8-2020

Transient Thermal Modeling of Bioprocessing Equipment

Cody M. Cummings
Utah State University

TRANSIENT THERMAL MODELING OF BIOPROCESSING EQUIPMENT

By

Cody M. Cummings

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Mechanical Engineering

Approved:

_____	_____
Hailei Wang, Ph.D. Major Professor	Barton Smith, Ph.D. Committee Member
_____	_____
Geordie Richards, Ph.D. Committee Member	Mark Smith, Ph.D. Committee Member

_____
Janis L. Boettinger, Ph.D.
Acting Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2020

ABSTRACT

Transient Thermal Modeling of Bioprocessing Equipment

by

Cody M. Cummings

Utah State University, 2020

Major Professor: Dr. Hailei Wang
Department: Mechanical and Aerospace Engineering

Bioprocessing is leveraging cells to produce high value, lifesaving products. Precise environmental control is essential to maintain integrity of the bioprocessing production process, which requires both appropriate equipment choice of the temperature control unit (TCU) and proper control parameter selection in order to reach the targeted process temperature in the desirable rate. To optimize the TCU selection and the associated control parameters, a transient thermal model of typical bioprocessing system is developed to help predict the process temperature profiles. The model captures the heat transfer processes and temperature dependent fluid and flow properties. The control systems for both the bioreactors and TCUs were modeled in detail to reflect their system response *in silico*. Physical experiments were also conducted across a range of bioreactors, from 50L to 2000L, in order to validate the model. Various TCU size ranged from 1.2kW to 18kW were used in the experiment in order to broaden the model application. The measured time associated with each temperature was compared with the model prediction, which shows in good agreement. According to the total of 42

experiments, the predicted overall heat transfer coefficients match reasonably well with the experimental data. The developed model was compared to a first order estimate, time to temperature set point predictions were significantly better in the developed model. Multiple jacket-side Nusselt number correlations were also compared against the experimental data to provide additional insight of the heat transfer process.

(130 pages)

PUBLIC ABSTRACT

Transient Thermal Modeling of Bioprocess Equipment

Cody M. Cummings

Bioprocessing is leveraging cells to produce high value, lifesaving products. Precise environmental control is needed to maintain integrity of the bioprocessing production process. Temperature control requires both appropriate equipment choice and correct control parameter selection. To aid in the equipment selection process, enable better understanding of equipment capacity, and enable optimization of control parameters, a transient thermal model of both heat transfer characteristics and control systems was created *in silico*.

ACKNOWLEDGMENTS

I would like to thank Dr. Hailei Wang for his mentorship and persistence in helping me complete this thesis, also Dr. Mark Smith and Thermo Fisher Scientific for the idea behind the project and resources of both funding and lab environment. I owe a debt to Dr. Areti Kiara for teaching me to always be deliberate, and for being an example of vested interest in the success of those around her.

I could not have finished this without the constant support of my wife and the smiles and giggles of my son.

Cody M. Cummings

CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Acronyms

ARW – Anti-Reset Windup

PID – Proportional Integral Derivative

SUB – Single Use Bioreactor

SUF – Single Use Fermentor

SUM – Single Use Mixer

TCU – Temperature Control Unit

Variables

A – Heat transfer area

c_p – Specific heat

Co – Controller output

Co_{bias} - Co bias, a baseline output

D – Diameter

d_{coil} – Diameter of coil curvature

D_e – Dean Number

d_h - Hydraulic diameter

D_T – Turbine diameter

dt – Time step

$e\%$ Percent error

e_k - Iteration error

e_{sum_k} - Summation of iteration error

$e(t)$ - Error

k – Thermal conductivity

K_c – Controller gain

K_d - Derivative gain

K_i - Integral gain

K_p – Proportional gain

k_v - Flow factor

L - Jacket flow path length

N – Revolutions per second

n_p – Number of impeller blades

Nu – Nusselt number

PB – Proportional band

Pr – Prandtl number

PV – Process value (temperature)

Q – Heat transfer

Re – Reynolds number

s - Uncertainty

SG – Specific gravity

SP – Set point temperature

$T50\%$ - Time to 50% of set point

$T90\%$ - Time to 90% of set point

$T95\%$ - Time to 95% of set point

T_d – Damping time

T_i – Integral time

ΔT_{lm}- Log mean temperature difference

T_n – Reset time (Lauda TCU)

T_r – Reset time

T_v – Derivative time (Lauda TCU)

U – Overall heat transfer coefficient

V – Surface velocity

Vi – Viscosity ratio

X_p – Proportional Band (Lauda TCU)

Xp_f – Slave proportional band (Lauda TCU)

$\forall$ - Volumetric flow Rate

ϵ – Power ratio

μ – Dynamic Viscosity

μ^* - Dynamic Viscosity (centipoise)

ν – Kinematic viscosity

ξ – Darcy friction factor

ρ - Density

ω – Radians per second

$\eta_{TCU} - \frac{Power\ Out}{Power\ Nominal}$

CHAPTER I

INTRODUCTION

1.1 Motivation

Bioprocessing is an increasingly popular method of manufacturing a product using living cells. It leverages cells as micro factories to produce high value, life changing biologics. Of the top ten selling prescription drugs, seven are produced through bioprocesses [1]. Five of those seven are for oncology (anti-cancer) treatments. To ensure the successful production of these lifesaving biologics, precise environmental controls are needed. To be successful, the production environment needs to mimic the native biological system of the cells. These controls include parameters such as pH, dissolved gasses, nutrients and temperature. Without precise control, the quality of the drug decreases, with worst case being potential toxicity to the end user. Homogeneity across repeated batches allows optimization of those processes, resulting in higher yields and lower costs.

In this work, accurate temperature modeling and control are the goal. For bioprocessing, a control system generally consists of a variety of equipment including jacketed vessels, temperature control units (TCUs), pumps, chillers, and heaters. When designing and implementing a temperature control system, two primary challenges arise 1) right-sizing equipment for performance and 2) optimizing the control strategy.

Sizing the equipment correctly and effectively requires balancing costs associated with the equipment, both in terms of up-front capital and verification costs, and ongoing utility, power, and maintenance costs. First, the equipment must be able to heat and cool

the bioreactors within desired timeframes. Second, while achieving the intended functions, it is important to optimize various system design parameters and component selections, such as chiller cooling capacity and coefficient of performance, pump power and the associated hose lengths, jacket thicknesses, and the heat transfer fluid.

Figure 1: Example bioreactor and controller. [2]

Selecting the control parameters for the bioreactor vessel and TCU controller also plays an important role in the optimization process. If the incorrect parameters are chosen several negative performance issues may occur, including temperature overshoot, ringing, or hesitance approaching the setpoint. Temperature overshoot can result in batch failure. Ringing can induce premature mechanical system failure due to increased cycles. Decreased performance can reduce actual ability to heat/chill compared to rated power.

Currently, physical experiments are performed in order to tune the control parameters. However, the tuning process can take days or weeks to complete and may cause disruption of current production runs. Having a virtual environment to test various vessel/TCU combinations in conjunction with control parameters would greatly reduce the timeframe required and would enable manufactures to make accurate parameter selections, avoiding all the aforementioned control problems while reducing labor and utility costs.

The goal of this work was to create a flow and heat transfer model from fundamental principles and published correlations of heat transfer and fluid mechanics that would improve predictive power of systems modelling. Experimental data on actual bioreactor systems was collected to refine and validate the model. The outcome was a validated model to 1) enable in silico evaluation for right-sizing thermal-systems equipment and 2) enable evaluation and optimization of control loop parameters.

1.2 Literature Review

In order to match with the actual experiment, both heat transfer modeling and controller modeling are important. The heat transfer model includes the following thermal resistances: convection within the vessel, conduction through the wall, and convection within the jacket.

As bioprocessing has grown in both capabilities and scale of use over the past century, thorough examinations have been conducted on the heat transfer phenomena involved. Chilton et al. [3] made strides in describing the forced flow within the vessel. A

general correlation to predict heat transfer coefficients within a vessel stirred by a flat paddle agitator (the common type during the time) was presented. On the jacket side, Lehrer [4] examined several different geometries and found good agreement between experimentally derived and calculated results. Mohan et al. [5] provided an in-depth review of published literature before 1992. Mohan concluded that the previously established correlations are only valid when geometric and process similarity exists, and that even then the range of scale within validation is limited. Using Reynolds numbers evaluated at the heat transfer surface is recommended rather than at the impeller tip, as most had done up to that point. The most commonly applied equation for the average Nusselt number within the vessel was of the form of Eq. 1 with k being impeller and geometry dependent.

$$Nu_R = k\, Re_R^a Pr_R^b\, Vi_R^c \; with \; a = \frac{2}{3}, b = \frac{1}{3}, c = 0.14 \qquad (1)$$

Different methods have been used to determine the heat transfer coefficient U. Kai and Shengyao [6] focused on a shear rate model for determining the apparent viscosity of fluids inside the vessel. The heat transfer coefficients were correlated to a generalized Reynolds number and a dimensionless group $\frac{\epsilon D^4}{\nu^3}$ where ϵ is the power per unit mass. These correlations extended better to non-Newtonian fluids than previous models but did not offer additional benefits for Newtonian fluids.

Haam et al [7] worked to predict local heat transfer coefficients inside mixing vessels. Using heat flux sensors, they found that Reynolds number was the primary driver

of the heat transfer coefficient and vessel geometry of secondary importance. Karcz [8] and Bielka et al [9] also did work determining local heat transfer coefficients. They produced experimental data for multiple impeller types, as well as gas-liquid systems.

A preliminary estimate for Reynolds number within the jacket for our planned experiments shows a range between 1500 and 20,000. This is dependent upon on temperature, pump power, hose length, vessel size, and glycol percentage within the water. Given the flow ranges from laminar, transitional, and fully turbulent regimes, the Gnielinski [10] correlation was considered initially to predict Nusselt number in this study. It is based on published experimental results across the transitional regime and used a linear interpolation of Reynolds number from 2300 to 4000. Taler [11] built on the Gnielinski correlation and changed the interpolation method to better fit new experimental data. The resulting equation for the Nusselt number inside tubes with a constant wall temperature was determined to be:

$$Nu = Nu_{m,T}(Re = 2300) + \frac{\frac{\xi}{8}(Re - 2300)Pr^{1.008}}{1.08 + 11.31\sqrt{\frac{\xi}{8}}(Pr^{2/3} - 1)} * \left[1 + \left(\frac{d_h}{L}\right)^{2/3}\right] \quad (2)$$

Additionally, for coiled jackets, Dhotre et al [12] provides a correlation for the jacket side Nusselt Number. This correlation cannot be directly applied due to the nature of coiled vs dimpled jacket flow paths, but future works may integrate this into the model.

$$Nu_{laminar} = 0.788(De)^{0.6}, Nu_{turbulent} = 0.21(De)^{0.7} \quad (3)$$

$$De = \left(\frac{d_h \rho V}{\mu}\right)\left(\frac{d_h}{d_{coil}}\right)^{0.5}$$

Since the Taler correlations are based on circular flow passages, for a non-circular jacket flow path, an examination of the applicability of general circular duct flow equations was necessary. Duan [13] examined heat transfer in turbulent flow in non-circular ducts. He proposed a more appropriate length scale for defining non-dimensional parameters suggesting that using the square root of the cross-sectional flow area rather than the hydraulic diameter produces comparatively better correlation between circular and non-circular ducts. Likewise, in another paper authored by Duan [14], pressure drop inside fully developed turbulent flows was examined. He found that similarly, using the square root of the cross-sectional area as the length scale, along with a modified Blasius equation resulted in a closer fit to experimental data across Reynolds numbers of 5000 to 100000 and varying shapes including the narrow rectangle used inside the jackets to be studied in this proposal.

Mori [15] examined the thermal resistances involved in scale up for stirred tank and tubular reactors. They found that the balance between conductive and convective thermal resistances changed with an increase in reactor size.

Löffelholz et al [16] reviewed different methods for parameterizing Single Use Bioreactors. Different recommendations were given based off the type of reactor used.

For the rigid cultivation systems used in our experiment, they recommend characterizing the vessel side convection via fluid velocity and torque measurement.

Recently (2018) Muller et al [17], examined the combined heat transfer of jacket, wall, and vessel in a single-use bioreactor (SUB), similar to those used in our experiments. Within the reactor, an equation of the form: $Nu_R = k\ Re_R^a Pr_R^b\ Vi_R^c$ was used. On the jacket side, Nusselt correlations were determined by approximating the geometry as a flow over a flat plate with rectangular cross section with heat transfer on one side. This demonstrated that the overall heat transfer coefficients for a SUB can be determined both in transient and steady state conditions. Within the subject configuration, the vessel side heat transfer had the least impact on the overall coefficient. The conduction thermal resistance of the vessel polymer film and the convection resistance of the jacket side are dominant. Differing from the experiments in this work, the jacket flow stayed in a laminar regime, and their scale was limited to 50L-200L. They found their combined correlations to underestimate the overall heat transfer coefficients by 10 percent. It should be noted that Muller et al was not attempting to predict the overall performance of a system (heat/chill timeframes, etc.), but rather to accurately compare the heat transfer characteristics of polymer lined single use technology (SUT) systems with traditional bioreactors.

Given the material properties of the heat transfer fluid are strongly dependent on temperature, an accurate model of fluid properties is necessary. Sun et al [18] worked to measure material properties of propylene glycol and water solutions. A formula for density, viscosity, and thermal conductivity based on mass/molar fraction and temperature was proposed. These were shown to be in good agreement across the

temperature ranges expected in our experiments.

The PID control scheme is widely used in the bioreactors as well as the TCUs. Chesaru et al [19] describe the efficiency of the scheme in regulating thermal models. In this study, the goal is not developing new control schemes, or even the optimization of current parameters, but rather building the control scheme into the thermal model. Thus, it provides a tool to enable quick and accurate prediction of performance with a selected vessel, TCU, and control parameters.

Cornieles et al [20] produced a survey of different modeling methods and control schemes for temperature regulation of water reservoirs. They found that dual loop control actions showed better performance against perturbations. The systems modeled in this thesis are also dual loop control schemes.

CHAPTER 2

METHODS

2.1 Variable Dependency Tree

While the ultimate outputs of the transient thermal model predict the time for the bioprocess temperature inside the vessel to reach the set point, there are many variables and steps needed to solve them. The variable dependency tree provides a visual schematic of the relationship between the variables. In this case, it has two main branches: the controller model and the heat transfer model. Figures 2 and 3 shows the general flow of information to solve the time to a set point for a given model. Figure 2 is a high-level overview, with Figure 3 showing the detailed dependency flow. Diamonds represent the fixed inputs, circles are iteration temperatures, and squares are equations or functions used.

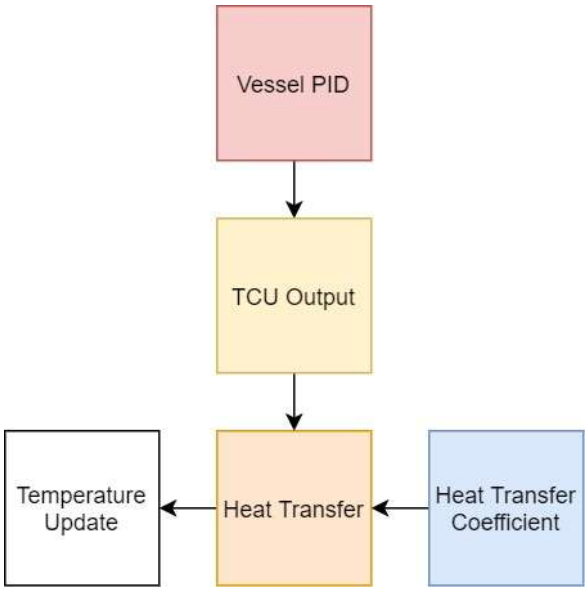

Figure 2: Basic information path for the model.

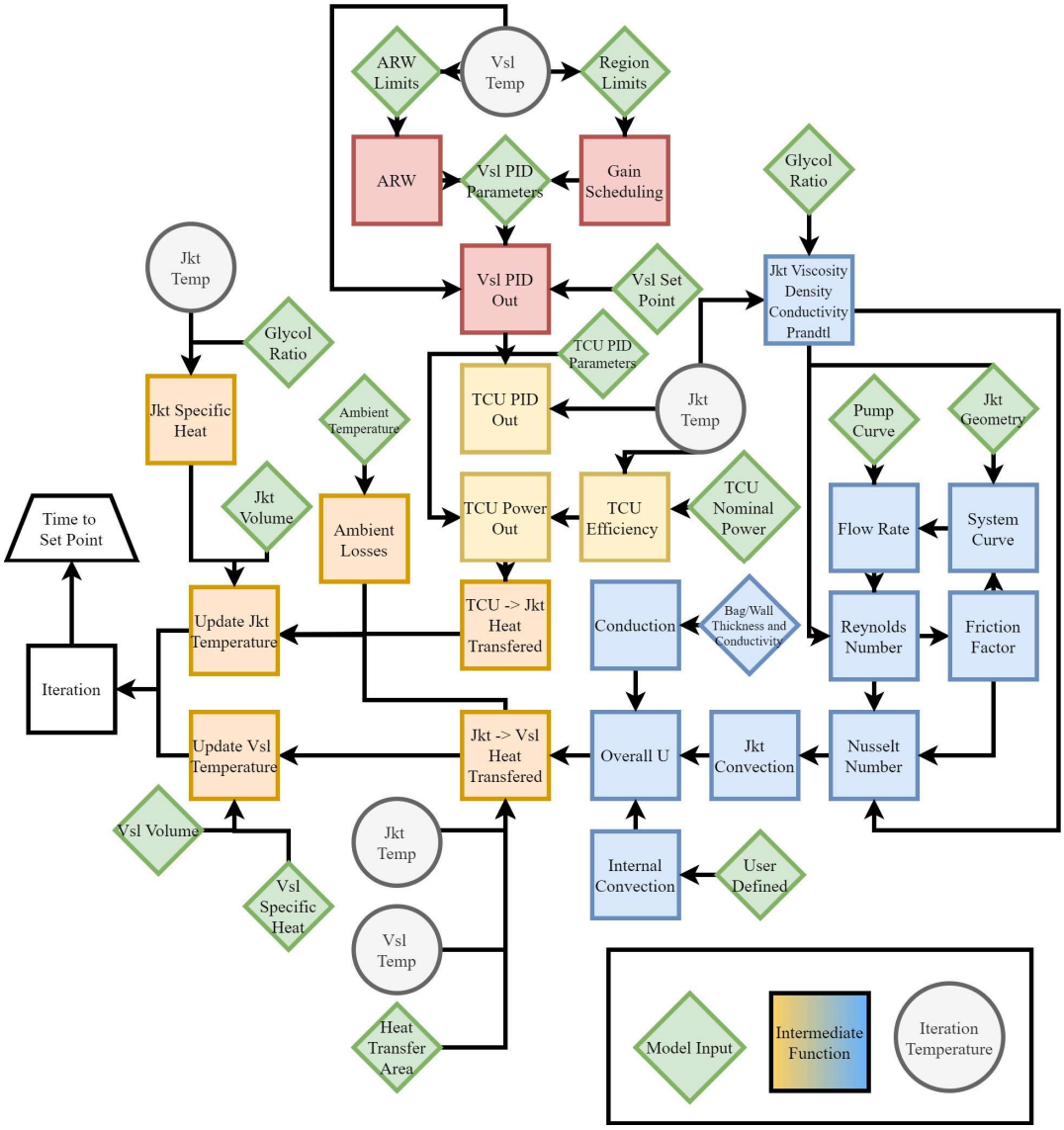

Figure 3: Variable and function dependency flow chart. Diamonds are model inputs, squares equations or functions, and circles are iteration temperature. Note the interdependencies within the jacket flow properties, and the many places thermally dependent material properties appear.

2.2 Controller Modeling

While not in a typical thermal model, the controller model of predicting the controller behavior in this case can be equally as important as predicting heat transfer

characteristics in order to accurately project the bioprocess temperature profiles. During each iteration, the vessel and TCU controllers determine how long and at what amplitude to activate the TCU (heater/chiller). Controller performance also directly regulates the amount of overshoot, ringing, or any hesitance in approaching set point.

The controllers in the modeled systems are a dual loop PID, (proportional, integral, derivative) based control.

2.1.1 PID Controller Background

A PID controller changes its output based on three terms, proportional, integral, and derivative. Each term is based on a different principle with regard to the error between the set point for the process variable and the current value of the process variable.

The proportional term adjusts the controller output by that error multiplied by a proportional gain. A large error results in a large output, while a small error results in a smaller output. Because the controller output decreases as the error decreases, a proportional term alone cannot drive the residual error to zero.

The integral term is based on an integral gain multiplied by the integration of the error over time. This term alone is slower acting than the proportional term but will always drive the residual error to zero in a well-tuned system. The integral term has the potential to cause a negative effect termed reset wind-up. Reset windup, also known as integral windup, is something that can occur if the controller is at saturation for any period of time. Reset windup occurs when the integral term in the PID controller

continues to grow during saturation, and, unless mitigated, would keep the controller at saturation long after it optimally should reduce, resulting in needless overshoots and ringing.

The derivative term acts on the rate at which the error is changing. It acts in the opposite direction of the change to temper, or dampen the controller action, providing stability to the system. Including the derivative term reduces the potential for ringing and overshoot but makes the controller output more sluggish. In practice, the derivative term is based on the process value, rather than the error. This is done to prevent a phenomenon called derivative kick which happens when the set point changes. Derivative kick is induced when the setpoint changes, resulting in a large instantaneous change to the error. That change causes an infinite value for the derivative of the error, and when multiplied by the derivative gain, causes an undesired large change to the controller output. When the set point is not changing, the derivative on process value is mathematically equivalent to the derivative on the error.

Below are three of the most common forms of the PID equation [21]. Each of these, when tuned correctly, will behave the same as the others. Choice is a matter of microcontroller or programming implementation, and ideally should be chosen to match what the user is familiar with.

Dependent Ideal Form: Dependent as the controller gain, K_c is multiplied across all three terms. Ideal in that the integral time and derivative time ($T_i \; and \; T_d$) do not influence the proportional term.

$$CO = CO_{bias} + Kc * e(t) + \frac{Kc}{Ti}\int e(t)dt - Kc * Td\frac{dPV}{dt} \quad (4)$$

Dependent Interacting Form: Dependent as the controller gain, K_c is multiplied across all three terms. Interacting as the integral and derivative times influence the proportional term. This form was initially developed as it is basically a PI and PD controller multiplied together and made analog PID controllers easier to build.

$$CO = CO_{bias} + Kc(1 + \frac{Td}{Ti}) * e(t) + \frac{Kc}{Ti}\int e(t)dt - Kc * Td\frac{dPV}{dt} \quad (5)$$

Independent PID Form: Independent as the three terms have no direct interaction with each other.

$$CO = CO_{bias} + Kp * e(t) + Ki\int e(t)dt - Kd\frac{dPV}{dt} \quad (6)$$

Parameters for each term can be expressed in different forms. The proportional term's parameter is often expressed as K_c, or K_p, controller, or proportional gain respectively. It can also be expressed as a proportional band, which is a description of the how big the error needs to be before the controller output is at saturation. $PB = \frac{100\%\ CO}{K_p}$.

For the integral term, K_i is the integral gain but can also be expressed as Integral

Time: $T_i = \frac{K_c}{K_i}$ or Reset Rate: $T_r = \frac{1}{T_i}$. Integral Time is how long it takes for a unit error to cause a unit accumulation for the controller output from the integral term. Reset Rate is simply the inverse of the Integral Time. Integral time is also known as Reset Time.

The derivative action can likewise be expressed in multiple ways, Derivative Gain: K_d or Derivative Time: $T_d = \frac{K_d}{K_c}$. Note that the relationship between derivative time, derivative gain, and controller gain is different than that of the integral parameters.

2.1.2 Controller Modeling Implementation

The simulated PID control system varies based on which TCU is being modeled, however, the basic structure remains the same. The outside loop is controlled by the computer on the vessel. It takes in the set point temperature for the internal vessel temperature, outputs a setpoint temperature for the jacket temperature, and is closed with a temperature sensor measuring the internal vessel temperature. The internal control loop is within the TCU. The setpoint is the output from the outer control loop, $P_{T_{jkt}}$. The output is the power of the heater/chiller. The loop is closed by a temperature measurement on the jacket fluid. The overall loop is visualized in Figure 4.

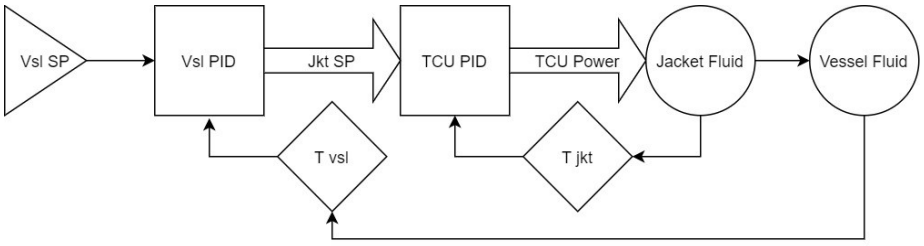

Figure 4: Diagram for the control circuit. A double PID loop, with an external vessel controller and internal TCU controller. The TCU loop is closed on the jacket temperature, while the vessel loop is closed on the vessel temperature

2.1.2.1 Vessel

The external, or vessel control loop is based on a DeltaV brand controller. It is a type of dependent ideal controller. The error is measured as the vessel set point temperature minus the current vessel temperature. The output from this loop is the set point temperature for the jacket.

Gain scheduling is a method of using different control parameters based on the distance of the process value from the set point. Gain scheduling is implemented within the vessel PID system. Three regions are used. Region 1 (R1) is defined as when the vessel temperature is above the gain scheduling upper limit. Region 2 (R2) is between the upper and lower limits, while Region 3 (R3) is below the lower limit.

The Proportional, Integral, and Derivative gains all change based on the current region. The discrete form of the PID equation used for the vessel controller is Eq. 7

$$CO_k = CO_{bias} + K_c\left(e_k + \frac{1}{T_i}\left(e_{sum_k} * dt\right) + \left(\frac{T_d}{dt}\right)\left(e_{k-1} - e_k\right)\right) \quad (7)$$

Anti-reset-windup, or ARW is a system used to mitigate the effects of reset wind up. When activated, the integral time is multiplied by 16. This is shown in Eq. 8. The effect of this change is to greatly reduce the relative weight of the integral term, preventing the influence of accumulated windup from causing excessive overshoot. ARW has upper and lower bounds within the control range. ARW is active when either the controller output has just come off saturation and is outside the ARW limits, or when the ARW is already activated and it is still outside the ARW limits. ARW is deactivated when neither of the two previously stated conditions are met, in which case the PID equation defaults to Eq 7.

$$CO_k = CO_{bias} + K_c\left(e_k + \frac{1}{16 * T_i}\left(e_{sum_k} * dt\right) - \left(\frac{T_d}{dt}\right)\left(e_{k-1} - e_k\right)\right) \quad (8)$$

2.1.2.2 TCU Controller

Each brand of TCU has their own specific type of PID system. Lauda-Brinkmann would not disclose the specific equations used in their TCUs but were willing to describe it as a two-stage controller, with the master loop comprising a full PID system that

outputs a set point for the jacket, and the slave loop being only a proportional controller. The controller uses the parameters: X_p- Proportional Band, T_n-Reset Time, T_V-Derivative time, T_d-Damping Time, and X_{p_f}-Slave Proportional band. Based on the parameters, units, and the limited information Lauda-Brinkmann was willing to provide, a dependent ideal form was chosen to represent the TCU controller.

This was implemented using Eq. 9 and Eq 10.

$$T_{set_{internal}} = T_{vslPID_{out}} + K_p\left(e + \frac{1}{T_n} * e_{sum} * dt + \frac{T_d}{dt}(e_{k-1} - e_k)\right) \quad (9)$$

Where: $e = T_{vslPID_{out}} - T_{jkt}$

$$CO = K_{p_{slave}} * (T_{set_{internal}} - T_{jkt}) \quad (10)$$

The ThermoFlex TCU system uses P-Proportional band (% of 100°C), I-Integral Value (Repeats/minute), and D-Derivative Value (minutes). This is implemented in a velocity form of the equation, shown below. The controller uses different gains based on heating or cooling. This TCU operates as an independent PID form. Based on the given units, the parameters are used in the model as $PB, K_i,$ and K_d. This is shown in Eq. 11.

$$CO = \frac{100}{PB} * e(t) + Ki * e_{sum} * dt - Kd\frac{dPV}{dt} \quad (11)$$

The Sterling TCUs use PB, T_i, T_r, Proportional Band, Reset Time, and Rate Time respectively. This was modeled as a dependent ideal PID form system. Seconds are used as the basic time unit. Eq. 12 is the form used.

$$CO = \frac{100}{PB} * \left(e(t) + \frac{1}{T_i} * e_{sum} * dt - T_d\frac{dPV}{dt}\right) \quad (12)$$

2.2 Heat Transfer Modeling

For the bioreactors, heat transfer takes place between the jacket and vessel. There are three main thermal resistances that need to be taken into account in the heat transfer model: 1) the convection resistance inside the jacket fluid; 2) the conduction resistance through the vessel polymer wall and jacket metal wall separating the jacket fluid from the bioprocessing fluids inside the vessel, as well as a conduction resistance based on the estimated airgap between the polymer and metal walls; 3) the convection inside the vessel. Figure 5 shows a cross-sectional view of a bioreactor - TCU system.

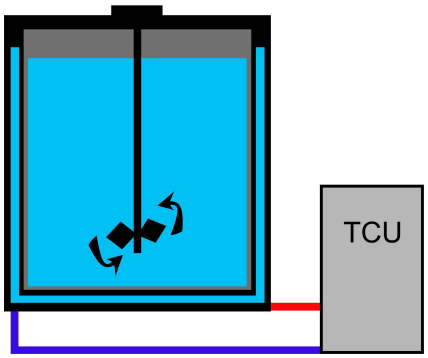

Figure 5: Vessel and TCU diagram.

There are two types of TCU systems within the scope of this project, open and closed. Closed systems have a heater and chiller built into the TCU. Open systems have an internal heater, but no chiller. Cooling potential is created via a pressurized house (external) chilled liquid feed. Modeling open systems has the added complexity of predicting the mass and energy transfer in and out of the system.

We modeled the convective heat transfer coefficient within the jacket closely. Given many physical properties of the working fluid are strongly dependent upon temperature, the value of the convective heat transfer coefficient was sensitive to working temperature. As a result, models of material properties such as viscosity, density, and Prandtl number were implemented. The Nusselt number correlations require Reynolds number, Prandtl number, and friction factor. Pump performance (i.e. pump curve) and fluid system interaction modeling is necessary to find these parameters.

2.2.1 Vessel Side Convection

Previous literature suggests that the convection within a well-mixed vessel is not a limiting factor for the overall heat transfer coefficient in these type of systems [17]. Initial estimations were generated via the Eq. 1 referenced in [3]. A basic sensitivity analysis was also conducted within the model. It was found that adjusting the internal convective coefficient from half the initial estimate to double the initial estimate only impacted the model's time to 90% of set point by less than 0.5%. With the intention of keeping the model conservative in its estimate of heat transfer capacity, a convective coefficient of 25% of the correlation was used.

$$Nu = CRe^{0.67}Pr^{0.33} \tag{13}$$

Where $Re = \frac{\rho N D^2}{\mu}, N = Impeller\frac{Revolutions}{second}, D = Impeller\ Diameter.$

Yamamoto [22] proposed Eq. 14 as a method of examining the Reynolds number at the heat transfer surface rather than at the impeller. With the bioreactor and impeller geometry used in our experiments, this produced a Reynolds number 50% larger.

$$V = 0.802\pi ND\left(\frac{D}{D_T}\right)^{0.80}\left(\frac{\omega}{D_T}\right)^{0.186}\left(n_p\right)^{0.162} \tag{14}$$

Table 1 shows increasing vessel side convective coefficients with their corresponding mean U values and time to 95% of set point. Note that as the convective

coefficient approaches ~10,000, the value predicted by Equation 1, the change to both U values and time to set point becomes minimal.

Table 1: Basic sensitivity analysis on the internal convective coefficient h on the overall heat transfer coefficient and time to set point. Simulated on a 18kW TCU with a 2000L bioreactor.

Vessel h	U	T95
500	259	4.173
1000	347	4.147
2000	420	4.123
4000	469	4.118
8000	498	4.115

2.2.2 Jacket Side Convection

We used the Nusselt Correlation put forward by Taler [11]:

$$Nu = Nu_{m,T}(Re = 2300) + \frac{\frac{\xi}{8}(Re - 2300)Pr^{\,1.008}}{1.08 + 12.39\sqrt{\left(\frac{\xi}{8}\right)}\,(Pr^{2/3} - 1\,)}\,x\,\left(\frac{Pr}{Pr_{\,w}}\right)^{1.1} \qquad (15)$$

Where $\mathrm{Nu_{m,T}(Re = 2300)}$ is the mean Nusselt number for a constant wall temperature, laminar flow case. This equation is applicable for $2300 < Re < 10^6$ and when $\frac{d_w}{L} < 1$. From Kays and Crawford [23] we find a correlation for the Nusselt number within a laminar, high aspect ratio, insulated on one side, constant wall temperature, laminar duct:

$$Nu_{m,T}(Re = 2300) = 4.86 \quad (16)$$

Combining Eqs. 15 and 16, we get a smooth function that approximates the transition from laminar to turbulent flow.

The jacket geometry within the vessels tested has regular interruptions to the flow caused by dimples in the jacket, which is expected to increase fluid mixing. The schematic of the jacket geometry is shown in Figure 6.

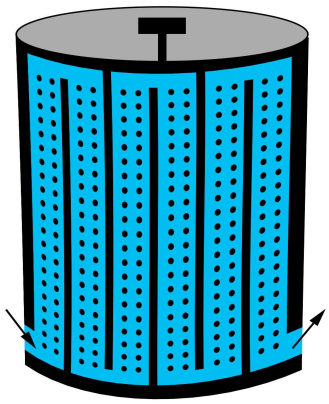

Figure 6: Example jacket geometry. Note the serpentine path with regular dimples interrupting the flow.

Due to this increase in mixing, we used a higher value for the laminar Nusselt number. We used the correlation for a high aspect ratio laminar duct with constant temperature and heat transfer from both sides. [23]

$$Nu_{m,T}(Re = 2300) = 7.54 \qquad (17)$$

As the Nusselt number is dependent on Reynolds number, Prandtl number, and friction factor, ways to solve each are required. Our model for Prandtl number simply is a function of glycol percentage and jacket fluid temperature. We use the correlations put forward by Sun and Teja [18] for viscosity and thermal conductivity.

$$\mu_{T,\%Glycol} = \frac{e^{w_1\eta_1 + w_2\eta_2 + (\eta_1 - \eta_2) * w_1 * w_2 * \left(1.523 - 5.0007 * w_1 + 9.818E^{-4} * T + 3.2453 * w_1^2\right)}}{1000} \qquad (18)$$

$$where: w_1 = Glycol\ Fraction, w_2 = 1 - w_1,$$
$$\eta_1 = 0.00021319 * T^2 - 0.06436 * T + 5.24,$$
$$\eta_2 = -3.358 + \frac{590.98}{T + 137.26}, \qquad T = Temperature\ (C)$$

$$k = w_1 * \lambda_1 + w_2 * \lambda_2 + (\lambda_1 - \lambda_2) * w_1 * w_2 * (C_4 + C_5 * w_1 + C_6 * T) \qquad (19)$$

$$where: \lambda_1 = k_{water,T} = -8.354E^{-6} * T_{kelvin}^2 + 6.53E^{-3} T_{kelvin} - 0.5981,$$
$$\lambda_2 = 0.5709 + 0.167E^{-2} * T - 0.609E^{-5} * T^2,$$
$$C_4 = 0.3622, \qquad C_5 = 9.034E^{-2}, \qquad C_6 = -2.09E^{-4}$$

Flow Modeling

Reynolds number and friction factor required a more holistic treatment. The interaction between pump and system curves needed to be solved. Pump curves were used where provided, and where not, a prediction of pump curves was made based on provided brake horsepower of the pump. An iterative function was written to solve for the intersection of the pump and system curve, then resolve for friction factor and Reynolds number with the new flow values.

The Swamee-Jain [24] equation was used to solve for friction factor. This is shown in Eq 20.

$$\xi = \frac{0.25}{\left[log\left(\frac{\frac{\epsilon}{D_h}}{3.7} + \frac{5.74}{Re^{0.9}}\right)\right]^2} \tag{20}$$

The square root of the cross-sectional flow area was used as the length scale for Reynolds number as put forward by Duan [13,14]. Viscosity of the jacket fluid varies greatly across temperature range and has a non-linear effect on the Reynolds number. It both directly changes the Reynolds number as formulated and has a strong influence on the mean flow velocity via the friction factor and system curve. We have anticipated the possibility of heating to turbulent or cooling to laminar within a single temperature ramp.

Open Systems

No additional treatment is needed for closed jacket-TCU systems. For open jacket-TCU systems the mass flow between the house feed and jacket loop must be calculated during cooling. Figure 7 shows the flow path configuration while cooling in detail. It is designed to not dead-head, or abruptly stop, the flow through the house loop to avoid pressure shocks to the system.

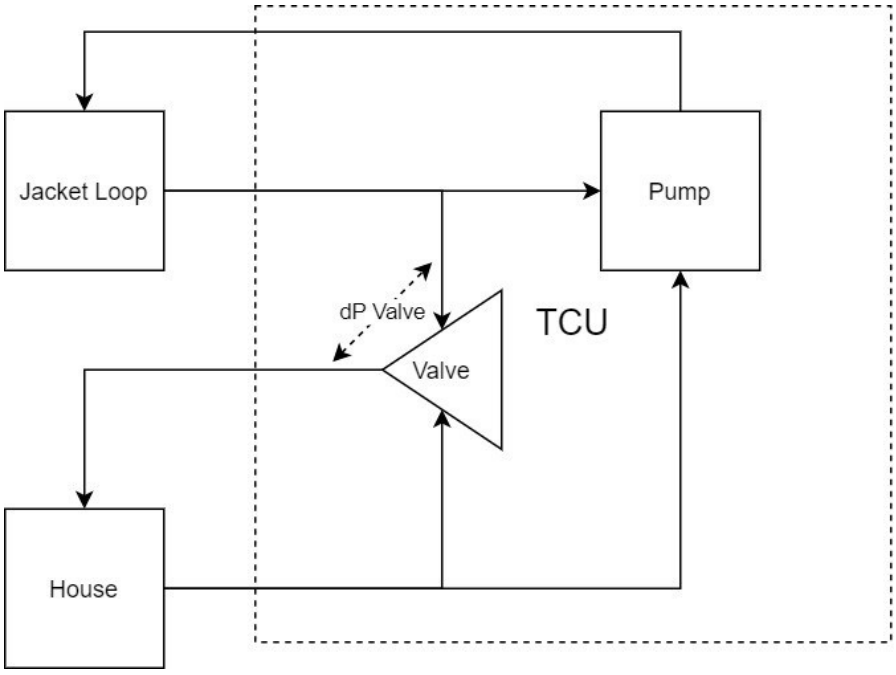

Figure 7: Flow path for open TCU systems. The pressure differential across the valve is the motivator for the mass exchange

Known variables include the flow factor for the three-way mixing valve and the pressure differential across the house feed. To develop a method of solving the flow

through the valve (a portion of the cooling fluid flowing through the jacket), the following equations were used.

$$\Delta P_{Pump} = \Delta P_{Jacket\ loop} \tag{21}$$

$$\Delta P_{House} = \Delta P_{valve} \tag{22}$$

The top equation assumes that the pressure differential from the pump is the sole driver of the flow through the jacket loop; the second equation indicates that the house pressure differential is the sole driver of the flow through the three-way mixing valve.

From the definition of a flow factor:

$$Q_{valve} = k_V \sqrt{\frac{\Delta P_{valve}}{SG}} \tag{23}$$

The base units for the flow factor, k_v are $\frac{m^3}{hr\sqrt{bar}}$. The flow factor is adjusted based on current viscosity according to Womack Machine [25]:

$$K_{v_{effective}} = \frac{k_{v_{nominal}}}{0.8813\ \mu^* + 0.1792} \tag{24}$$

where μ^* is the fluid viscosity in centistokes.

2.2.3 Conduction Resistances

Modeling the thermal resistance of the conduction between the jacket fluid and vessel fluid was done by Fourier's Law. The bag containing the vessel fluid has three types of polymers, and the jacket walls are made of stainless steel. The bags are designed to be well seated against the jacket walls, with the intention of minimizing potential air gaps.

Table 2: List of materials, thicknesses, and thermal conductivities of the jacket-vessel membrane

Material	Polyethylene	Polyester	EVOH	SS 316
Thickness (mm)	0.310	0.020	0.025	9.5
k (W/mK)	0.33	0.28	0.34	15

The conduction path is shown in Figure 8. For the bags used in the experiments, the polymer lining composition is as listed in Table 2.

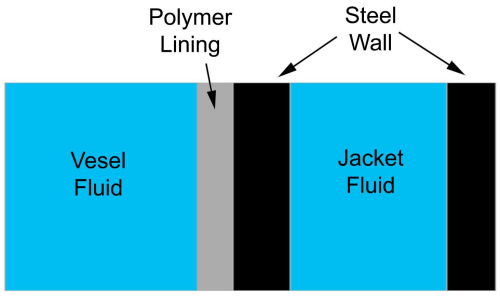

Figure 8: Section view of the conduction path

2.3 Iteration Update

In the model, during each iteration first the Vessel Controller function is called, followed by the TCU controller. Next the overall heat transfer coefficient, along with fluid and flow properties are calculated. Calculating the heat transferred from the TCU to the jacket and from the jacket to the vessel and updating the bulk vessel and jacket temperatures completes the iteration.

To solve the Heat output from the TCU, the controller output is multiplied by the nominal heating/cooling capacity and by an efficiency factor η, a function of temperature. Open TCU systems when cooling calculate the apparent output power by Eq. 25.

$$Q_{TCU\,Jkt} = -\left(\forall_{open}\rho\right) * \frac{PID_{out}}{100\%} * C_{p\,jkt}\left(T_{hou} \quad -\left(T_{jkt} - \left(\frac{Q_{jkt_{vsl}}}{m_{jkt} * Cp_{jkt}}\right)\right)\right) \tag{25}$$

Closed TCU systems and open TCU systems while heating use Eq. 26 to determine the output power from the TCU.

$$Q_{TCU\,Jkt} = PID_{out} * P_{TCU_{Nominal}} * \eta_{TCU_{T_{jkt}}} \tag{26}$$

2.4 Experimental Validation

Experiments were conducted to validate the model. These experiments were chosen to test a wide variety of TCU/vessel combinations. Due to time and availability constraints, physical experiments were limited, both in repetitions and scope. The used combinations are listed in Table 3. Each testing set followed the following vessel temperature sequence of 5-37-20-5-20-37-5 ℃, which helps offset the limited variety of TCU/Vessel combinations. The collected data sets between the heating and cooling processes were used to validate the transient thermal model.

Table 3: List of Experiments conducted

Test	*Vessel*	*TCU*
1	*2000L SUB*	*18k Sterling*
2	*50L SUB*	*VC1200 Lauda*
3	*50L Bioreactor*	*VC1200 Lauda*
4	*500L SUM*	*VC2000 Lauda*
5	*500L SUB*	*TF5000 ThermoFlex*
6	*300L SUF*	*9k Sterling*
7	*2000L ImPulse*	*9k Sterling*

Each vessel had temperature sensors inside the vessel and on the inlet and outlet of the vessel jacket. Pressure measurements were also taken on the inlet and outlet of the vessel jacket. For tests conducted using open system TCUs, additional temperature and pressure sensors were placed on the inlets and outlets to the house feeds. These locations are visualized in Figure 9.

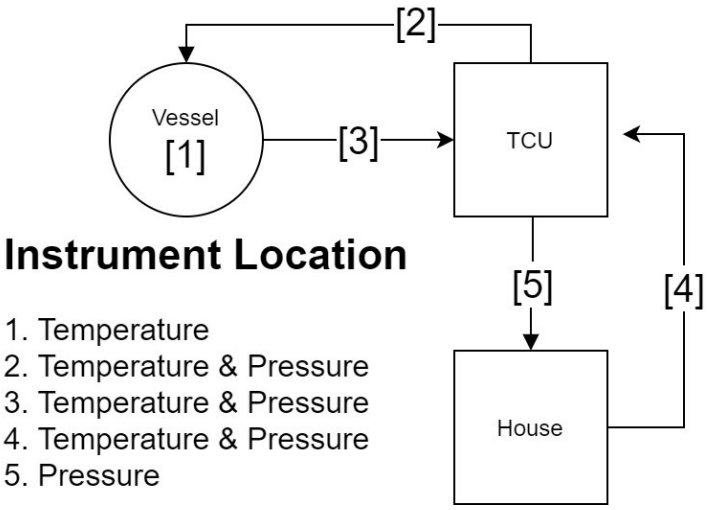

Figure 9: Locations of instrumentation of experimentation. Closed systems instrumentation is identical, but with [House], [4], and [5] removed.

Temperatures and pressures were measured directly to calculate log mean temperature difference, heat transfer rates, pump flow rates and pressure drops. As a critical parameter, the overall heat transfer coefficients of the bioreactors were calculated according to $U = \frac{Q}{\Delta T_{lm} A}$, where the heat transfer rate Q is calculated in two different methods:

$$Q = \frac{dT_{vsl}}{dt} * V_{vsl} * \rho_{vsl} * cp_{vsl} \quad (27)$$

$$Q = \left(T_{jkt_{in}} - T_{jkt_{out}}\right)\rho_{jkt} * cp_{jkt} * \forall_{jkt} \quad (28)$$

$$Where\ Q = Heat\ flow; T = temperature;\ t = time; V = volume;\ cp = specific\ heat; \rho = density;\ \forall = Volumetric\ flow\ rate$$

There are pros and cons associated with each method. The first uses a derivative of real data, and results in strong noise without smoothing. A gaussian smoothing function was employed to reduce the artifacts associated with this process. The second method works well for steady state but has the potential to have larger errors during transient states. Additional downsides include the uncertainty from the jacket flow rate and needing to know heat loss to ambient through the jacket walls. Using Eq. 27 for Q, the complete equation for U is:

$$U = \frac{\frac{dT_{vsl}}{dt} * m_{vsl} * c_{p_{vsl}}}{A\left(\frac{T_{jkt_{in}} - T_{jki_{out}}}{log\left(\frac{T_{jkt_{in}} - T_{vsl}}{T_{jkt_{out}} - T_{vsl}}\right)}\right)} \quad (29)$$

To solve for bias uncertainty, partial derivatives were calculated. They are shown in Equations 30-34.

$$\frac{\partial U}{\partial T_{vsl}} = -\frac{m * c_p * \frac{dT_{vsl}}{dt}}{A\left(T_{vsl} - T_{jkt_{inlet}}\right)\left(T_{jkt_{out}} - T_{vsl}\right)} \tag{30}$$

$$\frac{\partial U}{\partial T_{jkt_{out}}} = -\frac{m * c_p * \frac{dT_{vsl}}{dt}\left(\left(T_{vsl} - T_{jkt_{out}}\right) * log\left(\frac{T_{jkt_{in}} - T_{vsl}}{T_{jkt_{out}} - T_{vsl}}\right) + T_{jkt_{in}} - T_{jkt_{out}}\right)}{A * \left(T_{jkt_{in}} - T_{jkt_{out}}\right)^2\left(T_{jkt_{out}} - T_{vsl}\right)} \tag{31}$$

$$\frac{\partial U}{\partial T_{jkt_{in}}} = -\frac{m * c_p * \frac{dT_{vsl}}{dt} * \left(\left(T_{jkt_{in}} - T_{vsl}\right) * log\left(\frac{T_{jkt_{in}} - T_{vsl}}{T_{jkt_{out}} - T_{vsl}}\right) - T_{jkt_{in}} + T_{jkt_{out}}\right)}{A\left(T_{jkt_{in}} - T_{jk\ _{out}}\right)^2\left(T_{jkt_{in}} - T_{vsl}\right)} \tag{32}$$

$$\frac{\partial U}{\partial A} = -\frac{m * c_p * \frac{dT_{vsl}}{dt} * log\left(\frac{T_{jkt_{in}} - T_{vsl}}{T_{jk\ out} - T_{vsl}}\right)}{A^2\left(T_{jkt_{in}} - T_{jkt_{out}}\right)} \quad (33)$$

$$\frac{\partial U}{\partial m} = -\frac{c_p * \frac{dT_{vsl}}{dt} * log\left(\frac{T_{jkt_{in}} - T_{vsl}}{T_{jkt_{out}} - T_{vsl}}\right)}{A\left(T_{jkt_{in}} - T_{jkt_{out}}\right)} \quad (34)$$

Instrumentation used includes Omega PX119 series pressure transducers with a 0-100 psi operation range and accuracy of +/- 0.5 psi [26]. Temperature was measured with HOBO S-TMB-M0 series Sensors with resolution of 0.03°C and +/- 0.2°C of accuracy. [27].

The bias error was solved via Eq. 35.

$$s_{bias} = \left(S_{T_{jkt_{in}}}^2 * \left(\frac{\partial U}{\partial T_{jkt_{in}}}\right)^2 + S_{T_{jk\ out}}^2 * \left(\frac{\partial U}{\partial T_{jkt_{out}}}\right)^2 + S_{T_{vsl}}^2 * \left(\frac{\partial U}{\partial T_{vsl}}\right)^2 + S_A^2 * \left(\frac{\partial U}{\partial A}\right)^2 + S_m^2 * \left(\frac{\partial U}{\partial m}\right)^2 + S_{c_p}^2 * \left(\frac{\partial U}{\partial c_p}\right)^2\right)^{1/2} \quad (35)$$

Uncertainties were solved for every 5°C increment. Random standard deviation was solved within each bin, and a Student's T 90% double sided test was applied. The

overall uncertainty was calculated from Eq. 36.

$$s_U = \sqrt{s_{bias}^2 + s_{random}^2} \tag{36}$$

CHAPTER 3

RESULTS AND DISCUSSION

3.1 U Value Correlations

The U values (overall heat transfer coefficients) were calculated according to Eq. 29 from the experiment. They were binned in 5°C increments with the average values plotted with uncertainty bars of two standard deviations as shown from Figs. 10 to 16. The direct Model output U values from the corresponding tests are overlaid for comparison.

Examining the U values for the different runs gives a good picture of how well the model is performing. Figures 10-16 show the relationship between jacket temperature and the heat transfer coefficient. Overall, the model captures the trend of increasing U with increasing jacket temperatures observed in the experiments, however, there are still significant errors.

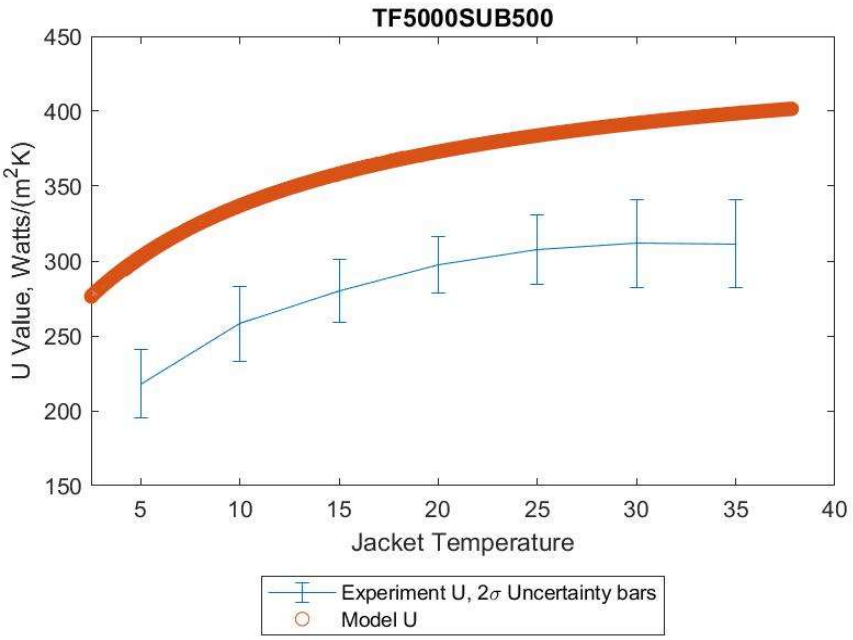

Figure 10: U values for ThermoFlex5000 TCU w/ 500L

Figure 11 shows the U values for the Sterling 18kW TCU. This TCU had the highest-powered pump of all the tests. The resulting higher flow rate and Reynolds number in the jacket raised the U value.

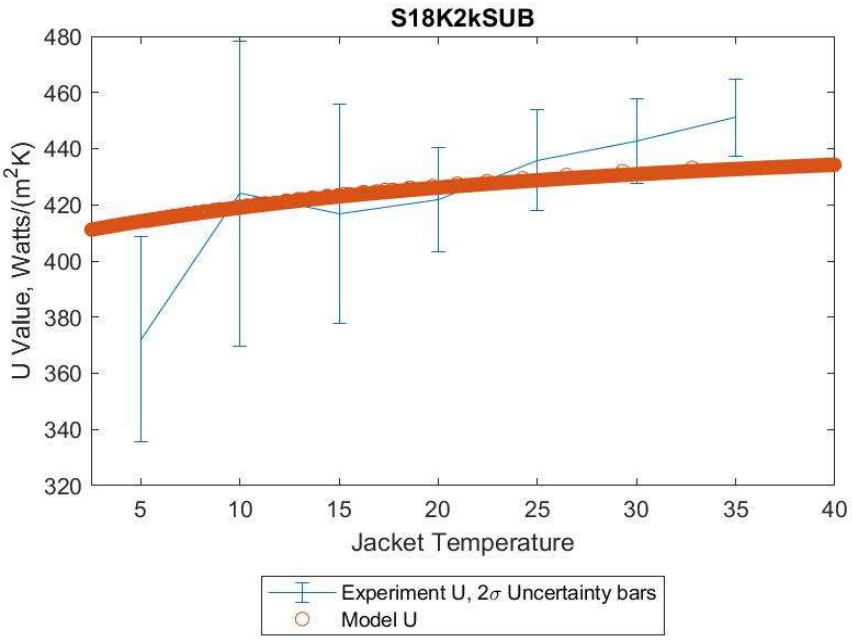

Figure 11: U values for Sterling 18kW TCU w/ 2000L SUB

The 2kL ImPulse vessel used in Figure 12 showed a significantly lower U value from the experiment than what the model predicted. A few reasons are posited. The mixing method inside the vessel is very different, with a different flow pattern. The jacket geometry is also different, with a different width of the serpentine path and number of dimples. This could decrease mean flow velocity, Reynolds number, and jacket side convective coefficient.

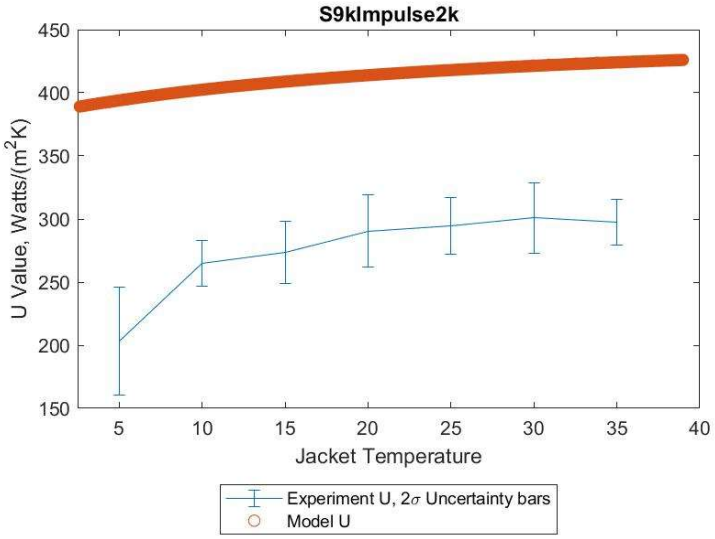

Figure 12: U values for Sterling 9kW TCU w/ 2000L ImPulse

We do not know what caused the different trend with the 300SUF in the 15-20 °C range. Unfortunately, due to time and resource constraints, retesting was not an option.

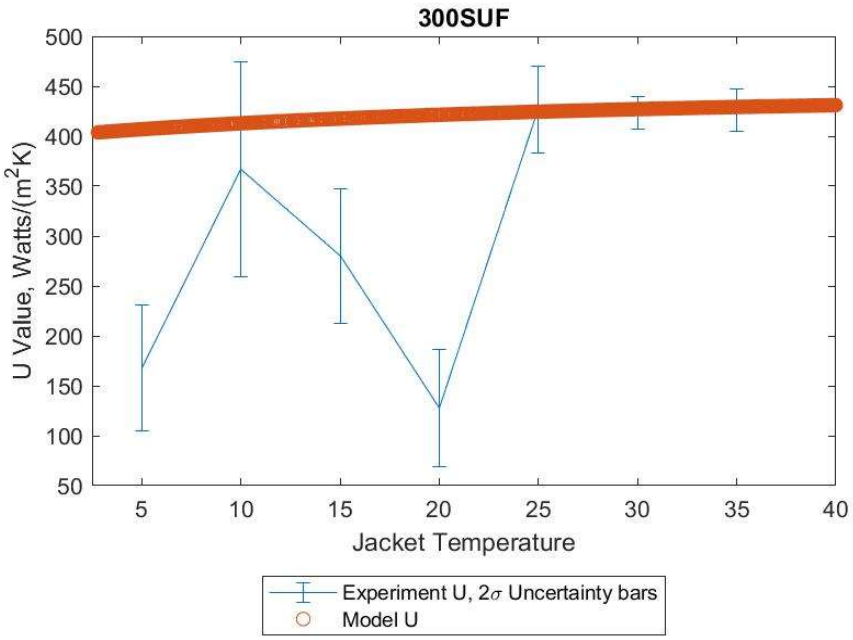

Figure 13: U values for Sterling 9kW TCU w/ 300 SUF

In Figures 14-16 we see a trend of the model underestimating the U value at lower temperature ranges. The model is predicting fully laminar flow when the U value levels off on the low end. It is likely that the dimples inside the jacket trip turbulence earlier and increase the convective coefficient.

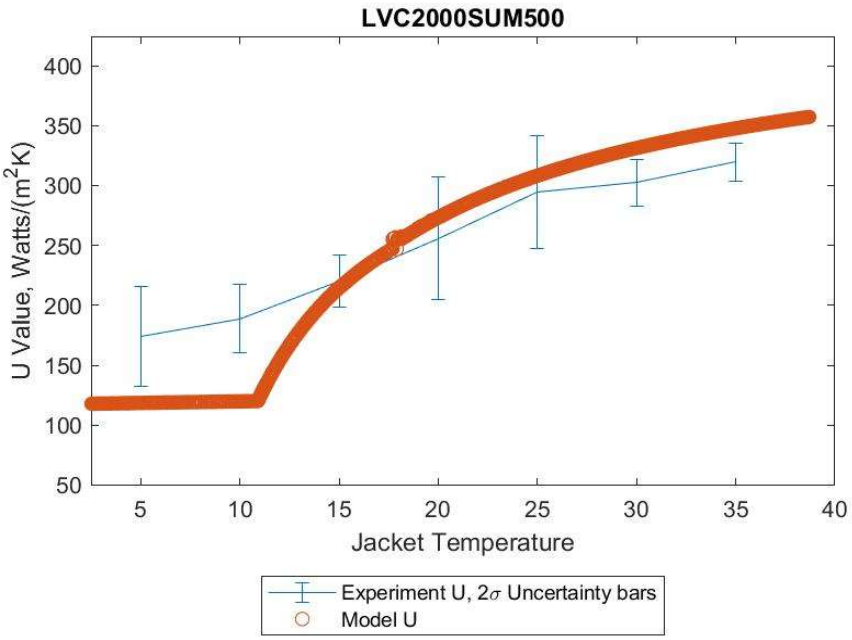

Figure 14: U values for Lauda VC2000 TCU w/ 500L SUM

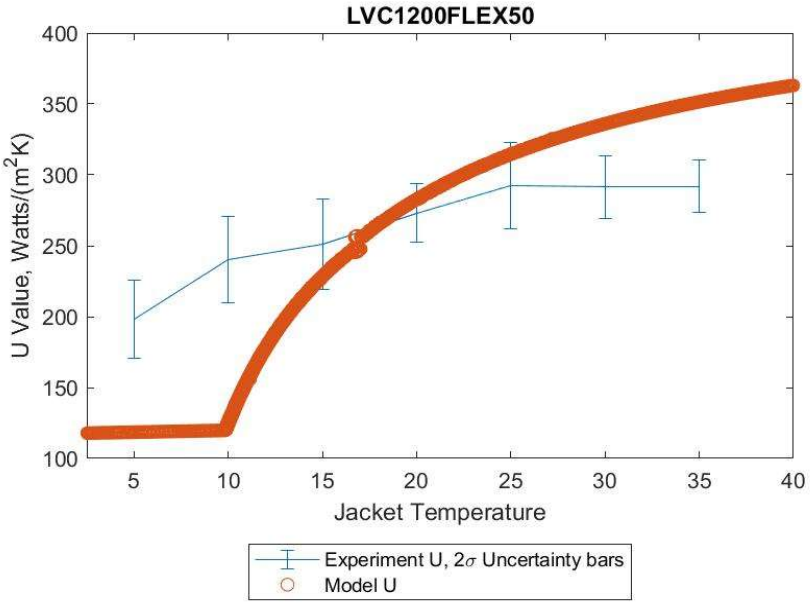

Figure 15: U values for Lauda VC1200 TCU w/ 50L FLEX

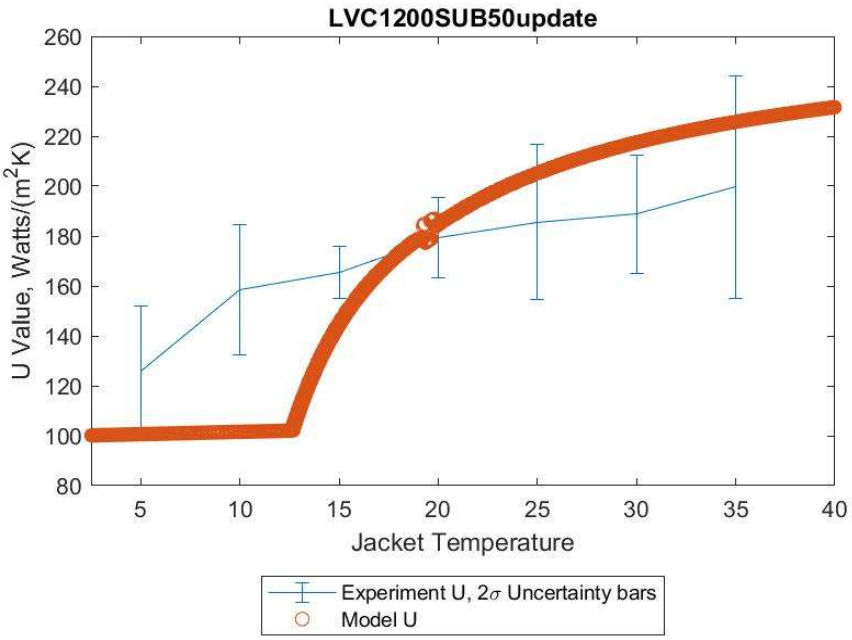

Figure 16: U values for Lauda VC1200 TCU w/ 50L SUB

3.2 Temperature Ramps

Overlaying the temperature ramps between actual experiment data and model results gives both a qualitative and quantitative sense of how well the model matched the corresponding experiment. A few characteristic results are shown in figures 17-20, with the whole collection of 42 tests reserved for Appendix A.

Dashed lines are experiment results, solid are model predictions. The experiment jacket inlet and outlet temperatures are shown. Log mean temperature differences are also plotted to show how well the model matched.

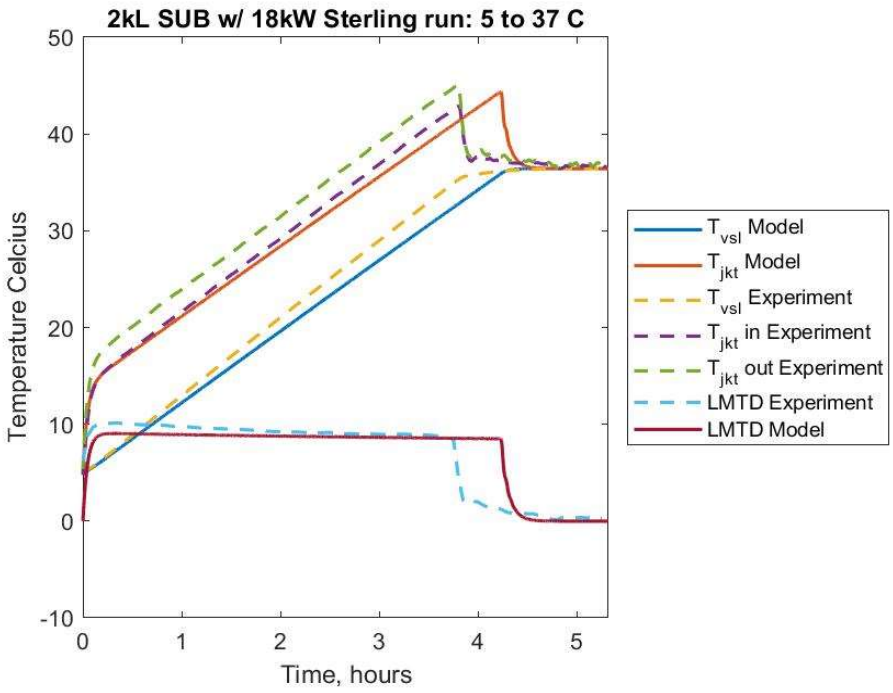

Figure 17: Temperature ramp for 2000L SUB and 18kW Sterling 5°C-37°C

Figure 18 shows an example of over estimation of the cooling potential of a TCU. Note that even though the log mean temperature difference was close through the run, the U value, as shown in Figure 10, was overestimated in the model.

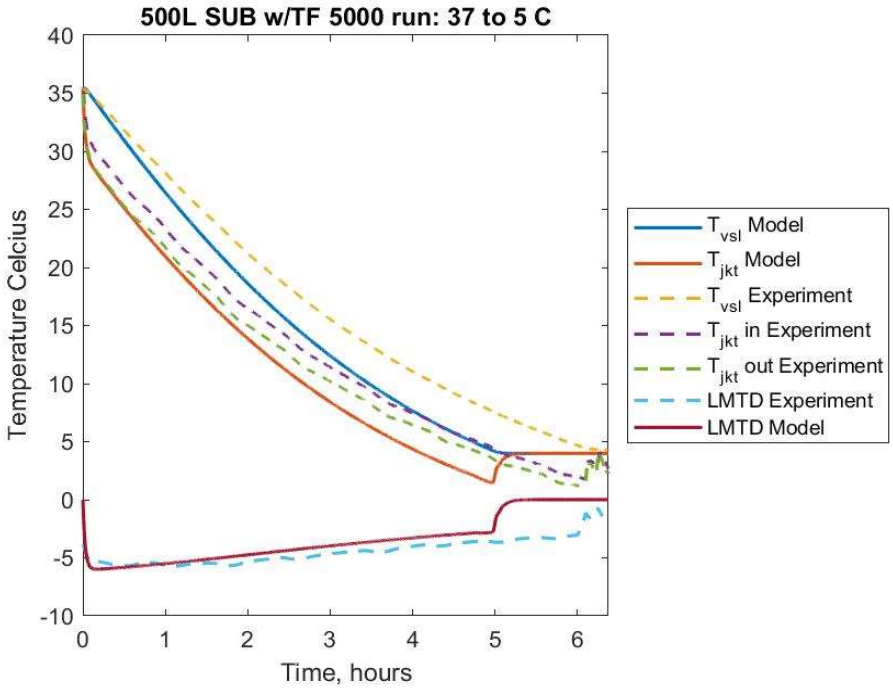

Figure 18: Temperature ramp for 500L SUB and TF5000 TCU 37°C-5°C

Figure 19 shows an example of overshoot on the experiment, but not on the model. This experimental set up had the highest ratio of thermal mass in the jacket and TCU reservoir to the thermal mass inside the vessel. In other words, it was most sensitive to poor controller parameters. Overshoot modeling is further discussed following figure 25.

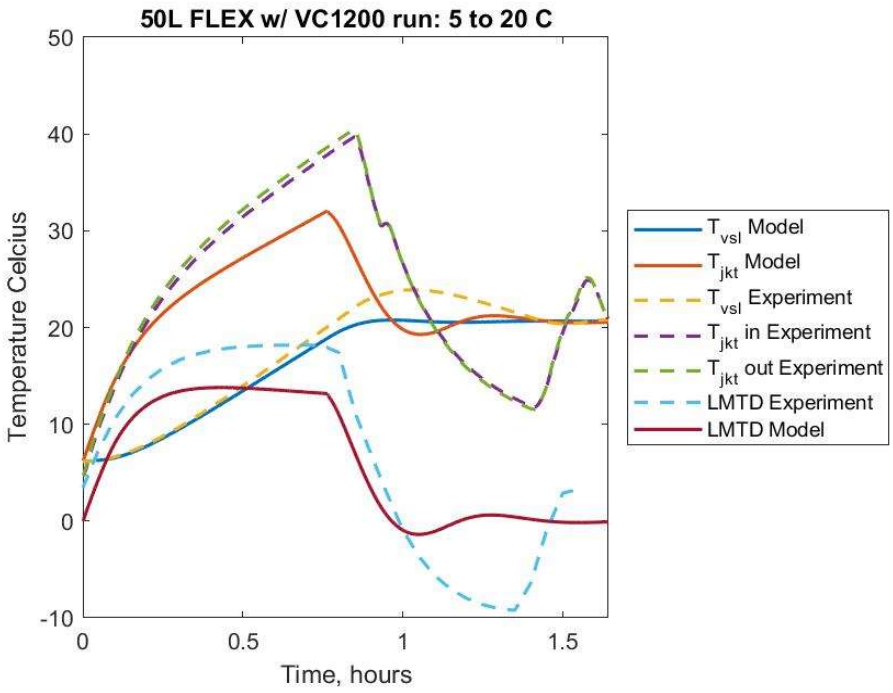

Figure 19: Temperature ramp for 50L SUB and VC1200 TCU 5°C-20°C

Figure 20 shows a problem that ends up occurring in the model. In some cases, the model U value dips lower than the experiment, causing a sudden shift in the log mean temperature difference. Comparing the U value in the 10°C-15°C range on Figure 14 with the resulting temperatures on Figure 20 clearly shows this effect.

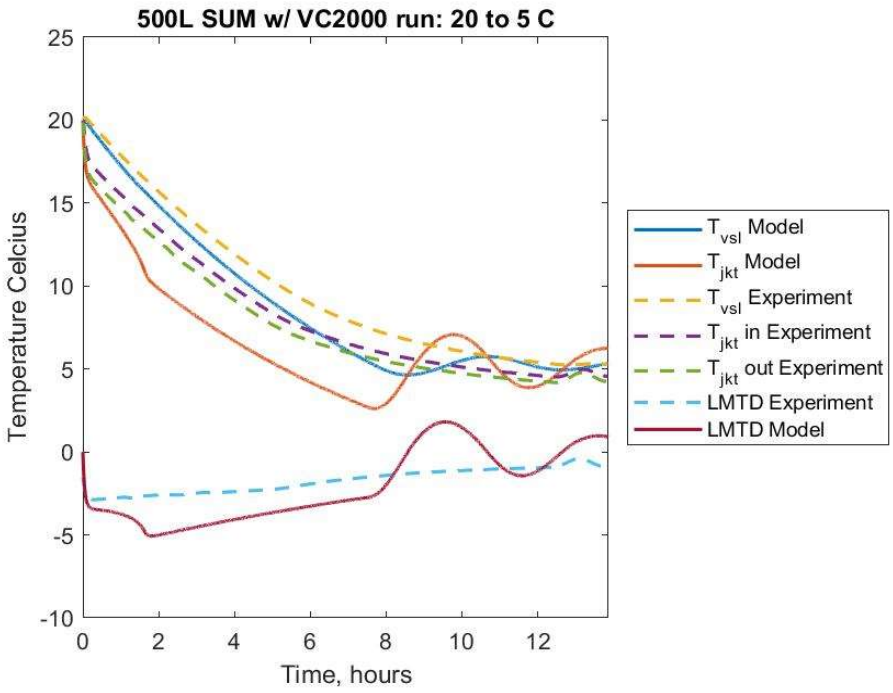

Figure 20: Temperature ramp for 500 SUM and VC2000 TCU 20°C-5°C

3.3 Group Analysis

Graphing information from all runs gives insight into the efficacy and weaknesses of the model and reveals some general trends. The experiment and models U values are the median U values are median values from the corresponding runs, filtered to only include the start time to set point. The Percent Error is defined as:

$$e\% = 100\% * \frac{Model - Experiment}{Experiment} \qquad (37)$$

Figure 21 is a U parity plot that shows that the model has a general tendency to overestimate U values.

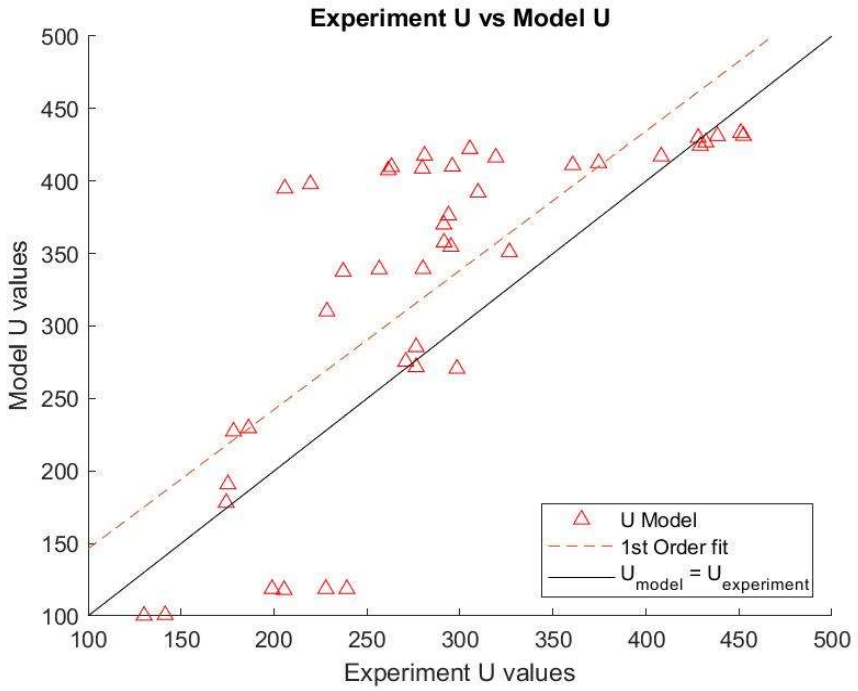

Figure 21: Median U of each ramp, model vs experiment. Solid line represents $U_{Model} = U_{Experiment}$

Figure 22 shows that the model did an excellent job predicting the time from start to 50% of set point. Time to 90% and 95% shown in Figures 23-24 trail behind in performance.

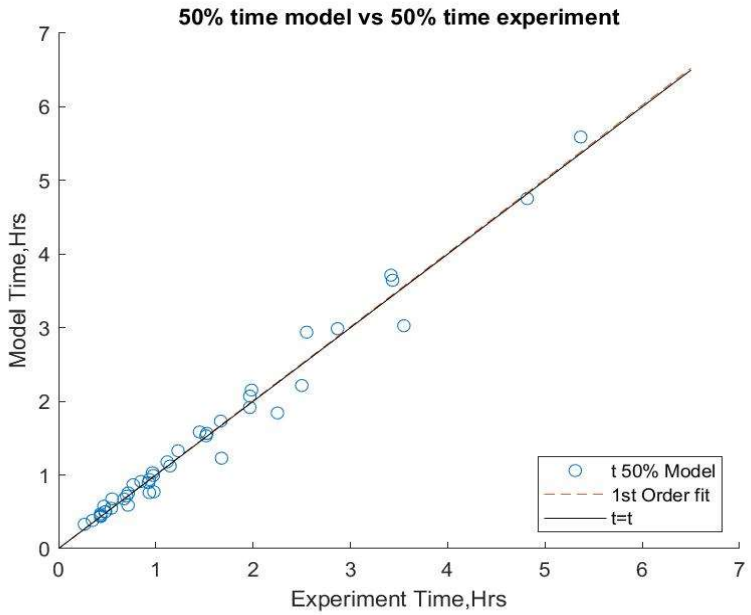

Figure 22: Time to 50% of set point, model vs experiment. Solid line represents $t_{50_{Model}} = t_{50_{Experiment}}$

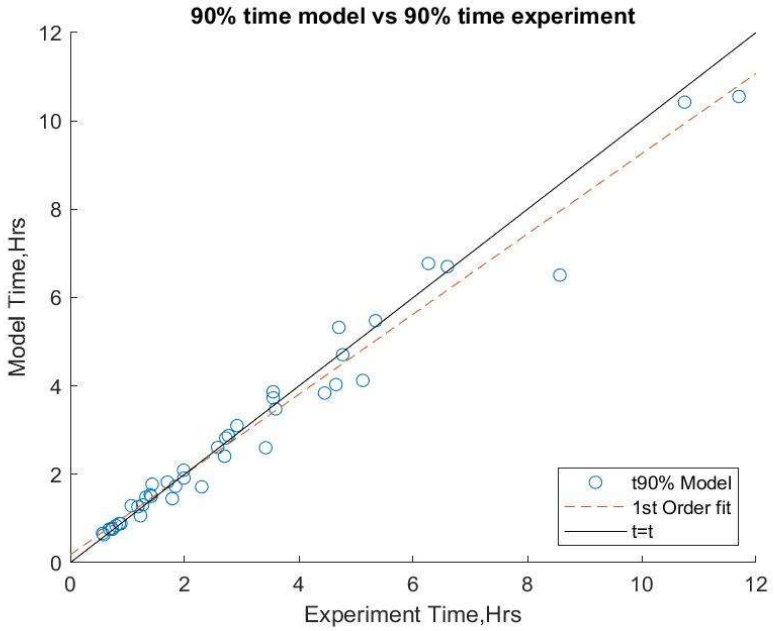

Figure 23: Time to 90% of set point, model vs experiment. Solid line represents $t_{90_{Model}} = t_{90_{Experiment}}$

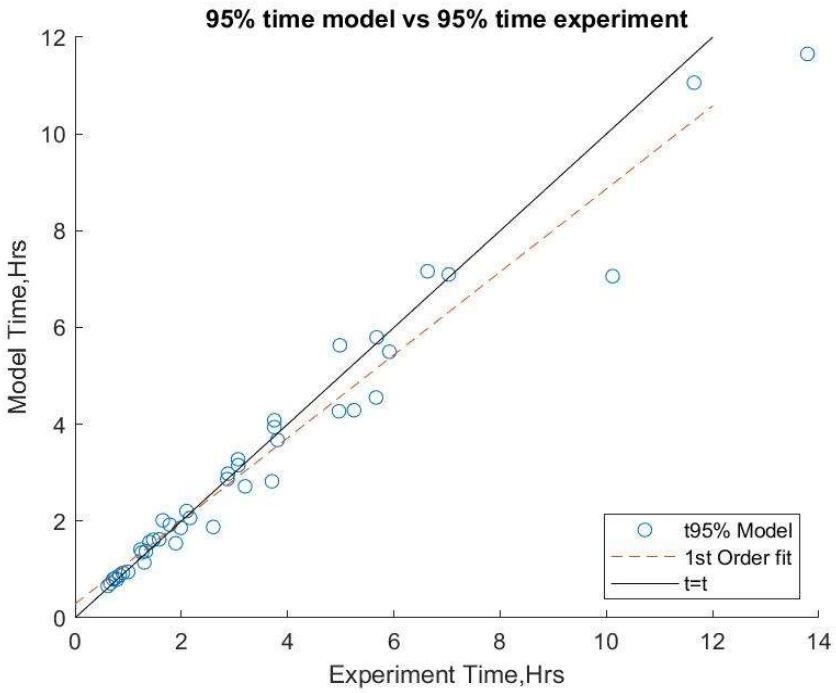

Figure 24: Time to 95% of set point, model vs experiment. Solid line represents $t_{95_{Model}} = t_{95_{Experiment}}$

Figure 25 shows that the model does a better job predicting heating times than cooling times. All heating ramps were within +15%/-5% of the corresponding experiment. Cooling times were less accurate, with a range of +25%/-28%.Potential reasons cooling performance is worse include inaccuracies in the mass flow rate for open systems, inaccuracies in the cooling potential of the TCU across different temperature ranges, the more abrupt shift to laminar flow in the model with the corresponding shift in U value, or viscosity effects on pump curves.

A 90% Student's T double sided test on the heating ramps suggests a model uncertainty range of negative 3.5% to 14.7% for the 90% to set point times. Applying the same test to cooling results in negative 35.3% to 25.4% uncertainty range.

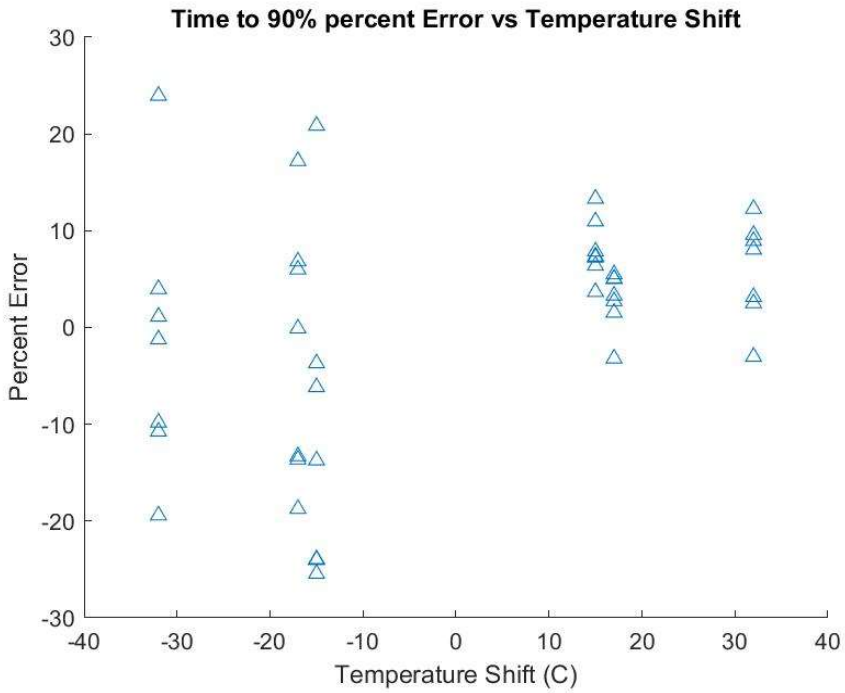

Figure 25: Percent error of time to 95% of set point vs temperature shift

Overshoot and time to 95% of set point are the two methods we used to determine the efficacy of the controller simulation. Overshoot, as shown in Figure 26 was not closely matched. There was very little correlation between the experiment overshoot and model overshoot. The model generally trended toward minimal overshoot. We propose several reasons why this deficiency occurs. First, it is possible that the model does not use the exact PID equations in use within the TCUs. Also, the exact control loop update period was not disclosed by the TCU manufactures, and if it is significantly longer than the timestep used in the model, it could cause overshoot. Additionally, there are delays in the control system that are not well integrated into the model, such as ramp times turning on the heater or chiller, or lag times as the jacket fluid completes its loop.

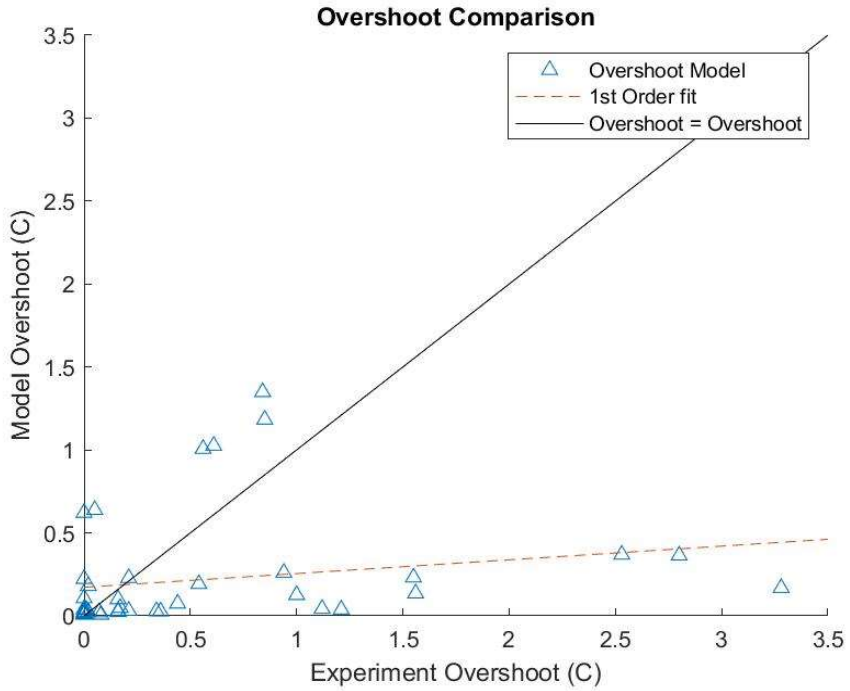

Figure 26: Overshoot, model vs experiment. Solid line represents $\text{Overshoot}_{\text{Model}} = \text{Overshoot}_{\text{Experiment}}$

Plotting the U percent error against vessel size and vessel type shows that there was no strong correlation between the error and these parameters. This bodes well for predicting scale up capabilities of different vessels and TCUs.

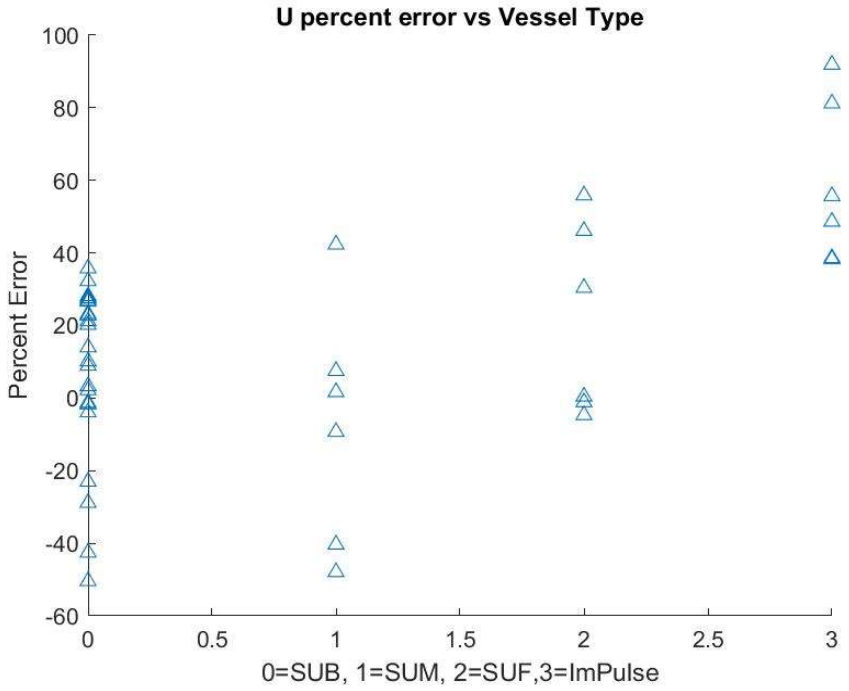

Figure 27: U percent error vs vessel type.

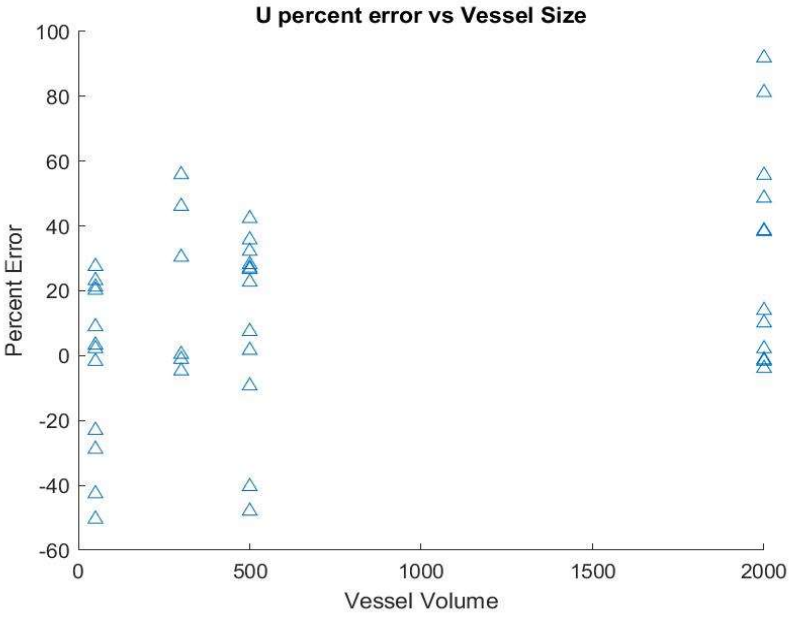

Figure: 28 U percent error vs vessel size:

On the importance of the U value to predicting accurate times, Figure 29 shows something particularly interesting. There is no strong trend between U percent error and T90% percent error. When the model U was 40% lower than or 100% higher than the corresponding experiment, it did not strongly deviate the model time from the experimentally measured time. We would expect a negative U percent error to be correlated with a positive T90% percent error and vice versa. Given most test cases spent the majority of the ramp cycle in controller saturation, meaning that the heater/chiller was at full output, which led to higher temperature differences between the jacket fluid and vessel fluid. As a result, it compensated for the lower U value. In short, the time to reach a targeted temperature is not only dependent upon the U value but also the TCU heating/cooling capacity.

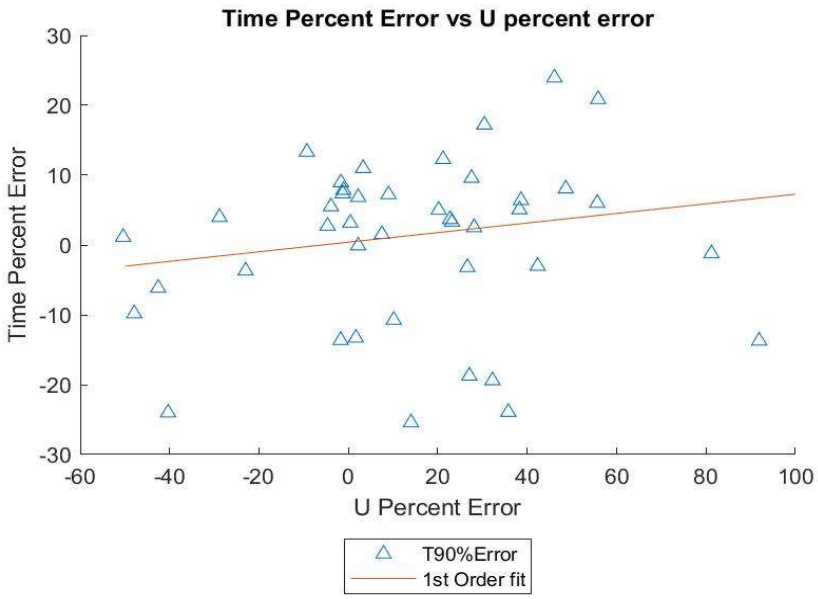

Figure 29: Time percent error vs U percent error.

A stronger trend between U value error and time error would be expected if the jacket fluid temperature is maintained at a constant value. One example of this would be cooling with an open system TCU. The jacket fluid temperature will never go below the temperature of the house feed, regardless of the mass flow rate. Figure 30 isolates these specific cases and shows that the expected trend is present.

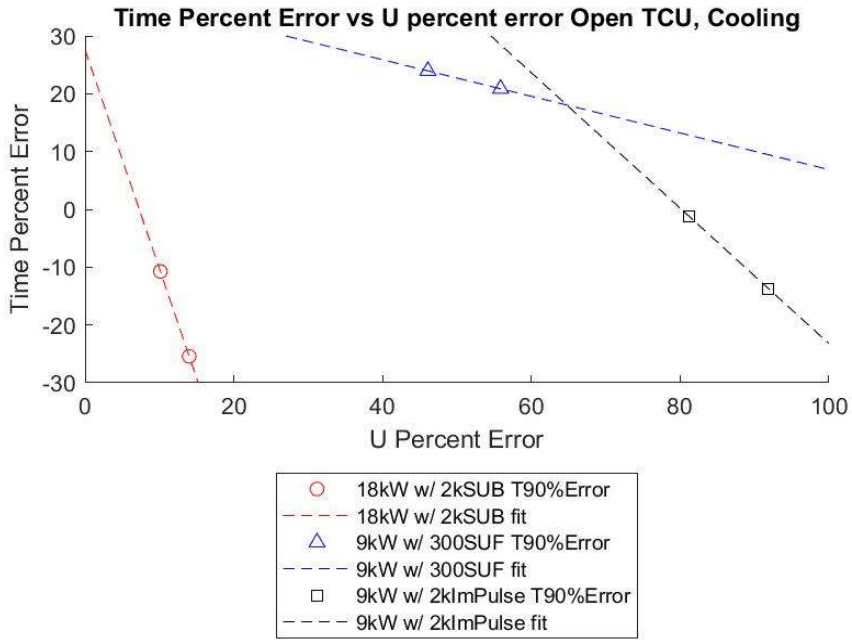

Figure 30: Time percent error vs U percent error, only open TCUs cooling to 5°C. Note that the negative slope is present for each TCU/vessel combination.

Figure 31 plots the results of Eq. 38, an error function that describes the difference in curve shape of the model and experiment vessel temperatures across the ramps.

$$E = \left[\sum\left(\frac{\left(T_{vs\ model} - T_{vsl_{experiment}}\right)^2}{n}\right)\right]^{0.5} \quad (38)$$

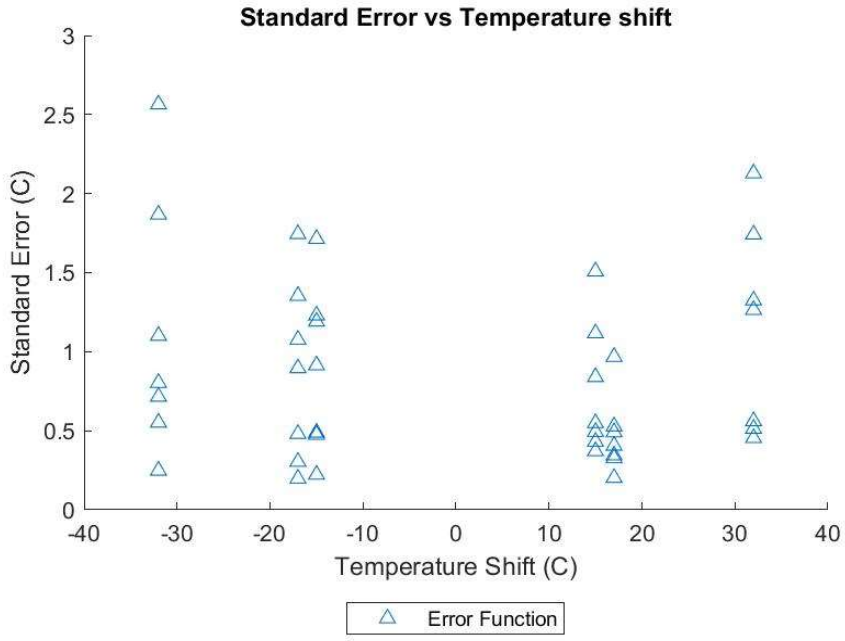

Figure 31: Standard error vs temp shift.

3.4 Nusselt Correlation Comparison

The nature of the computer model makes testing additional Nusselt correlations for the jacket trivial. Figures 32-33 show several different correlations compared with the experimental results. The following correlations were compared:

- Taler [11] correlation with Nu laminar = 7.54 [23]

- Taler correlation with Nu laminar = 4.86 [23]
- Gnielinski correlation [28] $Nu_D = \frac{\left(\frac{f}{8}\right)(Re_d - 1000)Pr}{1+12.7\left(\frac{f}{8}\right)^{\frac{1}{2}}\left(Pr^{\frac{2}{3}}-1\right)}$
- Dittus-Boelter Equation [29] $Nu_D = 0.023Re^{4/5}Pr^{0.3\ or\ 0.4}$
- Seban and Shimazaki [30] $Nu_D = 5.0 + 0.025Re^{0.8}Pr^{0.8}$

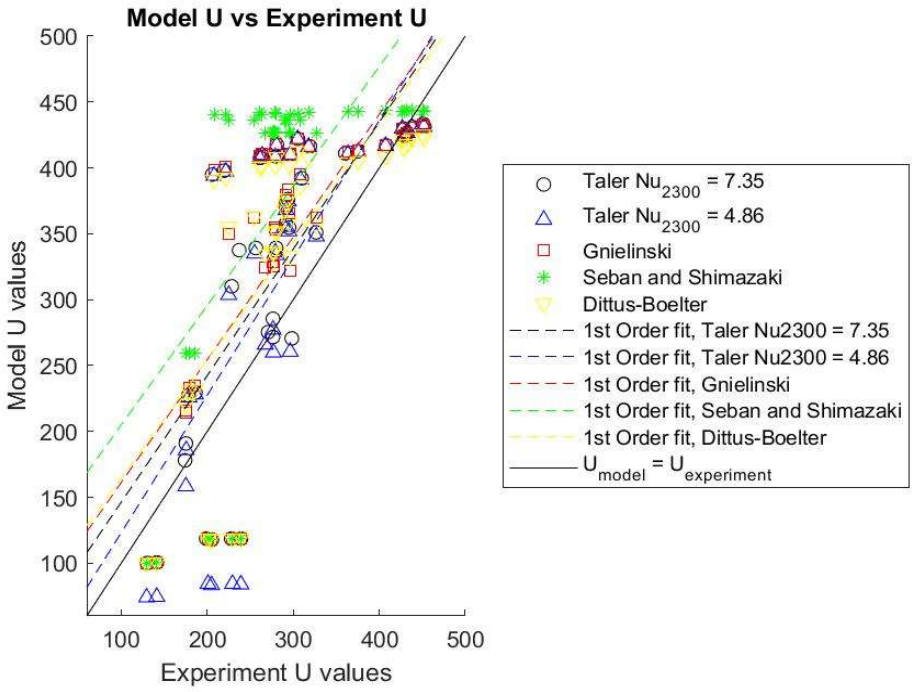

Figure 32: U value Nusselt number correlation comparison.

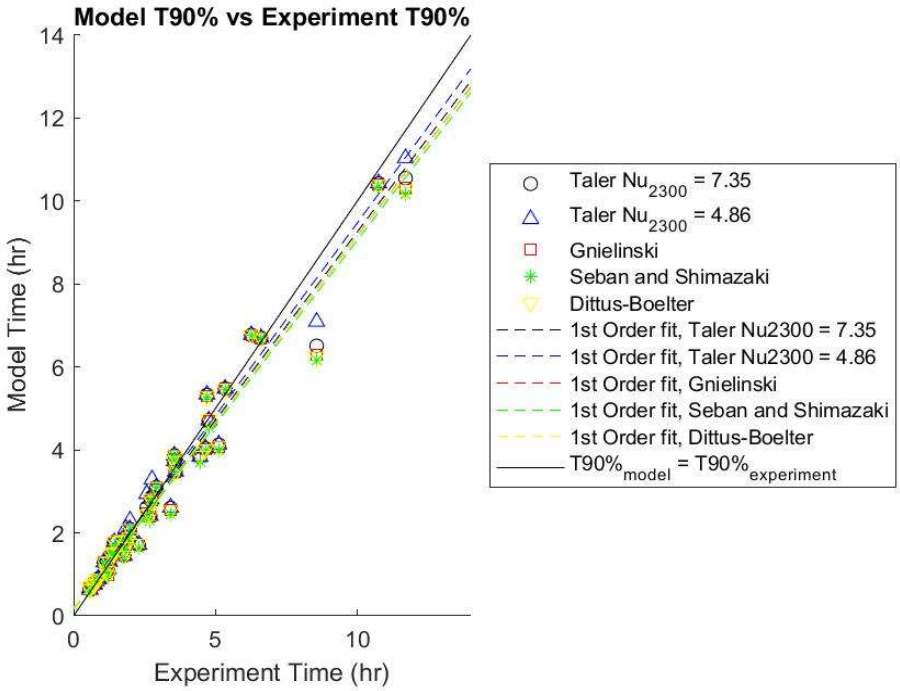

Figure 33:Time Nusselt number correlation comparison.

Taler's correlation with the raised laminar Nusselt number minimized the mean value of Eq. 38 across all runs.

3.5 First Order Estimate Comparison

When compared to a first order model, where time to set point is simply computed as thermal mass divided by TCU output power as in Eq. 39, the transient thermal model provides more accurate and consistent results.

(39)

$$t_{First\ Order} = \frac{m_{vsl} * cp_{vsl} * \Delta T_{vsl}}{P_{TCU}}$$

Figure 34 plots both the first order estimate and the model estimate for T90% error. This plot excludes cooling ramps that use an open TCU system as there is no direct first order method to predict the power from the TCU.

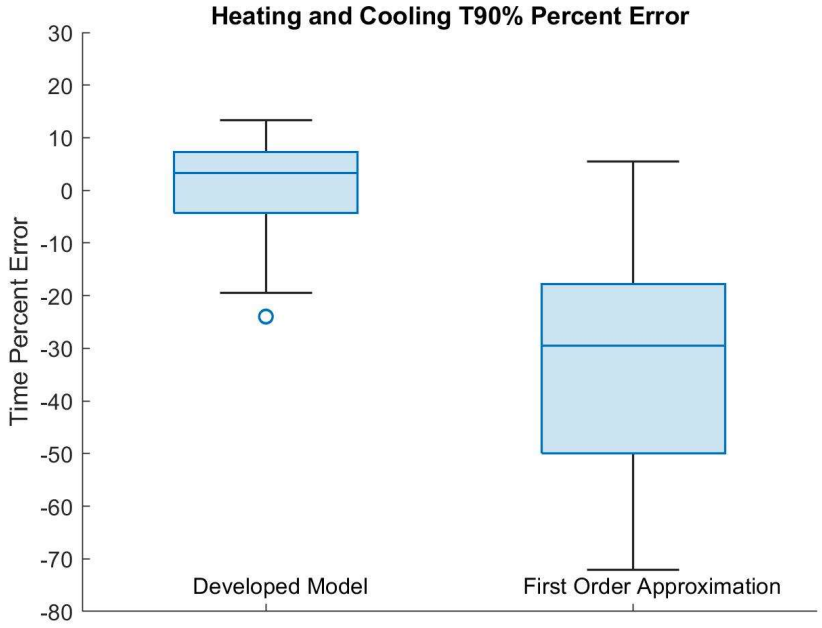

Figure 34: First order estimate vs full model T99% percent error.

The first order estimate has a standard deviation of 24.6%, with a mean percent error of -24.7%, this compares to the transient thermal model's standard deviation of

14.2%, with a mean percent error of +3.3%.

CHAPTER 4

CONCLUSION AND RECOMMENDATIONS

Overall, the developed model provides a good estimation for right-sizing vessel/TCU combinations. The flow and fluid property calculations captured the general trend of the overall heat transfer coefficient across tested vessels of 50L to 2000L.

Accuracy in the time to set point of the developed model differed between cooling and heating, with heating having a much smaller uncertainty. We expect that both using pump curve equations that are dynamically altered with changing viscosity and a closer examination of the Nusselt number correlations inside serpentine dimpled jackets at lower Reynold's numbers (cooler temperatures) could give insight to the discrepancies between the developed model and experimental results.

The developed model offers significant advantage over first order estimates, especially in its ability to predict the performance of open TCU systems where the cooling potential is not dictated by a nominal TCU chiller power, but by a house feed temperature and pressure drop. In examining the error in time to set point between model and experiment, the developed model had both a smaller standard deviation and a mean error less than 15% of the first order approximation.

The tool can also be used to enable users to quickly see the general effects of altering different PID parameters. While the model did not match real world overshoot measurements, it does provide a platform to see how thermal behavior trends with PID controls, allowing quicker real-world diagnosis of sub-optimal control choices. Future studies could be done to optimize the prediction of overshoot behavior. This model can

also be used to enable the end-user to examine both the effect and effectiveness of other control schemes, such as Gain Scheduling parameters, ARW parameters, and Cool Ratio.

REFERENCES

[1] Philippidis, Alex. "Top 15 Best-Selling Drugs of 2018: Sales for most treatments grow year-over-year despite concerns over rising prices." *Genetic Engineering & Biotechnology News* 39, no. 4 (2019): 16-17.

[2] Thermo Scientific HyPerforma Integrated Single-Use Bioreactors (S.U.B.). Thermo Fisher Scientific. Accessed July 16, 2020. https://www.bioprocessonline.com/doc/thermo-scientific-hyperforma-single-use-0002.

[3] Chilton, T. H., T. B. Drew, and R. H. Jebens. "Heat transfer coefficients in agitated vessels." *Industrial & Engineering Chemistry* 36, no. 6 (1944): 510-516.

[4] Lehrer, Isaac H. "Jacket-side Nusselt number." *Industrial & Engineering Chemistry Process Design and Development* 9, no. 4 (1970): 553-558.

[5] Mohan, Pankaj, Anthony Nicholas Emery, and Tariq Al-Hassan. "Review heat transfer to Newtonian fluids in mechanically agitated vessels." *Experimental thermal and fluid science* 5, no. 6 (1992): 861-883.

[6] Kai, Wang, and Yu Shengyao. "Heat transfer and power consumption of non-Newtonian fluids in agitated vessels." *Chemical engineering science* 44, no. 1 (1989): 33-40. [7] Haam, Seungjoo, Robert S. Brodkey, and Julian B. Fasano. "Local heat transfer in a mixing vessel using heat flux sensors." *Industrial & engineering chemistry research* 31, no. 5 (1992): 1384-1391. [8] Karcz, Joanna. "Studies of local heat transfer in a gas–liquid system agitated by double disc turbines in a slender vessel." *Chemical Engineering Journal* 72, no. 3 (1999): 217-227.

[9] Bielka, Iwona, Magdalena Cudak, and Joanna Karcz. "Local heat transfer process for a gas–liquid system in a wall region of an agitated vessel equipped with the system of CD6-RT impellers." *Industrial & Engineering Chemistry Research* 53, no. 42 (2014): 16539-16549.

[10] Gnielinski, V. "On heat transfer in tubes. International Journal of Heat and Mass Transfer." (2013): 134-140.

[11] Taler, Dawid. "A new heat transfer correlation for transition and turbulent fluid flow in tubes." *International Journal of Thermal Sciences* 108 (2016): 108-122.

[12] Dhotre, M. T., Z. V. P. Murthy, and N. S. Jayakumar. "Modeling & dynamic studies of heat transfer cooling of liquid in half-coil jackets." *Chemical engineering journal* 118, no. 3 (2006): 183-188. [13] Duan, Zhipeng. "New correlative models for fully developed turbulent heat and mass transfer in circular and noncircular ducts." *Journal of heat transfer* 134, no. 1 (2012).

[14] Duan, Zhipeng, M. M. Yovanovich, and Y. S. Muzychka. "Pressure drop for fully developed turbulent flow in circular and noncircular ducts." *Journal of Fluids Engineering* 134, no. 6 (2012).

[15] Mori, Yasuhiko H. "Thermal resistances in stirred-tank and tubular reactors for clathrate-hydrate formation: Estimating their reactor-scale dependences." *Chemical Engineering Research and Design* 117 (2017): 715-724.

[16] Löffelholz, Christian, Ute Husemann, Gerhard Greller, Wolfram Meusel, Jörg Kauling, Peter Ay, Matthias Kraume, Regine Eibl, and Dieter Eibl. "Bioengineering parameters for single-use bioreactors: overview and evaluation of suitable methods." *Chemie Ingenieur Technik* 85, no. 1-2 (2013): 40-56.

[17] Müller, Matthias, Ute Husemann, Gerhard Greller, Wolfram Meusel, and Matthias Kraume. "Heat transfer characteristics of a stirred single-use bioreactor." *Biochemical Engineering Journal* 140 (2018): 168-177.

[18] Sun, Tongfan, and Amyn S. Teja. "Density, viscosity and thermal conductivity of aqueous solutions of propylene glycol, dipropylene glycol, and tripropylene glycol between 290 K and 460 K." *Journal of Chemical & Engineering Data* 49, no. 5 (2004): 1311-1317.

[19] Chesaru, Valentin, Claudius Dan, and Mircea Bodea. "Pid Algorithm for controller processors." In *2006 International Semiconductor Conference*, vol. 2, pp. 429-432. IEEE, 2006.

[20] Cornieles, E., Maarouf Saad, Guy Gauthier, and Hamadou Saliah-Hassane. "Modeling and simulation of a multivariable process control." In *2006 IEEE International Symposium on Industrial Electronics*, vol. 4, pp. 2700-2705. IEEE, 2006.

[21] Controlguru. “Process Control Preliminaries.” Control Guru. Accessed July 16, 2020. https://controlguru.com/the-normal-or-standard-pid-algorithm/.

[22] Yamamoto, K., Hydrodynamic Study in Agitated Vessels, Ph.D. Thesis, Kyoto Univ., Kyoto, 1961.

[23] Kays, W. M., and M. E. Crawford. "Convective Heat and Mass Transfer, 3 d ed., McGraw-Hill, New York, 1993."

[24] Swanee, P. K., and Akalank K. Jain. "Explicit equations for pipeflow problems." *Journal of the hydraulics division* 102, no. 5 (1976).

[25] “How to Use Flow Coefficients (Cv) for Hydraulic Fluids.” Womack Machine Supply Company. Accessed July 15, 2020. https://www.womackmachine.com/engineering-toolbox/data-sheets/how-to-use-flow-coefficients-cv-for-hydraulic-fluids/.

[26] Omega Instrumentation, “PX119 Series Pressure Transmitters”. Accessed July 16, 2020. https://assets.omega.com/pdf/test-and-measurement-equipment/pressure/pressure-transducers/PX119.pdf

[27] Onset Instrumentation, “HOBO S-TMB-M0xx Sensor Datasheet”. Accessed July 16, 2020. https://www.onsetcomp.com/datasheet/S-TMB-M0xx

[28] Gnielinski, Volker. "New equations for heat and mass transfer in turbulent pipe and channel

flow." *Int. Chem. Eng.* 16, no. 2 (1976): 359-368.

[29] Whitaker, S. "Forced convection heat transfer calculations for flow in pipes, past flat plate, single cylinder, and for flow in packed beds and tube bundles." *AIChE J* 18 (1972): 361-371.

[30] Seban, Ralph Alois, and Trans Shimazaki. *Heat transfer to a fluid flowing turbulently in a smooth pipe with walls at constant temperature*. CALIFORNIA UNIV BERKELEY INST OF ENGINEERING RESEARCH, 1949.

APPENDICES

APPENDIX A
EXPERIMENTAL TEMPERATURE RAMPS

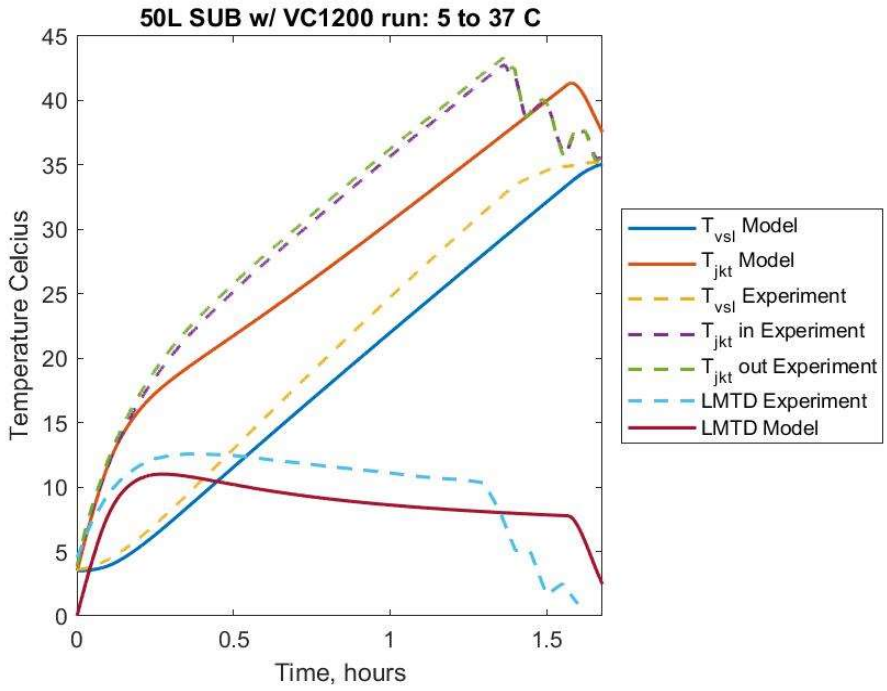

Figure 35:Lauda VC1200 with 50 SUB 5-37°C.

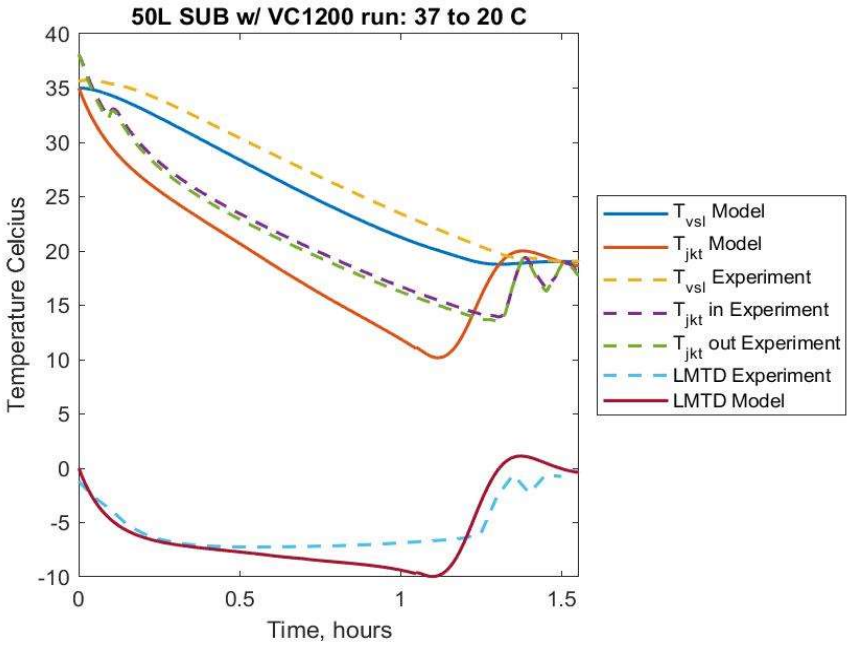

Figure 36:Lauda VC1200 with 50 SUB 37-20°C.

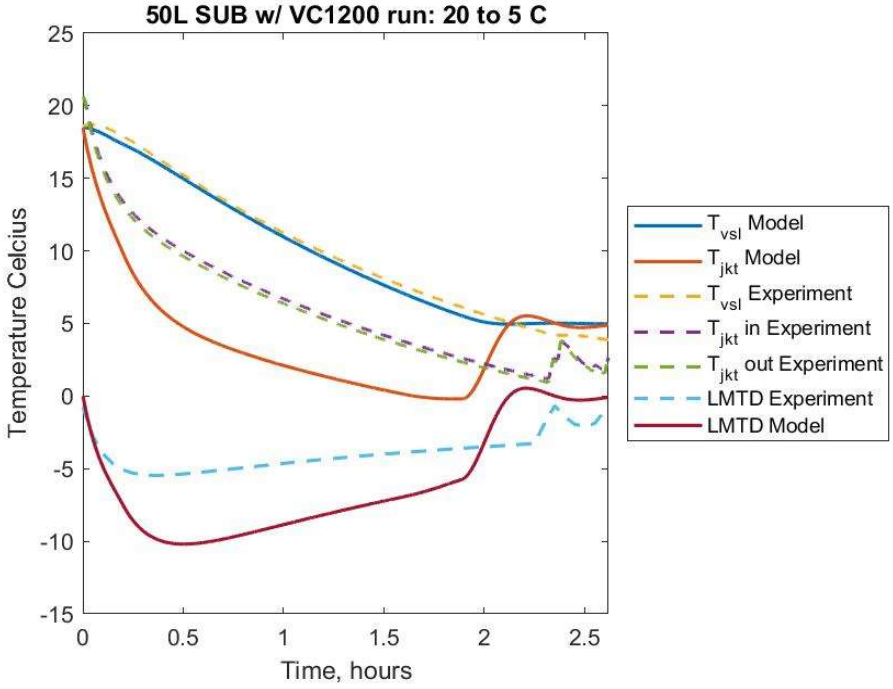

Figure 37: Lauda VC1200 with 50 SUB 20-5°C.

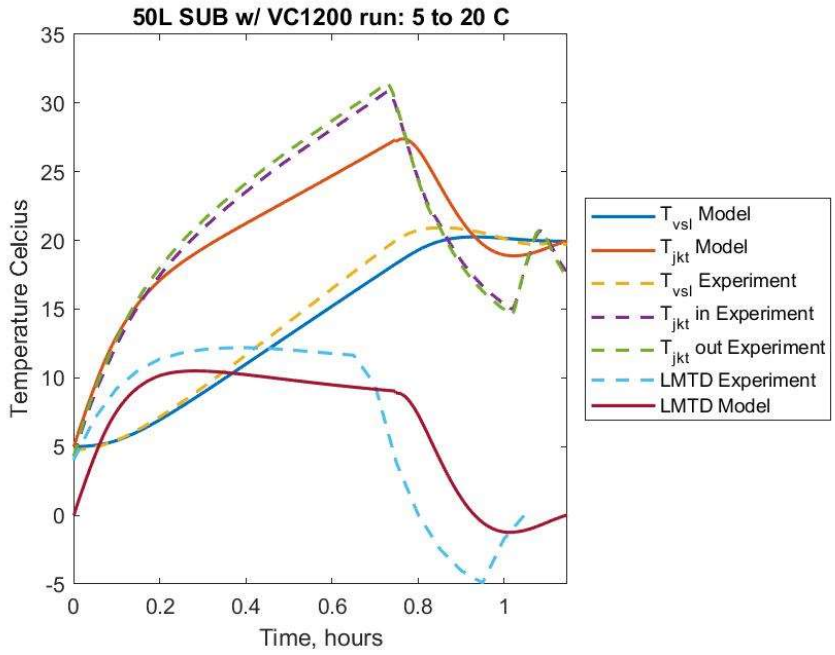

Figure 38: Lauda VC1200 with 50 SUB 5-20°C.

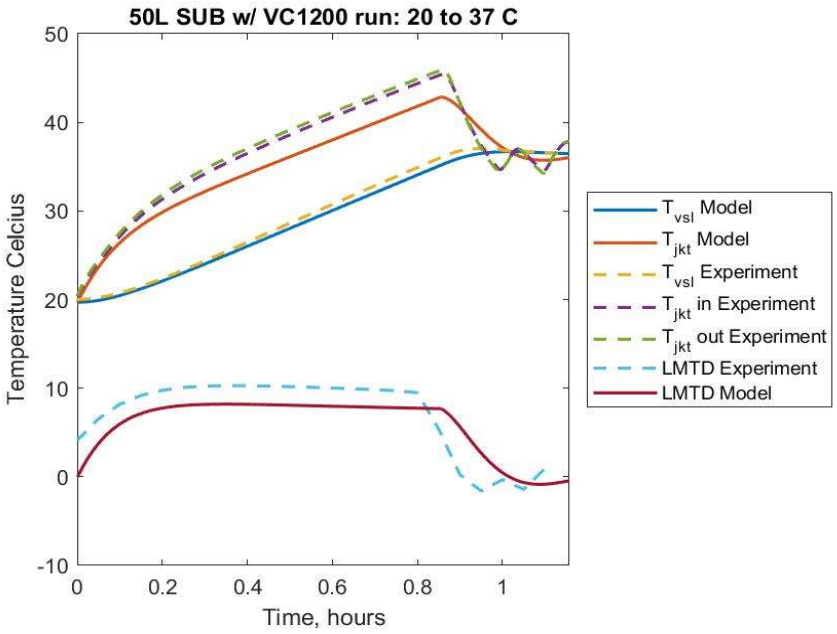

Figure 39: Lauda VC1200 with 50 SUB 20-37°C.

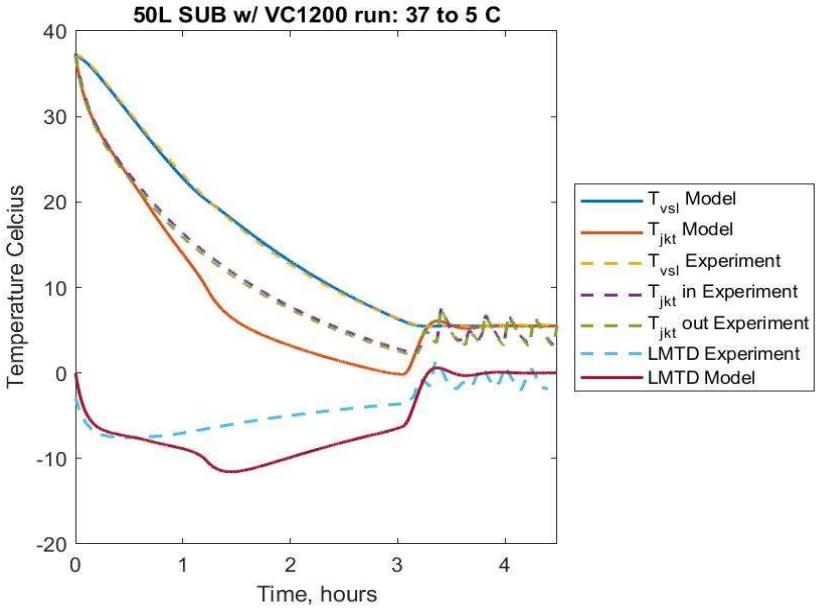

Figure 40: Lauda VC1200 with 50 SUB 37-5°C.

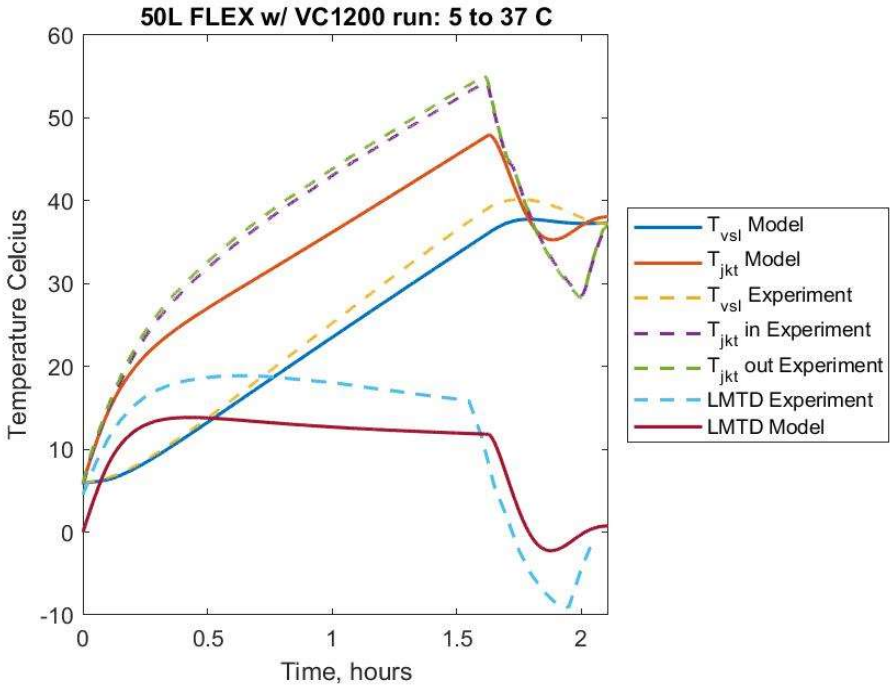

Figure 41: Lauda VC1200 with 50 FLEX 5-37°C.

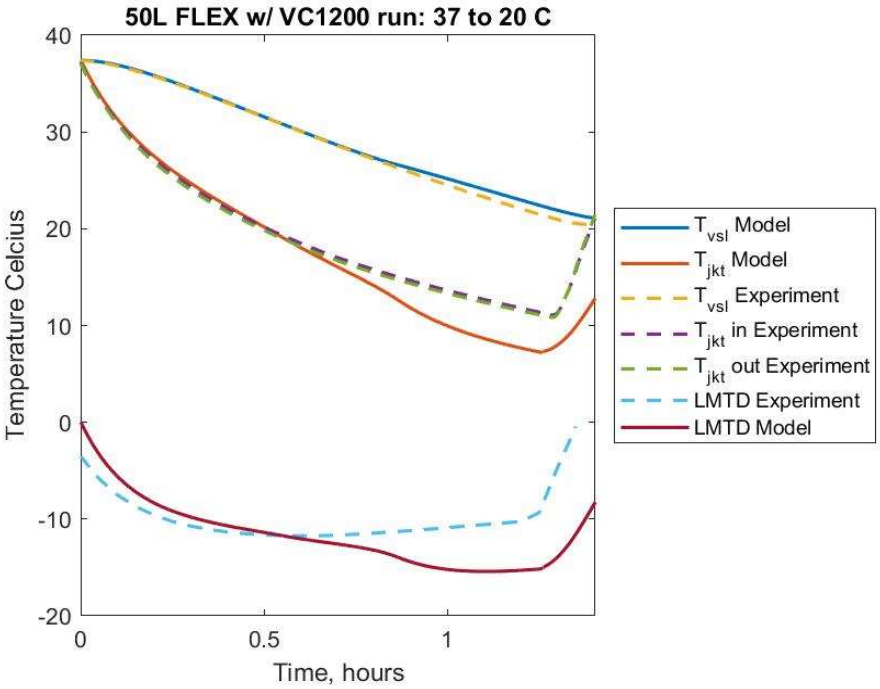

Figure 42: Lauda VC1200 with 50 FLEX 37-20°C.

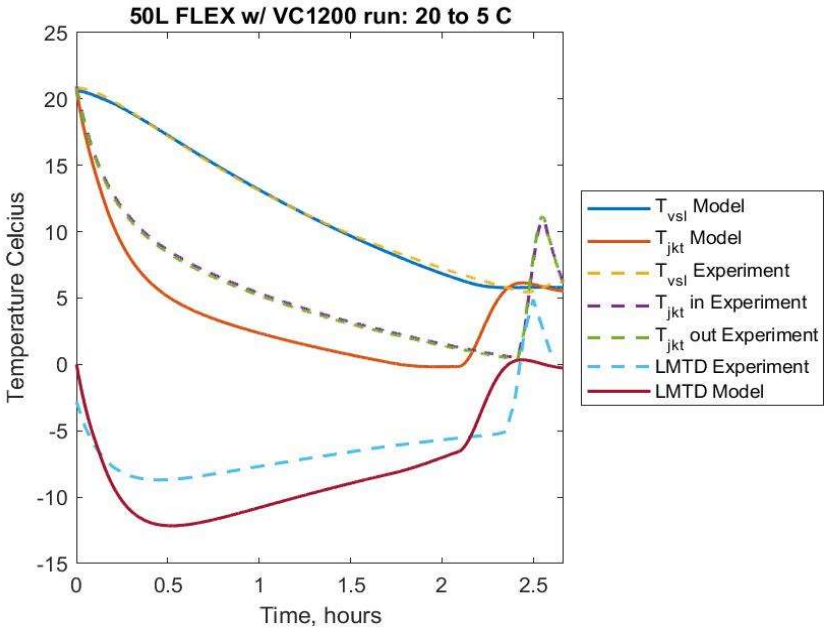

Figure 43: Lauda VC1200 with 50 FLEX 20-5°C

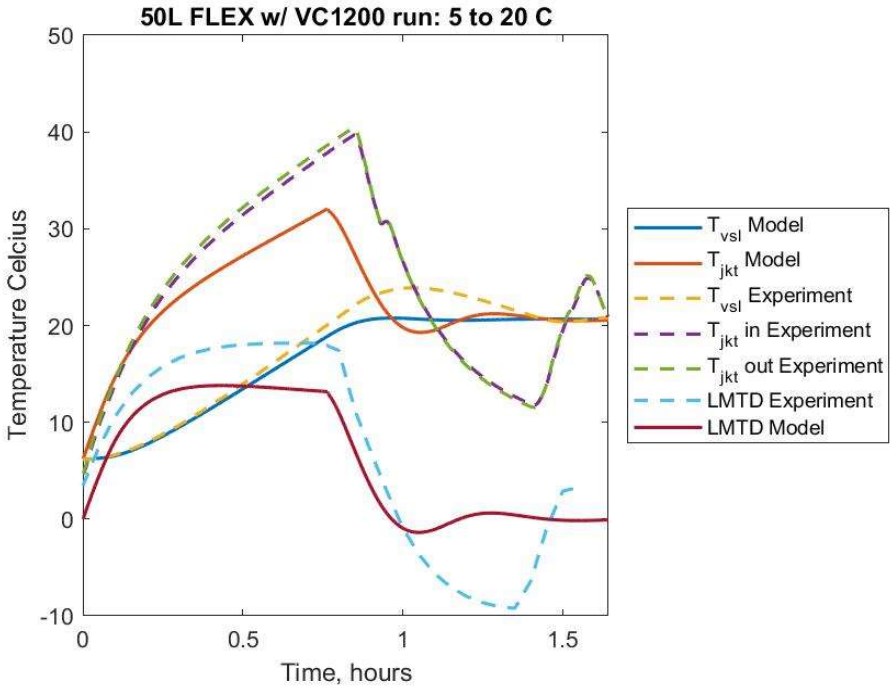

Figure 44: Lauda VC1200 with 50 FLEX 5-20°C

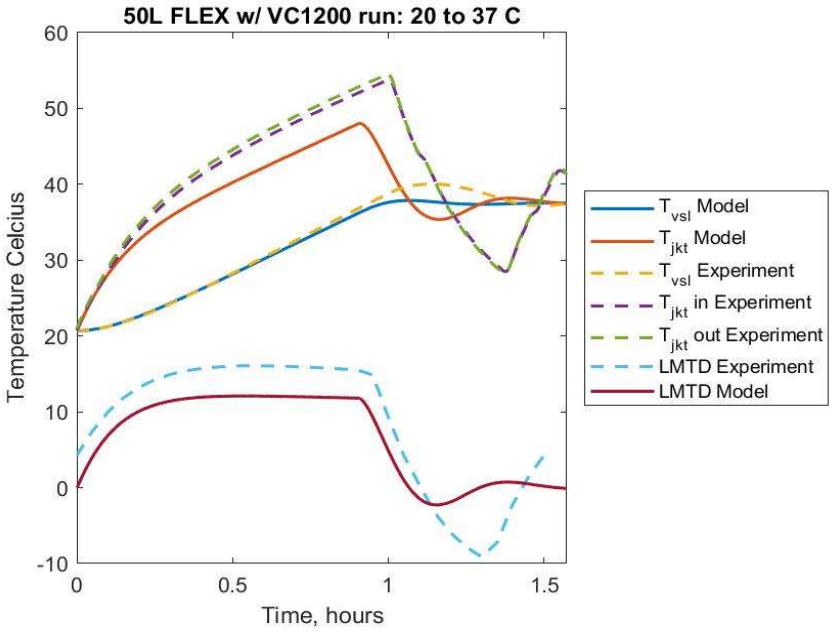

Figure 45: Lauda VC1200 with 50 FLEX 20-37°C.

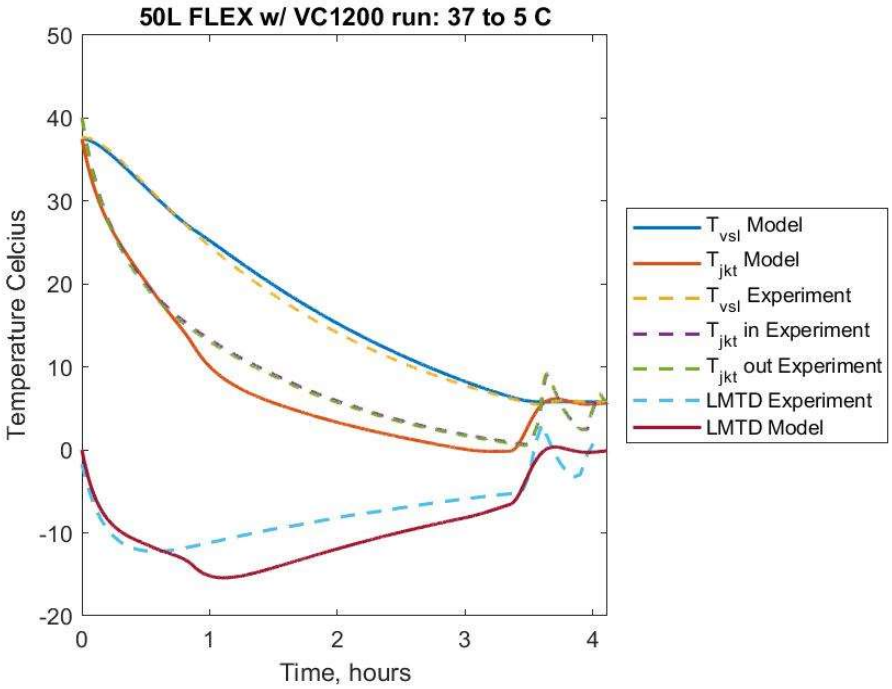

Figure 46: Lauda VC1200 with 50 FLEX 37-5°C.

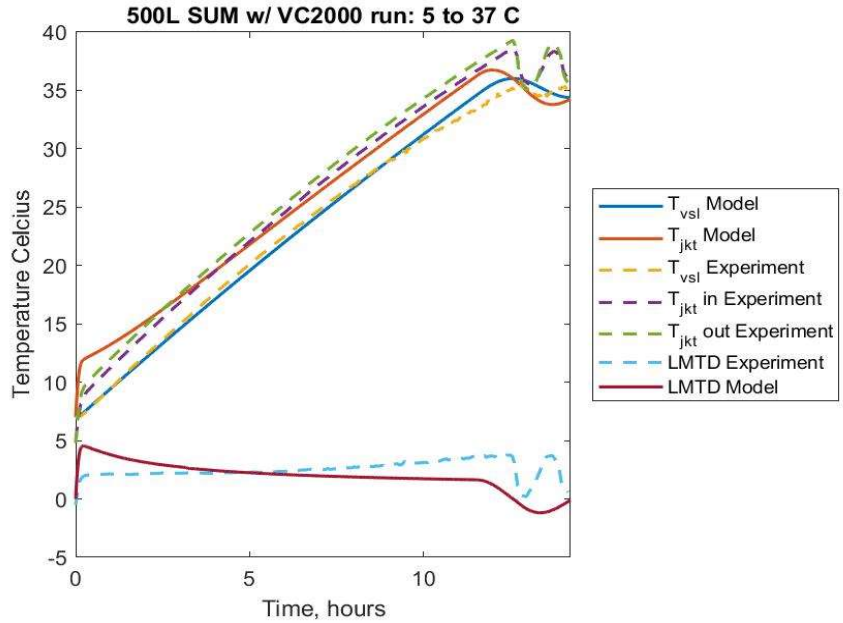

Figure 47: Lauda VC2000 with 500 SUM 5-37°C.

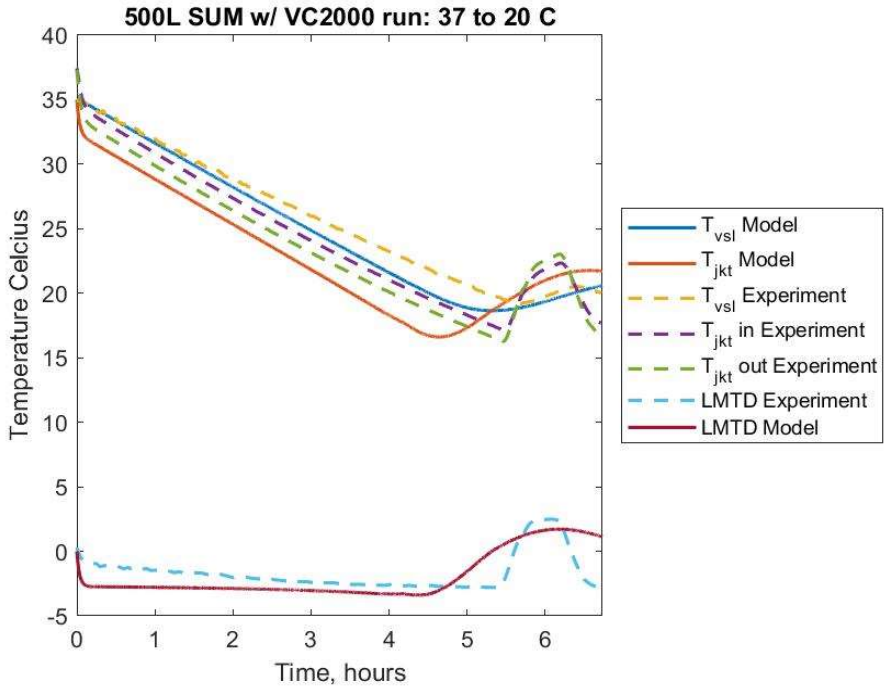

Figure 48: Lauda VC2000 with 500 SUM 37-20°C.

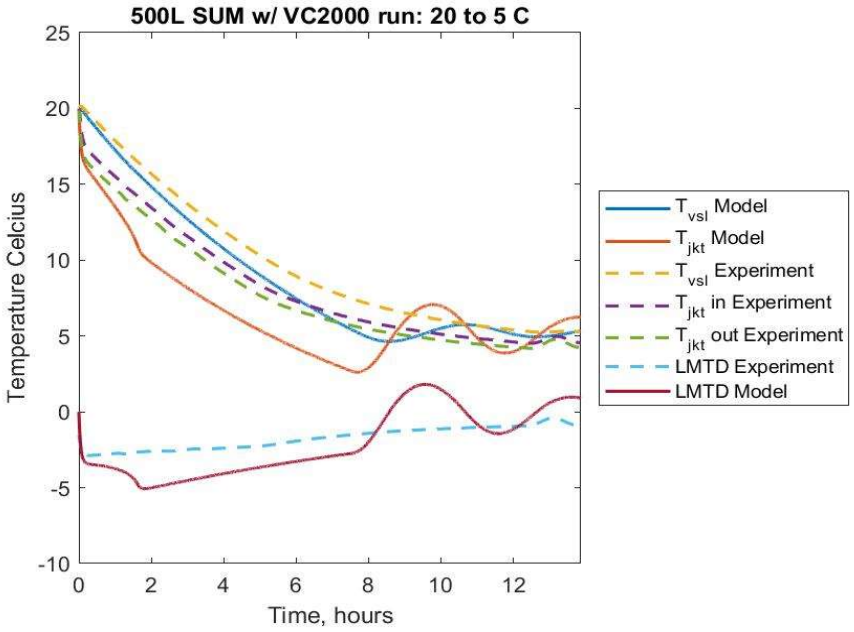

Figure 49: Lauda VC2000 with 500 SUM 20-5°C.

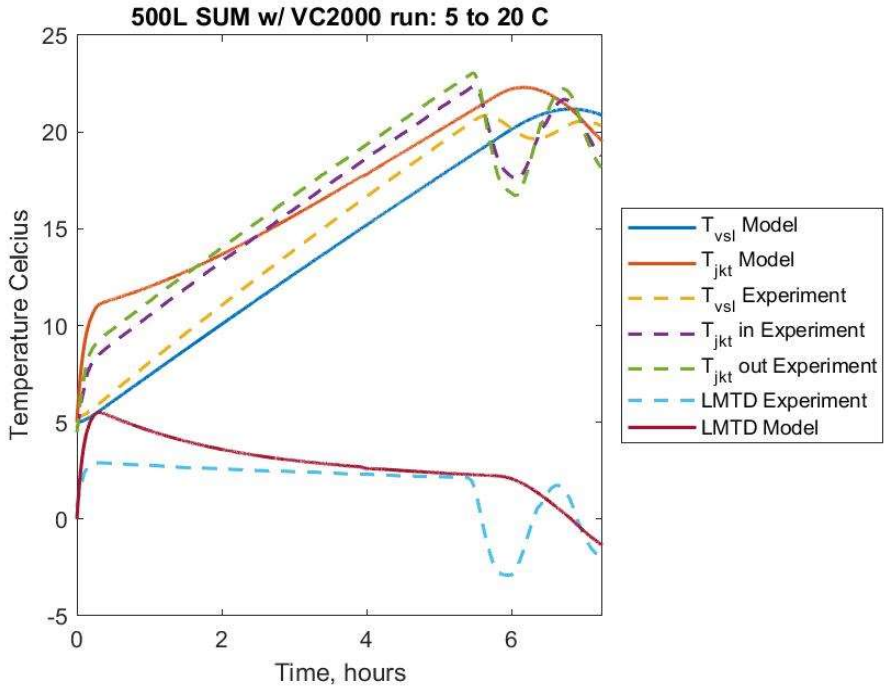

Figure 50: Lauda VC2000 with 500 SUM 5-20°C.

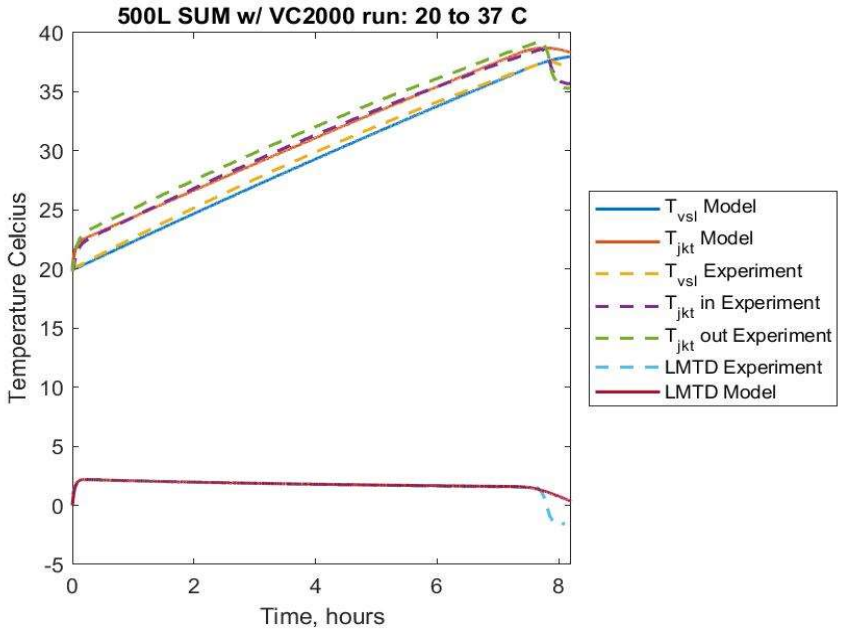

Figure 51: Lauda VC2000 with 500 SUM 20-37°C.

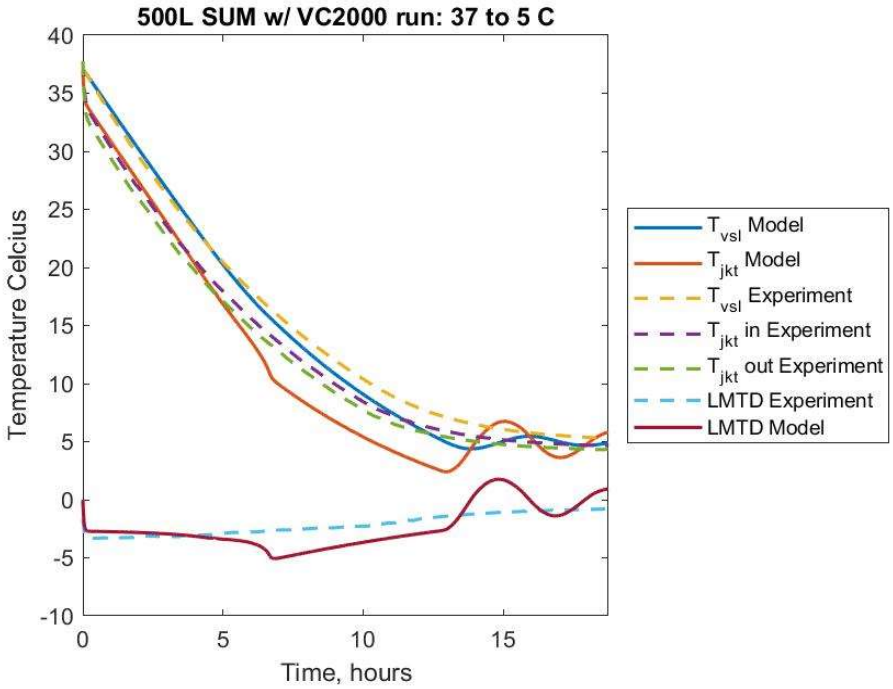

Figure 52: Lauda VC2000 with 500 SUM 37-5°C.

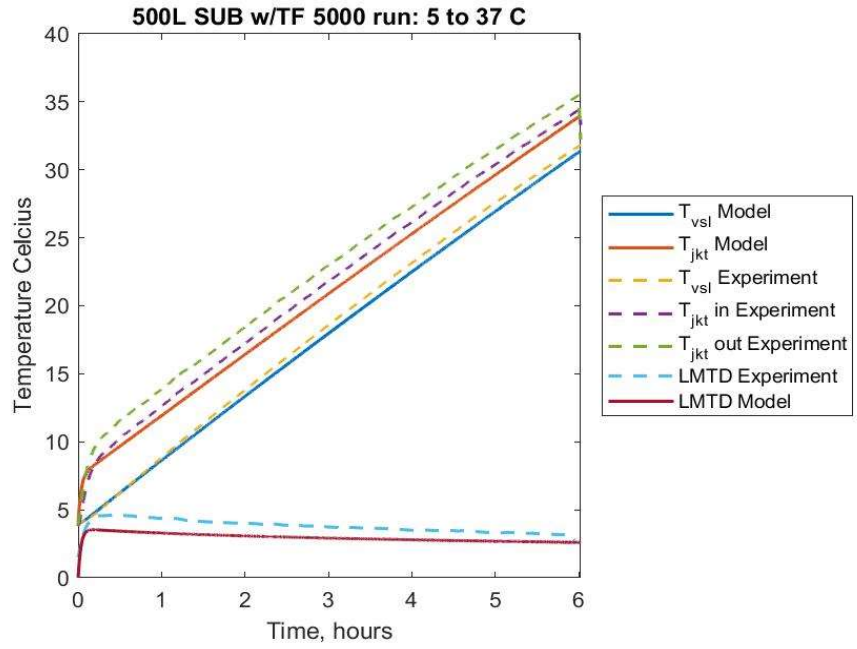

Figure 53: Thermoflex TF5000 with 500 SUB 5-37°C.

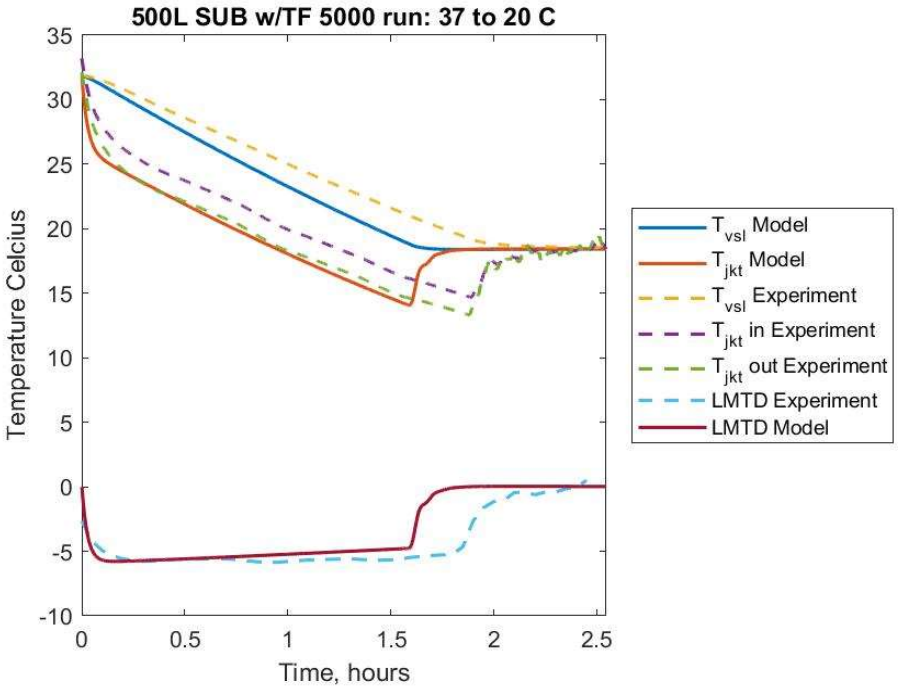

Figure 54: Thermoflex TF5000 with 500 SUB 37-20°C

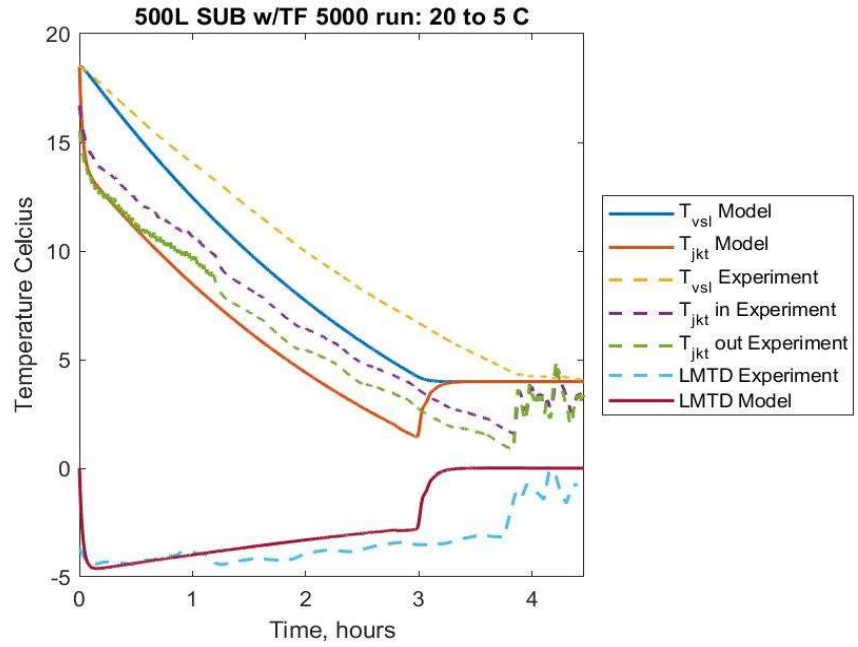

Figure 55: Thermoflex TF5000 with 500 SUB 20-5°C

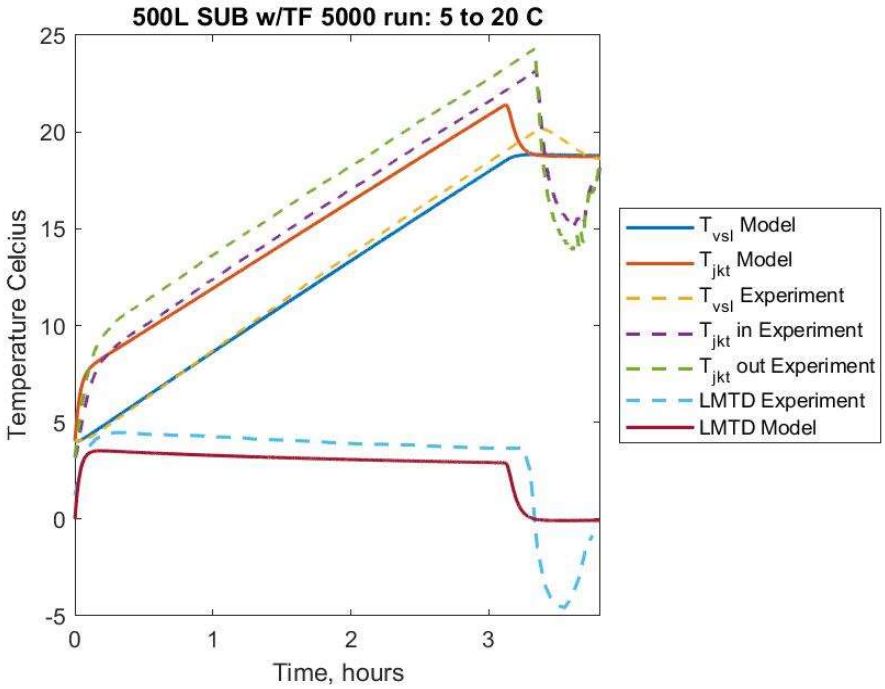

Figure 56: Thermoflex TF5000 with 500 SUB 5-20°C

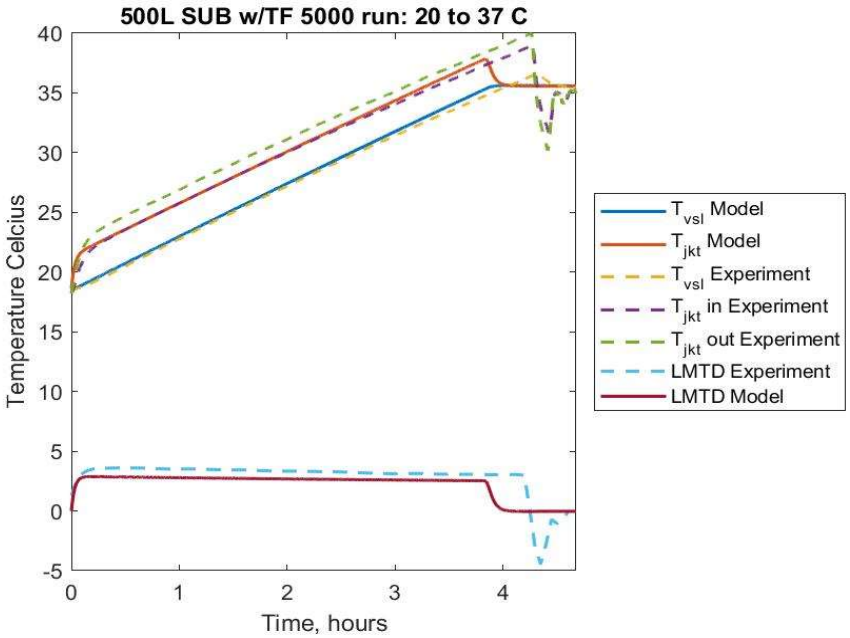

Figure 57: Thermoflex TF5000 with 500 SUB 20-37°C.

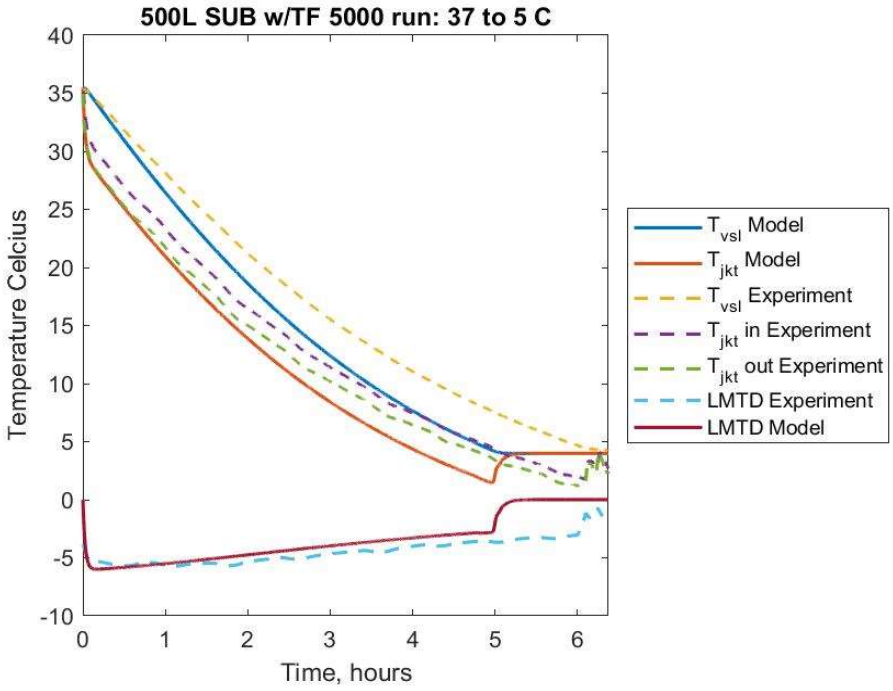

Figure 58: Thermoflex TF5000 with 500 SUB 37-5°C.

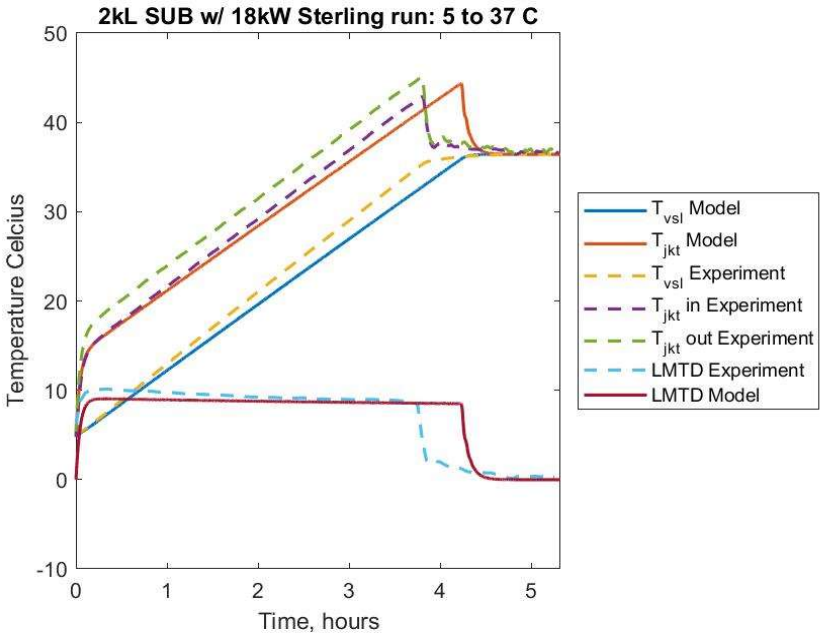

Figure 59: Sterling 18kW with 2k SUB 5-37°C.

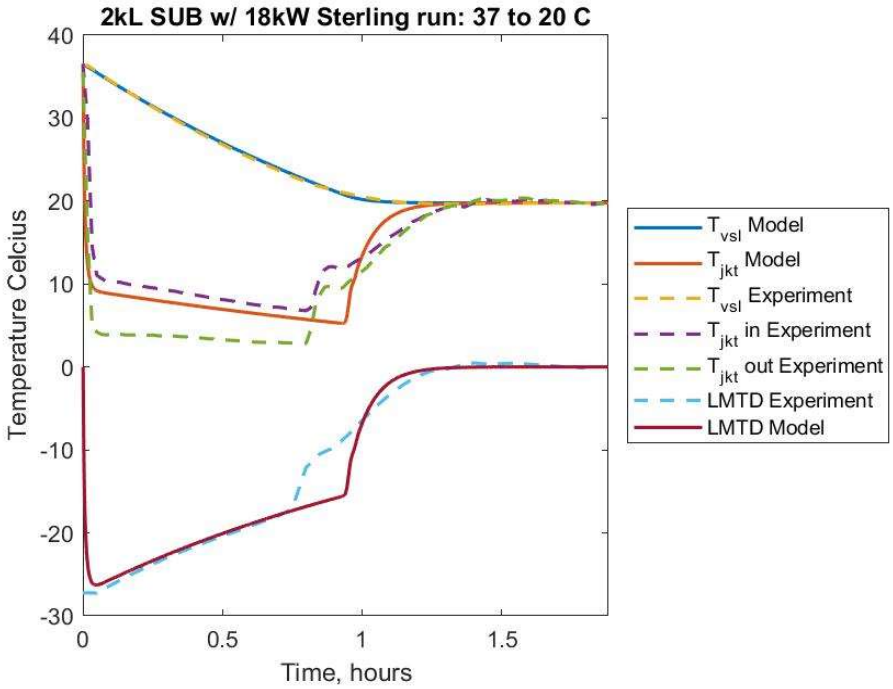

Figure 60: Sterling 18kW with 2k SUB 37-20°C.

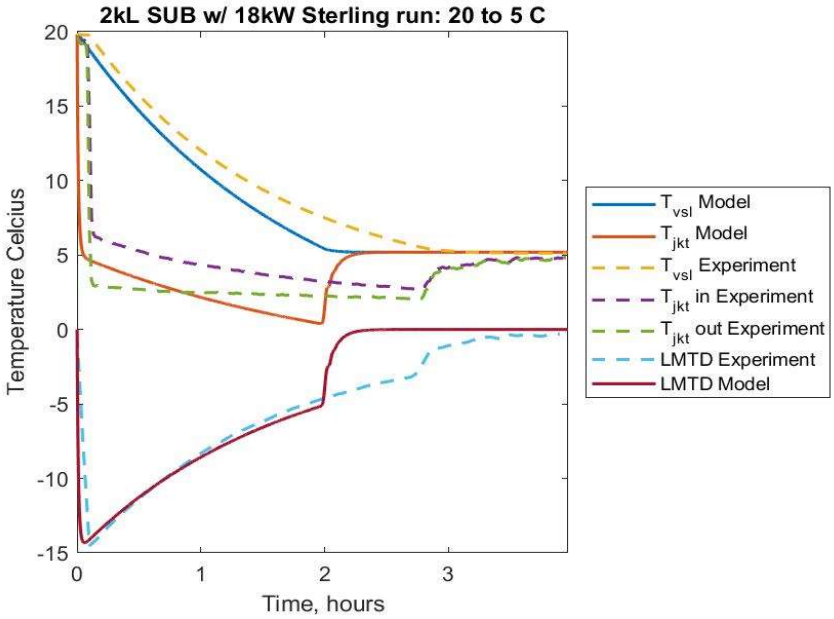

Figure 61: Sterling 18kW with 2k SUB 20-5°C.

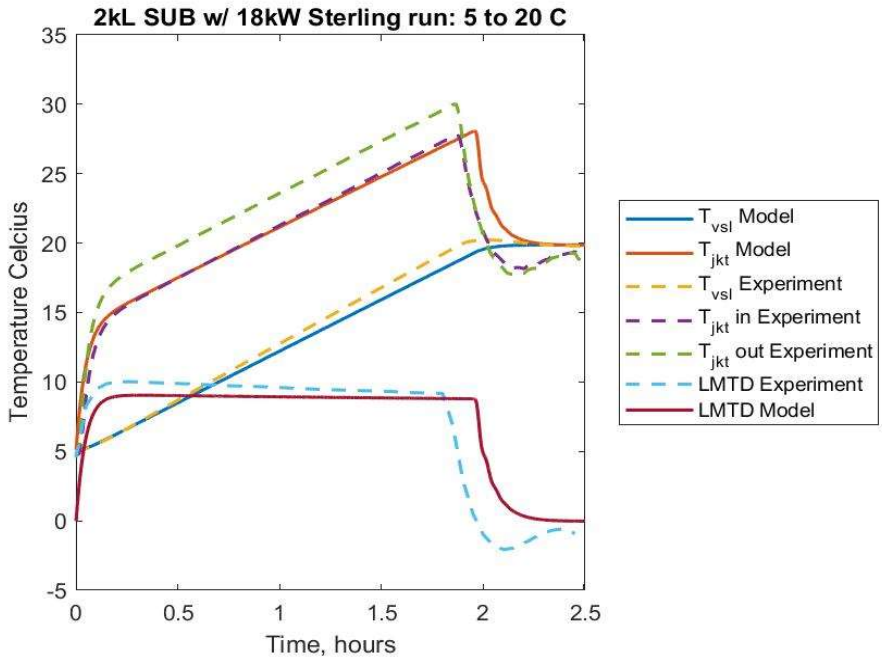

Figure 62: Sterling 18kW with 2k SUB 5-20°C

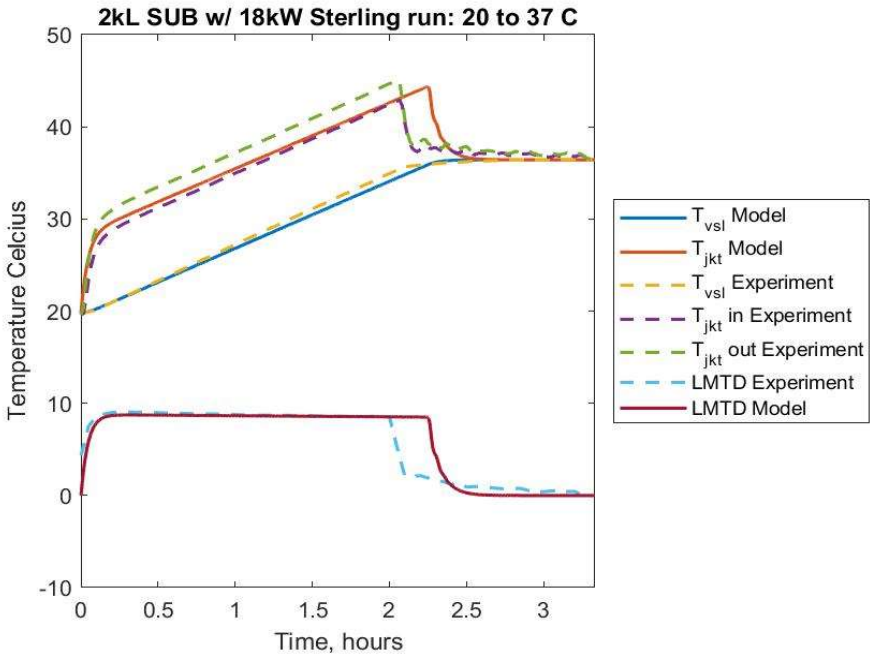

Figure 63: Sterling 18kW with 2k SUB 20-37°C

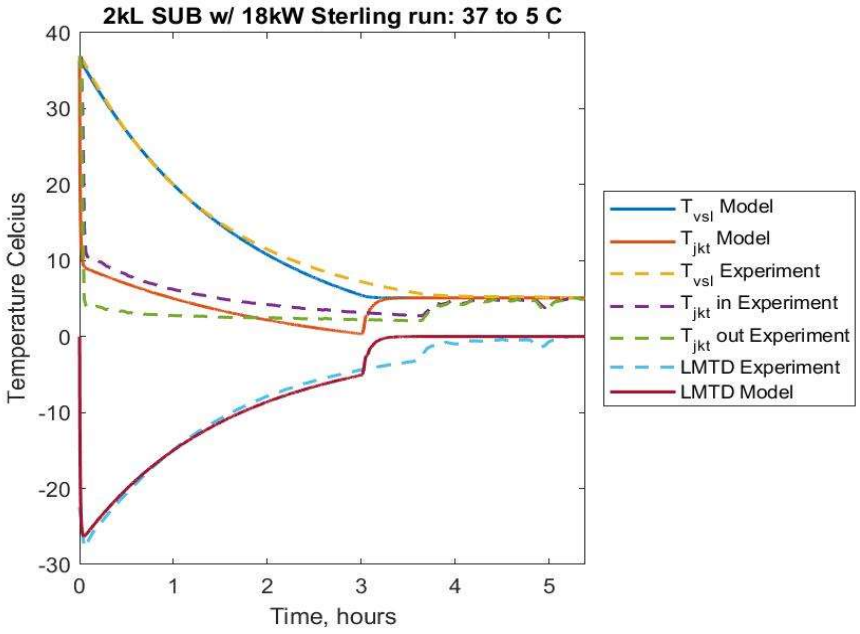

Figure 64: Sterling 18kW with 2k SUB 37-5°C.

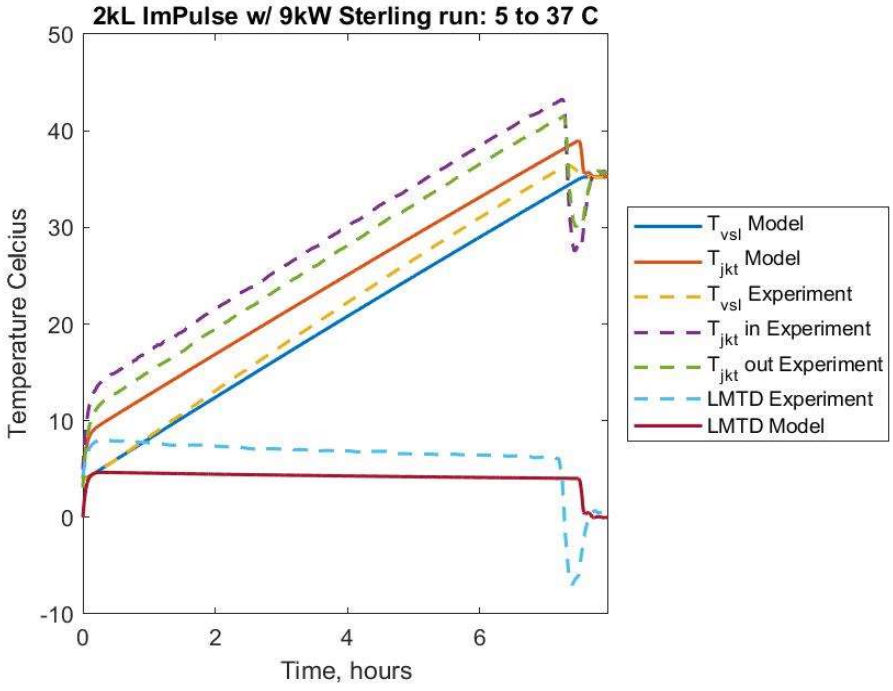

Figure 65 Sterling 9kW with 2k ImPulse 5-37°C.

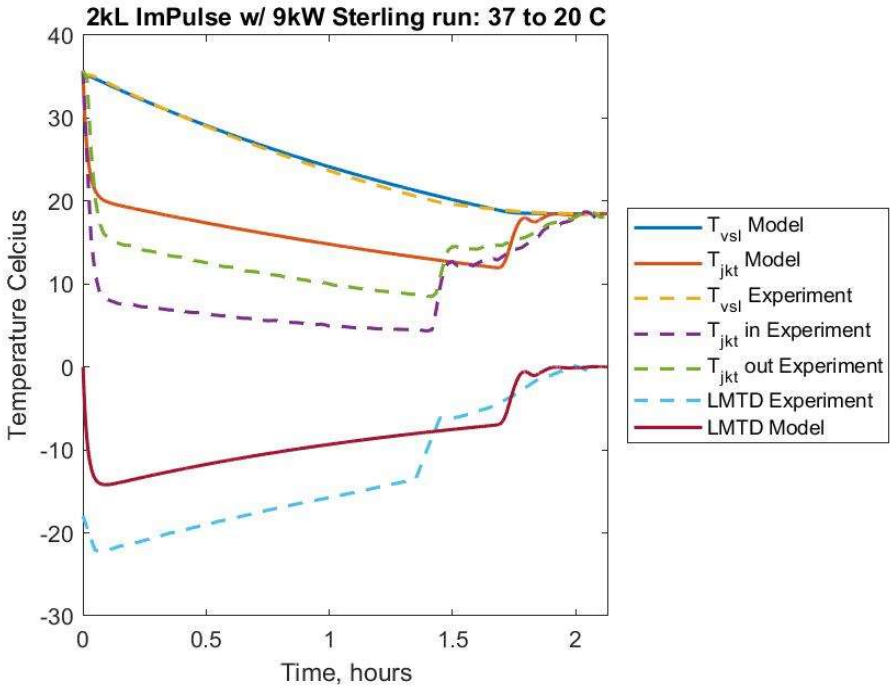

Figure 66: Sterling 9kW with 2k ImPulse 37-20°C.

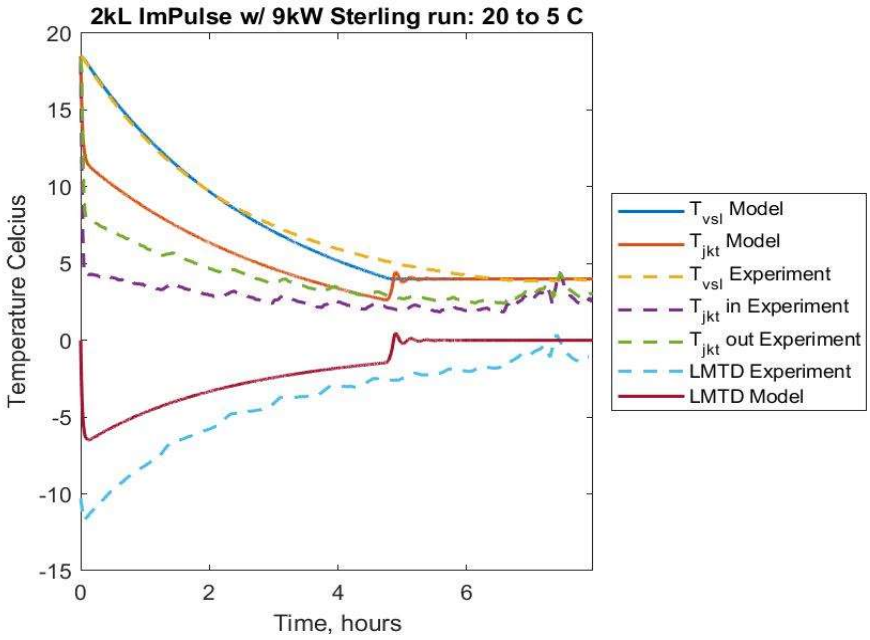

Figure 67: Sterling 9kW with 2k ImPulse 20-5°C.

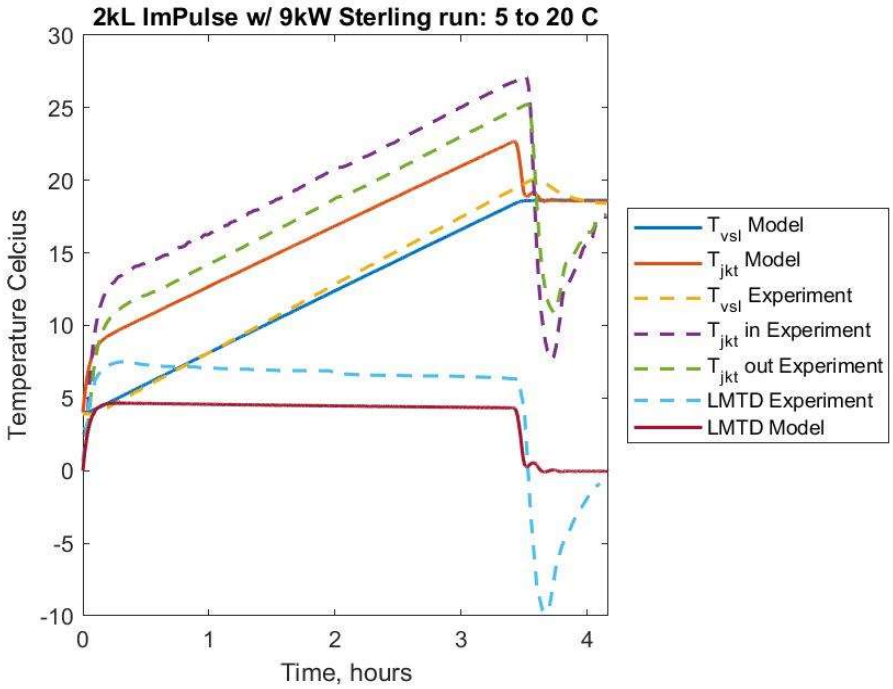

Figure 68: Sterling 9kW with 2k ImPulse 5-20°C

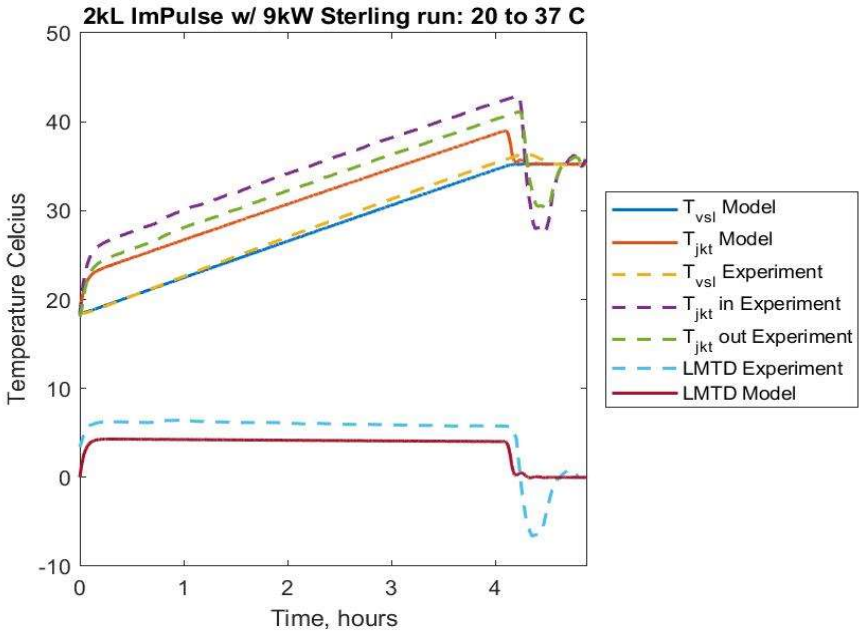

Figure 69: Sterling 9kW with 2k ImPulse 20-37°C.

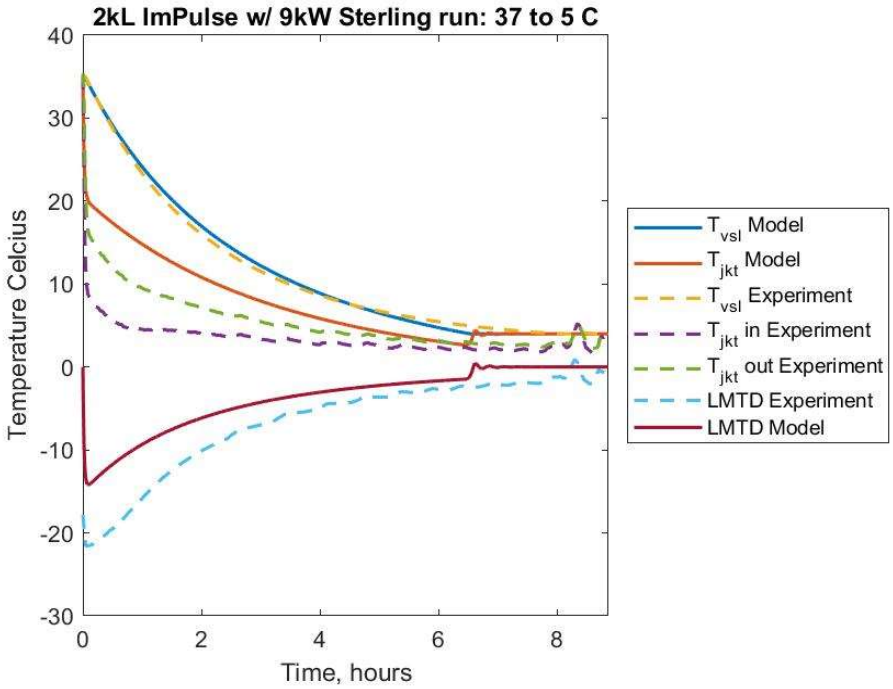

Figure 70: Sterling 9kW with 2k ImPulse 37-5°C.

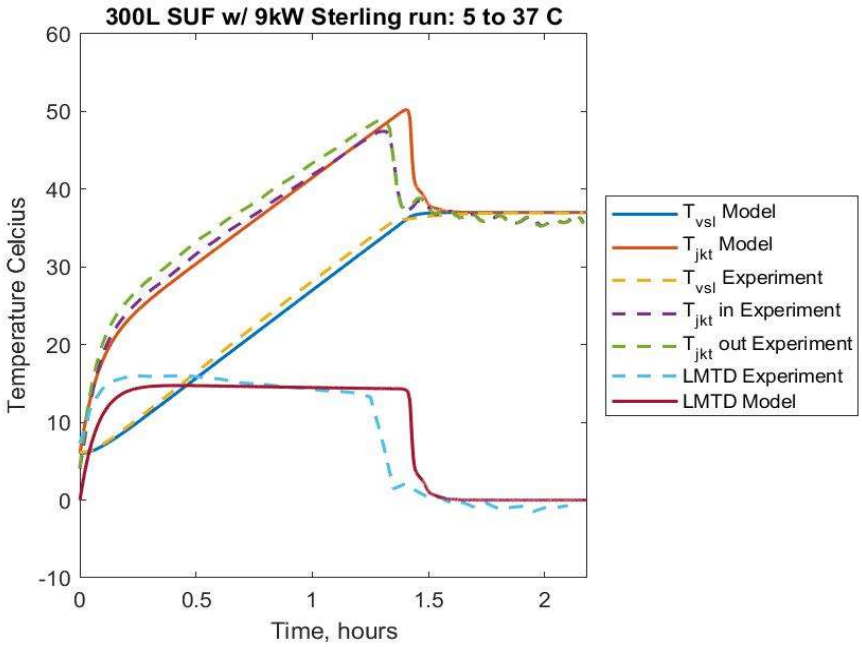

Figure 71: Sterling 9kW with 300 SUF 5-37°C.

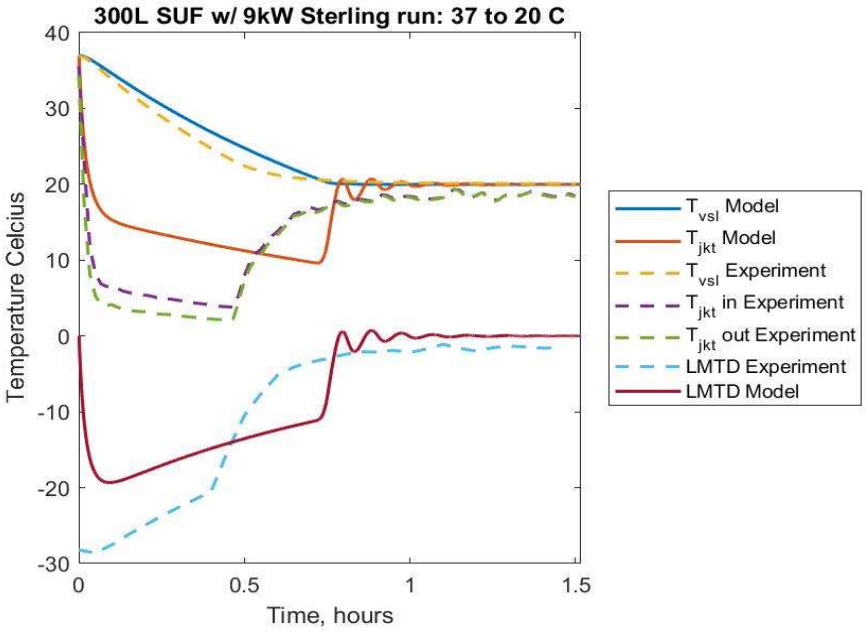

Figure 72: Sterling 9kW with 300 SUF 37-20°C.

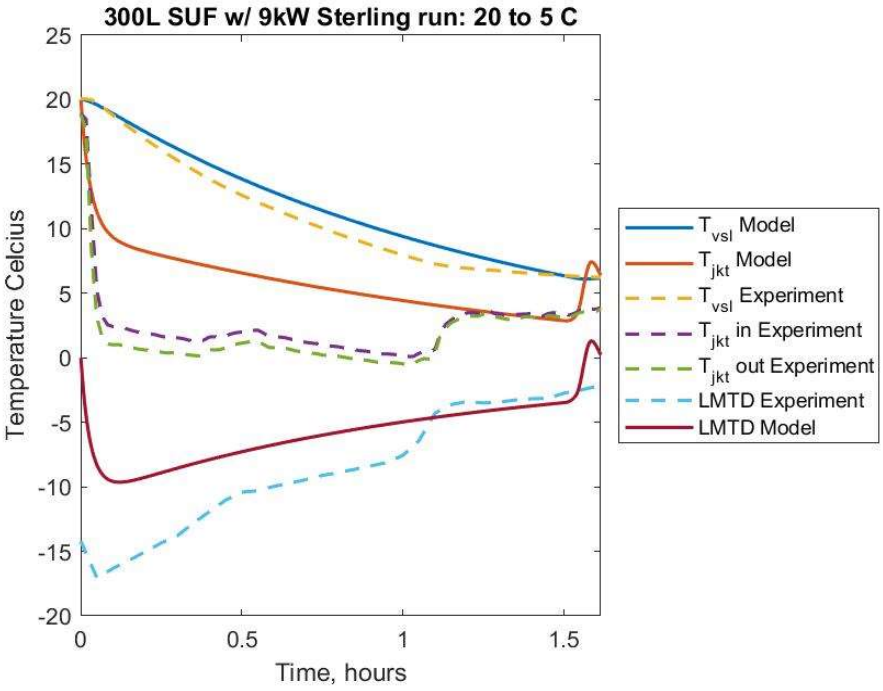

Figure 73: Sterling 9kW with 300 SUF 20-5°C.

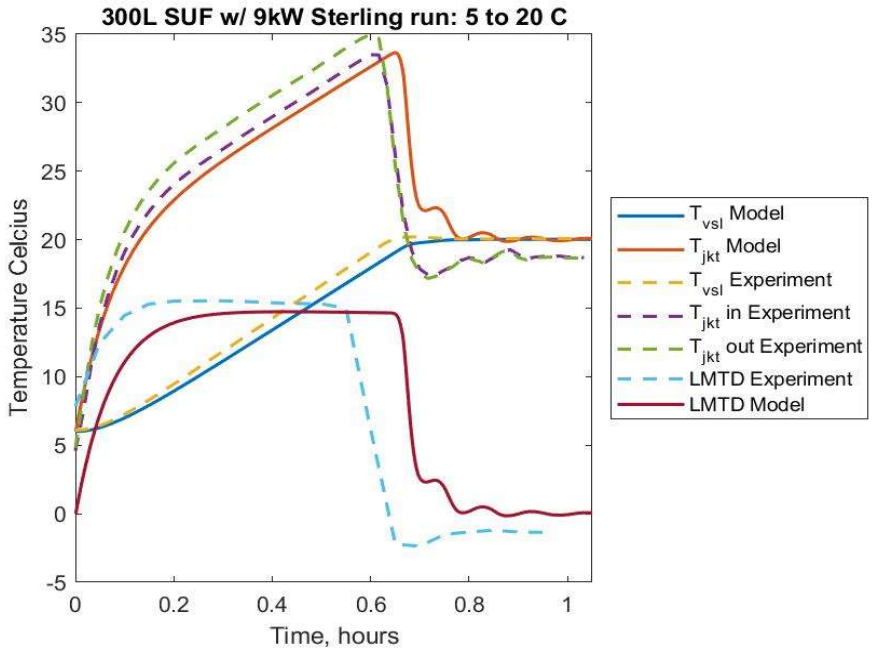

Figure 74: Sterling 9kW with 300 SUF 5-20°C.

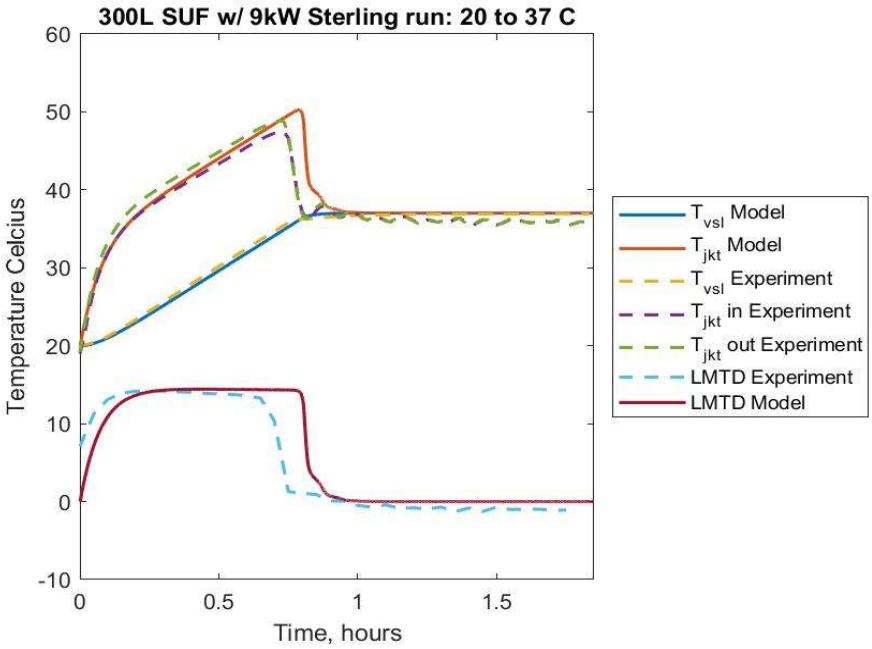

Figure 75: Sterling 9kW with 300 SUF 20-37°C.

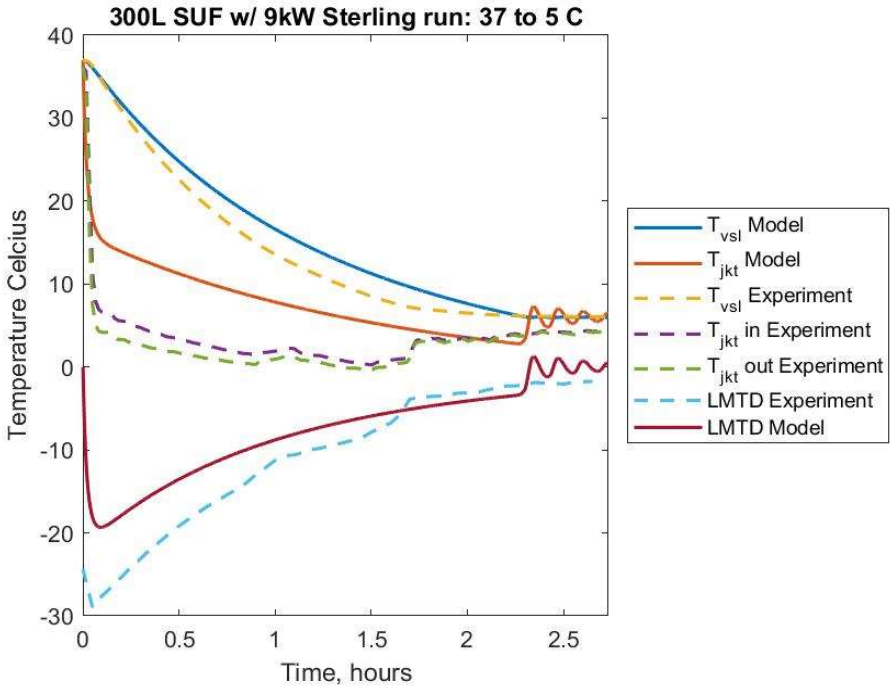

Figure 76: Sterling 9kW with 300 SUF 37-5°C.

APPENDIX B
CODE SELECTIONS

```
function ModelRun(app, event)


%Collect Variables
VslPropSet(app,event)
TcuPropSet(app,event)
PID_update(app,event)
InitialProp(app)
for k = 1:app.IterProp.n
app.IterProp.k = k;
IterationRun(app);
end
clear app.UIAxes;
%app.UIAxes = [];
app.UIAxes.cla;
yyaxis(app.UIAxes,'left')
plot(app.UIAxes,app.ModelResults.t/3600,app.ModelResults.Tvsl)
hold(app.UIAxes,'on')
plot(app.UIAxes,app.ModelResults.t/3600,app.ModelResults.Tjkt)
plot(app.UIAxes,app.ModelResults.t/3600,app.ModelResults.VslOut)
ylim(app.UIAxes, [min(app.ModelResults.Tjkt) max(app.ModelResults.Tjkt)])
yyaxis(app.UIAxes,'right')

    plot(app.UIAxes,app.ModelResults.t/3600,app.ModelResults.TCUPID)
    ylim(app.UIAxes,[min(app.ModelResults.TCUPID) max(app.ModelResults.TCUPID)]);
    xlim(app.UIAxes,[0,app.ModelResults.t(end)/3600])
    hold(app.UIAxes,'off')
    legend(app.UIAxes,'Tvsl Model','Tjkt Model','Vsl PID Out','TCU PID Out')
    if app.ModelResults.t50 ~=0
      app.t_50EditField.Value = app.ModelResults.t50;
    end
    if app.ModelResults.t90 ~=0
      app.t_90EditField.Value = app.ModelResults.t90;
    end
    if app.ModelResults.t95 ~=0
      app.t_95EditField.Value = app.ModelResults.t95;
    end
    if app.ModelResults.t99 ~=0
      app.t_99EditField.Value = app.ModelResults.t99;
    end
    if app.ModelResults.Overshoot ~=0
      app.OvershootEditField.Value = app.ModelResults.Overshoot;
    end
    app.AverageUEditField.Value = mean(app.ModelResults.U);
end
```

```
function [] = InitialProp(app)

%Set initial variables for first iteration
%Call functions to set Vsl,TCU,and PID values from GUI
% VslPropSet(app)
% TcuPropSet(app)
% PID_update(app)
app.ModelResults = []; %Clear out previous results
app.ModelResults.PTCU(1) = 0;
app.ModelResults.t99 =0;
app.ModelResults.t95 =0;
app.ModelResults.t90 =0;
app.ModelResults.t50 =0;
app.ModelResults.Overshoot =0;
    app.IterProp.SP = 0; %Flag if *Setpoint has been reached


    T0 = app.T0EditField.Value;
    SP = app.SetPointEditField.Value;
    %Start ARW off

    app.IterProp.ARW =0;
    app.IterProp.SP = 0;
    app.IterProp.VslIntegral=0;
    app.IterProp.TCUIntegral=0;
    app.IterProp.VslOut = T0;
    app.IterProp.VslOutPrev = T0;
    app.IterProp.VslOutPrevPrev = T0;
    app.IterProp.VslErrPrev = 0;
    app.IterProp.VslErr = 0;
    app.IterProp.VslErrPrevPrev = 0;
    app.IterProp.TCUOUT = 0;
    app.IterProp.PTCUPrev = 0;
    app.IterProp.TCUerrPrev = 0;
    app.IterProp.TCUerr = 0;

    %Set Run Properties
    app.IterProp.T0 = app.T0EditField.Value;
    app.IterProp.SPVsl = app.SetPointEditField.Value;
    app.IterProp.Tvsl = T0;
    app.IterProp.Tjkt = T0;
    app.IterProp.TjktPrev = T0;
    app.IterProp.TjktPrevPrev = T0;
    app.IterProp.s=2;
    %app.IterProp.

    %Find ROM required Energy
```

```
    E = app.Vsl.Vol*app.Vsl.Cp*(abs(T0-SP));
    P = app.TCU.Heat;
    EstTime = E/P;
    LengthMultiply = app.TimeRatioEditField.Value;
    RunTime = LengthMultiply*EstTime;
    dt = app.TimestepEditField.Value;%seconds
    timesteps = floor(RunTime/dt); % n

    app.IterProp.dt = dt;
    app.IterProp.n = timesteps;

end

function [] = IterationRun(app)

    %Get fluid Properties for iteration
    FluidProp(app);
    ARW(app);
    %Get Vessel PID
    PIDVslSpOut(app);
     %Get U value
    Uvalue(app);

    %Get TCU PID
    if app.LaudaButton.Value == 1
       PIDLaudaPowOut(app);
    elseif app.SterlingButton.Value == 1
       PIDSterlingPowOut(app);
    elseif app.ThermoflexButton.Value == 1
       PIDThermoflexPowOut(app);
    elseif app.MokonButton.Value == 1
       PIDMokonPowOut(app);
    end
    %Heat Transfer
    HeatTransfer(app);

    %Record and update values
    StoreData(app);

End

function [] = FluidProp(app)

    T = app.IterProp.Tjkt;
    PerGlycol = app.TCU.PerGlycol/100;
```

```
        Cp=4186.8*(-1.33452102949406 * PerGlycol/(T+273.15) + (-5.55637489691179E-06)*(T+273.15) +
1.04101341331825);

        w1=PerGlycol;
        w2=1-PerGlycol;
        Ac(1,1)=1003.7;Ac(2,1)=147.12;Ac(3,1)=-99.617;Ac(1,2)=-.20062;
        Ac(2,2)=-1.1024;Ac(3,2)=0.63102;Ac(1,3)=-2.5127*10^-3;
        Ac(2,3)=2.6902 * 10^-3; Ac(3,3)=-1.1267*10^-3;
        B=[-3.9701 1000.8 -104.1 1.5232 -5.0007 9.8106E-4 3.2452];
        C=[0.19116 1.1999E-4 -9.2459E-7 0.3622 9.0345E-2 -2.0935E-4];

        %eta1=B(1)+B(2)*(T+273)+B(3)*(T+273)^2;
        eta1=0.000213196*T^2-0.06436891*T+5.2411756;
        eta2=-3.758023 + 590.9808/(T+137.2645);
        T_abs = T+273.15;
        k_water=-8.354e-6*T_abs^2+6.53e-3*T_abs-0.5981;
        lambda1=k_water;%C(1)+C(2)/(T-C(3));
        %lambda1=C(1)+C(2)/(T-C(3));
        lambda2=0.570990+0.167156E-2*T-0.609054E-5*T^2;
        eta=w1*eta1+w2*eta2+(eta1-eta2)*w1*w2*(B(4)+B(5)*w1+B(6)*T+B(7)*w1^2);
        mu=exp(eta)/1000;
        k=PerGlycol*lambda1+(1-PerGlycol)*lambda2+(lambda1-lambda2)*PerGlycol*(1-
PerGlycol)*(C(4)+C(5)*PerGlycol+C(6)*T);
        %density
        sum=0;
        for z=1:3
          for j=1:3
            sum=sum+Ac(z,j)*PerGlycol^(z-1)*T^(j-1);
          end
        end
        rho=sum;
        sum=0;

        %Set Properties for iteration
        app.IterProp.JktCp = Cp;
        app.IterProp.JktMu = mu;
        app.IterProp.JktRho = rho;
        app.IterProp.JktK = k;
        app.IterProp.JktPr = Cp*mu/k;

    end

function [] = ARW(app)

        % ---------------------------------- ARWvsl preparation ------------------------------ %
        % Define PIDTvslSPOUT_Prev_Prev
```

```
        app.IterProp.VslOutPrevPrev = app.IterProp.VslOutPrev;% index out previous previous PIDTvslSPOUT from Tracker
        app.IterProp.VslOutPrev = app.IterProp.VslOut;

        % Determine Satvsl_Prev_Prev
        if app.IterProp.VslOutPrevPrev == app.PID.Vsl.OUTH || app.IterProp.VslOutPrevPrev == app.PID.Vsl.OUTL % Check the
PIDTvslSPOUT value
          % from the iteration before last
          app.IterProp.SatPrevPrev= 1; % PIDTCUpowOUT was saturated
        else
          app.IterProp.SatPrevPrev = 0; % PIDTCUpowOUT was not saturated
        end

        % Determine Satvsl_Prev

        if app.IterProp.VslOutPrev == app.PID.Vsl.OUTH || app.IterProp.VslOutPrev == app.PID.Vsl.OUTL % Check the
PIDTvslSPOUT value from
          % the last iteration
          app.IterProp.SatPrev = 1; % PIDTCUpowOUT is still saturated
        else
          app.IterProp.SatPrev = 0; % PIDTCUpowOUT is not saturated
        end

        % ARWvsl Detection
        if app.IterProp.SatPrevPrev == 1 && app.IterProp.SatPrev == 0 % Is the PIDTvslSPOUT value coming away from
saturation?
          if app.IterProp.VslOutPrev >= app.PID.Vsl.ARWH || app.IterProp.VslOutPrev <= app.PID.Vsl.ARWL % Is the
PIDTvslSPOUT
            % value outside the ARW limits?
            app.IterProp.ARW = 1; % Agressive PID functions
          else
            app.IterProp.ARW = 0; % Normal PID functions
          end
        elseif app.IterProp.ARW == 1 % Did the PIDTvslSPOUT value just come away from saturation
          % and is still outside the ARW limits?
          if app.IterProp.VslOutPrev >= app.PID.Vsl.ARWH || app.IterProp.VslOutPrev <= app.PID.Vsl.ARWL % Is the
PIDTvslSPOUT
            % value outside the ARW limits?
            app.IterProp.ARW = 1; % Agressive PID functions
          else
            app.IterProp.ARW = 0; % Normal PID functions
          end
        else
          app.IterProp.ARW = 0; % Normal PID functions
        end
      end
```

```
function [] = PIDVslSpOut(app)

app.IterProp.VslErrPrevPrev = app.IterProp.VslErrPrev;
app.IterProp.VslErrPrev = app.IterProp.VslErr;
app.IterProp.VslErr = -app.IterProp.Tvsl+app.IterProp.SPVsl;
if(app.DeltaVButton.Value ==0)
app.IterProp.VslOut = app.IterProp.SPVsl;
else
        e_0 = app.IterProp.VslErr;
        e_1 = app.IterProp.VslErrPrev;
        e_2 = app.IterProp.VslErrPrevPrev;
        dt = app.IterProp.dt;
        %Set appropiate gains for gain scheduling
        %Check what gain scheduling limits
        Reff = app.IterProp.Tvsl-app.IterProp.SPVsl;


        %Set appropiate P,I,D values for scheduling
        if Reff < app.PID.Vsl.R1R2
          P = app.PID.Vsl.PR1;
          I = app.PID.Vsl.IR1;
          D = app.PID.Vsl.DR1;

        elseif Reff < app.PID.Vsl.R2R3
          P = app.PID.Vsl.PR2;
          I = app.PID.Vsl.IR2;
          D = app.PID.Vsl.DR2;
        else
          P = app.PID.Vsl.PR3;
          I = app.PID.Vsl.IR3;
          D = app.PID.Vsl.DR3;
        end
        ARW = app.IterProp.ARW  * 15 +1;
        %Trying it the non-velocity form
        app.IterProp.VslDerivative = D*(e_0-e_1)/dt;
        app.IterProp.VslIntegral = app.IterProp.VslIntegral + e_0*dt/I/ARW;
        app.IterProp.VslOut = P*e_0 + app.IterProp.SPVsl + app.IterProp.VslIntegral + app.IterProp.VslDerivative;

        %Confirm output is within limits
        if app.IterProp.VslOut > app.PID.Vsl.OUTH
          app.IterProp.VslOut = app.PID.Vsl.OUTH;
          app.IterProp.VslIntegral = app.IterProp.VslIntegral - e_0*dt/I/ARW; %removing Integral windup
        elseif app.IterProp.VslOut < app.PID.Vsl.OUTL
          app.IterProp.VslOut = app.PID.Vsl.OUTL;
          app.IterProp.VslIntegral = app.IterProp.VslIntegral - e_0*dt/I/ARW; %removing integral windup
        end
      end
```

```
    %Testing
    %app.IterProp.VslOut = app.IterProp.SPVsl;
end %Function PIDVslSPOut
```

```
function [] = Uvalue(app)

%Function to calculate Overall Heat Transfer Coefficient between Vsl and Jkt
%Finds flow/pressure drop through iteration
L = app.Vsl.JktA/app.Vsl.JktW;
R_pp=.000939+7.26E-5+7.45E-5+.001+.000514+3.69E-6;%Thermal resistance of all but jkt fluid
R_pp = app.PEthylLEditField.Value/app.PEthylkEditField.Value + ...
app.PEstLEditField.Value/app.PEstkEditField.Value + ...
app.EVOHLEditField.Value/app.EVOHkEditField.Value + ...
app.SS316LEditField.Value/app.SS316kEditField.Value + ...
1/app.VesselhEditField.Value;
%Initial Pressure guess
dP = 6894*1.301*L;
%Initial Flow guess
s = app.IterProp.s;% m/s
%hydaulic Diameter
Dh = 0.00924;%Meters
Q=s*app.Vsl.JktAc;%m^3 per second
Ac = app.Vsl.JktAc;
Ac_star = app.Vsl.JktAc - 0.0001845;
%Testing to see if using Dh = Ac^0.5
Dh = sqrt(app.Vsl.JktAc);
Re=app.IterProp.JktRho*s*Dh/(app.IterProp.JktMu/5);
f=0.25/(log10(0.0015/3.7+5.74/Re^0.9))^2;%Swamee-Jain Equation
% if f >0.1
% f=0.1;
% end
if Re <2300
f=96/Re;
end
%f=0.03;
for j=1:4
for i = 1:3
[dP,Q] = FlowIter(app,Dh,dP,f,L,app.IterProp.JktRho,Q);
        zQLM = Q*1000*60;
        zdPbar =dP/100000;
    end
    s=Q/app.Vsl.JktAc;
    Re=app.IterProp.JktRho*s*Dh/app.IterProp.JktMu;
    f=0.25/(log10(0.0015/3.7+5.74/Re^0.9))^2;%Swamee-Jain Equation
    %       if f >0.1
    %          f=0.1;
    %       end
```

```
    if Re <2300
      f=96/Re;
    end

  end


  app.IterProp.dP = dP;
  app.IterProp.Re = Re;

  app.IterProp.s = s;
  Pr = app.IterProp.JktPr;

  x1 = 1.008;
  x2 = 1.08;
  x3 = 12.39;
  %Taking into account the dimples
  s=Q/Ac_star;
  Re=app.IterProp.JktRho*s*Dh/app.IterProp.JktMu;
  app.IterProp.Re = Re;
  Nu = 7.54 + ((f/8)*(Re-2300))*Pr^x1/(x2+x3*sqrt(f/8)*(Pr^(2/3)-1));
  %Nu = 4.86 + ((f/8)*(Re-2300))*Pr^x1/(x2+x3*sqrt(f/8)*(Pr^(2/3)-1));

  %Nu=(f/8)*(Re-1000)*Pr/(1+12.7*(f/8).^.5.*(Pr^(2/3)-1));

  if Re < 2300
    Nu =7.54;
    %Nu = 4.86;
  end

  app.IterProp.C1 = L*f*app.IterProp.JktRho/(2*app.Vsl.JktAc^2*Dh) ;
  app.IterProp.Nu = Nu;
  h=Nu*app.IterProp.JktK/Dh;
  app.IterProp.Q = Q;
  app.IterProp.Mdot = Q/60/1000*app.IterProp.JktRho;%kg/s
  U=(1/h+R_pp).^-1;
  app.IterProp.h = h;
  app.IterProp.U=(1/h+R_pp).^-1;

  if U <= 0
    pause = 3
  end
end% Function Uvalue
```

```
function [dP,Q] = FlowIter(app,Dh,dP,f,L,rho,Q)
```

```
        dP_new = Pump(app,Q);
        dP = (dP+dP_new)/2;
        Q = flowQ(app,dP,L,Dh,rho,f,Q);

    End
```

```
function [h] = Pump(app,Q)

%Yes, units are funky.
%Q is volume flow rate in gal/min
%p is pump power in Horse power
%h is PSI
if app.MokonButton.Value ==1
p = app.TCU.Pump;
a = -0.0001 * p^2 + 0.0022*p - 0.0116;
b = 0.0158*p - 0.0116;
c = -0.7212*p^2 + 12.143*p+28.7;
Q=Q*1000*60;%Convert to L/min from m^3/s
Q=Q*0.2641; %Convert L/min to Gal/min
h = a*Q^2 + b*Q + c;
if h<0
h=0;
end
h=h*6894.76; %Converting to Pascals from PSI
app.IterProp.P_pump = h;
end
%Lauda pump curves look a bit different,
if app.LaudaButton.Value ==1
if app.TCU.Pump == 0.25
maxh = 0.9; %Bar
maxq = 28; %L/min
elseif app.TCU.Pump == 0.5
maxh =3.2;
maxq = 37;
else
maxh = 4.8;
maxq = 37;
end
Q=Q*1000*60;%Convert to L/min from m^3/s
        %y = mx+b
        m = -maxh/maxq;
        b = maxh;
        h = m*Q +b;
        if h < 0
          h=0;
        elseif h >maxh
          h=maxh;
        end
```

```
        app.IterProp.P_pump = h*100000;%convert to pa from bar
        h=h*100000;

    elseif app.SterlingButton.Value == 1

        %Need max P, PSI
        %Need max Flow, L/min
        %Need Flow at max P, PSI
        index = str2num(app.SterlingPump.Value);

        hall = [32.7, 40,57.8, 63.5, 66.3,78.6,80.7];
        qall = [47, 53,65.3, 74.9,97,104.9,127.5];
        h_qall = [0,0,0,0,0,0,0];%[11.6, 12.7,22.6, 27.4,43.9,56.4,49];

        m = (h_qall(index)-hall(index))/(qall(index));
        b = hall(index);
        Q=Q*1000*60*0.2641;%convert from m^3/s to gal/min
        h = m*Q+b;
        app.IterProp.P_pump = h*6894.76; %Convert to pascal
        h = h*6894.76;
    elseif app.ThermoflexButton.Value == 1
        index = str2num(app.ThermoflexPump.Value);
        hall=[3,5,6];%Bar
        qall = [56,80,130];%L/min
        %Q=Q*1000*60;%Convert to L/min from m^3/s
        %y = mx+b


        maxh = 4;
        maxq = 60;
        Q=Q*1000*60;%Convert to L/min from m^3/s
        %y = mx+b
        m = -maxh/maxq;
        b = maxh;
        h = m*Q +b;
        if h < 0
            h=0;
        elseif h >maxh
            h=maxh;
        end
        app.IterProp.P_pump = h*100000;%convert to pa from bar
        h=h*100000;
    end
end
```

```
function [Q] = flowQ(app,dP,L,Dh,rho,f,Q_old)

    Ac = app.Vsl.JktAc;
    D = app.HoseDiameterEditField.Value; %Cm
    D = D/100;
    L_t = app.HoseLengthEditField.Value;
    Q_new = Ac * (abs(dP*2/rho/f * 1/(16*L_t*Ac^2/D^5/pi^2 + L/Dh)))^0.5;
    %Q_new = Ac * (2*dP*Dh/(L*f*rho));
    % Q_new = Ac*(abs(2*dP*Dh/L/rho/f))^0.5;
    Q = (Q_old+Q_new)/2;
    zQ=Q*1000*60;


    if app.LaudaButton.Value ==1
      if app.TCU.Pump == 0.25

        maxq = 28; %L/min

      elseif app.TCU.Pump == 0.5

        maxq = 37;

      else

        maxq = 37;
      end

      if Q>(maxq/1000/60)
        Q=maxq/1000/60;
        zQ2=Q*1000*60;
      elseif Q<0.00002
        Q=0.00002;
      end
    end

  end
function [] = PIDLaudaPowOut(app)

%Guide and slave controller outputs
% app.PID.TCU.Kpe
% app.PID.TCU.Tne
% app.PID.TCU.Tde
% app.PID.TCU.Xpf
% app.PID.TCU.Tve
yk = app.IterProp.Tjkt;
yk_1 = app.IterProp.TjktPrev;
yk_2 = app.IterProp.TjktPrevPrev;
```

```
ek = app.IterProp.VslOut-app.IterProp.Tjkt;
ek_prime=ek;
ek_1 = app.IterProp.VslOutPrev-app.IterProp.TjktPrev;
ek_1_prime = ek_1;
if app.TCUONLYButton.Value ==1
ek_prime = app.IterProp.VslOut - app.IterProp.Tvsl;
ek_1_prime = app.IterProp.VslOutPrev - app.IterProp.TvslPrev;
end
Tne = app.PID.TCU.Tne;
Kd = app.PID.TCU.Kpe*(app.PID.TCU.Tde+app.PID.TCU.Tve);
Td = app.PID.TCU.Tde;
Kp = app.PID.TCU.Kpe;
dt = app.IterProp.dt;
xpf = app.PID.TCU.Xpf;
Kp_slave = 100/app.PID.TCU.Xpf;
%Kv = 1;
if app.TCUONLYButton.Value ==1
bias = app.IterProp.VslOut;
else
bias = 0;
end
%Summing integral term
app.IterProp.TCUIntegral = app.IterProp.TCUIntegral + ek_prime;
%pk = controller output from master controller
pk = bias + Kp*(ek_prime + (dt * app.IterProp.TCUIntegral/Tne) - Kd/dt*(ek_1_prime-ek_prime));
%Set previous guide output
app.IterProp.TCUOUT = pk;

    %TCU ONLY - No DeltaV
    if app.TCUONLYButton.Value ==1
      TCU_pow = Kp_slave * (pk - app.IterProp.Tjkt); %Sets a SP for Temp
    else
      %** Slave Controller ** (With deltaV)
      TCU_pow = Kp_slave*(pk);
    end


    % Assign ouput
    PIDTCUpowOUT = TCU_pow;

    % Set output min and max
    PIDTCUpowOUTmin = -100*(-0.0006*yk^2+0.0462*yk+0.3429);%(0.022*(yk)+0.57);
    if PIDTCUpowOUTmin <-100
      PIDTCUpowOUTmin = -100;
    end
    % could be a GUI input
    % (0.022*(yk)+0.57) Is a fit to the cooling potential power based on current jacket temperature
```

```
        PIDTCUpowOUTmax = 100; % could be a GUI input
        % Verify that PIDjktOUT is valid (A.K.A. between the set max and min values)
        if PIDTCUpowOUT>PIDTCUpowOUTmax
           PIDTCUpowOUT = PIDTCUpowOUTmax;
           app.IterProp.TCUIntegral = app.IterProp.TCUIntegral - ek_prime;
        elseif PIDTCUpowOUT<PIDTCUpowOUTmin
           PIDTCUpowOUT = PIDTCUpowOUTmin;
           app.IterProp.TCUIntegral = app.IterProp.TCUIntegral - ek_prime;
        else
           % Do nothing. PIDjktOUT value is valid
        end

        app.IterProp.PTCUPrev = PIDTCUpowOUT; %Storing the final output for the iteration

        if PIDTCUpowOUT >0
           app.IterProp.PTCU = app.TCU.Heat * PIDTCUpowOUT/100;
        else
           app.IterProp.PTCU = app.TCU.Cool * PIDTCUpowOUT/100;
        end

    end%Lauda Power
```

function [] = PIDSterlingPowOut(app)

```
% function PIDTCUpowOUT = PIDTCUpow(pk_1,yk_1,yk,ek_1,ek,deltat,TCUheat_chill,HeatPB,CoolPB,...
% DelivReset,DelivRate,CRatio)
% PIDTCUpow is a PID controller in the velocity form. It takes input data from current
% and previous steps and calulates the target TCU power.
%
if app.TCUONLYButton.Value ==1
app.IterProp.TCUerrPrevPrev = app.IterProp.TCUerrPrev;
app.IterProp.TCUerrPrev = app.IterProp.TCUerr;
app.IterProp.TCUerr = -app.IterProp.Tvsl+app.IterProp.VslOut;
else
app.IterProp.TCUerrPrevPrev = app.IterProp.TCUerrPrev;
app.IterProp.TCUerrPrev = app.IterProp.TCUerr;
app.IterProp.TCUerr = -app.IterProp.Tjkt+app.IterProp.VslOut;
end
% INPUTS:
deltat = app.IterProp.dt;
pk_1 = app.IterProp.PTCUPrev;
ek_2 = app.IterProp.TCUerrPrevPrev;
ek_1 = app.IterProp.TCUerrPrev;
ek = app.IterProp.TCUerr;
yk = app.IterProp.Tjkt;
yk_1 = app.IterProp.TjktPrev;
if app.IterProp.PTCUPrev< 0
```

```
app.IterProp.Chill = 1;
Kc = 100/app.PID.TCU.CoolPB;
else
app.IterProp.Chill = 0;
Kc = 100/app.PID.TCU.HeatPB;
end
TauI = app.PID.TCU.Reset;
TauD = app.PID.TCU.Rate;
% Set output min and max
PIDTCUpowOUTmin = -100; % could be a GUI input
PIDTCUpowOUTmax = 100; % could be a GUI input
% If structure so NaN isn't returned by the D portion of PID due to dividing by 0
%Was velocity form, now using standard form
% if TauD == 0
% pk = pk_1 + Kc * ((ek-ek_1) + (deltat/TauI)*ek);
% else
% pk = pk_1 + Kc * ((ek-ek_1) + (deltat/TauI)*ek - (TauD/deltat)*(yk - yk_1));
% end
app.IterProp.TCUIntegral = app.IterProp.TCUIntegral + ek;%Not true integral, only summing iterationi error. Still needs time
portion
pk = app.IterProp.VslOut + Kc*(ek + deltat*app.IterProp.TCUIntegral/TauI -TauD/deltat*(yk - yk_1));
% Assign ouput
PIDTCUpowOUT = pk;
% Verify that PIDjktOUT is valid (A.K.A. between the set max and min values)
if PIDTCUpowOUT>PIDTCUpowOUTmax
PIDTCUpowOUT = PIDTCUpowOUTmax;
app.IterProp.TCUIntegral = app.IterProp.TCUIntegral - ek;
elseif PIDTCUpowOUT<PIDTCUpowOUTmin
PIDTCUpowOUT = PIDTCUpowOUTmin;
app.IterProp.TCUIntegral = app.IterProp.TCUIntegral - ek;
end
app.IterProp.PTCUPrev = PIDTCUpowOUT;



    if PIDTCUpowOUT <0

        mu_f=app.IterProp.JktMu*1000; %converting to centistokes
        factor=(0.8813*mu_f+.18);
        dP_house = (app.TCU.HPSI_In - app.TCU.HPSI_Out)*6984.76/100000; %House dP in Bar from psi

        kv = app.TCU.kv / factor;

        Q2 = (kv*(dP_house)^0.5)/3600;

        %This is a mess, see my thesis for full explanation
        %Essentially, assumption that flow through valve is small
        %compared to flow through jacket
```

```
            %Solves for flow through valve based on kv

            %Q2 = Q2_pos;%sqrt(abs(C2/(2*C1*Q1+C1+C3)));
            mexMax=Q2 * app.IterProp.JktRho / app.PID.TCU.CoolRatio;
            app.IterProp.mexMax = mexMax;

            Mjkt = (app.TCU.Vol+app.Vsl.JktVol)*app.IterProp.JktRho/1000;

            qJktVsl = app.IterProp.U*app.Vsl.JktA*(app.IterProp.Tjkt-app.IterProp.Tvsl);
            app.IterProp.PTCU = -mexMax*PIDTCUpowOUT/100*app.IterProp.JktCp*(app.TCU.Thouse-(app.IterProp.Tjkt-
(qJktVsl/Mjkt/app.IterProp.JktCp)));%(Positive is heat -> Jkt)

        else
            app.IterProp.PTCU = app.TCU.Heat*PIDTCUpowOUT/100;
        end

    end %End Sterling
```

function [] = PIDThermoflexPowOut(app)

```
%Guide and slave controller outputs
% app.PID.TCU.CoolP = app.CoolPEditField.Value;
% app.PID.TCU.HeatP = app.HeatPEditField.Value;
% app.PID.TCU.CoolI = app.CoolIrepeatsminEditField.Value;
% app.PID.TCU.HeatI = app.HeatIrepeatsminEditField.Value;
% app.PID.TCU.CoolD = app.CoolDminEditField.Value;
% app.PID.TCU.HeatD = app.HeatDminEditField.Value;
app.IterProp.TCUprev = app.IterProp.TCUOUT;
if app.IterProp.TCUprev < 0
            P = app.PID.TCU.CoolP;
            I = app.PID.TCU.CoolI;
            D = app.PID.TCU.CoolD;
        else
            P = app.PID.TCU.HeatP;
            I = app.PID.TCU.HeatI;
            D = app.PID.TCU.HeatD;
        end

        yk = app.IterProp.Tjkt;
        yk_1 = app.IterProp.TjktPrev;
        yk_2 = app.IterProp.TjktPrevPrev;
        ek = app.IterProp.VslOut-app.IterProp.Tjkt;
        ek_1 = app.IterProp.VslOutPrev-app.IterProp.TjktPrev;
        dt = app.IterProp.dt;
        bias = 0 ;
        %Using C type from http://bestune.50megs.com/typeABC.htm
        %pk = app.IterProp.TCUprev - P * (yk-yk_1) + I*dt*ek - D/dt*(yk-2*yk_1+yk_2);
```

```
        if app.TCUONLYButton.Value ==1
            ek = app.IterProp.VslOut-app.IterProp.Tvsl;
            ek_1 = app.IterProp.VslOutPrev - app.IterProp.TvslPrev;
            bias = app.IterProp.VslOut;
        end

        app.IterProp.TCUIntegral = app.IterProp.TCUIntegral + ek;%Not true integral, only summing iterationi error. Still needs
time portion

        pk = bias + (100/P)*(ek) + I*(dt*app.IterProp.TCUIntegral)-(D/dt)*(ek_1-ek);

        %

        % Assign ouput
        PIDTCUpowOUT = pk;

        % Set output min and max
        PIDTCUpowOUTmin = -100*(-0.0006*yk^2+0.0462*yk+0.3429); % could be a GUI input
        % (0.022*(yk)+0.57) Is a fit to the cooling potential power based on current jacket temperature

        PIDTCUpowOUTmax = 100; % could be a GUI input
        % Verify that PIDjktOUT is valid (A.K.A. between the set max and min values)
        if PIDTCUpowOUT>PIDTCUpowOUTmax
            app.IterProp.TCUIntegral = app.IterProp.TCUIntegral - ek; %Holding integral term if saturated
            PIDTCUpowOUT = PIDTCUpowOUTmax;
        elseif PIDTCUpowOUT<PIDTCUpowOUTmin
            app.IterProp.TCUIntegral = app.IterProp.TCUIntegral - ek; %Holding integral term if saturated
            PIDTCUpowOUT = PIDTCUpowOUTmin;
        else
            % Do nothing. PIDjktOUT value is valid
        end

        app.IterProp.TCUOUT = PIDTCUpowOUT;

        if PIDTCUpowOUT >0
            app.IterProp.PTCU = app.TCU.Heat * PIDTCUpowOUT/100;
        else
            app.IterProp.PTCU = app.TCU.Cool * PIDTCUpowOUT/100;
        end

        app.IterProp.PTCUPrev = PIDTCUpowOUT; %Store for tracker


    end %End thermoflex PID
```

```
function [] = HeatTransfer(app)

    %Function to transfer energy from TCCU->Jkt->Vsl
    %called after Uvalues and TCU Power out are known

    %dtJkt = app.IterProp.PTCU/(app.IterProp.JktCp*app.IterProp.Mdot);
    %qTCUJkt = app.IterProp.PTCU; %(+ is heat flowing to Jkt)
    qTCUJkt = app.ModelResults.PTCU(app.IterProp.k);

    qJktVsl = app.IterProp.U*app.Vsl.JktA*(app.IterProp.Tjkt-app.IterProp.Tvsl)*app.Vsl.PercentFull/100;%q jacket to vsl (+ is heat flowing to Vessel)
    app.IterProp.Qjktvsl= qJktVsl;
    qJktAmbient = 5*app.Vsl.JktA*(app.IterProp.Tjkt-app.AmbientCEditField.Value);
    dT_vsl = app.IterProp.dt*qJktVsl/(app.Vsl.Cp*app.Vsl.Vol*app.PercentFullEditField.Value/100);


    dT_jkt = app.IterProp.dt*(qTCUJkt-qJktVsl-qJktAmbient)/(app.IterProp.JktCp*(app.TCU.Vol+app.Vsl.JktVol)/1000*app.IterProp.JktRho);
    app.IterProp.Tjkt_new = app.IterProp.Tjkt+dT_jkt;
    app.IterProp.Tvsl_new = app.IterProp.Tvsl+dT_vsl;
  end

function [] = StoreData(app)

    %Updates app.IterProp and app.ModelResults
    k = app.IterProp.k;
    app.IterProp.TjktPrevPrev = app.IterProp.TjktPrev;
    app.IterProp.TjktPrev = app.IterProp.Tjkt;
    app.IterProp.TvslPrev = app.IterProp.Tvsl;
    app.IterProp.Tjkt=app.IterProp.Tjkt_new;
    app.IterProp.Tvsl=app.IterProp.Tvsl_new;

    app.IterProp.VslOutPrev = app.IterProp.VslOut;

    app.ModelResults.TCUPID(k) = app.IterProp.PTCUPrev;
    app.ModelResults.PTCU(k+1) = app.IterProp.PTCU;
    app.ModelResults.VslOut(k) = app.IterProp.VslOut;
    app.ModelResults.Tjkt(k)=app.IterProp.Tjkt;
    app.ModelResults.Tvsl(k)=app.IterProp.Tvsl;
    app.ModelResults.U(k) = app.IterProp.U;
    app.ModelResults.h(k) = app.IterProp.h;
    app.ModelResults.Q(k) = app.IterProp.Q;
    app.ModelResults.t(k) = app.IterProp.dt*k;
    app.ModelResults.dP(k) = app.IterProp.dP;
    app.ModelResults.Nu(k) = app.IterProp.Nu;
    app.ModelResults.Qjktvsl(k) = app.IterProp.Qjktvsl;
    app.ModelResults.Pr(k) = app.IterProp.JktPr;
```

```
        app.ModelResults.Re(k) = app.IterProp.Re;
        %app.ModelResults

        %       app.IterProp.VslErrPrevPrev = app.IterProp.VslErrPrev;
        %       app.IterProp.VslErrPrev = app.IterProp.VslErr;
        %       app.IterProp.VslErr = app.IterProp.Tvsl-app.IterProp.SPVsl;

        %Find milestones
        t_current = app.IterProp.dt*k/3600;%Hours
        SP = app.SetPointEditField.Value;
        T0 = app.T0EditField.Value;
        dT = SP-T0;
        Ti = app.IterProp.Tvsl;
        sign = dT/abs(dT);
        dTi = Ti-T0;

        if  sign*dTi > sign*dT*0.5 && app.ModelResults.t50 ==0
            app.ModelResults.t50 = t_current;
        end
        if  sign*dTi > sign*dT*0.90 && app.ModelResults.t90 ==0
            app.ModelResults.t90 = t_current;
        end
        if  sign*dTi > sign*dT*0.95 && app.ModelResults.t95 ==0
            app.ModelResults.t95 = t_current;
        end
        if  sign*dTi > sign*dT*0.99 && app.ModelResults.t99 ==0
            app.ModelResults.t99 = t_current;
        end
        if sign*dTi>sign*dT
            if ((Ti-SP)*sign) > app.ModelResults.Overshoot
                app.ModelResults.Overshoot = (Ti-SP)*sign;
            end
        end

    end
```

Function to Compare Experiment and Model Data

```
%script to compare
filename2 = 'columns2.xls';
%functions to write
load('AllRuns.mat');
%truncate to same length
%standard error

%percentage off on t50,90,95,99
%maximum overshoot
%exp: matrix with 1:time, 2:T_vsl, 3:Tjkt_in, 4:Tjkt_out, 5:P_out, 5:Pin
for setup_iter = 1:7
   MULT =1; %Number to ensure same indexing
```

```
sv = 0.2;
si = 0.2;
so = 0.2;
scp = 0;


T0name=[5,37,20,5,20,37];
SPname=[37,20,5,20,37,5];
cp = 4188;

PRESSURE =1;

% %500 SUB w/ TF 5000

% %exp = run500SUBThermoflex_4_31pt8; %1
% %exp = run500SUBThermoflex_31pt8_18pt5; %2
% %exp = run500SUBThermoflex_18pt5_4; %3
% %exp = run500SUBThermoflex_4_18pt6; %4
% %exp = run500SUBThermoflex_18pt6_35pt5; %5
% %exp = run500SUBThermoflex_35pt5_4; %6
if setup_iter ==1
    runtitle='TF5000SUB500';
    run = 1; %Which run (for the sheet)
    Cell =
{run500SUBThermoflex_4_31pt8;run500SUBThermoflex_31pt8_18pt5;run500SUBThermoflex_18pt5_4;run500SUBThermofle
x_4_18pt6;run500SUBThermoflex_18pt6_35pt5;run500SUBThermoflex_35pt5_4;};
    SPall = [31.8,18.5,4,18.6,35.5,4];
    Unit = 0;
    Vol = 500;
    TCUpow = 5000;
    TCUname = 2;
    m = 500; %kg
    A = 2.43;
end

% % % 50 FLEX

% run50SUBold_18pt5_5,
% run50SUBold_19pt7_36pt5,
% run50SUBold_35_19,
% run50SUBold_37_5pt5,
% run50SUBold_3pt5_35,
% % run50SUBold_5_20
if setup_iter ==2
    runtitle='LVC1200FLEX50';
    run = 2; %Which run (for the sheet)
    Cell =
{run50flexVC1200_3pt5_35;run50flexVC1200_35_19;run50flexVC1200_18pt5_5;run50flexVC1200_5_20;run50flexVC1200_19
pt7_36pt5;run50flexVC1200_37_5pt5};
    SPall = [35,19,5,20,36.5,5.5];
    Unit = 0;
    Vol = 50;
    TCUpow = 1200;
    TCUname = 0;
    m = 50; %kg
    A = .41;
    PRESSURE =1;
end
% % % 50 SUB Updated

% % run50SUBupdated_6_37pt4,
% % run50SUBupdated_37pt4_20pt6,
% % run50SUBupdated_20pt6_5pt8,
% % run50SUBupdated_6pt2_20pt6
% % run50SUBupdated_20pt7_37pt5,
```

```
% % run50SUBupdated_37pt5_5pt8,
if setup_iter ==3
    Vol = 50;
    runtitle='LVC1200SUB50update';
    run = 3; %Which run (for the sheet)
    Cell =
{LVC1200SUB50update_6_37pt4;LVC1200SUB50update_37pt4_20pt6;LVC1200SUB50update_20pt6_5pt8;LVC1200SUB50upd
ate_6pt2_20pt6;LVC1200SUB50update_20pt7_37pt5;LVC1200SUB50update_37pt5_5pt8};
    SPall = [37.4,20.6,5.8,20.6,37.5,5.8];
    Unit = 0;
    TCUpow = 1200;
    TCUname = 0;
    m = 50; %kg
    A = .41;
end



% % % 500 SUM Lauda VC2000

% % run500SUM_20_37
% % run500SUM_20_5
% % run500SUM_35_20
% % run500SUM_37_5
% % run500SUM_5_20
% % run500SUM_7_35
if setup_iter ==4
    runtitle='LVC2000SUM500';
    run = 4;
    Cell = {run500SUM_7_35;run500SUM_35_20;run500SUM_20_5; run500SUM_5_20;run500SUM_20_37; run500SUM_37_5};
    SPall = [35,20,5.28,20,37,5];
    Unit = 1;
    Vol = 500;
    TCUpow = 2000;
    TCUname = 0;
    m = 500; %kg
    A = 2.2;
end


%%% Sterling 18k
% run2kSUB18kSterling_19pt8_36pt4,
% run2kSUB18kSterling_19pt8_5pt2,
% run2kSUB18kSterling_36pt4_19pt8
% run2kSUB18kSterling_36pt4_5pt1,
% run2kSUB18kSterling_5pt1_36pt4,
% run2kSUB18kSterling_5pt2_19pt8
if setup_iter ==5
    runtitle = 'S18K2kSUB';
    run = 5;
    Cell =
{run2kSUB18kSterling_5pt1_36pt4;run2kSUB18kSterling_36pt4_19pt8;run2kSUB18kSterling_19pt8_5pt2;run2kSUB18kSterli
ng_5pt2_19pt8;run2kSUB18kSterling_19pt8_36pt4;run2kSUB18kSterling_36pt4_5pt1};
    SPall = [36.4, 19.8, 5.2, 19.8, 36.4, 5.1];
    SPstart=[5.1,  36.4, 19.8,5.1,  19.8, 36.4];
    MULT = 20;
    Unit = 0;
    Vol = 2000;
    TCUpow = 18000;
    TCUname = 1;
    m = 2000; %kg
    A = 4.54;
end

% % Sterling 9k 2k Impulse
```

```
    % % run2kimpulse_neg2_18pt5_35pt2,
    % % run2kimpulse_neg2_18pt5_4,
    % % run2kimpulse_neg2_35pt2_18pt5,
    % % run2kimpulse_neg2_35pt2_4,
    % % run2kimpulse_neg2_4_18pt5,
    % % run2kimpulse_neg2_4_35pt2
    if setup_iter ==6
        runtitle = 'S9kImpulse2k';
        run = 6;
        Cell =
{run2kimpulse_neg2_4_35pt2;run2kimpulse_neg2_35pt2_18pt5;run2kimpulse_neg2_18pt5_4;run2kimpulse_neg2_4_18pt5;ru
n2kimpulse_neg2_18pt5_35pt2;run2kimpulse_neg2_35pt2_4};
        SPall = [35.2, 18.5, 4,   18.5, 35.2, 4];
        SPstart=[4,    35.2, 18.5,4,    18.5, 35.2];
        MULT=20;
        Unit = 3;
        Vol = 2000;
        TCUpow = 9000;
        TCUname = 1;
        m = 2000; %kg
        A = 5.38;
    end


    % % run300SUF9kSterling_20_37
    % % run300SUF9kSterling_20_6,
    % % run300SUF9kSterling_37_20
    % % , run300SUF9kSterling_37_6
    % % , run300SUF9kSterling_6_20
    % % , run300SUF9kSterling_6_37
    if setup_iter ==7
        runtitle = '300SUF';
        run = 7;
        Cell =
{run300SUF9kSterling_6_37;run300SUF9kSterling_37_20;run300SUF9kSterling_20_6;run300SUF9kSterling_6_20;run300SUF
9kSterling_20_37;run300SUF9kSterling_37_6};
        SPall = [37, 20, 6.2,   20, 37, 6];
        SPstart=[6,    37, 20,6,    20, 37];
        MULT=20;
        Unit = 2;
        Vol = 300;
        TCUpow = 9000;
        TCUname = 1;
        unit = 'SUF';
        m = 300; %kg
        A = 1.26;
    end



    runnameall = [32;-17;-15;15;17;-32];

    Umod_cols = [];
    Tjktmod_cols = [];
    U_col = [];
    LMTD_col = [];
    Tjkto_col = [];
    Tvsl_col = [];
    Tjkti_col = [];
    VslGrad_col = [];


    for z=1:6
        ramp =z; %Which ramp on the run (6 total)
```

```
modelname = 'run'+string(run) + '_'+string(z)+'.mat';
load(modelname);


Exp = Cell(ramp);
Exp = cell2mat(Exp);

%fix for 500 SUM VC2000
%    inter = exp;
%    exp(:,2 )=inter(:, 4); %Tvsl =
%    exp(:,3) = inter(:,3); %Tjkti =
%    exp(:,4) = inter(:,2); %Tjkto =
%

% Finding U and q values
test = Exp;
time =test(:,1)*3600;
dt = time(2,1)-time(1,1);
dt_o = 180;
Tvsl = test(:,2);
%Test to see

Tjkti = test(:,3);
Tjkto = test(:,4);

Tjkti_r = Resample(dt,dt_o,Tjkti)';
Tjkto_r = Resample(dt,dt_o,Tjkto)';
Tvsl_r = Resample(dt,dt_o,Tvsl)';
time_r = 0:dt_o/3600:((length(Tvsl_r)-1)*dt_o)/3600;
time_r = time_r';
%Tjkti = smoothdata(Tjkti,'Gaussian',round(50/MULT));
%Tjkto =smoothdata(Tjkto,'Gaussian',round(50/MULT));
%Tvsl = smoothdata(Tvsl,'Gaussian',round(50/MULT));
dTa = Tjkti_r - Tvsl_r;
dTb = Tjkto_r - Tvsl_r;
LMTD = (dTa-dTb)./(log(abs(dTa./dTb))) ;
dTvsl = gradient(Tvsl_r);
Q = m*cp*dTvsl/dt_o;
U = Q./A./LMTD;
U_col = vertcat(U_col,U);
LMTD_col = vertcat(LMTD_col,LMTD);
Tjkto_col = vertcat(Tjkto_col,Tjkto_r);
Tvsl_col = vertcat(Tvsl_col,Tvsl_r);
Tjkti_col = vertcat(Tjkti_col,Tjkti_r);

VslGrad_col = vertcat(VslGrad_col,dTvsl/dt_o);
Utest=U(abs(LMTD) >2);

medU = median(Utest);

% continuing compare program


SP=SPall(ramp);



exp_length = length(Exp(:,1));

model_length = length(ModelResults.Tvsl)/MULT;
model = horzcat((ModelResults.t./3600)',ModelResults.Tvsl',ModelResults.Tjkt',ModelResults.dP');
dt = Exp(2,1)-Exp(1,1);
dt_mod = ModelResults.t(2)./3600 -ModelResults.t(1)./3600;
MULT = round(dt/dt_mod);
n = min(exp_length,model_length);
```

```
modelU = ModelResults.U(abs(ModelResults.Tvsl - ModelResults.Tjkt) > 0.15);
modelTjkt = ModelResults.Tjkt(abs(ModelResults.Tvsl - ModelResults.Tjkt) > 0.15);
Umod_cols = vertcat(Umod_cols,modelU');
Tjktmod_cols = vertcat(Tjktmod_cols,modelTjkt');
modU = median(modelU);


T0 = Exp(1,2);
dT = SP-T0;
sign = dT/abs(dT);
t50=0;t90=0;t95=0;t99=0;Overshoot=0;


sum=0;
k=2;


for i =1:n
   sum = sum + (Exp(i,k)-model(i*MULT,2)).^2 ;
   %Find t _50
   %t_current = app.IterProp.dt*k/3600;%Hours
   Ti = Exp(i,2);
   dTi = Ti-T0;
   if  sign*dTi > sign*dT*0.5 && t50 ==0
      t50 = Exp(i,1);
   end
   if  sign*dTi > sign*dT*0.90 && t90 ==0
      t90 = Exp(i,1);
   end
   if  sign*dTi > sign*dT*0.95 && t95 ==0
      t95 = Exp(i,1);
   end
   if  sign*dTi > sign*dT*0.99 && t99 ==0
      t99 = Exp(i,1);
   end
   if sign*dTi>sign*dT
      if ((Ti-SP)*sign) > Overshoot
         Overshoot = (Ti-SP)*sign;
      end
   end


end
sum=sqrt(sum/n);

plot(model(:,1),model(:,2),'-','LineWidth',1.5,'DisplayName','T_v_s_l Model')
hold on
plot(model(:,1),model(:,3),'-','LineWidth',1.5,'DisplayName','T_j_k_t Model')
plot(Exp(:,1),Exp(:,2),'--','LineWidth',1.5,'DisplayName','T_v_s_l Experiment')
plot(Exp(:,1),Exp(:,4),'--','LineWidth',1.5,'DisplayName','T_j_k_t in Experiment')
plot(Exp(:,1),Exp(:,3),'--','LineWidth',1.5,'DisplayName','T_j_k_t out Experiment')
plot(time_r,LMTD,'--','LineWidth',1.5,'DisplayName','LMTD Experiment')
plot(model(:,1),-model(:,2)+model(:,3),'-','LineWidth',1.5,'DisplayName','LMTD Model')
xlabel('Time, hours')
ylabel('Temperature Celcius')

%       if(PRESSURE == 1)
%          yyaxis right
%          plot(model(:,1),model(:,4)./6894.76,'--','LineWidth',1.5,'DisplayName','\Delta P Model')
%          plot(Exp(:,1),smoothdata(Exp(:,5),'Gaussian',10),'--','LineWidth',1.5,'DisplayName','\Delta P Experiment')
%          ylabel('Pressure (PSI)')
%
%          ylim([0 max( max(model(:,4)/6894.76), max(Exp(:,5)))])
%          yyaxis left
%
```

```
    %       end
    xlim([0 n*dt]);

    legend
    hold off

    title(runtitle +" run: "+string(T0)+' to '+string(SP)+' C')
    filename = runtitle +string(z);%string(T0)+'_'+string(SP);
    savefig(filename+'.fig')
    close
    % %Writing results to excel sheet
    % filename = 'test.xls';

    LetterTranslate = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

    location = LetterTranslate((ramp-1)*3+2)+string((run-1)*8+2);
    location_label = LetterTranslate((ramp-1)*3+2)+string((run-1)*8+1);
    runname = table(runnameall(z));

    %Vars = [t50,t90,t95,t99,Overshoot,sum];
    %ModelVars = [ModelResults.t50,ModelResults.t90,ModelResults.t95,ModelResults.t99,ModelResults.Overshoot,sum];
    %names = [{'t_50'},{'t_90'},{'t_95'},{'t_99'},{'Overshoot'},{'Error'}];
    %percenterror = (Vars-ModelVars)./Vars*100;

    NewTable =
table(Unit,Vol,TCUpow,TCUname,runnameall(z),T0name(z),SPname(z),medU,t50,t90,t95,t99,Overshoot,modU,ModelResults.t
50,ModelResults.t90,ModelResults.t95,ModelResults.t99,ModelResults.Overshoot,sum);
    %t = table(Vars',ModelVars',percenterror','VariableNames',[{'Experiment'},{'Model'},{'PercentError'}]');

    %writetable(t,filename,'Sheet',1,'Range',location)
    %writetable(runname,filename,'Sheet',1,'Range',location_label,'WriteVariableName',false)
    writetable(NewTable,filename2,'Sheet',1,'Range','A'+string(((run-1)*6)+z+1),'WriteVariableName',false);
  end

  %Generate figure for U values vs temperature
  figure(1)
  if setup_iter ==4
    trim = 1;
  elseif setup_iter ==5
    trim = 2;
  elseif setup_iter == 6
    trim = 2;
  else
    trim = 2;
  end


  U_plot = U_col(abs(LMTD_col) >trim & abs(Tjkto_col-Tvsl_col) > 1);
  Tvsl_plot = Tvsl_col(abs(LMTD_col) >trim & abs(Tjkto_col-Tvsl_col) > 1);
  LMTD_plot=LMTD_col(abs(LMTD_col) > trim & abs(Tjkto_col-Tvsl_col) > 1);
  VslGrad_col = VslGrad_col(abs(LMTD_col)>trim & abs(Tjkto_col-Tvsl_col) > 1);
  Tjkti_col = Tjkti_col(abs(LMTD_col)>trim & abs(Tjkto_col-Tvsl_col) > 1);
  Tjkto_col = Tjkto_col(abs(LMTD_col)>trim & abs(Tjkto_col-Tvsl_col) > 1);


error_stddev(U_plot,LMTD_plot+Tvsl_plot,VslGrad_col,Tvsl_plot,Tjkti_col,Tjkto_col,m,A,cp,0.03/dt,sv,si,so,m*0.005,A*0.005,scp
);
  if setup_iter ==4
    close
    open('U_LVC2000SUM500_redo.fig');
  end
  hold on
  scatter(Tjktmod_cols,Umod_cols,'DisplayName','Model U');
  hold off
  title(runtitle)
  ylabel('U Value, Watts/(m^2K)')
```

```
   xlabel('Jacket Temperature')
   legend
   filename = "U_"+runtitle;
   savefig(filename+'.fig')
   close

   %scatter(Tjkt_plot,U_plot);

   names = table("Unit","Vol","TCU Pow","TCU Type","delta
T","T0","SP","MedianU","t50","t90","t95","t99","Overshoot","ModU","t50","t90","t95","t99","Overshoot","Error");
   writetable(names,filename2,'Sheet',1,'Range','A1','WriteVariableName',false)
   close all

end
```

function []=error_stddev(U,LMT,VslGrad,tv,ti,to,m,A,cp,sdvdt,sv,si,so,sm,sA,scp)

```
st_all=[6.314,2.92,2.353,2.132,2.015,1.943,1.895,1.86,1.833,1.812,1.796,1.782,1.771,1.761,1.753,1.746,1.74,1.734,1.729,1.725,
1.7210,1.717,1.714,1.711,1.708,1.706,1.703,1.701,1.699,1.697];
st_inf = 1.645;

space = 5;
T_graph = 5:space:36;
Bias=[];
for i = 1:length(T_graph)
   %i is upper bound of Tjkt being examined

   sum(i)=0;
   numb(i)=0;
   for k = 1:length(LMT)
      if (LMT(k)< T_graph(i)+space/2)
         if (LMT(k) > (T_graph(i)-space/2))
            if U(k) > 0
               sum(i)=sum(i)+U(k);
               numb(i)=numb(i)+1;
            end
         end
      end
   end

   U_avg(i) = sum(i)/numb(i);

   U_stddev(i)=0;
   for k = 1:length(LMT)
      if (LMT(k)<T_graph(i)+space/2)
         if (LMT(k) > (T_graph(i)-space/2))
            if U(k) >0
               U_stddev(i)= U_stddev(i)+(U_avg(i) -U(k))^2;
            end
         end
      end
   end


end
for i = 1:length(T_graph)
   %i is upper bound of Tjkt being examined

   sum(i)=0;
   numb(i)=0;
   for k = 1:length(LMT)
      if (LMT(k)< T_graph(i)+space/2)
         if (LMT(k) > (T_graph(i)-space/2))
            if U(k) > 0
```

```
            if abs(U(k)-U_avg(i)) < 100
               sum(i)=sum(i)+U(k);
               numb(i)=numb(i)+1;
            end
         end
      end
   end
end

U_avg(i) = sum(i)/numb(i);

U_stddev_2(i)=0;
n=0;
Bias(i)=0;
for k = 1:length(LMT)
   if (LMT(k)<T_graph(i)+space/2)
      if (LMT(k) > (T_graph(i)-space/2))
         if U(k) >0
            if abs(U(k)-U_avg(i))< 100
               U_stddev_2(i)= U_stddev_2(i)+(U_avg(i) -U(k))^2;
               Bias(i) = Bias(i)+Ubias(VslGrad(k),tv(k),ti(k),to(k),m,A,cp,sdvdt,sv,si,so,sm,sA,scp);
               n=n+1;
            end
         end
      end
   end
end
U_stddev_2(i)=(U_stddev_2(i)/(n-1))^0.5;
if n <=30
   st(i) = st_all(n);
else
   st(i) = st_inf;
end

Bias(i)=Bias(i)/n;

end



U_uncertain = (st.*U_stddev_2.^2+Bias.^2).^0.5;

errorbar(T_graph,U_avg,U_uncertain,'DisplayName','Experiment U, 2\sigma Uncertainty bars') % error 2 std deviations
end


%calcualte bias uncertainty

function bias = Ubias(dvdt,v,i,o,m,A,cp,sdvdt,sv,si,so,sm,sA,scp)


%variables: Tvsl, Tjktin, Tjktout, mass, cp, Area

ddVsl=(m*cp*dvdt)/(A*(v-i)*(o-v));
ddin=-(m*cp*dvdt*( (i-v)*log(abs((i-v)/(o-v))) -i+o ))/(A*(i-o)^2*(i-v)  );
ddout=-(m*cp*dvdt*( (v-o)*log(abs((i-v)/(o-v))) +i-o ))/(A*(i-o)^2*(o-v)  );
ddm=-(cp*dvdt*log(abs((i-v)/(o-v))))/(A*(i-o));
ddcp=-(m*dvdt*log(abs((i-v)/(o-v))))/(A*(i-o));
ddA=-(m*cp*dvdt*log(abs((i-v)/(o-v))))/(A^2*(i-o));

a=ddVsl*sv;
b=ddin*si;
c=ddout*so;
d=ddm*sm;
e=ddcp*scp;
```

```
f=ddA*sA;
bias = ( a^2+b^2  +c^2 +d^2  +e^2  +f^2 )^0.5;
end
```