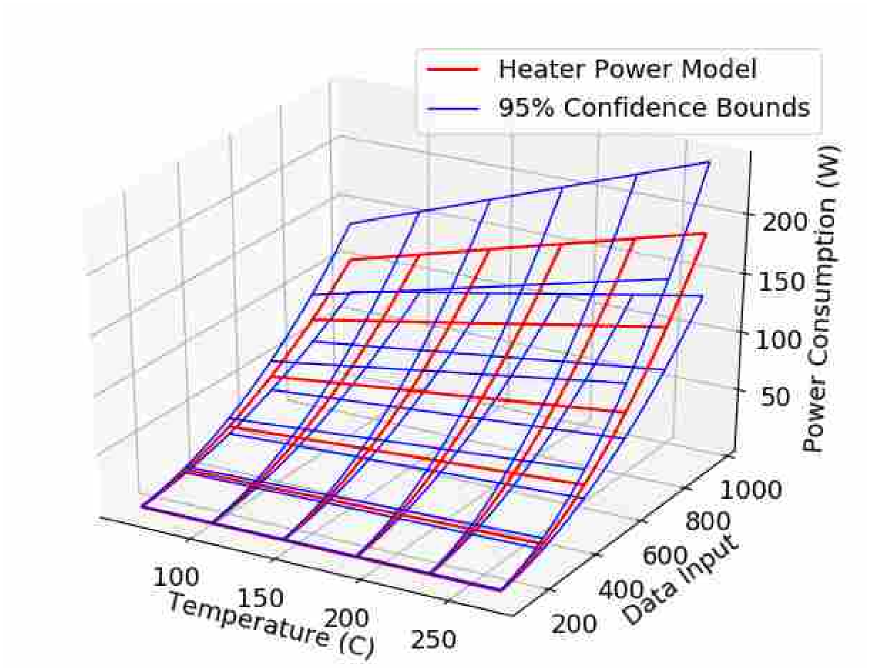


Figure E.1: Power regression curve with upper and lower 95% confidence bounds.

Using the data gathered for calibrating the code, linear regressions to the data were computed for both the current and the resistance of the heater. As power was to be computed via data on current and heater resistance and, given that linear regression models were fit to both the current and resistance data, the final relationship for power was defined by Equation E.1. Results for the linear models and their associated 95% confidence intervals are shown in Figure E.1.

$$p = I^2 R_H = (\beta_3 + \beta_0 Data_I)^2 (\beta_1 + \beta_2 T) \tag{E.1}$$

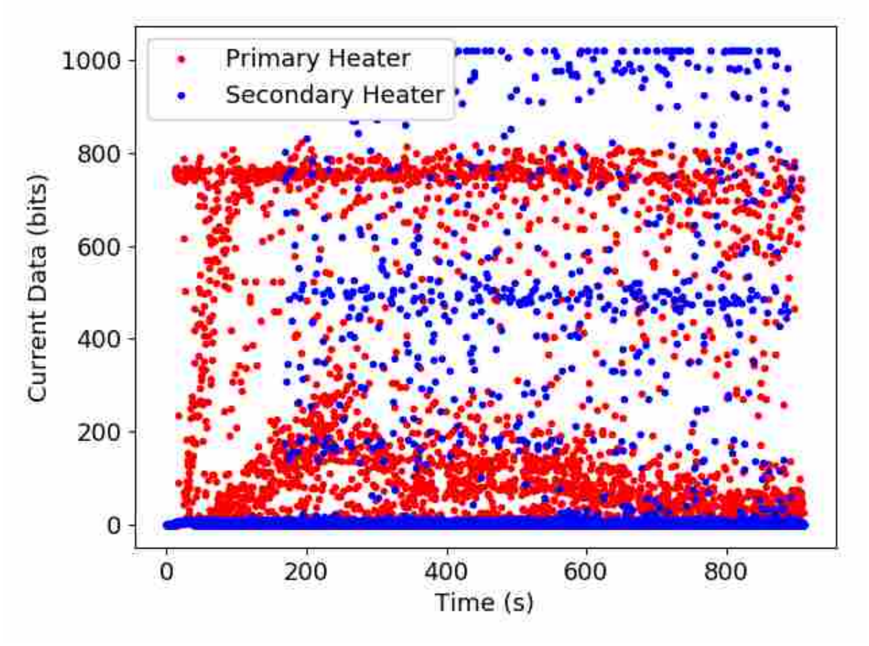


Figure E.2: Example of the noise levels found in the data recorded by the Arduino.

E.2 Power Extraction

The computed value for power could then be used to estimate the power consumed by each heater for any particular experimental run. However before being able to compute the power consumed, the current data read in by the Arduino had to be cleaned up as the Arduino filtering method was unable to remove all the noise in the data (Figure E.2). In order to ensure that only correct values were read in, a method was implemented in a Python script that allowed the user to select where the correct data was. This was done by clicking the approximate mean and upper bound at up to 50 points along the line of interest. The code would then find the 50 nearest data points to the point selected by the user that were within the user identified bound. The mean of these 50 values would then be computed along with a 95% confidence interval on the mean. These data could then be used in the linear model computed earlier to estimate the power consumed by the heater.

E.3 PWM Scaling

Once the heater power consumption was estimated the next step was to determine the percentage of the power that actually entered the heater. In the experimental system this percentage was controlled by the PWM signal and as this value was recorded by the DAQ system it was the value that could be used to properly scale the power that was estimated to be flowing through the heater.

Studies of the behavior of the relay control circuitry using an oscilloscope demonstrated that the although the PWM signal enters the relay as a square wave, the power leaves the relay with a decaying trend as shown in Figure E.3.

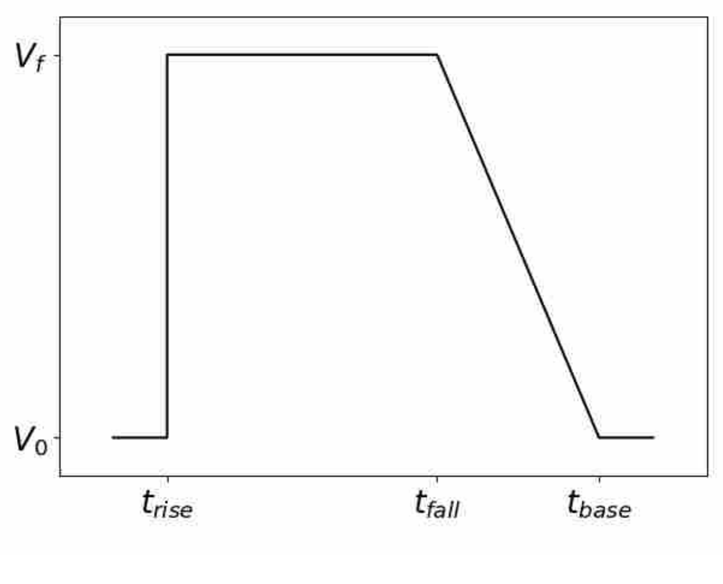


Figure E.3: Voltage behavior to heater exhibited by relays used in circuitry. The PWM signal controlling the relay intended for the heater to be on between t_{rise} and t_{fall}.

The ramp on the back side of the on segments indicated that a simple relation 0% to 100% method for scaling the power wouldn't correctly model the power percentage that was passing through the heater as controlled by the PWM signal. A more accurate representation of the PWM power percentage was computed by integrating the power with respect to time across one full PWM cycle and then dividing the result of that integral by the time of one cycle. The result of this integration was that for PWM values where the ramp didn't intersect with the next step up the power percentage was the following:

$$\%P = \frac{PWM}{255} + 0.042 \tag{E.2}$$

And for PWM values where the ramp did intersect with the next step the power percentage was the following:

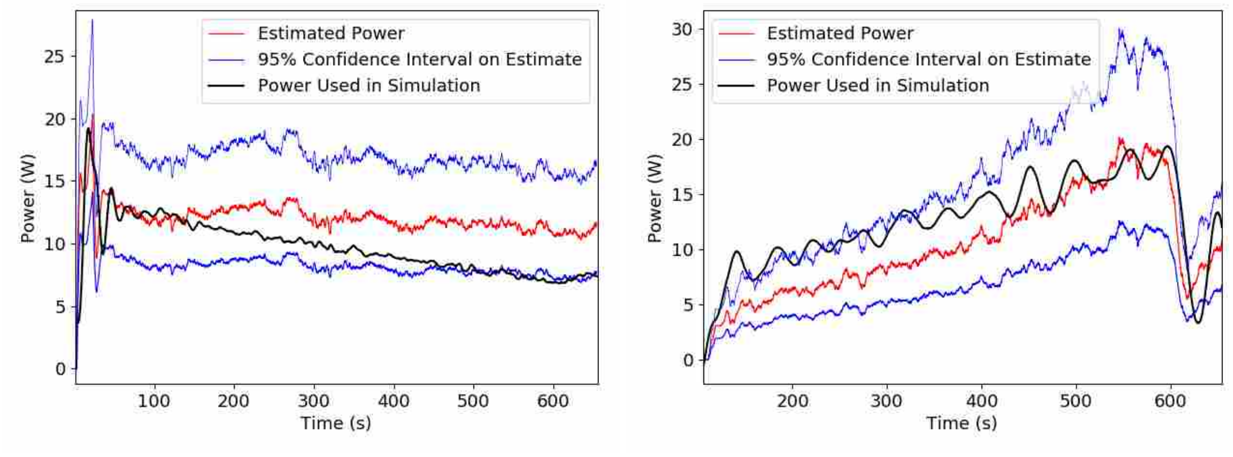
$$\%P = \frac{1}{3}\left(\frac{m_V t_T}{V_f}\right)^2 \left(1 - \frac{PWM}{255}\right)^3 + \left(\frac{m_V t_T}{V_f}\right)\left(\frac{PWM}{255} - 1\right)^2 + 1 \tag{E.3}$$

Where m_V is the slope of the voltage ramp, V_f is the maximum voltage, and t_T is the time that elapses in one PWM cycle. For the above equation m_V = -33.33 V/ms, V_f = 35 V, and t_T = 8.33 ms.

These relations were used to properly scale the power that was computed via the linear regression models discussed above.

E.4 Simulation Verification

As stated in the body of the thesis, it was determined the most accurate and reliable method for controlling the power input for surface heater simulations was the use of a PI controller that was controlled by the error between simulated and experimental temperatures. The PI control method was implemented in STAR-CCM+ by reading the thermocouple data from the experimental runs as tabulated data. This temperature data from the system was then used as the set point temperature and the PI control on the heaters was implemented via a series of reports and field functions. For an example of this control method refer to the simulation in the following location on the CAEDM J-drive: J:\groups\fluxlab\TGGC\Austin\STAR CCM Simulations\46 Steel Chip Tolley Paper\Simulation Runs\03_TransientTest7_PIControl.sim. It was important that the power input results used by the STAR-CCM+ simulation be within the correct bounds so as to ensure that the simulation be reliable. In order to check the validity of the power values used by the PI controller, the controller power input values were recorded and then plotted against the calibrated power values discussed above (Figure E.4). The results of these comparisons showed that for the most part the power computed by the PI controller stayed within the 95% confidence bounds computed by the linear regression fit and current data noise.



(a)

(b)

Figure E.4: Comparison between STAR-CCM+ heating values used in simulations compared to estimated power values for both the (a) primary heater and the (b) secondary heater.

APPENDIX F. SPIRAL COLUMN SYSTEM SIMULATION NOTES

As mentioned in chapter 3, a combined analytical and numerical model was developed to explore the possibility of using a spiral column on a radial substrate instead of a serpentine column on a rectangular substrate. The results from the model showed sufficient promise to motivate moving towards a spiral system.

F.1 Combined Model

Due to the relative simplicity of working with a radial system it was determined that an analytical model should be pursued instead of using STAR-CCM+. Since the thickness of the aluminum disk was thin (0.5 mm) in comparison to the disk's diameter (152.4 mm) it was determined that the thermal behavior perpendicular to the disk could be lumped, thus reducing the heating of the disk to a two-dimensional problem. The governing equation for conduction through the disk was then derived by applying Fourier's law to a differential control annulus with insulation and convective heat losses from the top and bottom leading to the PDE given in Equation F.1.

$$\frac{\partial}{\partial r}\left(k_{Al} r \frac{\partial T}{\partial r}\right) + \frac{r}{z}\left(q''_{in} - 2q''_{cond} - (h_t + h_b)(T - T_\infty)\right) = r\rho c_p \frac{\partial T}{\partial t} \tag{F.1}$$

It was determined that the above equation could not be solved analytically due to the following reasons: (1) the thermal conductivity could not reasonably be treated as constant in the r-direction due to the presence of the column and due to varying temperatures along the disk, (2) the heat flux input was not constant in the r-direction because the printed heater would not necessarily cover the entire disk, (3) the heat loss into the foam was not constant in the r direction because the insulating foam would not necessarily cover the entire disk, and (4) the convective heat loss was not constant in the r-direction because only a portion of the disk would be exposed to air. In order to make obtaining a solution possible the governing equation was discretized using a central difference approach in the r-direction and an explicit approach for time. The numerical solution was then computed using a Python script (Section F.5).

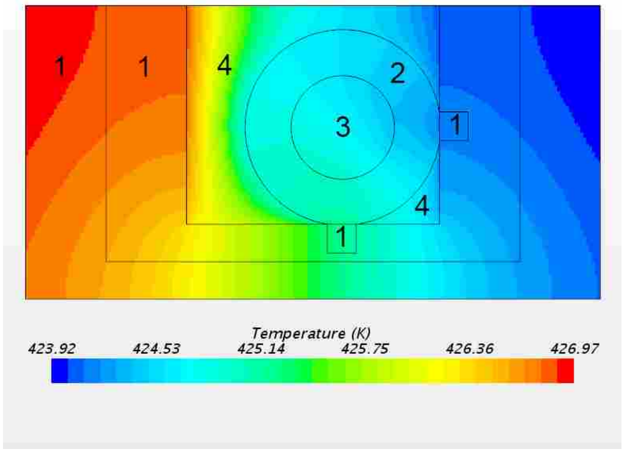
F.2 Effective Thermal Conductivity

One of the difficulties that had to be addressed in order to obtain a meaningful model of the system was determining how the presence of the spiraled column influenced heat conduction through the disk. The design that was under consideration involved a stainless steel capillary column that was placed in a spiral path that had been milled into the disk. In order to accurately model the conduction through this spiraled region of the disk a model had to be developed to accurately represent its heating characteristics. In order to do this a 2D STAR-CCM+ model was prepared of one cross-sectional segment of the milled column region (See Figure F.1a) and was tested under various conditions to determine the temperature dependent effective thermal conductivity of the spiraled segment. The temperature for each test was determined by averaging the temperature across the entire segment, and the effective thermal conductivity was computed by measuring the heat flux and temperature drop across the segment, both of which were then applied to Fourier's law to find k_{eff}. The effective thermal conductivities were then brought together and a second order polynomial fit was made to the data as shown in Figure F.1b.

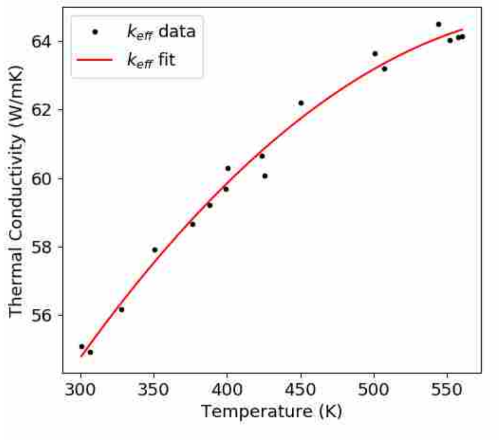
F.3 Foam Heat Loss

As conduction through a two dimensional solid where the temperature is not known on the opposite side is a complex heat transfer phenomenon, a simple model for insulation heat loss could not be used in solving the aluminum disk governing equation. In order to obtain accurate estimates for the insulation heat loss it was determined that the best option would be to compute the temperature gradient in the insulation analytically and then use that gradient to compute the conduction into the foam. Starting with the radial heat diffusion equation, an insulation governing equation of the following form was derived by assuming axisymmetric, steady state conditions with no generation (Equation F.2).

$$2r\frac{\partial^2 T}{\partial r^2}+2\frac{\partial T}{\partial r}+R_S\frac{\partial^2 T}{\partial z^2}=0 \tag{F.2}$$



(a)



(b)

Figure F.1: Thermal model for the effective thermal conductivity of the spiraled column in the radial direction: (a) Cross section of simulated spiral segment with different materials labeled as (1) aluminum plate, (2) stainless-steel capillary column, (3) helium, and (4) air; (b) Data on effective thermal conductivity as a function of temperature with a quadratic fit to the data.

Subject to a fixed temperature boundary at the base (z=0), and a finite boundary at the center (r=0), and convective boundaries at the other two faces. The analytical solution was computed at each time step in the Python script from which the conductive heat loss on the chip was calculated. It should be noted that the above equation in a steady-state solution to the heat diffusion equation in the insulation and thus the heat flux value given represents an approximation of the actual heat that would be lost into the insulation under transient conditions.

F.4 Results

The combined numerical and analytical model for the conduction in the aluminum disk output tabulated temperature data as a function of both radial position and time. These data could then be used in conjunction with a column temperature extraction method to determine the resultant temperature along the length of the column. Results from the model are shown in Figure 3.9 in Section 3.4.

F.5 Combined Model Code

```python
# Title: Transiet Finite Difference Simulation of Radial System
# Author: Austin Foster
# Date: October 10, 2018

# Import modules
import numpy as np
import matplotlib.pyplot as plt
import scipy.special as sp
import scipy.optimize as optimize
from mpl_toolkits.mplot3d import axes3d

# Define thermal conducttivity function
def kAl(T):

    C = 1
    return (87.6050528 + 0.304146932*T - 1.942604246e-4*T**2)*C

def kEff(T):

    return -8.76747e-5*T**2 + 0.1121433*T + 29.01643

# Define parameters
R = 0.0762
##z = 0.001
T_inf = 300
T_i = 70+273.15
N = 30

# Build radial array
delta_r = R/N
r = np.arange(delta_r/2,R,delta_r)

# Build convection array
r_ins = 0.065
```

```
h_tVal = 9.587
h_bVal = 5.171
h_t = np.zeros_like(r)
h_b = np.zeros_like(r)
h_t[r>=r_ins] = h_tVal
h_b[r>=r_ins] = h_bVal

# Define chip thickness
z = 0.00051

# Initalize T
T = np.ones(N)*T_i

# Build thermal conductivity array
r_inner = 0.00578
r_outer = 0.0405
k = kAl(T)
k[(r>=r_inner) & (r<=r_outer)] = kEff(T[(r>=r_inner) & (r<=r_outer)])

# Locate thermocouple control locations
ctrl1 = np.argmin(abs(r-r_inner))
ctrl2 = np.argmin(abs(r-r_outer))

# --- Compute insulation heat loss terms --- #
#Parameters
global r1
r1 = r_ins
h_Ins = 9.8
k_Ins = 0.06
global H
H = h_Ins/k_Ins
global L
L = 0.0245
R_cont = 0.09132
delta_z = 0.001225
gridN = 50
```

```
n = 45
rootEps = 0.01
#Create position arrays
r_Ins = np.linspace(0,r1,gridN)
z_Ins = np.linspace(0,delta_z,2)
r_Ins, z_Ins = np.meshgrid(r_Ins,z_Ins)
# Eigenvalue root funciton
def lambFunc(lamb):
    return -lamb*sp.jn(1,lamb*r1) + H*sp.jn(0,lamb*r1)
# Root finding function
def lambRoot(lambStrt,eps):
    i = 0
    running = 1
    while running:
        lamb = lambStrt + i*eps
        if lambFunc(lamb)*lambFunc(lamb+eps) < 0:
            zero = optimize.bisect(lambFunc,lamb,lamb+eps)
            running = 0
        else:
            i = i + 1
    returnSet = [zero,lamb+eps]
    return returnSet
# Solve for roots
lambdaSet = np.zeros(n)
lambStrt = rootEps
for i in range(n):
    [lambdaSet[i],lambStrt] = lambRoot(lambStrt,rootEps)
# Eigenfunctions
def Rn(n,lamb,r):
    return sp.jn(0,lamb[n]*r)
def Zn(n,lamb,z):
    return -lamb[n]*np.cosh(lamb[n]*(z-L))/H + np.sinh(lamb[n]*(z-L))
# Define function
f = np.interp(r_Ins[0,],r[r<=r_ins],T[r<=r_ins]) - T_inf
# Series constant computer
def c_n(n,f,r,lamb):
    num = np.trapz(r_Ins[0,]*Rn(n,lamb,r_Ins[0,])*f,r_Ins[0,])
    den = (0.5*r1**2*((H/lamb[n])**2+1)*(Rn(n,lamb,r1))**2)*(Zn(n,lamb,0)
    )
    return num/den
# Compute insulation flux
def fluxSolve(r_Ins,lambdaSet,z_Ins,k_Ins,delta_z,r,Tin):
    # Compute edge temperature
    f = np.interp(r_Ins[0,],r[r<=r_ins],Tin[r<=r_ins]) - T_inf
    # Solve for gradient
    Theta = np.zeros_like(r_Ins)
    for i in range(n):
        Theta += c_n(i,f,r_Ins,lambdaSet)*Rn(i,lambdaSet,r_Ins)*\
                 Zn(i,lambdaSet,z_Ins)
    # Compute flux
    q = -k_Ins*(Theta[1,]-Theta[0,])/delta_z
    # SmoothFlux
    q_smooth = np.copy(q)
    window = 15
    for i in range(int(len(q)/2)):
        if i < window:
            q_smooth[i] = np.mean(q[2:i+window])
        else:
            q_smooth[i] = np.mean(q[i-window:i+window])
    q_smooth_Fit = np.interp(r,r_Ins[0,],q_smooth,right=0)
    return q_smooth_Fit

q_cond = fluxSolve(r_Ins,lambdaSet,z_Ins,k_Ins,delta_z,r,T)

# Define time step
delta_t = 0.02
t_max = 100

rMat = np.copy(r)
tMat = np.arange(0,t_max,delta_t)
tArr = np.copy(tMat)
timeN = len(tMat)
```

```
rMat,tMat = np.meshgrid(rMat,tMat)
TMat = np.zeros_like(rMat)

# Build temperature control arrays
strtWt = 30
T_strt = 70 + 273.15
T_fnl = 350+273.15
Tctrl1 = np.zeros_like(tArr)
Tctrl1 = T_strt + tArr*(T_fnl-T_strt)/(t_max-strtWt) - strtWt*(T_fnl-
    T_strt)/(t_max-strtWt)
Tctrl1[tArr<=strtWt] = T_strt

deltaTemp = 30
Tctrl2 = Tctrl1 - deltaTemp

# Set initial temperature
TMat[0,:] = T

# Set values for rho and cp
rho = np.ones(N)*2700.0
rho[(r>=r_inner) & (r<=r_outer)] = 2433.83

def c_pComp(Tarr):

    c_pOut = -6.214e-9*Tarr**4 + 1.62e-5*Tarr**3 - 1.536e-2*Tarr**2 + \
                6.745*Tarr - 5.606
    Tspiral = Tarr[(r>=r_inner) & (r<=r_outer)]
    c_pOut[(r>=r_inner) & (r<=r_outer)] = -3.775e-9*Tspiral**4 + \
                                           1.002e-5*Tspiral**3 - \
                                           0.0097*Tspiral**2 + \
                                           4.516*Tspiral + 94.919
    return c_pOut

c_p = c_pComp(T)

# Build empty q_in array
```

```
q_in = np.zeros_like(r)
r_heater = 0.05

# Run computation loop
# Iteration parameters
err = [1]
eps = 1e-3
errLast = 10000

print('Go!')

for i in range(1,timeN):


    # Compute conduction into insulation
    q_cond = fluxSolve(r_Ins,lambdaSet,z_Ins,k_Ins,delta_z,r,TMat[i-1,:])

    # Compute thermal conductivity
    k = kAl(TMat[i-1,:])
    k[(r>=r_inner) & (r<=r_outer)] = kEff(TMat[i-1,(r>=r_inner) & (r<=
   r_outer)])

    # Compute specific heat
    c_p = c_pComp(TMat[i-1,:])

    # Compute heat inputs
    A = rho[ctrl1]*c_p[ctrl1]*r[ctrl1]*delta_r*z/delta_t
    k_l = np.mean([k[ctrl1],k[ctrl1-1]])
    B = k_l*z*(r[ctrl1]-0.5*delta_r)/delta_r
    C = (h_t[ctrl1]+h_b[ctrl1])*r[ctrl1]*delta_r
    k_r = np.mean([k[ctrl1],k[ctrl1+1]])
    D = k_r*z*(r[ctrl1]+0.5*delta_r)/delta_r
    I = r[ctrl1]*delta_r*(q_cond[ctrl1]+q_cond[ctrl1])
    q_inner = (A*Tctrl1[i] + (B+C+D-A)*TMat[i-1,ctrl1] - D*TMat[i-1,ctrl1
   +1] - \
                B*TMat[i-1,ctrl1-1] - C*T_inf + I)/(r[ctrl1]*delta_r)

    A = rho[ctrl2]*c_p[ctrl2]*r[ctrl2]*delta_r*z/delta_t
    k_l = np.mean([k[ctrl2],k[ctrl2-1]])
    B = k_l*z*(r[ctrl2]-0.5*delta_r)/delta_r
    C = (h_t[ctrl2]+h_b[ctrl2])*r[ctrl2]*delta_r
    k_r = np.mean([k[ctrl2],k[ctrl2+1]])
    D = k_r*z*(r[ctrl2]+0.5*delta_r)/delta_r
    I = r[ctrl2]*delta_r*(q_cond[ctrl2]+q_cond[ctrl2])
    q_outer = (A*Tctrl2[i] + (B+C+D-A)*TMat[i-1,ctrl2] - D*TMat[i-1,ctrl2
+1] - \
               B*TMat[i-1,ctrl2-1] - C*T_inf + I)/(r[ctrl2]*delta_r)
    if q_outer<0:
        q_outer = 0

    q_in = np.ones_like(r)*q_inner
    q_in[r>=0.5*r_heater] = q_outer
    q_in[r>r_heater] = 0

    for j in range(N):

        # Check for edges
        if (j==0):

            # Compute constants
            A = rho[j]*c_p[j]*r[j]*delta_r*z/delta_t
            B = 0
            C = (h_t[j]+h_b[j])*r[j]*delta_r
            k_r = np.mean([k[j],k[j+1]])
            D = k_r*z*(r[j]+0.5*delta_r)/delta_r
            S = q_in[j]*r[j]*delta_r
            I = r[j]*delta_r*(q_cond[j]+q_cond[j])
            # Compute next point
            TMat[i,j] = ((A-B-C-D)*TMat[i-1,j] + B*0 + \
                          D*TMat[i-1,j+1] + C*T_inf + S - I)/A

        elif (j==N-1):

            # Compute constants
            A = rho[j]*c_p[j]*r[j]*delta_r*z/delta_t
            k_l = np.mean([k[j],k[j-1]])
            B = k_l*z*(r[j]-0.5*delta_r)/delta_r
            C = (h_t[j]+h_b[j])*r[j]*delta_r
            D = 0
            S = q_in[j]*r[j]*delta_r
            I = r[j]*delta_r*(q_cond[j]+q_cond[j])
            # Compute next point
            TMat[i,j] = ((A-B-C-D)*TMat[i-1,j] + B*TMat[i-1,j-1] + \
                         D*0 + C*T_inf + S - I)/A

        else:

            # Compute constants
            A = rho[j]*c_p[j]*r[j]*delta_r*z/delta_t
            k_l = np.mean([k[j],k[j-1]])
            B = k_l*z*(r[j]-0.5*delta_r)/delta_r
            C = (h_t[j]+h_b[j])*r[j]*delta_r
            k_r = np.mean([k[j],k[j+1]])
            D = k_r*z*(r[j]+0.5*delta_r)/delta_r
            S = q_in[j]*r[j]*delta_r
            I = r[j]*delta_r*(q_cond[j]+q_cond[j])
            # Compute next point
            TMat[i,j] = ((A-B-C-D)*TMat[i-1,j] + B*TMat[i-1,j-1] + \
                         D*TMat[i-1,j+1] + C*T_inf + S - I)/A

# Save results
TMatScale = TMat - 273.15
np.savetxt('rMat.csv',rMat,delimiter=',')
np.savetxt('tMat.csv',tMat,delimiter=',')
np.savetxt('TMatScale.csv',TMatScale,delimiter=',')

# Plot results
fig = plt.figure()
```

```
ax = fig.add_subplot(111,projection='3d')
ax.plot_wireframe(rMat,tMat/60.0,TMatScale)
ax.set_xlabel('Radial Position (m)')
ax.set_ylabel('Time (min)')
ax.set_zlabel(r'Temperature ($\degree$C)')
ax.set_title('Radial Gradient')
plt.show()

# Compute column gradient
rCol = rMat[:,(r>=r_inner) & (r<=r_outer)]
tCol = tMat[:,(r>=r_inner) & (r<=r_outer)]
TCol = TMat[:,(r>=r_inner) & (r<=r_outer)] - 273.15
K_spiral = 0.001/(2*np.pi)
lCol = rCol**2/K_spiral - r_inner**2/K_spiral
fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
ax.plot_wireframe(lCol,tCol/60.0,TCol)
ax.set_title('Column Gradient')
ax.set_xlabel('Column Position (m)')
ax.set_ylabel('Time (min)')
ax.set_zlabel(r'Temperature ($\degree$C)')
plt.show()

# Compute delta T
deltaT = np.zeros(timeN)
for i in range(timeN):

    T_hi = np.interp(r_inner, r, TMat[i,:])
    T_Lo = np.interp(r_outer, r, TMat[i,:])
    deltaT[i] = T_hi - T_Lo

plt.plot(np.arange(0,t_max,delta_t)/60.0,deltaT)
plt.xlabel('Time (min)')
plt.ylabel(r'$\Delta$T ($\degree$C)')
plt.title(r'Gradient $\Delta$T')
plt.show()
```

APPENDIX G. SOLIMIDE PROPERTY EXPERIMENT

As described briefly in Section 3.3, experiments were performed to determine the thermal conductivity of Solimide as well as the contact resistance between Solimide foam and a flat surface. After preliminary efforts to calibrate the simulation of heating in the silicon chip it was found that heat loss into the Solimide foam had a significant influence on the shape of the gradient. It was therefore determined that it would be necessary to gather experimental data to better quantify the thermal properties.

An experiment was devised to imitate the conditions of the Solimide foam in the experimental system. A 0.25 inch thick plate of copper was connected on its bottom face to a mica surface heater, and on its top face to a 3 inch square heat flux sensor. A one inch thick section of Solimide was then laid on top of the heat flux sensor and was weighed down by another piece of copper in the same way as was done in the experimental setup (Figure G.1).

Six thermocouples were used to gather data on the system. The first was taped to the top of the copper plate adjacent to the heat flux sensor and was used in controlling the temperature of the copper block. Another thermocouple taped to the top surface of the heat flux sensor, and three thermocouples were inserted into the foam at varying distances from the heat flux sensor. A final thermocouple was placed on the top face of the Solimide foam (Figure G.2). Note that thermocouples two through six were all placed near the center of the Solimide piece so as to minimize the effects of heat loss at the edges of the insulation.

The copper was heated to varying temperatures to match the temperatures experienced by the insulation in TGGC runs (50-100 °C) and data was gathered on the heat flux passing through the heat flux sensor as well as the temperatures of each of the thermocouples. In order to compute the thermal conductivity of the foam the temperatures of thermocouples 3 through 6, along with the distances between each thermocouple and the heat flux in the heat flux sensor were applied

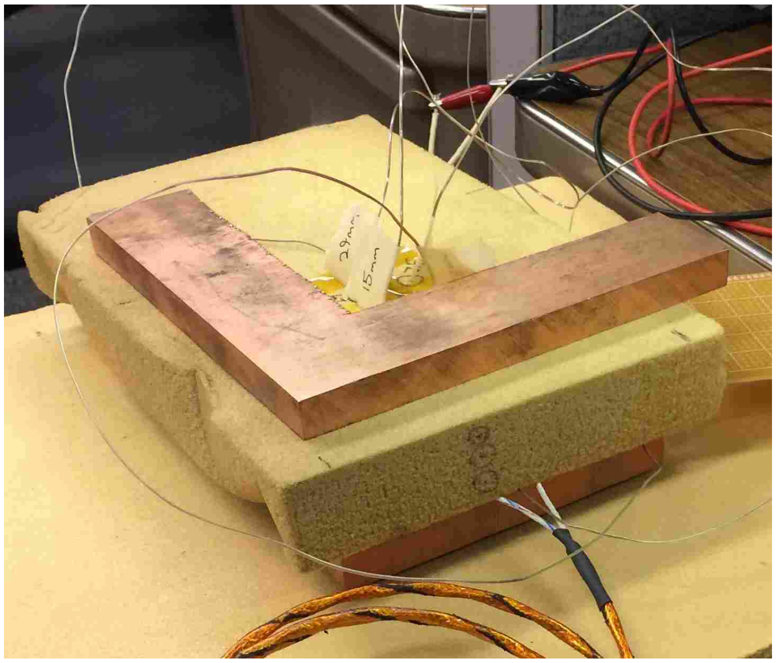


Figure G.1: Experimental setup used to gather data needed to determine the thermal conductivity and contact resistance for Solimide.

to Fourier's law of conduction. Assuming that the conduction was one-dimensional the thermal conductivity could be computed as follows.

$$k_S = -\frac{q''_S \Delta y}{\Delta T} \tag{G.1}$$

In order to compute the contact resistance between the heat flux sensor the slope between thermocouples 3 and 4 was extrapolated back to the surface of the heat flux sensor. The difference between this extrapolated temperature and the temperature from thermocouple 2 were used together with the heat flux to compute the thermal contact resistance as shown in Equation G.2 [45]. It was then assumed for the purpose of the simulations run in STAR-CCM+ that this contact resistance would be approximately equal to the contact resistance for the experimental systems being modeled. The resultant thermal conductivity and thermal contact resistance values are shown in Figure G.3.

$$R''_{t,c,S} = \frac{T_2 - T_{ext}}{q''_S} \tag{G.2}$$

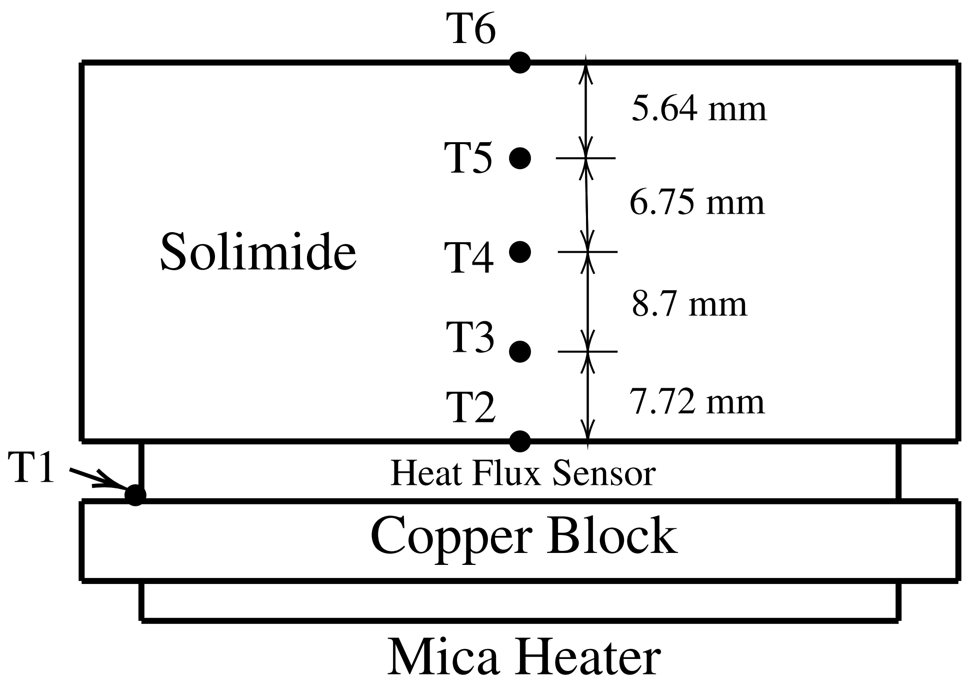
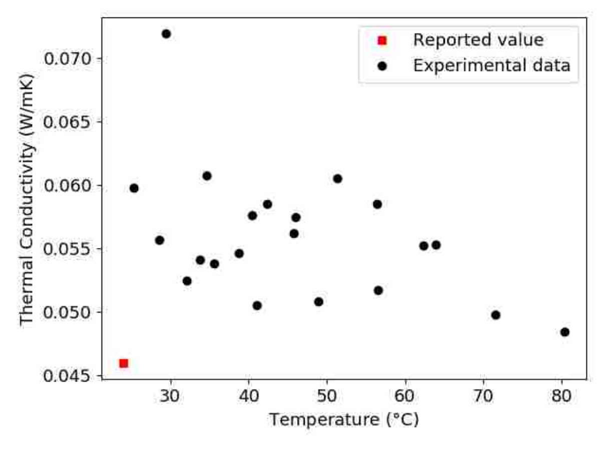
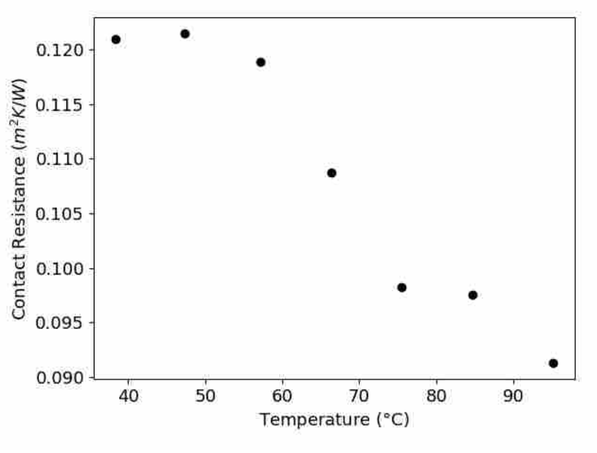


Figure G.2: Schematic of experimental setup used to determine the thermal conductivity and contact resistance of Solimide.



(a)



(b)

Figure G.3: Computational results for both (a) thermal conductivity and (b) contact resistance for Solimide derived from data gathered from the experimental system.

APPENDIX H.   GRADIENT EXTRACTION CODE

Upon successful validation of a thermal model, the next step was to extract the thermal gradient along the column in the substrate of interest. In order to do this the path of the column along the substrate had to be converted into a form that would be useful in extracting associated temperatures. To this end a Python script was developed that created a table of the points along the column path by consulting the design drawings for the column of interest. The code for this extraction on the stainless-steel column is included below.

Along with having data on the column's path, it was necessary to have tabulated data on the column's temperature. To obtain this data the validated STAR-CCM+ simulation was run and temperature data was extracted at every node in the simulated column substrate at regular time steps (usually about 1 second). This data was then used in conjunction with the column path data in a Python code that computes a 3D regression fit of the temperature at any point as a function of the temperatures of the 10 nearest nodes in the simulation. The extracted temperatures were then smoothed using a moving window averaging technique. The codes used for this simulation to column path temperature extraction are included below.

H.1   Column Path Extraction Code

```python
# Title: Python Serpentine Path Extractor 3.0
# Author: Austin Foster
# Date: February 8, 2018

# Import modules
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

```

# ---- Define functions ---- #

# Point finder along linear segment
def lineData(Start, slack, angle, length, gapSmall, gapLarge, dataOrg,
    ind):

    # Compute segment endpoint
    End = [Start[0]+length*np.cos(angle),Start[1]+length*np.sin(angle)]

    # Check for slack overshoot (no data points)
    if (slack > length):
        slack_out = slack - length

    # If data is to be gathered, gather it
    else:

        # Compute the first point
        #Point = [Start[0]+slack*np.cos(angle),Start[1]+slack*np.sin(
    angle)]
        Point = [Start[0],Start[1]]
        if len(dataOrg) == 0:
            dataOrg = [ind,0,Point[0],Point[1]]
        else:
            dataOrg = np.vstack((dataOrg,[ind,gapSmall-slack,Point[0],
    Point[1]]))
        ind += 1

        # Compute running length
        #L_r = slack
        L_r = 0

        # Enter iterative loop
        while (L_r + gapLarge < length):

            # Compute data point
            Point = [Point[0]+gapLarge*np.cos(angle),Point[1]+gapLarge*np.sin(angle)]
            dataOrg = np.vstack((dataOrg,[ind,gapLarge,Point[0],Point[1]]))
            ind += 1

            # Compute running length
            L_r += gapLarge

        # Compute remaining slack
        if L_r < length:
            L_r += gapLarge
        slack_out = L_r - length

    # Return results
    return dataOrg, End, slack_out, ind

# ---------------------------------------------------

# Point finder around a corner
def cornerData(Start, slack, angle1, angle2, radius, gapSmall, gapLarge, dataOrg, ind):

    # Compute angle rounded by corner (sign sensitive)
    phi = angle2 - angle1

    # Compute directional value
    direc = phi/abs(phi)

    # Compute path end point
    psi = 0.5*(np.pi-abs(phi))
    gamma = angle1 + direc*(0.5*np.pi-psi)
    c = 2.*radius*np.sin(0.5*abs(phi))
    End = [Start[0]+c*np.cos(gamma),Start[1]+c*np.sin(gamma)]

    # Compute arc length
    L = abs(phi*radius)

    # Compute first point
    phi_s = slack/radius
    psi_s = 0.5*(np.pi-phi_s)
    gamma_s = angle1 + direc*(0.5*np.pi - psi_s)
    c_s = 2.*radius*np.sin(0.5*phi_s)
    #Point = [Start[0]+c_s*np.cos(gamma_s),Start[1]+c_s*np.sin(gamma_s)]
    Point = [Start[0],Start[1]]
    dataOrg = np.vstack((dataOrg,[ind,gapLarge-slack,Point[0],Point[1]]))
    ind += 1

    # Comptute new position angle
    theta = angle1 #+ direc*phi_s

    # Compute running length
    #L_r = slack
    L_r = 0

    # Enter iterative loop
    while (L_r + gapSmall < L):

        # Compute data point
        phi_k = gapSmall/radius
        psi_k = 0.5*(np.pi-phi_k)
        gamma_k = theta + direc*(0.5*np.pi - psi_k)
        c_k = 2.*radius*np.sin(0.5*phi_k)
        Point = [Point[0]+c_k*np.cos(gamma_k),Point[1]+c_k*np.sin(gamma_k
)]
        dataOrg = np.vstack((dataOrg,[ind,gapSmall,Point[0],Point[1]]))
        ind += 1

        # Comptute new position angle
        theta += direc*phi_k

        # Compute running length
        L_r += gapSmall

    # Compute remaining slack
    if L_r < L:
        L_r += gapSmall
    slack_out = L_r - L

    # Return results
    return dataOrg, End, slack_out, ind

# ----------------------------------------------------

# Parameters
dataOrg = []

data = np.genfromtxt('NoTipGradient.csv',delimiter=',',skip_header=1)
eps = 0.0001
TData = data[:,0]
xData = data[:,1] - np.min(data[:,1])
yData = -1*data[:,3]

StartDistZ = 0.000647475
InHoleGap = 0.0254
NumChan = 86
L1 = 0.0015
ang1 = (-90)*np.pi/180 + 0.5*np.pi
R1 = 0.00031
L2 = 0.01381
ang2 = (-0)*np.pi/180 + 0.5*np.pi
R2 = 0.00031
Lchan = 0.05362306748
angChan1 = (-179.76)*np.pi/180 + 0.5*np.pi
angChan2 = (-0.24)*np.pi/180 + 0.5*np.pi
Rchan = 0.00031
R3 = 0.00031
L3 = 0.05482177947
```

```python
ang3 = (-179.76)*np.pi/180 + 0.5*np.pi
R4 = 0.00031
L4 = 0.053195 + 0.0924983976440889
ang4 = (-270)*np.pi/180 + 0.5*np.pi
R5 = 0.00031
L5 = 0.01499
ang5 = (-360)*np.pi/180 + 0.5*np.pi
R6 = 0.00031
L6 = 0.00148
ang6 = (-270)*np.pi/180 + 0.5*np.pi

#dataOrg = np.zeros((1,4))
slack = 0.0
gapSmall = 0.00005
gapLarge = 0.001
ind = 0

# Intermediate computations
Start = [InHoleGap/2, StartDistZ]

# Horizontal channel 1
dataOrg, Start, slack, ind = lineData(Start, slack, ang1, L1, gapSmall,
    gapLarge, dataOrg, ind)

# Curved corner 1
dataOrg, Start, slack, ind = cornerData(Start, slack, ang1, ang2, R1,
    gapSmall, gapLarge, dataOrg, ind)

# Angle channel 1
dataOrg, Start, slack, ind = lineData(Start, slack, ang2, L2, gapSmall,
    gapLarge, dataOrg, ind)

# Curved corner 2
dataOrg, Start, slack, ind = cornerData(Start, slack, ang2, angChan1, R2,
     gapSmall, gapLarge, dataOrg, ind)

```

```
# Line segments 2
for i in range(NumChan):

    print(100.*i/NumChan)

    # Length 1
    dataOrg, Start, slack, ind = lineData(Start, slack, angChan1, Lchan,
   gapSmall, gapLarge, dataOrg, ind)
    # Round 1
    dataOrg, Start, slack, ind = cornerData(Start, slack, angChan1,
   angChan2, Rchan, gapSmall, gapLarge, dataOrg, ind)
    # Length 2
    dataOrg, Start, slack, ind = lineData(Start, slack, angChan2, Lchan,
   gapSmall, gapLarge, dataOrg, ind)
    # Round 2
    if i < NumChan-1:
        dataOrg, Start, slack, ind = cornerData(Start, slack, angChan2,
   angChan1, Rchan, gapSmall, gapLarge, dataOrg, ind)

# Curved corner 3
dataOrg, Start, slack, ind = cornerData(Start, slack, angChan2, ang3, R3,
    gapSmall, gapLarge, dataOrg, ind)

# Angle Channel 3
dataOrg, Start, slack, ind = lineData(Start, slack, ang3, L3, gapSmall,
   gapLarge, dataOrg, ind)

# Curved corner 4
dataOrg, Start, slack, ind = cornerData(Start, slack, ang3, ang4, R4,
   gapSmall, gapLarge, dataOrg, ind)

# Angle Channel 4
dataOrg, Start, slack, ind = lineData(Start, slack, ang4, L4, gapSmall,
   gapLarge, dataOrg, ind)

# Curved corner 5
```

```
dataOrg, Start, slack, ind = cornerData(Start, slack, ang4, ang5, R5,
    gapSmall, gapLarge, dataOrg, ind)

# Angle Channel 5
dataOrg, Start, slack, ind = lineData(Start, slack, ang5, L5, gapSmall,
    gapLarge, dataOrg, ind)

# Curved corner 6
dataOrg, Start, slack, ind = cornerData(Start, slack, ang5, ang6, R6,
    gapSmall, gapLarge, dataOrg, ind)

# Angle Channel 5
dataOrg, Start, slack, ind = lineData(Start, slack, ang6, L6, gapSmall,
    gapLarge, dataOrg, ind)

# Center data along the x axis (in the y direction)
centerY = np.mean(dataOrg[:,3])
dataOrg[:,3] = dataOrg[:,3]-centerY

print(sum(dataOrg[:,1]))
print(dataOrg[-1,2]-dataOrg[0,2])

# Save serpentine path data
outFilename = 'VariableStepPath.csv'
np.savetxt(outFilename,dataOrg,delimiter=',')

# Plot results
plt.plot(dataOrg[:,2],dataOrg[:,3],'k.')
plt.axis('equal')
plt.show()
```

H.2 Temperature Fit Header File

```
# Title: Data Analysis TGGC
# Author: Austin Foster
# Date: February 12, 2018
```

```python

# ---- Import modules ---- #
import tkinter as tk
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# ---- Define Functions ---- #

# Filepath dialog ------------------------------
def filePaths():

    # Get first two simulation data filepaths
    root = tk.Tk()
    root.withdraw()
    file_path1 = tk.filedialog.askopenfilename(title = "Select first
    simulation data file")
    file_path2 = tk.filedialog.askopenfilename(title = "Select second
    simulation data file")

    # return data
    return file_path1, file_path2

# Filename time data extractor ------------------------------
def timeDataExtract(file_path1,file_path2):
    path1Parsed = list(file_path1)
    path2Parsed = list(file_path2)
    pathEnd = 0
    fileLeadEnd = 0
    for i in range (len(path1Parsed)):
        if(path1Parsed[i]=='/'):
            pathEnd = i
        if(path1Parsed[i]=='_'):
            fileLeadEnd = i
    fileLead = ''.join(path1Parsed[0:fileLeadEnd+1])
    time1 = float(''.join(path1Parsed[fileLeadEnd+1:fileLeadEnd+9]))
    *10.0**float(''.join(path1Parsed[fileLeadEnd+10:fileLeadEnd+13]))
    time2 = float(''.join(path2Parsed[fileLeadEnd+1:fileLeadEnd+9]))
    *10.0**float(''.join(path2Parsed[fileLeadEnd+10:fileLeadEnd+13]))
    deltaT = time2-time1
    return time1,time2,deltaT,fileLead

# Filename creation function ------------------------------
def fileNameCreator(fileLead,deltaT,i,time1):

    roundDigit = 2
    roundNum = round(float(i)*deltaT + time1,roundDigit)

    digits = [d for d in str(roundNum)]
    if(len(digits) > 8):
        digits = digits[0:7]
    pointPos = 0
    digitStart = 0
    digitEnd = 0
    for k in range(len(digits)):
        if(digits[k]=='.'):
            pointPos = k
    multiplier = [d for d in str(pointPos-1)]
    if(pointPos == 1 and digits[0] == '0'):
        multSign = '-'
        l = pointPos
        done2 = 0
        while (done2 == 0):
            l = l+1
            if(digits[l] != '0'):
                digitStart = l
                done2 = 1
                multiplierStr = '0' + str(l-pointPos)
        l = digitStart
        done2 = 0
        while (done2 == 0):
            l = l+1
            if(l == len(digits)):
                digitEnd = l-1
                done2 = 1
            elif(digits[l] == '0'):
                digitEnd = l
                done2 = 1
    else:
        multSign = '+'
        multiplierStr = '0'+''.join(multiplier)
    if(pointPos == 1 and multSign == '+'):
        number = ''.join(digits)
        for k in range(8-len(digits)):
            number = number + '0'
    elif(pointPos == 1 and multSign == '-'):
        number = ''.join(digits[digitStart]) + '.' + ''.join(digits[
   digitStart+1:digitEnd+1])
        numberDigits = [d for d in number]
        for k in range(8-len(numberDigits)):
            number = number + '0'
    else:
        number = ''.join(digits[0]) + '.' + ''.join(digits[1:pointPos]) +
    ''.join(digits[pointPos+1:len(digits)])
        for k in range(8-len(digits)):
            number = number + '0'

    fileTime = number + 'e' + multSign + multiplierStr
    fileName = fileLead + fileTime + '.csv'
    return fileName

# Define data fit and extraction function ------------------------------
def dataExtract(Point, xData, yData, zData, propData):

    #----Expanatory Notes------------------------------#
    # xData, zData, and propData have to be 1D arrays
    # Point is a 1D array with two elements
    #--------------------------------------------------#

    # Trim search region
    deltaBox = 0.002
    maskX = (xData<Point[0]+deltaBox) & (xData>Point[0]-deltaBox)
    maskZ = (zData<Point[1]+deltaBox) & (zData>Point[1]-deltaBox)
    maskY = (yData<Point[2]+0.0007) & (yData>Point[2]-0.0007)
    xDatTrim = xData[maskX & maskY & maskZ]
    yDatTrim = yData[maskX & maskY & maskZ]
    zDatTrim = zData[maskX & maskY & maskZ]
    propDatTrim = propData[maskX & maskY & maskZ]

    # Find the closest points (3D)
    dist = np.sqrt((Point[0]-xDatTrim)**2. + (Point[1]-zDatTrim)**2. + (
   Point[2]-yDatTrim)**2.)
    k = 50       # This defines the number of points used for the data
   fit
    indDist = np.argpartition(dist,k)[:k]

    # Build interpolation solution
    X = np.zeros((k,4))
    X[:,0] = 1.
    y = np.zeros((k,1))
    for j in range(k):
        X[j,1] = xDatTrim[indDist[j]]
        X[j,2] = yDatTrim[indDist[j]]
        X[j,3] = zDatTrim[indDist[j]]
        y[j,0] = propDatTrim[indDist[j]]
    x,resid,rnk,s = np.linalg.lstsq(X,y,rcond=None)
    TOut = x[0] + x[1]*Point[0] + x[2]*Point[2] + x[3]*Point[1]

    return TOut

# Path data read in function ------------------------------
def pathDataReadIn(filename):

    pathData = np.genfromtxt(filename,delimiter=',')

    # Shift path data
    edgeDist = 0.00635
    pathData[:,2] = pathData[:,2] - np.min(pathData[:,2]) + edgeDist

    # Compute step distance
    step = np.sqrt((pathData[2,1]-pathData[1,1])**2.
                   + (pathData[2,2]-pathData[1,2])**2.)

    return pathData, step

# Simulation data read in function ------------------------------
def simDataReadIn(filename):

    simData = np.genfromtxt(filename,
                        delimiter=',',skip_header=1)

    # Reshape data
    simX = np.concatenate((simData[:,2],simData[:,2]),axis=0)
    simX = simX - np.min(simX)
    simY = np.concatenate((simData[:,3],simData[:,3]),axis=0)
    simZ = np.concatenate((simData[:,4],-1*simData[:,4]),axis=0)
    simTemp = np.concatenate((simData[:,1],simData[:,1]),axis=0)

    return simX,simY,simZ,simTemp

# Define normal distribution function ------------------------------
def normDist(x,mu,sigma):
    return (1/(np.sqrt(2*np.pi)*sigma))*np.exp(-(x-mu)**2/(2*sigma**2))

# Define smoothing function -----------------------------------------
def colGradSmooth(X,T,N):
    # Create empty vector
    Ts = np.zeros_like(T)
    # Open interative loop
    for i in range(len(T)):
        # Compute surrounding delta_x
        if i==0:
            delta_x = X[1] - X[0]
        elif i == len(T)-1:
            delta_x = X[i] - X[i-1]
        else:
            delta_x = 0.5*(X[i+1]-X[i-1])
        # Compute standard deviation
        sigma = 0.741301065*(N-1)*delta_x
        # Extract points of interest
        Xdist = X[(X>=X[i]-4*sigma)&(X<=X[i]+4*sigma)]
        Tdist = T[(X>=X[i]-4*sigma)&(X<=X[i]+4*sigma)]
        # Create midpoint X array
        Xmid = 0.5*(Xdist[0:len(Xdist)-1]+Xdist[1:len(Xdist)])
        if (Xmid[0] != X[i]-4*sigma) & (X[i]-4*sigma>=0):
            Xmid = np.hstack((X[i]-4*sigma,Xmid))
        else:
            Xmid = np.hstack((X[0],Xmid))
        if (Xmid[-1] != X[i]+4*sigma) & (X[i]-4*sigma<=max(X)):
            Xmid = np.hstack((Xmid,X[i]+4*sigma))
        else:
            Xmid = np.hstack((Xmid,max(X)))
        # Compute delta_x in between values
        delta_xMid = Xmid[1:len(Xmid)]-Xmid[0:len(Xmid)-1]
        # Use normal distribution averaging for bends
        if delta_x <= 0.0006:
            # Compute normal distribution values at Xmid values
            Dmid = normDist(Xmid,X[i],sigma)
            # Compute probabilities
            rho = 0.5*delta_xMid*(Dmid[1:len(Dmid)]+Dmid[0:len(Dmid)-1])
            # Check for bend edge
            if ((X[i]-X[i-1])/(X[i+1]-X[i])>4) or ((X[i+1]-X[i])/(X[i]-X[
    i-1])>4):
                Ts[i] = T[i]
            # Compute weighted average
```

```python
            else:
                Ts[i] = sum(rho*Tdist)/sum(rho)
        # If not a bend use delta_x weigted moving window
        else:
            windowLo = i-int((N+1)/2)
            windowHi = i+int((N+1)/2)+1
            if windowLo < 0:
                windowLo = 0
            if windowHi > len(X):
                windowHi = len(X)
            Twindow = T[windowLo:windowHi]
            XmidAll = np.hstack((X[0],0.5*(X[0:len(X)-1]+X[1:len(X)]),X[-1]))
            deltaXmidAll = XmidAll[1:len(XmidAll)]-XmidAll[0:len(XmidAll)-1]
            deltaXmidW = deltaXmidAll[windowLo:windowHi]
            weights = deltaXmidW/sum(deltaXmidW)
            if min(deltaXmidW)<0.0006:
                Ts[i] = T[i]
            else:
                Ts[i] = sum(Twindow*weights)


    #Return Ts
    return Ts

# Column data extraction loop ------------------------------
def gradientExtractor(pathData,step,simX,simY,simZ,simTemp):

    j = 0
    for i in range(len(pathData[:,0])):

        if(10*i/len(pathData[:,0])>j):
            print('|',end='')
##            print('|')
            j+=1

        if i == 0:
            dataOut = [pathData[i,1],dataExtract([pathData[i,2],pathData[i,3],pathData[i,4]],simX, simY, simZ, simTemp)[0]]
        elif i == 1:
            dataOut = np.vstack((dataOut,[dataOut[0]+pathData[i,1],dataExtract([pathData[i,2], pathData[i,3], pathData[i,4]], simX, simY, simZ, simTemp)]))
        else:
            dataOut = np.vstack((dataOut,[dataOut[i-1,0]+pathData[i,1],dataExtract([pathData[i,2], pathData[i,3], pathData[i,4]], simX, simY, simZ, simTemp)]))
    # Smooth data
    dataOut[:,1] = colGradSmooth(dataOut[:,0],dataOut[:,1],5)
    print('')
    return dataOut

# Data stacker ------------------------------
def dataStacker(dataIn,dataOut):

    if(len(dataOut)==0):
        dataOut = dataIn
    else:
        dataOut = np.vstack((dataOut,dataIn))

    return dataOut

# Main funciton -----------------------------
if __name__ == '__main__':
    pass
```

H.3 Temperature Fit Main File

```
# Title: Temperature Table Creator
# Author: Austin Foster
# Date: February 12, 2018
```

```python

# Import modules
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import dataAnalysisTGGC as dat

# Parameters
tMax = 600

# Select gradient files
file_path1, file_path2 = dat.filePaths()

# Extract file info
time1,time2,deltaT,fileLead = dat.timeDataExtract(file_path1,file_path2)
i = 0

# Read in serpentine path file
filenameP = r'SSChipSerpentinePath.csv'
pathData,step = dat.pathDataReadIn(filenameP)

# Iterative loop
TEMP = []
X = []
while i*deltaT+time1 < tMax:

    filename = dat.fileNameCreator(fileLead,deltaT,i,time1)
    simX,simY,simTemp = dat.simDataReadIn(filename)
    print(str(i*deltaT+time1)+'s:',end='')
##    print(str(i*deltaT+time1)+'s:')
    grad = dat.gradientExtractor(pathData,step,simX,simY,simTemp)
    if(len(TEMP)==0):
        X = grad[:,0]
        T = np.array([i*deltaT+time1])
    else:
        T = np.append(T,i*deltaT+time1)

    TEMP = dat.dataStacker(grad[:,1],TEMP)
    i += 1

# Export results
X,T = np.meshgrid(X,T)
np.savetxt('xDataOut.csv',X,delimiter=',')
np.savetxt('tDataOut.csv',T,delimiter=',')
np.savetxt('temperatureDataOut.csv',TEMP,delimiter=',')
```

APPENDIX I. TRANSPORT MODEL CALIBRATION NOTES

I.1 Experimental Methods

As discussed in Chapter 4, in order to correctly calibrate the transport model for the stainless-steel column, data had to be gathered on the column's separation performance under various conditions. To gather these data isothermal runs had to be performed on the stainless steel column under varying temperature and inlet pressure conditions. To this end a set of 24 experiments were run on the stainless-steel column.

The experiments consisted of four temperature set points (125 °C, 134 °C, 143 °C, 155 °C), and three inlet pressure set points (47.89 psi, 57.07 psi, 65.21 psi). Each pressure setting was to be run twice at each temperature. The oven used in the experiments took a relatively long time to heat up and cool down (approximately 15 minutes), so it was determined that randomizing the oven temperature between runs would not be feasible. Changing the inlet pressure value, however, was a quick change, so it was determined that the order in which the pressure was run should be randomized. For each temperature experiments were run in sets of three pressures, and the order in which these pressures were run was randomized in each set of 3. In order to account for effects on fluctuations in the absolute pressure, atmospheric pressure data was also taken from the weather station on the top of the Eyring Science Center at BYU. As shown in Chapter 4, C12, C13, and C14 were injected in to the column, and following completion of all the runs the data recorded by the GC software was saved as comma separated value files for post-processing.

I.2 Data Preparation

For runs with lower temperatures it was observed that the n-alkane peaks were sitting on the tail end of the solvent peak (Figure I.1). Due to this fact, the software package was unable to successfully identify the peaks and report their elution time and widths at half max. In order to remedy this, a Python script was developed that was capable of working with peaks on the solvent peak. The code relied on user click inputs to select points along the solvent peak, to which a polynomial fit was computed. This polynomial was then subtracted from the data effectively removing the solvent peak from the data. The code then relied on more user input mouse clicks to identify the bounds of each peak. It would then compute the elution time and the peak width at half max for each of the three peaks in each experimental run.

Table I.1: Data from ITGC calibration runs

Run	Temp (°C)	Inlet P (psi)	Outlet P (kPa)	$t_{R,C12}$ (min)	$w_{h,C12}$ (min)	$t_{R,C13}$ (min)	$w_{h,C13}$ (min)	$t_{R,C14}$ (min)	$w_{h,C14}$ (min)
1	125	65.21	83.6	1.591	0.035	2.441	0.058	3.933	0.101
2	125	57.07	83.7	1.818	0.037	2.785	0.061	4.481	0.103
3	125	47.89	83.7	2.191	0.042	3.353	0.068	5.392	0.116
4	125	57.07	83.7	1.818	0.037	2.783	0.059	4.478	0.103
5	125	47.89	83.7	2.184	0.043	3.344	0.068	5.376	0.114
6	125	65.21	83.6	1.583	0.033	2.425	0.055	3.900	0.093
7	143	47.89	83.6	1.509	0.026	2.072	0.037	3.007	0.054
8	143	57.07	83.6	1.257	0.025	1.724	0.033	2.501	0.048
9	143	65.21	83.6	1.101	0.022	1.509	0.030	2.189	0.046
10	143	47.89	83.6	1.518	0.025	2.085	0.036	3.024	0.053
11	143	47.89	83.5	1.510	0.027	2.072	0.036	3.003	0.052
12	143	65.21	83.5	1.098	0.021	1.506	0.029	2.185	0.044
13	143	57.07	83.5	1.252	0.023	1.718	0.033	2.493	0.048
14	155	47.89	83.5	1.268	0.024	1.630	0.030	2.212	0.038
15	155	57.07	83.5	1.057	0.022	1.359	0.027	1.842	0.036
16	155	65.21	83.5	0.920	0.018	1.183	0.023	1.606	0.032
17	155	57.07	83.5	1.051	0.018	1.351	0.023	1.835	0.033
18	155	47.89	83.6	1.264	0.024	1.626	0.030	2.207	0.039
19	155	65.21	83.6	0.919	0.018	1.181	0.023	1.604	0.032
20	134	65.21	83.7	1.293	0.027	1.877	0.038	2.874	0.061
21	134	47.89	83.7	1.779	0.030	2.578	0.045	3.940	0.069
22	134	57.07	83.7	1.482	0.028	2.147	0.0417	3.278	0.064
23	134	65.21	83.7	1.295	0.025	1.876	0.038	2.867	0.061
24	134	47.89	83.7	1.782	0.031	2.579	0.044	3.938	0.069
25	134	57.07	83.8	1.479	0.030	2.143	0.042	3.272	0.064

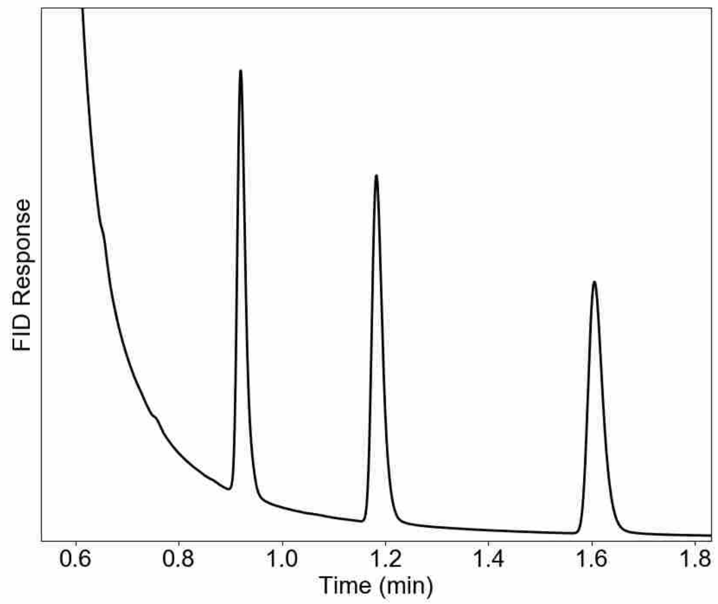


Figure I.1: Chromatogram of an ITGC separation at 155 $°C$ demonstrating the C12, C13, and C14 peaks' tendency to elute on the solvent peak.

I.3 ITGC Parameter Estimation

In order to estimate the enthalpy and entropy parameters for the retention factor model as given in Section 4.4 a model for the average gas velocity was developed by combining the definition for the friction factor with the equation that represents the shear stress that results from Poiselle flow (Equation I.1) [48]. Assuming a linear pressure gradient along the length of the column allowed for the computation of the average velocity (Equation I.2) which was then used in conjuction with the column length and each anayte's retention time to compute the retention factor for each run (Equation I.3).

$$C_f = \frac{2\tau}{\rho u^2} \; ; \; \tau = \frac{area}{perimeter}\left(-\frac{dP}{dx}\right) \tag{I.1}$$

$$\bar{u} = -\frac{D_h^2 \Delta P}{2\mu L (C_f Re_D)} \tag{I.2}$$

$$k = \frac{t_R - t_m}{t_m} = \frac{t_R L}{\bar{u}} - 1 \tag{I.3}$$

Applying the computed retention factors to the equation reported by Blumberg (Equation 4.2) allowed for the use of a linear regression model to determine the values of the enthalpy and entropy parameters. The linear regression fits for these parameters all reported p-values below 4.2×10^{-18} indicating strong statistical significance of the parameters used in the model. Efforts in fitting dispersion parameters was not started until focus shifted from the ITGC data to the TGGC data, thus the results do not demonstrate a close match to peak width. The code used to compute the linear regression fit parameters is given in Section I.8. Using the above method for calibrating the parameters the transport model was shown to give good matchup as shown in Figure I.2. These initial ITGC validations served to demonstrate the utility of the retention factor model indicating it would be a good model in the TGGC runs.

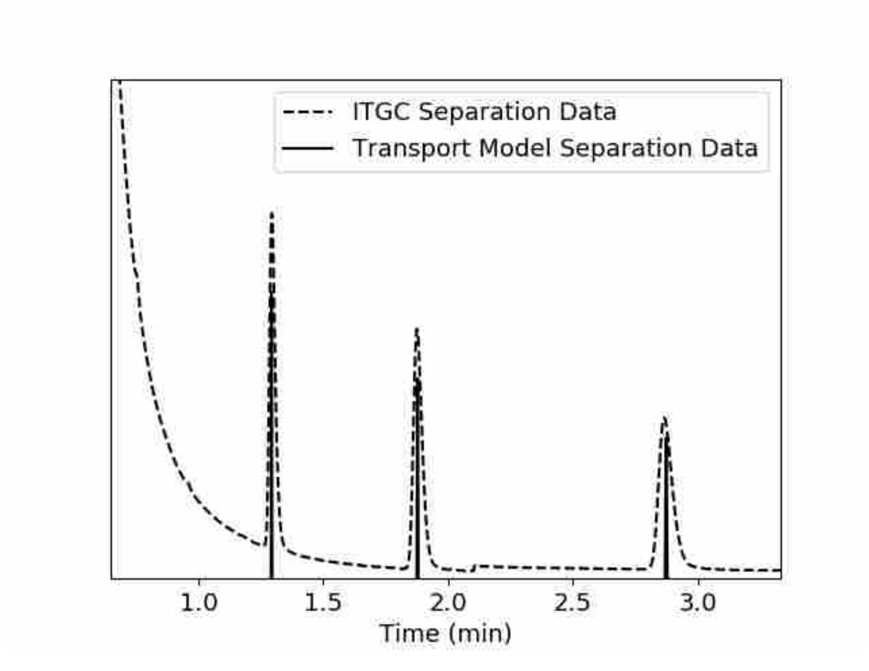


Figure I.2: Plot demonstrating the ITGC transport model with calibrated parameters providing excellent matchup to the experimental data.

I.4 TGGC Parameter Estimation

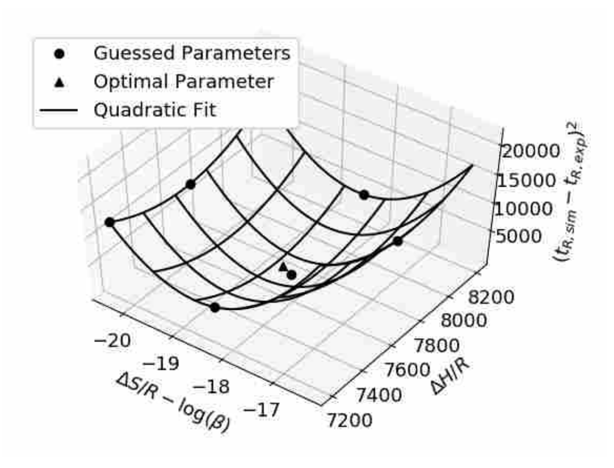
As mentioned in Section 4.4 it was found that the parameters determined from the ITGC runs did not adequately match the elution times from the TGGC runs. Although more work is needed to determine the cause for this incongruency between heating modes, areas of concern include the mobile phase velocity, the channel dimensions, and the true column temperature. As shown in Equation I.2, an average velocity was used along the column length for the ITGC calibrations. It is known, however, that the velocity is non-uniform along the column due to the steep pressure drop near the column outlet, but the calibration method used for the ITGC parameters does not work when using a variable velocity. It has also been difficult to determine the exact dimensions of the channel cross section, which, in turn, make it difficult to determine the correct velocity along the channel. Lastly, doubt has been raised regarding whether the temperatures reported by the GC oven used for ITGC runs are correct, which, if incorrect, would make correct calibration of parameters difficult. Due to these questions, along with the inability of the ITGC parameters to correctly predict TGGC separation behavior, a secondary method was developed to determine the model parameters for TGGC conditions. In order to maintain accuracy in the model, the position dependent pressure and velocity terms were used in the model. These relationships were developed by assuming a non-constant pressure gradient in Equation I.2 [48]. The flow rate was then substituted for the average velocity, and the mobile phase was assumed to be an ideal gas, which allowed for computing the position dependent pressure gradient given in Equation I.4. The position dependent pressure could then be used to compute the position dependent velocity using Equation I.5.

$$P_x = \left(P_i^2 - \frac{\left(P_i^2 - P_o^2\right) \int_0^x T\mu dx}{\int_0^L T\mu dx} \right)^{1/2} \tag{I.4}$$

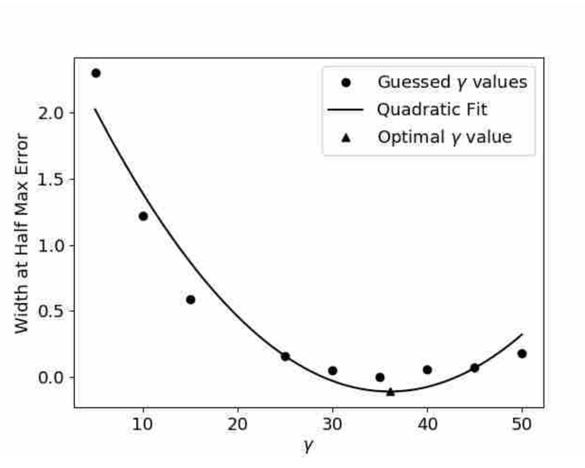
$$u_x = -\left(\frac{dP_x}{dx}\right) \frac{D_h^2}{2\mu(C_f Re_D)} \tag{I.5}$$

This involved running a series of transport model runs with a varying range of parmeters selected as inputs. The results from each of these runs were compared with the experimental results (Table I.2) and the square of the residual was computed in each case for both the retention time and the peak width. Quadratic functions were then fit to the squared residual and the gradient of the quadratic fit was used to estimate the optimal parameter values (see Figure I.3). Note that the γ value indicated in Figure I.3 is a lumping of all the unknown parameters in the Chapman-Enskog model for gas diffusion (Equation I.6, see nomenclature section for definitions of unknown parameters). This process was repeated tightening in the guessed parameter bounds for two different TGGC runs until the optimal parameter values stopped changing. The results from this parameter fitting method is given in Section 4.6, and the code for the quadratic fitting methods are given in Sections I.9 and I.10 below.

$$\gamma = \frac{A\sqrt{1/M_1 + 1/M_2}}{\sigma_{12}^2 \Omega} \tag{I.6}$$



(a)



(b)

Figure I.3: Quadratic fits to computed error data for both (a) retention factor and (b) mass diffusivity for TGGC separations.

I.5 Parameters Estimation Method Comparison

Included in Table I.3 below are parameter values for both calibration methods discussed above. Squared error values between model and experiment for both ITGC and TGGC heating conditions are given for each calibration method as well. It should be noted that the squared error for the ITGC runs using the TGGC quadratic fit method are significantly higher than all other values indicating that more work is needed to refine the model to a point where results can be quantitatively trusted over a wide range of heating conditions. Despite these discrepancies it is believed that the transport model is of value as a qualitative comparison tool between different heating methods.

Table I.2: Data from TGGC calibration runs

Run	Inlet P (psi)	Outlet P (inHg)	Ramp Time (min)	$t_{R,C12}$ (min)	$w_{h,C12}$ (min)	$t_{R,C13}$ (min)	$w_{h,C13}$ (min)	$t_{R,C14}$ (min)	$w_{h,C14}$ (min)
1	65.2	25.13	12	2.246	0.048	3.197	0.058	4.270	0.063
2	65.2	25.15	8	2.313	0.040	3.054	0.044	3.813	0.045
3	65.2	25.17	15	2.622	0.057	3.762	0.069	5.025	0.074
4	47.9	25.17	15	3.721	0.059	5.037	0.066	6.375	0.068
5	47.9	25.17	12	3.004	0.049	4.058	0.054	5.164	0.056
6	47.9	25.18	8	3.067	0.039	3.846	0.039	4.602	0.038
7	65.2	25.16	12	2.490	0.050	3.483	0.058	4.560	0.063

Table I.3: Data comparing results for ITGC and TGGC parameter estimation methods. Note that $\beta_1 = \Delta H/R$ and $\beta_0 = \Delta S/R - \log(\beta)$. Dashes indicate calibrations were not performed for the associated parameters.

		Parameters			ITGC Squared Error		TGGC Squared Error	
Calibration Method	Compound	β_1	β_0	γ	t_R	w_h	t_R	w_h
ITGC Regression	C12	-33.09	13230	-	0.004	-	383.61	-
	C13	-22.19	9135.9	-	0.032	-	193.35	-
	C14	-19.34	8247.5	-	0.257	-	105.00	-
TGGC Quadratic Fit	C12	-22.36	9041.2	27.96	874.27	0.551	34.036	0.239
	C13	-18.69	7967.8	22.19	7191.9	2.806	38.316	0.370
	C14	-17.64	7843.5	16.41	32387	5.364	43.613	1.257

I.6 TPGC Validation Check

As mentioned in Chapter 4 following completion of the calibration and validation process using TGGC data, a series of TPGC runs were performed on the stainless-steel column. The data from these runs are included in Table I.4. For each of the runs the oven was initially set to 50 °C and was ramped to 300 °C. The ambient pressure was 85 kPa for each run, and the split ratio was set at 10:1 for each run. Note also that the inlet pressure given is the gauge pressure reported by the GC oven.

Table I.4: Data from TPGC calibration runs

Run	Ramp Rate (°C/min)	Inlet P (psi)	$t_{R,C12}$ (min)	$w_{h,C12}$ (min)	$t_{R,C13}$ (min)	$w_{h,C13}$ (min)	$t_{R,C14}$ (min)	$w_{h,C14}$ (min)
1	30	47.89	3.872	0.023	4.407	0.023	4.906	0.023
2	30	65.21	3.527	0.028	4.056	0.029	4.55	0.029
3	20	65.21	4.393	0.037	5.159	0.038	5.878	0.04
4	20	47.89	4.828	0.031	5.6	0.03	6.326	0.03
5	20	47.89	4.827	0.032	5.599	0.031	6.325	0.031
6	20	65.21	4.391	0.037	5.155	0.038	5.874	0.04
7	30	65.21	3.524	0.027	4.055	0.028	4.549	0.028
8	30	47.89	3.867	0.023	4.402	0.023	4.9	0.022

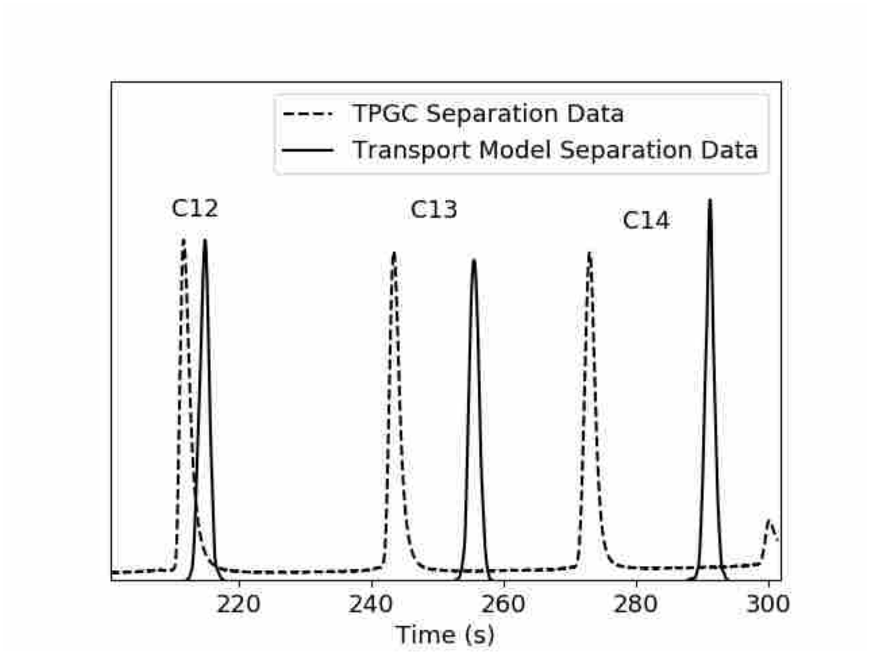


Figure I.4: Comparison of transport model run under TPGC conditions with associated experimental data

Following completion of the TPGC experiments, the transport model was tested using the parameters from the TGGC quadratic fit method discussed in Section I.4 and the conditions from test 6 (Table) I.4. A plot showing the comparison between model results and experimental data is given in Figure I.4. The results from this comparison demonstrate that more work is needed to properly calibrate the transport model across different heating methods. However, the acquisition of this data will be useful in calibrating the transport model as it can work as an intermediate heating method between ITGC and TGGC.

I.7 Data Preparation Code

```
import pdb
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# Read in data
data = np.genfromtxt('TEST 25 121918.csv',delimiter=',',
                     skip_header=3)

# Extract signal
signal = data[:,1]

# Smooth time data
delta_t = 0.001/3
add = 0
if(data[-1,0]==data[-2,0]):
    add += delta_t
    if(data[-1,0]==data[-3,0]):
        add += delta_t
t_i = 0
t_f = data[-1,0]+add
time = np.linspace(t_i,t_f,len(data[:,0]))

# Identify solvent peak curve
fig = plt.figure()
```

```python
plt.title('Select 20 points along the solvent peak')
ax = fig.add_subplot(111)
ax.plot(time,data[:,1])
ax.set_ylim(70000,1600000)
ax.set_xlim(0,2)
coords = []

def onclick(event):
    global ix, iy
    ix, iy = event.xdata, event.ydata
##    print(ix,iy)
    ax.plot(ix,iy,'k*')
    fig.canvas.draw()

    global coords
    coords.append((ix, iy))

    if len(coords) == 20:
        fig.canvas.mpl_disconnect(cid)
        plt.close()

    return coords
cid = fig.canvas.mpl_connect('button_press_event', onclick)
plt.show()

tPts = np.zeros(20)
sPts = np.zeros(20)
for i in range(20):
    temp1 = round(coords[i][0],3)
    temp3 = data[:,0]
    temp4 = np.abs(temp3-temp1)
    itemindex = np.where(temp4 == temp4.min())
    if len(itemindex[0]) > 1:
        temp5 = [item[0] for item in itemindex]
        itemindex = temp5
    tPts[i] = temp1
    sPts[i] = data[itemindex[0],1]
coords2 = coords

# Compute solvent peak fit
def f(t,tau,A,B,C):
    return A/(tau*(t-B))+C

##init = [20,5e5,0.65,6e4]
init = [ 1.21071996e+01,  2.05929691e+06,  7.02688282e-01, -1.53243723e
   +05]
popt,pcov = curve_fit(f,tPts,sPts,init)
#print(popt)
tFit = np.linspace(tPts[0],tPts[-1],1000)
#sFit = f(tFit,*popt)
z = np.polyfit(tPts,sPts,10)
z1 = np.poly1d(z)
sFit = z1(tFit)

plt.plot(time,data[:,1],'r--')
plt.plot(tPts,sPts,'k*')
plt.plot(tFit,sFit,'g:')
plt.xlim(min(tFit)-0.5,max(tFit)+0.5)
plt.ylim(min(sPts)-10000,max(sPts)+10000)
plt.show()
pdb.set_trace()

# Subtract solvent peak
tPh1 = time[time>min(tFit)]
sPh1 = signal[time>min(tFit)]
tInt = tPh1[tPh1<max(tFit)]
sInt = sPh1[tPh1<max(tFit)]
#solv = f(tInt,*popt)
solv = z1(tInt)
peaks = sInt - solv


```

```
# Find new baseline
base1 = np.mean(peaks)
base = np.mean(peaks[peaks<base1])
error = np.mean(abs(peaks[peaks<base1]-base))
print error

plt.plot(tInt,peaks)
plt.plot([tInt[0],tInt[-1]],[base,base])
plt.show()

# Identify individual peaks
pk3 = peaks[peaks>base]
tpk3 = tInt[peaks>base]

fig = plt.figure()
plt.title('Select left and right sides of each peak')
ax = fig.add_subplot(111)
ax.plot(tpk3,pk3,'k.')
coords = []

def onclick(event):
    global ix, iy
    ix, iy = event.xdata, event.ydata
##    print(ix,iy)
    ax.plot(ix,iy,'g*')
    fig.canvas.draw()

    global coords
    coords.append((ix, iy))

    if len(coords) == 6:
        fig.canvas.mpl_disconnect(cid)
        plt.close()

    return coords
cid = fig.canvas.mpl_connect('button_press_event', onclick)
```

```
plt.show()

C12bnds = [coords[0][0],coords[1][0]]
C13bnds = [coords[2][0],coords[3][0]]
C14bnds = [coords[4][0],coords[5][0]]

tC12 = tpk3[(tpk3>=C12bnds[0]) & (tpk3<=C12bnds[1])]
sC12 = pk3[(tpk3>=C12bnds[0]) & (tpk3<=C12bnds[1])]

tC13 = tpk3[(tpk3>=C13bnds[0]) & (tpk3<=C13bnds[1])]
sC13 = pk3[(tpk3>=C13bnds[0]) & (tpk3<=C13bnds[1])]

tC14 = tpk3[(tpk3>=C14bnds[0]) & (tpk3<=C14bnds[1])]
sC14 = pk3[(tpk3>=C14bnds[0]) & (tpk3<=C14bnds[1])]

# Find elution times
teC12 = tC12[sC12==max(sC12)][0]
teC13 = tC13[sC13==max(sC13)][0]
teC14 = tC14[sC14==max(sC14)][0]

# Find widths at half max
halfMax = max(sC12)/2
halfC12 = abs(sC12 - halfMax)
l12 = tC12[halfC12==min(halfC12[tC12<teC12])][0]
r12 = tC12[halfC12==min(halfC12[tC12>teC12])][0]
whC12 = r12-l12
hwhC12 = teC12-l12


halfMax = max(sC13)/2
halfC13 = abs(sC13 - halfMax)
l13 = tC13[halfC13==min(halfC13[tC13<teC13])][0]
r13 = tC13[halfC13==min(halfC13[tC13>teC13])][0]
whC13 = r13-l13
hwhC13 = teC13-l13

```

```
halfMax = max(sC14)/2
halfC14 = abs(sC14 - halfMax)
l14 = tC14[halfC14==min(halfC14[tC14<teC14])][0]
r14 = tC14[halfC14==min(halfC14[tC14>teC14])][0]
whC14 = r14-l14
hwhC14 = teC14-l14

baseMax = max(sC12)*.1
baseC12 = abs(sC12 - baseMax)
bl12 = tC12[baseC12==min(baseC12[tC12<teC12])][0]
br12 = tC12[baseC12==min(baseC12[tC12>teC12])][0]
basewidth12 = br12-bl12

baseMax = max(sC13)*.1
baseC13 = abs(sC13 - baseMax)
bl13 = tC13[baseC13==min(baseC13[tC13<teC13])][0]
br13 = tC13[baseC13==min(baseC13[tC13>teC13])][0]
basewidth13 = br13-bl13

baseMax = max(sC14)*.1
baseC14 = abs(sC14 - baseMax)
bl14 = tC14[baseC14==min(baseC14[tC14<teC14])][0]
br14 = tC14[baseC14==min(baseC14[tC14>teC14])][0]
basewidth14 = br14-bl14

#Check that points were not chosen inside basewidth
for i in range(20):
    temp1 = round(coords2[i][0],3)
    if abs(teC12 - temp1)<basewidth12/2:
       print('Point ', i, ' was too close to C12')
    if abs(teC13 - temp1)<basewidth12/2:
       print('Point ', i, ' was too close to C13')
    if abs(teC13 - temp1)<basewidth12/2:
       print('Point ', i, ' was too close to C14')

if max(sC12)> max(sC13):
    if max(sC12) > max(sC14):
        peak_int = np.mean(abs(sC12-base))
        percenterr = error/peak_int
if max(sC13)> max(sC12):
    if max(sC13) > max(sC14):
        peak_int = np.mean(abs(sC13-base))
        percenterr = error/peak_int
if max(sC14)> max(sC13):
    if max(sC14) > max(sC12):
        peak_int = np.mean(abs(sC14-base))
        percenterr = error/peak_int

plt.plot(tC12,sC12,'k')
plt.plot(teC12,max(sC12),'g*')
plt.plot(l12,max(sC12)/2,'r*')
plt.plot(r12,max(sC12)/2,'b*')

plt.plot(tC13,sC13,'k')
plt.plot(teC13,max(sC13),'g*')
plt.plot(l13,max(sC13)/2,'r*')
plt.plot(r13,max(sC13)/2,'b*')

plt.plot(tC14,sC14,'k')
plt.plot(teC14,max(sC14),'g*')
plt.plot(l14,max(sC14)/2,'r*')
plt.plot(r14,max(sC14)/2,'b*')

print('C12:')
print('t = ',teC12)
print('wh = ',whC12)
print('hwh = ',hwhC12)

print('C13:')
print('t = ',teC13)
print('wh = ',whC13)
print('hwh = ',hwhC13)

print('C14:')
print('t = ',teC14)
print('wh = ',whC14)
print('hwh = ',hwhC14)

print('Percent Error', percenterr)

plt.show()
```

I.8 ITGC Parameter Linear Regression Code

```
# Import modules
import pandas as pd
import statsmodels.formula.api as smf
import HeliumViscocity as hv
import numpy as np
import matplotlib.pyplot as plt

# Parameters
Dh = 57e-6
L = 9.62496
CfRe = 15.697072267489121

# Filename
filename = 'SteelChipDataIsothermal-020519'

# Read in data
df = pd.read_csv(filename+'.csv')
##print(df)

# Add viscocity column
df['Viscocity'] = hv.Helium_visco_2(df['TEMP']+273.15)


# Compute pressure drop
```

```
df['deltaP'] = df['P_IN']*6894.757

# Compute velocities
df['Vcomp'] = Dh**2*df['deltaP']/(2*df['Viscocity']*L*CfRe)

# Compute mobile phase time
df['tm'] = L/df['Vcomp']

# Compute retention factors
df['k12'] = (60*df['TR_C12']-df['tm'])/df['tm']
df['k13'] = (60*df['TR_C13']-df['tm'])/df['tm']
df['k14'] = (60*df['TR_C14']-df['tm'])/df['tm']
print(df['k12'])

# Compute natural log of retention factor
df['ln_k12'] = np.log(df['k12'])
df['ln_k13'] = np.log(df['k13'])
df['ln_k14'] = np.log(df['k14'])

# Compute inverse kelvin
df['TK_inv'] = 1/(df['TEMP']+273.15)

# Plot data
plt.plot(df['TK_inv'],df['ln_k12'],'r.')
plt.plot(df['TK_inv'],df['ln_k13'],'g.')
plt.plot(df['TK_inv'],df['ln_k14'],'b.')
plt.show()

# Create model
mod = smf.ols(formula='ln_k12 ~ TK_inv', data = df)

# Fit model
res = mod.fit()

# Display results
print(res.summary())
print(res.pvalues)

# Plot
Tinv = np.linspace(1/400,1/550,100)
T = 1/Tinv
fit12 = res.params[0] + res.params[1]*Tinv
k12 = np.exp(fit12)
plt.plot(Tinv,fit12,'r--')

# Create model
mod = smf.ols(formula='ln_k13 ~ TK_inv', data = df)

# Fit model
res = mod.fit()

# Display results
print(res.summary())
print(res.pvalues)

# Plot
fit13 = res.params[0] + res.params[1]*Tinv
k13 = np.exp(fit13)
plt.plot(Tinv,fit13,'g--')

# Create model
mod = smf.ols(formula='ln_k14 ~ TK_inv', data = df)

# Fit model
res = mod.fit()

# Display results
print(res.summary())
print(res.pvalues)

# Plot
fit14 = res.params[0] + res.params[1]*Tinv
```

```
k14 = np.exp(fit14)
plt.plot(Tinv,fit14,'b--')

plt.show()

plt.plot(T,k12,'r-')
plt.plot(T,k13,'g-')
plt.plot(T,k14,'b-')
plt.show()
```

I.9 TGGC Retention Parameters Fit Code

```
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt

# Parameters
C12 = 138.6
C13 = 183.2
C14 = 228.6

# Read in data
df = pd.read_csv('BetaValues.csv')
print(df.columns.values)

# Squared error parameters
errC12 = (df['tR_C12']-C12)**2
errC13 = (df['tR_C13']-C13)**2
errC14 = (df['tR_C14']-C14)**2

# Compute C12 values
BetaMat = np.transpose(np.vstack((np.ones(6),df['Beta0_C12'],df['
    Beta1_C12'],df['Beta0_C12']*df['Beta1_C12'],df['Beta0_C12']**2,df['
    Beta1_C12']**2)))
fVec = np.transpose(errC12)
```

```
alpha = np.linalg.solve(BetaMat,fVec)

AMat = np.array([[2*alpha[4],alpha[3]],[alpha[3],2*alpha[5]]])
bVec = np.array([[-alpha[1]],[-alpha[2]]])
BetaOpt12 = np.linalg.solve(AMat,bVec)
print(BetaOpt12)

# Plot C12 Fit
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100
Beta0_grid = np.linspace(min(df['Beta0_C12']),max(df['Beta0_C12']),n)
Beta1_grid = np.linspace(min(df['Beta1_C12']),max(df['Beta1_C12']),n)
Beta0_gird,Beta1_grid = np.meshgrid(Beta0_grid,Beta1_grid)
sqrErr = alpha[0] + alpha[1]*Beta0_grid + alpha[2]*Beta1_grid + alpha[3]*
    Beta0_grid*Beta1_grid + alpha[4]*Beta0_grid**2 + alpha[5]*Beta1_grid
    **2
ax.plot_wireframe(Beta0_grid, Beta1_grid, sqrErr, rstride=10, cstride=10)
ax.plot(df['Beta0_C12'],df['Beta1_C12'],errC12,'k.')
BetaOptErr = alpha[0] + alpha[1]*BetaOpt12[0] + alpha[2]*BetaOpt12[1] +
    alpha[3]*BetaOpt12[0]*BetaOpt12[1] + alpha[4]*BetaOpt12[0]**2 + alpha
    [5]*BetaOpt12[1]**2
ax.plot(BetaOpt12[0],BetaOpt12[1],BetaOptErr,'r*')
ax.set_xlabel(r'$\beta_0$')
ax.set_ylabel(r'$\beta_1$')
ax.set_zlabel(r'$(t_{R,sim}-t_{R,exp})^2$')
ax.xaxis.labelpad=10
ax.yaxis.labelpad=10
ax.zaxis.labelpad=10
plt.show()

# Compute C13 values
BetaMat = np.transpose(np.vstack((np.ones(6),df['Beta0_C13'],df['
    Beta1_C13'],df['Beta0_C13']*df['Beta1_C13'],df['Beta0_C13']**2,df['
    Beta1_C13']**2)))
fVec = np.transpose(errC13)
```

```python
alpha = np.linalg.solve(BetaMat,fVec)

AMat = np.array([[2*alpha[4],alpha[3]],[alpha[3],2*alpha[5]]])
bVec = np.array([[-alpha[1]],[-alpha[2]]])
BetaOpt13 = np.linalg.solve(AMat,bVec)
print(BetaOpt13)

# Plot C13 Fit
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100
Beta0_grid = np.linspace(min(df['Beta0_C13']),max(df['Beta0_C13']),n)
Beta1_grid = np.linspace(min(df['Beta1_C13']),max(df['Beta1_C13']),n)
Beta0_gird,Beta1_grid = np.meshgrid(Beta0_grid,Beta1_grid)
sqrErr = alpha[0] + alpha[1]*Beta0_grid + alpha[2]*Beta1_grid + alpha[3]*
    Beta0_grid*Beta1_grid + alpha[4]*Beta0_grid**2 + alpha[5]*Beta1_grid
    **2
ax.plot_wireframe(Beta0_grid, Beta1_grid, sqrErr, rstride=10, cstride=10)
ax.plot(df['Beta0_C13'],df['Beta1_C13'],errC13,'k.')
BetaOptErr = alpha[0] + alpha[1]*BetaOpt13[0] + alpha[2]*BetaOpt13[1] +
    alpha[3]*BetaOpt13[0]*BetaOpt13[1] + alpha[4]*BetaOpt13[0]**2 + alpha
    [5]*BetaOpt13[1]**2
ax.plot(BetaOpt13[0],BetaOpt13[1],BetaOptErr,'r*')
ax.set_xlabel(r'$\beta_0$')
ax.set_ylabel(r'$\beta_1$')
ax.set_zlabel(r'$(t_{R,sim}-t_{R,exp})^2$')
ax.xaxis.labelpad=10
ax.yaxis.labelpad=10
ax.zaxis.labelpad=10
plt.show()

# Compute C14 values
BetaMat = np.transpose(np.vstack((np.ones(6),df['Beta0_C14'],df['
    Beta1_C14'],df['Beta0_C14']*df['Beta1_C14'],df['Beta0_C14']**2,df['
    Beta1_C14']**2)))
fVec = np.transpose(errC14)
```

```
alpha = np.linalg.solve(BetaMat,fVec)

AMat = np.array([[2*alpha[4],alpha[3]],[alpha[3],2*alpha[5]]])
bVec = np.array([[-alpha[1]],[-alpha[2]]])
BetaOpt14 = np.linalg.solve(AMat,bVec)
print(BetaOpt14)

# Plot C14 Fit
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100
Beta0_grid = np.linspace(min(df['Beta0_C14']),max(df['Beta0_C14']),n)
Beta1_grid = np.linspace(min(df['Beta1_C14']),max(df['Beta1_C14']),n)
Beta0_gird,Beta1_grid = np.meshgrid(Beta0_grid,Beta1_grid)
sqrErr = alpha[0] + alpha[1]*Beta0_grid + alpha[2]*Beta1_grid + alpha[3]*
    Beta0_grid*Beta1_grid + alpha[4]*Beta0_grid**2 + alpha[5]*Beta1_grid
    **2
ax.plot_wireframe(Beta0_grid, Beta1_grid, sqrErr, rstride=10, cstride=10)
ax.plot(df['Beta0_C14'],df['Beta1_C14'],errC14,'k.')
BetaOptErr = alpha[0] + alpha[1]*BetaOpt14[0] + alpha[2]*BetaOpt14[1] +
    alpha[3]*BetaOpt14[0]*BetaOpt14[1] + alpha[4]*BetaOpt14[0]**2 + alpha
    [5]*BetaOpt14[1]**2
ax.plot(BetaOpt14[0],BetaOpt14[1],BetaOptErr,'r*')
ax.set_xlabel(r'$\beta_0$')
ax.set_ylabel(r'$\beta_1$')
ax.set_zlabel(r'$(t_{R,sim}-t_{R,exp})^2$')
ax.xaxis.labelpad=10
ax.yaxis.labelpad=10
ax.zaxis.labelpad=10
plt.show()
```

I.10 TGGC Dispersion Parameter Fit Code

```
import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import axes3d
```

```
import matplotlib.pyplot as plt
import matplotlib
font = {'size'   : 13}
matplotlib.rc('font', **font)

# Parameters
C12 = 2.4
C13 = 2.65
C14 = 2.7
C12_T3 = 3.4
C13_T3 = 4.15
C14_T3 = 4.45

# Read in data
df = pd.read_csv('GammaValues.csv')
print(df.columns.values)

# Squared error parameters
errC12 = (df['WhT2_C12']-C12)**2
errC13 = (df['WhT2_C13']-C13)**2
errC14 = (df['WhT2_C14']-C14)**2

errC12_T3 = (df['WhT3_C12']-C12)**2
errC13_T3 = (df['WhT3_C13']-C13)**2
errC14_T3 = (df['WhT3_C14']-C14)**2

##errC12 += errC12_T3
##errC13 += errC13_T3
##errC14 += errC14_T3

# Compute C12 values
mask = df['Gamma0_C12']<500
P12 = np.polyfit(df['Gamma0_C12'][mask],errC12[mask],2)
gammaMin = -P12[1]/(2*P12[0])
Emin = P12[0]*gammaMin**2 + P12[1]*gammaMin + P12[2]
print('Min: ',-P12[1]/(2*P12[0]))
```

```
P12_T3 = np.polyfit(df['Gamma0_C12'][mask],errC12_T3[mask],2)

# Plot C12 Fit
GammaArr = np.linspace(min(df['Gamma0_C12']),max(df['Gamma0_C12']),100)
errFit = np.zeros_like(GammaArr)
errFit_T3 = np.zeros_like(GammaArr)
for i in range(len(P12)):
    exp = 2-i
    errFit += P12[i]*GammaArr**exp
    errFit_T3 += P12_T3[i]*GammaArr**exp
plt.plot(df['Gamma0_C12'],errC12,'ko',label=r'Guessed $\gamma$ values')
plt.plot(df['Gamma0_C12'],errC12_T3,'b.',label='Data Points: T3')
plt.plot(GammaArr,errFit,'k-',label='Quadratic Fit')
plt.plot(GammaArr,errFit_T3,'m-',label='Data Fit: T3')
plt.plot(gammaMin,Emin,'k^',label=r'Optimal $\gamma$ value')
plt.xlabel(r'$\gamma$')
plt.ylabel('Width at Half Max Error')
plt.legend(loc=0)
plt.show()


# Compute C13 values
mask = df['Gamma0_C13']<500
P13 = np.polyfit(df['Gamma0_C13'][mask],errC13[mask],2)
print('Min: ',-P13[1]/(2*P13[0]))
P13_T3 = np.polyfit(df['Gamma0_C13'][mask],errC13_T3[mask],2)

# Plot C13 Fit
GammaArr = np.linspace(min(df['Gamma0_C13']),max(df['Gamma0_C13']),100)
errFit = np.zeros_like(GammaArr)
errFit_T3 = np.zeros_like(GammaArr)
for i in range(len(P13)):
    exp = 2-i
    errFit += P13[i]*GammaArr**exp
    errFit_T3 += P13_T3[i]*GammaArr**exp
plt.plot(df['Gamma0_C13'],errC13,'k.',label='Data Points: T2')
plt.plot(GammaArr,errFit,'r-',label='Data Fit: T2')
plt.plot(df['Gamma0_C13'],errC13_T3,'b.',label='Data Points: T3')
plt.plot(GammaArr,errFit_T3,'m-',label='Data Fit: T3')
plt.show()


# Compute C14 values
mask = df['Gamma0_C14']<500
P14 = np.polyfit(df['Gamma0_C14'][mask],errC14[mask],2)
print('Min: ',-P14[1]/(2*P14[0]))
P14_T3 = np.polyfit(df['Gamma0_C14'][mask],errC14_T3[mask],2)

# Plot C14 Fit
GammaArr = np.linspace(min(df['Gamma0_C14']),max(df['Gamma0_C14']),100)
errFit = np.zeros_like(GammaArr)
errFit_T3 = np.zeros_like(GammaArr)
for i in range(len(P14)):
    exp = 2-i
    errFit += P14[i]*GammaArr**exp
    errFit_T3 += P14_T3[i]*GammaArr**exp
plt.plot(df['Gamma0_C14'],errC14,'k.',label='Data Points: T2')
plt.plot(GammaArr,errFit,'r-',label='Data Fit: T2')
plt.plot(df['Gamma0_C14'],errC14_T3,'b.',label='Data Points: T3')
plt.plot(GammaArr,errFit_T3,'m-',label='Data Fit: T3')
plt.show()
```