ANALYTICAL SENSITIVITY ANALYSIS METHODOLOGY FOR THE CO-DESIGN OF THERMAL MANAGEMENT SYSTEMS

BY

MINDA JOY WAGENMAKER

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Mechanical Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Adviser:

Professor Andrew Alleyne

Abstract

This thesis presents an analytical sensitivity analysis methodology to be used in a co-design approach to simultaneously optimize plant parameter variables and controller gains. Sensitivity analysis methods are commonly used in system design. The sensitivity of the system output to design variables can be calculated either analytically or numerically. These sensitivities inform the engineer which variables should be the focus of the majority of time and effort to yield the optimal result or , conversely, which variables are inconsequential and can be removed from the optimization design space. Sensitivity analysis methods are also used in control system design. The controller design is influenced by the analysis of how system parameters affect the controlled states. This thesis presents a plant and controller co-design approach where the system parameters which affect the plant output and the ones which affect the controlled states the most are identified. These parameters are optimized using a brute-force numerical method at the same time as the controller gains. This improved plant and controller pair is designed to lower the control input without sacrificing performance. By designing both the plant and the controller simultaneously, better results can be achieved than by only optimizing either the plant or controller.

The methodology explored in this thesis is widely applicable to many other systems. However, it has been specifically designed to work with a graph-based modelling framework. This project was carried out with the intention of improving thermal management systems for electrified vehicles. A fluid loop with additional heat loads was used as the example framework to showcase the sensitivity analysis methods. This thesis demonstrates how optimizing the most influential parameters along with the controller gains, lowers the necessary control input, or pump energy, while improving tracking of a reference signal.

To Tim and Sharon Wagenmaker

Acknowledgements

I would like to thank my advisor, Dr. Andrew Alleyne, for his guidance, encouragement, instruction, and support. Thank you for investing in me and helping me develop skills as a researcher and professional. I appreciate all the time and attention you pour into mentoring your students and I'm so grateful to have such a dedicated and excellent advisor.

I also want to thank my family. My parents have been an incredible source of strength and encouragement to me. They have instilled in me the value of life-long learning and have supported me in every endeavour. I want to thank my brother, Peter, and my sister, Amy, for their encouragement as well.

Next, I want to thank the members of Alleyne Research Group I've had the pleasure of working with. First, I want to thank Herschel Pangborn for all the ways he helped me get up to speed during my first year and for how he was always happy take the time to answer my questions or step in and help me out. I also want to thank Chris Aksland for his willingness to help me troubleshoot when I got stuck, for all his explanations of how things work, even if it's not my first time asking, and for taking an interest in my work and asking me how my research is progressing. I want to thank Cary Laird for her willingness to answer questions as well and for her leadership on the SLC. I also want to thank all the other members of ARG, Donald Docimo, Pamela Tannous, Oyuna Angatkina, Ashley Armstrong, Nate Weir, Spencer Igram, who helped me onboard into the lab and made me feel welcome. Finally, I want to thank the remaining current members of ARG, Dylan Charter, Chris Urbanski, Frank Andujar, Kurt Kuipers, Phil Renkert, Kayla Russell, and Reid Smith, for your friendship and support.

I also want to thank Christiaan Hazlett for his constant support and encouragement.

I would also like to thank the faculty, staff, and students of the Center for Power Optimization of Electro Thermal Systems (POETS).

Finally, I would like to acknowledge the support of the National Science Foundation Engineering Research Center for Power Optimization of Electro Thermal Systems (POETS) with cooperative agreement EEC-1449548.

Table of Contents

List of Figures

Chapter 1
Introduction

1.1 Motivation

Sensitivity analysis methods are useful in understanding system models, because they help quantify how dependent the model outputs are on modeling assumptions or characteristics such as parameter values, initial conditions, and inputs. Sensitivity analysis methods can be either numerical or analytical. The finite difference method is a good example of a numerical approach, and involves numerically solving the governing differential equations of a model, while the modeling characteristics are being adjusted to different values. Then the effect had on the resulting model outputs can be compared for each characteristic adjustment to discern which characteristics the model outputs are most sensitive to. On the other hand, an analytical sensitivity analysis technique can use the model's differential equations to analytically determine the dependence of the model output to various modeling characteristics. Analytical methods can be much quicker at calculating sensitivity than numerical methods and may have the potential to do so dynamically as the model itself changes.

The motivation behind developing a sensitivity analysis methodology for graph-based models is to help inform the design process—this information will help the plant and controller designers know which modeling characteristics are insensitive and can be removed from the design optimization problem, and which ones have the largest impact on the system, and thus should be the focus of the majority of time, effort, and money spent on optimization.

1.2 Background

1.2.1 Overview of Sensitivity Analysis Applications

Sensitivity Analysis is a versatile and widely applicable field. The main goal of a sensitivity analysis is usually to either identify the most influential characteristics to focus on optimizing those or to find the least influential characteristics so that they can be discarded.

Examples of situations where the most influential characteristics are sought after include risk mitigation, uncertainty analysis and design optimization. Sensitivity analyses are useful in risk management because they can identify the most significant risk factors within a process. Then those factors can be prioritized to improve them first and thus mitigate the risk as quickly as possible. Additionally, the critical control points (CCP), or the points which have the potential to effect significant change in a system, could also be identified and focused on to achieve the desired results [1]. Sensitivity Analyses are often used in conjunction with Uncertainty Analyses of models. Once the uncertainties of the system outputs are known, a sensitivity analysis can be conducted on those uncertainties with respect to each of the inputs and parameters [2]. The sensitivity information will narrow down and help identify which components of the model should be improved to yield the greatest increase in overall model reliability. Finally, sensitivity analyses are useful in design optimization. Once the design objective is identified, a sensitivity analysis can be applied to discover which system parameters would have the greatest ability to reach that design objective. These few system parameters should then be the focus on the optimization efforts.

There are also some scenarios where the least influential characteristics should be discarded. Sensitivity Analyses are also useful in model reduction for large and complex models [3]. If the main control parameters in the model are identified, they can be used to formulate a smaller model, which can produce similar outputs and requires less computational power and time to run.

As discussed in the motivation section, there are two types of sensitivity analysis: numerical and analytical. The following sections in this chapter will discuss what others have done with sensitivity analysis tools in control system design and plant design optimizations. Most of the sensitivity analysis methods used in these discussions happen to be analytical methods, due to the

existence of linear models for the systems in question. When a linear model does not exist, a slower, less-accurate numerical method might be used, such as the finite-difference method. Regardless of being numerical or analytical, a sensitivity analysis involves calculating (either analytically or numerically) the gradient, or Jacobian, of how changing a particular model characteristic affects the model output. Suppose the model output, Y, is a function of the set of model characteristics, X, as shown in Eq. (1.1)

$$Y = f(X) \tag{1.1}$$

Then the sensitivity index of a particular characteristic, X_j, is shown in Eq. (1.2).

$$Sensitivity\ index \equiv \frac{\partial Y}{\partial X_j} \tag{1.2}$$

To find the most influential characteristic, the sensitivity indices for each characteristic can be compared. Each of the sensitivity indices together form the Jacobian of the model output. The different types of sensitivity analyses and their applications are covered in more detail in Chapter 3. The purpose of this chapter is to introduce the reader to the variety of situations where sensitivity analyses can be applied and get familiar with some examples of others who have applied sensitivity analysis tools to solve control system optimization and plant optimization problems. The final purpose of this chapter is to introduce and motivate opportunities to apply sensitivity analysis techniques to the intersection of control system and plant optimization in the form of a co-design approach.

1.2.2 Sensitivity Analysis in Control Systems

In every control system, there is always a plant which is given an exogenous control input signal to meet requirements, which often involves attaining a target. Thus, the main goal of a controller is to guarantee that the entire range of actual outputs fall within the set of values which satisfy a set of requirements. [4]

This rule can be demonstrated by Eq (1.3), where Y is a vector of generalized outputs from the plant, M_Y is the set of values of Y for which the states of the plant satisfy the requirements specified by the engineer, and R_Y is the set of values of Y which can take place during plant operation. [4]

$$R_Y \subset M_Y \tag{1.3}$$

This equation might be satisfied for the plant without any control signal if all the possible output vectors R_Y are already admissible states and satisfy the requirements for the plant, and thus $R_Y \subset M_Y$. However, once the plant is affected by disturbances this rule might become violated, necessitating outside help from a controller.

Systems cannot be perfectly determined by their state variables and exogenous disturbances alone, they also have factors called parameters, which either effect the properties of the output vector Y directly, indirectly through affecting the properties of the disturbances, or by adjusting the initial conditions. Thus, the output is affected by a set of parameters, which could push it outside of the set of outputs which satisfy the system requirements.

Parameters fall into two categories: technical parameters and parameters of environment and operating conditions. [4] Technical parameters are properties or values which can affect the system output for different example cases under the same operating conditions. An example of a technical parameter would be a physical property of the plant, such as plant size or material properties. Environmental parameters are ones which change the operating conditions of individual example cases, such as changing the properties of the disturbances or varying the initial conditions. An example of an environmental parameter is the ambient temperature or air humidity. Since it is the responsibility of the controller to keep the outputs within the set of allowable values while these technical and environmental parameters could push the system output outside of the set, the system's sensitivity to variations in parameter values is critical for controller design. Therefore, sensitivity analysis techniques are applied to the system output to discern which parameters might push the system output outside of the permissible set M_Y. If these parameters are expected to vary significantly during system operation, then this advises the controller design process.

Precup et al. [5] incorporates a sensitivity analysis into their controller design, by constructing an objective function which contains the weighted sums of the control error along with the output sensitivity function. Thus, when a gravitation search algorithm (GSA) is used to minimize the objective function, the sensitivity with respect to parameters will be minimized at the same time as the control error. So, when the solution to the objective function is found, the system will also have the lowest sensitivity to changes in the parameters and thus be more reliable.

Lin et al. [6] combined sensitivity theory with ordinal optimization methodology [STOO] to solve an optimization problem. The ordinal optimization process works by using probabilities to solve for a sufficient solution, rather than the definitive best solution. Sensitivity theory is used to calculate the gradient of the objective function with respect to the change in each discrete variable's value. These sensitivities are then compared for all the variables, and the ones with the smallest sensitivities are chosen as part of the selected subset. This helps significantly reduce the search space, and significantly reduce computation time compared to most optimization techniques.

Brenna et al. [7] used sensitivity theory to design an automatic distributed voltage regulation system to better control the node voltages in the midst of the emerging distributed generation (DG) technology, which involves installing more generators into the medium-voltage distribution network (MVDN). Currently, the voltages of MVDNs are regulated by the On-Load Tap Changer (OLTC) of the high voltage / medium voltage (HV/MV) transformer. The OLTC controller does not guarantee that the network nodes will be at acceptable voltage levels after generators inject power into the network. Once reactive power is injected, the voltages rise in all the network nodes, some more than others. These nodes are shown in Fig. 1.1 as dots. Node 0 represents the MV busbar and is regulated at a constant voltage value. The voltages of the other four nodes, V_i, depend on the reactive energy absorption, Q_j, occurring at the generators/general loads of the other nodes. Since the generators in DG systems are often based on renewable energy, they can have unpredictable power-time profiles. Therefore, they could inject high power during a low load condition and cause some nodes to exceed the maximum voltage threshold. Each generator has its own Generator Remote Terminal Unit (GRTU) to connect it to a central Generator Control Center (GCC) control system to manage the generator reactive power. When the voltage in a generator's node passes the maximum threshold, a signal is sent from the GRTU to the GCC. Then the GCC uses sensitivity analysis techniques to determine which generator has the capability to provide the maximum effect on the overloaded node voltage by absorbing the most reactive energy.

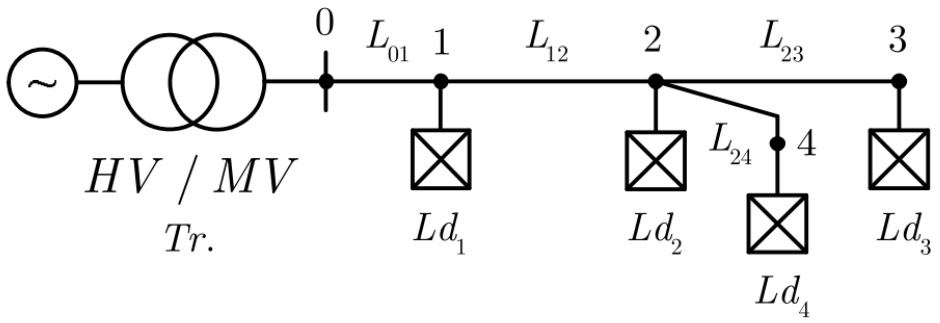

Figure 1.1: Medium Voltage Distribution Network with Four Nodes.

This sensitivity analysis involves building a reactive sensitivity matrix to calculate the partial derivative of the node voltages, V_i, with respect to reactive power-variations, Q_j. These partial derivatives are like the "gain" for the voltage variation in a certain node when a reactive power variation occurs in another node. These sensitivities, act as coefficients and are multiplied by the reactive power variation values to compose the sensitivity product. The sensitivity product is simply a normalized version of the sensitivity index, or the gain, described in Eq. (1.2). A normalized sensitivity product might be used if the engineer recognizes that the cost of incrementing the influential parameter is negligible, and the effect had by optimizing the parameter is more relevant. However, if it was costly to increment the parameters, an design engineer would be more interested in the ratio of effect had on the output with respect to the amount the parameter was increased. In this scenario an unnormalized sensitivity index would be utilized.

$$sensitivity\ product = \frac{\partial V_i}{\partial Q_j} \Delta Q_j \tag{1.4}$$

The best generator (BG) is found by comparing these sensitivity products and choosing the one with the most influence on the node where the voltage threshold was exceeded. This generator is then switched to reactive power absorption mode so that the voltage can be returned to acceptable levels.

Zad et al. [8] built upon this work and worked on optimizing the reactive power absorption to keep the node voltages through the medium voltage distribution network within safe ranges. The sensitivity indices relating voltage, V_i, and reactive power variation, Q_{DGx}, of each of the N

distributed generation (DG) units are used to formulate an equality constraint for an optimization algorithm. This algorithm picks which nodes should be used to absorb reactive power and how much power each of them should absorb to return the system voltages to appropriate levels. The objective function and constraints are shown in Eq. (1.5) – (1.7) below.

Minimize objective function:

$$\sum_{x=1}^{N} \left| \Delta Q_{DG_x} \right| \tag{1.5}$$

Constraints:

$$\Delta V_{to\ recover} = -\Delta V_{\max deviation} = \sum_{x=1}^{N} \frac{\partial V_{\max deviation}}{\delta Q_{DG_x}} \Delta Q_{DG_x} \tag{1.6}$$

$$Q_{DG_{x,min}} \leq Q_{DG_x} \leq Q_{DG_{x,max}} \tag{1.7}$$

Heuristic optimization tools, such as the particle swarm optimization (PSO) algorithm, can be used to solve this problem.

These examples where sensitivity analysis tools were applied to control systems introduce some of the many applications of sensitivity analysis techniques. These examples show how sensitivity analyses can be used successfully in controller design. This next section will defend the benefits of applying sensitivity analysis techniques to plant design optimization problems through examples from the literature. Together these motivate the ability to apply sensitivity analysis tools to the co-design of both the plant and the controller, which is the topic of this thesis.

1.2.3 Sensitivity Analysis in Design Optimization

Engineers often rely heavily on their experience and intuition when it comes to plant design. These decisions could be better informed through a sensitivity analysis which could predict the outcome of design changes without requiring trial and error. Design sensitivity analysis (DSA) has been applied to system design problems to find the sensitivity of the system response to variations in its design variables. The system response is typically considered to be either the output values of state variables or the objective function of an optimization problem. The

sensitivity coefficients are defined as the gradient of the response functions with respect to the design variables. These sensitivities can inform the design optimization process to ensure the designed system's performance is able to meet its objectives for the entire range of possible parameter variation or the entire range of loading conditions. Numerical sensitivity analysis methods, such as the finite difference method, or analytical sensitivity analysis methods can be used in design sensitivity analysis problems. Design sensitivity analyses can also be dynamic to account for time-varying uncertainties. This helps the engineer design products with the product's entire lifecycle and corresponding changing uncertainties in mind.

Caliano et al. [9] designed the absorption chiller and thermal energy storage system of a combined cooling, heat and power (CCHP) system with the help of a numerical sensitivity analysis. The nonlinear nature of the system model restricted the sensitivity analysis to be numerical, rather than analytical. A biomass-fueled combined heat and power (CHP) unit is used to provide electricity and hot water throughout the year. During the winter, it is also used for ambient heating and in the summer, it is used to power the absorption chiller. The thermal energy storage system is used to store excess heat generated by the CHP unit and use it later when the heat demand exceeds availability.

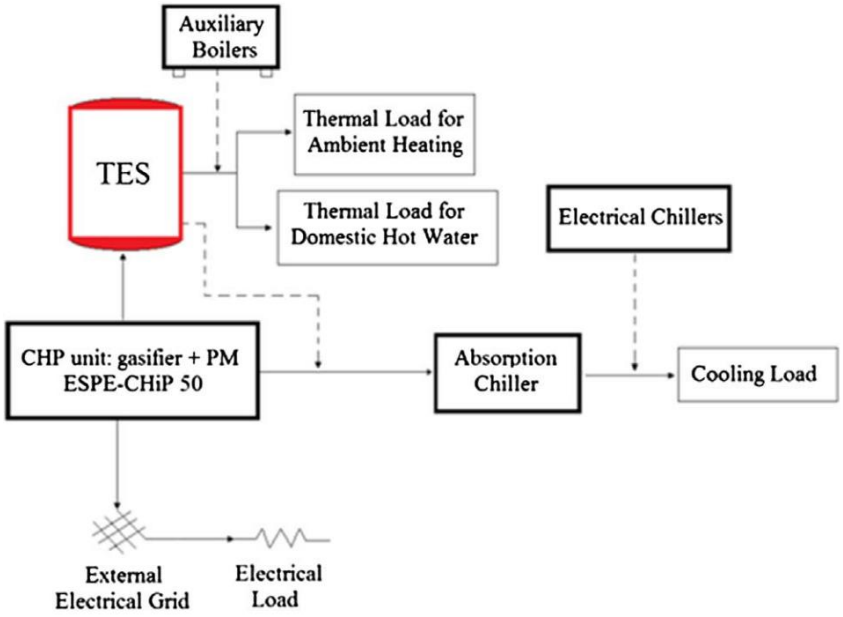

Figure 1.2: Layout of the CCHP plant [9].

First, a sensitivity analysis was conducted on the effect the feed-in tariffs and premiums had on the economic viability and system design of these biomass-fired CCHP systems in Italy. Next, the sensitivities of the thermal energy storage system and absorption chiller sizes with respect to the economic and energetic performance of CCHP systems were evaluated. This iterative, numerical sensitivity analysis showed that the feed-in premium had a large effect on the thermal energy storage systems and the absorption chiller sizes. However, the feed-in tariff value had a negligible effect on the sizes of these components. The second sensitivity analysis showed the absorption chiller size to be significantly more influential on the CCHP system's economic and energetic performance than the size of the thermal energy storage. Therefore, in designing this biomass-fired CCHP system, more effort should be placed in the design of the absorption chiller.

Cho et al. [10] developed an analytical DSA approach to solve a topology design optimization problem for weakly coupled thermo-elasticity problems. Their goal was to find the optimal material distribution to maximize stiffness or minimize compliance of the structural system. The thermal and displacement fields were considered coupled since the design variables, the bulk densities of each element, are associated with both Young's modulus and thermal conductivity. Therefore, temperature and displacement were considered in a common domain and used to derive the design sensitivity equations. An analytical sensitivity analysis technique was used to take the first-order derivatives of the general performance equation with respect to each of the design variables. The performance measure was the compliance of the plate. Through sensitivity analysis methods, the engineers could determine the effect the bulk density of each element had on the structural compliance, given a variety of both thermal and mechanical loads. Since topology optimization involves many design variables, one for each element, a gradient-based optimization was used in conjunction with the sensitivity analysis. These results informed the engineer's design decisions in designing the material distribution to optimize stiffness, specific to the expected loading conditions.

Xu et al. [11] used a time-varying sensitivity analysis to optimize the design of a mechanical system which experiences time-varying uncertainties. Design optimization is often inaccurate if it does not consider the time varying nature of uncertainties, such as material properties, operating environments, and loads. Therefore, there have already been efforts made to consider the time-varying uncertainty of a single variable in the optimization design process to

maximize product value for the entire product life cycle. However, since mechanical systems usually have multiple variables, each with their own time-varying uncertainties, these all need to be considered. This motivates a sensitivity analysis to determine which time-varying uncertainties of which variables have the most influence on the system performance, or the highest reliability sensitivity. The sources for these uncertainties are the ones which should be strictly controlled during design and manufacture. Xu et al. proposed a novel multidisciplinary robust design optimization (MRDO) method which takes the time-varying uncertainties into account and causes the objective function to be more robust as well as the system time-varying reliability sensitivity. The optimization results for the proposed MRDO method were 3.3% higher than for the conventional multidisciplinary design optimization (MDO) method.

Sensitivity analyses are a well-established method for aiding in plant design. They can be applied to virtually any type of plant and all work in a similar way. They all identify the parameters which influence the system output the most, either through numerical methods or by analytically calculating the Jacobian if a linear or linearized model exists. There are many more examples in the literature of sensitivity analysis techniques being applied to system. These examples range from biomass supply chains [12] to wind turbines [13] to ethyl acetate production flowsheets [14].

Clearly applying sensitivity analysis tools to optimize the plant design is not new. However, there exists room for innovation in the intersection of plant and controller co-design. This thesis aims to develop a set of sensitivity analysis tools for graph-based models that can identify the plant parameters which effect the system output and the controller effort the most. Once these parameters are identified, they can be optimized alongside the controller gains to yield the best controller and plant design pair according to the system objectives. The following subsection seeks to explore what has already been done in the co-design field and motivate the opportunity for future work in this area.

1.2.4 Sensitivity Analysis in Controller Co-design

1.2.4.1 Sampling-Based Global Sensitivity Analysis

Abolmoali et al. [15] analyzed the suitability of a sampling-based (numerical), global sensitivity analysis (GSA) technique to help a PI controller drive a coolant supply temperature to a set point in the presence of a pulsed thermal load. Sobol's variance decomposition method was

used to identify the component parameters and input factors which have the most influence on the thermal management system's performance for the entire distribution of the output. Knowing which regions of the design space should be explored, and which regions to ignore greatly simplifies the challenging design optimization task.

The Sobol method starts with a model, which maps the k-vector X, belonging to $\mathbb{R}^k$, to the output Y, belonging to $\mathbb{R}$, as shown in Eq. (1.8).

$$Y = f(X), \qquad X \in \Omega, \qquad \Omega \subset \mathbb{R}^k \tag{1.8}$$

As will be discussed further in Chapter 3, a local sensitivity analysis, shown in Eq. (1.9), involves finding the first derivative of the model with respect to a parameter of interest, X_j. These partial derivatives of the model output are also referred to as the Jacobian.

$$Local\ sensitivity\ index\ \equiv \frac{\partial Y}{\partial X_j} \tag{1.9}$$

However, the Sobol method is a global sensitivity analysis and can explore the entire range of parameter variation, not just a small interval around the nominal values. Suppose the system model $f(X)$ from Eq. (1.8) is a square-integrable function. We can express it as a sum of the mean of $f(X)$ over the k-dimensional domain (f_0) and a series of sums of functions dependent on 1,2, …, k factors, as shown in Eq. (1.10).

$$Y\ =\ f(X) =\ f_0 + \sum_{\mathrm{i}=1}^{\mathrm{k}} f_i(X_i) + \sum_{i=1}^{k-1} \sum_{j=\mathrm{i}+1}^{k} f_{ij}\left(X_i, X_j\right) + \cdots + \mathrm{f}_{12\ldots\mathrm{k}} \tag{1.10}$$

Since the functions $f_i, f_{ij}, \ldots, f_{12\ldots k}$ are orthogonal with respect to the indices corresponding to their subscripts over the domain of integration, the variance of $f(X)$ can be expressed as in Eq. (1.11).

$$V[f(X)] = \int_0^1 \left[\sum_{i=1}^{k} f_i^2(X_i) + \sum_{i=1}^{k-1} \sum_{j=\mathrm{i}+1}^{k} f_{ij}^2\left(X_i, X_j\right) + f_{12\ldots k}^2\right] dX \tag{1.11}$$

Since $Y = f(X)$ is dependent on the values of the parameters X_i, the conditional variance of Y can be written as in Eq. (1.12)

$$Conditional\ Variance\ of\ Y\ \equiv V(Y|X_i) \tag{1.12}$$

Similarly, the unconditional variance of Y can be written as $V(Y)$. The expected value of the conditional variance is shown in Eq. (1.13) and the variance of the conditional expectation in Eq. (1.14) [16].

$$Expected\ value\ of\ conditional\ variance\ \equiv\ E_{x_i}[V(Y|X_i)] \tag{1.13}$$

$$Variance\ of\ the\ conditional\ expectation\ \equiv\ V_{x_i}[E(Y|X_i)] \tag{1.14}$$

The first order Sobol sensitivity index of Y with respect to X_i can be written as the reduction of variance in Y if the input factor X_i was fixed, as in Eq. (1.15).

$$S_i = \frac{V_i}{V[Y]} = \frac{V_{X_i}\left[E_{X_{\sim i}}[Y|X_i]\right]}{V[Y]} \tag{1.15}$$

Another metric used for analysis is the total effects index shown in Eq. (1.16). This index describes the main effect of X_i in addition to all the interactions involving $X_{\sim i}$.

$$T_i = \frac{V[Y] - V_{X_{\sim i}}\left[E_{X_i}[Y|X_{\sim i}]\right]}{V[Y]} \tag{1.16}$$

The DAKOTA software suite [17]**,** developed by Sandia National Laboratories, is used to create gaussian process surrogate models to accelerate the computation of the sensitivity indices. The generalized Sobol indices are then used to rank the input parameters in terms of influence on the output for a finite time interval.

Abolmoali et al. [15] applied this methodology to the design of a vapor cycle system and a PI controller regulating the compressor speed to effectively reject a pulsed thermal load and drive a coolant supply temperature to a set point. The sensitivity analysis showed that the compressor speed control gain was the dominant factor and had the greatest effect on performance, followed by the condenser height, evaporator width, evaporator height, and valve exit diameter. The condenser length and width along with the evaporator length had a negligible effect on the output variance.

Abolmoali et al.'s work [15] is an example of a co-design optimization problem where a sensitivity analysis was conducted on the design of both the plant and controller to meet a set of goals. Since this method is based on Sobol's variance-based approach, it is a numerical method and required creating a surrogate model and running 500,000 model evaluations to obtain results.

1.2.4.2 Plant-Limited Co-Design Optimization

Another instance where a sensitivity analysis was applied to a codesign problem is when Allison [18] explored the idea of using a sensitivity analysis to identify the most influential parameters of a plant to modify concurrently with control redesign to meet the performance requirements with new system applications. Complete plant redesign is expensive, but partial plant redesign is often necessary when control redesign alone cannot achieve the desired performance. Allison applies these Plant-Limited Co-Design (PLCD) techniques to a two-link robotic manipulator to achieve system-optimal designs with minimum-cost plant modifications. [18] The analytical sensitivity analysis is used to calculate the Jacobian (J_p), or the partial derivative of each of the output requirements that still need to be met ($\bar{g}_{ri}(p)$) with respect to each of the model parameters (p_j) shown in Eq. (1.17). The p in $\bar{g}_{ri}(p)$ indicates that the output requirements are dependent on the parameters, p. This makes sense, because for the Jacobian to be a non-zero value, $\bar{g}_{ri}(p)$ must be at least somewhat dependent on some of the parameters. The j subscript on p_j indicates which model parameter the partial derivative is being taken according to.

$$J_p = \frac{\partial \bar{g}_{ri}(p)}{\partial p_j}, \qquad i = 1, 2, \dots, n_r, \qquad j = 1, 2, \dots, n_p \tag{1.17}$$

where n_r is the number of unmet requirements and n_p is the number of model parameters.

In addition to calculating the Jacobian of the requirements that need to be met, it is also worth considering the cost of changing model parameters. These costs can be presented along with the effectiveness of changing the model parameters by calculating the cost Jacobian (J_c), shown in Eq. (1.18).

$$J_c = \frac{\partial \bar{g}_{ri}(p)}{\partial p_j}\left(\frac{\partial C(p)}{\partial p_j}\right)^{-1} = \frac{\partial \bar{g}_{ri}(p)}{\partial c_j}, \qquad i = 1, 2, \dots, n_r, \qquad j = 1, 2, \dots, n_p \tag{1.18}$$

where $C(p)$ approximates the cost of changing all model parameters and c_j represents the cost of changing model parameter p_j.

Once the set of most influential model parameters ($\bar{p}$), such as inertia values or damping rates, are selected from the Jacobian matrices, it becomes necessary to choose the corresponding candidate set of physical plant modifications $(\bar{x}_p)$, such as specific link lengths or radii. The set of control design variables (x_c) are not limited since it is less expensive to modify the control

design variables. The PLCD optimization problem can then be constructed and is shown in Eq. (1.19). [18]

$$\begin{aligned} &\min_{x=[\bar{x}_p^T, x_c^T]^T} \phi(x) \\ &s.t.\ g_p(x) \leq 0 \\ &g_r(x) \leq 0 \end{aligned} \tag{1.19}$$

where $\phi(x)$ is a measure of deviation from the original plant design, $g_p(x)$ are plant design constraints and $g_r(x)$ are system performance requirements formulated as inequality constraints.

Applying these PLCD methods to a mechatronic system redesign, Allison was able to identify the most influential physical design changes and optimize them alongside the controller design variables to produce a superior system-optimal design while greatly reducing plant modification cost. [18]

1.2.4.3 Motivation for Co-Design in the Graph-Based Modelling Space

This section has established the benefit and feasibility of using sensitivity analysis techniques in both the control design and plant design space. It has also introduced a few instances where sensitivity analysis tools benefitted the co-design process. This thesis seeks to add to that field of knowledge by developing a sensitivity analysis methodology to be used in a co-design application. This thesis will focus on developing an analytical sensitivity analysis, due to the pre-existing, linearized, graph-based models, which will be covered in detail in Chapter 2, that are used to model the systems of interest. The analytical nature of this method will be able to solve for the local sensitivity on a ms timescale. This opens up the possibility to further this work and develop a method to dynamically solve for the sensitivities, using the current operating points as the nominal values. As for further details about the types of sensitivity analysis methods and which ones are best suited for the graph-based models, these can be found in Chapter 3. There, more detail will be provided to demonstrate how sensitivity analyses can be used to narrow the scope of variables to the most impactful ones on the plant and controller. This would allow for the optimization of controller gains and the most sensitive components at the same time to arrive at the optimized plant and controller pair in the least amount of time.

1.3 Organization of Thesis

This thesis aims to introduce an analytical sensitivity analysis methodology applied to the DAEMOT graphing tools and the graph-based models which can be generated from them. This sensitivity analysis will aid in the plant and controller codesign to optimize system performance over a given mission profile.

The remainder of the thesis is organized as follows. Chapter 2 explains the DAEMOT modeling framework, as well as the control-oriented, graph-based models which are developed from the governing equations of the DAEMOT models. Chapter 2 will provide an example of a fluid loop to demonstrate the power flow between components. Chapter 3 explores the pre-existing sensitivity analysis tools and the types of systems which are best suited for each sensitivity analysis approach. Chapter 4 will introduce the novel sensitivity analysis technique and apply it to the simple model introduced in Chapter 2. Chapter 4 will also introduce a thermal management system with parallel fluid flow branches and apply the sensitivity analysis methodology to the system. Chapter 5 will discuss the results of the sensitivity analysis and focus on the plant design optimization. Chapter 6 will consider both the controller gains and plant parameters in a co-design problem and analyze the results of that. Chapter 7 summarizes the conclusions and discusses opportunities for future work.

Chapter 2

Modeling

2.1 DAEMOT Toolbox

This thesis uses the DAEMOT toolbox in Simulink to model the hydraulic and thermal domains of single-phase, fluid-thermal component models. Each of the DAEMOT components is governed by differential equations for the temperature, pressure, and mass flow rates. These differential equations are solved numerically over the course of the simulation in Simulink. Although this thesis focuses on fluid-thermal component models, the sensitivity analysis proposed in this work could be extended to any combination of thermal, hydraulic, mechanical, or electrical power domains, where storage, transport, or conversion of conserved quantities, such as mass or energy, happens. [19]

Each of the DAEMOT components has an input and output bus, which contains a temperature, pressure, and mass flow rate value. When building a simulation-based model, it is important to remember that the calculation of mass flow rate and pressure are linked and alternate. For example, the pressure of the previous and following components are used to calculate mass flow rate through the component in the middle, as shown in Fig.2.1 and Eq. (2.1).

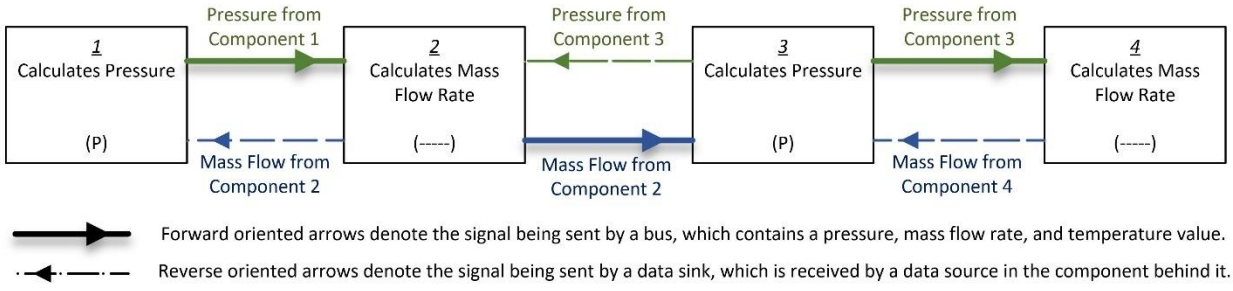

Figure 2.1: Alternating Mass Flow Rate and Pressure Calculations.

$$\dot{m} = \rho A_c \sqrt{\frac{2(p^{tail} - p^{head} + \rho g \Delta h)}{\rho \left(f \frac{L}{D} + K_L \right)}} \tag{2.1}$$

where p^{tail} and p^{head} are the pressures of the upstream and downstream components, ρ is the density of the fluid, A_c, L, D are the cross sectional area, length, and diameter of the component, respectively, g is the gravitational constant, Δh is the height difference between the inlet and outlet flow, f is the friction factor, and K_L is the minor loss coefficient.

Similarly, the mass flow rates into the upstream and downstream components are used to calculate the outlet pressure of the component in between them, as shown in Fig. 2.1 and Eq. (2.2).

$$\dot{p}_{tail} = \frac{(\dot{m} - \dot{m}_{tail})}{V \cdot \rho \cdot \left(\frac{1}{E_{fluid}} + D \frac{\left(1 - \frac{v}{2}\right)}{t \cdot E} \right)} \tag{2.2}$$

where $\dot{p}_{tail}$ is the time derivative of the output pressure, $\dot{m}$ and $\dot{m}_{tail}$ are the mass flow rates into and out of the component, V is the volume of fluid in the component, ρ is the density of that fluid, E_{fluid} is the bulk modulus, D is the diameter of the component, v is the tube poisson ratio, t is the tube wall thickness, and E is the tube modulus of elasticity.

Sometimes a component, such as the cold plate, might have both a mass flow rate-calculating and pressure-calculating block, but the mass flow and pressure calculations will always alternate. By alternating the pressure and mass flow calculating blocks, the DAEMOT component models can be combined to form a model of a system, making the toolbox modular and useful for building up complex models out of these simple components.

The following section will discuss how the governing equations for these component models can be represented as graph-based models, comprised of vertices and oriented edges. The capacitances and resistances of these vertices and edges are directly determined from the governing differential equations of the components.

2.2 Graph Based Modeling Basics

Graph-based models are used to perform dynamic modeling and analysis of a system by representing the power flow in a system. One of the biggest strengths graph-based models offer is

their ability to represent the power flow through models which span the thermal, hydraulic, mechanical, and electrical power domains. This power domain agnosticism comes from the models being founded in conservation of energy equations, which supports easy conversion between different domains. This allows for the modeling of a wide variety of systems, such as thermal management systems, aircraft and hybrid-electric UAV powertrains, and hybrid energy storage systems [20] [21] [22] [23]. These models can also be built up in a modular way since individual components or subsystems can be combined to build up larger systems. This greatly simplifies the validation of a large and complex model, since each of its components can be isolated and validated separately. Graph-based models also allow for variable fidelity and the number of dynamic states can easily be expanded to form more complex or simplified to create less complex and detailed models. The structure of graph-based models also allows them to be easily scaled to various sizes and can be solved in a computationally efficient way [24]. For these reasons, graph-based models are an excellent modeling medium for control-oriented efforts for the energy exchange of multi-domain power systems.

The general idea of the oriented graph model $G = (v, e)$ is to represent the storage and exchange of energy in the system it is modeling. G consists of vertices v and edges e. v represents the set of vertices and is an equivalent representation to $[v_i], i \in [1: N_v]$ where N_v is the number of vertices in G. Similarly, e represents the set of edges and is equivalent to $[e_j], j \in [1: N_e]$ where N_e is the number of edges in G. Each oriented edge, e_j, connects a vertex at the tail of the edge, v_j^{tail}, to another vertex at its head, v_j^{head}. The set of edges which are oriented into a vertex, v_i, are represented by the set $e_i^{head} = \{e_j | v_j^{head} = v_i\}$. Similarly, the set of edges oriented out of a vertex, v_i, are represented by the set $e_i^{tail} = \{e_j | v_j^{tail} = v_i\}$ [25]. Each vertex, v_i, of the graph G has a dynamic state, x_i, which represents the amount of energy stored in it, and a corresponding storage capacitance, C_i. And each edge, e_j, has an associated positive power flow, P_j, which is oriented from v_j^{tail} to v_j^{head}. The dynamic states, x, and the power flows, P, form the system S, which follows the conservation equation, as shown in Eq. (2.3).

$$C_i \dot{x}_i = \sum_{\{j | e_j \in e_i^{head}\}} P_j - \sum_{\{j | e_j \in e_i^{tail}\}} P_j \tag{2.3}$$

Eq. (2.4) shows how the power flow, P_j, of edge j is a function of the adjacent vertices as well as the input signal u_j.

$$P_j = f_j\left(x_j^{tail}, x_j^{head}, u_j\right) \tag{2.4}$$

Eq. (2.3) and (2.4) are illustrated in Fig. 2.2 in a notional graph example.

External disturbances are modeled in graph-based models by source edges, $e^s = \left[e_j^s\right], j \in [1{:}\, N_s]$. The power flows for these edges are represented by $P^s = [P_j^s]$. Since source edges and power flows are external to the system, they are differentiated by dotted vertices or edges in Fig. 2.2 and are not included in the internal edges e of graph G or the internal power flows P of system S. Similarly, sink states, modeled as $x^t = \left[x_j^t\right], j \in [1, N^t]$, are also considered disturbances to the system and are not included in the state vector x of system S. However, the sink vertices, $v^t = [v_j^t]$, are still counted as vertices v of graph G. Note that edges $e_j \in e$ cannot connect two sink vertices to each other. An edge which is oriented into a sink must also be connected to a vertex, which is not a sink.

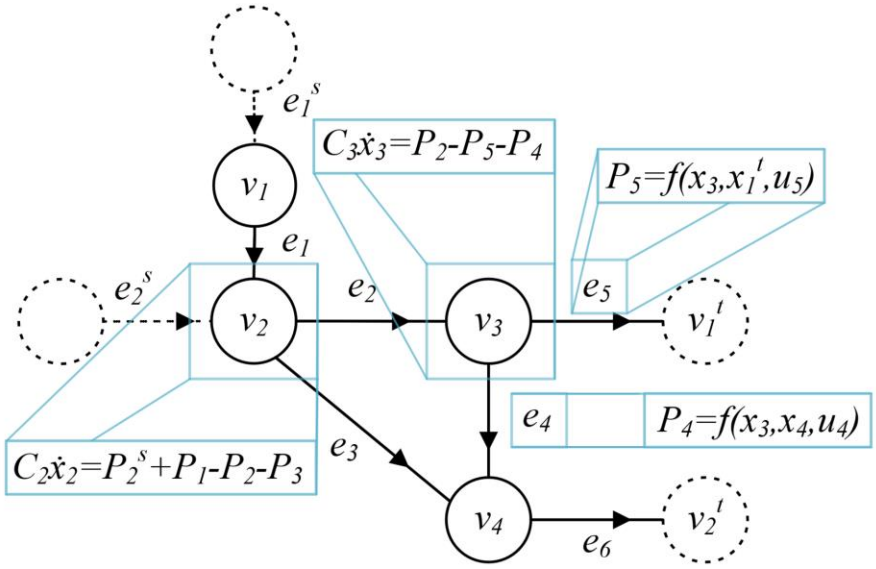

Figure 2.2: Notional graph-based model. Modified from [19]**,**[24]**.**

The system model, S, resulting from the graph, G, can be converted to matrix form for efficient calculations, as shown in Eq. (2.5). This equation follows from the conservation equation Eq. (2.3) holding for each state x in system S.

$$S: \qquad C\dot{x}(t) = -\bar{M}P(t) + DP^s(t) \tag{2.5}$$

where $C = diag([C_i])$ is a diagonal matrix of the capacitances of the states x.

The matrix $M = [m_{ij}] \in \mathbb{R}^{N_v \times N_e}$ is the incidence matrix which maps the oriented edges to their adjacent verticies. Each of the values in the matrix M can either be 1, -1, or 0 as shown in Eq. (2.6)

$$m_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is the tail of } e_j, \\ -1 & \text{if } v_i \text{ is the head of } e_j, \\ 0 & \text{else.} \end{cases} \tag{2.6}$$

M can be partitioned into $\bar{M}$, the part of the incidence matrix which maps the power flows to the non-sink states, x, and $\underline{M}$, the part of the incidence matrix which maps the power flows to the sink states, x_t (Eq. 2.7).

$$M = \begin{bmatrix} \bar{M} \\ \underline{M} \end{bmatrix} \text{ with } \bar{M} \in \mathbb{R}^{(N_v - N_t) \times N_e} \text{ and } \underline{M} \in \mathbb{R}^{N_t \times N_e} \tag{2.7}$$

Similarly, matrix $D = [d_{ij}] \in \mathbb{R}^{(N_v - N_t) \times N_s}$ maps the source power flows to the dynamic, non-sink states, x. Each element of D is either a 0 or a 1, as shown in Eq. (2.8).

$$d_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is the head of } e_j^s, \\ 0 & \text{else.} \end{cases} \tag{2.8}$$

The M and D matrices which represent the structure of the example graph in Fig. 2.2 are in Eq. (2.9).

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2.9}$$

Now that the general form of the graph-based model is established, the following subsections will cover the governing equations for each of the components used to build up system models. First the governing equations used in the DAEMOT models will be highlighted, followed by how these equations translate into the thermal domain of the graph-based modeling framework.

2.2.1 Reservoir Component

The reservoir, or tank, component is very commonly used in modeled systems and functions as a thermal storage element. The DAEMOT reservoir model has three states it is solving for: the temperature of the fluid in the reservoir (T_f), the temperature of the reservoir wall (T_w),

and the pressure in the reservoir (p). To represent the pressure, p, in kPa units, a conversion rate of 1000 must be included in Eq. (2.12). The equations for the state derivatives are as follows:

$$\dot{T}_f = \frac{\dot{m}_{in} \cdot c_{P_f} \cdot \left(T_{in} - T_f\right)}{M_f \cdot c_{P_f}} - \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_f \cdot c_{P_f}} \tag{2.10}$$

$$\dot{T}_w = \frac{Q_{amb}}{M_w \cdot c_{P_w}} + \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_w \cdot c_{P_w}} \tag{2.11}$$

$$\dot{p} = \frac{(\dot{m}_{in} - \dot{m}_{out}) \cdot g}{1000 \cdot A_{cross}} \tag{2.12}$$

where c_{P_f} and c_{P_w} are the specific heat values for the fluid and wall, respectively, M_f and M_w are the masses of the fluid and wall, respectively, $\dot{m}_{in}$ and $\dot{m}_{out}$ are the mass flow rates into and out of the reservoir, respectively, H is the heat transfer coefficient and A_{ht} is the heat transfer area between the fluid and the wall, A_{cross} is the cross sectional area of the reservoir, D and h are the reservoir diameter and height, respectively, g is the gravitational constant, and Q_{amb} is the heat load applied to the reservoir wall from the ambient air. If the reservoir is assumed to be cylindrival, A_{ht}, A_{cross}, and M_f can be expressed in Eq. (2.13) – (2.15).

$$A_{ht} = \pi \cdot h \cdot D \tag{2.13}$$

$$A_{cross} = \pi \cdot {D^2}/{4} \tag{2.14}$$

$$M_f = \rho \cdot A_{cross} \cdot h \tag{2.15}$$

To model the interaction between the ambient air and the reservoir as a temperature difference instead of a heat load, simply rewrite Eq. (2.11) as Eq. (2.16).

$$\dot{T}_w = \frac{H_{amb} \cdot A_{ht} \cdot (T_{amb} - T_w)}{M_w \cdot c_{P_w}} + \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_w \cdot c_{P_w}} \tag{2.16}$$

where H_{amb} is the heat transfer coefficient between the ambient air and the reservoir wall and T_{amb} is the temperature of the ambient air.

A schematic of the reservoir inputs and outputs is shown in Fig. 2.3. The green, blue, and red lines, which represent the pressure, mass flow rate, and temperature signals, are combined in a bus and are the first input of the reservoir component. The second input is a heat load from the ambient air. This could either be a positive value, adding thermal energy to the reservoir, or a negative value, which removes thermal energy. The red circle with a line through it represents a terminator block. This means that the reservoir does not use the upstream pressure, which was sent

to it in the bus for its calculations. Since the reservoir is a pressure calculating block, it requires the upstream and downstream mass flow rates and not the upstream pressure. The transparent line represents information which came from a source block. Since the reservoir does not calculate mass flow rate, but requires the upstream and downstream mass flow rates to calculate pressure, it is passed the downstream mass flow rate by a sink block and accesses it with a source Simulink block.

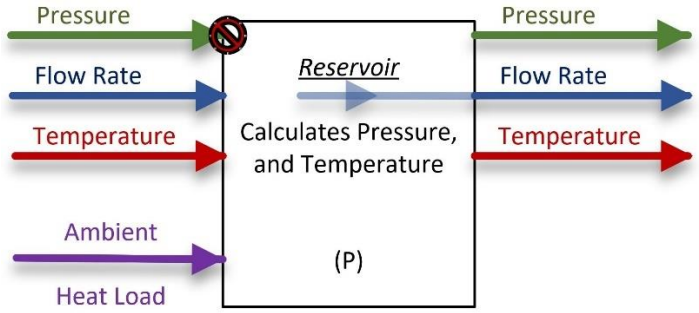

Figure 2.3: Schematic of a Reservoir Component.

Equations (2.10) – (2.12) are solved in Simulink by an ordinary differential equation (ODE) solver to find the temperature of the fluid in the reservoir (T_f), the temperature of the reservoir wall (T_w), and the pressure in the reservoir. For modeling in the thermal energy domain, only the temperature differential equations, Equations (2.10) and (2.11), are converted into a graph-based modeling form.

In graph-based modeling, the dynamics are derived from conservation equations. For the thermal domain, the conservation of thermal energy is used, shown in Eq. (2.17).

$$\dot{E}_{st} = P_{in} - P_{out} \tag{2.17}$$

where P_{in} and P_{out} are the power flows into and out of the component, respectively, and E_{st} is the energy stored in the component as shown in Eq. (2.18).

$$E_{st} = M \cdot c_P \cdot T \tag{2.18}$$

The time derivative of the stored energy is shown in Eq. (2.19).

$$\dot{E}_{st} = \dot{M} \cdot c_P \cdot T + M \cdot c_P \cdot \dot{T} \tag{2.19}$$

However, most components do not have a dynamic fluid mass and are considered flooded; so, $\dot{E}_{st} \approx M \cdot c_P \cdot \dot{T}$. The only exception to this is the reservoir's fluid vertex, because reservoirs have a dynamic mass due to the flow in not always equalling the flow out. To be able to state that

$\dot{E}_{st} \approx M \cdot c_P \cdot \dot{T}$ for all thermal components, the term $\dot{M} \cdot c_P \cdot T$ can be considered a virtual power flow to represent the thermal energy that is lost or gained due to the changing fluid mass. This assumes perfect mixing within the reservoirs. This virtual power edge leads into a separate sink vertex which tracks the reservoir storage. Since the reservoir's $\dot{M} \cdot c_P \cdot T$ term is captured by a virtual power flow, the vertex storage capacitance for all thermal components can be represented as $M \cdot c_P$, where the mass, M, can be calculated by multiplying the density of the vertex material by the volume of the vertex in any given component, and c_p is the specific heat of the vertex material. These are the capacitance values that compose the matrix $C = diag([C_i])$ from Eq. (2.5).

The power flow matrix $P(t)$ from Eq. (2.5) is composed of the convective, advective, or external input power flows. Convective edges, shown in Eq. (2.20), describe the power flow between wall and fluid vertices, or between the ambient air mass and a wall vertex, if the ambient air is represented as a temperature instead of as a heat load. These edges are modeled as red arrows in Fig. 2.4, 2.6, and 2.8. Advective edges, shown in Eq. (2.21), represent the thermal energy that is transported by fluid flow through the system. These edges are modeled as blue arrows in Fig. 2.4, 2.6, 2.8, 2.10, 2.12, and 2.14.

$$P_{convective} = H \cdot A_{ht} \cdot \left(x^{tail} - x^{head}\right) \tag{2.20}$$

$$P_{advective} = \dot{m} \cdot c_P \cdot x^{tail} \tag{2.21}$$

where x^{tail} is the upstream fluid temperature entering the component and x^{head} is the downstream fluid temperature exiting the vertex. For the graph-based models, the dynamic mass flow rates, $\dot{m}$, are treated as virtual inputs.

It is not difficult to see that Eq. (2.10) and (2.11) share the same structure as Eq. (2.17). If both sides of Eq. (2.10) were multiplied by the fluid capacitance, $C_{fluid} = M_f \cdot c_{P_f}$, and both sides of Eq. (2.11) were multiplied by the wall capacitance, $C_{wall} = M_w \cdot c_{P_w}$ then we would get the equations shown in Eq. (2.22) and Eq. (2.23) respectively.

$$M_f \cdot c_{P_f} \cdot \dot{T}_f = \dot{m}_{in} \cdot c_{P_f} \cdot \left(T_{in} - T_f\right) - H \cdot A_{ht} \cdot \left(T_f - T_w\right) \tag{2.22}$$

$$M_w \cdot c_{P_w} \cdot \dot{T}_w = Q_{amb} + H \cdot A_{ht} \cdot \left(T_f - T_w\right) \tag{2.23}$$

where $M_f \cdot c_{P_f} \cdot \dot{T}_f = C_{fluid} \cdot \dot{T}_f$ and $M_w \cdot c_{P_w} \cdot \dot{T}_w = C_{wall} \cdot \dot{T}_w$ are the time derivatives of the energy storage, $\dot{E}_{st}$, from Eq. (2.17), $\dot{m}_{in} \cdot c_{P_f} \cdot T_{in}$ and $-\dot{m}_{in} \cdot c_{P_f} \cdot T_f$ are the advective flows into and out of the fluid vertex respectively, $-H \cdot A_{ht} \cdot \left(T_f - T_w\right)$ is the convective power flow

out of the fluid vertex and into the wall vertex, and Q_{amb} is the input power flow into the reservoir wall vertex.

The power flows are structured in terms of edge coefficients when building matrix P. The power flow equation for all components can be represented by the general equation shown in Eq. (2.24).

$$P = e_1 \cdot T^{tail} + e_2 \cdot T^{head} + e_3 \cdot T^{tail} \cdot \dot{m} + e_4 \cdot T^{head} \cdot \dot{m} \tag{2.24}$$

where c_i are the edge coefficients.

Using Eq. (2.22) as an example, the edge coefficients for the reservoir's fluid vertex are shown in Eq. (2.25).

$$e_1 = H \cdot A_{ht}; \quad e_2 = -H \cdot A_{ht}; \quad e_3 = \dot{m}_1 \cdot c_{P_f}; \quad e_4 = -\dot{m}_2 \cdot c_{P_f} \tag{2.25}$$

Similarly, the edge coefficients which represent the power flows into the wall vertex in Eq. (2.23) are shown in Eq. (2.26).

$$e_1 = -H \cdot A_{ht}; \quad e_2 = H \cdot A_{ht} \tag{2.26}$$

Arranging these equations in the C, $\dot{x}(t)$, $\bar{M}$, $P(t)$, D, and $P^s(t)$ matrices, in Eq. (2.27) – (2.32), we can model the reservoir in the form of Eq. (2.5).

$$C = diag\left(\begin{bmatrix} M_f \cdot c_{P_f} & M_w \cdot c_{P_w} \end{bmatrix}\right) \tag{2.27}$$

$$\dot{x}(t) = \begin{bmatrix} \dot{T}_f & \dot{T}_w \end{bmatrix}^T \tag{2.28}$$

$$\bar{M} = \begin{bmatrix} -1 & -1 \\ 1 & 0 \end{bmatrix} \tag{2.29}$$

$$P(t) = \begin{bmatrix} H \cdot A_{ht} \cdot (T_f - T_w) & \dot{m}_2 \cdot c_P \cdot T_f \end{bmatrix}^T \tag{2.30}$$

$$D = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

$$P^s(t) = \begin{bmatrix} \dot{m}_1 \cdot c_P \cdot T_{source} & (\dot{m}_2 - \dot{m}_1) \cdot c_P \cdot T_f & Q_{amb} \end{bmatrix}^T \tag{2.32}$$

Therefore, a reservoir can be modeled as a graph-based model as shown in Fig. 2.4.

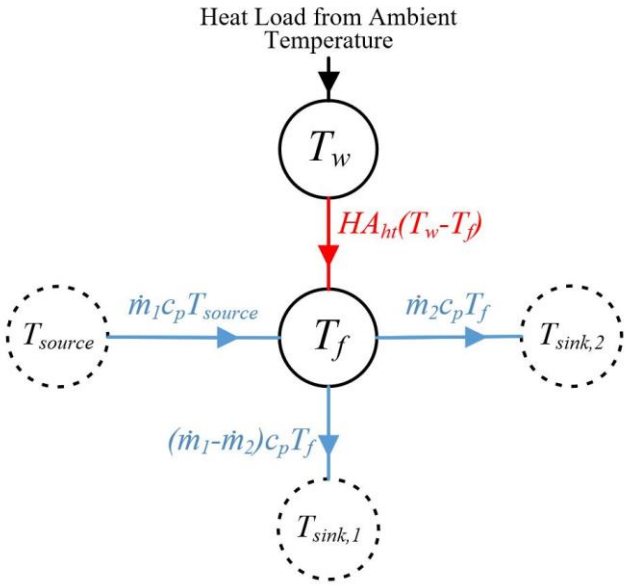

Figure 2.4: Graph-Based Model of a Reservoir Component.

2.2.2 Cold Plate Component

Another common component is the cold plate, which is used to increase or decrease the thermal energy in a system. It consists of an aluminum plate with copper tubing running through it. The model structure of the cold plate is very similar to the reservoir component, except that it is flooded, meaning the mass of fluid in the cold plate remains constant. The equations for solving the state derivatives, temperature of the fluid in the cold plate (T_f), temperature of the cold plate wall (T_w), and pressure in the cold plate (p), are shown in Eq. (2.33) – (2.35).

$$\dot{T}_f = \frac{\dot{m}_{in} \cdot c_{P_f} \cdot \left(T_{in} - T_f\right)}{M_f \cdot c_{P_f}} - \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_f \cdot c_{P_f}} \tag{2.33}$$

$$\dot{T}_w = \frac{Q_{amb}}{M_w \cdot c_{P_w}} + \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_w \cdot c_{P_w}} \tag{2.34}$$

$$\dot{p} = \frac{(\dot{m} - \dot{m}_{out})}{V \cdot \rho \cdot \left(\frac{1}{E_{fluid}} + D\frac{\left(1 - \frac{v}{2}\right)}{t \cdot E}\right)} \tag{2.35}$$

where V is the volume of the cold plate tube, ρ is the density of the fluid, E_{fluid} is the bulk modulus of the fluid, v is the poisson ratio, t is the tube thickness, and E is the bulk modulus of the tube. The volume of the cold plate is defined by its shape and shown in Eq. (2.36).

$$V = L \cdot \pi \cdot {D^2}/{4} \tag{2.36}$$

where L is the total length of the cold plate tube and D is the diameter of that tube. A schematic of the cold plate component's inputs and output is shown in Fig. 2.5.

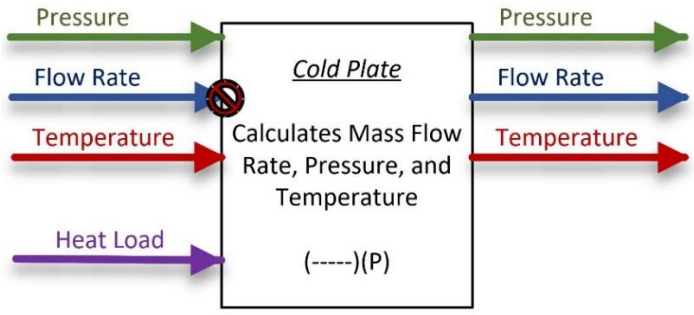

Figure 2.5: Schematic of a Cold Plate Component.

Like the reservoir, to model the cold plate as a graph-based model in the thermal domain, it must be broken down into vertex capacitances and edge coefficients. The capacitance of the fluid vertex and wall vertex are described as Eq. (2.37) and (2.38) respecticely.

$$C_{fluid} = M_f \cdot c_{P_f} = \rho \cdot V \cdot c_{P_f} \tag{2.37}$$

$$C_{wall} = M_w \cdot c_{P_w} \tag{2.38}$$

The edge coefficients for a cold plate component contain advective and convective terms, similar to the reservoir component. These coefficients for the fluid and wall vertices are shown in Eq. (2.39) and (2.40) respectively.

$$e_1 = H \cdot A_{ht}; \quad e_2 = -H \cdot A_{ht}; \quad e_3 = \dot{m}_{in} \cdot c_{P_f}; \quad e_4 = -\dot{m}_{in} \cdot c_{P_f} \tag{2.39}$$

$$e_1 = -H \cdot A_{ht}; \quad e_2 = H \cdot A_{ht} \tag{2.40}$$

Therefore, a cold plate modeled as a graph-based model is shown in Fig. 2.6.

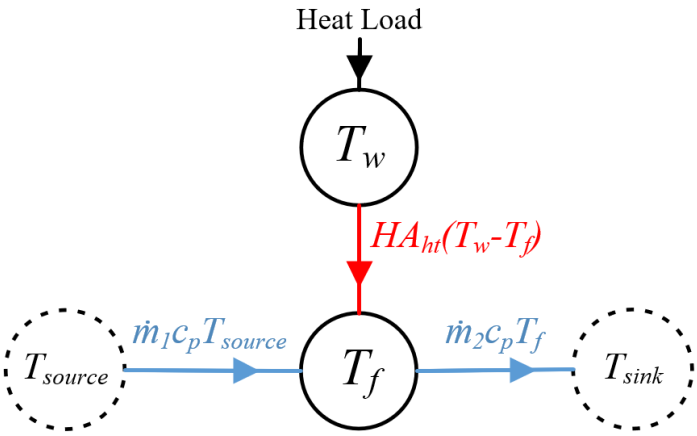

Figure 2.6: Graph-Based Model of a Cold Plate Component.

2.2.3 Heat Exchanger Component

Another component often used in modeling is the brazed plate liquid-to-liquid heat exchanger, which allows thermal energy to transfer between liquid loops. The heat exchanger has five states it solves for in the DAEMOT model: the fluid temperatures on sides one and two, the wall temperature, and the pressures on sides one and two. A schematic of the heat exchanger inputs and outputs is shown in Fig. 2.7.

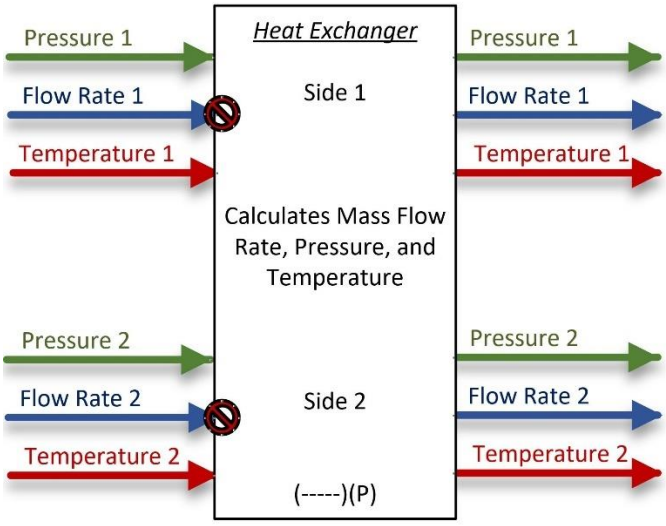

Figure 2.7: Schematic of a Liquid-to-Liquid Heat Exchanger Component.

The differential equations which are used to solve for the heat exchanger's states are shown in equations (2.41) – (2.45).

$$\dot{T}_{f,1} = \frac{\dot{m}_{in,1} \cdot c_{P_{f,1}} \cdot \left(T_{in,1} - T_{f,1}\right)}{M_{f,1} \cdot c_{P_{f,1}}} - \frac{H_1 \cdot A_{ht,1} \cdot \left(T_{f,1} - T_w\right)}{M_{f,1} \cdot c_{P_{f,1}}} \tag{2.41}$$

$$\dot{T}_{f,2} = \frac{\dot{m}_{in,2} \cdot c_{P_{f,2}} \cdot \left(T_{in,2} - T_{f,2}\right)}{M_{f,2} \cdot c_{P_{f,2}}} - \frac{H_2 \cdot A_{ht,2} \cdot \left(T_{f,2} - T_w\right)}{M_{f,2} \cdot c_{P_{f,2}}} \tag{2.42}$$

$$\dot{T}_w = \frac{H_1 \cdot A_{ht,1} \cdot \left(T_{f,1} - T_w\right)}{M_w \cdot c_{P_w}} + \frac{H_2 \cdot A_{ht,2} \cdot \left(T_{f,2} - T_w\right)}{M_w \cdot c_{P_w}} \tag{2.43}$$

$$\dot{p}_1 = \frac{\left(\dot{m}_1 - \dot{m}_{out,1}\right)}{V_1 \cdot \rho_1 \cdot \left(\frac{1}{E_{fluid,1}}\right)} \tag{2.44}$$

$$\dot{p}_2 = \frac{\left(\dot{m}_2 - \dot{m}_{out,2}\right)}{V_2 \cdot \rho_2 \cdot \left(\frac{1}{E_{fluid,2}}\right)} \tag{2.45}$$

However, for the thermal domain graph-based model only Eq. (2.41) – (2.43) are necessary and these can be rearranged into the form of the general equation: Eq. (2.5) by populating the following matrices in Eq. (2.46) – (2.51).

$$C = diag\left(\begin{bmatrix} M_{f,1} \cdot c_{P_{f,1}} & M_{f,2} \cdot c_{P_{f,2}} & M_w \cdot c_{P_w} \end{bmatrix}\right) \tag{2.46}$$

$$\dot{x}(t) = \begin{bmatrix} \dot{T}_{f,1} & \dot{T}_{f,2} & \dot{T}_w \end{bmatrix}^T \tag{2.47}$$

$$\bar{M} = \begin{bmatrix} -1 & 0 & -1 & 0 \\ 0 & -1 & 0 & -1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.48}$$

$$P(t) = \begin{bmatrix} H_1 \cdot A_{ht} \cdot \left(T_{f,1} - T_w\right) \\ H_2 \cdot A_{ht} \cdot \left(T_{f,2} - T_w\right) \\ \dot{m}_3 \cdot c_P \cdot T_{f,1} \\ \dot{m}_4 \cdot c_P \cdot T_{f,2} \end{bmatrix} \tag{2.49}$$

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{2.50}$$

$$P^s(t) = \begin{bmatrix} \dot{m}_1 \cdot c_P \cdot T_{source,1} & \dot{m}_2 \cdot c_P \cdot T_{source,2} \end{bmatrix}^T \tag{2.51}$$

The heat exchanger can be represented pictorially by Fig. 2.8 below.

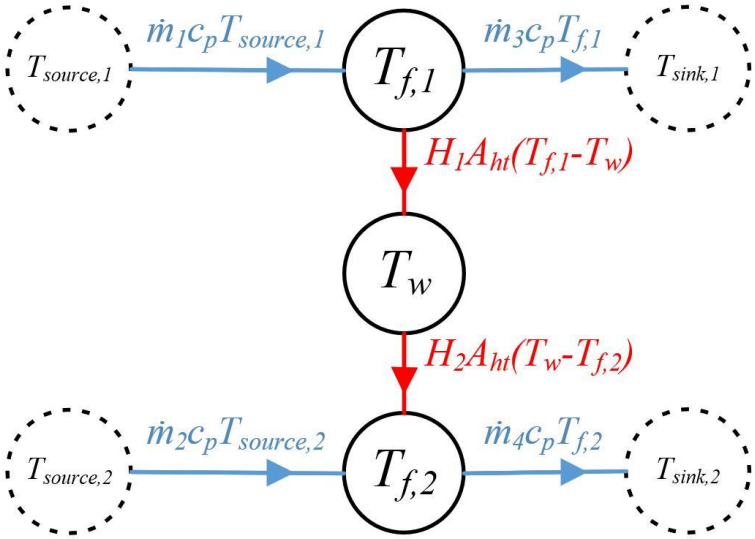

Figure 2.8: Graph-Based Model of a Liquid-to-Liquid Heat Exchanger.

2.2.4 Split/Junction Component

A split or junction component allows several fluid flows to either split off from a single flow or combine to form a single flow. The governing equations for the time derivative of the split/junction states are shown in Eq. (2.52) and (2.53).

$$\dot{T}_f = \frac{c_{P_f} \cdot \left(\sum_{n=1}^{number\ inputs} \dot{m}_{in,n} \cdot T_{in,n} - \sum_{m=1}^{number\ outputs} \dot{m}_{out,m} \cdot T_f \right)}{M_f \cdot c_{P_f}} \tag{2.52}$$

$$\dot{p} = \frac{(\dot{m} - \dot{m}_{out})}{V \cdot \rho \cdot \left(\frac{1}{E_{fluid}} + D \frac{\left(1 - \frac{v}{2}\right)}{t \cdot E} \right)} \tag{2.53}$$

Schematics of both the split and junction components are shown in Fig. 2.9. Again, the transparent arrows represent information that was obtained from a source block. The downstream components sent the mass flow rates they were receiving to the split or junction component through a sink block, so that the split or junction could know how much mass flow to output.

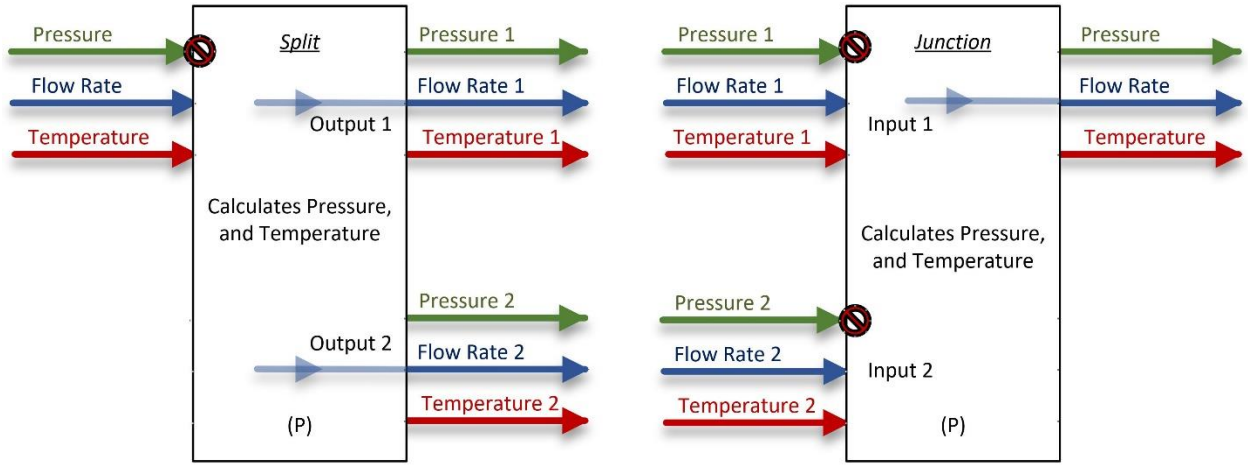

Figure 2.9: Schematic of Split and Junction Components.

Here the capacitance of the temperature state vertex is $M_f \cdot c_{P_f}$ and the edge coefficients are each of the mass flow rates into or out of the component multiplied by the specific heat of the fluid. The graph-based model of a split or junction component is shown in Fig. 2.10. This figure shows two inputs and two outputs to the split/junction component, however there can be up to any number of inputs or any number of outputs, making the component either a junction or a split.

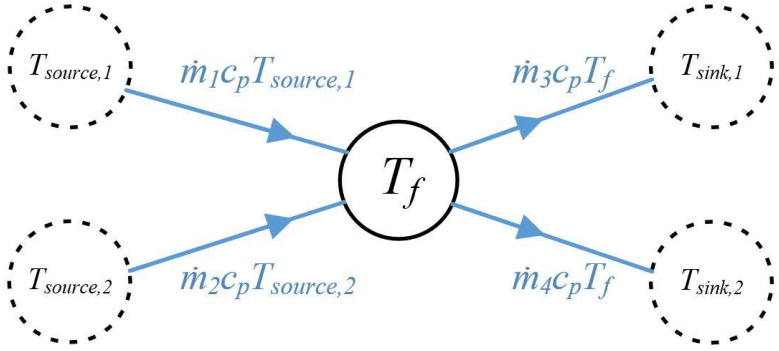

Figure 2.10: Graph-Based Model of a Split/Junction.

2.2.5 Valve Component

The valve component is used to direct mass flow through a system by controlling the percentage split. Like the split component, there is only one vertex with a single temperature state, and this is shown in Eq. (2.54)

$$\dot{T}_f = \frac{\dot{m}_{in} \cdot c_{P_f} \cdot \left(T_{in} - T_f\right)}{M_f \cdot c_{P_f}} \tag{2.54}$$

A schematic of the valve component's inputs and outputs for the DAEMOT model is shown in Fig. 2.11.

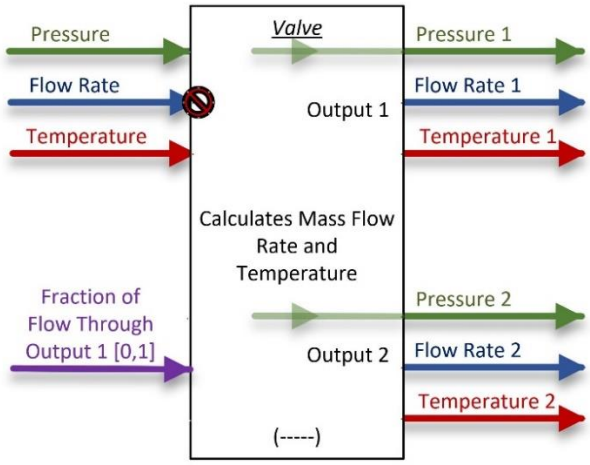

Figure 2.11: Schematic of Valve Component.

The capacitance of the valve is described as $C = M_f \cdot c_{P_f}$, the power flow is given by $P = -\dot{m}_{in} \cdot c_{P_f} \cdot T_f$, and the input power flow, P^s is $\dot{m}_{in} \cdot c_{P_f} \cdot T_{in}$. The valve can be represented by Fig. 2.12 below.

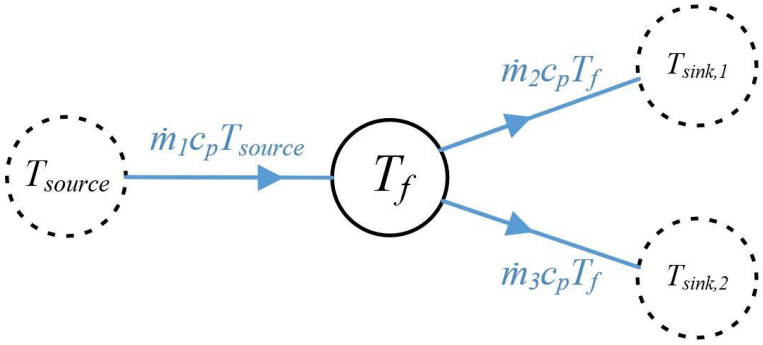

Figure 2.12: Graph-Based Model of a Valve.

2.2.6 Tube Component

The tube component, also called a pipe, is used to direct flow through the system. The tube components are rarely included in the graph-based model, since most of the time the head loss experienced by the tube is so small, that the tube dynamics are negligible. Regardless, they are still commonly used in the DAEMOT models and those governing equations are shown in Eq. (2.55) – (2.56).

$$\dot{T}_f = \frac{\dot{m}_{in} \cdot c_{P_f} \cdot \left(T_{in} - T_f\right)}{M_f \cdot c_{P_f}} - \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_f \cdot c_{P_f}} \tag{2.55}$$

$$\dot{T}_w = \frac{H1 \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_w \cdot c_{P_w}} + \frac{H2 \cdot A_{ht} \cdot \left(T_{amb} - T_w\right)}{M_w \cdot c_{P_w}} \tag{2.56}$$

The schematic of the tube component in DAEMOT is shown in Fig 2.13.

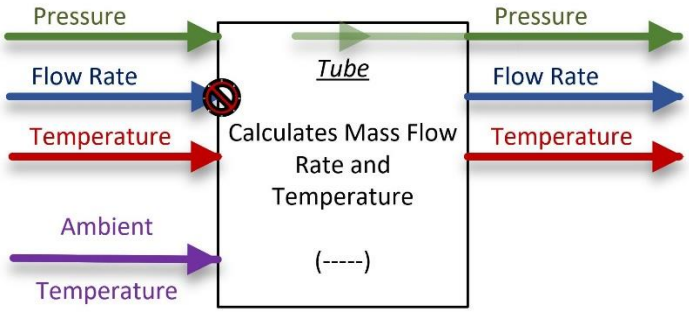

Figure 2.13: Schematic of Tube Component.

Should the tube be included in the graph-based model, it would be represented as a structure shown in Fig. 2.14.

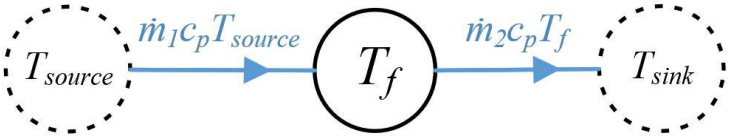

Figure 2.14: Graph-Based Model of a Tube.

2.3 Example System 1

2.3.1 Modeling with the DAEMOT Toolbox

The purpose of this next section is to give of an example of how to build up a DAEMOT model and convert it to a graph-based model. This example will model a flow coming from a mass flow rate source through two cold plates in series before exiting into a pressure sink. Fig. 2.15 shows a schematic of this system and Fig. 2.16 shows the same system, just as a DAEMOT model.

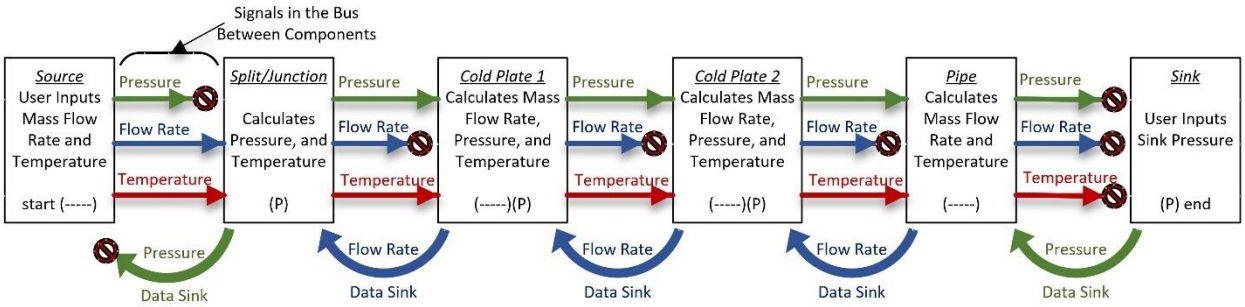

Figure 2.15: Schematic of Example System.

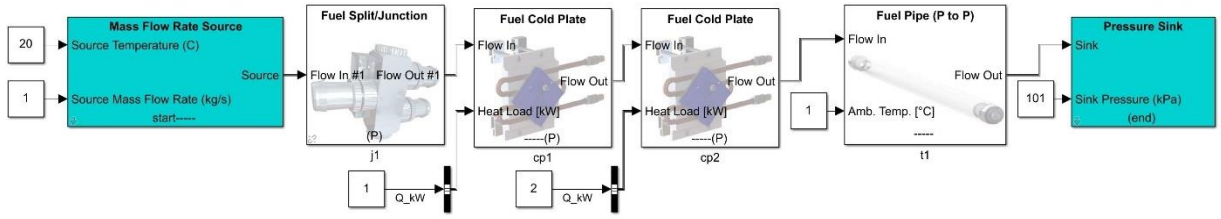

Figure 2.16: Example System as a DAEMOT Model.

Notice how the component blocks alternate between calculating the mass flow rate and pressure. The cold plate component calculates both the mass flow rate and pressure, but it calculates the mass flow rate first, which is why the symbols at its base have the "-----" symbol, which represents mass flow calculation, before the "(P)" symbol, which represents pressure calculation. Therefore, since the cold plate calculates mass flow rate first, it is expecting a pressure input and because it calculates pressure last, it will produce a pressure output. This means we can connect two cold plates together in series without requiring any conversion blocks between them.

There are two components commonly used as make-shift conversion blocks. The split/junction block requires a mass flow rate input and uses knowledge of that mass flow rate

input along with the downstream mass flow rate of the following component to calculate the pressure in the component. When the split/junction component is set to only have one input and one output it can essentially act as a pressure calculator. Similarly, the tube, or pipe, can be used as a mass flow calculator. If the heat transfer coefficient between the ambient temperature is set to zero and the tube length is given a tiny value, the interaction with the ambient air and pressure loss across the tube can be minimized.

Each of the components are given names composed of the component letters followed by a number. This allows the "Batch_Assign_Params.m" script to populate all of the component parameters based on the values set in the "Setup_Component_Params.m" script. Make sure the number of any component does not exceed the total number of that component specified in "Setup_Component_Params.m". The easiest way to avoid this issue is to name the components in ascending order. For example, if you state the number of cold plates (N_cp) is two, you should name the cold plates "cp1" and "cp2." If you name your cold plate components "cp2" and "cp3," only cp2 will have its parameters replaced with the variable names that are updated in "Setup_Component_Params.m," and cp3 will keep the nominal parameter values from the library and cannot be edited as easily.

The cold plate component is expecting a bus with a signal called "Q_kW." Either the heat load should be named "Q_kW" and fed into a bus creator, as shown in Fig. 2.16, or the bus selector under the hood of the cold plate component should be deleted.

Figure 2.17 shows the temperatures in the system immediately after the source, after the fluid has passed through Cold Plate 1, and after the fluid has passed through Cold Plate 2. Since the heat load into Cold Plate 2 is 2 kW, whereas the heat load into Cold Plate 1 is only 1 kW, the increase in temperature due to Cold Plate 2 is expected to be higher than the increase due to Cold Plate 1.

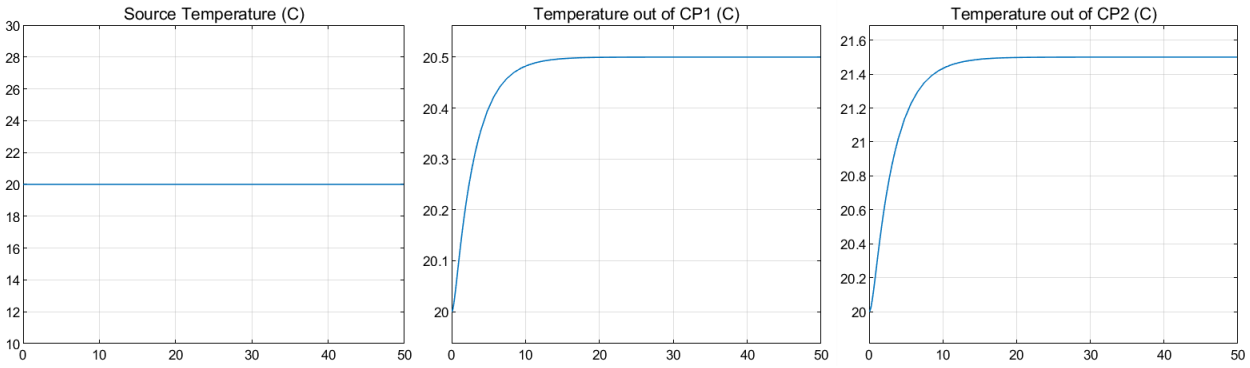

Figure 2.17: Temperatures Throughout Example System.

From Fig. 2.17 it is clear the heat load on Cold Plate 1 has increased the fluid temperature by 0.5 °C and Cold Plate 2 is responsible for the remaining 1 °C increase in the fluid temperature as it flowed through the system.

Suppose the capacitance of the wall in Cold Plate 2 is doubled from 1 kg to 2 kg. The system should eventually reach the same temperature at steady state; however, the added capacitance in Cold Plate 2's wall should slow down the heat transfer rate. This intuition is verified by the temperature plots in Fig. 2.18.

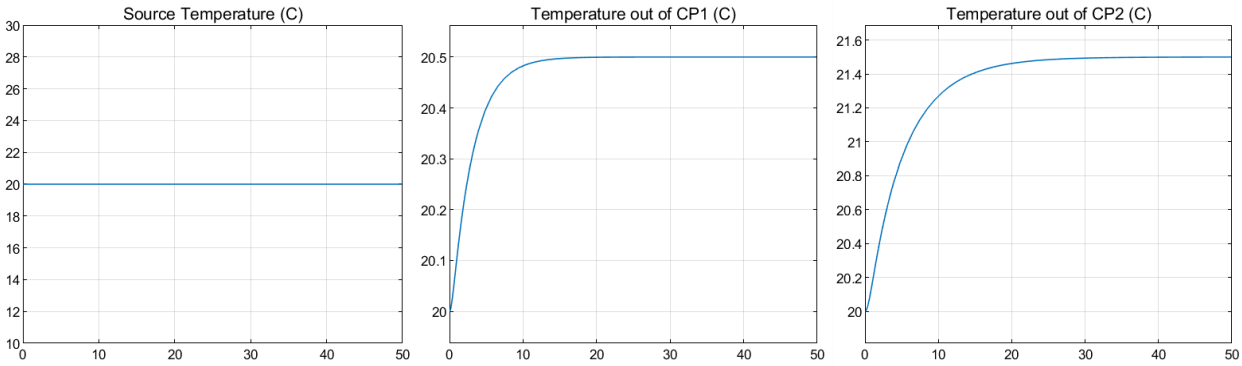

Figure 2.18: Temperatures Through Example with Large Cold Plate 2 Wall Mass.

Suppose the heat transfer coefficient for Cold Plate 2 is halved from 4000 $\frac{W}{m^2 \cdot K}$ to 2000$\frac{W}{m^2 \cdot K}$. The rate of heat transfer should decrease even more, which is demonstrated in Fig 2.19 below.

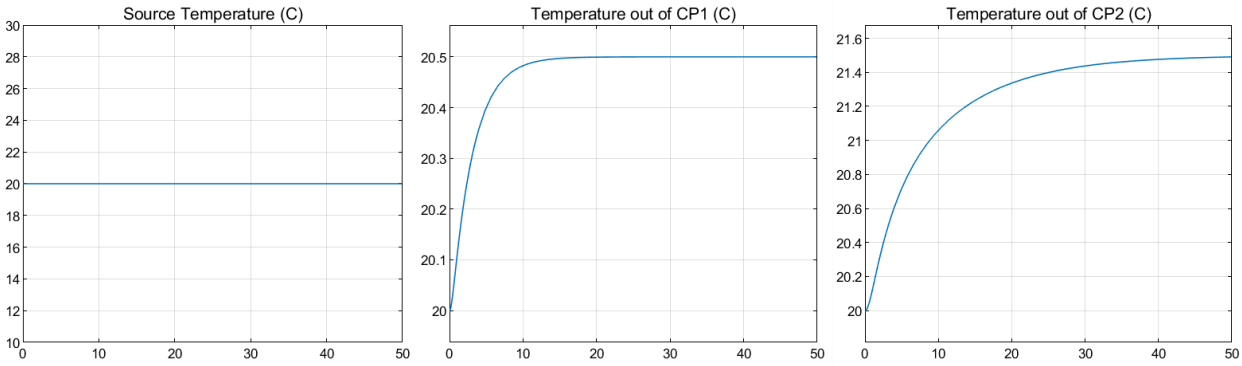

Figure 2.19: Temperatures Through Example with Large Wall and Small HTC for CP2.

This logical verification of the Example 1 system DAEMOT model completes this section on modeling in DAEMOT. The next section will cover how to convert the example system, shown in Fig 2.16, into a graph-based model.

2.3.2 Modeling as a Graph-Based Model

Each of the DAEMOT components has an equivalent graph-based model representation shown in Fig. 2.4, 2.6, 2.8, 2.10, 2.12, and 2.14. Since graph-based models are inherently modular, these individual component graph models can simply be connected, similar to how the DAEMOT model was constructed.

Since the split/junction component used in Example 1 only has one input and one output, its graph model is identical to that of the tube. When all the components that were present in the DAEMOT model are replaced by their graph-based model counterparts, it results in the model shown in Fig. 2.20.

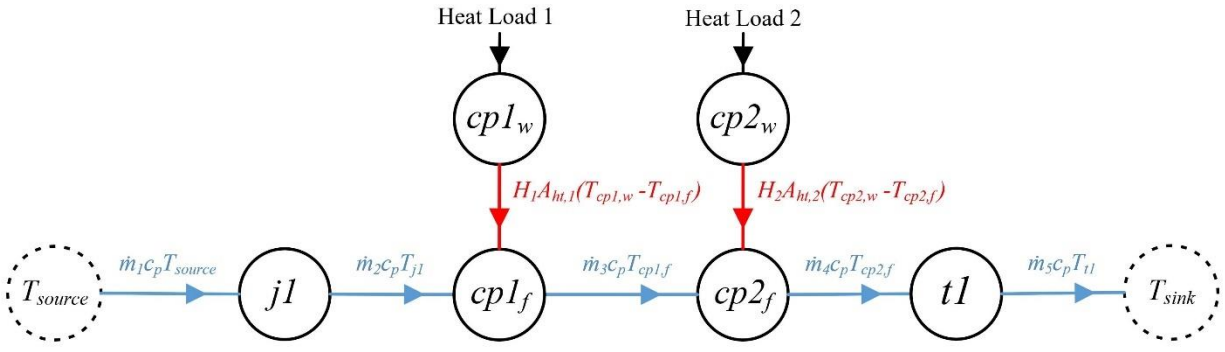

Figure 2.20: Full Graph-Based Model of Example 1.

Both the split/junction and tube component, used to calculate the pressure and mass flow rate for the DAEMOT system, have a negligible effect on the dynamics of the system. Therefore, these components can be excluded from the graph-based model for simplicity. This simplified model is shown in Fig. 2.21.

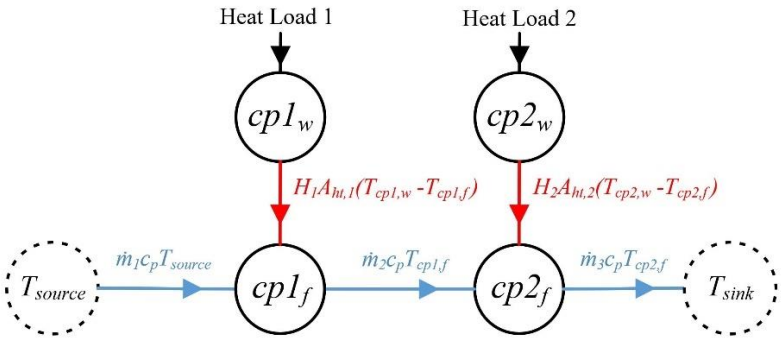

Figure 2.21: Simplified Graph-Based Model of Example 1.

One of the first steps of creating a graph-based model is to number all the vertices and edges to keep track of them. The simplest numbering scheme is shown in Fig. 2.22.

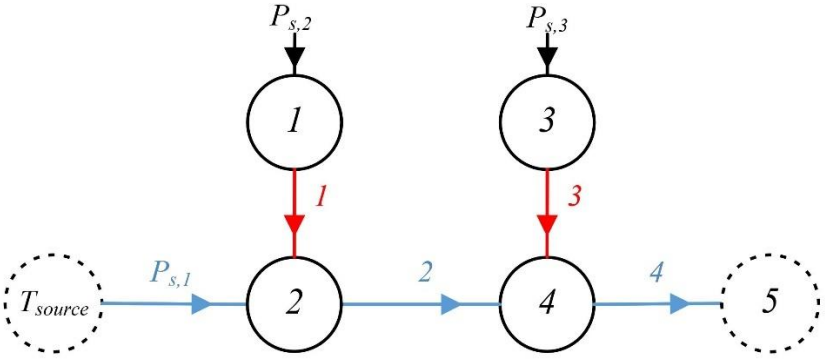

Figure 2.22: Numering Scheme for Example 1 Graph Model.

The model in Fig 2.21 has four non-sink state vertices, numbered 1-4, in Fig. 2.22. There is also one sink vertex, which is numbered 5, in Fig. 2.22. Only vertices 1-4 have associated capacitance values and these can be calculated using Eq. (2.37) and (2.38). There are four edges, numbered 1-4, in Fig. 2.22. The arrow which is labeled $P_{s,1}$ is not an edge, but an input power flow. The heat loads acting on the cold plate walls are also considered input power flows and are numerated as $P_{s,2}$ and $P_{s,3}$. The power flows along edges 1-4 are denoted in Fig. 2.21 and the edge coefficients can be found using Eq. (2.39) and (2.40).

Once all the information regarding the structure of the graph model is entered, both the incidence matrix and D matrix can be built. The incidence matrix maps each of the edges to their neighboring vertices and the D matrix maps each of the input power flows to the correct vertices. These matrices for the model in Fig. 2.22 are shown in Eq. (2.57).

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \qquad D = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.57}$$

The capacitance matrix, C, can be built by diagonalizing the vector of the vertex capacitances. The input power flow matrix, P^s, is a vector of each of the input power flows. The mass flow rates through edges 2 and 4 can be recorded from the DAEMOT model and sent to the graph model as a virtual input. These mass flow rates are then multiplied with the correct edge coefficients to form part of the power flow matrix, P. Then the power flows which require knowledge of the temperature states are calculated using the temperatures of the vertices from the previous time step. Finally, the state derivatives are solved for using Eq. (2.58). Further details regarding graph-based modeling can be found in [19] – [25].

$$\dot{x}(t) = inv(C) * -\bar{M}P(t) + inv(C) * DP^s(t) \tag{2.58}$$

If the setup in Fig. 2.19 is replicated, where the wall mass is doubled and the heat transfer coefficient is halved for Cold Plate 2, the graph-based model in Fig. 2.21 can produce the same results as the DAEMOT model in Fig. 2.16. The system temperatures are compared in Fig. 2.23.

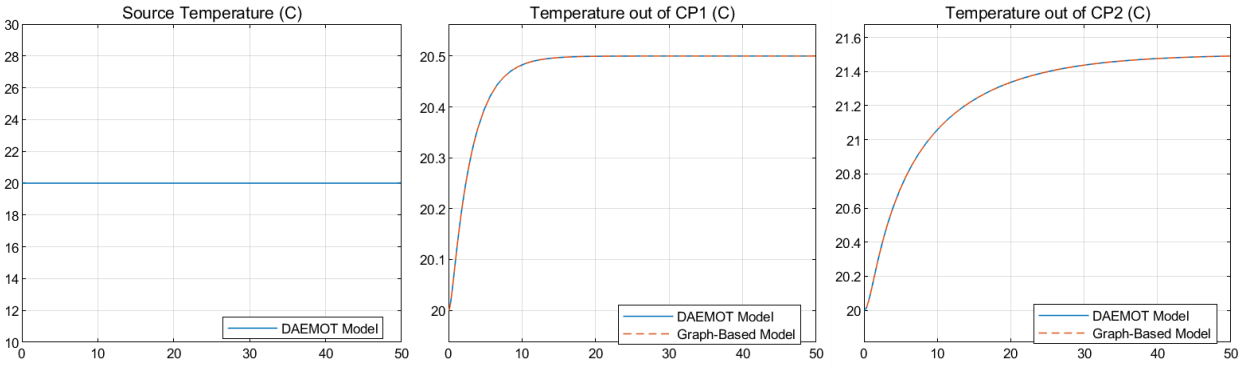

Figure 2.23: Temperatures of DAEMOT Model and Graph-Based Model Compared.

These modeling procedures are used throughout this thesis to model the different systems. The structure of the graph-based modeling framework will be particularly important, since it is integral to the sensitivity analysis outlined in the following chapters.

Chapter 3
Sensitivity Analysis

3.1 Background

Sensitivity analyses can help to understand how variation in a model's parameters or inputs affect the model's output. This information is useful for figuring out which parameters should be the focus for reducing output uncertainty, which parameters are insignificant and can be removed from the model, which inputs most affect the output variability, and which parameters are most correlated to the output. These techniques are also useful for testing to see what might happen to the model if an input parameter is changed without actually having to change the parameter. [26]

Sensitivity analyses can be classified as either local or global. Local sensitivity analyses focus on the model parameters and help determine how their uncertainty, or deviation from an optimal reference parameter set, would affect the model performance. A local sensitivity analysis is valid only for parameter values near the nominal values. The local sensitivity can be solved by analytically calculating the partial derivatives of the model output, Y, with respect to the parameters of interest, θ_j, as shown in Eq. (3.1). Together these partial derivatives are referred to as the Jacobian. Another method for solving the local sensitivity includes using a numerical finite difference approach, further explained in subsection (3.3.1).

$$Local\ sensitivity\ index\ \equiv \frac{\partial Y}{\partial \theta_j} \tag{3.1}$$

A global sensitivity analysis is valid for the entire range of parameter values and thus is difficult to calculate for nonlinear systems. For this reason, most sensitivity analysis methods for nonlinear systems focus on finding the local sensitivity, and the results will be dependent on the nominal set of parameter values, or a specific operating point, when the sensitivity analysis was performed.

If the general form of the system fits and can be represented by the differential equation in Eq. (3.2), then the sensitivity function of that system can be represented by Eq. (3.3).

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, \theta) \tag{3.2}$$

$$\sigma(t, \theta_0) = \frac{\partial x(t, \theta)}{\partial \theta} \tag{3.3}$$

where x is an n-dimensional vector of state variables, θ is a p-dimensional vector of system parameters and t is the independent variable. The states are sometimes referred to as the output of the system, so the sensitivity of the system with respect to its parameters could be called the output sensitivity. The output sensitivity function is defined as the limit in Eq. (3.4).

$$\sigma(t, \theta_0) = \frac{\partial x(t, \theta_j)_i}{\partial \theta_j} = \lim_{\partial \theta_j \to 0} \frac{x_i(t, \theta_0 + \delta\theta_j) - x_i(t, \theta_0)}{\delta \theta_j} \tag{3.4}$$

3.1.1 Input vs. Parameter Sensitivity

Sensitivity analyses can be used to calculate the sensitivities of the system output to both the inputs and the parameters. From the example system of two cold plates in series, laid out in Section 2.3, we can consider the system output to be the temperature of the fluid after it passes through both cold plates and heads towards the sink. The system inputs are the temperature of the fluid entering the system and the heat loads applied to the cold plates. The system parameters of Example 1 from Section 2.3 would include sizing variables, material physical properties, or mass flow rates.

An input sensitivity analysis could determine whether changing the input fluid temperature by a single unit of one degree Celsius would influence the final output temperature more than if either of the cold plate's heat loads were increased by a single unit of 1 kW. This sensitivity analysis would also be able to reveal which cold plate's heat load has a greater effect on the output temperature. Any difference between the heat load effectiveness could be due to differences in either cold plate's nominal parameters, such as sizing or thermal properties, or the discrepancy could be due to the structure of the system. However, since the cold plates in Example 1 are in series, any difference in input heat load influence would be due to parameter differences.

A parameter sensitivity analysis would be able to discern which parameters have the most influence on the system output. Assuming all the inputs (source temperature or heat loads) are held constant, this analysis would be able to reveal information such as how one cold plate's heat

transfer coefficient affects the final temperature of the system compared to another cold plate's heat transfer coefficient. It could also reveal which parameters of the most influential cold plate is the most influential of them all, because maybe a sizing variable can make more of a difference than the heat transfer coefficient. The sensitivity analysis would also be able to compare the effect of adjusting the mass flow rate through the system against the effect of adjusting the sizing and material property parameters. This comparison is useful to inform the engineer of the optimal changes they could make to the system to meet their goals.

3.2 Analytical Techniques

Generally, graph models where the system can be represented by a cluster of vertices and edges are often able to be solved analytically. Some graph models include Markov chain models, Bond Graphs, and the Graph-Based Models detailed in Chapter 2. Sections 3.2.1 – 3.2.5 will touch on some of the work that has already been done to perform analytical sensitivity analyses on these types of models.

The analytical sensitivity of a system is equivalent to the partial derivatives of the system outputs with respect to each of the parameters or inputs, otherwise known as the Jacobian. These derivatives can be solved by symbolic differentiation, as described in Section 3.2.1, or by automatic differentiation, as described in 3.2.2. Section 3.2.3 explains how a transfer function structure inherently communicates the sensitivity of the output with respect to an input. Additionally, if a system was able to be represented as a transfer function, the partial derivative of the transfer function with respect to a parameter is equivalent to finding the sensitivity of the system transfer function to a parameter. To find the sensitivity of the system output with respect to that parameter, the sensitivity of the system transfer function could be multiplied by the system input at the frequency of interest. Sections 3.2.4 and 3.2.5 cover work that has already been done in finding the sensitivities of specific model structures, such as discrete event dynamic systems and bond graphs.

3.2.1 Symbolic Differentiation

Symbolic differentiation is an exact method for finding the sensitivities and can generate the symbolic expressions, which may be useful in combining modular sub-systems. However, this method is very memory intensive and slow. Fig. 3.1 demonstrates an example of a symbolic differentiation calculation done in Matlab, using the matlab function “diff()”.

```
1 -    syms a b c d
2
3 -    eqtn = a^2+3*b+5*c^3+d;
4
5 -    diff_a = diff(eqtn,a);
6 -    diff_b = diff(eqtn,b);
7 -    diff_c = diff(eqtn,c);
8 -    diff_d = diff(eqtn,d);
```

Figure 3.1: Example of Symbolic Differentiation in Matlab.

If this code is run in Matlab, the resulting partial derivatives of the equation “eqtn” are shown in Fig 3.2.

```
diff_a = 2*a
diff_b = 3
diff_c = 15*c^2
diff_d = 1
```

Figure 3.2: Symbolic Differentiation Example Solution.

These derivatives are still in symbolic form and can be replaced by numerical values using the matlab function “subs().” This method is very precise at solving for the exact derivatives of large and complex equations. However, it is computationally expensive and could be very slow for large systems with many variables. An alternative analytical differentiaton method, which is equally exact but has the potential to save on computation time, is automatic differentiation, further explained in Section 3.2.2.

3.2.2 Automatic Differentiation

Automatic differentiation is a way to calculate the Jacobian of a function, or the partial derivatives of the function with respect to some of the inputs (sensitivity). Margossian [27]

describes this method by converting a function, such as Eq. 3.5, into an expression graph, an example of which is shown in Fig. 3.3.

$$f(y,\mu,\sigma) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - log(\sigma) - \frac{1}{2}log(2\pi) \tag{3.5}$$

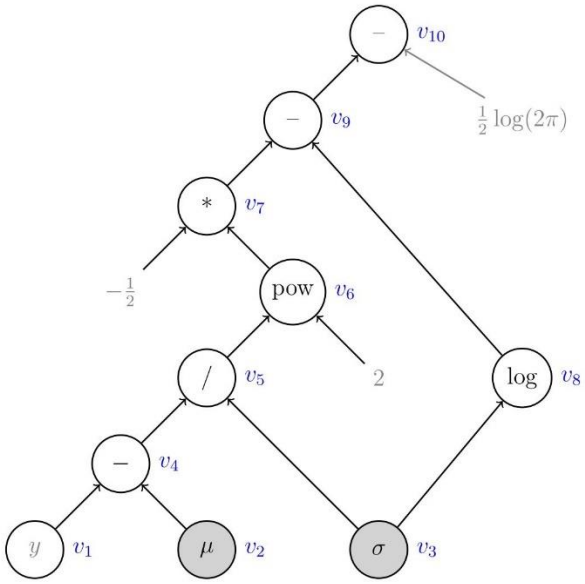

Figure 3.3: An Expression Graph Representation of Eq. 3.5. [27]

The parameters of interest, μ and σ, are represented as gray nodes at the base, or leaves, of the tree, and the final output is the root of the tree. There are two algorithmic mechanisms which can solve this expression graph: the forward mode and the reverse mode, each with its own benefits and tradeoffs.

The Jacobian is a matrix of each of the partial derivatives of f and each element of J is represented by $J_{ij} = \frac{\partial f_i}{\partial x_j}$. If f can be represented as a composite function, $J = J_{h\circ g} = J_h\big(g(x)\big) \cdot J_g(x)$, then $J_{ij} = \frac{\partial f_i}{\partial x_j} = \frac{\partial h_i}{\partial g_1}\frac{\partial g_1}{\partial x_j} + \frac{\partial h_i}{\partial g_2}\frac{\partial g_2}{\partial x_j} + \cdots + \frac{\partial h_i}{\partial g_k}\frac{\partial g_k}{\partial x_j}$ by the chain rule.

The target function f can be broken into several composite functions, corresponding to several Jacobian matrices. I.e., if $f = f^L \circ f^{L-1} \circ \ldots \circ f^1$, then $J = J_L \cdot J_{L-1} \cdot \ldots \cdot J_1$. If u is the initial tangent, or the vector of input variables whose sensitivities are being assessed, and $u_l = J_l \cdot u_{l-1}$, then the forward sweep, $J \cdot u$, can be represented as $J_L \cdot J_{L-1} \cdot \ldots \cdot J_1 \cdot u = J_L \cdot u_{L-1}$.

For the forward mode, the initial tangent, u, should be set to 1 for the j^{th} input and 0 for all the others to isolate one column of the Jacobian matrix and find the partial derivatives of all the system outputs with respect to one input at a time. Thus $J_{\cdot j} = J \cdot u_j$, where $J_{\cdot j}$ is the entire j^{th} column of the Jacobian matrix. Therefore, using the forward mode, it takes as many passes as there are individual inputs to solve for the entire Jacobian matrix. During the trace calculations, an evaluation trace to calculate the values of the function for each of the intermediate steps and a derivative trace to calculate each of the directional derivatives for each node are calculated simultaneously. An example of trace calculations is shown in Fig. 3.4.

Forward evaluation trace	**Forward derivative trace**
$v_1 = y = 10$	$\dot{v}_1 = 0$
$v_2 = \mu = 5$	$\dot{v}_2 = 1$
$v_3 = \sigma = 2$	$\dot{v}_3 = 0$
$v_4 = v_1 - v_2 = 5$	$\dot{v}_4 = \frac{\partial v_4}{\partial v_1}\dot{v}_1 + \frac{\partial v_4}{\partial v_2}\dot{v}_2 = 0 + (-1) \times 1 = -1$
$v_5 = v_4 / v_3 = 2.5$	$\dot{v}_5 = \frac{\partial v_5}{\partial v_4}\dot{v}_4 + \frac{\partial v_5}{\partial v_3}\dot{v}_3 = \frac{1}{v_3} \times (-1) + 0 = -0.5$
$v_6 = v_5^2 = 6.25$	$\dot{v}_6 = \frac{\partial v_6}{\partial v_5}\dot{v}_5 = 2v_5 \times (-0.5) = -2.5$
$v_7 = -0.5 \times v_6 = 3.125$	$\dot{v}_7 = \frac{\partial v_7}{\partial v_6}\dot{v}_6 = -0.5 \times (-2.5) = 1.25$
$v_8 = \log(v_3) = \log(2)$	$\dot{v}_8 = \frac{\partial v_8}{\partial v_3}\dot{v}_3 = 0$
$v_9 = v_7 - v_8 = 3.125 - \log(2)$	$\dot{v}_9 = \frac{\partial v_9}{\partial v_7}\dot{v}_7 + \frac{\partial v_9}{\partial v_8}\dot{v}_8 = 1 \times 1.25 + 0 = 1.25$
$v_{10} = v_9 - 0.5\log(2\pi) = 3.125 - \log(4\pi)$	$\dot{v}_{10} = \frac{\partial v_{10}}{\partial v_9}\dot{v}_9 = 1.25$

Figure 3.4: Forward Trace Calculations [27]

The reverse mode involves calculating the adjoints of various variables with respect to the output variables. A reverse mode sweep calculates $J^T\bar{u}$, where $\bar{u}$ is an initial cotangent vector. Similar to how u was set to be 1 for the j^{th} input and 0 for all others in the forward mode, $\bar{u}$ can be set to 1 for the i^{th} element and 0 for all others to isolate one row of the Jacobian matrix. Thus $J_{i\cdot} = J \cdot J^T\bar{u}_i$, where $J_{i\cdot}$ is the entire i^{th} row of the Jacobian matrix. Once the forward evaluation trace is executed, the reverse mode begins with the final output, sets the adjoint with respect to

itself to 1, and calculates back the successive adjoints until the inputs are reached. These reverse adjoint traces are run for as many times as there are outputs. However, since the forward evaluation trace must be run first to determine the function value and values at each node before the adjoint trace is run, this method is marginally slower and requires more overhead to save the function values until they are needed for the reverse trace. But, if there are significantly more inputs than outputs, the reverse mode performs better for solving the automatic differentiation problem. An example of the reverse trace calculations is shown in Fig. 3.5.

Reverse adjoint trace

$$\bar{v}_{10} = 1$$

$$\bar{v}_9 = \frac{\partial v_{10}}{\partial v_9}\bar{v}_{10} = 1 \times 1 = 1$$

$$\bar{v}_8 = \frac{\partial v_9}{\partial v_8}\bar{v}_9 = (-1) \times 1 = -1$$

$$\bar{v}_7 = \frac{\partial v_9}{\partial v_7}\bar{v}_9 = 1 \times 1 = 1$$

$$\bar{v}_6 = \frac{\partial v_7}{\partial v_6}\bar{v}_7 = (-0.5) \times 1 = -0.5$$

$$\bar{v}_5 = \frac{\partial v_6}{\partial v_5}\bar{v}_6 = 2v_5 \times \bar{v}_6 = 2 \times 2.5 \times (-0.5) = -2.5$$

$$\bar{v}_4 = \frac{\partial v_5}{\partial v_4}\bar{v}_5 = \frac{1}{v_3} \times (-2.5) = 0.5 \times (-2.5) = -1.25$$

$$\bar{v}_3 = \frac{\partial v_5}{\partial v_3}\bar{v}_5 + \frac{\partial v_8}{\partial v_3}\bar{v}_8 = -\frac{v_4}{v_3^2} \times (-2.5) + \frac{1}{v_3} \times (-1) = 2.625$$

$$\bar{v}_2 = \frac{\partial v_4}{\partial v_2}\dot{v}_4 = (-1) \times (-1.25) = 1.25$$

Figure 3.5: Reverse Trace Calculations [27]

Automatic differentiation is a quick and accurate method of finding the system Jacobian, or the sensitivity of each of the outputs to each of the inputs. It is also significantly less memory intensive than a symbolic differentiation method. The main drawback for automatic differentiation algorithms is that they require thought and to be coded carefully, whereas a simple symbolic differentiation function doesn't require much structuring or planning. Since the automatic differentiation method is not automated (yet), it must be implemented carefully for accurate results and optimal performance.

3.2.3 Transfer Function Method

Another method for calculating the sensitivities of a system with respect to its parameters is the transfer function method. First, the system must be represented as a transfer function, $\frac{y(s)}{u(s)}$. This form represents the system's output with respect to the system input in the frequency domain. This representation is an alternative to the output function in the time domain, $y(t)$, which simply represents the system states or output. Therefore, calculating the sensitivity of the transfer function with respect to the parameters yields Eq. 3.6, as opposed to the output sensitivity function, shown in Eq. 3.3. Borutzky [28] defines the unnormalized sensitivity function defined in the frequency domain as follows:

$$S(s) = \frac{\partial \left(\frac{y(s)}{u(s)}\right)}{\partial\ (\theta)} \tag{3.6}$$

where $y(s)$ are the system states and also the outputs, $u(s)$ are the system inputs, and θ are the system parameters for which the transfer function sensitivity will be calculated. Calculating the sensitivities of the system transfer functions requires the system to be linear or linearized so that it can first be represented as a transfer function or a transfer matrix. This may restrict the types of models which this approach is applicable to.

The resulting sensitivities of transfer functions won't depend on the input signal values themselves. Thus, the input signals do not need to be standardized as step or impulse functions to allow for parameter sensitivity comparisons across the various inputs. Another way to look at the transfer function sensitivity is that it determines the sensitivity of the output to each of the parameters, given a single unit of the input. Therefore, increasing or decreasing a particular input value can scale up and down the contribution to the output sensitivity of that input.

Calculating the sensitivity in the frequency domain makes it easier to calculate the system sensitivities. The reason behind this is that for modular modeling frameworks, each component can be represented as a single transfer function and combining individual component's transfer functions is quite trivial. Two transfer functions in series can simply be multiplied together to yield the system transfer function. Similarly, two transfer functions in parallel can be added together to find the system transfer function. These principles can be applied to any system composed of a combination of subsystems in series, parallel, and in feedback loops where each subsystem can be

represented as a separate transfer function. An example of common block diagram reduction rules is shown in Fig. 3.6.

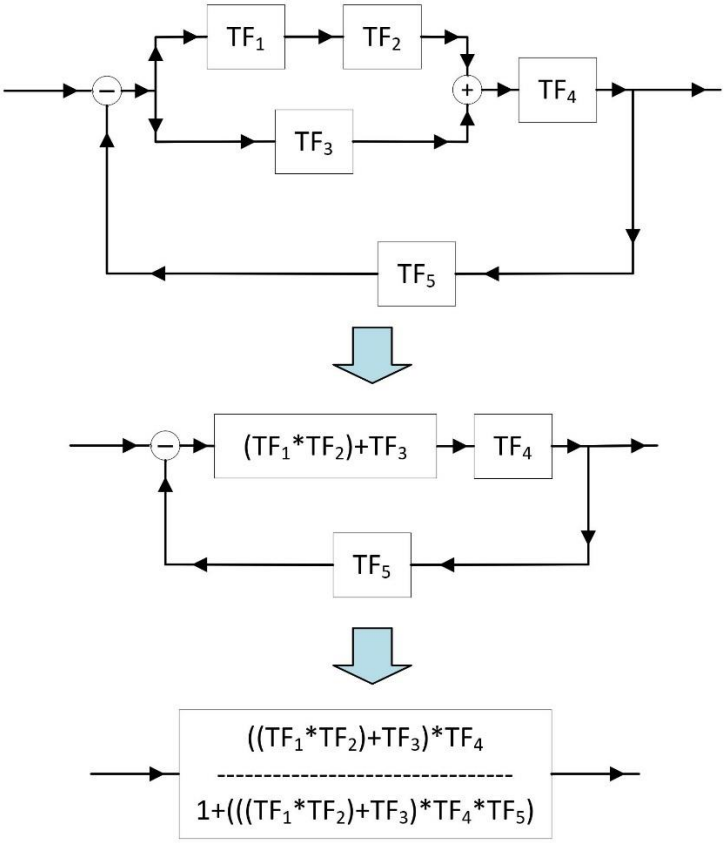

Figure 3.6: Transfer Function Combination Rules.

This method is the one which was chosen for this study to develop a sensitivity analysis method for pre-existing graph-based modeling techniques. These graph-based models are composed of clusters of vertices and edges which represent different components across various energy domains. Each of these components is governed by a set of ordinary differential equations (ODEs) which can be linearized without much trouble to allow them to be written as a transfer function in the frequency domain. Therefore, combining these transfer functions together to form the entire system equation becomes very simple. This ability to build up the system equation from the models so easily is the main reason for selecting this approach. The representation of the system output in a single equation makes calculating the sensitivity of the output with respect to the parameters very simple. Using the symbolic differentiation tools in MATLAB, the partial derivatives of the transfer function equation can be calculated quickly and precisely. Assuming a

reasonably sized system, this is a quick calculation method, especially when compared to numerical approaches.

3.2.4 Discrete Event Dynamic Systems Model Sensitivity

Sensitivity analyses have several potential applications. A common use is to identify the most influential variables to the system outputs when compared to the other variables. This helps to narrow down and focus design efforts on the few most important variables when optimizing the system for desired performance metrics. Sensitivity analyses can also be used to design a more reliable plant, by identifying the parameters or components which contribute the most to the system's uncertainty. This is done by applying a sensitivity analysis to an uncertainty analysis of a system, thus revealing which components should be focused on to improve the overall reliability the most.

One of the most prevalent graphical modeling methods for modeling uncertainty is a fault tree. Ou and Dugan [29] have shown how cyclical modular dynamic fault trees can be simplified as acyclical Markov chains for the sensitivity calculation.

Cao and Ho [30] have done work in designing an infinitesimal perturbation analysis (IPA) sensitivity analysis approach for Markov chains and other discrete event dynamic systems (DEDS). A DEDS differs from a continuous variable dynamic system (CVDS), in that it is not based on a set of differential equations. Instead, it models the complex interactions in the timing of various discrete objects or events, like jobs and resources. The trajectories of a CVDS model are the solutions to a set of differential equations, whereas the trajectories of a DEDS model are piecewise, constant, and event-driven, while still being very much dynamic.

Markov chain models are often used in the intersection of sensitivity and uncertainty analyses. There has been some interesting work done to calculate the sensitivity of these discrete event dynamic system models. This is a very important problem, as many real-world systems such as production lines, computer communication networks, and traffic systems are described as discrete event dynamic systems, and knowing how to optimize throughput, latency, and work-in-progress of such processes would greatly improve many areas of life. The sample paths or behaviors of the system can be better designed to control tasks such as admission of a job into a

queue, routing the job to appropriate resources, and determining the order of service by the resources if the sensitivities of the model to specific system parameters are known. However, since the focus on this thesis is to design a sensitivity analysis for a continuous-variable dynamic system, such as the one described in Section 2.3, a DEDS sensitivity analysis methodology will not be relevant to this study.

3.2.5 Bond Graph Sensitivity Analysis Techniques

Bond Graphs are the most similar pre-existing model structure to the graph-based models considered in this study. Like graph-based models, bond graphs can model energy flow across various energy domains by use of conservation equations [31]. Each of the directional bonds, which represented the energy exchange between two ports of different nodes, has two power variables: effort and flow. Depending on the energy domain, effort and flow may represent different variables. In the electronic energy domain, the effort is voltage, and the flow is current. In the thermodynamic energy domain, effort is temperature and flow is the entropy flow rate. And, in the hydraulic domain effort is pressure and flow is volumetric flow rate. A notable amount of work has already been done to calculate the sensitivity of these bond graphs to their system parameters, therefore these studies are useful in attempting to develop a sensitivity analysis for the graph-based model structure.

Borutzky [32] uses a symbolic method to solve for system sensitivity of a bond graph. He creates an "incremental bond graph," which is like the standard bond graph, except that it relates the incremental efforts and flows (delta e, delta f) to specific parameter changes, represented by sources. These sources, which represent the parameter changes, are the only differences between the incremental bond graphs and the initial bond graphs. Therefore, solving the equations of the incremental bond graph directly solves for the unnormalized sensitivities—the partial derivatives. This methodology is shown below and is equivalent to symbolically differentiating the system equations in state-space form with respect to each of the parameters.

Suppose the system equations of the initial bond graph can be written in the general state space form, shown in Eq. (3.7) – (3.8).

$$\dot{x}(t) = A(p)x(t) + B(p)u(t) \tag{3.7}$$

$$y(t) = C(p)x(t) + D(p)u(t) \tag{3.8}$$

where $\dot{x}(t)$ is the time derivative of the state vector, $x(t)$ is the state vector, $u(t)$ is the input vector, $y(t)$ is the output vector, $A(p), B(p), C(p)$, and $D(p)$ are all time-invariant and parameter dependent matrices, called the system matrix, input matrix, output matrix, and feedthrough matrix, respectively. Then the incremental bond graph can be represented by Eq. (3.9) and (3.10).

$$\Delta\dot{x}(t) = A(p)\Delta x(t) + \tilde{B}(x(t), u(t), p)\Delta p \tag{3.9}$$

$$\Delta y(t) = C(p)\Delta x(t) + \tilde{D}(x(t), u(t), p)\Delta p \tag{3.10}$$

where Δp represents the parameter changes and matrices $\tilde{B}$ and $\tilde{D}$ depend on both the nominal values for $x(t)$ and $u(t)$ which satisfy the initial bond graph equations, and on the parameter p. Eq. (3.11) and (3.12) show the results from taking the Laplace transform of both sets of equations.

$$(sI - A)(\mathcal{L}x)(s) = B\mathcal{L}(s)u \tag{3.11}$$

$$(sI - A)(\mathcal{L}\Delta x)(s) = \left(\mathcal{L}\tilde{B}\right)(s)\Delta p \tag{3.12}$$

When $\mathcal{L}\Delta x$ in Eq. (3.11) and (3.12) is substituted with $\mathcal{L}(C^{-1}\Delta y(t) - C^{-1}\,\tilde{D}(x(t), u(t), p)\Delta p)$, it yields Eq. (3.13) below.

$$\mathcal{L}\Delta y(t)(s) = \left[C(sI - A)^{-1}\mathcal{L}\tilde{B} + \mathcal{L}\tilde{D}\right]\Delta p \tag{3.13}$$

If Δp is considered an infinitesimal parameter change, then Eq. (3.13) approximates the sensitivity matrix in Eq. (3.14).

$$S(s) \coloneqq \frac{\partial \mathcal{L}y}{\partial p} = \left[C(sI - A)^{-1}\mathcal{L}\tilde{B} + \mathcal{L}\tilde{D}\right] \tag{3.14}$$

It is worth noting that the $\tilde{B}$ and $\tilde{D}$ matrices are equivalent to the partial derivatives of the initial bond graph equations with respect to a parameter. This equivalence is shown in Eq. (3.15) – (3.16).

$$\tilde{B} = \frac{\partial}{\partial p}(Ax + Bu)(t) \tag{3.15}$$

$$\tilde{D} = \frac{\partial}{\partial p}(Cx + Du)(t) \tag{3.16}$$

This shows that the sensitivity matrix can be computed by symbolic differentiation and that the incremental bond graph method is mathematically equivalent to simply taking the partial

derivative of a state space model. However, the strength of using the incremental bond graph to derive the sensitivity matrix in Eq. (3.14) is that it is a simple way to calculate the sensitivity matrix when starting from a bond graph model. Once the incremental bond graph is built, the equations are derived in the same way as they would be for a normal bond graph. This incremental bond graph approach is very powerful for calculating all the unnormalized sensitivities of linear time-invariant models in one step using symbolic differentiation.

The main drawback of this approach is that it only works for bond graphs and would require converting the model of interest into an initial bond graph before constructing the incremental bond graph to find the sensitivities. And since a symbolic differentiation is mathematically equivalent, if the model is not already in a bond graph structure, it will be simpler to calculate the partial derivatives with a symbolic toolbox directly. The goal of this thesis is to develop a sensitivity analysis methodology for pre-existing graph-based modeling techniques, which already have many more analysis tools (e.g. model order reduction) for advanced control design than bond graphs have to offer [24]. Therefore, it would be counter-productive to remodel all of the components as bond graph models simply for the sensitivity analysis which can also be obtained by calculating the symbolic differentiation of the system equations with respect to parameters, as described in Section 3.2.1 and will be described further in Chapter 4.

Other work on developing sensitivity analysis tools have been done by Cabanelllas et al. [33] and Gawthrop [34] to develop “pseudo-bond graphs” and “sensitivity bond graphs”, respectively. However, these methods also require the models to be restructured as bond graphs for the sensitivity analysis. It would be much simpler to develop a sensitivity analysis methodology for the existing graph-based modeling technology.

3.3 Numerical Techniques

Numerical techniques are suitable sensitivity analysis methods when the model is very complicated or non-linear that it cannot be solved analytically. Numerical methods usually take longer to solve and do not reach exact answers, but they can reach close approximations and are usually easier to set up. The most common numerical technique is the finite difference method and will be discussed in Section 3.3.1. Then Section 3.3.2 will explain a numerical approach which

uses ordinary differential equation (ODE) solvers to solve for the sensitivities of a complex graph-theoretic model structure.

3.3.1 Finite Difference Method

The finite difference method is sometimes referred to as the brute force or indirect method and is commonly used to solve for local sensitivities of non-linear functions. The output sensitivity function, shown in Eq. (3.17) can be solved by picking a small perturbation of the parameters and calculating the output with and without that perturbation added to the nominal parameter values, then dividing by the chosen perturbation amount.

$$\sigma(t,\theta_0) = \frac{\partial y(t,\theta_j)_i}{\partial \theta_j} = \lim_{\partial\theta_j \to 0} \frac{y_i(t,\theta_0+\delta\theta_j) - y_i(t,\theta_0)}{\delta\theta_j} \tag{3.17}$$

Therefore, the perturbation of the parameters, or $\delta\theta_j$, from the nominal parameter values must be very small to give an accurate partial derivative. The central difference formula, shown in Eq. (3.18), can also be used to approximate the partial derivative.

$$\frac{\partial y(t,\theta_j)_i}{\partial \theta_j} \approx \frac{y_i(t,\theta_0+\delta\theta_j) - y_i(t,\theta_0-\delta\theta_j)}{2\delta\theta_j} \tag{3.18}$$

The principal behind this method is explained by the Taylor series expansion in Eq. (3.19) below.

$$\frac{\partial y(t,\theta_j)_i}{\partial \theta_j} = \frac{y_i(t,\theta_0+\delta\theta_j) - y_i(t,\theta_0) - \frac{1}{2}\left(\frac{\partial^2 y(t,\theta_j)_i}{\partial \theta_j^2}\right)\Delta\theta_j{}^2 - higher\ order\ terms}{\Delta\theta_j} \tag{3.19}$$

De Pauw and Vanrolleghem [35] found that an optimal perturbation factor exists for the central difference formula. This is because there are numerical inaccuracies using the finite difference method when the perturbation factor is too small. And, if the perturbation factor is too large, the central difference formula strays further from the true partial derivative and leads to less accurate results due to model non-linearities. The optimal perturbation factor is calculated by minimizing a set of four criteria which relate to the error, or difference, between the sensitivity function with a positive and with a negative perturbation. These criteria are the sum of squared errors (SSE), the sum of absolute errors (SAE), the maximum relative error (MRE), and the sum

of relative errors (SRE) [35]. Since these errors are calculated around a set of nominal parameters, the optimal perturbation factor is a function of the nominal parameters chosen.

Once the optimal perturbation factor is found, each of the system parameters should be adjusted according to that perturbation factor one at a time. Other than the single parameter whose sensitivity is being assessed, all parameters should be set to nominal values. The parameter which, when perturbed, results in the overall system's largest output difference compared to the nominal system's output is the parameter the system is most sensitive to. Likewise, the second-most sensitive parameter results in the second-largest output difference from the nominal case. In the same way, the parameters for which, when perturbed, the system's output is closest to that of the nominal system are the ones which the system is least sensitive to. These parameters can be removed from time-consuming optimization problems, allowing more resources to be focused on the parameters with the highest sensitivity.

The finite difference method is particularly beneficial because it allows for finding the sensitivity of the system to all the parameters and inputs, even if the system's governing equations are impossible to solve analytically. If the system's equations can be solved with a numerical solver, it is possible to perturb each of the parameters individually and compare the outputs of those simulations.

However, the main drawback of the finite difference method is that since the system equations are being solved numerically, the sensitivity formula can only be approximated. This approximation is also dependent on the chosen perturbation factor, which should be optimized for the set of nominal parameters. Secondly, this method is extremely time consuming. Each simulation or calculation takes time and must be run again for each parameter perturbation. If the number of parameters is large and the simulation is slow, this could easily lead to a very long computation time. Therefore, if a model's governing equations can be solved analytically, using an analytical sensitivity analysis increases the accuracy of the result and takes less time to calculate.

3.3.2 Graph-Theoretic Sensitivity Analysis

Banerjee [36] describes graph-theoretic models as a method for modeling systems. These models, like bond graphs and graph-based models, are oriented linear graphs made up of a combination of nodes and edges. Much like bond graphs, each of the edges have an effort and a flow variable, which are described as "across" and "through" variables, respectively.

At the heart of this proposed algorithm is the ability to generate governing equations and sensitivity equations simultaneously from the linear graph representation of the system. Rather than solve for the sensitivity of pre-existing governing equations, the algorithm generates simpler and more controllable sensitivity and system equations, allowing for more efficient computation.

Once the graph model has been constructed, the governing equations can be extracted from the model and modified to be a linear set of ordinary differential equations (ODEs) in matrix form, shown in Eq. (3.20). This example is for a mechanical, robot system, however it could be generalized to any power domain.

$$M\dot{q} = F(q) \tag{3.20}$$

where M is the mass matrix, q is the state vector and F represents the torques and is a function of q. This linear graph model can be expanded into a sensitivity graph model, from which the sensitivity equations can be derived. A new combined mass matrix, Γ, a new state matrix, ϑ, and a new torque matrix, Ω, are developed as shown in the equations (3.21) – (3.23).

$$\Gamma = \begin{bmatrix} M_b + \sum_{k=1}^{n} \frac{\partial M_{ij}}{\partial q_k} q_{kb} & M \\ M & 0 \end{bmatrix} \tag{3.21}$$

$$\dot{\vartheta} = \begin{bmatrix} \dot{q} \\ \dot{q}_b \end{bmatrix} \tag{3.22}$$

$$\Omega = \begin{bmatrix} F_b + F_q q_b \\ F \end{bmatrix} \tag{3.23}$$

where M_b is the mass matrix, with each element as the partial derivative of M_{ij} with respect to parameter b. Similarly, q_{kb}, is the partial derivative of the k^{th} element of q with respect to b, and q_b is the entire vector q with respect to b. Finally, F_b is the vector F with respect to b.

Together, these expanded matrices and the vector form the governing equations for the sensitivity graph, shown in Eq. (3.24).

$$\Gamma\dot{\vartheta} = \Omega \tag{3.24}$$

This set of sensitivity ordinary differential equations (ODEs) can be solved numerically to find the state variables and their sensitivities with respect to time. This method is very successful at applying numerical sensitivity analysis techniques to a graph model and coming up with a systematic procedure to find the sensitivities with respect to all the parameters. However, this algorithm relies on numerical methods, which cannot give an exact solution and take a lot of time to solve.

Therefore, for this thesis an analytical sensitivity analysis method has been chosen, since the graph-based models can be linearized about a single operating point considering constant mass flow rates. As mentioned in Section 3.2.3, the transfer function sensitivity analysis method will be used to represent the graph-based model as a transfer function equation. Then MATLAB's symbolic differentiation toolbox will be used to solve for the sensitivities of the model to each of the parameters. More details regarding this procedure will be covered in Chapter 4.

Chapter 4
Sensitivity Analysis Methodology for Transfer Function System Representation

4.1 Summary

This section will walk through how to represent two example systems, the one from Section 2.3 and a new system, as set of transfer functions, with one transfer function for each input signal. This system transfer function describes the ratio of the amplitude of the output signal with respect to the amplitude of the input signal for each input signal frequency. This is called the transfer function's gain. There is also a phase component to the transfer function; however, only the gain component will be discussed in this study. The partial derivative of each individual transfer function with respect to the system parameters will be calculated, resulting in a "sensitivity bode plot." This plot describes how sensitive the ratio of the system's output amplitude to the system's input amplitude is to one of the system parameters, given a specific input signal frequency. Each sensitivity bode plot details the contribution from each input to the change in output signal amplitude when the input signal's amplitude and frequency are known. Therefore, at a set of given input signal amplitudes and frequencies, the contributions from each of the input's sensitivity bode plots can be added to determine the total effect each parameter has on the output amplitude. These parameter sensitivities can then be compared and the parameters which have the highest combined effect on the output signal amplitude can be identified. These are the parameters where most of the the optimization efforts should be placed to yield the greatest effect on the system.

4.2 Conducting a Sensitivity Analysis of Example System 1 from Chapter 2

This section covers the sensitivity analysis of Example 1, described in Section 2.3. The graph model is pictured in Fig. 4.1 below for reference.

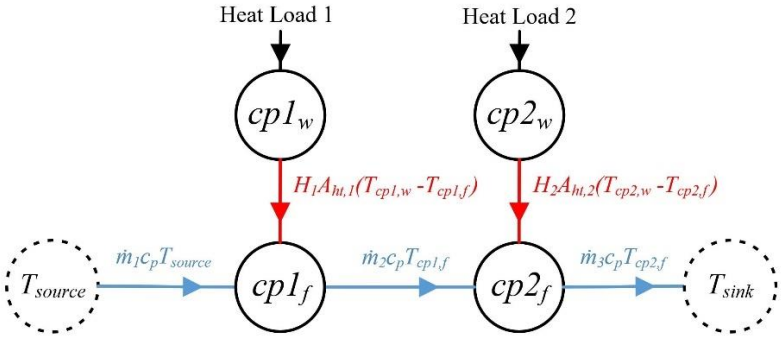

Figure 4.1: Simplified Graph-Based Model of Example System 1

The output of this system is selected to be the temperature of the fluid exiting the system, which is equivalent to the temperature of Cold Plate 2: $T_{cp2,f}$. The numbering scheme is included in Fig. 4.2 below for reference.

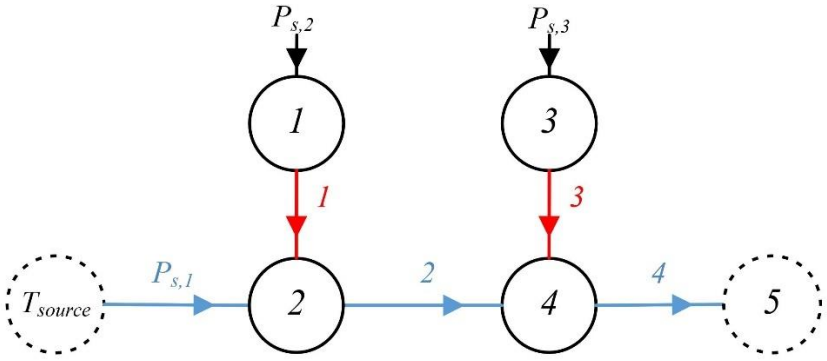

Figure 4.2: Numering Scheme for Example Graph Model.

4.2.1 Representing Example 1 as a set of Transfer Functions

The first step in conducting the transfer function method sensitivity analysis is building up the transfer functions which represent the system model. First, the system output, which the sensitivity analysis will be applied to, must be identified. Each of the individual subsystem level transfer functions must be combined to represent the chosen output as a total system transfer

function. Example System 1 consists of two components, and thus two subsystems, which must be combined to yield the output value. Each component is considered its own subsystem and for the cold plate, this subsystem consists of two capacitances: a wall capacitance and a fluid capacitance. For this study, the output temperature of the fluid as it leaves the system via the sink is chosen as the output value of interest. The sensitivity analysis will be applied to this value to determine its sensitivity to each of the parameters.

The component subsystems must be broken up into transfer functions for each input and output of each component before they can be combined. The governing differential equations of the first component, Cold Plate 1, are shown in Eq. (4.1) and (4.2).

$$\dot{T}_f = \frac{\dot{m}_{in} \cdot c_{P_f} \cdot \left(T_{in} - T_f\right)}{M_f \cdot c_{P_f}} - \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_f \cdot c_{P_f}} \tag{4.1}$$

$$\dot{T}_w = \frac{Q_{amb}}{M_w \cdot c_{P_w}} + \frac{H \cdot A_{ht} \cdot \left(T_f - T_w\right)}{M_w \cdot c_{P_w}} \tag{4.2}$$

The inputs to the component subsystem are the temperature of the incoming fluid, T_{in}, and the heat load applied to the sub-system, Q_{amb}. The outputs are the fluid temperature, T_f, and the wall temperature, T_w. Together, these result in four transfer functions representing the power flow through a cold plate component: $\frac{T_f}{T_{in}}$, $\frac{T_f}{Q_{amb}}$, $\frac{T_w}{T_{in}}$, and $\frac{T_w}{Q_{amb}}$. It is important to keep in mind the end goal of what was chosen as the total system output in identifying which component-level transfer functions affect that value. Since the fluid temperature as it exits the system was chosen as the output, only transfer functions $\frac{T_f}{T_{in}}$ and $\frac{T_f}{Q_{amb}}$ are relevant to the sensitivity analysis.

These signals must also be converted from temperature to energy before they can be added, due to the conservation of energy principles. For example, assume the flow passing through a cold plate reaches 20 °C when there is no heat load, and the heat load heats the fluid in the cold plate to 25 °C when there is no fluid movement. Together, these do not result in an exit temperature of 45 °C. But the energy increase due to both the advective fluid flow and the convective interaction with the heat load can be added since energy is conserved. Then the resulting energy can be converted back to temperature by dividing by the component capacitance. So, rather than setting the component subsystem outputs to the temperature contribution of the fluid due to the inputs, the outputs should be set to the energy contribution: $\frac{E_f}{T_{in}}$ and $\frac{E_f}{Q_{amb}}$. Once the energy contributions from

both inputs are added together, dividing by the fluid capacitance will convert the signal back to temperature to be read as the temperature input for the next sub-system, as illustrated in Fig. 4.3.

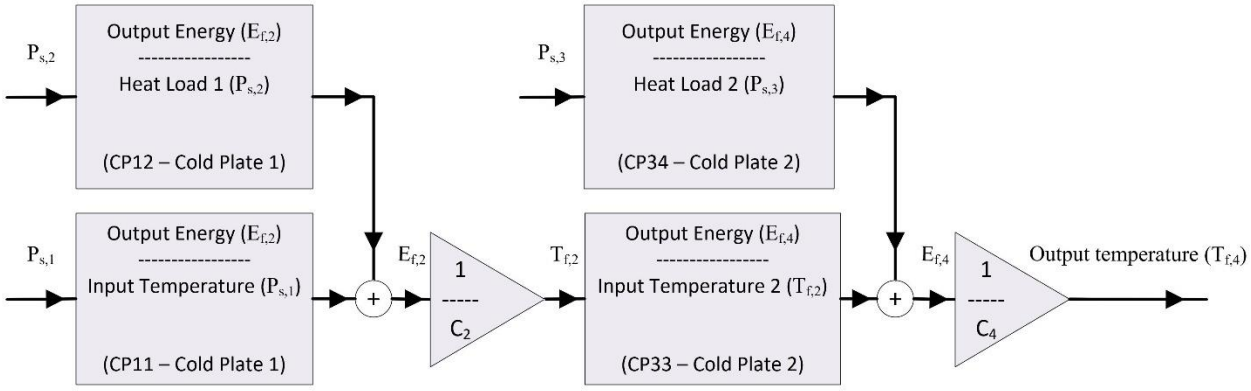

Figure 4.3: Block Diagram Representation of Example System 1

Eq. (4.1) and (4.2) can be converted to Eq. (4.3) and (4.4) to represent the energy of the vertices as the states, instead of the system states being the temperatures of the vertices. Note that the conversion factor of 1000 in Eq. (4.4) converts the input heat load from kW to W.

$$\dot{E}_f = \dot{T}_f \cdot M_f \cdot c_{P_f} = \dot{m}_{in} \cdot c_{P_f} \cdot \left(T_{in} - \frac{E_f}{M_f \cdot c_{P_f}}\right) - H \cdot A_{ht} \cdot \left(\frac{E_f}{M_f \cdot c_{P_f}} - \frac{E_w}{M_w \cdot c_{P_w}}\right) \tag{4.3}$$

$$\dot{E}_w = \dot{T}_w \cdot M_w \cdot c_{P_w} = 1000 \cdot Q_{amb} + H \cdot A_{ht} \cdot \left(\frac{E_f}{M_f \cdot c_{P_f}} - \frac{E_w}{M_w \cdot c_{P_w}}\right) \tag{4.4}$$

Eq. (4.3) and (4.4) can be written in a general state space form as in Eq. (4.5). This will make it easier to convert the system equations from the time domain to the frequency domain.

$$\begin{bmatrix} \dot{E}_f \\ \dot{E}_w \end{bmatrix} = \begin{bmatrix} -\frac{\dot{m}_{in} \cdot c_{P_f}}{M_f \cdot c_{P_f}} - \frac{H \cdot A_{ht}}{M_f \cdot c_{P_f}} & \frac{H \cdot A_{ht}}{M_w \cdot c_{P_w}} \\ \frac{H \cdot A_{ht}}{M_f \cdot c_{P_f}} & -\frac{H \cdot A_{ht}}{M_w \cdot c_{P_w}} \end{bmatrix} \begin{bmatrix} E_f \\ E_w \end{bmatrix} + \begin{bmatrix} \dot{m}_{in} \cdot c_{P_f} & 0 \\ 0 & 1000 \end{bmatrix} \begin{bmatrix} T_{in} \\ Q_{amb} \end{bmatrix} \tag{4.5}$$

For simplicity, in this study the parameters of interest have been selected to be the edge coefficients and vertex capacitances, instead of the sizing parameters or material properties. Therefore, each of the terms in the matrices in Eq. (4.5) must be represented in terms of the vertex capacitance and edge coefficient values. The vertex capacitances of a cold plate component were defined in Eq. (2.37) and (2.38) and are shown again in Eq. (4.6) and (4.7).

$$C_{fluid} = M_f \cdot c_{P_f} = \rho \cdot V \cdot c_{P_f} \tag{4.6}$$

$$C_{wall} = M_w \cdot c_{P_w} \tag{4.7}$$

According to the numbering system in Fig. 4.2, Cold Plate 1's wall vertex is vertex 1 and the fluid vertex is vertex 2. Therefore, Eq. (4.8) and (4.9) show C_1 and C_2.

$$C_1 = M_w \cdot c_{P_w} \tag{4.8}$$

$$C_2 = M_f \cdot c_{P_f} \tag{4.9}$$

The term $\dot{m}_{in} \cdot c_{P_f}$ represents the advective edge entering Cold Plate 1. According to Fig 4.2, this edge is the input power flow, $P_{s,1}$. However, since the cold plate component is flooded, the mass flow rates into and out of the system are equal. This means the edge coming out of Cold Plate 1, E_2, is also equal to the advective edge coefficient: $\dot{m}_{in} \cdot c_{P_f}$. The term $H \cdot A_{ht}$ represents the convective heat transfer between the wall and the fluid vertices, which is represented by E_1, according to Fig. 4.2.

Therefore, replacing these capacitance and edge coefficient values with their symbols yields a simplified state space representation of a cold plate component in Eq. (4.10).

$$\begin{bmatrix} \dot{E}_f \\ \dot{E}_w \end{bmatrix} = \begin{bmatrix} -\frac{E_2}{C_2} - \frac{E_1}{C_2} & \frac{E_1}{C_1} \\ \frac{E_1}{C_2} & -\frac{E_1}{C_1} \end{bmatrix} \begin{bmatrix} E_f \\ E_w \end{bmatrix} + \begin{bmatrix} E_2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} T_{in} \\ Q_{amb} \end{bmatrix} \tag{4.10}$$

Eq. (4.10) only represents the component Cold Plate 1. The system and input matrices, A and B, look different for each component type. It is helpful to represent each component by the basic state space form in Eq. (4.11) and (4.12) and simply substitute back in the capacitances and edge coefficients for each component, once they have all been converted to the frequency domain in Eq. (4.23) and (4.24).

$$\dot{x} = Ax + Bu \tag{4.11}$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 & 0 \\ 0 & b_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{4.12}$$

The substitutions for Cold Plate 1 are shown in Eq. (4.13) – (4.17).

$$\dot{x}_1 = \dot{E}_{f,2};\ x_1 = E_{f,2} \tag{4.13}$$

$$\dot{x}_2 = \dot{E}_{w,1};\ x_2 = E_{w,1} \tag{4.14}$$

$$u_1 = T_{in,1};\ u_2 = Q_{amb,1} \tag{4.15}$$

$$a_1 = -\frac{E_2}{C_2} - \frac{E_1}{C_2};\ a_2 = \frac{E_1}{C_1};\ a_3 = \frac{E_1}{C_2};\ a_4 = -\frac{E_1}{C_1} \tag{4.16}$$

$$b_1 = E_2;\ b_2 = 1000 \tag{4.17}$$

Similarly, the substitutions for Cold Plate 2 are shown in Eq. (4.18) – (4.22). In the interest of grouping co-dependent edge coefficients together for a more accurate sensitivity analysis, all edge coefficients which represent the advective flow through the cold plates will be written as E_2. This is only valid for advective edges which share the same mass flow rate and specific heat value, meaning E_2 is the only edge for which this applies in this system.

$$\dot{x}_1 = \dot{E}_{f,4};\ x_1 = E_{f,4} \tag{4.18}$$

$$\dot{x}_2 = \dot{E}_{w,3};\ x_2 = E_{w,3} \tag{4.19}$$

$$u_1 = T_{in,2};\ u_2 = Q_{amb,2} \tag{4.20}$$

$$a_1 = -\frac{E_2}{C_4} - \frac{E_3}{C_4};\ a_2 = \frac{E_3}{C_3};\ a_3 = \frac{E_3}{C_4};\ a_4 = -\frac{E_3}{C_3} \tag{4.21}$$

$$b_1 = E_2;\ b_2 = 1000 \tag{4.22}$$

The basic state space representation in Eq. (4.12) can be converted to the frequency domain by taking the Laplace transform of it, yielding Eq. (4.23) and (4.24).

$$sX_1(s) = a_1X_1(s) + a_2X_2(s) + b_1U_1(s) \tag{4.23}$$

$$sX_2(s) = a_3X_1(s) + a_4X_2(s) + b_2U_2(s) \tag{4.24}$$

Combining the terms, $X_1(s)$ and $X_2(s)$, results in Eq. (4.25) and (4.26).

$$X_1(s)(s - a_1) = a_2X_2(s) + b_1U_1(s) \tag{4.25}$$

$$X_2(s)(s - a_4) = a_3X_1(s) + b_2U_2(s) \tag{4.26}$$

Substituting Eq. (4.26) into (4.25) results in Eq. (4.27) – (4.29).

$$X_1(s)(s - a_1) = a_2\frac{a_3X_1(s) + b_2U_2}{(s - a_4)} + b_1U_1(s) \tag{4.27}$$

$$\rightarrow X_1(s)\left(s - a_1 - \frac{a_2a_3}{(s - a_4)}\right) = \frac{a_2b_2U_2}{(s - a_4)} + b_1U_1(s) \tag{4.28}$$

$$\rightarrow X_1(s)\left(\frac{s(s - a_4) - a_1(s - a_4) - a_2a_3}{(s - a_4)}\right) = \frac{a_2b_2U_2}{(s - a_4)} + b_1U_1(s) \tag{4.29}$$

To solve for the transfer functions of interest, namely $\frac{E_f}{T_{in}}$ and $\frac{E_f}{Q_{amb}}$, Eq. (4.29) must be arranged into the forms $\frac{X_1(s)}{U_1(s)}$ and $\frac{X_1(s)}{U_2(s)}$ respectively, as shown in Eq. (4.30) and (4.31).

$$\frac{X_1(s)}{U_1(s)} = \frac{b_1(s - a_4)}{s(s - a_4) - a_1(s - a_4) - a_2a_3} = \frac{b_1s - b_1a_4}{s^2 - s(a_1 + a_4) + a_1a_4 - a_2a_3} \tag{4.30}$$

$$\frac{X_1(s)}{U_2(s)} = \frac{a_2 b_2}{s(s-a_4) - a_1(s-a_4) - a_2 a_3} = \frac{a_2 b_2}{s^2 - s(a_1 + a_4) + a_1 a_4 - a_2 a_3} \tag{4.31}$$

This is the general form of a component subsystem's transfer functions. Notice that the signs in the denominators of the transfer functions have alternating signs. However, this does not necessarily mean that the poles are unstable, unless the coefficients a_1, a_2, a_3, and a_4 all have the same sign. Once the values for these coefficients are substituted back in, the denominators reveal all positive coefficients, indicating stable poles and thus stable dynamics in Eq. (4.32) – (4.35). For each individual component, the terms $X_1(s)$, $U_1(s)$, and $U_2(s)$ in Eq. (4.30) and (4.31) need to be replaced with the appropriate outputs and inputs, typically with energy as the output (so it can be added to other energy terms of the same vertex) and with input temperature as one of the input signals. In this case, $X_1(s)$ is standing in for $E_{f,2}(s)$, and $U_1(s)$ and $U_2(s)$ represent $T_{in,1}(s)$ and $Q_{amb,1}(s)$, respectively. The terms a_1, a_2, a_3, a_4, b_1, and b_2 can be replaced with the appropriate vertex capacitances and edge coefficients for the modeled component, such as Eq. (4.16) and (4.17) or Eq. (4.21) and (4.22). The transfer function equations which model Cold Plate 1 are shown in Eq. (4.32) and (4.33) while the equations for Cold Plate 2 are shown in Eq. (4.34) and (4.35). These result in the block diagram representation shown in Fig. 4.4.

$$\frac{E_{f,2}(s)}{T_{in,1}(s)} = \frac{E_{f,2}(s)}{P_{s,1}(s)} = \frac{E_2 s + \frac{E_1 E_2}{C_1}}{s^2 + s\left(\frac{E_2}{C_2} + \frac{E_1}{C_2} + \frac{E_1}{C_1}\right) + \frac{E_1 E_2}{C_1 C_2}} \tag{4.32}$$

$$\frac{E_{f,2}(s)}{Q_{amb,1}(s)} = \frac{E_{f,2}(s)}{P_{s,2}(s)} = \frac{\frac{E_1}{C_1}}{s^2 + s\left(\frac{E_2}{C_2} + \frac{E_1}{C_2} + \frac{E_1}{C_1}\right) + \frac{E_1 E_2}{C_1 C_2}} \tag{4.33}$$

$$\frac{E_{f,4}(s)}{T_{in,2}(s)} = \frac{E_{f,4}(s)}{\frac{E_{f,2,total}(s)}{C_2}} = \frac{E_2 s + \frac{E_2 E_3}{C_3}}{s^2 + s\left(\frac{E_2}{C_4} + \frac{E_3}{C_4} + \frac{E_3}{C_3}\right) + \frac{E_2 E_3}{C_3 C_4}} \tag{4.34}$$

$$\frac{E_{f,4}(s)}{Q_{amb,2}(s)} = \frac{E_{f,4}(s)}{P_{s,3}(s)} = \frac{\frac{E_3}{C_3}}{s^2 + s\left(\frac{E_2}{C_4} + \frac{E_3}{C_4} + \frac{E_3}{C_3}\right) + \frac{E_2 E_3}{C_3 C_4}} \tag{4.35}$$

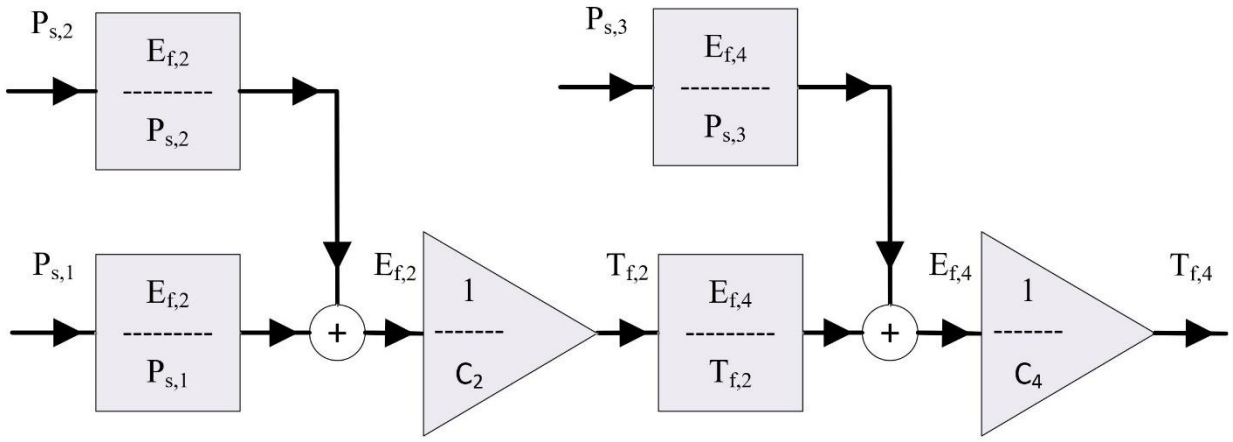

Figure 4.4: Symbolic Block Diagram of Example 1.

The block diagram above can be simplified into Eq. (4.36) using the foundational block diagram reduction rules shown in Fig. 3.6.

$$\left(\left(P_{s,1} * \frac{E_{f,2}(s)}{P_{s,1}(s)} + P_{s,2} * \frac{E_{f,2}(s)}{P_{s,2}(s)}\right) * \frac{1}{C_2} * \frac{E_{f,4}(s)}{T_{f,1}(s)} + P_{s,3} * \frac{E_{f,4}(s)}{P_{s,3}(s)}\right) * \frac{1}{C_4} = T_{f,4} \tag{4.36}$$

The last step before taking the partial derivatives of each of the system transfer functions is to represent the system output of interest, $T_{f,4}$, in terms of each of the system inputs, namely $P_{s,1}$, $P_{s,2}$, and $P_{s,3}$. This yields the total system transfer functions in Eq. (4.37) – (4.39).

$$\frac{T_{f,4}(s)}{P_{s,1}(s)} = \frac{E_{f,2}(s)}{P_{s,1}(s)} * \frac{1}{C_2} * \frac{E_{f,4}(s)}{T_{f,2}(s)} * \frac{1}{C_4} \tag{4.37}$$

$$\frac{T_{f,4}(s)}{P_{s,2}(s)} = \frac{E_{f,2}(s)}{P_{s,2}(s)} * \frac{1}{C_2} * \frac{E_{f,4}(s)}{T_{f,2}(s)} * \frac{1}{C_4} \tag{4.38}$$

$$\frac{T_{f,4}(s)}{P_{s,3}(s)} = \frac{E_{f,4}(s)}{P_{s,3}(s)} * \frac{1}{C_4} \tag{4.39}$$

When the transfer functions $\frac{E_{f,2}(s)}{P_{s,1}(s)}$, $\frac{E_{f,2}(s)}{P_{s,2}(s)}$, $\frac{E_{f,4}(s)}{T_{f,2}(s)}$, and $\frac{E_{f,4}(s)}{P_{s,3}(s)}$ in Eq. (4.37) – (4.39), are replaced by their capacitance and edge coefficient values from Eq. (4.32) – (4.35), they yield the final form of the system transfer functions which will be used in the sensitivity analysis. These equations are shown below in Eq. (4.40) – (4.42).

$$\frac{T_{f,4}(s)}{P_{s,1}(s)} = \frac{E_2 s + \frac{E_1 E_2}{C_1}}{s^2 + s\left(\frac{E_2}{C_2} + \frac{E_1}{C_2} + \frac{E_1}{C_1}\right) + \frac{E_1 E_2}{C_1 C_2}} * \frac{1}{C_2} * \frac{E_2 s + \frac{E_2 E_3}{C_3}}{s^2 + s\left(\frac{E_2}{C_4} + \frac{E_3}{C_4} + \frac{E_3}{C_3}\right) + \frac{E_2 E_3}{C_3 C_4}} * \frac{1}{C_4} \tag{4.40}$$

$$\frac{T_{f,4}(s)}{P_{s,2}(s)} = \frac{\frac{E_1}{C_1}}{s^2 + s\left(\frac{E_2}{C_2} + \frac{E_1}{C_2} + \frac{E_1}{C_1}\right) + \frac{E_1 E_2}{C_1 C_2}} * \frac{1}{C_2} * \frac{E_2 s + \frac{E_2 E_3}{C_3}}{s^2 + s\left(\frac{E_2}{C_4} + \frac{E_3}{C_4} + \frac{E_3}{C_3}\right) + \frac{E_2 E_3}{C_3 C_4}} * \frac{1}{C_4} \quad (4.41)$$

$$\frac{T_{f,4}(s)}{P_{s,3}(s)} = \frac{\frac{E_3}{C_3}}{s^2 + s\left(\frac{E_2}{C_4} + \frac{E_3}{C_4} + \frac{E_3}{C_3}\right) + \frac{E_2 E_3}{C_3 C_4}} * \frac{1}{C_4} \quad (4.42)$$

4.2.2 Finding the Sensitivities of each of the System Transfer Functions

As discussed in Section 3.2.3, the sensitivity of a transfer function to system parameters is found by taking the partial derivative of the transfer function with respect to the parameters of interest. This is re-iterated in Eq. (4.43) below.

$$S(s) = \frac{\partial\left(\frac{y(s)}{u(s)}\right)}{\partial\,(\theta)} \quad (4.43)$$

Therefore, the sensitivities of the three system transfer functions, $\frac{T_{f,4}(s)}{P_{s,1}(s)}$, $\frac{T_{f,4}(s)}{P_{s,2}(s)}$, and $\frac{T_{f,4}(s)}{P_{s,3}(s)}$, can be found by taking the partial derivatives of each of them individually to each of the vertex capacitances and edge coefficients.

Suppose there is interest in knowing the effect parameter C_2 has on the output temperature $T_{f,4}$, compared to parameter C_4. The plots in Fig. 4.5 demonstrate the sensitivity bode plots for each input's transfer function.

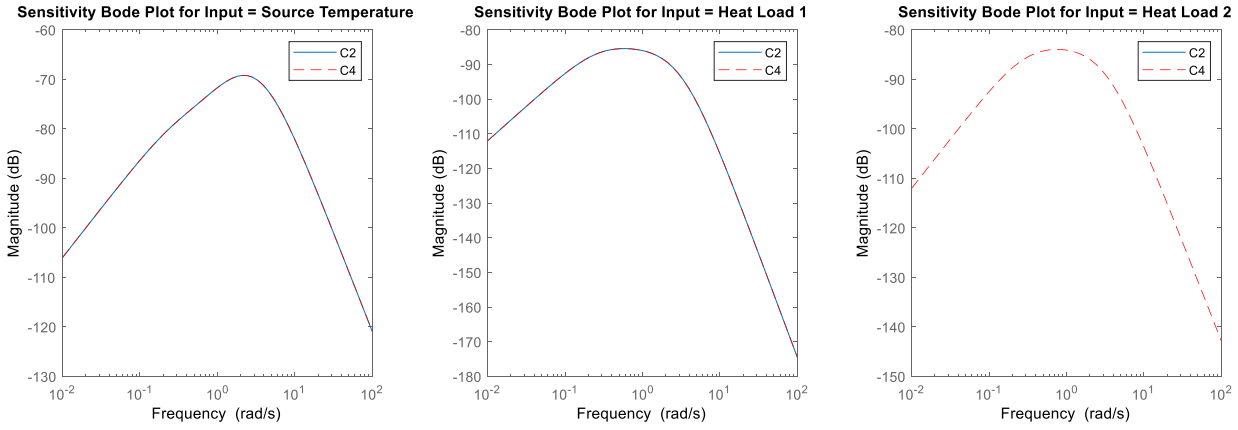

Figure 4.5: Sensitivities of the System Transfer Functions to Parameters C_2 and C_4.

Notice that the transfer function relating the output temperature to the input Heat Load 2 is only dependent on parameter C_4, since parameter C_2 is the fluid vertex of Cold Plate 1, which is not along the path of the Heat Load 2 transfer function. Also notice that at the lowest frequencies the sensitivities of each of the transfer functions to either parameter decreases to miniscule values. This suggests that at steady state neither parameter has any effect on the output signal's amplitude. Therefore, to see an effect on the output signal when compared to the nominal case, the input signal needs to have a non-zero frequency.

Let all three inputs have a frequency of 0.5 rad/s. The transfer functions for each of these inputs can be used to determine the output signal amplitude for nominal parameter values. This is done by selecting the frequency of each of the input signals, finding the ratio of output amplitude to input amplitude at that frequency, and multiplying that ratio by the input amplitude for each input. This results in three different contributions to the total output amplitude. Adding the three contributions from each of the inputs follows from the principle of superposition, as shown in Fig. 4.6.

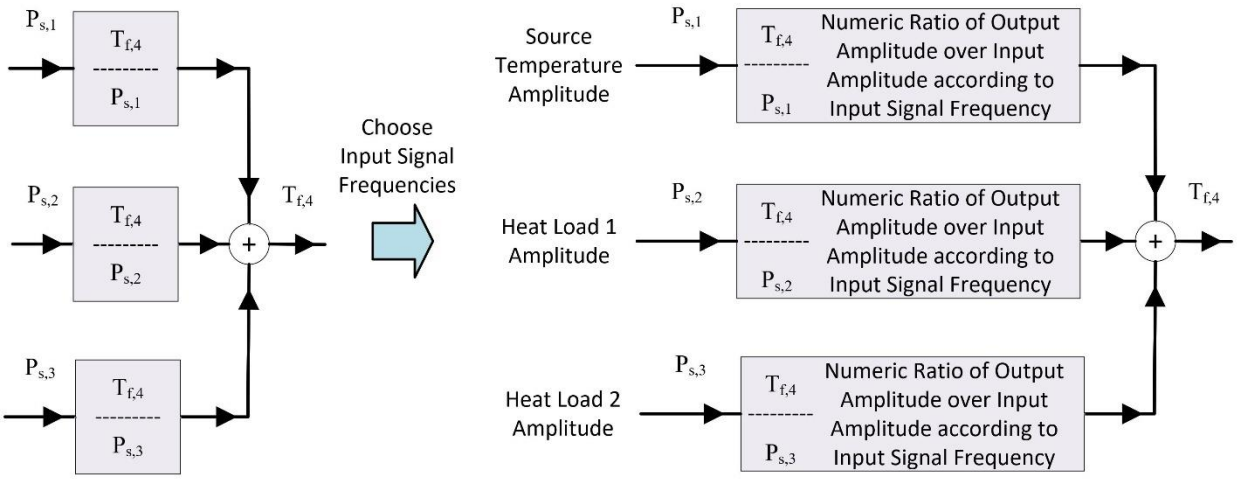

Figure 4.6: Applying Superposition Principle at Specific Input Frequencies.

To demonstrate this by numerical simulation, let the source temperature have an amplitude of 20 (°C), and Heat Load 1 and 2 have amplitudes of 1 (kW), and 2 (kW), respectively. Let the frequency for all the input signals be 0.5 rad/s. Fig. 4.7 shows the resulting output amplitude estimation following from the procedure illustrated in Fig. 4.6.

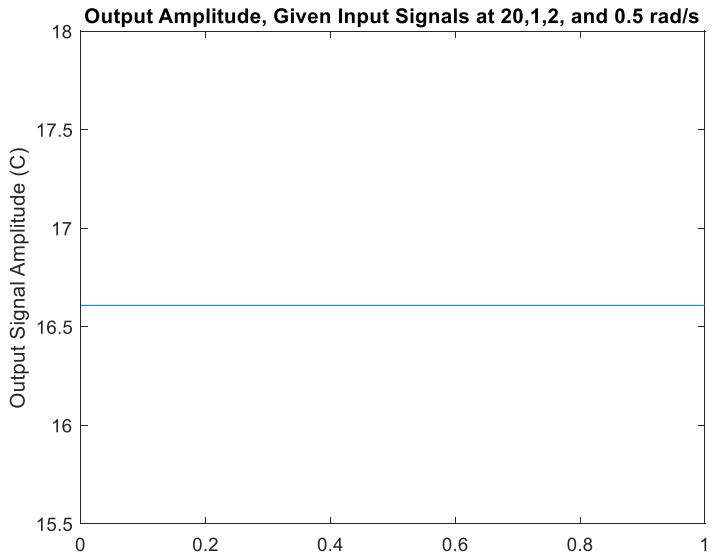

Figure 4.7: Output Amplitude Based on Input Signal Amplitudes and Frequencies.

The estimated output amplitude of Example System 1 is 16.61 as shown in Fig. 4.7 above. Note that the x scale is meaningless for all such superposition plots. This plot represents the single value 16.61 as the output amplitude, given the input signal amplitudes and frequencies. The numerical verification of the actual output signal (purple line) is shown in Fig. 4.8.

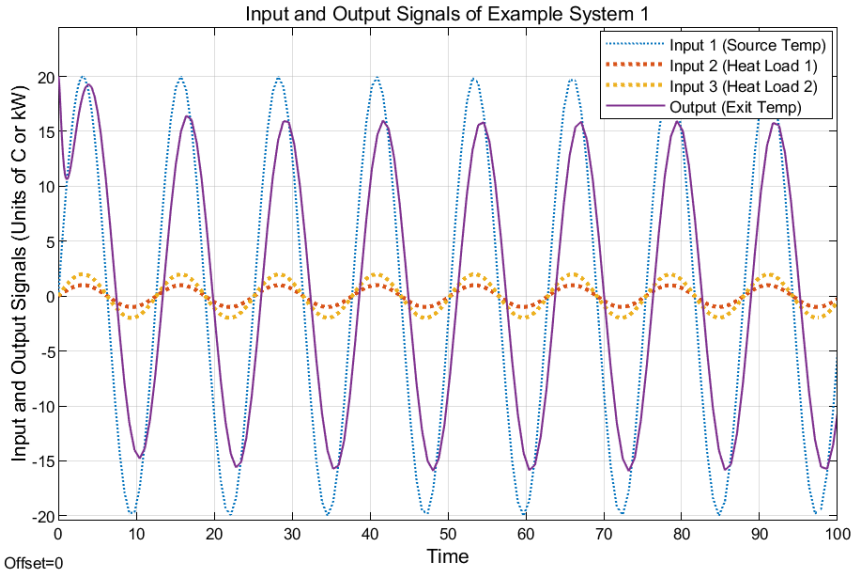

Figure 4.8: Numerical Verification of Output Signal Amplitude.

Finding the total effect of each parameter on the final output signal amplitude is a similar process. Instead of multiplying the output to input amplitude ratio by the input amplitude, the

partial derivative of the output to input amplitude, or the input-specific transfer function, should be multiplied by the input signal amplitude. Once the sensitivities of each transfer function to a specific parameter are calculated at the chosen input signal frequencies, these can be added together to find the total impact the parameter has on the output signal amplitude. These steps are illustrated in Fig. 4.9 below.

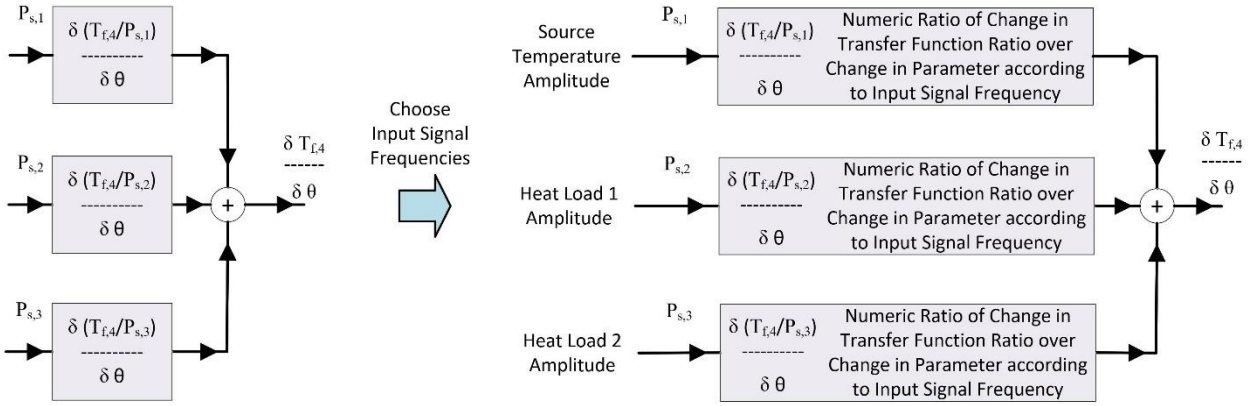

Figure 4.9: Applying Superposition Principle to Transfer Function Sensitivity Function.

This tool can help figure out whether parameter C_2 or C_4 is more influential to the output amplitude, given all three inputs have frequencies of 0.5 rad/s. Returning to the sensitivity bode plots in Fig. 4.5, recall that at 0.5 rad/s, both the Source Temperature and Heat Load transfer functions are equally sensitive to C_2 and C_4. However, Heat Load 4 is only sensitive to C_4. So, altogether the output amplitude will be more sensitive to parameter C_4. As expected, Fig. 4.10 predicts that when all input signals are at 0.5 rad/s, the output temperature is more sensitive to C_4 than to C_2.

Like Fig. 4.7 representing the output amplitude, Fig. 4.10 represents the sensitivity of the output amplitude with respect to each parameter. These sensitivities are single values and could have been communicated as a bar chart with one bar for each parameter. However, this would make it more difficult to compare the far-left bars with the far-right ones if there were more than a few parameters being plotted. For this reason, a 2-dimensional plot was chosen instead to better compare each of the parameter sensitivities. Each parameter sensitivity is plotted as a horizontal line across the figure. This means that the x-axis is meaningless and the y-axis, which shows the sensitivities of the output signal amplitude, is the focus of the plot.

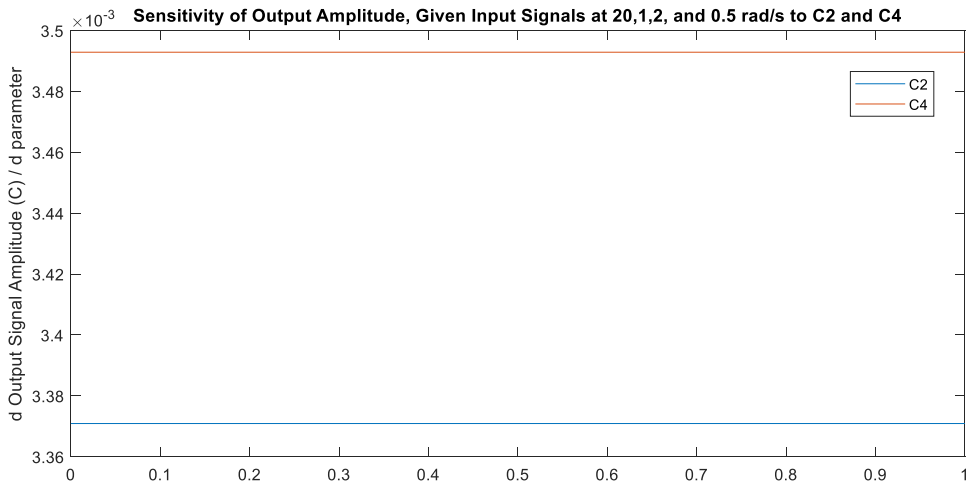

Figure 4.10: Sensitivity of Output Amplitude to Parameters C_2 and C_4.

Since the difference in output signal amplitude for the parameters is so small, the numerical verification, shown in Fig. 4.11, shows a marginal difference in influence between parameters C_2 and C_4.

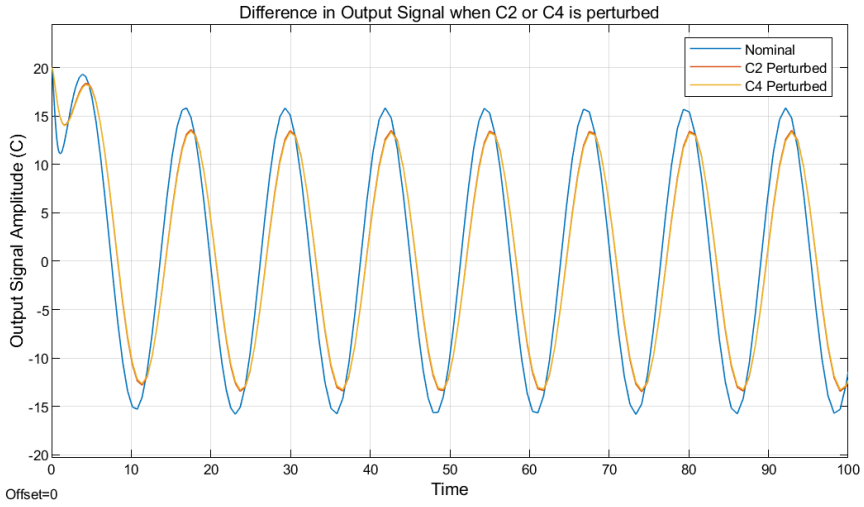

Figure 4.11: Numerical Verification of Parameter C_4 having a higher influence on the Output Amplitude.

Therefore, let the amplitude of Heat Load 2 be 50 kW to highlight this difference in sensitivity better, when shown through numerical simulation. Fig. 4.12 shows the new predicted sensitivity of the output amplitude to parameters C_2 and C_4, and Fig. 4.13 shows the numerical verification of how each of the parameters influence the output amplitude when they are perturbed.

Notice how, in Fig. 4.13, when C_4 is perturbed the resulting output signal has a smaller amplitude and is further from the nominal case (where neither parameter is perturbed) than when C_2 is perturbed.

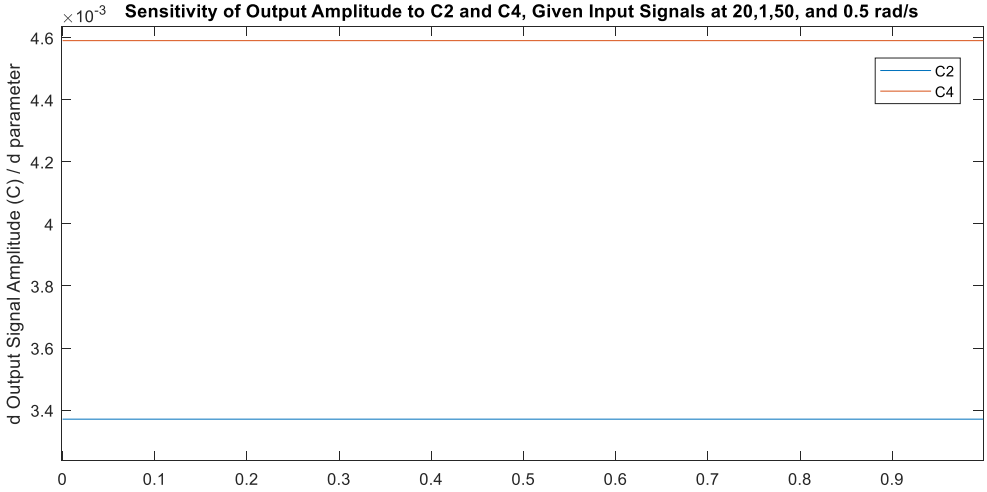

Figure 4.12: Sensitivity of Output Amplitude to C_2 and C_4 when Heat Load 2 = 50 kW.

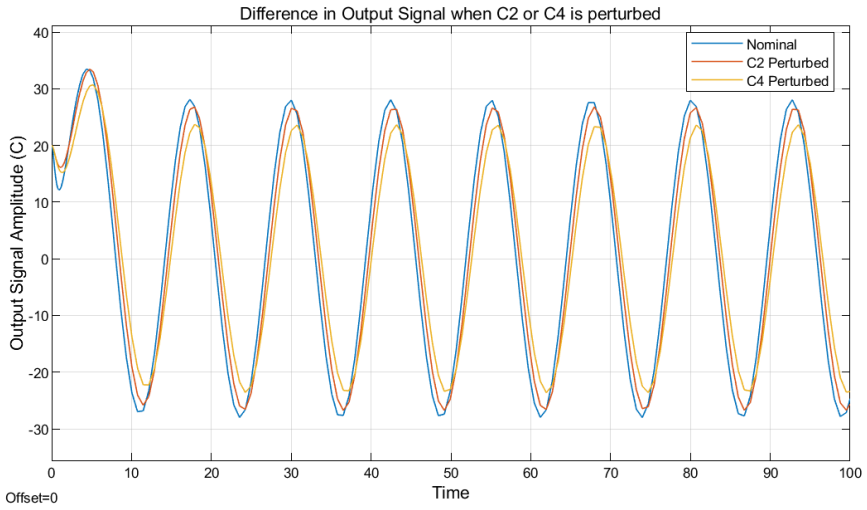

Figure 4.13: Numerical Verification of Parameter C_4 having a higher influence when Heat Load 2 = 50 kW.

Now that the basics of comparing the sensitivities of the output amplitude to two parameters are understood, the next subsection will cover identifying and numerically verifying the most influential parameters of this system, given a set of input signal amplitudes and frequencies.

4.2.3 Identifying the Most Influential Parameters to the System Output Amplitude

To identify the parameters the system is most sensitive to, the "sensitivity bode plots", or the partial derivatives of each of the input-specific transfer functions to the parameters, are crucial. These are shown below in Fig. 4.14 – 4.16.

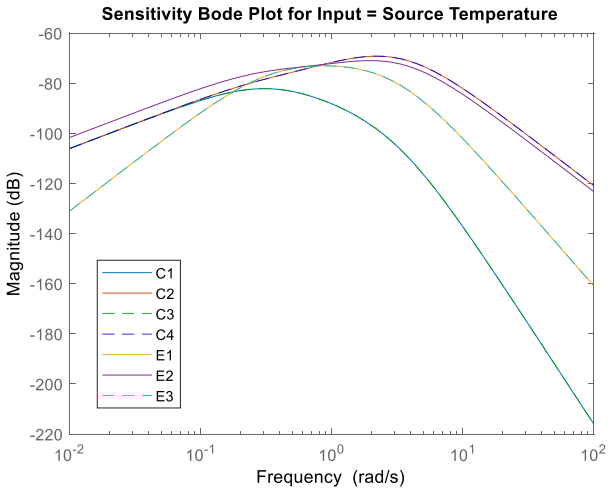

Figure 4.14: Sensitivity Bode Plot of Input 1: Source Temperature.

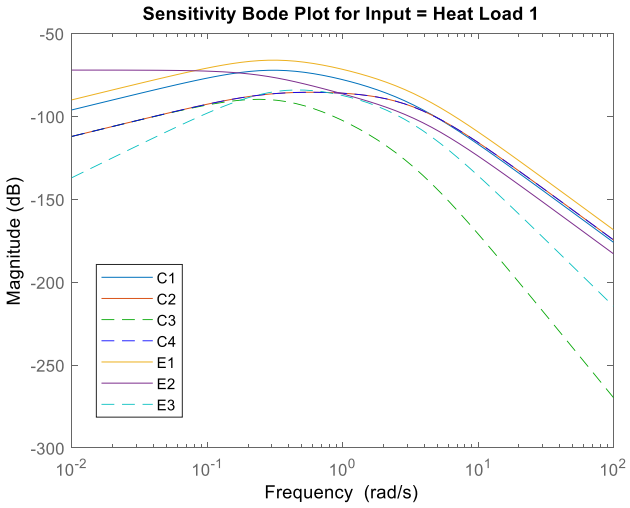

Figure 4.15: Sensitivity Bode Plot of Input 2: Heat Load 1.

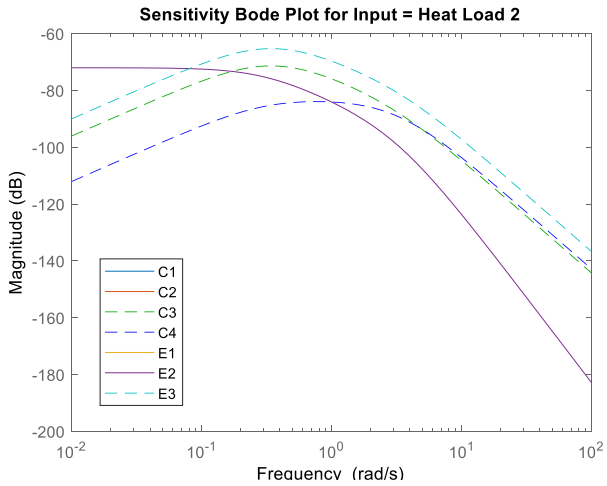

Figure 4.16: Sensitivity Bode Plot of Input 3: Heat Load 2.

Each sensitivity bode plot is zoomed in around the frequency 1 rad/s in Fig. 4.17 – 4.19.

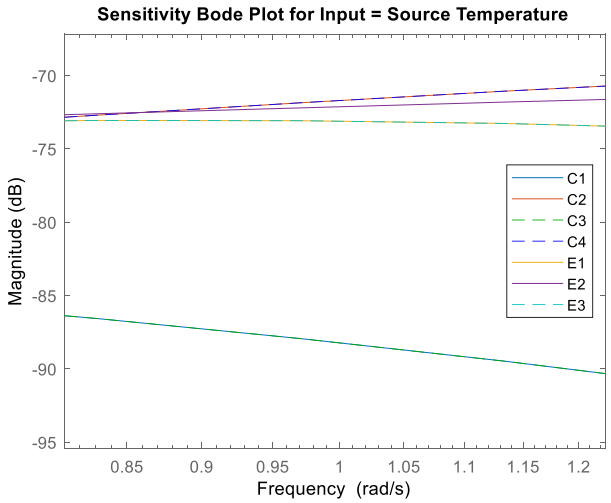

Figure 4.17: Close up of Fig. 4.14 around 1 rad/s.

The parameters the source transfer function appears to be most sensitive to, for an input signal at 1 rad/s, are C_2 and C_4, closely followed by E_2, and then by E_1 and E_3. According to Fig. 4.2, these are the fluid capacitances of Cold Plate 1 and 2, the advective edge of fluid flow through the system and the convective edges between Cold Plate 1 and 2's fluid and wall vertices. This makes intuitive sense that the transfer function which relates the output temperature to the input temperature the most are the fluid capacitances. Since for this input-specific transfer function all the heat loads are considered to be zero, the capacitances of the cold plate walls will not be as influential to the output temperature.

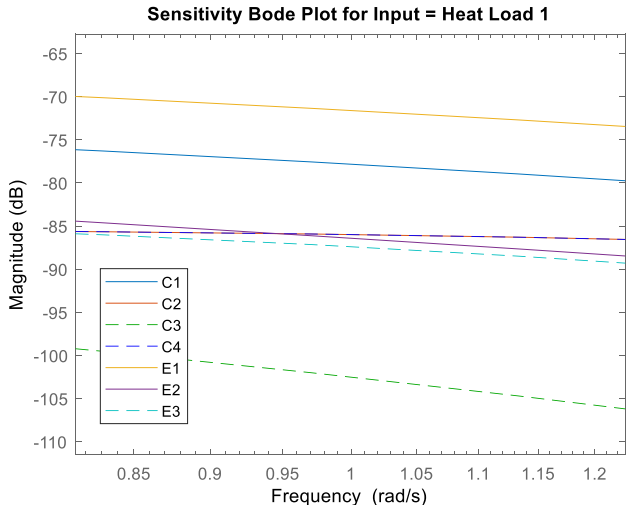

Figure 4.18: Close up of Fig. 4.15 around 1 rad/s.

The parameters that the Heat Load 1 transfer function is most sensitive to include E_1, the convective edge between Cold Plate 1's fluid and wall vertices, followed by C_1, the wall capacitance of cold plate 1 which Heat Load 1 is applied to. This also makes intuitive sense that the cold plate wall capacitance and the heat transfer coefficient between the cold plate fluid and the wall for Cold Plate 1 would have the greatest impact on the output temperature for the transfer function whose input is the heat load on Cold Plate 1.

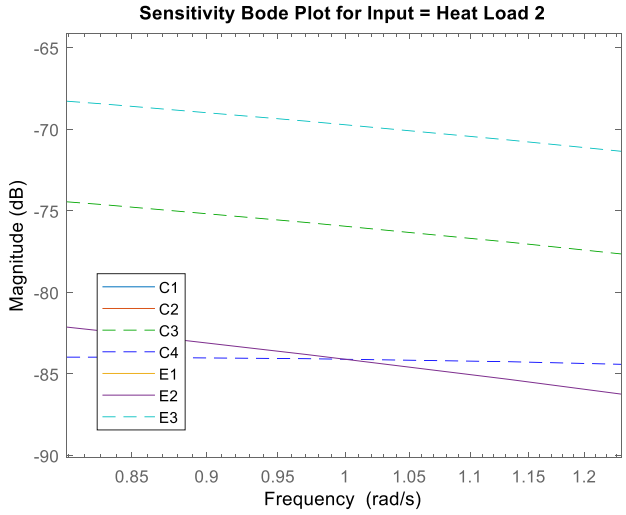

Figure 4.19: Close up of Fig. 4.16 around 1 rad/s.

The Heat Load 2 transfer function is most sensitive to the parameter E_3, which represents the convective edge between Cold Plate 2's fluid and wall vertices, followed by C_3, the wall

capacitance of Cold Plate 2. Similar to the case for Cold Plate 1, it also makes intuitive sense that the wall capacitance and heat transfer coefficient for Cold Plate 2 are the most influential for the energy transfer from Heat Load 2 to the output temperature.

When all three sensitivity bode plots are multiplied by the amplitude of their input signal and then added together, following the procedure from Fig. 4.9, they result in the ranking shown in Fig. 4.20.

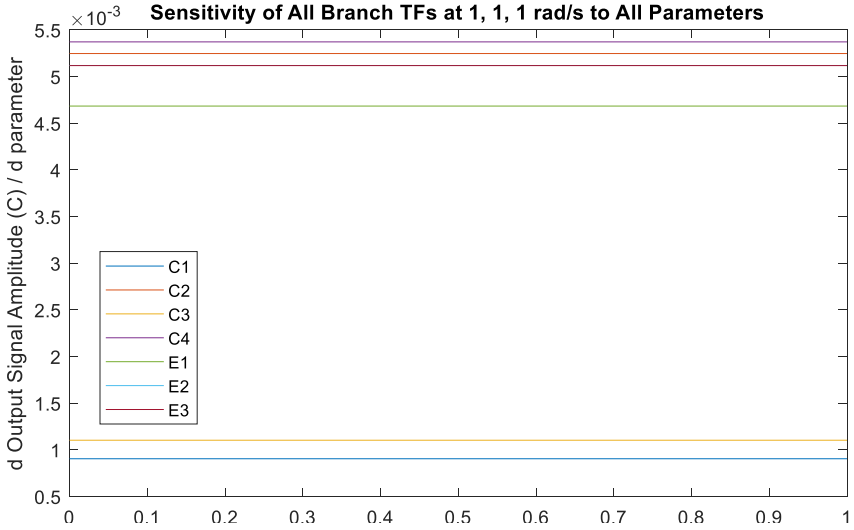

Figure 4.20: Sensitivity of Output Amplitude to All Parameters.

This ranking can be numerically verified by Fig. 4.21.

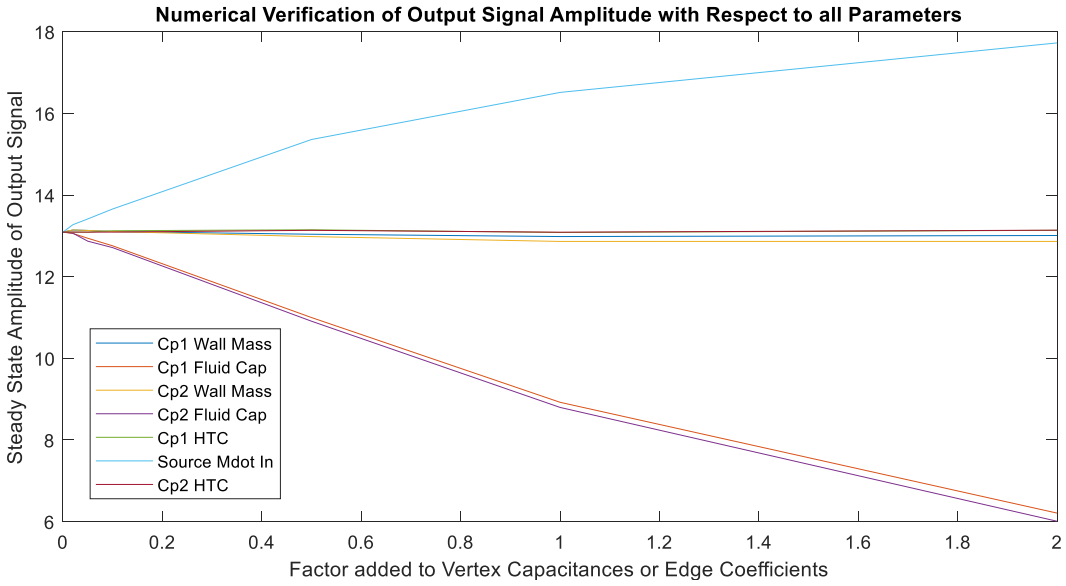

Figure 4.21: Numerical Verification of System with 1 rad/s Inputs

Notice how the source mass flow rate parameter has a much larger effect on the system than it is expected to. This is because the sensitivity analysis is only accurate around the nominal mass flow rates, and perturbing the mass flow rate, especially by as much as 2 kg/s, changes the linearized model and thus the sensitivity analysis based on that model considerably. Since the vertex capacitances do not affect the mass flow rate, they are expected to produce more accurate results with the sensitivity analysis than the edge coefficients. Thus, the vertex capacitances will be the focus of the sensitivity analyses for this thesis.

The estimated ranking of the output amplitude sensitivities to only the capacitance parameters is shown in Fig. 4.22 below, followed by the numerical verification of the capacitance parameters in Fig. 4.23.

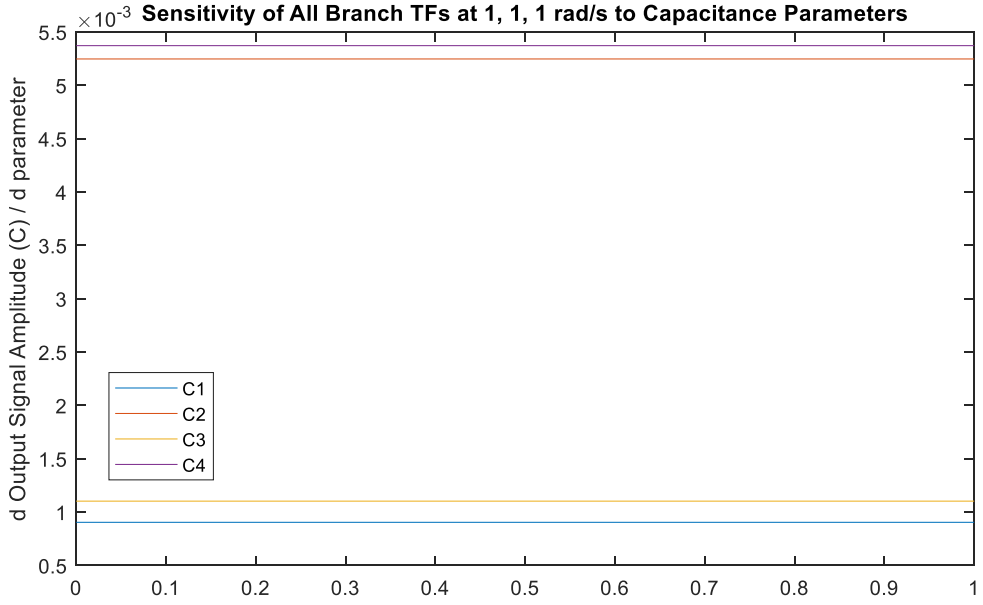

Figure 4.22: Sensitivity of Output Amplitude to Capacitance Parameters.

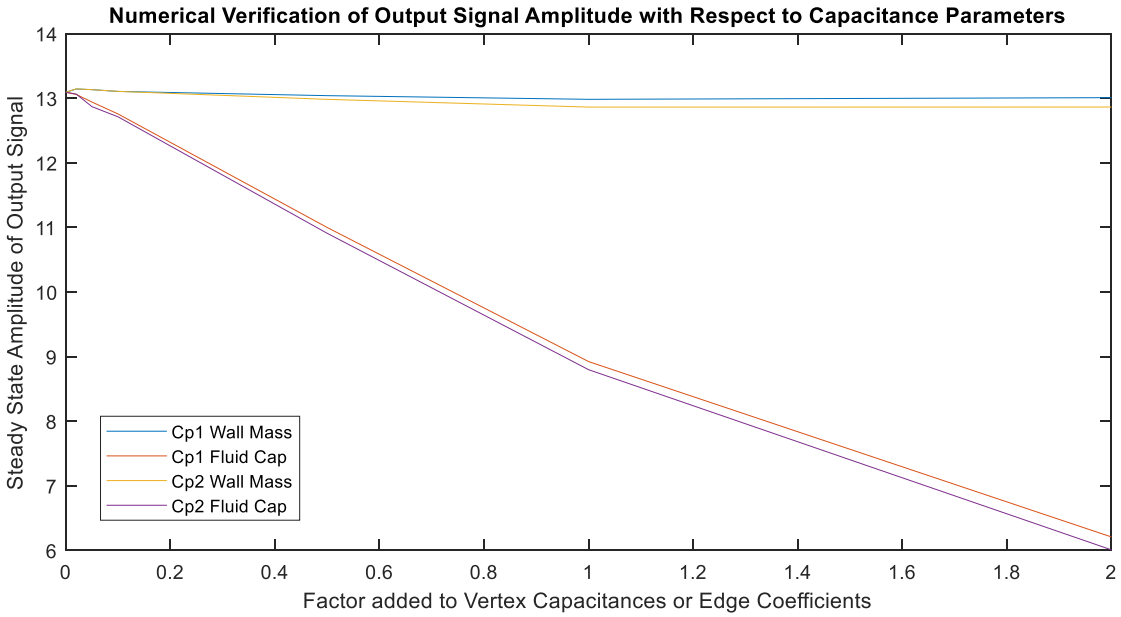

Figure 4.23: Numerical Verification of System to Capacitance Parameters.

The parameter expected to have the greatest effect on the system, C_4, does in fact lower the steady state amplitude of the output signal the most. This is closely followed by C_2, and then C_3 and C_1. Fig. 4.22 and 4.23 show agreement between them as to which parameters are more influential on the output signal amplitude. This concludes the walk-through of Example 1.

4.3 Conducting a Sensitivity Analysis of Example System 2

This next section applies the sensitivity analysis techniques discussed in Section 4.2 to another system, which will be referred to as Example 2 moving forward. Example 2 is the subject of the plant and controller co-design methods, which will be further explored in Chapter 6. The remainder of this chapter will set up the plant sensitivity analysis for Example 2 and the results of the Sensitivity Analysis will be discussed in Chapter 5.

4.3.1 Defining Example 2

Example 2 has been chosen to be a fluid loop with a single source input and a single sink output. Immediately after the source, the fluid flow passes through Cold Plate1. Then it reaches a split, where the flow branches off into two parallel flows through the system. The top branch has two cold plates in series and the bottom branch has a single cold plate component within it. This

flow split is controlled by a valve component, which designates how much fluid flows into the top branch by a number between 0 and 1. This schematic of Example 2 is shown in Fig. 4.24.

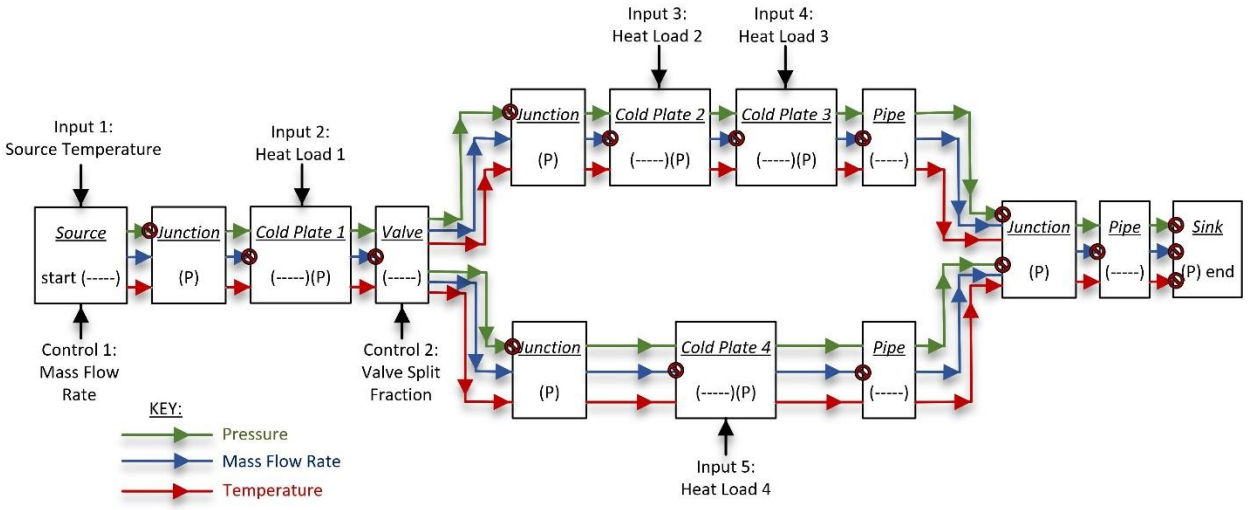

Figure 4.24: Schematic of Example 2.

The red circle with a line through it represents terminator blocks which ignore the information coming from the previous component through the bus. Although not pictured in the schematic, each component is also passing back the first bus signal property it calculates by a sink block to the previous component. For example, the cold plate components pass the mass flow rate they calculate to their previous component, and the junction components pass back the pressure they calculate to all the components that flow into them. This system is set up with five inputs which can be altered by the user: the temperature of the source flow and the heat loads on Cold Plates 1 – 4. These inputs values can range from constant values of 0 rad/s to sinusoidal signals of any frequency. There are also two factors which can be controlled to adjust the temperature states in this system: the mass flow rate of fluid entering the system and the percentage split of fluid leading to the top branch, as opposed to the bottom branch. For now, the model is given a constant mass flow rate of 1.67348 kg/s and valve split of 0.5. These values will hold until controllers are introduced to the system in Chapter 6. A graph model representation of Example 2 is shown in Fig. 4.25.

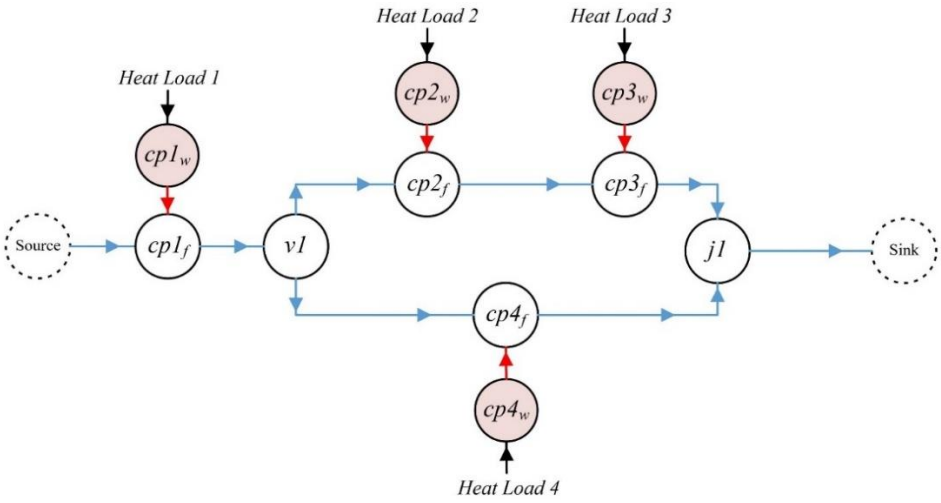

Figure 4.25: Simplified Graph Model of Example 2.

The graph model shown in Fig. 4.25 is simplified from Fig 4.24, because there is no need to represent the conversion blocks, such as pipes or junctions which only have one input, since they have a negligible effect on the system dynamics. They are only included in the Simulink model to convert between calculating mass flow rate and pressure. The valve component also does not have a large effect on the system temperatures, other than by directing flow to the two branches. The temperature of the flow entering the valve is the same as the temperatures which will pass through to both the top branch and the bottom branch. However, the valve is still included in the graph model for clarity of how the flow rates are adjusted. Each of the vertices and each of the edges are numbered in the graph-based model, shown in Fig. 4.26.

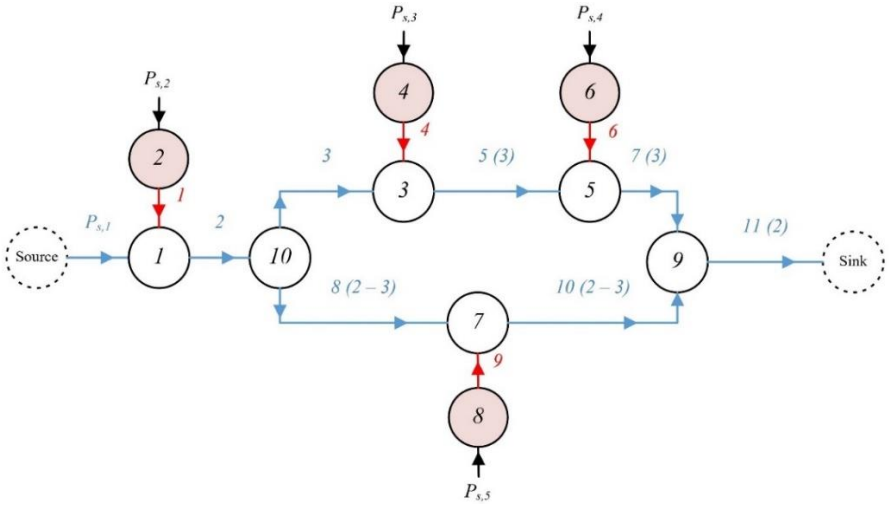

Figure 4.26: Numbered Graph Model of Example 2.

All the components in Example 2 are flooded, meaning the mass flow into a component is the same as the mass flow out of the component. Therefore, there are two independent mass flow rates which describe the mass flow through the entire system: the flow rate into the system, $E2$, and the flow rate to the top branch, $E3$. Thus, all the advective edges can be described as either $E2$, $E3$, or $E2 - E3$.

Now that Example 2 has been defined and modeled, Section 4.3.2 will expand on how it can be represented as a set of transfer functions, one for each input.

4.3.2 Representing Example 2 as a set of Transfer Functions

The graph model of Example 2 consists of three component types: cold plates, a valve, and a junction. These are the components for which the output-to-input transfer functions must be derived. Since the temperature of the fluid leaving the system along edge 11 has been chosen as the output value, the individual exit temperatures for each of the components should be the outputs for their transfer functions. One of the inputs needs to be the temperature of the previous component so that the component transfer functions can be linked together as shown in Fig. 4.3.

For the valve component, the two transfer functions would be $\frac{output\ 1\ temperature}{input\ temperature}$ and $\frac{output\ 2\ temperature}{input\ temperature}$. Both are equal to 1 since the valve component does not affect the temperature of the fluid flow, meaning that the valve component transfer function can be omitted from the total system transfer function equation. The junction component has two inputs and one output, resulting in the following two transfer functions: $\frac{output\ temperature}{input\ 1\ temperature}$ and $\frac{output\ temperature}{input\ 2\ temperature}$. Each of the cold plate components has two outputs and two inputs, resulting in the following four transfer functions: $\frac{output\ temperature}{input\ temperature}$, $\frac{output\ temperature}{heat\ load}$, $\frac{wall\ temperature}{input\ temperature}$, and $\frac{wall\ temperature}{heat\ load}$. However, the wall temperature transfer functions are not included in the total system transfer function equation and can be omitted from the analysis. Since the cold plate components have two transfer functions which represent the output temperature from two different inputs, they can be added together by the principle of superposition. However, the temperatures cannot be added directly and should be converted to energy first, as shown in Fig. 4.3. Example 2 can be represented as the combination of each of the component transfer functions, as shown by the block diagram in Fig. 4.27.

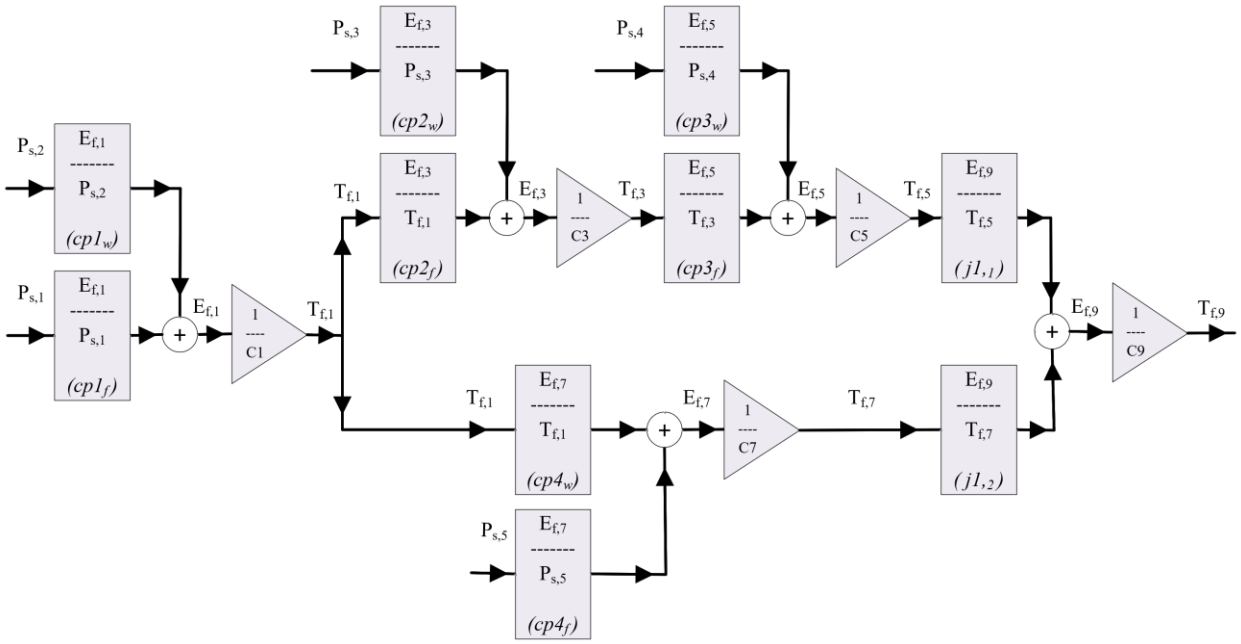

Figure 4.27: Block Diagram of Example 2.

The cold plate component transfer functions were derived in Section 4.2.1. Following the general state space form in Eq. (4.44) and the numbering scheme defined in Fig. 4.26, each of the substitutions for Cold Plate 1 is shown in Eq. (4.45 – 4.49).

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}a_1 & a_2\\ a_3 & a_4\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}b_1 & 0\\ 0 & b_2\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix} \tag{4.44}$$

$$\dot{x}_1 = \dot{E}_{f,1};\ x_1 = E_{f,1} \tag{4.45}$$

$$\dot{x}_2 = \dot{E}_{w,2};\ x_2 = E_{w,2} \tag{4.46}$$

$$u_1 = T_{in,1}\,(P_{s,1})\ ;\ u_2 = Q_{amb,1}\,(P_{s,2}) \tag{4.47}$$

$$a_1 = -\frac{E_2}{C_1} - \frac{E_1}{C_1};\ a_2 = \frac{E_1}{C_2};\ a_3 = \frac{E_1}{C_1};\ a_4 = -\frac{E_1}{C_2} \tag{4.48}$$

$$b_1 = E_2;\ b_2 = 1000 \tag{4.49}$$

For Cold Plate 2, the substitutions are shown in Eq. (4.50) – (4.54).

$$\dot{x}_1 = \dot{E}_{f,3};\ x_1 = E_{f,3} \tag{4.50}$$

$$\dot{x}_2 = \dot{E}_{w,4};\ x_2 = E_{w,4} \tag{4.51}$$

$$u_1 = T_{in,3}\,(T_{f,1})\ ;\ u_2 = Q_{amb,2}\,(P_{s,3}) \tag{4.52}$$

$$a_1 = -\frac{E_3}{C_3} - \frac{E_4}{C_3};\ a_2 = \frac{E_4}{C_4};\ a_3 = \frac{E_4}{C_3};\ a_4 = -\frac{E_4}{C_4} \tag{4.53}$$

$$b_1 = E_3;\ b_2 = 1000 \tag{4.54}$$

For Cold Plate 3, the substitutions are shown in Eq. (4.55) – (4.59).

$$\dot{x}_1 = \dot{E}_{f,5};\ x_1 = E_{f,5} \tag{4.55}$$

$$\dot{x}_2 = \dot{E}_{w,6};\ x_2 = E_{w,6} \tag{4.56}$$

$$u_1 = T_{in,5}\ (T_{f,3})\ ;\ u_2 = Q_{amb,3}\ (P_{s,4}) \tag{4.57}$$

$$a_1 = -\frac{E_3}{C_5} - \frac{E_6}{C_5};\ a_2 = \frac{E_6}{C_6};\ a_3 = \frac{E_6}{C_5};\ a_4 = -\frac{E_6}{C_6} \tag{4.58}$$

$$b_1 = E_3;\ b_2 = 1000 \tag{4.59}$$

For Cold Plate 4, the substitutions are shown in Eq. (4.60) – (4.64).

$$\dot{x}_1 = \dot{E}_{f,7};\ x_1 = E_{f,7} \tag{4.60}$$

$$\dot{x}_2 = \dot{E}_{w,8};\ x_2 = E_{w,8} \tag{4.61}$$

$$u_1 = T_{in,7}\ (T_{f,1})\ ;\ u_2 = Q_{amb,4}\ (P_{s,5}) \tag{4.62}$$

$$a_1 = -\frac{(E_2 - E_3)}{C_7} - \frac{E_9}{C_7};\ a_2 = \frac{E_9}{C_8};\ a_3 = \frac{E_9}{C_7};\ a_4 = -\frac{E_9}{C_8} \tag{4.63}$$

$$b_1 = (E_2 - E_3);\ b_2 = 1000 \tag{4.64}$$

Each of these substitutions can replace the terms in the frequency domain transfer function equations in Eq. (4.65) and (4.66).

$$\frac{X_1(s)}{U_1(s)} = \frac{b_1(s - a_4)}{s(s - a_4) - a_1(s - a_4) - a_2 a_3} = \frac{b_1 s - b_1 a_4}{s^2 - s(a_1 + a_4) + a_1 a_4 - a_2 a_3} \tag{4.65}$$

$$\frac{X_1(s)}{U_2(s)} = \frac{a_2 b_2}{s(s - a_4) - a_1(s - a_4) - a_2 a_3} = \frac{a_2 b_2}{s^2 - s(a_1 + a_4) + a_1 a_4 - a_2 a_3} \tag{4.66}$$

These result in the transfer function equations for the four cold plates, shown in Eq. (4.67) – (4.74).

$$\frac{E_{f,1}}{P_{s,1}} = \frac{E_2 s + \frac{E_1 E_2}{C_2}}{s^2 + s\left(\frac{E_2}{C_1} + \frac{E_1}{C_1} + \frac{E_1}{C_2}\right) + \frac{E_1 E_2}{C_1 C_2}} \tag{4.67}$$

$$\frac{E_{f,1}}{P_{s,2}} = \frac{\frac{E_1}{C_2} * 1000}{s^2 + s\left(\frac{E_2}{C_1} + \frac{E_1}{C_1} + \frac{E_1}{C_2}\right) + \frac{E_1 E_2}{C_1 C_2}} \tag{4.68}$$

$$\frac{E_{f,3}}{T_{f,1}} = \frac{E_3 s + \frac{E_3 E_4}{C_4}}{s^2 + s\left(\frac{E_3}{C_3} + \frac{E_4}{C_3} + \frac{E_4}{C_4}\right) + \frac{E_3 E_4}{C_3 C_4}} \tag{4.69}$$

$$\frac{E_{f,3}}{P_{s,3}} = \frac{\frac{E_4}{C_4} * 1000}{s^2 + s\left(\frac{E_3}{C_3} + \frac{E_4}{C_3} + \frac{E_4}{C_4}\right) + \frac{E_3 E_4}{C_3 C_4}} \tag{4.70}$$

$$\frac{E_{f,5}}{T_{f,3}} = \frac{E_3 s + \frac{E_3 E_6}{C_6}}{s^2 + s\left(\frac{E_3}{C_5} + \frac{E_6}{C_5} + \frac{E_6}{C_6}\right) + \frac{E_3 E_6}{C_5 C_6}} \tag{4.71}$$

$$\frac{E_{f,5}}{P_{s,4}} = \frac{\frac{E_6}{C_6} * 1000}{s^2 + s\left(\frac{E_3}{C_5} + \frac{E_6}{C_5} + \frac{E_6}{C_6}\right) + \frac{E_3 E_6}{C_5 C_6}} \tag{4.72}$$

$$\frac{E_{f,7}}{T_{f,1}} = \frac{(E_2 - E_3)s + \frac{(E_2 - E_3)E_9}{C_8}}{s^2 + s\left(\frac{(E_2 - E_3)}{C_7} + \frac{E_9}{C_7} + \frac{E_9}{C_8}\right) + \frac{(E_2 - E_3)E_9}{C_7 C_8}} \tag{4.73}$$

$$\frac{E_{f,7}}{P_{s,5}} = \frac{\frac{E_9}{C_8} * 1000}{s^2 + s\left(\frac{(E_2 - E_3)}{C_7} + \frac{E_9}{C_7} + \frac{E_9}{C_8}\right) + \frac{(E_2 - E_3)E_9}{C_7 C_8}} \tag{4.74}$$

The system equations of the junction will be necessary to calculate the transfer function equations, $\frac{output\ temperature}{input\ 1\ temperature} = \frac{E_{f,9}}{T_{f,5}}$ and $\frac{output\ temperature}{input\ 2\ temperature} = \frac{E_{f,9}}{T_{f,7}}$, from Fig. 4.27. The junction component was detailed in Section 2.2.4 where the differential equation in Eq. (4.75) was introduced to define the junction dynamics.

$$\dot{T}_f = \frac{c_{P_f} \cdot \left(\sum_{n=1}^{number\ inputs} \dot{m}_{in,n} \cdot T_{in,n} - \sum_{m=1}^{number\ outputs} \dot{m}_{out,m} \cdot T_f\right)}{M_f \cdot c_{P_f}} \tag{4.75}$$

With two inputs and one output, Eq. (4.75) can be rewritten as Eq. (4.76) and (4.77).

$$\dot{T}_f = \frac{c_{P_f} \cdot \dot{m}_{in,1} \cdot T_{in,1} + c_{P_f} \cdot \dot{m}_{in,2} \cdot T_{in,2} - c_{P_f} \cdot \dot{m}_{out} \cdot T_f}{M_f \cdot c_{P_f}} \tag{4.76}$$

$$\dot{E}_f = c_{P_f} \cdot \dot{m}_{in,1} \cdot T_{in,1} + c_{P_f} \cdot \dot{m}_{in,2} \cdot T_{in,2} - c_{P_f} \cdot \dot{m}_{out} \cdot \frac{T_f}{M_f \cdot c_{P_f}} \tag{4.77}$$

Eq. (4.77) can be written in the general state space form for a single state and two inputs as shown in (4.78). Then Eq. (4.79) – (4.82) shows the substitutions of each of the general terms in Eq. (4.78) with the variables specific to the junction component in Example 2.

$$[\dot{x}] = [a][x] + [b_1 \quad b_2]\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{4.78}$$

$$\dot{x} = \dot{E}_{f,9};\ x = E_{f,9} \tag{4.79}$$

$$u_1 = T_{f,5};\ u_3 = T_{f,7} \tag{4.80}$$

$$a = -\frac{E_2}{C_9} \tag{4.81}$$

$$b_1 = E_2;\ b_2 = (E_2 - E_3) \tag{4.82}$$

Once the Laplace transform is applied to Eq. (4.78) it results in Eq. (4.83), which can be arranged to the junction component transfer functions in Eq. (4.84) – (4.85).

$$X_1(s) * s = a * X_1(s) + b_1 * U_1(s) + b_2 * U_2(s) \tag{4.83}$$

$$\frac{X_1(s)}{U_1(s)} = \frac{E_{f,9}}{T_{f,5}} = \frac{E_2}{s + \frac{E_2}{C_9}} \tag{4.84}$$

$$\frac{X_1(s)}{U_2(s)} = \frac{E_{f,9}}{T_{f,7}} = \frac{(E_2 - E_3)}{s + \frac{E_2}{C_9}} \tag{4.85}$$

Finally, all the component transfer functions can be combined to form the entire system transfer functions which relate the output fluid temperature to each of the system inputs. The block diagram shown in Fig. 4.27 can be simplified to Eq. (4.86) – (4.91).

$$P_{s,1} * \frac{T_{f,9}}{P_{s,1}} + P_{s,2} * \frac{T_{f,9}}{P_{s,2}} + P_{s,3} * \frac{T_{f,9}}{P_{s,3}} + P_{s,4} * \frac{T_{f,9}}{P_{s,4}} + P_{s,5} * \frac{T_{f,9}}{P_{s,5}} = T_{f,9} \tag{4.86}$$

where

$$\frac{T_{f,9}}{P_{s,1}} = \frac{E_{f,1}}{P_{s,1}} * \frac{1}{C_1} * \left(\frac{E_{f,3}}{T_{f,1}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} + \frac{E_{f,7}}{T_{f,1}} * \frac{1}{C_7} * \frac{E_{f,9}}{T_{f,7}}\right) * \frac{1}{C_9} \tag{4.87}$$

$$\frac{T_{f,9}}{P_{s,2}} = \frac{E_{f,1}}{P_{s,2}} * \frac{1}{C_1} * \left(\frac{E_{f,3}}{T_{f,1}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} + \frac{E_{f,7}}{T_{f,1}} * \frac{1}{C_7} * \frac{E_{f,9}}{T_{f,7}}\right) * \frac{1}{C_9} \tag{4.88}$$

$$\frac{T_{f,9}}{P_{s,3}} = \frac{E_{f,3}}{P_{s,3}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} * \frac{1}{C_9} \tag{4.89}$$

$$\frac{T_{f,9}}{P_{s,4}} = \frac{E_{f,5}}{P_{s,4}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} * \frac{1}{C_9} \tag{4.90}$$

$$\frac{T_{f,9}}{P_{s,5}} = \frac{E_{f,7}}{P_{s,5}} * \frac{1}{C_7} * \frac{E_{f,9}}{T_{f,7}} * \frac{1}{C_9} \tag{4.91}$$

Eq. (4.87) – (4.91) represent the five input-specific transfer functions which relate the system output to all the input signals. The component-level transfer functions within Eq. (4.87 – 4.91) can be replaced with Eq. (4.67) – (4.74) and (4.84) – (4.85) to put the system-level transfer functions in terms of all the vertex capacitances and edge coefficients. Then the partial derivative of the system, expressed as a set of transfer functions, with respect to each of the capacitances and edge coefficients can be solved for. This sensitivity analysis will disclose the vertex capacitances or edge coefficients which have the greatest effect on the output temperature, $T_{f,9}$.

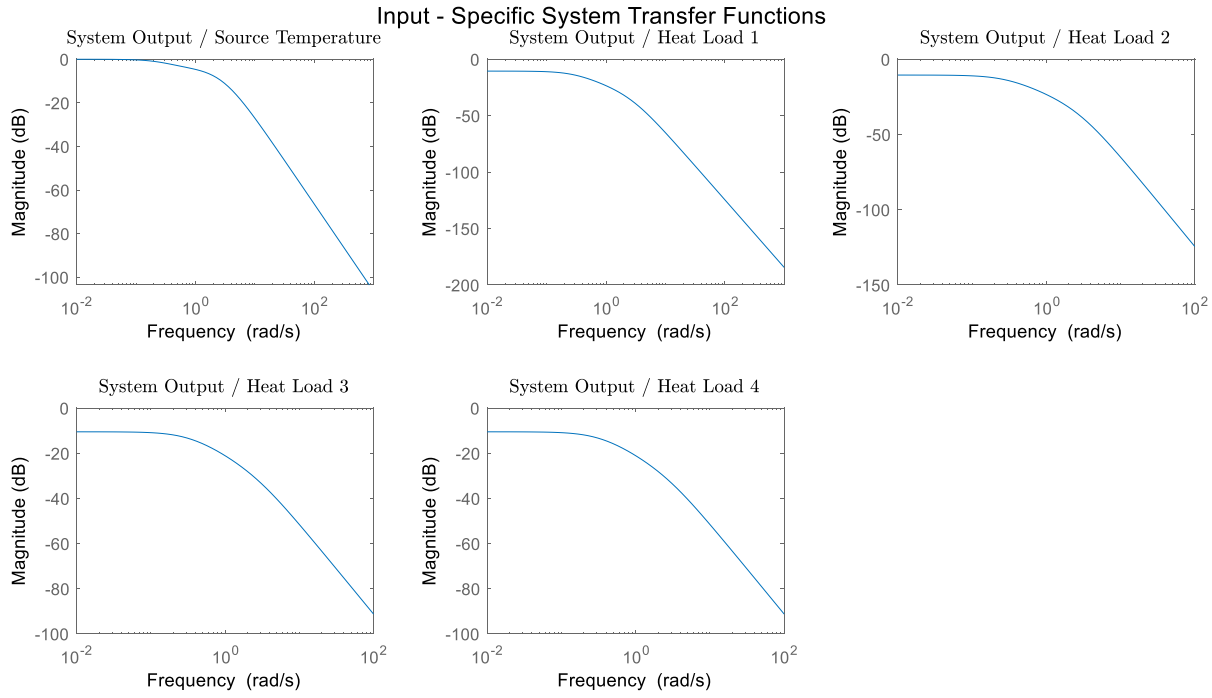

Figure 4.28: Bode Plots of the System Transfer Function Equations.

4.3.3 Finding the Sensitivities of each of the System Transfer Functions

The final step of the sensitivity analysis is to take the partial derivatives of each of the system transfer functions with respect to each of the system parameters: vertex capacitances and edge coefficients. MATLAB's symbolic toolbox can be utilized to perform these calculations. The parameter sensitivities must be calculated at specified input signal amplitudes and frequencies and then added together to generate the total effect each parameter has on the output signal for a

specific set of input signals. This is equivalent to taking the partial derivative of Eq. (4.86) with respect to system parameters, θ, shown in Eq. (4.92).

$$P_{s,1} * \frac{\partial \frac{T_{f,9}}{P_{s,1}}}{\partial \theta} + P_{s,2} * \frac{\partial \frac{T_{f,9}}{P_{s,2}}}{\partial \theta} + P_{s,3} * \frac{\partial \frac{T_{f,9}}{P_{s,3}}}{\partial \theta} + P_{s,4} * \frac{\partial \frac{T_{f,9}}{P_{s,4}}}{\partial \theta} + P_{s,5} * \frac{\partial \frac{T_{f,9}}{P_{s,5}}}{\partial \theta} = \frac{\partial T_{f,9}}{\partial \theta} \tag{4.92}$$

The method of taking the partial derivative of the system transfer function equations is outlined in detail for Example 1 in Section 4.2.2. These same steps can be applied to Example 2 in a similar manner.

This concludes the setup and discussion of the transfer function sensitivity analysis method. The results and discussion for the sensitivity analysis of Example 2 are found in Chapter 5.

Chapter 5

Sensitivity Analysis Results and Discussion

5.1 Summary

This chapter focuses on the sensitivity analysis and its simulation-based numerical verification of Example System 2, shown in Fig. 5.1.

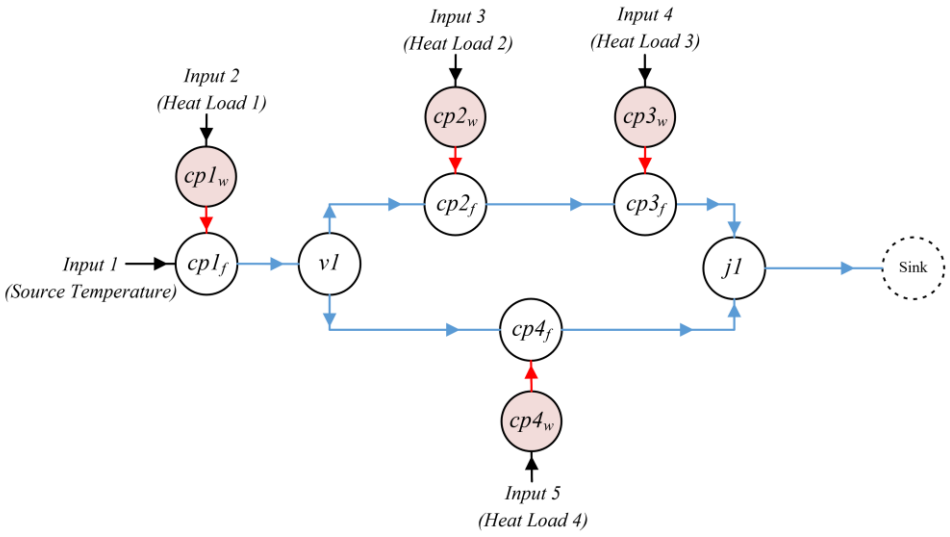

Figure 5.1: Graph Model of Example 2.

The sensitivity of the output temperature with respect to each of the parameters will be determined for four different scenarios. For the first scenario, all the input signals will have constant values. This means the input signals for this scenario will be at 0 rad/s. The second scenario involves the first input, the source temperature, having a non-zero frequency of 1 rad/s. Then for the third scenario, only the second and third inputs, Heat Loads 1 and 2, will have non-zero frequencies of 1 rad/s and 0.5 rad/s, respectively. Finally, the fourth scenario involves all the inputs signals having non-zero frequencies of 1 rad/s, 1 rad/s, 0.5 rad/s, 0.1 rad/s, and 1.5 rad/s for Inputs 1-5 respectively. The sensitivity bode plots will be referenced for each of the input signals

at their respective frequencies and the total contributions each parameter has on the combined transfer functions will be calculated. Finally, each of the parameters will be perturbed individually by several increasing values and the effects had on the output signal will be recorded and compared across the parameters. This will show which parameter has the greatest influence on the output signal by how far the output signal amplitude deviates from the nominal value when that parameter is perturbed.

Before the partial derivatives of the input-specific transfer functions with respect to the system parameters can be calculated, those system transfer functions must be derived. This step was shown in detail in Section 4.3 and the resulting transfer function bode plots are included again in Fig. 5.2.

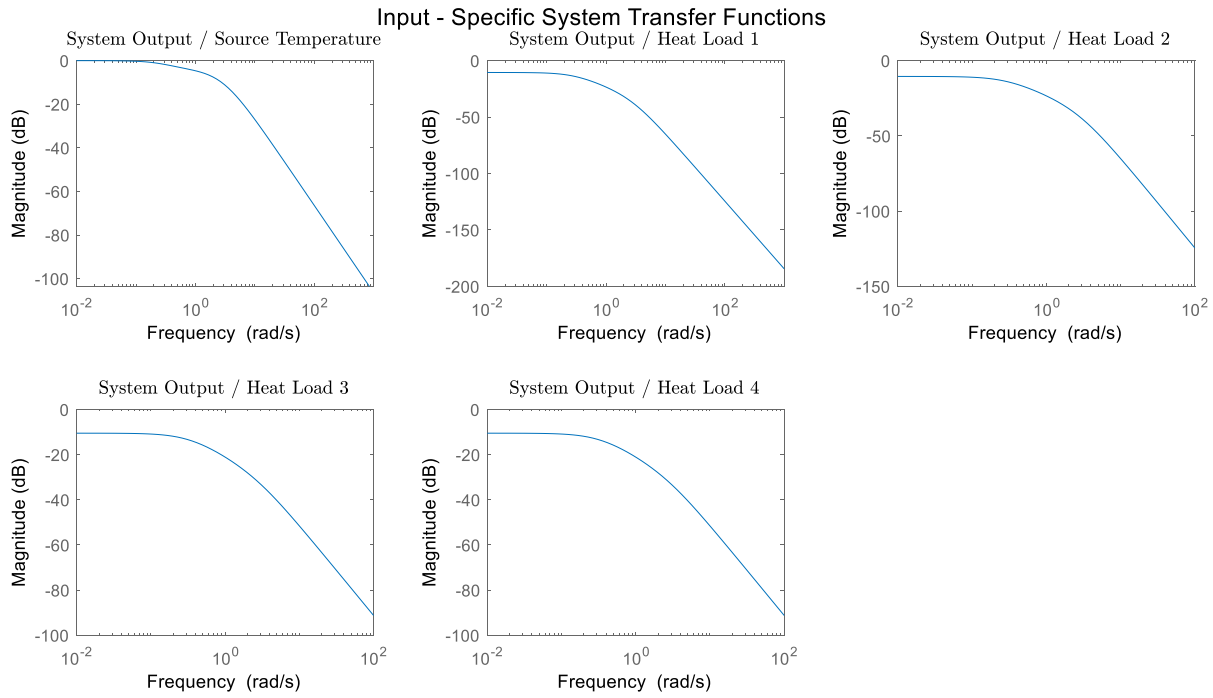

Figure 5.2: Input-Specific System Transfer Functions for Example 2.

MATLAB's symbolic toolbox can then be utilized in taking the partial derivatives of each of the input transfer function equations to the system parameters. The resulting partial derivatives are shown in Fig. 5.3.

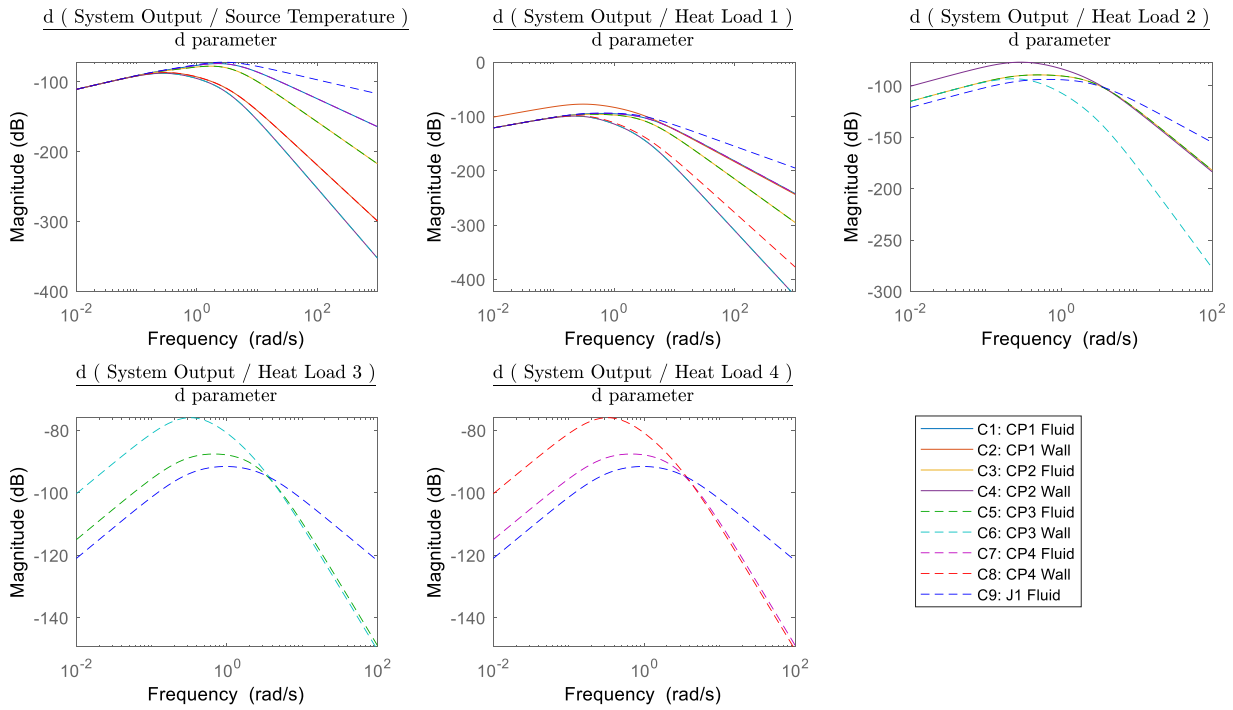

Figure 5.3: Sensitivity Bode Plots for Example 2.

Each of these five plots shows how the sensitivity of the transfer function gains to the system parameters change with the input frequencies. The next sections explain how the sensitivity bode plots can be used to figure out the sensitivity of the system output for any combination of input signal frequencies.

5.2 Constant Inputs

The first scenario to be discussed is when all the input signals, namely the source temperature and Heat Loads 1-4, are at constant values. They are not sinusoidal and do not change with time. Such constant values are at 0 rad/s or have a frequency of zero. The system transfer functions, shown in Fig. 5.4, can be used to determine how each of the input signals affect the resulting output signal amplitude.

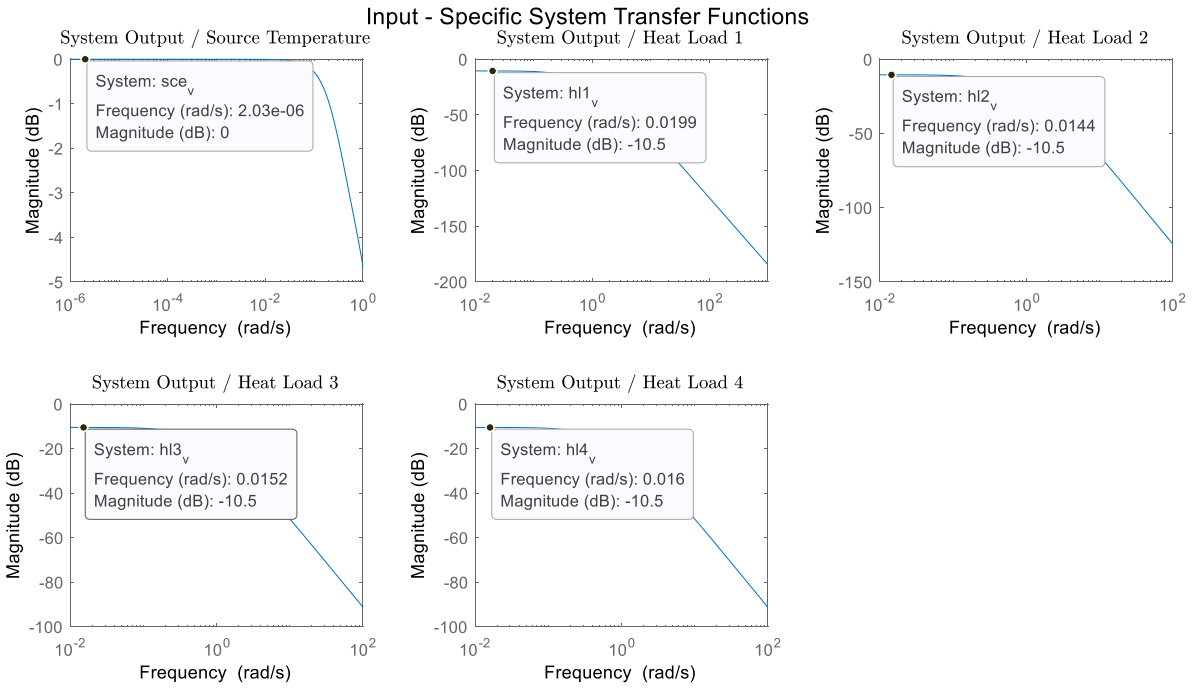

Figure 5.4: Transfer Function Gains for Input Signals at 0 rad/s.

The gain at zero frequency, or the DC gain, is found by reading the magnitude of the bode plot at the lowest frequency shown. For Input Signal 1, the DC gain is 0 dB. The conversion between dB and the transfer function gain is shown in Eq. (5.1).

$$10^{value\ in\ dB/20} = gain\ ratio \tag{5.1}$$

Thus, the gain of the output with respect to the input when Input 1 is the only input, is 1- or unity. Physically, this means that the output temperature equals the input temperature when there are no heat loads. Notice that the DC gain for Heat Load 1-4 are all -10.5 dB, or a gain of 0.2985. These are the heat loads which are applied to Cold Plate 1 before the valve, Cold Plate 2 in the top branch, Cold Plate 3 also in the top branch, and Cold Plate 4 in the top branch. Fig. 5.1 can be referenced as a reminder of the system layout. Having equal DC gains implies that all four inputs would have an equal effect on the output temperature if the input signal amplitudes were also equal. However, from row 3 of Table 5.1, it is clear that Heat Load 4 has the highest amplitude, followed by Heat Load 3, then Heat Loads 1 and 2. This means that Heat Load 4 has a greater impact in this scenario than Heat Loads 1, 2, or 3, as seen in row 4 of Table 5.1.

	Source	Heat Load 1	Heat Load 2	Heat Load 3	Heat Load 4	Total
0 freq value (dB)	0	-10.5	-10.5	-10.5	-10.5	
0 freq (gain)	1	0.2985	0.2985	0.2985	0.2985	
Amplitude	20	1	1	2	5	
Amplitude*gain	20	0.2985	0.2985	0.5971	1.4927	22.69

Table 5.1: Breakdown of Contributions to Output Amplitude from Input Signals.

The final column in Table 5.1 shows the sum of each of the contributions to the output temperature from the five input signals. Each of the input signal amplitudes and frequencies factor into this value. These add up to a total value of 22.69 °C. The numerical verification for the estimated output amplitude for this scenario is shown in Fig. 5.5 and verifies the final output temperature of 22.69 °C. This indicates that using the principle of superposition to combine the system transfer functions of each input signal is valid. Fig. 5.6 is included below as a visualization of the calulations shown in Table 5.1.

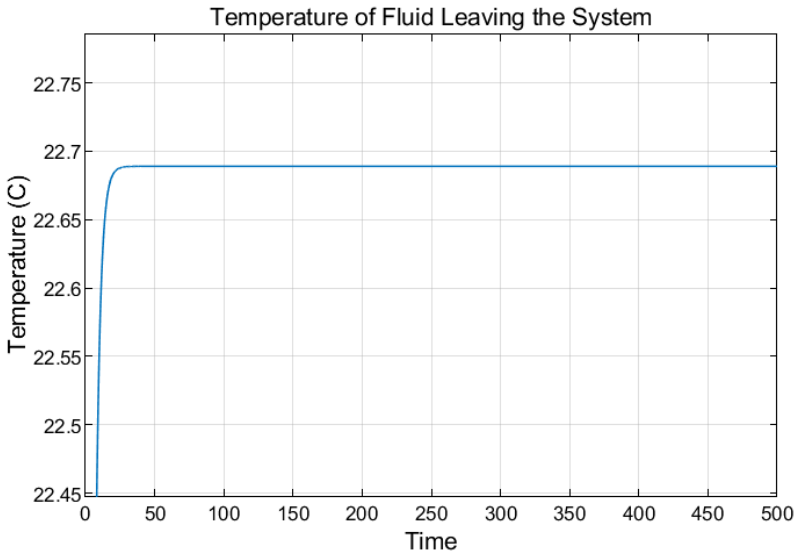

Figure 5.5: Numerical Verification of Output Amplitude with Constant Inputs.

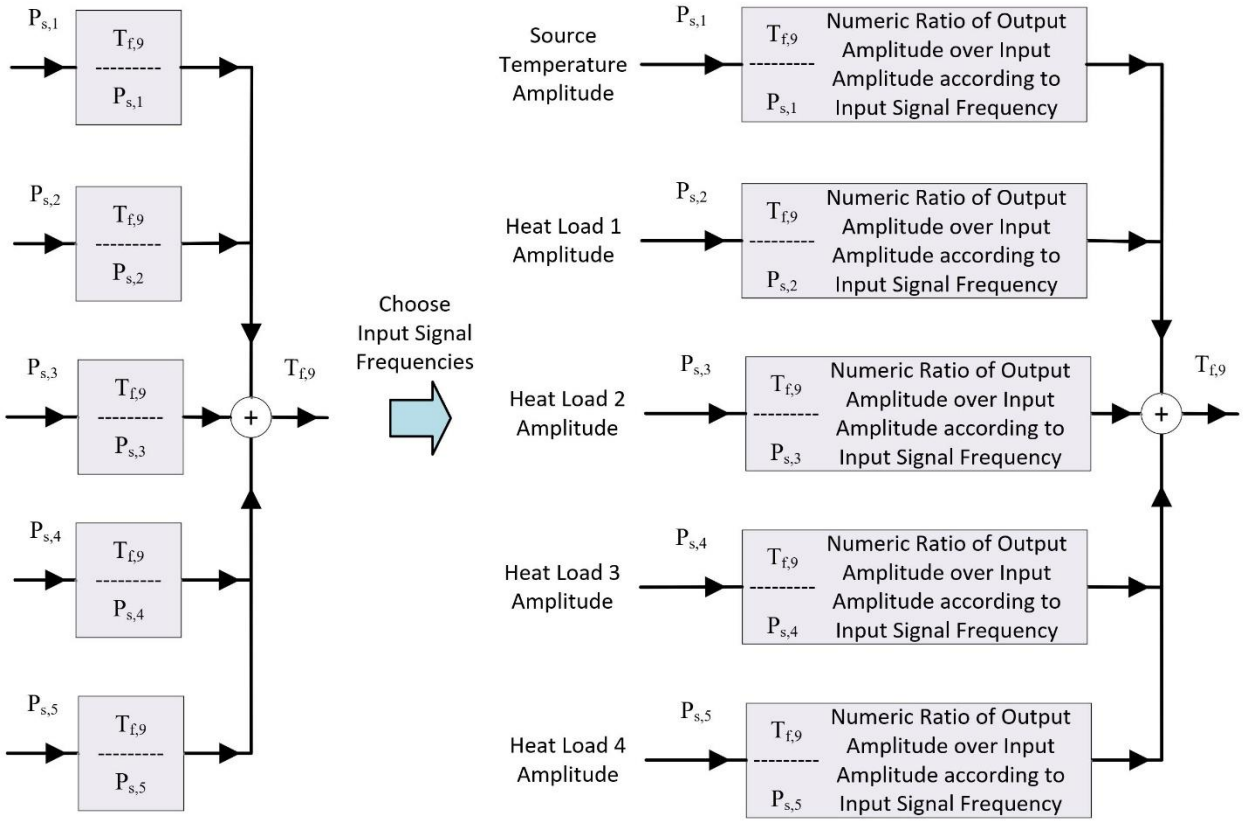

Figure 5.6: Applying the Superposition Principle at Specific Input Frequencies.

Next, the partial derivatives of each of the system transfer functions, shown in Fig. 5.4, to the capacitance parameters must be taken. These are called the "sensitivity bode plots" and are shown in Fig. 5.3. Since the frequencies of all the input signals are at 0 rad/s, only the far-left side of the bode plots, where the frequencies are low, is relevant. Notice how as the frequencies get lower, the sensitivities of the transfer functions to each of the parameters also gets lower. This implies that when all the input signals are at 0 rad/s, none of the capacitance values have any effect on the system transfer function gains. Regardless of how large any of the capacitances are, the ratio of output temperature to each of the inputs will always be the same at steady state and result in a temperature around 22.7 °C. In this case, when all the inputs are constant signals, it is not hard to tell that the sensitivity of the output with respect to the parameters will be zero, but it will not always be this easy to figure out. This supports the need for the procedure visualized in Fig. 5.7 to combine the sensitivity bode plots in a more readable format, shown in Fig. 5.8.

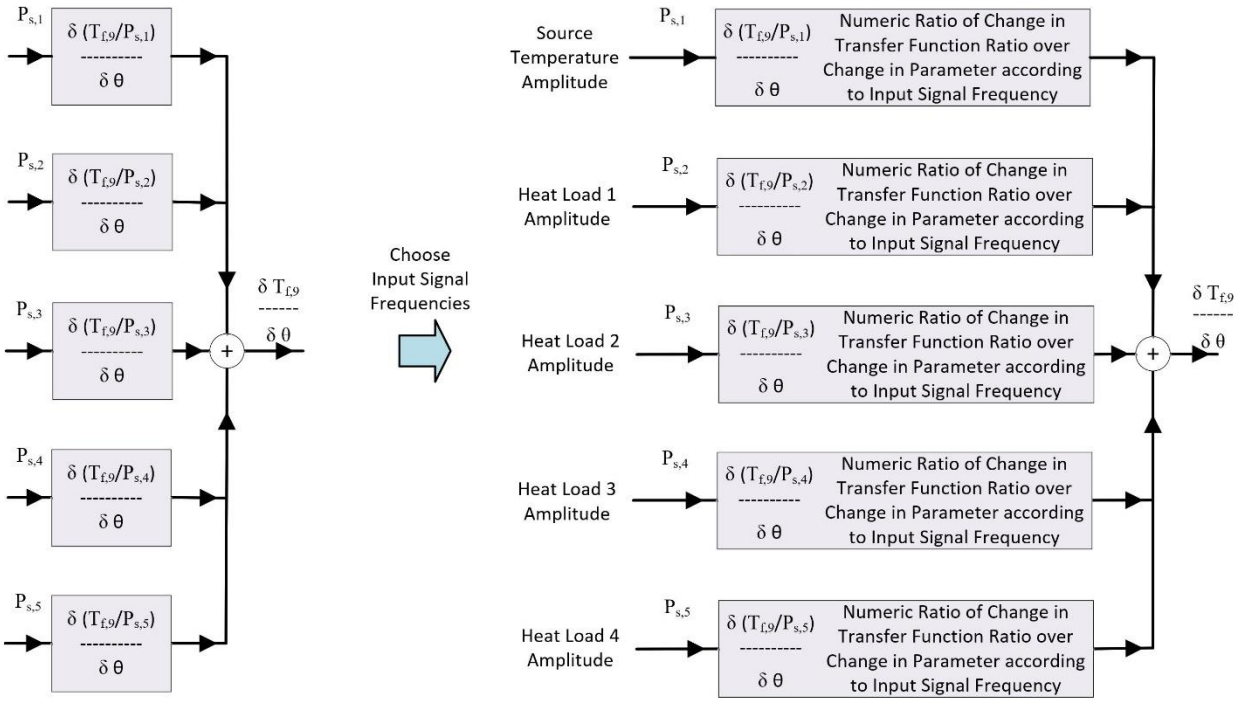

Figure 5.7: Applying Superposition Principle to Transfer Function Sensitivity Function.

Note that the numeric ratios in Fig. 5.7 are all zero. Therefore, the actual amplitudes of each of the input signals are irrelevant and the sum $\frac{\partial T_{f,9}}{\partial \theta}$ equals zero. Refer to Eq. 5.2 for a written representation of Fig. 5.7.

$$P_{s,1} * \frac{\partial \frac{T_{f,9}}{P_{s,1}}}{\partial \theta} + P_{s,2} * \frac{\partial \frac{T_{f,9}}{P_{s,2}}}{\partial \theta} + P_{s,3} * \frac{\partial \frac{T_{f,9}}{P_{s,3}}}{\partial \theta} + P_{s,4} * \frac{\partial \frac{T_{f,9}}{P_{s,4}}}{\partial \theta} + P_{s,5} * \frac{\partial \frac{T_{f,9}}{P_{s,5}}}{\partial \theta} = \frac{\partial T_{f,9}}{\partial \theta} \qquad (5.2)$$

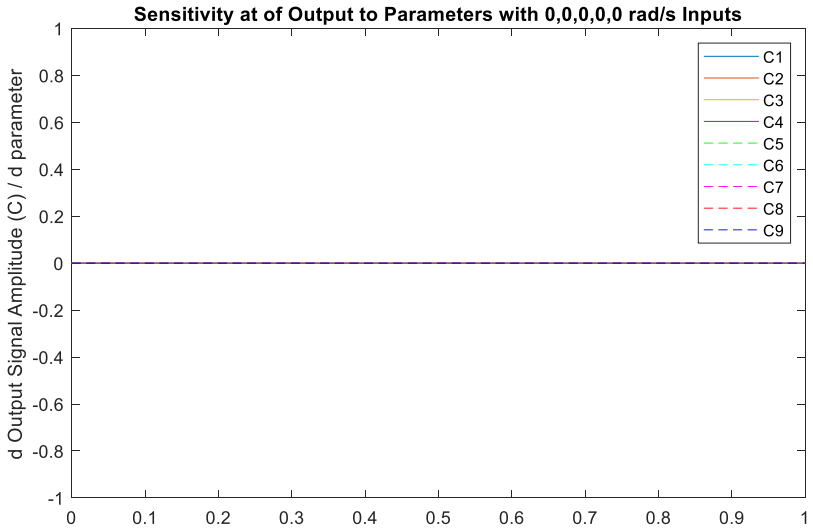

Figure 5.8: Sensitivity of the Output to the Capacitance Parameters under Constant Inputs.

Fig. 5.8 shows the results of Eq. (5.2) for each of the parameters. Since Eq. (5.2) yields a single number, this information could have been communicated as a bar chart with one bar for each parameter. However, this would make it more difficult to compare parameters which are located on opposite ends of the plot, although in this case each of the bars would all have a height of zero. For this reason, a 2-dimensional plot was chosen instead to better compare each of the parameters. This plot is shown in Fig. 5.8 and each parameter value is plotted as a horizontal line across the figure. This means that the x-axis is meaningless and the y-axis, which shows the sensitivities of the output signal amplitude, is the focus of the plot. The total system output was chosen to be the temperature of the fluid as it leaves the system, namely the temperature leaving vertex 9. Therefore, the total system output can be written as $T_{f,9}$, or the temperature of the fluid in vertex 9. As expected, the sensitivities of each of the total system output, $T_{f,9}$, are all zero.

5.2.1 Numerical Verification

To support the prediction in Fig. 5.8, each of the capacitance parameters can be perturbed individually before re-running the simulation to see if changing the parameter values affects the system output. Fig. 5.9 shows a series of numbers added to each of the parameters on the x-axis. The y-axis shows the steady-state amplitude of the output signal when the parameter was perturbed by the amount shown on the x-axis.

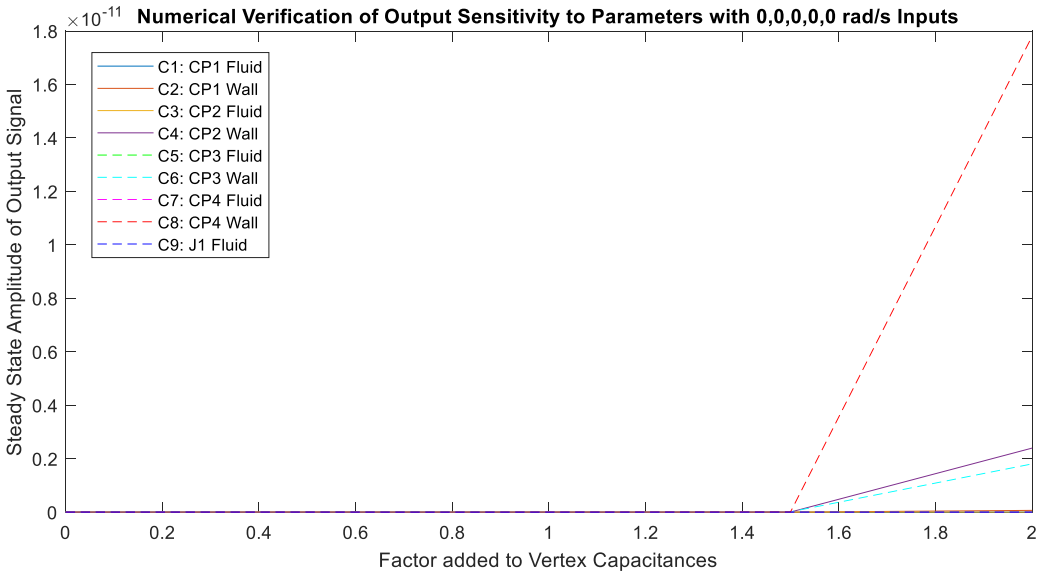

Figure 5.9: Numerical Verification of Sensitivity Analysis with Constant Inputs.

Notice the scale on the y-axis and how the amplitude of the output signal, regardless of any parameter perturbation, is always zero. The relative increases in amplitude between parameter perturbation values of 1.5 and 2 are due to the imperfections in the linearized model. The perturbation value of 2 is the farthest from the nominal values, where the model was linearized about. Therefore, this point is the least accurate. There are also minor inaccuracies between the other perturbation values; however, they do not appear on this plot since they are much less, due to being closer to the nominal parameter values. Hence, based on the scale of the y-axis, this plot demonstrates that the steady state output amplitude is insensitive to all the capacitance parameters. Fig. 5.10 shows a time trace of the system output and demonstrates how the nominal set of parameters and the set of parameters when either C_3, the fluid capacitance of Cold Plate 2, or C_4, the wall capacitance of Cold Plate 2, is perturbed do not affect the steady state amplitude of the output signal.

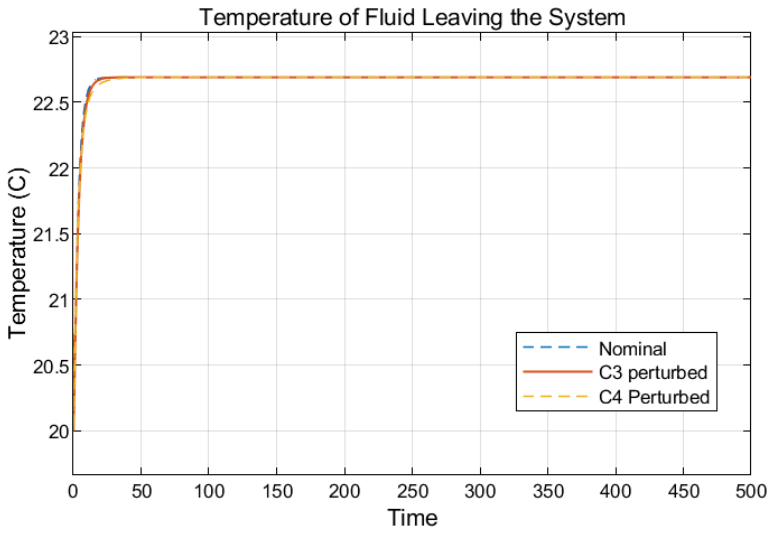

Figure 5.10: Perturbed Simulations Compared Against Nominal Case.

Even though the steady-state responses of the system with differing parameter values are equal, the time responses do vary when the capacitance values are adjusted. Fig. 5.11 demonstrates how the nominal set of parameters show the quickest time response and the smallest time constant. However, when the fluid and wall capacitances of Cold Plate 2 are increased, they slow down the time response, resulting in a greater time constant. Physically perturbing the fluid capacitance adds more fluid to the system, specifically in Cold Plate 2. This results in slowing down the rate at which the fluid in the system, and thus the fluid exiting the system, heats up. Similarly, when the capacitance of the wall in Cold Plate 2 is larger, the cold plate wall takes longer for the heat load to heat it up and subsequently takes longer to heat up the fluid passing by it. This also results in increasing the time taken for the temperature of the fluid leaving the system to reach its steady state value.

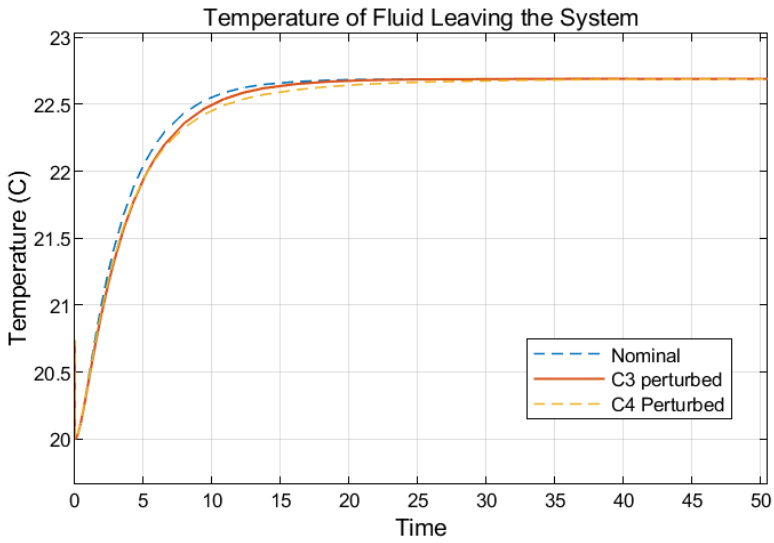

Figure 5.11: Zoomed-In Image of Fig 5.10 to Show the Varying Time Responses.

From Fig. 5.11, perturbing C_4 is shown to slow down the time response more than perturbing C_3 does. This has not yet been analytically determined using the sensitivity analysis methods outlined in this thesis. This work discusses the sensitivities of the steady state transfer function gain only, and not the sensitivity of the transient dynamics. Although this is outside the scope of this thesis, this would be an interesting topic for further study and will be discussed again in the future work section of Chapter 7.

5.3 Input 1 as a 1 rad/s Sinusoidal Wave

The second scenario to be discussed is when all the input signals are constant except for Input 1, the source temperature, which has a frequency of 1 rad/s. The estimated output amplitude contributions from each of the transfer functions are shown in Fig. 5.12 and summed together in Table 5.2.

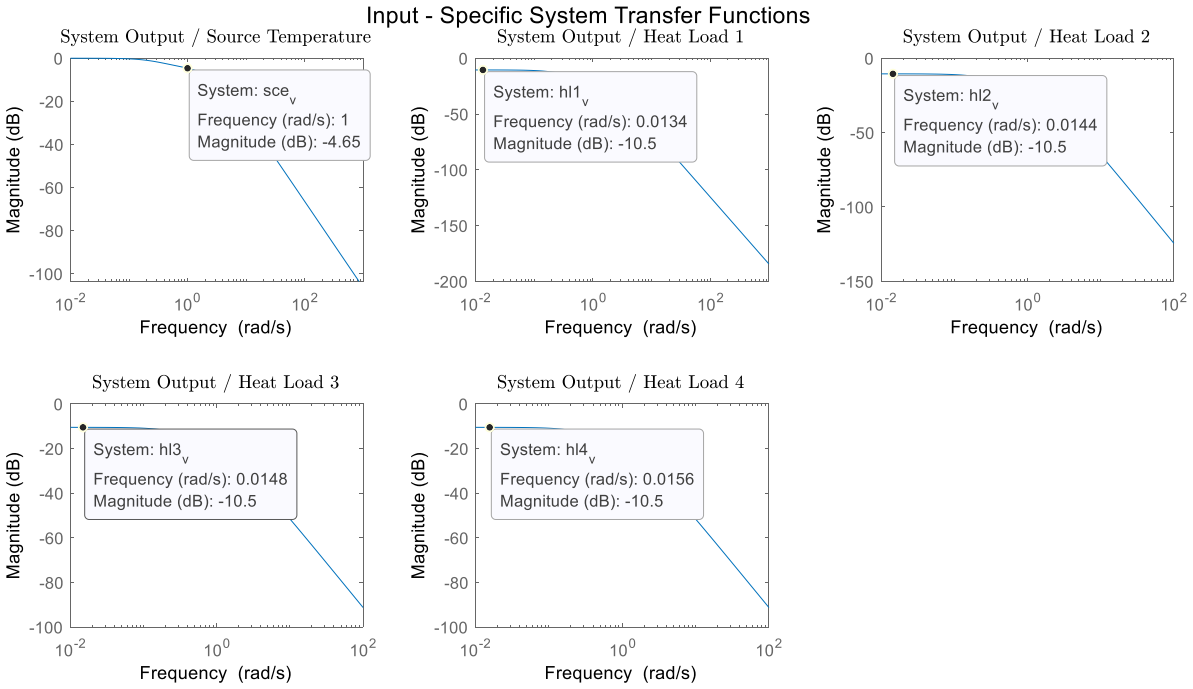

Figure 5.12: Transfer Function Gains when Input 1 is at 1 rad/s and the Rest are Constant.

	Source	Heat Load 1	Heat Load 2	Heat Load 3	Heat Load 4	Total
Value at freq of interest (dB)	-4.65	-10.5	-10.5	-10.5	-10.5	
Convert to gain	0.5855	0.2985	0.2985	0.2985	0.2985	
Amplitude	20	1	1	2	5	
Max Value *gain	+11.71	0.2985	0.2985	0.5971	1.4927	14.40
Min Value *gain	-11.71	0.2985	0.2985	0.5971	1.4927	-9.02

Table 5.2: Breakdown of Contributions to Output Amplitude from Input Signals.

Notice in Table 5.2 that the amplitude of the source temperature is 20, but it has a maximum value of 20 and a minimum of -20. This is because Input 1 now has a frequency of 1 rad/s, so the amplitude oscillates between values of -20 and 20. The first row of Table 5.2 records the gain in dB for each of the input frequencies of interest. For Input 1, the frequency of interest is 1 rad/s, and for the rest of the inputs the frequency of interest is 0 rad/s. Fig. 5.12 shows how the magnitude in dB is found for each of the input-specific system transfer functions at their desired frequencies. Row 2 of Table 5.2 converts these values to gains and row 3 shows the amplitude of the input

signal. The amplitude is used to determine the maximum and minimum values of the signal, which will be multiplied by the gains to yield the entries in rows 4 and 5. The final column in table 5.2 shows the sum of the contributions from each transfer function and predicts that the output signal will oscillate between a maximum value of 14.40 and a minimum of -9.02. The numerical verification of Scenario 2 is shown in Fig. 5.13.

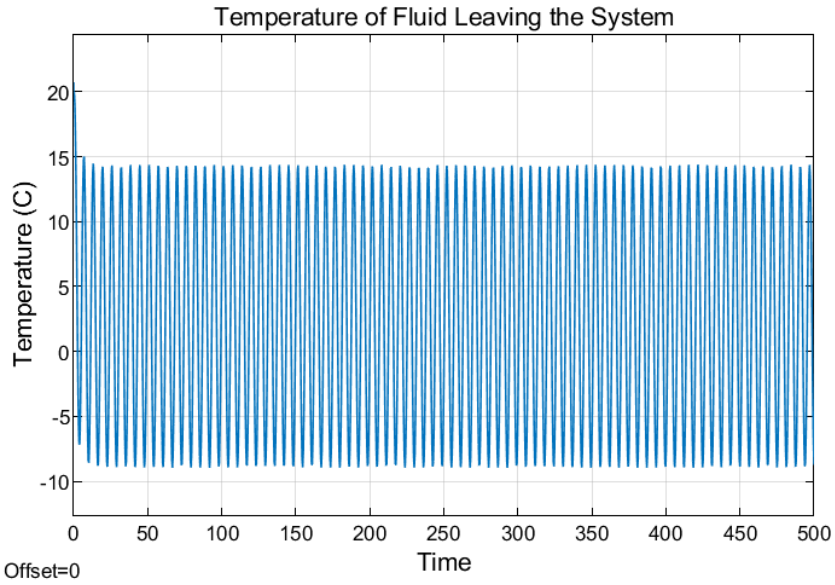

Figure 5.13: Numerical Verification of Output Amplitude with Input 1 at 1 rad/s.

The peak of the output signal is very close to 14.39 and the trough is close to -8.97. These values are very close and build confidence in the system equations that will be used for the sensitivity analysis. Any minor discrepancy between the estimate and the numerical verification could either be due to the assumption made by the graph model that the mass flow rates are always constant or due to the lack of definition in the sinusoidal input signal. To improve this, the sample time in the sine wave block in Simulink could be set to a small value, such as 0.01 s or 0.001 s. This would lead to more accurate results; however, the simulation will take much longer and the level of detail in Fig. 5.13 is already more than sufficient to identify the top few parameters in the sensitivity analysis.

Fig. 5.14 highlights the sensitivity of the system transfer functions, shown in Fig. 5.12, to each of the parameters. The partial derivatives of the output over the first input transfer function are zoomed in to show the ranking of the parameter influence around 1 rad/s. Notice how the top five most-influential parameters are all fluid capacitances, and the bottom four least-influential

parameters are all wall capacitances. This makes physical sense, because the source temperature transfer function relates the gain between the output flow temperature over the input flow temperature. Therefore, increasing the fluid capacitance of the components will have a greater effect in filtering out the high frequencies of the sinusoidal input signal, because they directly impact the total system's fluid capacitance. If there is more fluid in the system, the rate at which the system reacts to the input flow, which dips down to -20 °C and increases again to 20 °C, will slow down. This has a filtering effect on the system temperature and results in a lower amplitude of the output temperature signal. While the wall capacitances might slow down the heat transfer through the system by their convective interactions with the fluid, they cannot come close to the effect of directly increasing the fluid capacitances. This is demonstrated by the sensitivity ranking of Input 1's sensitivity bode plot, shown in the top left plot of Fig. 5.14.

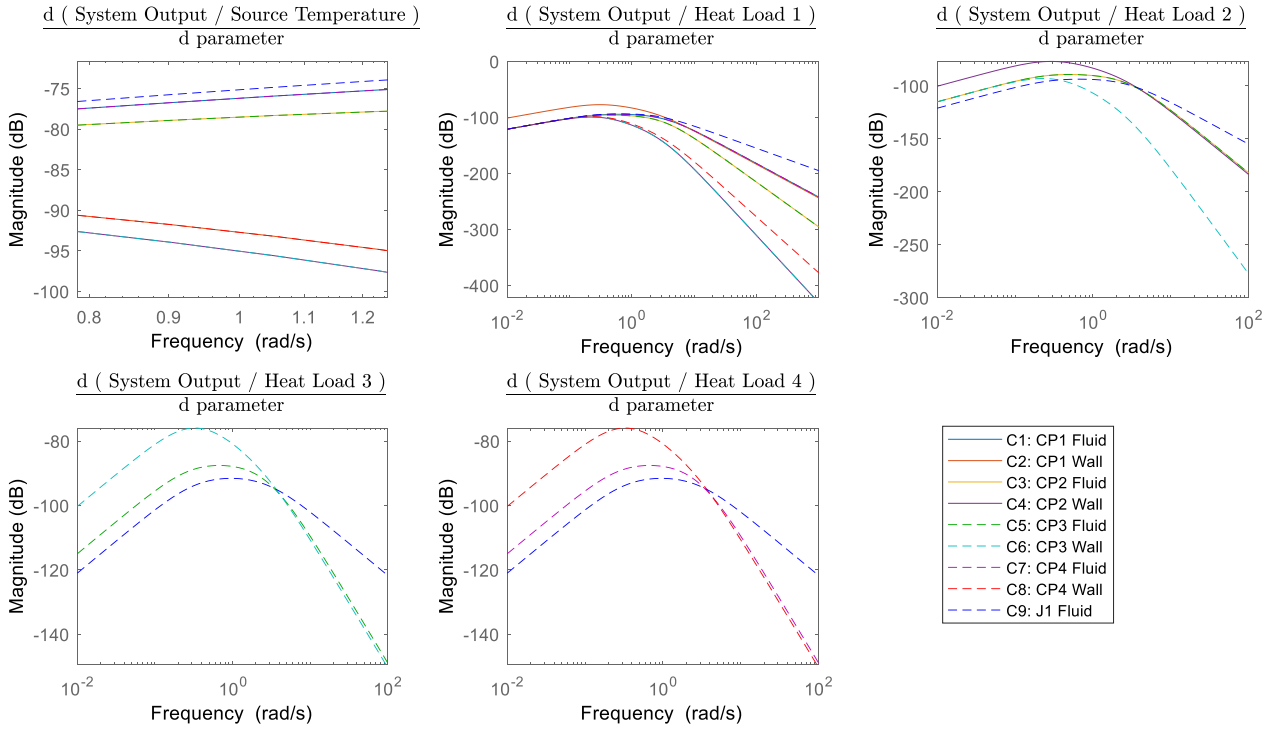

Figure 5.14: Sensitivity Bode Plots when Input 1 is at 1 rad/s and the Rest are Constant.

Combining all the sensitivity bode plots for each input together in the manner described in Fig. 5.7 results in the sensitivity plot in Fig. 5.15. Since all the inputs, other than Input 1, have frequencies of zero and the DC gains of these sensitivity bode plots are negligible, the combined

sensitivity plot greatly resembles the sensitivity bode plot of Input 1 at the frequency of interest: 1 rad/s.

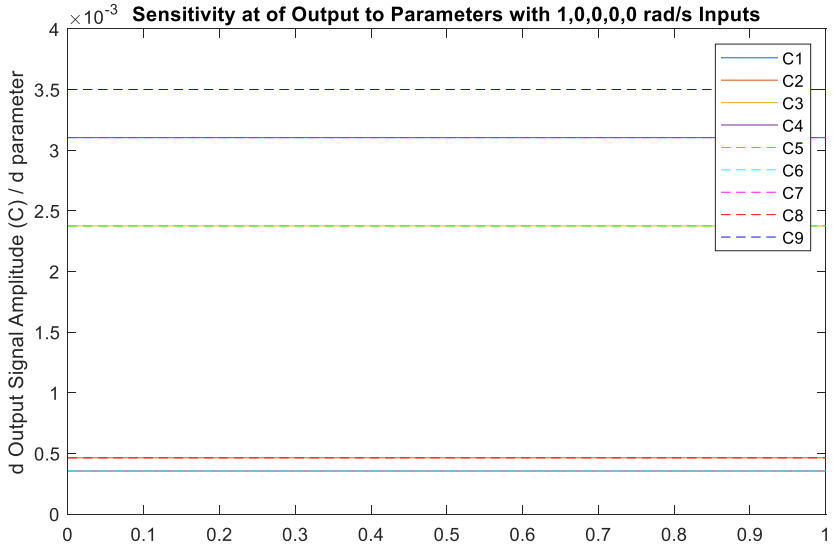

Figure 5.15: Sensitivity Plot of Scenario 2.

Notice how the fluid capacitances, C_9, C_1, C_7, C_3, and C_5, are significantly more influential to the output temperature than the wall capacitances, C_2, C_8, C_6, and C_4. This ranking is verified through numerical simulation in Section 5.3.1 below.

5.3.1 Numerical Verification

Fig. 5.16 shows the resulting amplitude of the output signal at steady state when each of the capacitance parameters are perturbed.

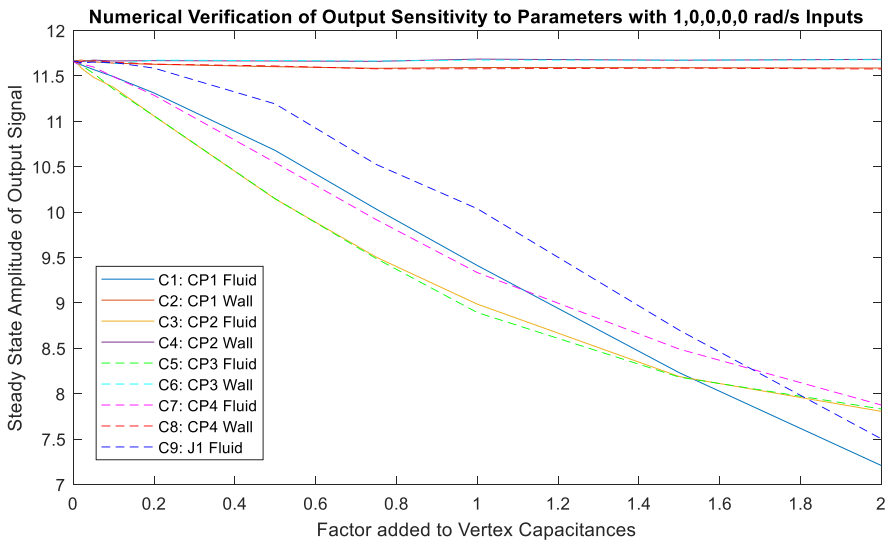

Figure 5.16: Numerical Verification of Scenario 2.

A horizontal line means that no matter what the parameter values were perturbed to, the resulting output signal remained constant in amplitude. This indicates that the output signal amplitude is insensitive to the parameters which produce horizontal lines. However, the lines with the greatest slope where the parameters deviate from the nominal parameters (small perturbations) indicate the parameters which the output temperature is the most sensitive to. The sensitivity analysis is most accurate when the parameter values are close to the nominal values, because that is the point the graph-based model was linearized about and the mass flow rates were taken from. The further the deviation from the nominal values, the less accurate the model which was used to predict sensitivity becomes. The parameters which produce the steepest lines near the nominal values for Scenario 2 are C_3, C_5, C_7, C_1, and C_9, respectively. These don't line up perfectly with the predictions in Fig. 5.15, which shows C_9 as the most influential, followed by C_7 and C_1, then C_3 and C_5. However, both identify that the fluid capacitances are close together in influence and significantly more influential than the wall capacitances. Possible explanations for this deviation include the model imperfections due to linearizing it and assuming constant mass flow rates, low definition in the sinusoidal waves acting as inputs, and inaccuracies in how the steady state amplitude is calculated. Once the simulation has finished running, the last 10% of the temperature data is examined, and the maximum and the minimum values in that section are added together and divided by two. This is then recorded as the steady state amplitude when the parameter in question was perturbed by a given amount. There can be inaccuracies in this calculation if the

output signal value spikes at any point in the final 10% or dips lower than it should due to numeric errors in the simulation.

Fig. 5.17 shows a time trace of the simulation with the nominal set of parameters compared to the simulation results when parameters C_3 and C_4 are individually perturbed by a value of 2. Notice that the fluid capacitance, C_3, has a much greater filtering effect on the output signal than the wall capacitance, C_4. Perturbing the wall capacitance did not change the output signal from what it was with the set of nominal parameters, just as the horizontal line for parameter C_4 in Fig. 5.16 demonstrated. Fig. 5.18 shows a zoomed-in version of Fig. 5.17 to display the amplitude comparisons more clearly.

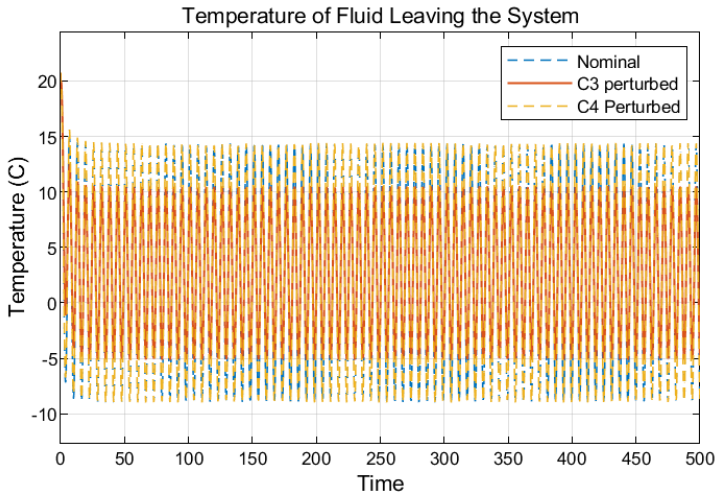

Figure 5.17: Perturbed Simulations Compared Against Nominal Case for Scenario 2.

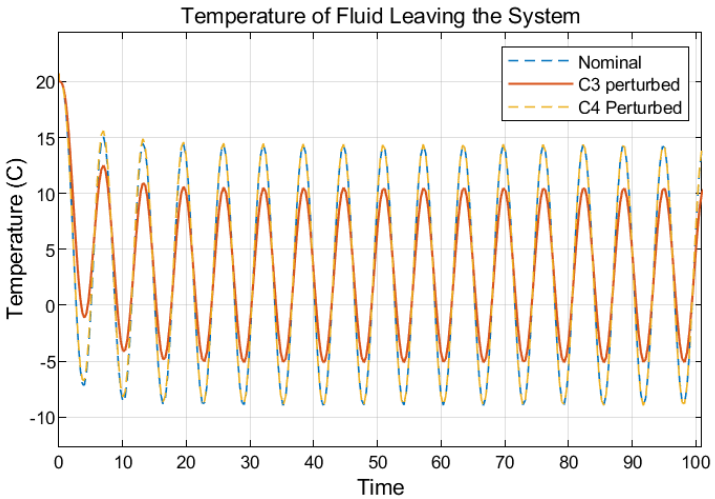

Figure 5.18: Zoomed-In Image of Fig 5.17.

5.4 Inputs 2 and 3 as Sinusoidal Waves at 1 and 0.5 rad/s

The third scenario is defined by setting all the input signal frequencies to 0 rad/s, except for Inputs 2 and 3, which have frequencies of 1 rad/s and 0.5 rad/s, respectively. The system transfer functions shown in Fig. 5.19 are used to estimate the resulting output amplitude based on the input signal frequencies. The magnitude of the gain is converted from dB and multiplied by each of the input signal amplitudes in Table 5.3. Finally, the contributions to the output signal amplitude are added together to estimate the total output signal amplitude.

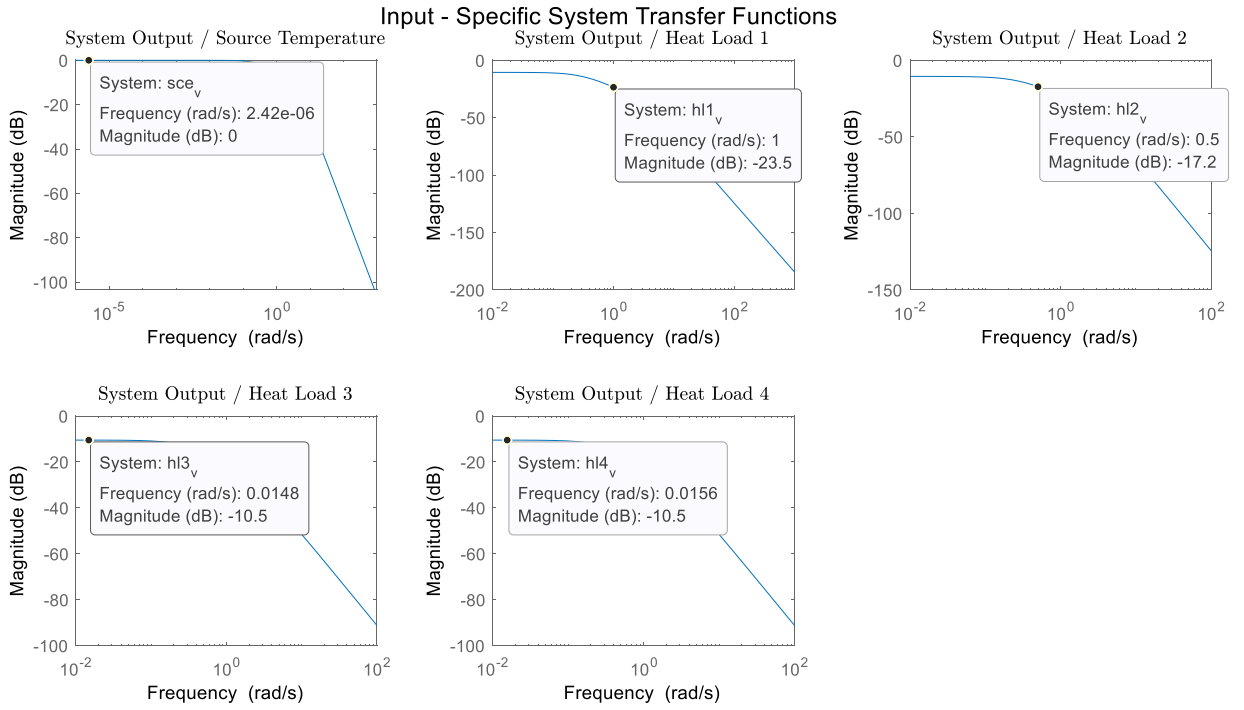

Figure 5.19: Transfer Function Gains for Scenario 3.

	Source	Heat Load 1	Heat Load 2	Heat Load 3	Heat Load 4	Total
Value at freq of interest (dB)	0	-23.5	-17.2	-10.5	-10.5	
Convert to gain	1	0.0668	0.1380	0.2985	0.2985	
Amplitude	20	1	1	2	5	
Max Value *gain	20	+0.0668	+0.1380	0.5971	1.4927	22.29
Min Value *gain	20	-0.0668	-0.1380	0.5971	1.4927	21.89

Table 5.3: Breakdown of Contributions to Output Amplitude from Input Signals.

The final column of Table 5.3 indicates that the output signal will oscillate between 22.22 °C and 21.96 °C. The numerical verification shows oscillation between 22.29 °C and 21.95 °C in Fig. 5.20. Therefore, Fig. 5.20 shows the system transfer functions remain reliable at other frequencies and supports their further use in the next step of the sensitivity analysis.

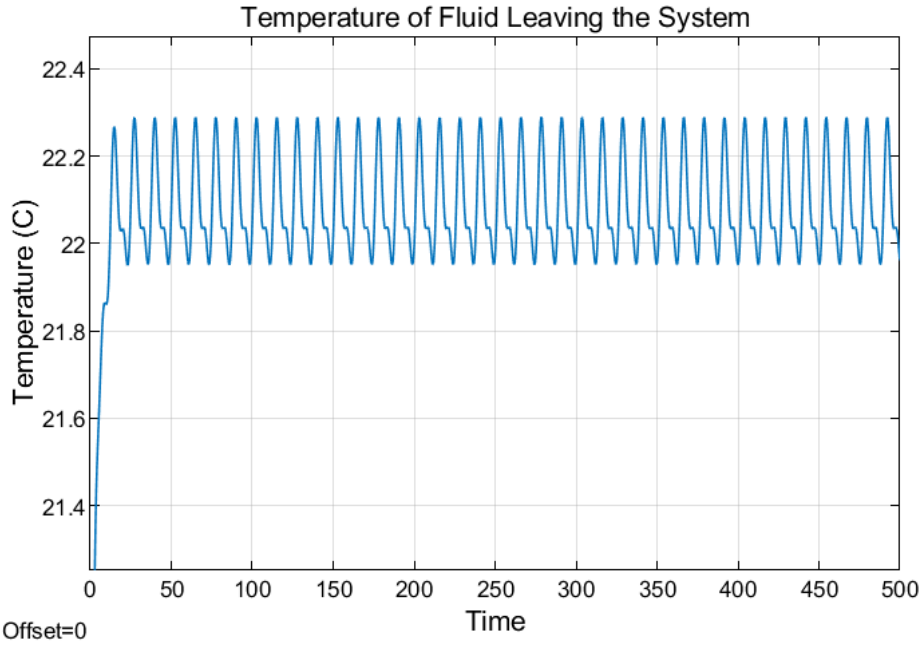

Figure 5.20: Numerical Verification of Output Amplitude for Scenario 3.

Next, the partial derivatives of each of the system transfer functions with respect to the parameters are calculated. These sensitivity bode plots for Scenario 3 are shown in Fig. 5.21. Notice that the sensitivities of the transfer functions for Inputs 1, 4, and 5 continue to decrease at low frequencies. Since Inputs 1, 4, and 5 are constant values, the sensitivity bode plots indicate the contributions to the output temperature from Input Signals 1, 4, and 5 are insensitive to any of the parameters. Therefore, the only contributions to the output temperature which are sensitive to changes in parameters are the ones from Input Signals 2 and 3. The sensitivity bode plots for the transfer functions of Inputs 2 and 3 are zoomed-in to the frequencies of interest, namely 1 rad/s and 0.5 rad/s, respectively.

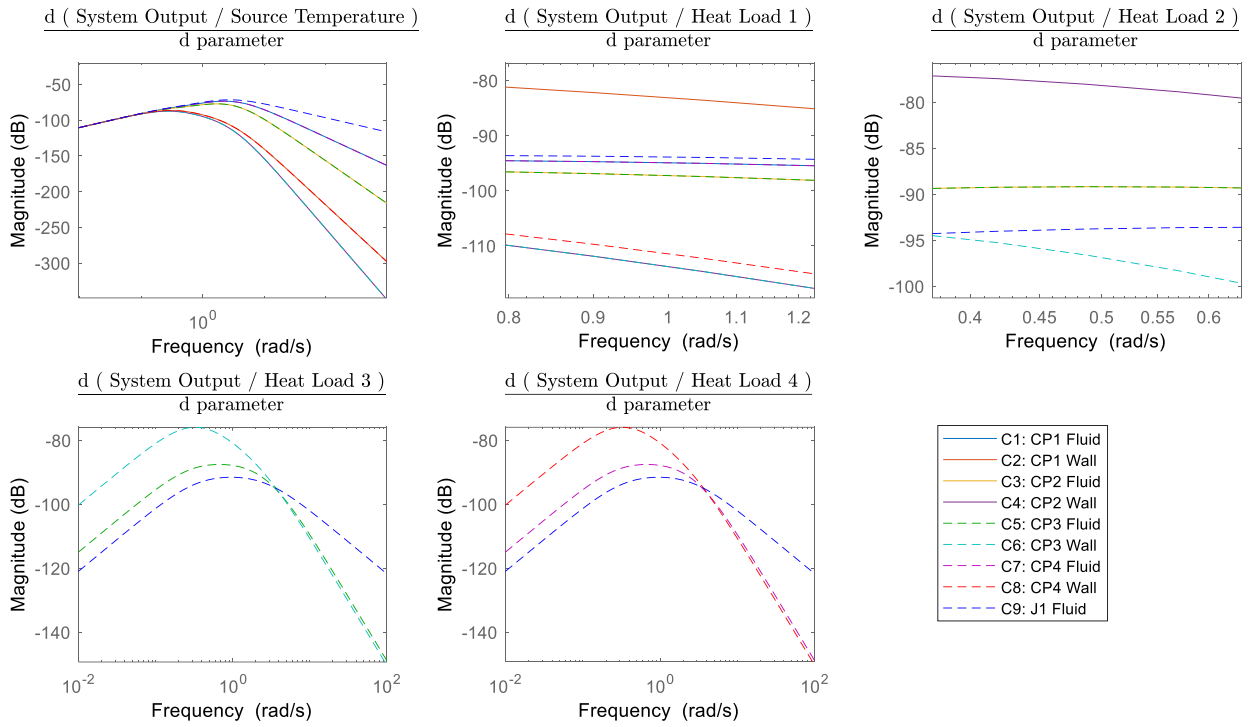

Figure 5.21: Sensitivity Bode Plots for Scenario 3.

Remember that both Heat Load 1 and Heat Load 2 have the same magnitude of 1 kW, meaning that they will be weighted equally. Across both transfer functions of Heat Loads 1 and 2, parameter C_4 has the highest value of -78 dB. Since the transfer functions are equally weighted, this might mean C_4 is the most influential. However, C_4 is only the most influential parameter in one sensitivity bode plot, and other parameters, such as C_3 and C_5, have high magnitudes in both sensitivity bode plots of interest, namely the sensitivities of the transfer functions relating the output temperature to Heat Loads 1 and 2. Therefore, it is not immediately apparent whether parameter C_4 or parameters C_3 and C_5 will be the most influential overall.

Parameter C_2 is the most influential parameter for the Heat Load 1 transfer function. However, it does not even appear in the sensitivity bode plot for the transfer function of Heat Load 2, indicating it has no effect on that transfer function. This is because C_2 is the wall capacitance of Cold Plate 1, and the transfer function of Heat Load 2 happens after the fluid has already passed through the first cold plate. Therefore, C_2 has no effect on that transfer function which relates the output to the third input. This makes it impossible for C_2 to surpass parameter C_4 in influence, since C_4 had a higher magnitude to begin with and also appears in more transfer function equations.

The parameters that transfer functions 2 and 3 are most sensitive to are the wall capacitances. This makes sense physically, because Inputs 2 and 3 are heat loads on Cold Plates 1 and 2. And the larger the capacitance of the cold plate wall is, the more energy it can absorb before heating up. This results in slowing down the transient dynamics and filtering the sine wave of the heat load applied to the cold plate wall. This ability to filter out some of the higher frequencies is increased when the capacitance of the cold plate wall increases. The reason C_2 is the most impactful parameter to the Heat Load 1 transfer function and C_4 is the most impactful parameter to the Heat Load 2 transfer function is that Heat Load 1 is applied directly to the wall capacitance C_2 and Heat Load 2 is applied directly to C_4. The walls of the cold plates have the greatest impact on the transfer functions relating the output to the heat load applied to that cold plate.

Although C_4 has the single highest magnitude in the sensitivity bode plots, both transfer functions are affected significantly by parameters C_3 and C_5, the fluid capacitances of cold plates 2 and 3. These cold plates are the only cold plates downstream of both sinusoidal heat loads, resulting in the output being the most sensitive to C_3 and C_5 out of all the other fluid capacitances.

When the sensitivity bode plots are combined, following the methodology shown in Fig. 5.7, they result in the sensitivity plot in Fig. 5.22.

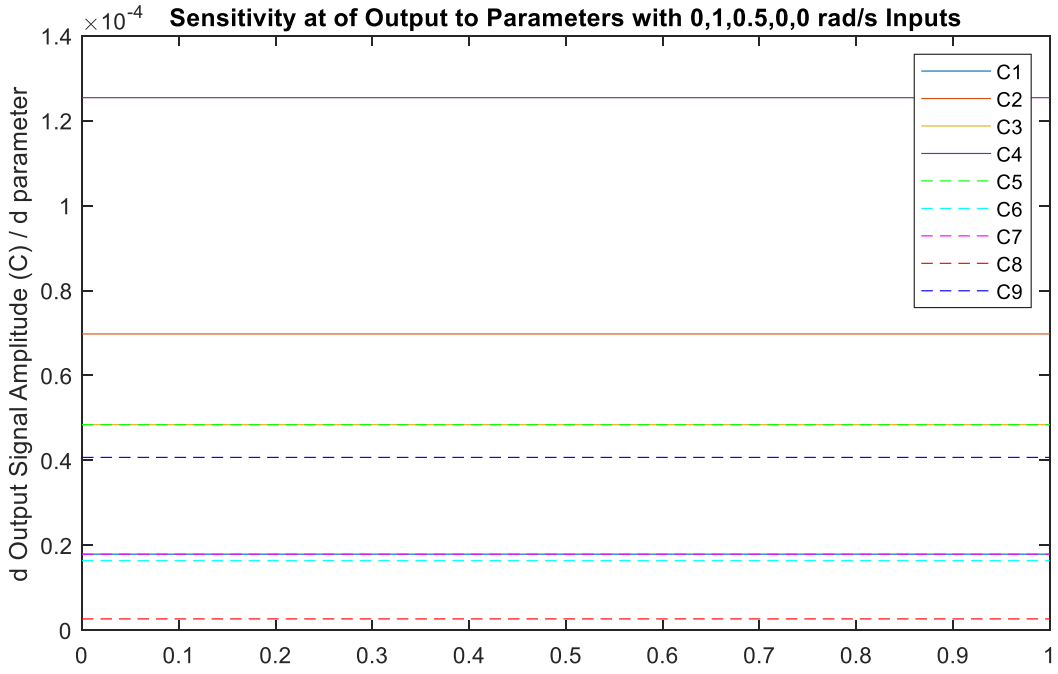

Figure 5.22: Sensitivity Plot of Scenario 3.

Figure 5.22 shows that the wall capacitance of Cold Plate 2, C_4, is expected to be the most influential, followed by the wall capacitance of Cold Plate 1, C_2. Then the fluid capacitances, C_3 and C_5, are expected to be next, followed by the fluid capacitance of the junction, C_9. Finally, parameters C_7, C_1, C_6, and C_8 are predicted to have the next greatest effects on the output signal, in that order. The numerical verification is shown in Fig. 5.23.

5.4.1 Numerical Verification

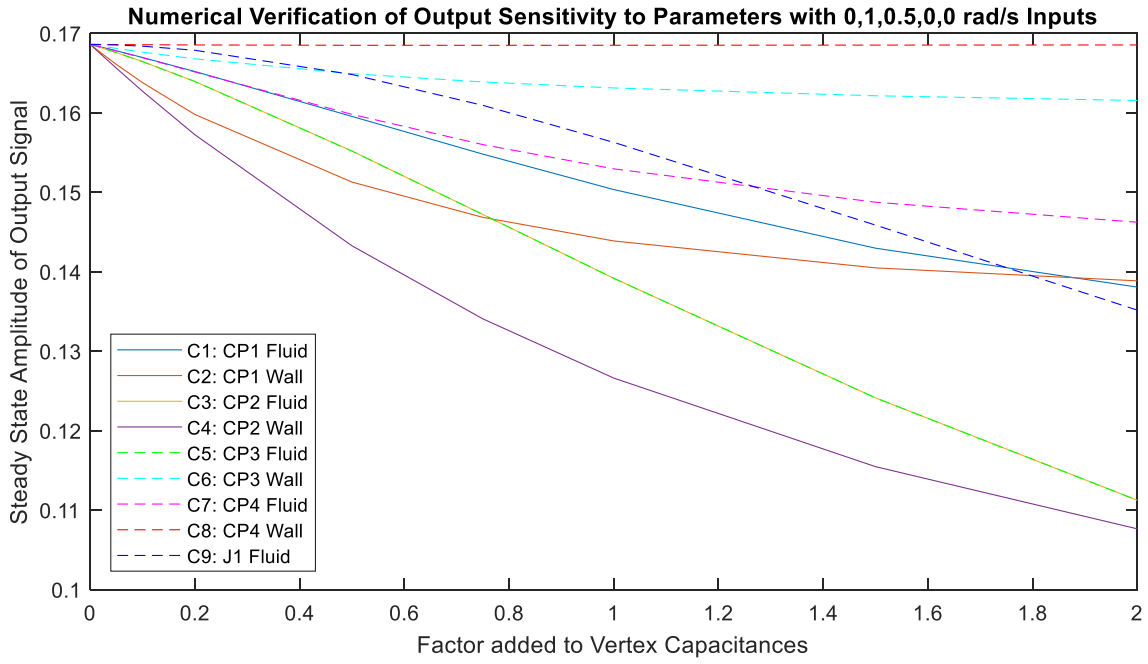

Figure 5.23: Numerical Verification of Scenario 3.

Fig. 5.23 shows the steady state amplitude of the output signal for a variety of parameter perturbations. The highest four parameters, C_4, C_2, C_5, and C_3, are the top four parameters for both the sensitivity plot in Fig. 5.22 and the numerical verification in Fig. 5.23. The order of these parameters is correct as well, when looking at the slopes of the lines closest to the nominal parameters. The next four parameters are close, but the order is slightly jumbled up. However, this method is sufficiently accurate to identify the top few parameters, which become the focus of the optimization efforts. Fig. 5.24 and Fig. 5.25 below show the time traces of the output signal for both the nominal case and when parameters C_3 and C_4 are perturbed. Clearly, these parameters are

two of the ones which had the greatest impact on the system, as shown by the notable difference in amplitude when compared to the nominal case.

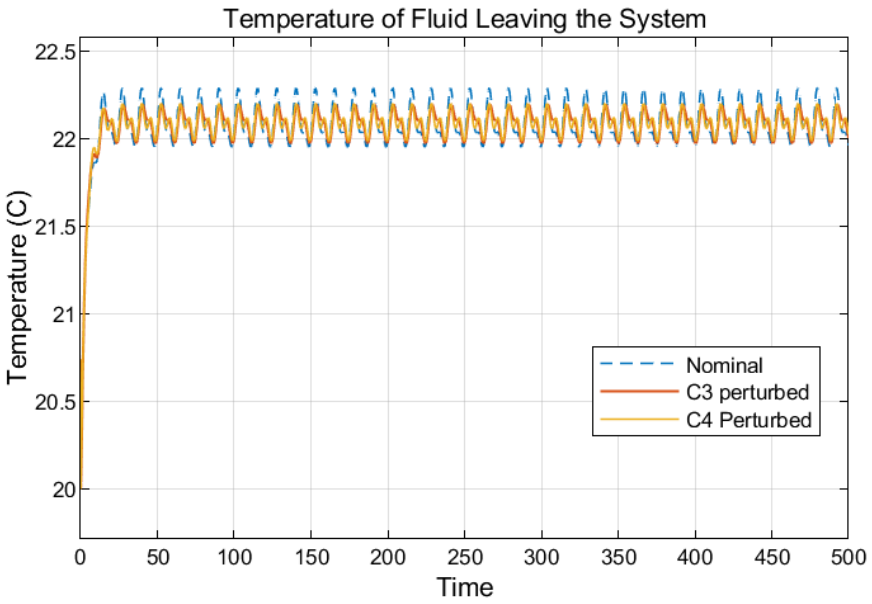

Figure 5.24: Perturbed Simulations Compared Against Nominal Case for Scenario 3.

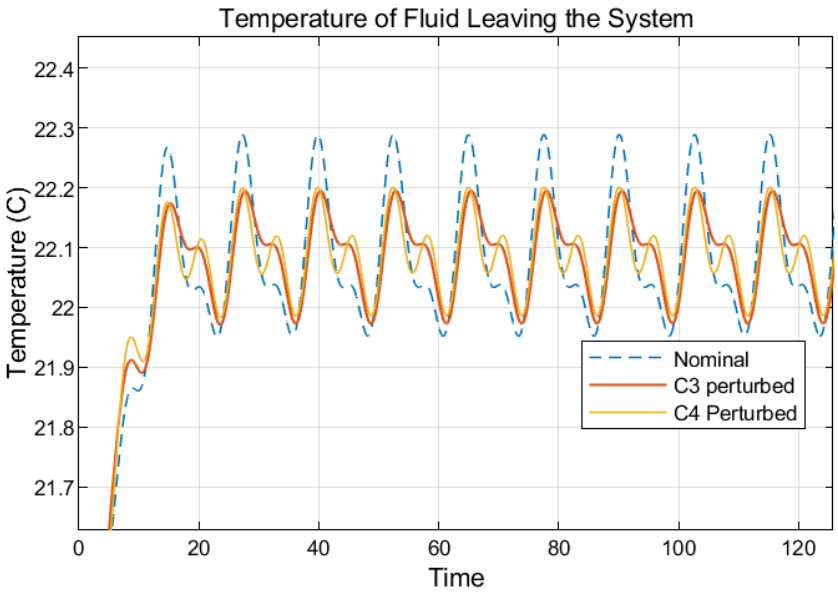

Figure 5.25: Zoomed-In Image of Fig 5.24.

Fig. 5.25 demonstrates how close the influence each of these parameters have on the output amplitude is. It is hard to tell from looking at the plot, but the yellow line for C_4 has a marginally smaller amplitude when compared to the orange line for C_3. This can be seen best when comparing

the troughs of the waveforms. Since both these parameters are in the top four and have similar influences on the system, this suggests that the top several parameters should be selected for optimization efforts. Only optimizing the top parameter will not have as large of an effect as optimizing the top few parameters. The goal of the sensitivity analysis is to identify the top several parameters which the outputs are similarly sensitive to, and thus reduce computational time needed for optimization by discarding the parameters which the system is insensitive to. This means that the exact ranking of the top 4-5 parameters is not ultimately important since the purpose of the sensitivity analysis is simply to identify those parameters which are the best candidates for optimization. However, in this case, the sensitivity analysis and numerical verification agree on the ranking of the top four parameters. Fig. 5.26 demonstrates the importance of optimizing more than one parameter at a time – notice how perturbing both C_3 and C_4 results in a smaller output signal amplitude than either of the two parameters could achieve alone.

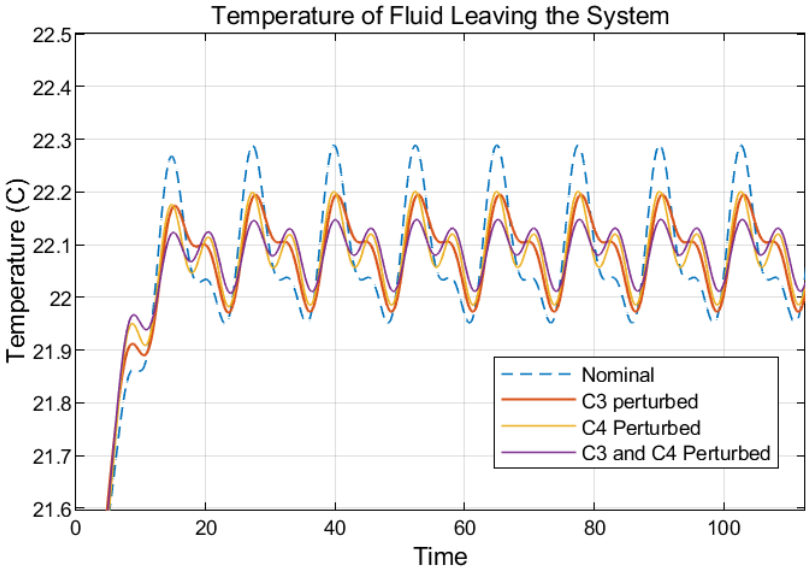

Figure 5.26: Perturbed Simulation Where C_3 and C_4 are Both Perturbed.

5.5 All Inputs as Sinusoidal Waves

This final section discusses Scenario 4, where all of the input signals are sinusoidal. The frequencies of Inputs 1-5 are 1 rad/s, 1 rad/s, 0.5 rad/s, 0.1 rad/s, and 1.5 rad/s, respectively. Notice that the frequencies for the first three inputs are the same as they were in Scenarios 1 and 2. The

system transfer functions for each of the inputs in Fig. 5.27 show the gains for the output over input amplitudes at each of the frequencies of interest.

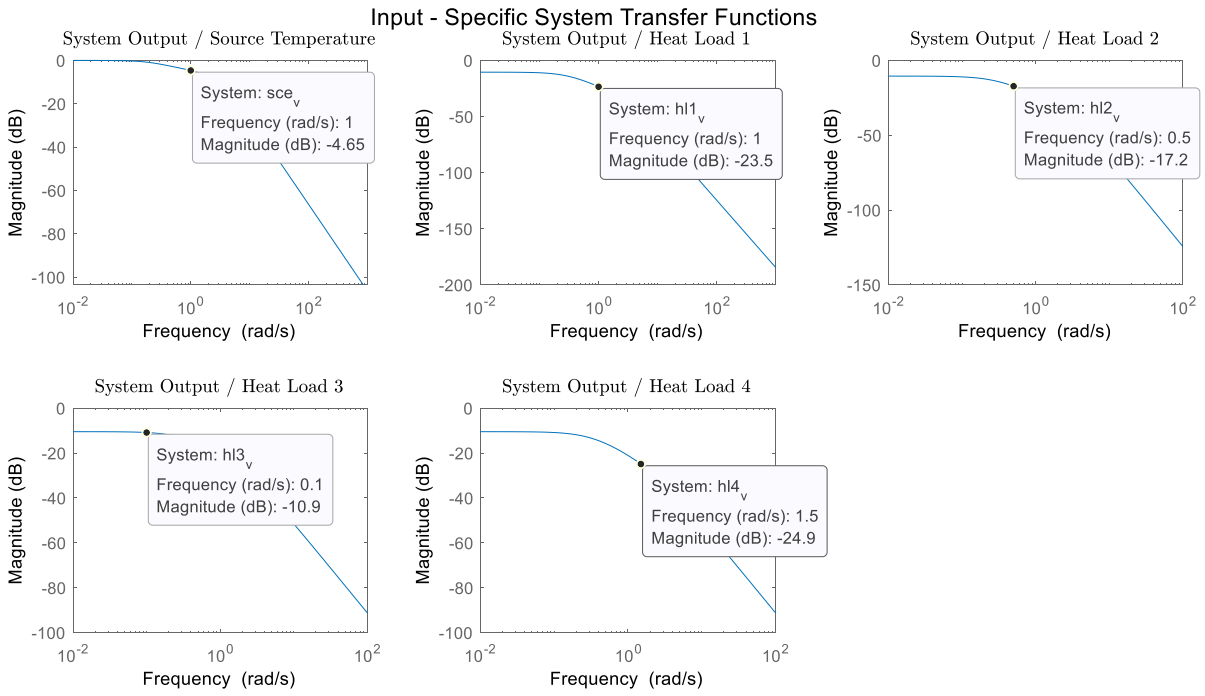

Figure 5.27: Transfer Function Gains for Scenario 4.

These transfer function gains can be converted from dB and multiplied by the input amplitude to yield the output amplitude contribution from each input-specific transfer function. Then all the output signal contributions can be added together in the final column of Table 5.4, estimating the total amplitude expected for the output signal in Scenario 4. The expected output signal will range from 12.8 °C to -12.8 °C.

	Source	Heat Load 1	Heat Load 2	Heat Load 3	Heat Load 4	Total
Value at freq of interest (dB)	-4.65	-23.5	-17.2	-10.9	-24.9	
Convert to gain	0.5855	0.0668	0.1380	0.2851	0.0569	
Amplitude	20	1	1	2	5	
Max Value *gain	+11.71	+0.0668	+0.1380	+0.5702	+0.2844	+12.8
Min Value *gain	-11.71	-0.0668	-0.1380	-0.5702	-0.2844	-12.8

Table 5.4: Breakdown of Contributions to Output Amplitude from Input Signals.

Figure 5.28 shows the numerical verification of the output amplitude, which oscillates between 12.5 °C and -12.2 °C. This matches the expected amplitude well and builds confidence in the system transfer functions, which will be needed for the next step of the sensitivity analysis.

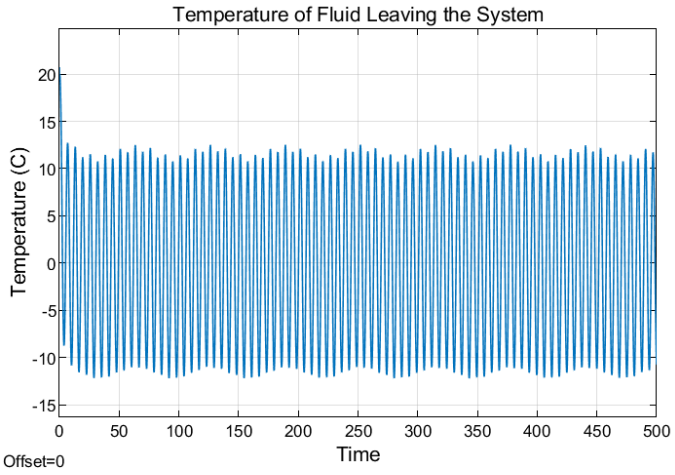

Figure 5.28: Numerical Verification of Output Amplitude for Scenario 4.

The next step is to take the partial derivatives of the system transfer functions shown in Fig. 5.27 with respect to each of the capacitance parameters. These sensitivity bode plots are shown in Fig. 5.29.

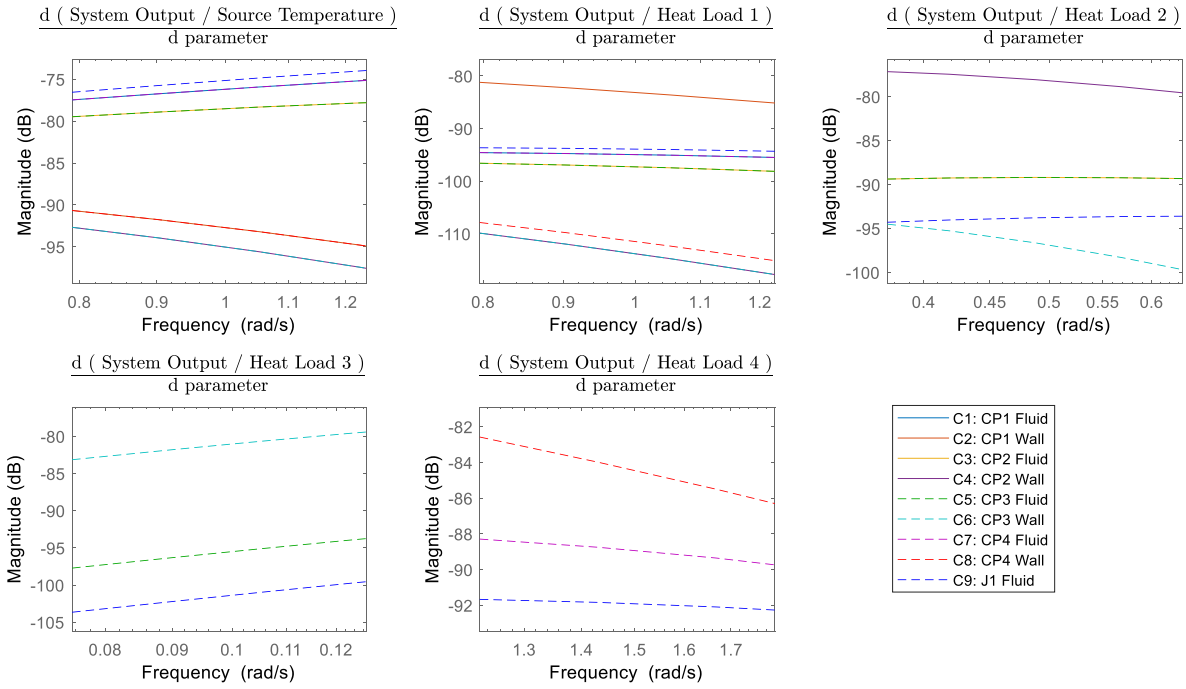

Figure 5.29: Sensitivity Bode Plots for Scenario 4.

In Fig. 5.29, notice that only three parameters show data for the transfer function relating the output temperature to Heat Load 3. This is because that transfer function equation only consists of the parameters C_5, C_6, and C_9, which are the fluid and wall capacitances of Cold Plate 3 and the fluid capacitance of the junction. As shown in the system schematic in Fig. 5.1, once the fluid passes through Cold Plate 3, it heads straight to the junction and then immediately exits the system. Similarly, the fluid in Cold Plate 4, located in the opposite branch from Cold Plate 3, passes straight to the junction before exiting the system. Therefore, the transfer function for Cold Plate 4 only has the Cold Plate 4 fluid and wall capacitances as well as the junction fluid capacitance in its system transfer function equation. When the partial derivatives of that transfer function equation are calculated, only the parameters C_7, C_8, and C_9 will have non-zero values. For both of these sensitivity bode plots, the output sensitivities to the wall capacitance parameters are highest. This makes sense since the heat load is applied directly to the wall, giving the wall capacitance the most ability to filter the sinusoidal heat load.

Combining all the sensitivity bode plots together, following the method outlined in Fig. 5.7, results in the sensitivity plot in Figure 5.30. This shows that although there are 4 different heat loads, whose transfer functions are more sensitive to their respective cold plate walls, the fluid capacitances are still the most influential to the system. This is likely the case for two reasons. First, although the wall capacitances are the most influential for their specific transfer function, they do not have much of an effect on the other input-specific transfer functions. Meanwhile, all the transfer functions are moderately sensitive to the fluid capacitances along the route of the transfer function. So, the fluid capacitances have a larger combined effect since they are influential to multiple transfer functions, instead of being very influential to only one transfer function. The second reason is that the source temperature has the highest magnitude of all the other input signals. This magnitude multiplies the partial derivative of the output gain to yield the partial derivative of the output signal with respect to the parameters before being added to the other inputs, which are multiplied by their sensitivity bode plots. This amplifies the sensitivities of the source temperature transfer function, which is most sensitive to the fluid capacitances. Therefore, this supports why the output temperature is most sensitive to the fluid capacitance parameters, as shown in Fig. 5.30.

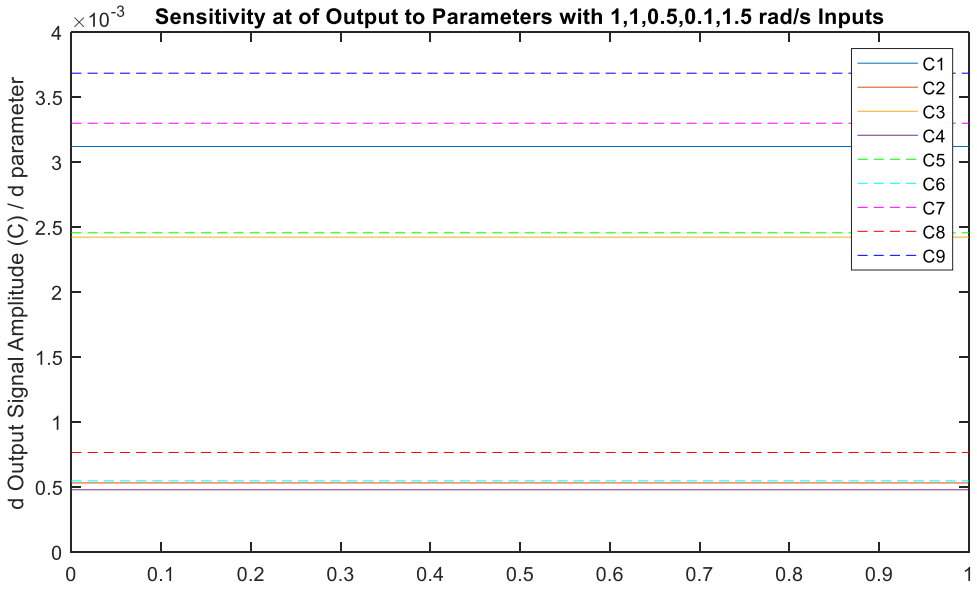

Figure 5.30: Sensitivity Plot of Scenario 4.

5.5.1 Numerical Verification

Fig 5.31 shows the numerical verification of Scenario 4. In the cases closest to the nominal parameter values, it is clear that the fluid capacitance parameters have the steepest slopes. The plots for the wall capacitances are close to horizontal and suggests that the wall capacitances do not have much of an effect on the output signal amplitude, as was predicted in Fig. 5.30.

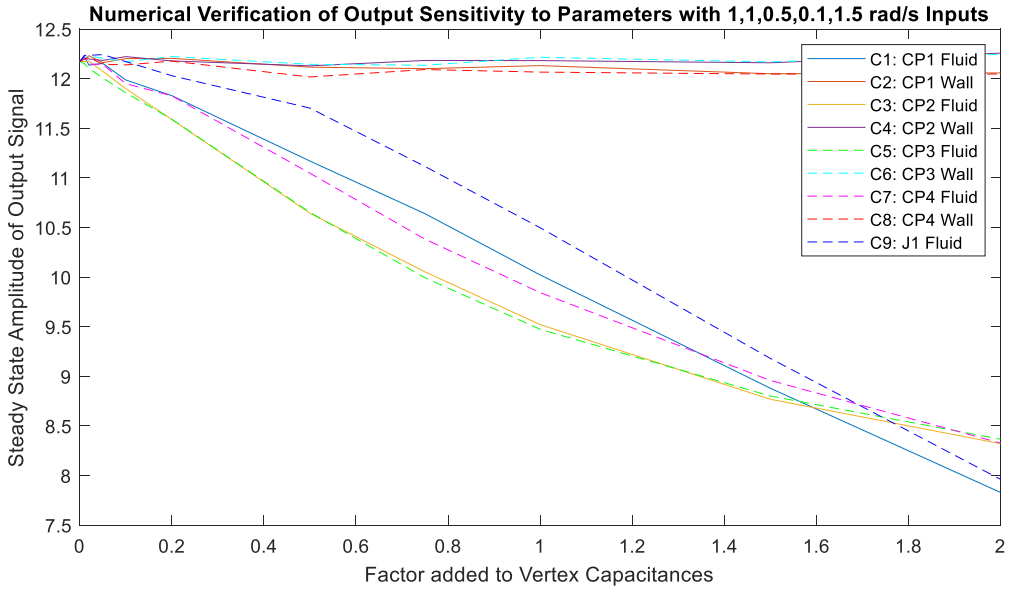

Figure 5.31: Numerical Verification of Scenario 4.

Fig. 5.32 below shows a time trace of the temperature of the fluid leaving the system for the nominal case, as well as when a wall capacitance, C_4, and a fluid capacitance, C_3, are perturbed. When C_4 is perturbed, it hardly deviates from the nominal case, indicating that the output temperature is insensitive to parameter C_4. However, when C_3 is perturbed, it noticeably filters some of the higher frequencies out of the output temperature amplitude, indicating that the amplitude of the output temperature is sensitive to parameter C_3 given the set of input amplitudes and frequencies in Scenario 4.

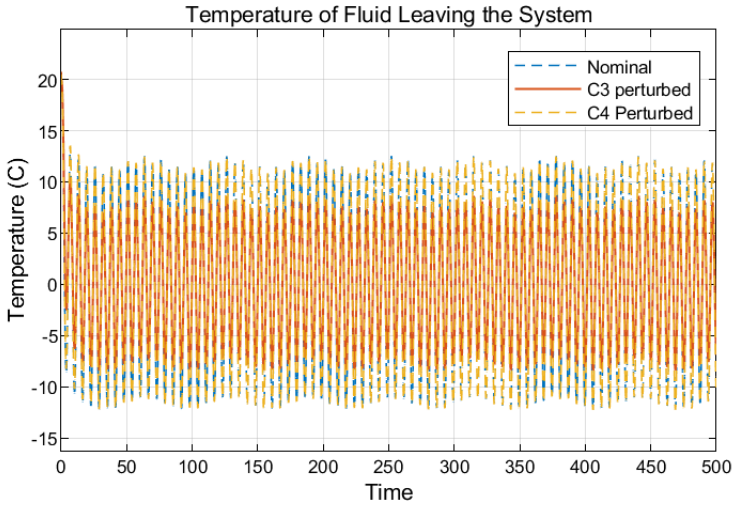

Figure 5.32: Perturbed Simulations Compared Against the Nominal Case for Scenario 4.

Figure 5.33 shows a zoomed-in version of the time trace. This simply makes the sensitivities discussed above easier to view.

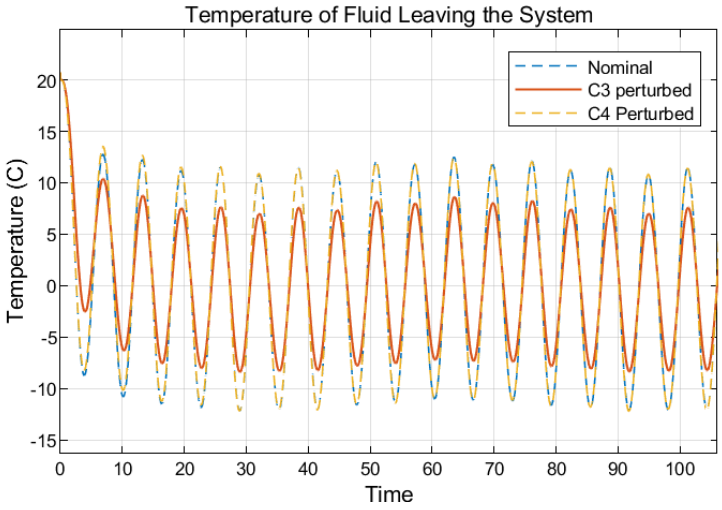

Figure 5.33: Zoomed-In Image of Fig 5.32.

This concludes the results and discussion section of the sensitivity analysis. The next chapter will apply closed loop controllers to Example System 2. These controllers will work to minimize the effect of an added heat load by increasing the mass flow rate into the system and maintain a temperature differential between the two branches by managing the valve percentage of flow to the top branch. A sensitivity analysis will be applied to discover which parameters would have the greatest effect in changing the output temperature, which is the subject of Controller 1, and thus which parameters should be optimized to minimize the control effort most efficiently. The setup in Scenario 3, where Heat Loads 1 and 2 are sinusoidal signals of amplitudes 1 kW and frequencies of 1 rad/s and 0.5 rad/s, respectively has been chosen as the setup for the co-design case study coming up in Chapter 6.

Chapter 6

Plant and Controller Co-design

6.1 Plant and Controller Setup

This section starts with the sensitivity analysis done for Example 2, the schematic of which is shown in Fig. 6.1, and the input signal configuration of Scenario 3 from Chapter 5. This scenario involves having two non-constant inputs, Heat Load 1 and Heat Load 2, both with amplitudes of 1 kW and at frequencies of 1 rad/s and 0.5 rad/s, respectively.

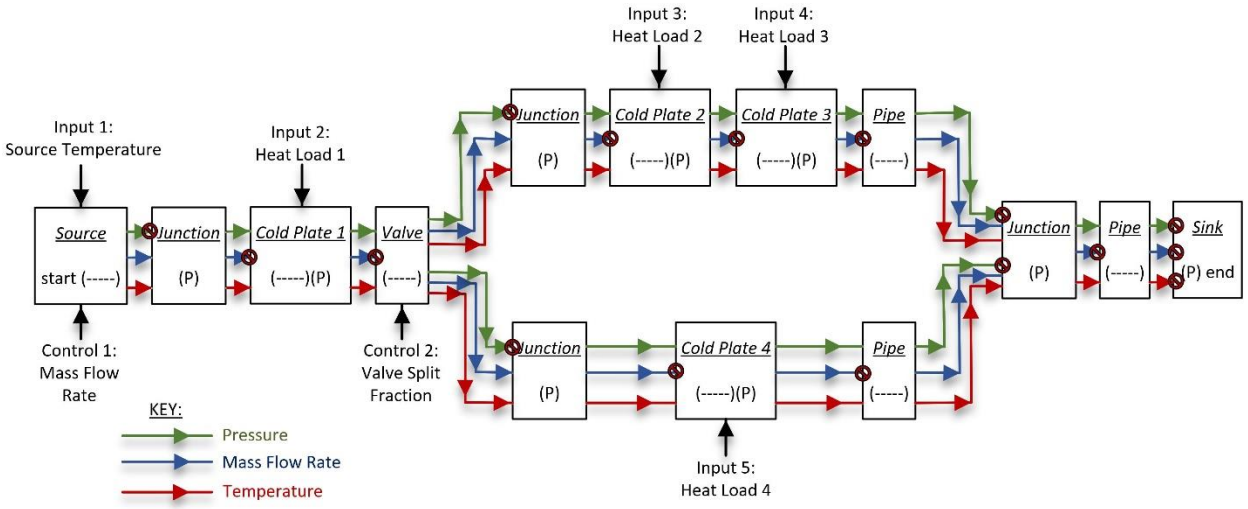

Figure 6.1: Schematic of Example 2.

The only difference in the system which will be analyzed in this chapter from the system of Scenario 3 in the previous chapter is the fourth input: Heat Load 3. Rather than Heat Load 3 being a constant heat load of 2 kW, it is now a series of step inputs. Heat Load 3 now rises to 15 kW for 200 seconds, 2000 seconds into the simulation, and then rises to 5 kW for 1500 seconds, 8000 seconds into the simulation. The mission profile for Heat Load 3 is shown in Fig. 6.2.

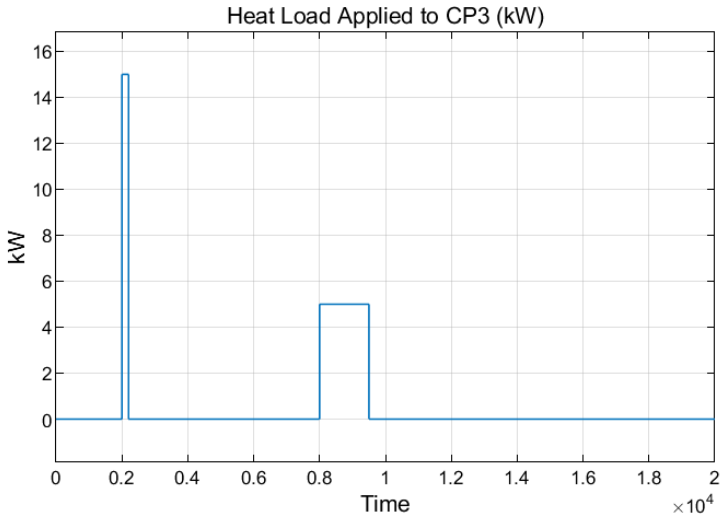

Figure 6.2: Heat Load 3 Profile.

6.1.1 Controller Goals

This system has two independent controllers acting on it. The schematic of the system with the controllers is shown in Fig. 6.3.

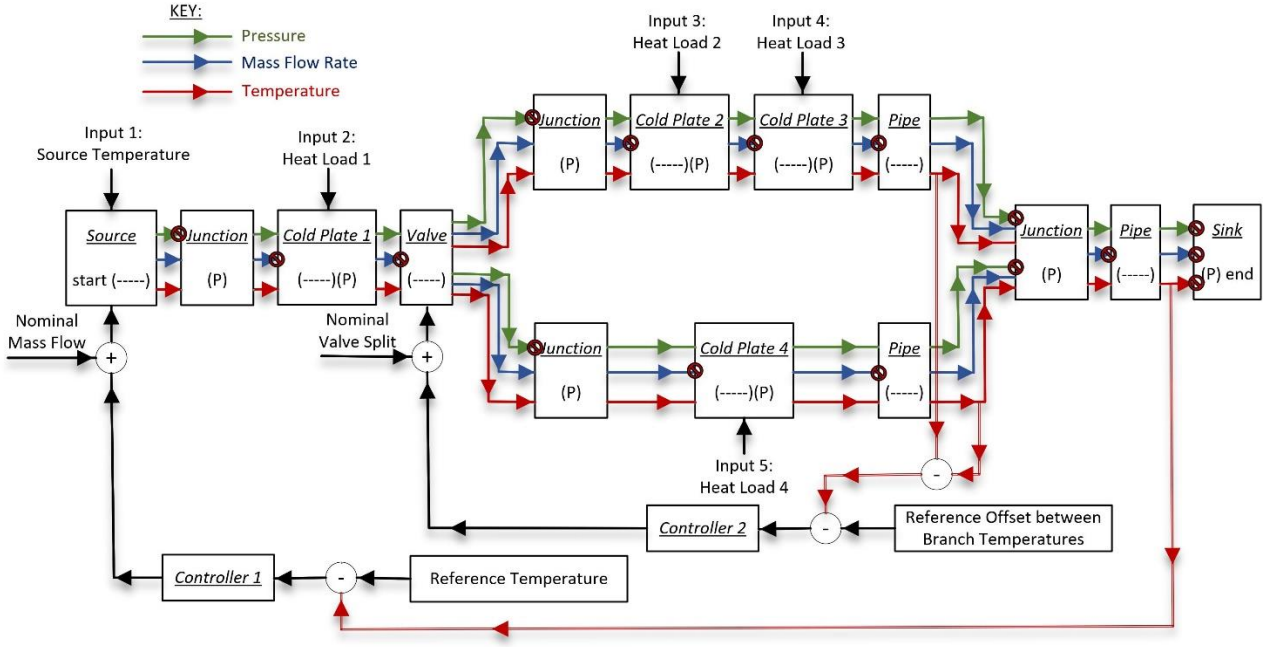

Figure 6.3: Schematic of Example 2 with Controllers 1 and 2.

The goal of Controller 1 is to keep the temperature of fluid exiting the system close to a reference temperature of 20.5 °C. A proportional (P) controller was chosen for the co-design analysis in this chapter, with a proportional gain of 3 as the nominal case. A proportional-integral (PI) controller does a better job of minimizing error; however, it makes the effect of perturbing different parameters less apparent. Therefore, since the purpose of this thesis is to discuss how sensitivity analysis techniques can be used in co-design efforts, a simple P controller was selected instead.

The goal of Controller 2 is to keep the offset, or difference, between the top branch and bottom branch temperatures at a reference value of 3 °C. A P controller was also chosen for Controller 2 for the same reasons as Controller 1. The nominal P gain for Controller 2 is set to 3 as well.

6.1.2 Nominal Case

The nominal case with the nominal controller gains is shown in Fig. 6.4.

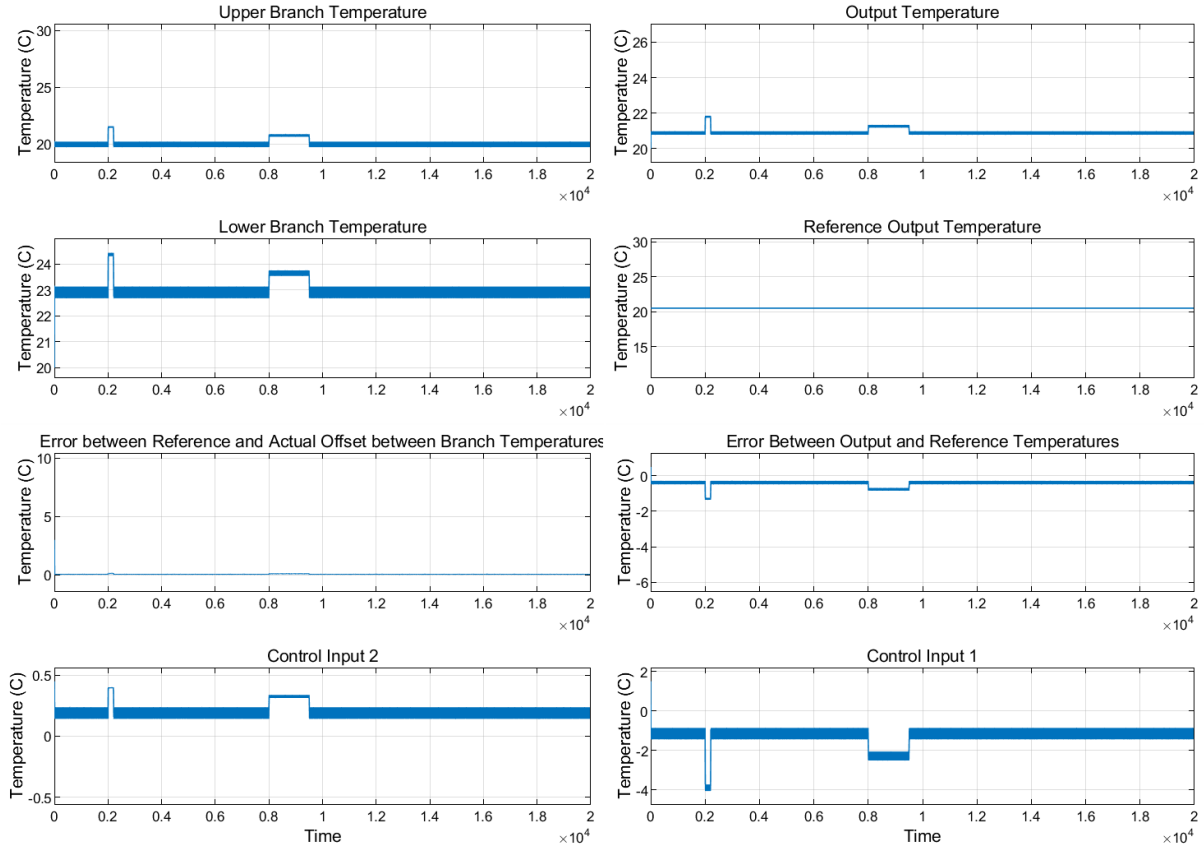

Figure 6.4: Nominal Plant and Controller Performance.

This can be contrasted from the performance of the system with Heat Load 3 acting on it, but without either controller to manage the temperatures, shown in Fig. 6.5.

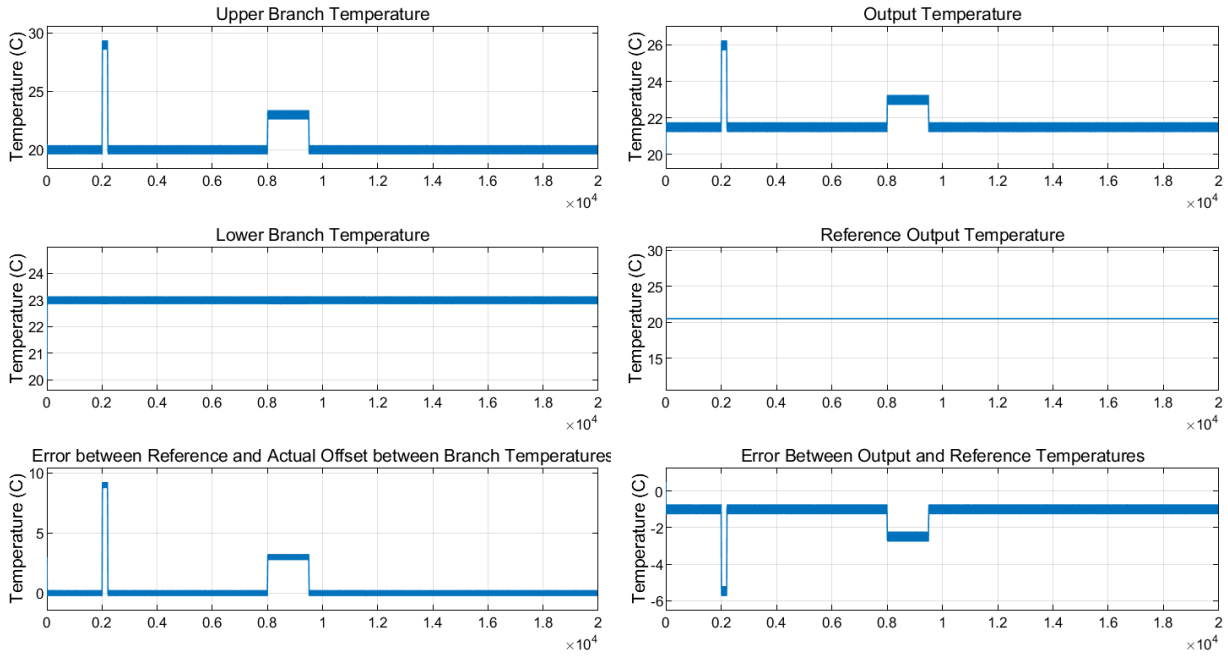

Figure 6.5: Nominal Plant without Controllers.

Notice how the addition of the controllers lowered the peak of the output temperature in the upper right plot from over 26 °C to just under 22 °C at 2000 seconds. This is also communicated in how the magnitude of the peak error between the output temperature and the reference decreased from 5.5 °C, to just over 1 °C. Similarly, Controller 2 worked to decrease the offset between the two branch temperatures drastically, from 9 °C over the reference offset to less than 0.14 °C at around 2000 seconds. The impact of Controller 2 adjusting the flow percentage between the two branches can be seen in how the lower branch increased in temperature when the mass flow rate through it decreased to direct more flow to the upper branch. This resulted in a cooling effect on the upper branch and a heating effect on the lower branch when compared to the nominal case without controllers.

6.2 Sensitivity Analysis of Nominal Case

Now that the nominal plant and controllers have been defined, the most influential parameters on those control inputs can be determined. The goal is to minimize the control effort

without sacrificing performance. Since the controllers are proportional controllers, the amplitudes of the control inputs will scale with the error values. And the amplitudes of the errors scale with the output temperature signals, since the error terms are composed of constant reference values subtracted by sinusoidal output temperature signals. Therefore, the parameters which affect the output temperature amplitudes the most will also be the parameters which affect the amplitude of the control input the most. The sensitivity analysis discussed in Chapters 4 and 5 can be utilized in finding the most influential parameters on the control input, since it is designed to determine the parameters the output temperature amplitude is most sensitive to.

Control Input 1 and the error for Controller 1 should scale with the output temperature: the fluid temperature of the junction, $T_{f,9}$, as shown in Fig. 6.6. Control Input 2 and the error for Controller 2 depend on the temperature at the ends of the upper and lower branches instead. These are the fluid temperatures of Cold Plate 3 and Cold Plate 4, written as $T_{f,5}$ and $T_{f,7}$ respectively. Therefore, the sensitivity analysis will need to be carried out with respect to three output signals: $T_{f,5}$, $T_{f,7}$, and $T_{f,9}$.

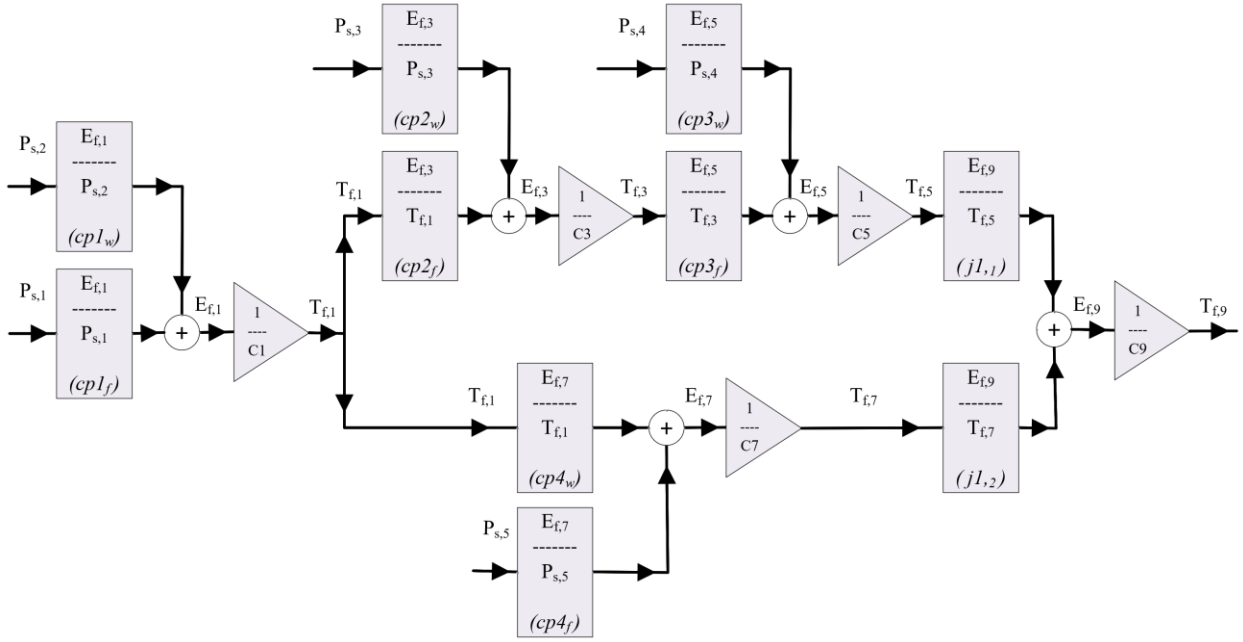

Figure 6.6: Block Diagram of Example 2.

The procedure to calculate the sensitivities of the output temperature signals has been covered in detail in Chapters 4 and 5 and can be performed again for this case study. The only

difference between the sensitivity analysis of $T_{f,9}$ under the Scenario 3 input signals done in Chapter 5 and in this chapter is the Heat Load 3 step function. Heat Load 3 maintains constant values for a vast majority of the simulation and is only transient when it instantaneously steps between different heat loads four times. Therefore, Heat Load 3 can be assumed to be at a constant value, just like it is in Scenario 3 from Chapter 5, and the sensitivity analysis of Scenario 3 can be assumed to be the same as the sensitivity of $T_{f,9}$ in this example as well.

6.2.1 Sensitivity Analysis for Each Output Temperature of Interest

6.2.1.1 Temperature of Fluid Exiting the System, $T_{f,9}$

The equations representing the output temperature, $T_{f,9}$, in terms of all the input signals are shown in Eq. (6.1) – (6.6). Each of the input-specific transfer functions represented in Eq. (6.2) – (6.6) is plotted as a bode plot in Fig. 6.7 below.

$$P_{s,1} * \frac{T_{f,9}}{P_{s,1}} + P_{s,2} * \frac{T_{f,9}}{P_{s,2}} + P_{s,3} * \frac{T_{f,9}}{P_{s,3}} + P_{s,4} * \frac{T_{f,9}}{P_{s,4}} + P_{s,5} * \frac{T_{f,9}}{P_{s,5}} = T_{f,9} \tag{6.1}$$

where

$$\frac{T_{f,9}}{P_{s,1}} = \frac{E_{f,1}}{P_{s,1}} * \frac{1}{C_1} * \left(\frac{E_{f,3}}{T_{f,1}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} + \frac{E_{f,7}}{T_{f,1}} * \frac{1}{C_7} * \frac{E_{f,9}}{T_{f,7}} \right) * \frac{1}{C_9} \tag{6.2}$$

$$\frac{T_{f,9}}{P_{s,2}} = \frac{E_{f,1}}{P_{s,2}} * \frac{1}{C_1} * \left(\frac{E_{f,3}}{T_{f,1}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} + \frac{E_{f,7}}{T_{f,1}} * \frac{1}{C_7} * \frac{E_{f,9}}{T_{f,7}} \right) * \frac{1}{C_9} \tag{6.3}$$

$$\frac{T_{f,9}}{P_{s,3}} = \frac{E_{f,3}}{P_{s,3}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} * \frac{1}{C_9} \tag{6.4}$$

$$\frac{T_{f,9}}{P_{s,4}} = \frac{E_{f,5}}{P_{s,4}} * \frac{1}{C_5} * \frac{E_{f,9}}{T_{f,5}} * \frac{1}{C_9} \tag{6.5}$$

$$\frac{T_{f,9}}{P_{s,5}} = \frac{E_{f,7}}{P_{s,5}} * \frac{1}{C_7} * \frac{E_{f,9}}{T_{f,7}} * \frac{1}{C_9} \tag{6.6}$$

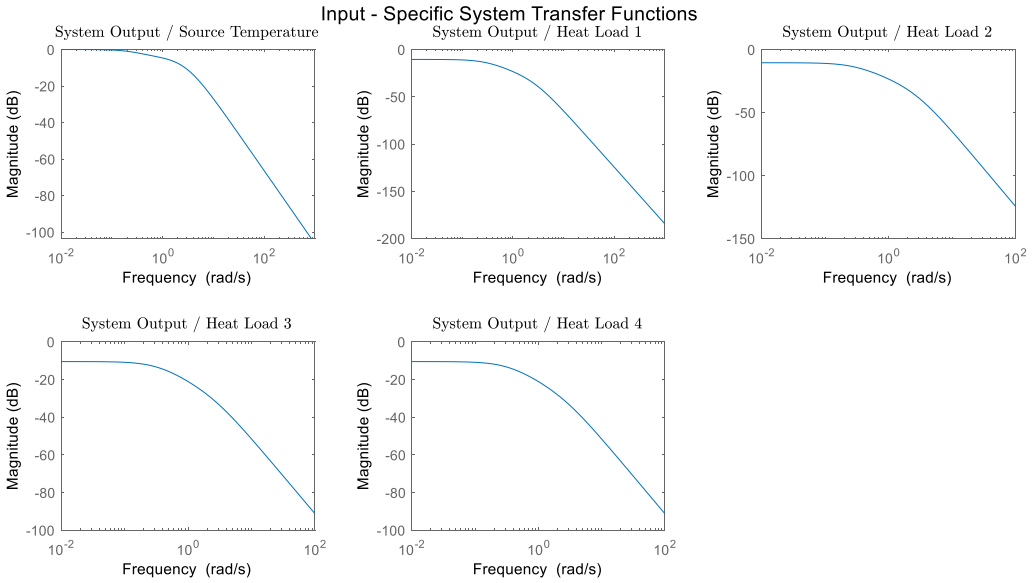

Figure 6.7: Input-Specific Transfer Functions for $T_{f,9}$.

The partial derivatives for each of the input-specific transfer functions in Fig. 6.7 with respect to the capacitance parameters are shown in Fig. 6.8.

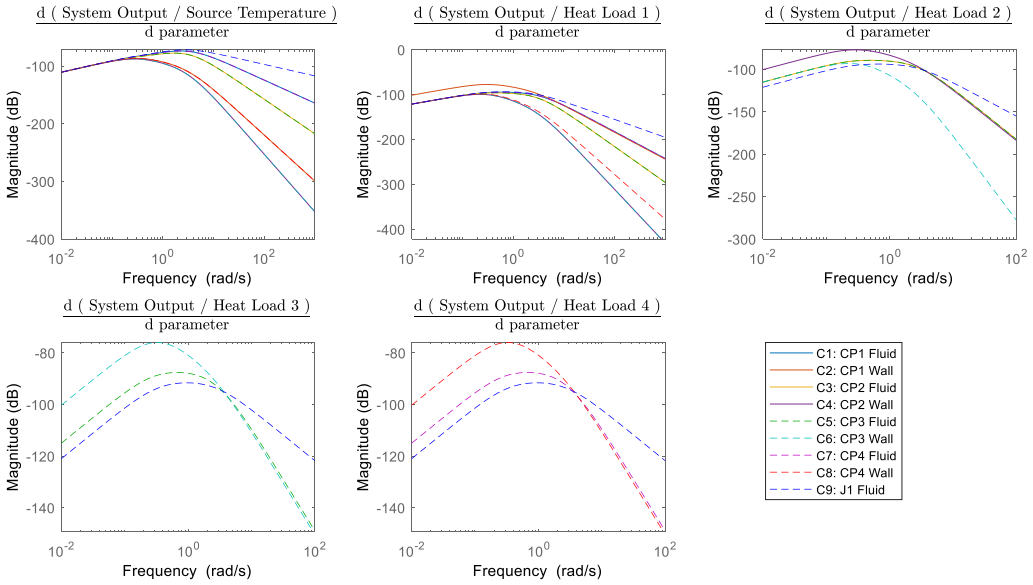

Figure 6.8: Sensitivity Bode Plots for $T_{f,9}$.

When the sensitivity bode plots are each multiplied by the amplitudes of their respective input signals and added together, they yield the sensitivity plot in Fig. 6.9.

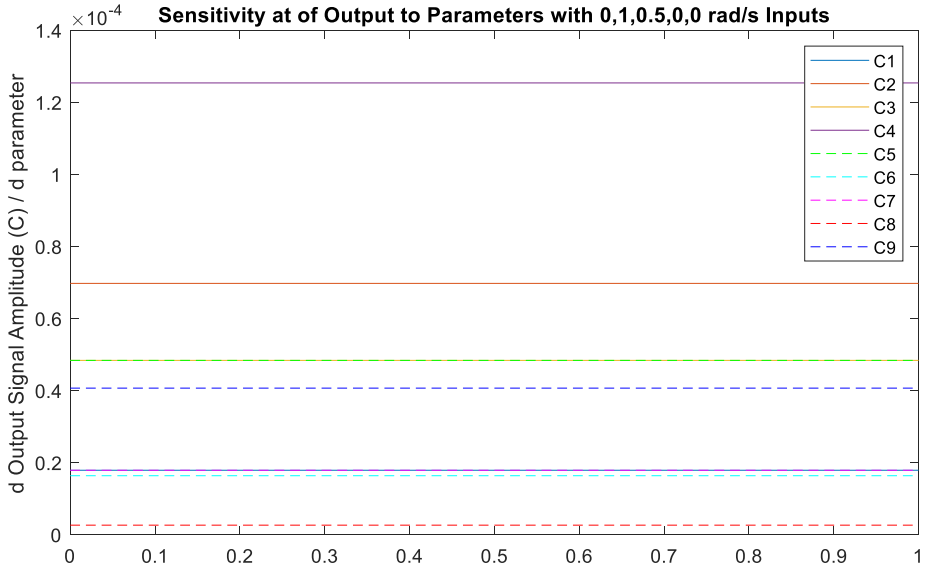

Figure 6.9: Sensitivity Plot for $T_{f,9}$.

The sensitivity plot in Fig. 6.9 conveys that parameter C_4, the wall mass of Cold Plate 1, is the most influential parameter to the temperature of the fluid exiting the system, $T_{f,9}$. Since Controller 1 controls the mass flow rate into the system to drive $T_{f,9}$ to a reference value, the parameters which affect $T_{f,9}$ are strongly correlated to the error between $T_{f,9}$ and the reference value, as well as the control input necessary to minimize that error.

6.2.1.2 Temperature of Fluid Exiting the Top Branch, $T_{f,5}$

The goal of the second controller is to maintain the difference between temperatures of the fluids exiting the two branches, $T_{f,5}$ and $T_{f,7}$, at a reference offset. This ensures that one branch will not get extremely cold to compensate for the other branch overheating when trying to keep the output temperature down. The goal is to keep both branches at mild temperatures and controlling the offset between the two branches helps achieve this. Thus, the temperatures in the two branches have a huge effect on the error between their temperature difference and the specified offset, as well as on the control input for Controller 2. The equations which represent the temperature of the fluid as it exits the top branch, $T_{f,5}$, can be derived from the schematic in Fig. 6.6 and are shown in Eq. (6.7) – (6.12).

$$P_{s,1} * \frac{T_{f,5}}{P_{s,1}} + P_{s,2} * \frac{T_{f,5}}{P_{s,2}} + P_{s,3} * \frac{T_{f,5}}{P_{s,3}} + P_{s,4} * \frac{T_{f,5}}{P_{s,4}} + P_{s,5} * \frac{T_{f,5}}{P_{s,5}} = T_{f,5} \tag{6.7}$$

where

$$\frac{T_{f,5}}{P_{s,1}} = \frac{E_{f,1}}{P_{s,1}} * \frac{1}{C_1} * \frac{E_{f,3}}{T_{f,1}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} \tag{6.8}$$

$$\frac{T_{f,5}}{P_{s,2}} = \frac{E_{f,1}}{P_{s,2}} * \frac{1}{C_1} * \frac{E_{f,3}}{T_{f,1}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} \tag{6.9}$$

$$\frac{T_{f,5}}{P_{s,3}} = \frac{E_{f,3}}{P_{s,3}} * \frac{1}{C_3} * \frac{E_{f,5}}{T_{f,3}} * \frac{1}{C_5} \tag{6.10}$$

$$\frac{T_{f,5}}{P_{s,4}} = \frac{E_{f,5}}{P_{s,4}} * \frac{1}{C_5} \tag{6.11}$$

$$\frac{T_{f,9}}{P_{s,5}} = 0 \tag{6.12}$$

The graphical representations of each of the individual transfer functions in Eq. (6.8) – (6.12) are shown in Fig. 6.10 below. Notice that the transfer function which relates the output amplitude of the top branch temperature to the fifth input, Heat Load 4, does not exist. This is because Heat Load 4 is applied to the Cold Plate 4 component, which has no effect on the top branch temperature, since they are in parallel branches and do not meet up until the junction component, which is after the exit temperature of branch one is recorded.

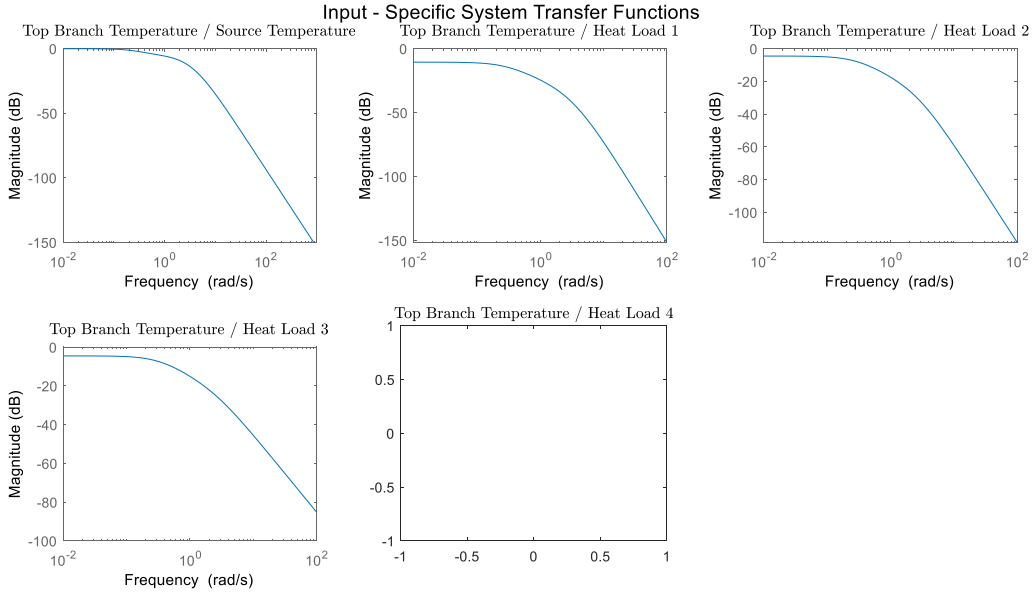

Figure 6.10: Input-Specific Transfer Functions for $T_{f,5}$.

The partial derivatives of the transfer functions in Fig. 6.10 are shown in Fig. 6.11. These sensitivity bode plots outline how the capacitance parameters effect each of the input-specific transfer functions when perturbed.

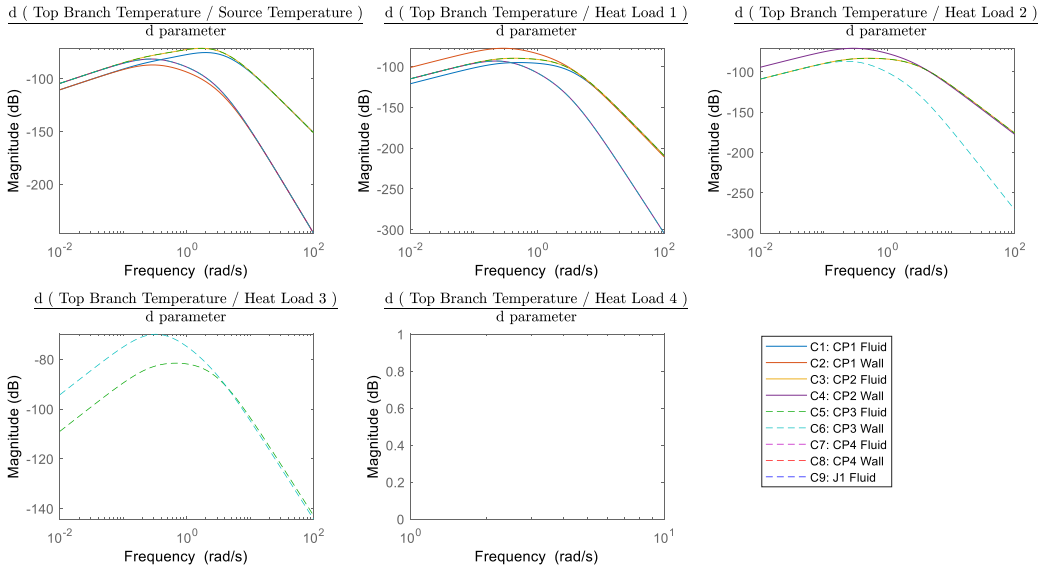

Figure 6.11: Sensitivity Bode Plots for $T_{f,5}$.

The sensitivity bode plots are then multiplied by the amplitudes of their input signals and added together to yield the sensitivity plot in Fig. 6.12. The plot below demonstrates how the top branch exit temperature, $T_{f,5}$, is most sensitive to parameter C_4, the capacitance of the wall in Cold Plate 2. The second-most influential parameters are C_3 and C_5, the fluid capacitances in Cold Plate 2 and 3, followed by C_2, the capacitance of the wall in Cold Plate 1.

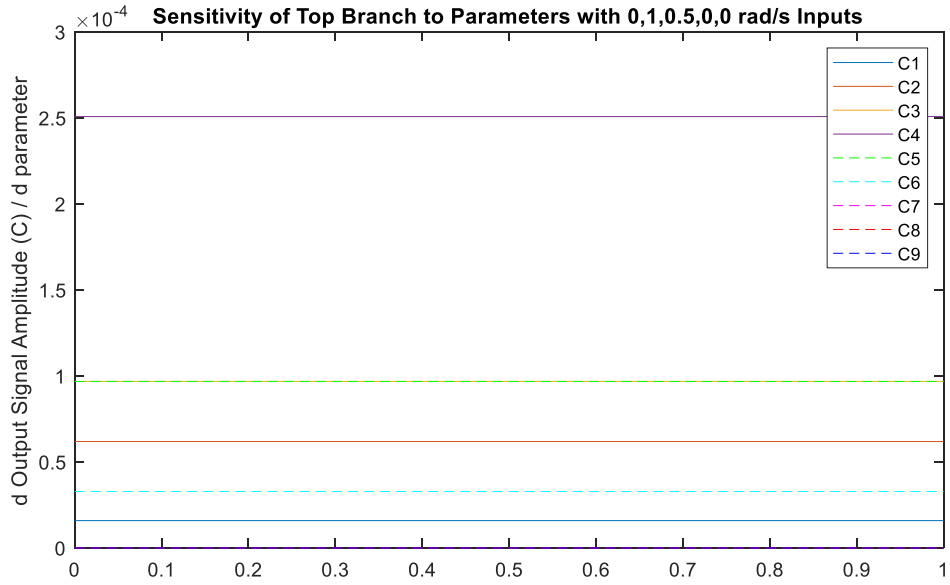

Figure 6.12: Sensitivity Plot for $T_{f,5}$.

6.2.1.3 Temperature of Fluid Exiting the Bottom Branch, $T_{f,7}$

The control input of Controller 2, which regulates the offset between the exit temperatures of the top and bottom branch, is also coupled to the temperature of the bottom branch, $T_{f,7}$. The transfer functions which represent $T_{f,7}$ in terms of each of the input signals are shown in Eq. (6.14) – (6.18) and combine to yield Eq. (6.13).

$$P_{s,1} * \frac{T_{f,7}}{P_{s,1}} + P_{s,2} * \frac{T_{f,7}}{P_{s,2}} + P_{s,3} * \frac{T_{f,7}}{P_{s,3}} + P_{s,4} * \frac{T_{f,7}}{P_{s,4}} + P_{s,5} * \frac{T_{f,7}}{P_{s,5}} = T_{f,7} \tag{6.13}$$

where

$$\frac{T_{f,7}}{P_{s,1}} = \frac{E_{f,1}}{P_{s,1}} * \frac{1}{C_1} * \left(\frac{E_{f,7}}{T_{f,1}} * \frac{1}{C_7}\right) \tag{6.14}$$

$$\frac{T_{f,7}}{P_{s,2}} = \frac{E_{f,1}}{P_{s,2}} * \frac{1}{C_1} * \left(\frac{E_{f,7}}{T_{f,1}} * \frac{1}{C_7}\right) \tag{6.15}$$

$$\frac{T_{f,7}}{P_{s,3}} = 0 \tag{6.16}$$

$$\frac{T_{f,7}}{P_{s,4}} = 0 \tag{6.17}$$

$$\frac{T_{f,7}}{P_{s,5}} = \frac{E_{f,7}}{P_{s,5}} * \frac{1}{C_7} \tag{6.18}$$

The transfer functions in Eq. (6.14) – (6.18) are shown as bode plots in Fig. 6.13, with one plot for how the bottom branch temperature is affected by each system input. Notice how the transfer functions for Inputs 3 and 4, the heat loads applied to Cold Plate 2 and 3, are non-existent. This is because Cold Plates 2 and 3 are in the top branch and have no effect on the exit fluid temperature of the bottom branch.

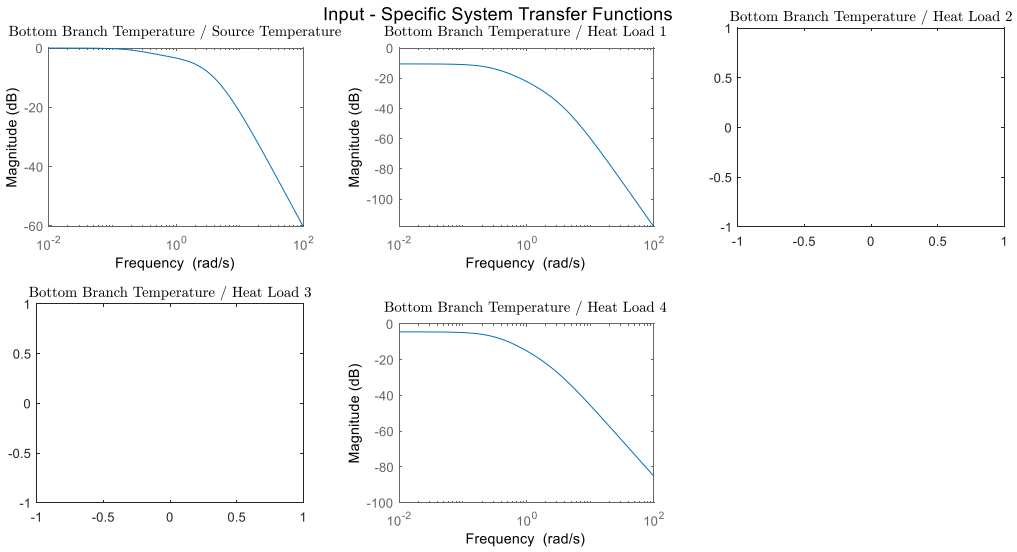

Figure 6.13: Input-Specific Transfer Functions for $T_{f,7}$.

The partial derivatives of the transfer functions shown in Fig. 6.13 with respect to the capacitance parameters are shown in Fig. 6.14.

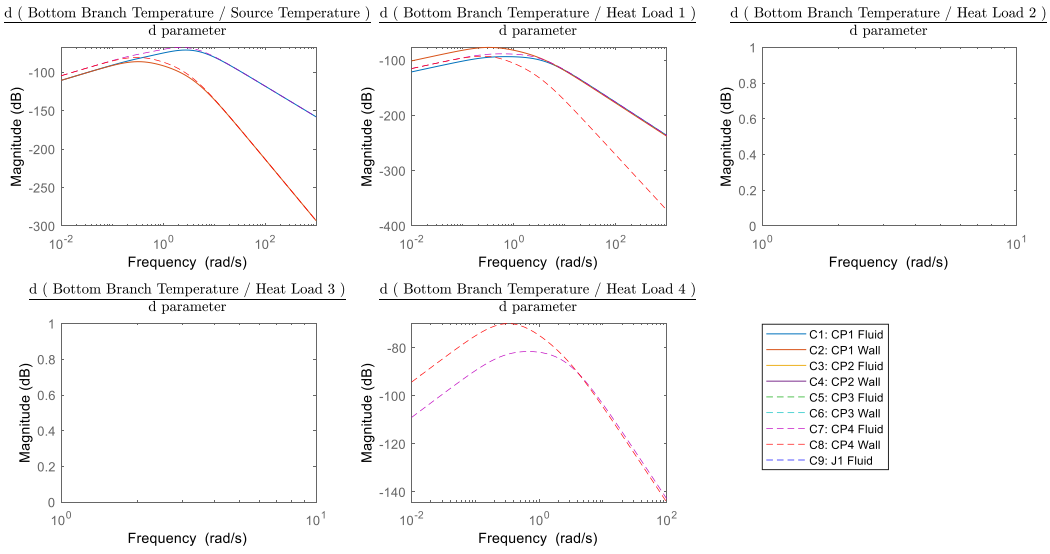

Figure 6.14: Sensitivity Bode Plot for $T_{f,7}$.

Each of the sensitivity bode plots in Fig. 6.14 can be multiplied by their respective input signal amplitudes and summed to yield the sensitivity plot in Fig. 6.15, which ranks the cumulative effect each capacitance parameter has on the bottom branch exit temperature, $T_{f,7}$.

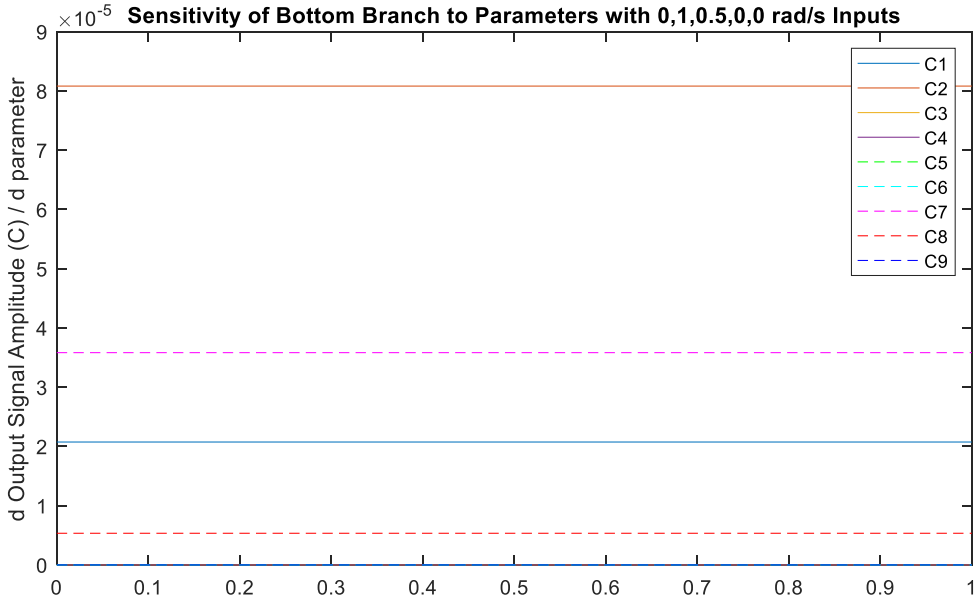

Figure 6.15: Sensitivity Plot for $T_{f,7}$.

Fig. 6.15, shows the strongest influence on $T_{f,7}$ is parameter C_2, the capacitance of the wall in Cold Plate 1. Comparitively, C_2 is the fourth most influential parameter for the amplitude of $T_{f,5}$. The top three parameters which $T_{f,5}$ is most sensitive to are C_4, C_3, and C_5, none of which even appear in the sensitivity plot for $T_{f,7}$. This is because these parameters represent components in the top branch and $T_{f,7}$ is the exit temperature for the bottom branch.

To identify which parameters are most influential on the error and control input for Controller 2, the sensitivity plots for $T_{f,5}$ and $T_{f,7}$ need to be assessed together. The magnitudes of the effects that parameters C_4, C_3, and C_5 have on the top branch temperature are greater than any of the parameters' effects on the bottom branch temperature. Therefore, when identifying the top four parameters' combined effect on the branch exit temperatures, more weight is given to C_4, C_3, and C_5. Comparing across Fig. 6.12 and Fig. 6.15, the most influential parameters on Controller 2 are C_4, C_3 and C_5, and C_2, in that order. These are the same top parameters which Controller 1's control input and error are most sensitive to. Therefore, it is safe to say these four parameters

should be the ones optimized to have the greatest effect in lowering the control input for both controllers.

6.2.2 Numerical Verification of Parameter Influence on Controlled Plant

This next section walks through the numerical verification of the sensitivity analyses when applied to the controlled system. Note that because the controllers are now applied to the system, they will change the mass flow rates into the system and the percentage split, depending on the applied heat loads at any moment. For example, previously parameters pertaining to Cold Plates 2 and 3, located in the top branch, had no effect on the temperature in the bottom branch. Now, since Controller 2 is working to keep the temperatures in both branches close together, when the heat loads are applied to Cold Plates 2 and 3, the flow is diverted from the bottom branch to the top branch to lower the temperature. So when C_4, the capacitance of Cold Plate 3, is increased, the rate of heat transfer to the top branch temperature is lessened, allowing more fluid to stay in the bottom branch, thus not causing the bottom branch temperature to rise as much. This demonstrates how the bottom branch temperature is dependent to parameters in the top branch when the controllers are applied, even though that correlation did not show up in the sensitivity analysis.

6.2.2.1 Temperature Signal Amplitudes

The following three figures illustrate how each of the temperatures in the system are most sensitive to parameters C_2, C_3, C_4, and C_5.

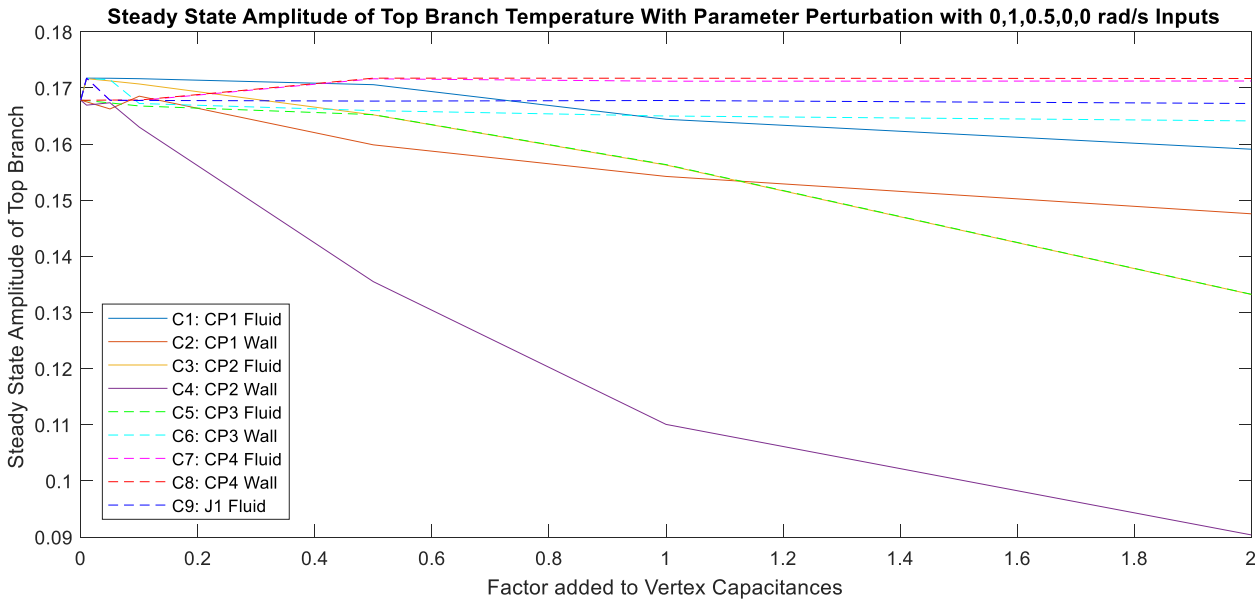

Figure 6.16: Amplitude of $T_{f,5}$ when Each Parameter is Perturbed Individually.

Figure 6.16 demonstrates a clear sensitivity of the top branch temperature amplitude to parameter C_4, followed by sensitivities to parameters C_2, C_3, and C_5 as predicted in Fig. 6.12.

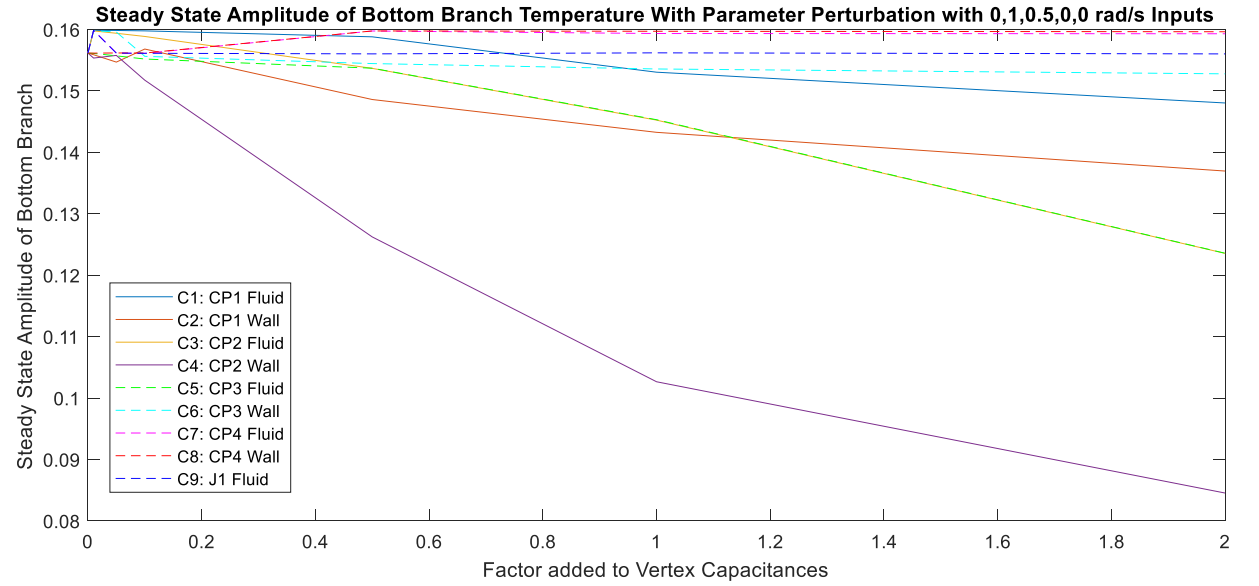

Figure 6.17: Amplitude of $T_{f,7}$ when Each Parameter is Perturbed Individually.

Although Fig. 6.17 does not directly mirror the sensitivity plot for $T_{f,7}$ in Fig. 6.15, due to the presence of Controller 2, it matches the assumption made that Controller 2 and the related temperatures would be most sensitive to parameter C_4, followed by parameters C_2, C_3, and C_5.

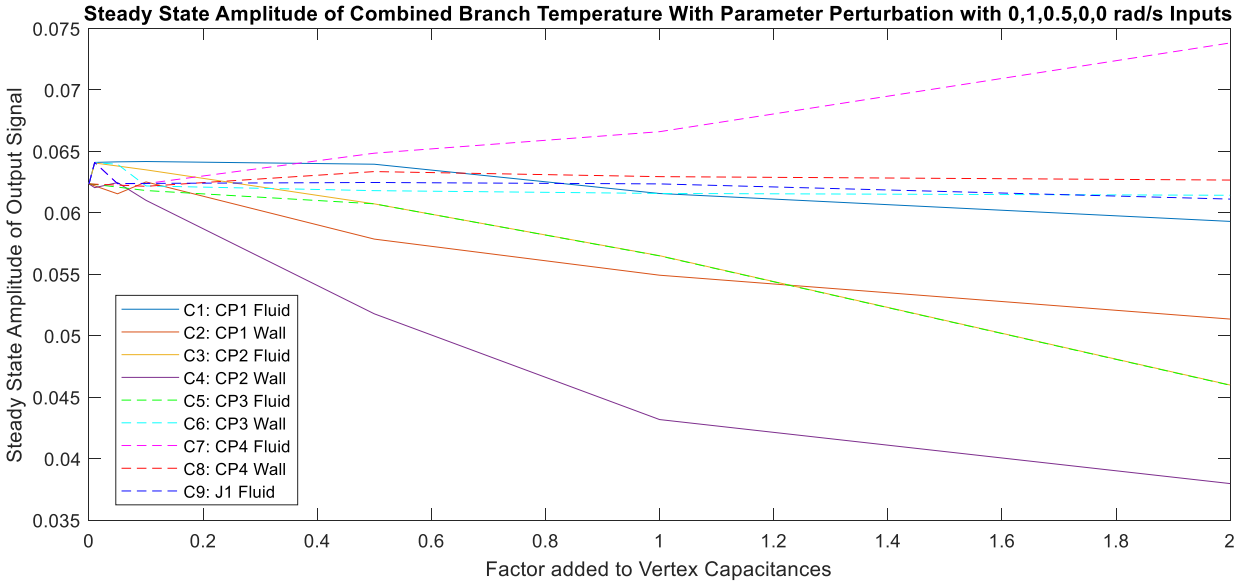

Figure 6.18: Amplitude of $T_{f,9}$ when Each Parameter is Perturbed Individually.

Fig. 6.18 matches the prediction made in the sensitivity plot for $T_{f,9}$ in Fig. 6.9 for the top four parameters, namely C_4, C_2, C_3, and C_5. Parameter C_7 also appears to increase in influence on the output temperature between perturbation values of 1 and 2. However, it is the smaller perturbations, closer to the nominal values where the model was linearized about, that are more reliable. Additionally, parameters C_4, C_2, C_3, and C_5 produce the desired effect of minimizing the steady state amplitude, whereas C_7 has the opposite effect.

6.2.2.2 Control Error Amplitudes

The goal of this co-design exercise is to optimize the plant and adjust the controller gains to minimize the control input. However, since the controller gains directly affect the control input, it might appear as though setting very small controller gains is the optimal solution simply because it will lead to lower control inputs. But just because the control input is artificially lower, does not mean the controller is more successful in minimizing the control error. Therefore, it is also important to consider the parameter influence on the control error, or the difference between the

reference temperature and the actual temperature. Fig. 6.19 and 6.20 show the amplitude of the error terms for Controllers 1 and 2, respectively.

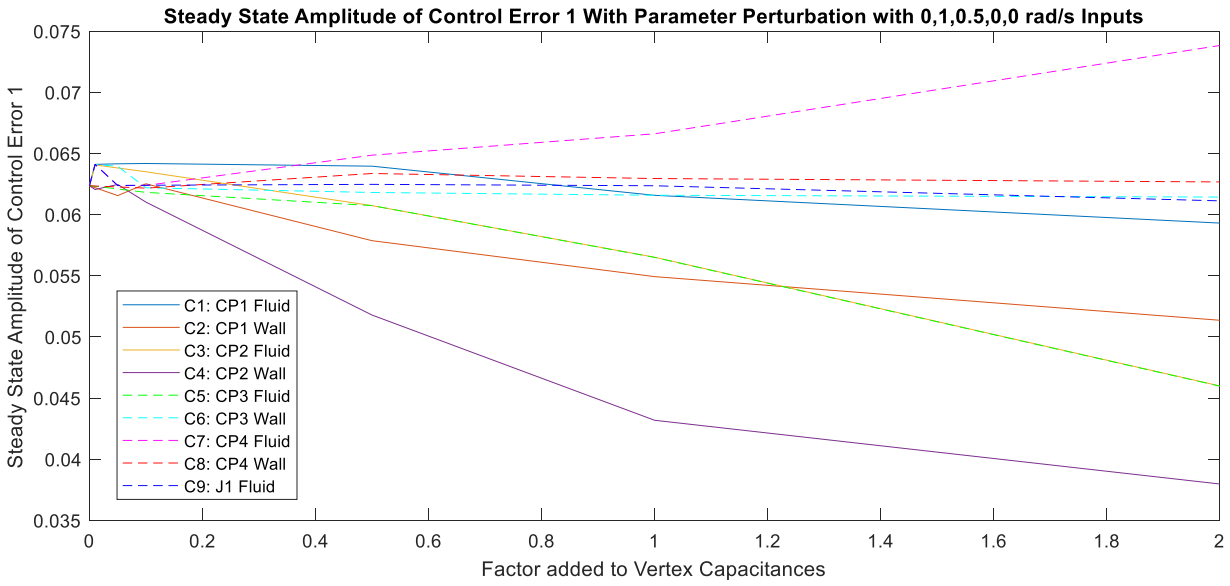

Figure 6.19: Amplitude of Control Error 1 with Parameter Perturbations.

The pattern of the most influential parameters being C_4, C_2, C_3, and C_5 holds, as is expected since the control error amplitude scales with the temperature signal amplitudes. Control Error 1 is calculated by subtracting the reference temperature for the fluid exiting the system by the actual fluid temperature. Therefore, since Control Error 1 equals a constant subtracted by a sinusoidal term, the control error amplitude will exactly equal the amplitude of the temperature. This is shown by Fig. 6.18 and Fig. 6.19 being identical.

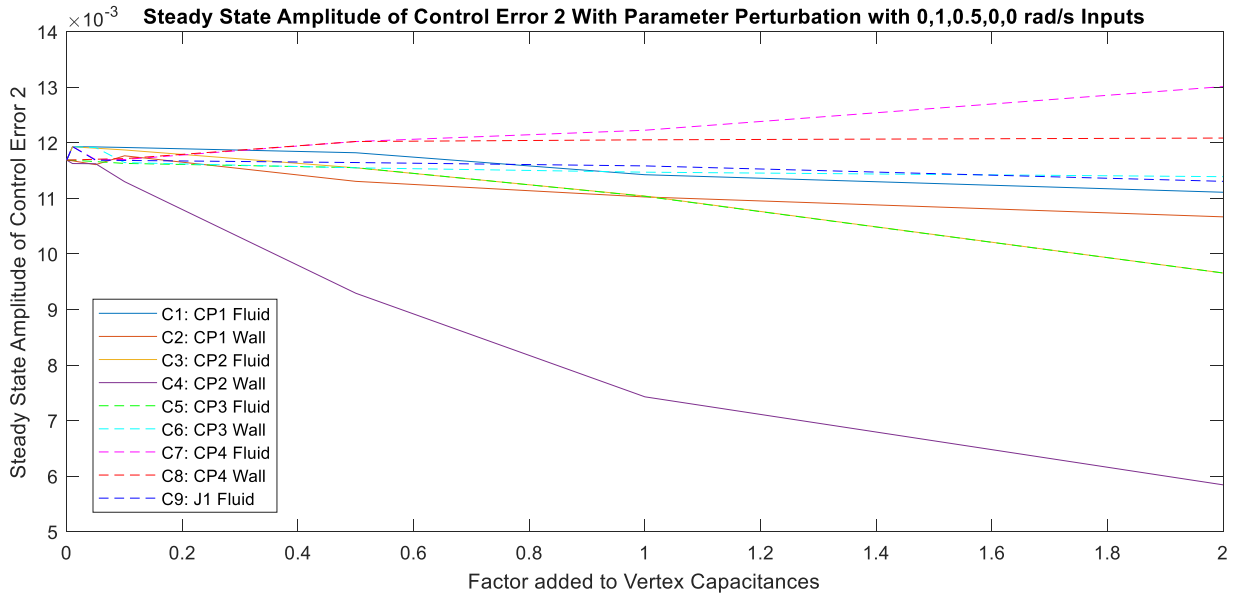

Figure 6.20: Amplitude of Control Error 2 with Parameter Perturbations.

Control Error 2 is calculated by subtracting the reference offset by the actual offset. The offset between the two branches is defined as the difference between their temperature signals. These two branch temperatures have different amplitudes, but both are shown to be dependent on the same top four parameters in Fig. 6.16 and Fig. 6.17. Therefore, the amplitude of their difference will also be dependent on those same four parameters. This is seen in Fig. 6.20, where the top four parameters which result in a lower control error, are parameters C_4, C_2, C_3, and C_5.

6.2.2.3 Control Input Amplitudes

Since the main goal of the co-design methodology is to adjust parameters to achieve the lowest control input amplitudes while not sacrificing the controller performance, the sensitivities of the control input amplitudes to the parameters are very important. These are plotted in Fig. 6.21 and Fig 6.22.

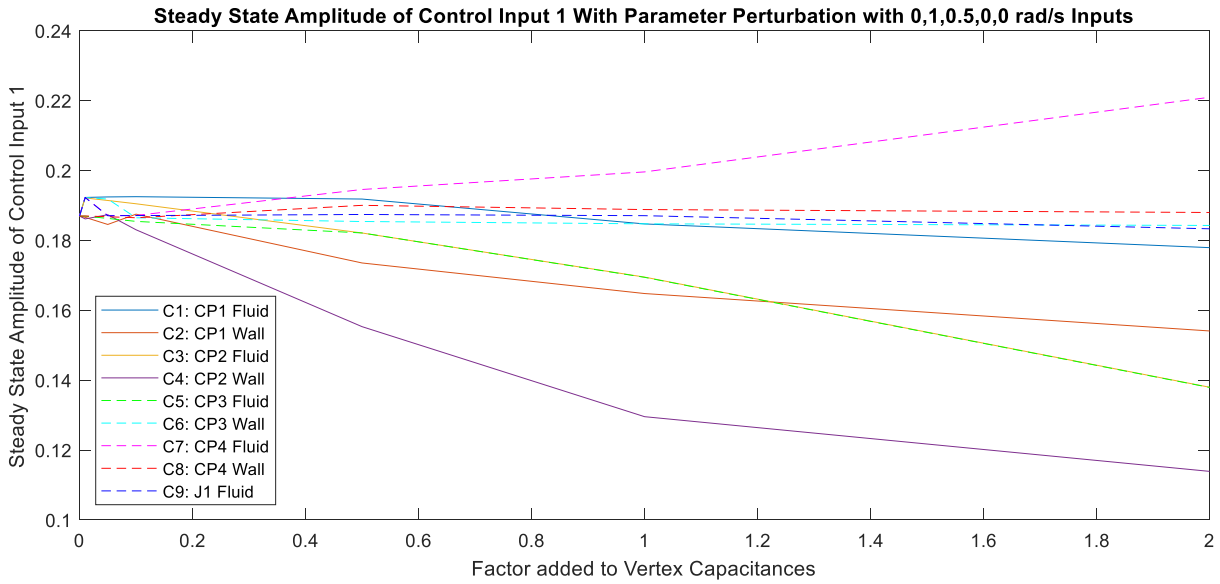

Figure 6.21: Amplitude of Control Input 1 with Parameter Perturbations.

Fig. 6.21 shows how the Control Input 1 amplitude varies with the parameter perturbations. Since the controller used in this study is a proportional controller, the control input scales directly from the control error. For this reason, Fig. 6.21 shares the same shape as Fig. 6.19; it has simply been multiplied by the controller gain. Although these plots convey the same information for a constant controller gain, once the controller gains are adjusted in the tuning process, it will be important to save both sets of information. This will allow the controller gain and parameter perturbation combinations to be compared more easily by searching for the combinations which have the lowest control error and lowest control inputs.

Controller 1 adjusts the mass flow rate into the system, allowing more flow to come through when the output temperature is higher than the reference, having a cooling effect. Similarly, less flow will be let into the system if the output temperature is lower than the reference, allowing the heat loads to be more effective in raising the fluid temperature. A saturation block keeps the mass flow rate going into the system from ever being negative since the model requires positive mass flow rates to function properly. The control input for Controller 1 relates to pumping power. By minimizing the amplitude of the required control input, energy can be saved that otherwise would have been necessary to ramp the pump up and down to higher speeds.

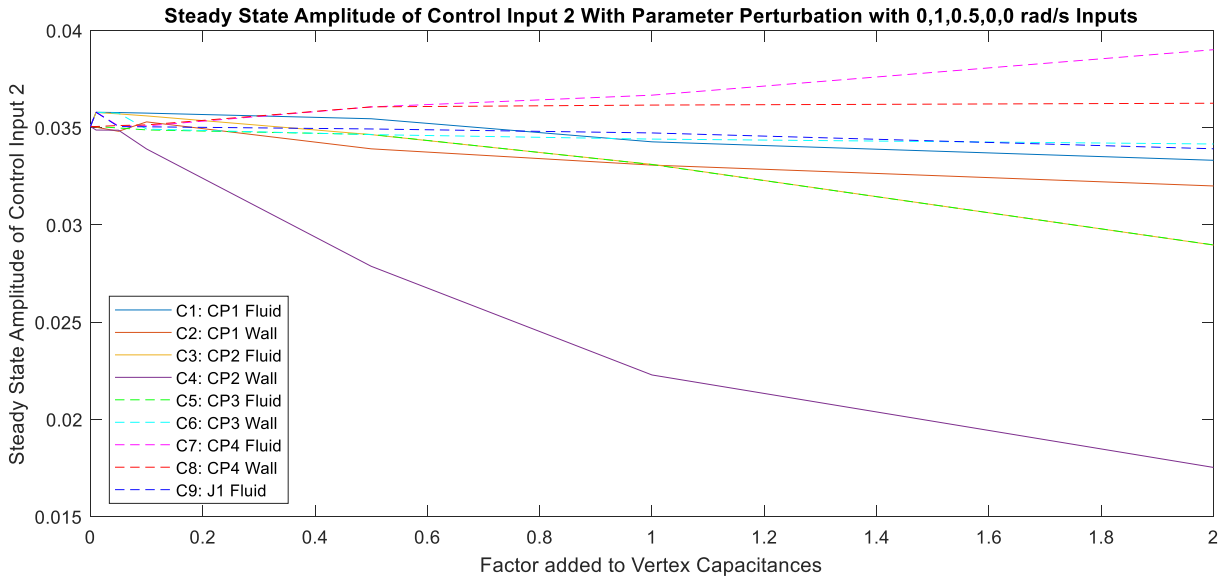

Figure 6.22: Amplitude of Control Input 2 with Parameter Perturbations.

Fig. 6.22 shows how the Control Input 2 amplitude varies with parameter perturbations. Notice how the Control Input 2 scales directly by a factor of three, the nominal controller gain, from the control error plot in Fig. 6.20. Controller 2 adjusts the valve percentage of flow which is directed to the top branch. A saturation block is used for this signal as well to ensure the valve percentage is always within the bounds of zero and one, where zero means 0% of the flow entering the valve will go to the top branch and one means 100% of the flow will be directed to the top branch. It is a good idea to saturate this control input signal so that it never equals zero or one, which would send all the flow to a single branch. The temperature in the other branch would then increase rapidly as a result since it would have no flow coming through in the presence of constant heat loads. The control input for Controller 2 is the valve adjustment. Since this requires significantly less work to physically adjust when compared to Controller 1, Control Input 1 will be the focus instead of Control Input 2 moving forward. The incentive to lower the control input for Controller 1 from an energy standpoint is much higher, which is why it will be the focus of this co-design case-study in Section 6.3.

6.3 Plant and Controller Co-design

6.3.1 Controller Gain and Plant Parameter Sweep

This final section walks through a parameter and controller gain sweep to find the optimal combinations of controller gains and parameter values. Since both the parameter values and controller gains affect the controller errors and control inputs, there is value in optimizing these simultaneously. The top four parameters to be optimized were selected in Section 6.2, namely C_2, C_3, C_4, and C_5. These will be referred to as Parameters 1, 2, 3, and 4, respectively. For each run of the simulation, either one of these parameter capacitances will be perturbed by a value of one, with all other parameters remaining at their nominal values. Or, one of the parameters will be perturbed by one and another parameter will be perturbed by 0.5, with all other parameters remaining at their nominal values. These configuration combinations are illustrated in Fig. 6.23 below for the nominal controller gains of 3. Notice how the rows signify the larger parameter perturbation of 1 and the columns signify the smaller parameter perturbation of 0.5.

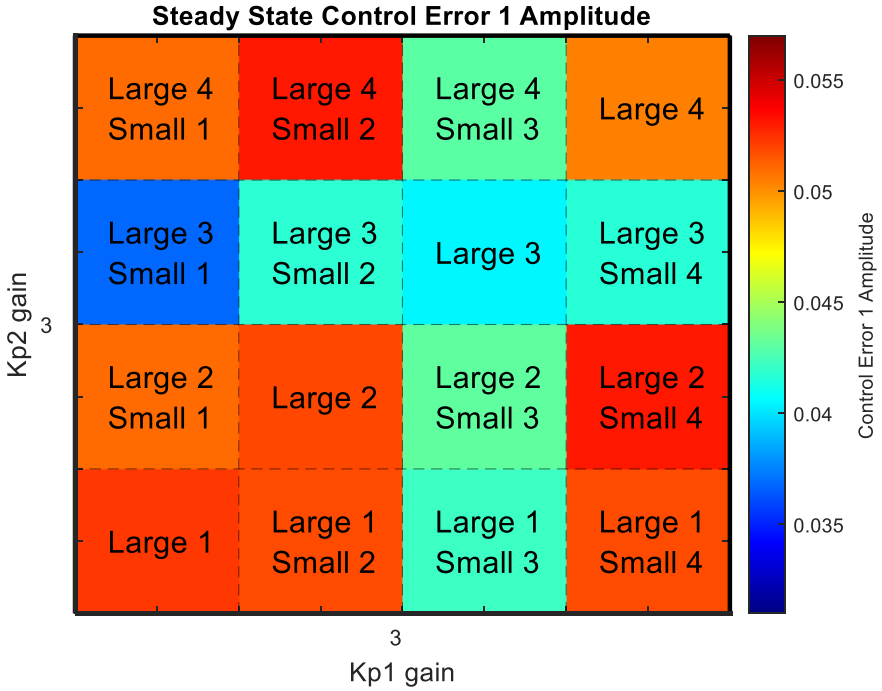

Figure 6.23: Nominal Controller Gains Parameter Sweep.

For the nominal case, the dark blue square indicates that a large perturbation of Parameter 3, C_4, and a small perturbation of Parameter 1, C_2, result in the lowest control error for Controller 1. Physcially, this means that when the heat loads on Cold Plates 1 and 2 are sinusoidal, the best way to filter the resulting sinusoidal signal of the fluid temperature is to increase the masses of the walls in Cold Plates 1 and 2. This will minimize the error between the fluid temperature and a constant reference value, resulting in a lower control input requirement to minimize the fluid temperature signal's amplitude. Fig. 6.23 also indicates that the control error is significantly more sensitive to Parameter 3 than any other parameter. The row where Parameter 3 has the large perturbation contains the lowest control error values of the entire plot, highlighted by containing the most blue and green shades. The column where Parameter 3 has the small perturbation also has the lowest control error values out of all the other columns, as evidenced by the entire column being a teal color. The sensitivity plot in Fig. 6.9 successfully predicted this behavior. It determined that C_4, followed by C_2, was the most influential on the fluid temperature as it exits the system, which is directly related to the control error of Controller 1.

Fig. 6.24 demonstrates the parameter sweep outlined in Fig. 6.23 for combinations of controller gains ranging from 2.5 to 4.5 as the proportional gain terms of Controller 1 and Controller 2. The Kp gains for Controller 1 are marked by the columns, with the simulations in the first 4 columns, or first block column, having a proportional gain of 2.5 for Controller 1, and the final block column, or columns 17-20, having a proportional gain of 4.5. Similarly, the Kp gains for Controller 2 are marked by the rows, with the gain increasing by 0.5 for each new block row, or set of 4 square rows.

The large parameter perturbations are demarcated by bold numbers along the left side of the plot, and the small perturbations are demarcated by the non-bolded text along the base of the plot. The location of any square reveals the parameters which were perturbed for that simulation. For example, every square in the upper left corner of the 4x4 controller gain blocks represents a large change in parameter four and a small change in parameter one. The row will be labelled as a bold four at the far left of the plot and the column it is in will be labelled by a non-bold 1 at the base of the plot.

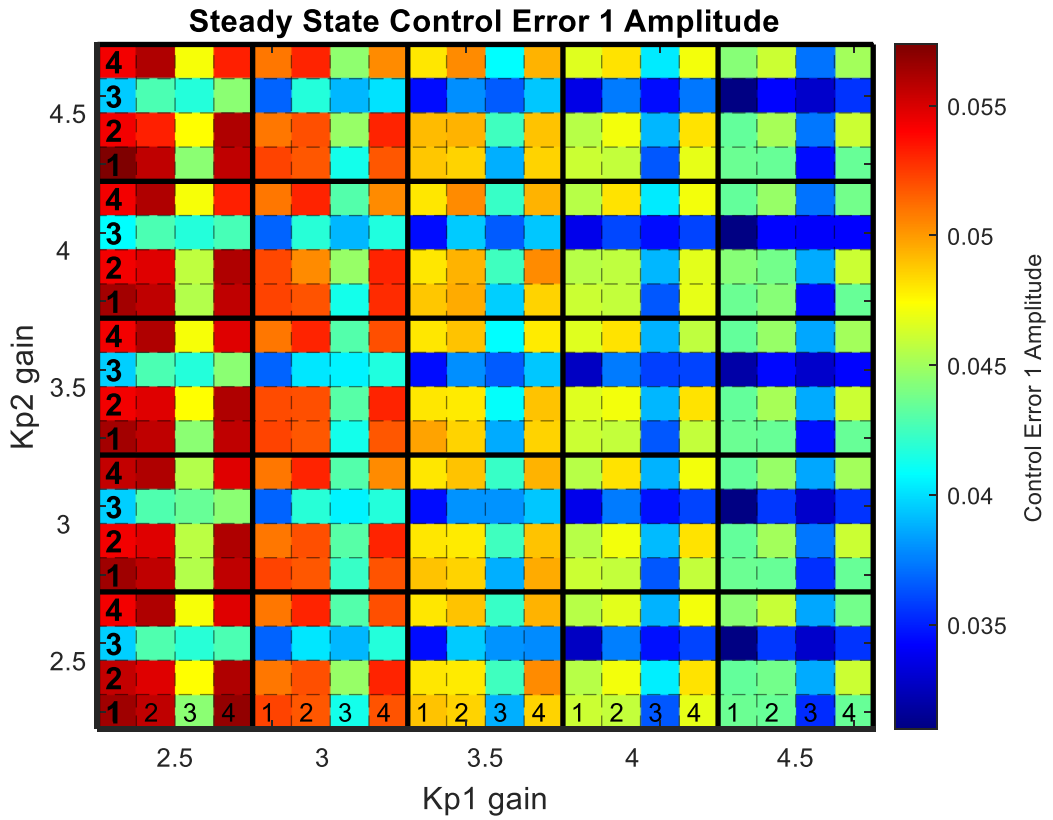

Figure 6.24: Controller Gain and Parameter Sweep for Control Error 1.

Notice how as the Kp gains for Controller 1 increase, moving from the left columns to the right columns, the control error decreases. This makes intuitive sense: when the controller gains are higher, the controller has more ability to close the gap between the reference temperature and the actual temperature by increasing the mass flow rate into the system at a higher rate. This trend is highlighted with an arrow in Fig. 6.25.

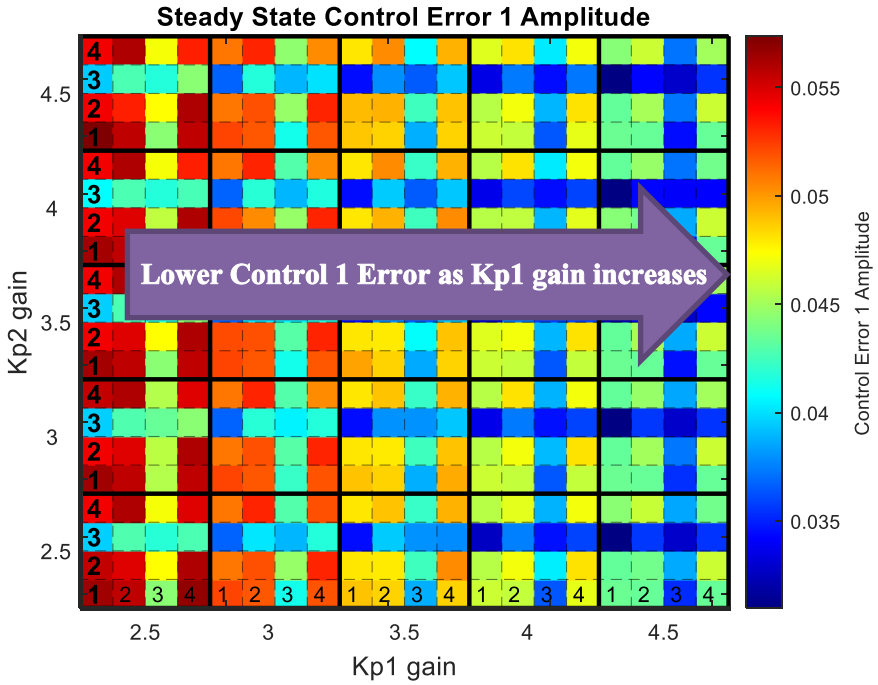

Figure 6.25: Control Error 1 as Kp1 Increases Trend.

Fig. 6.26 shows how Control Error 2 changes with the adjustments in controller gains and parameter values. Notice that this time, instead of the control error decreasing as the Kp1 gain increases, the control error decreases as the Kp2 gain increases. Again, this makes sense since the control error plotted is the error between the reference offset for the branch temperatures and the actual offset between branch temperatures, which Controller 2 manages. Thus, when Controller 2 has higher gains, it can adjust the valve percentage more quickly, thereby lowering the control error for Controller 2. This trend of a higher Kp2 gain resulting in a lower control error for Controller 2 is highlighted with an arrow in Fig. 6.27.

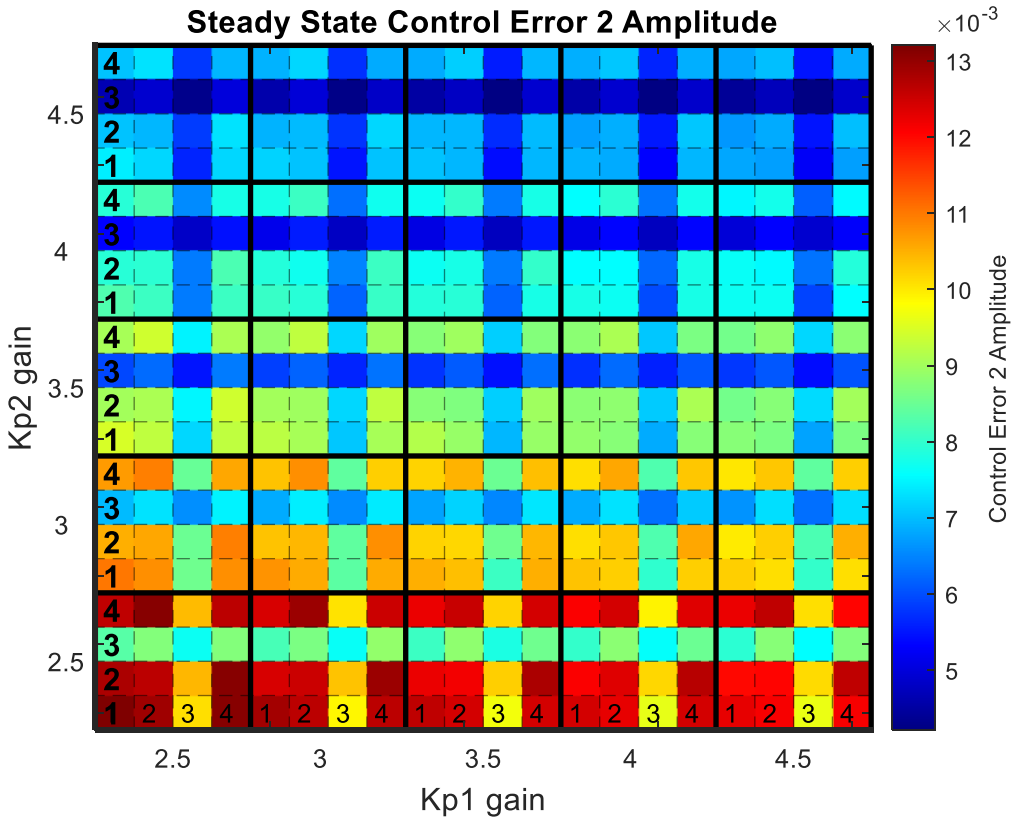

Figure 6.26: Controller Gain and Parameter Sweep for Control Error 2.

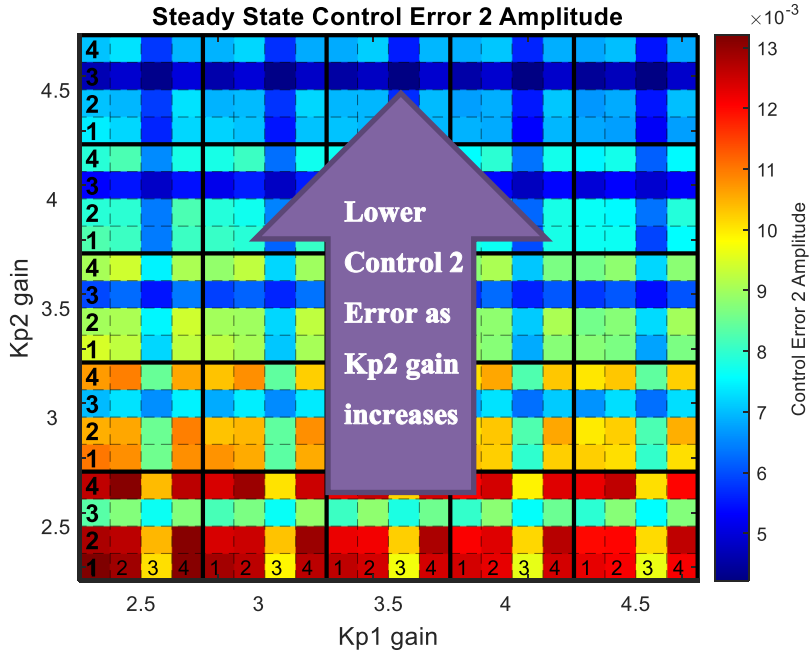

Figure 6.27: Control Error 2 as Kp2 Increases Trend.

When the normalized control error amplitudes are summed together, they yield Fig. 6.28 below. Individually, the control errors decreased as their respective controller gains increased. Therefore, it is not surprising that the configuration with the least control error occurs when both controller gains are at their maximum values of 4.5.

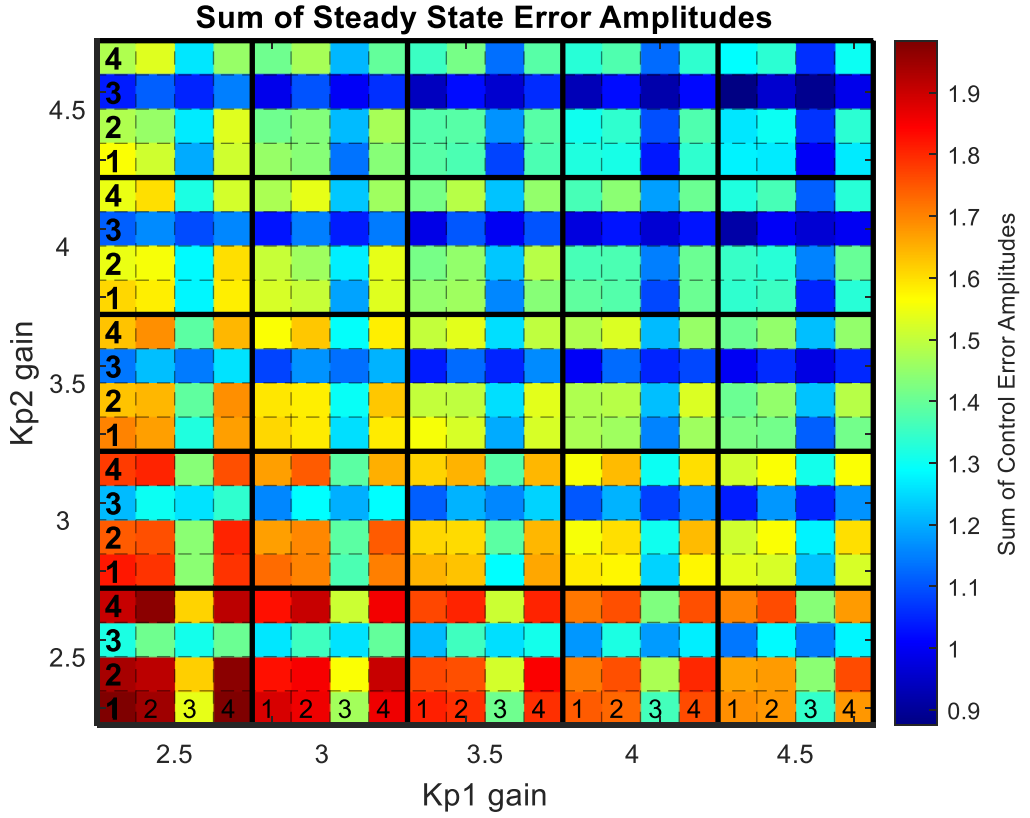

Figure 6.28: Controller Gain and Parameter Sweep for Control Errors 1 and 2.

The absolute lowest control error is at the intersection of having the highest controller gains and having set the parameters where Parameter 3 has the larger perturbation and Parameter 1 has the smaller perturbation. This lines up well with the sensitivity analysis performed in Chapters 4 and 5 as well as intuition about how higher controller gains affect the error term. This optimal configuration for lowering the control error is highlighted by a bright yellow square in Fig. 6.29. This figure also accentuates the pattern of larger controller gains leading to lower control errors.

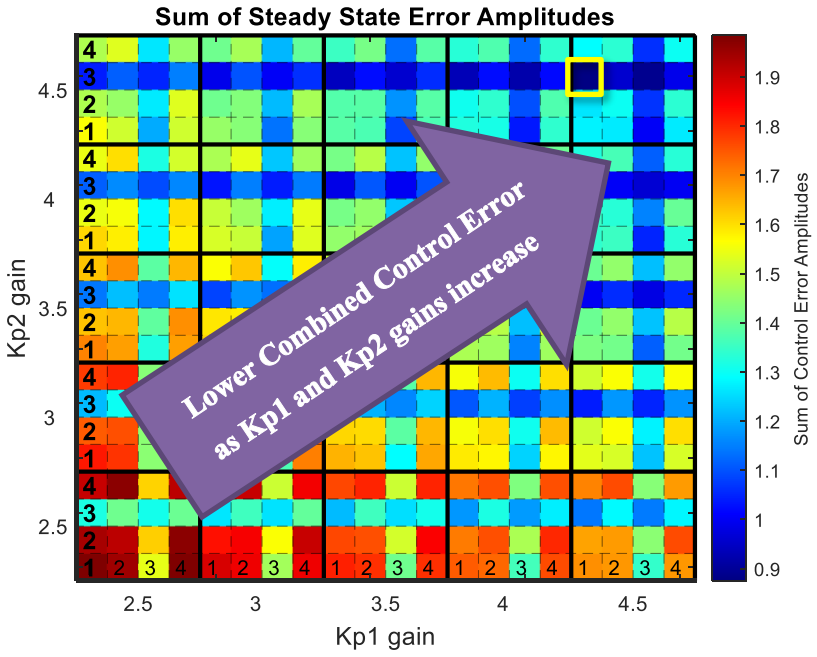

Figure 6.29: Control Error Terms as Both Gains Increase Trend.

Now that the optimal configuration for lowering the control error has been discovered, the control input must also be compared. The amplitudes of the control inputs for Controller 1 are shown in Fig. 6.30. Fig. 6.30 demonstrates a pattern where lower gains for Controller 1 yield lower control inputs. This makes intuitive sense, because the control input is simply the control error multiplied by the proportional gain. Thus, just because the control input is lower, does not indicate a better configuration. This necessitates the control error be plotted as well. From Fig. 6.29 and Fig. 6.30 it is clear there is a trade-off between minimizing control error and minimizing control input, when tuning the controller gain. This is left to the engineer to decide how stringent the error tolerances are for the temperature amplitude, or how necessary it is to conserve power and keep the control input amplitude as low as possible. Regardless of this trade-off, the optimal parameter combination is to focus efforts on increasing the capacitance of Cold Plate 2, followed by Cold Plate 1, before any other improvement to the physical plant. This is assuming there is an equal cost to increase any of the parameter capacitances.

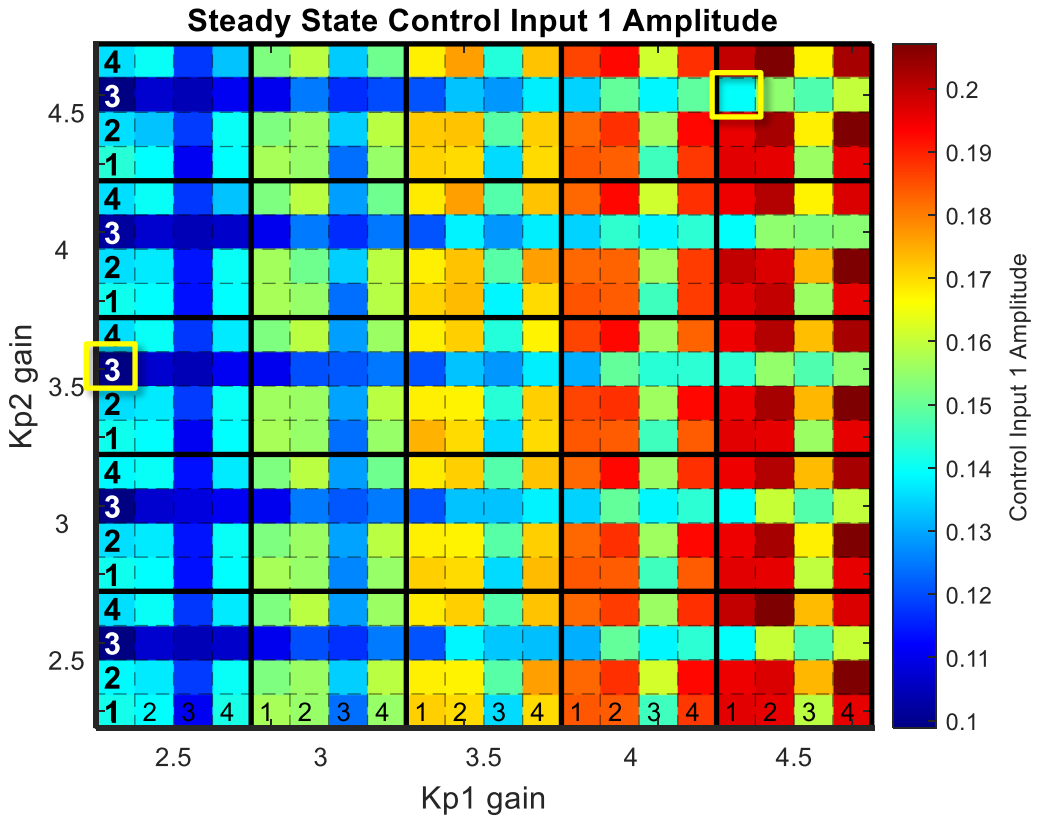

Figure 6.30: Controller Gain and Parameter Sweep for Control Input 1.

6.3.2 Time Trace Visualization of the System Performance

This next subsection presents the time domain simulations for three different configurations. The first is the nominal case, where the controller gains for both controllers are 3 and there are no parameter perturbations. The parameter values are at their nominal values. The second configuration is the one with the lowest error, highlighted with a yellow box in Fig. 6.29. This configuration has controller gains of 4.5 for both controllers and Parameter 3 has the larger perturbation while Parameter 1 has the smaller perturbation. The final configuration which will be compared against the previous two is the other square with a yellow box around it in Fig. 6.30. It is the case with the lowest control input for Controller 1. Again, the control input for Controller 1 is the one being minimized because it requires work to pump varying levels of fluid into the system, whereas the control input for Controller 2 involves adjusting a valve, which is not as costly. This final configuration sees a Kp1 gain of 2.5 and a Kp2 gain of 3.5. The parameter perturbations are

the same as the lowest control error case, with a large Parameter 3 perturbation and a small Parameter 1 perturbation.

Fig. 6.31 shows the Control Error 1 signals for each of the configurations. Visually, the lowest control error configuration is the lowest in both amplitude and magnitude. The amplitude, or the distance from the peak to the trough divided by two, is less than the lowest control input configuration and both are less than the nominal case. The magnitude of the lowest error is also closest to zero, demonstrating how the lowest error configuration can bring the output temperature of the system closest to the reference temperature. The next lowest magnitude belongs to the nominal case, which makes sense because the nominal case has a Kp1 gain of 3, whereas the lowest control input configuration has a Kp1 gain of 2.5. The plot in Fig. 6.24 depicts that the lowest control error configuration has a smaller control error amplitude when compared to the lowest control input configuration by having a darker shade of blue.

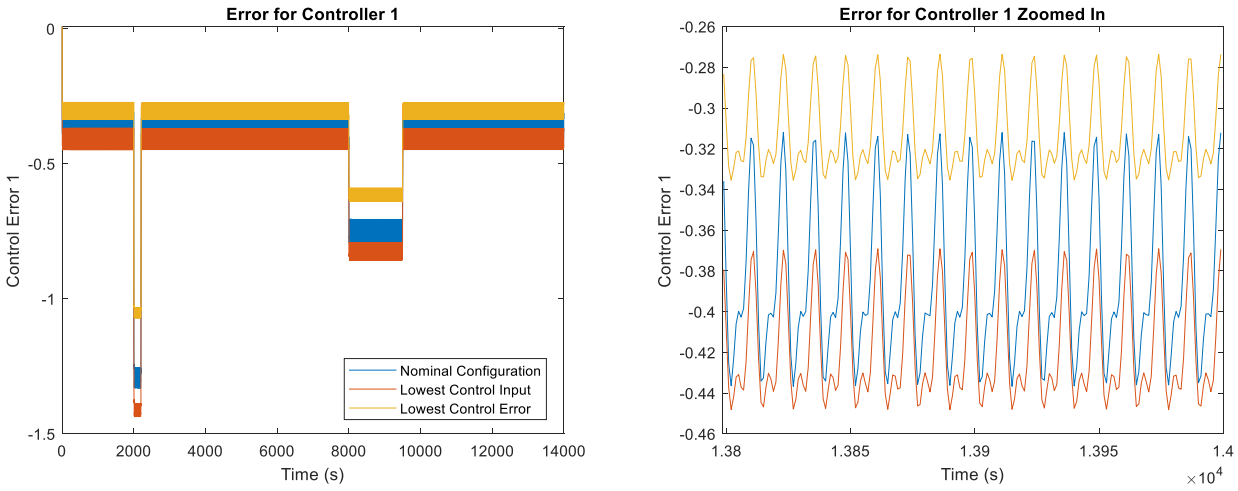

Figure 6.31: Time Trace of Controller 1 Error Signals.

The next figure presents the time trace of the error between the reference offset and the actual offset between the top and bottom branch temperatures. This is also called the error for Controller 2, and it is plotted for the three configurations in Fig. 6.32. It is easy to see that both the amplitude and magnitude of the lowest control error configuration are the smallest out of the three. The lowest control input configuration has the next lowest amplitude and magnitude, followed by the nominal case. The magnitude of the control error signals follow logically from how the lowest error configuration has a Kp2 gain of 4.5, the lowest control input configuration has a Kp2 gain of

3.5, and the nominal case has a Kp2 gain of 3. The ranking of the difference in amplitude follows mostly from the parameter perturbations and the sensitivity of the Control Error 2 signal to Parameter 3 followed by Parameter 2, as shown in Fig. 6.12 and 6.15. The effect that the combination of the parameters and controller gains has on the Control Error 2 amplitude is demonstrated in Fig. 6.26, where the lowest control error configuration clearly has a darker blue shade than the lowest control input configuration.

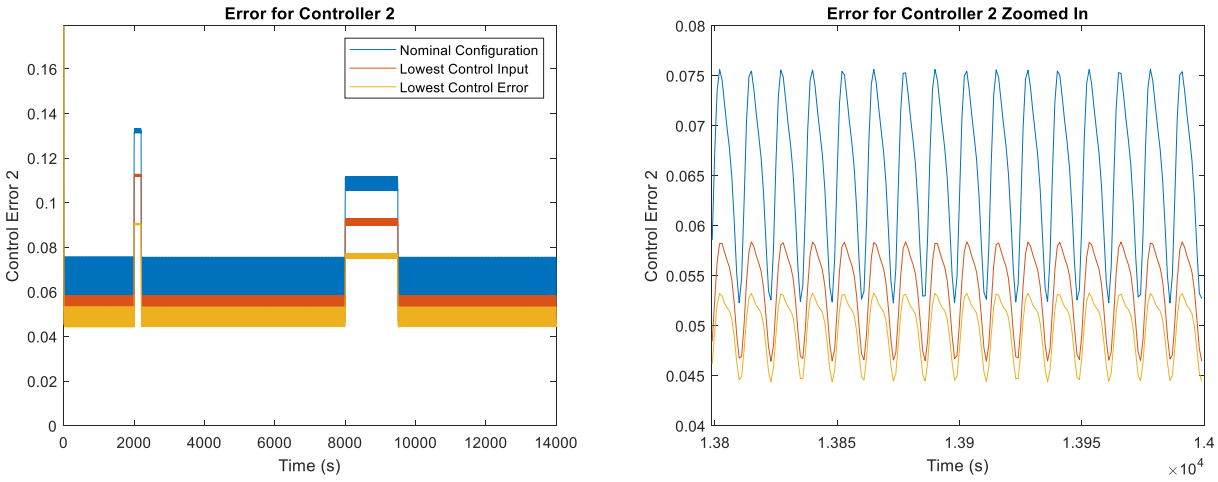

Figure 6.32: Time Trace of Controller 2 Error Signals.

The final figure below demonstrates that although the lowest error configuration had a lower error for both Controller 1 and 2, the lowest control input configuration has a lower control input. Fig 6.33 below shows the lowest control input configuration clearly has the smallest amplitude in addition to being closer to 0, signifying a lower magnitude as well. Logically, since the lowest control input configuration has lower Kp1 and Kp2 gains than the lowest error configuration, it makes sense that the control input would also be lower. The nominal case falls somewhere in the middle, having smaller gains than the lowest control error case, and a higher Kp1 gain, which is more relevant to the control input for Controller 1, than the lowest control input configuration. Although the lowest control error configuration has a larger control input value, it still has a smaller signal amplitude than the nominal case, since the parameters most effective in filtering the control input signal amplitude have been perturbed. The ranking of the Control Input 1 amplitudes is also shown in Fig. 6.30, where the lowest control input configuration has a darker blue, representing a smaller amplitude, than the lowest error configuration.

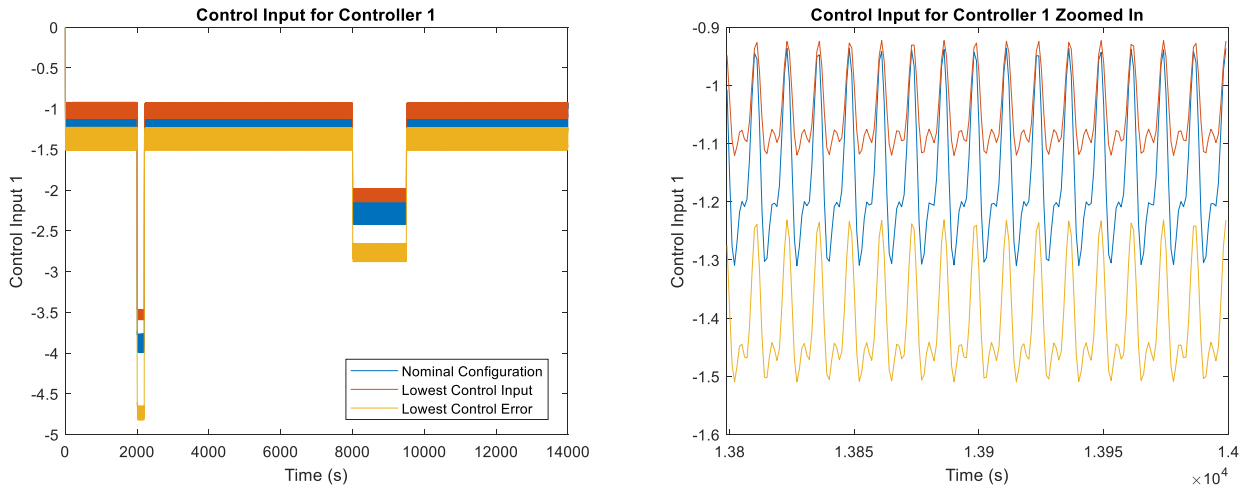

Figure 6.33: Time Trace of Control Input 1 Signals.

This study demonstrated that the sensitivity analysis can be used to accurately predict which parameters would have the greatest effect on filtering the control input or error term sinusoidal signals. These parameters can then be adjusted to have the desired effect on both the control input and the error. However, this co-design section showed that the controller gains also have an impact on the signal amplitudes, as evidenced by Fig. 6.33. Although the lowest control input and lowest control error configurations have identical perturbations, the amplitude for the lowest control input configuration is smaller than the lowest error case. This is also seen in the colorful gain and parameter sweep plots in Fig. 6.24 – 6.30. Although the same parameter perturbation pattern exists in each 4x4 block, the blocks still change colors, representing changing signal amplitudes, as the controller gains are also adjusted. There is a trade-off evident from comparing the control error and control input gain and parameter sweep plots. The squares which have the lowest control input will not necessarily have the lowest control error. Similarly, the square which was selected as having the lowest control error, had a higher control input than many of the similarly configured cases with lower controller gains. This study also demonstrated that the magnitude of the control error or control input signal is directly related to the controller gains, which can be tuned to either sacrifice the accuracy of having lower control errors or the associated power required for higher control inputs. Overall, the sensitivity analysis tools developed in Chapters 4 and 5 can be used to narrow down the most influential parameters, which will save both time and computational costs when the co-design optimization is performed.

Chapter 7
Conclusions and Future Work

7.1 Conclusions

In this study, a sensitivity analysis was applied to a graph-based model to inform which plant parameters were most influential to the modeled system. The list of parameters was narrowed down to the top four and these were optimized together with two controller gains. This reduction in the space of parameters greatly simplified the optimization problem and saved time and computational effort for the co-design process. Then the combination of gains and parameter values which lowered the control gain amplitude the most were selected. Thus, the plant and controller were designed simultaneously to minimize the control input of the controller, while achieving the best performance, defined as having the lowest control error.

One of the biggest benefits of the transfer function sensitivity analysis method is the ability to piece together subsystems in the frequency domain and easily build up an equation which represents the dynamics of the entire model. Having a single equation to represent the model output allows the partial derivatives to be calculated in one step, easily solving the sensitivity of the system to each of the parameters. The ability to represent the system as a block diagram in the frequency domain also allows the signal which is considered as the output to be calculated easily through simple block diagram manipulation rules. Additionally, the methods of building up system equations from each of the subsystems allows the sensitivity analysis of graph-based models to be equally scalable to the models themselves. Therefore, it would be simple to apply these sensitivity analysis tools and allow for the co-design of a graph-based model of any size in the future.

However, a drawback of this method is that the sensitivity can only be calculated for one output at a time. To find the sensitivity of several outputs simultaneously, they would have to be calculated individually and added together with a weighting scheme to select the most influential

parameters. However, this would require further analysis to decide on appropriate weighting values. Another drawback of this method is the dependence on having a linearized model. Since the mass flow rates in most thermal management systems are not constant, the models will be nonlinear and the sensitivity analysis is only accurate around the operating point used for linearization.

7.2 Future Work

In Section 5.2.1 of Chapter 5, Fig. 7.1 below was included to begin a discussion of how the capacitance parameters affected the transient dynamics.

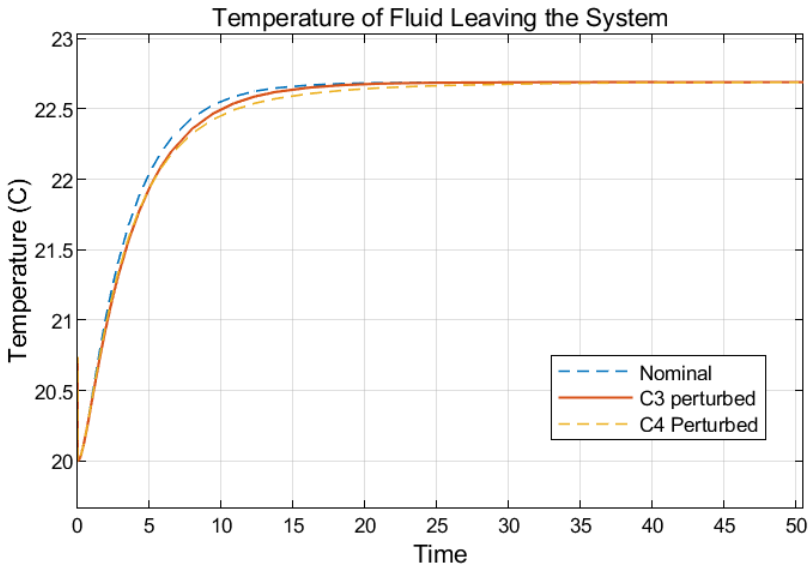

Figure 7.1: Perturbations of C3 and C4 Compared to the Nominal Case.

In this example, parameter C_4 slows down the dynamics more than parameter C_3 does. The system's output temperature is also more sensitive to parameter C_4, according to the sensitivity analysis. This is demonstrated by how the output temperature amplitude changes more when C_4 is perturbed than C_3, even though they are perturbed by the same amount. Recall that the sensitivity analysis determines the ratio of the sensitivity output signal amplitude over input signal amplitude when the output reaches steady state for different input signal frequencies. However, there appears to be a correlation between the sensitivity analysis and the transient dynamics of the system. This leads us to examine whether the sensitivity analysis can be useful in determining the sensitivity of the system at other frequencies and not just at steady state.

Fig. 7.2 shows a series of step inputs acting on the graph-based model pictured in Fig. 7.3. The first step input occurring at 50 s is Input 1, or the source temperature signal. It rises from a value of 20 °C to 21 °C, while all the other inputs signals are zero. At 100 s, the second input, or Heat Load 1, rises from 0 kW to 1 kW. Next, Heat Load 2, or the third input signal, also increases from 0 kW to 1 kW, this one at 150 s. Then Heat Load 3, the fourth input signal, increases from 0 kW to 2 kW at 200s. Finally, at 250 s, the final input signal, Heat Load 4, increases from 0 kW to 5 kW.

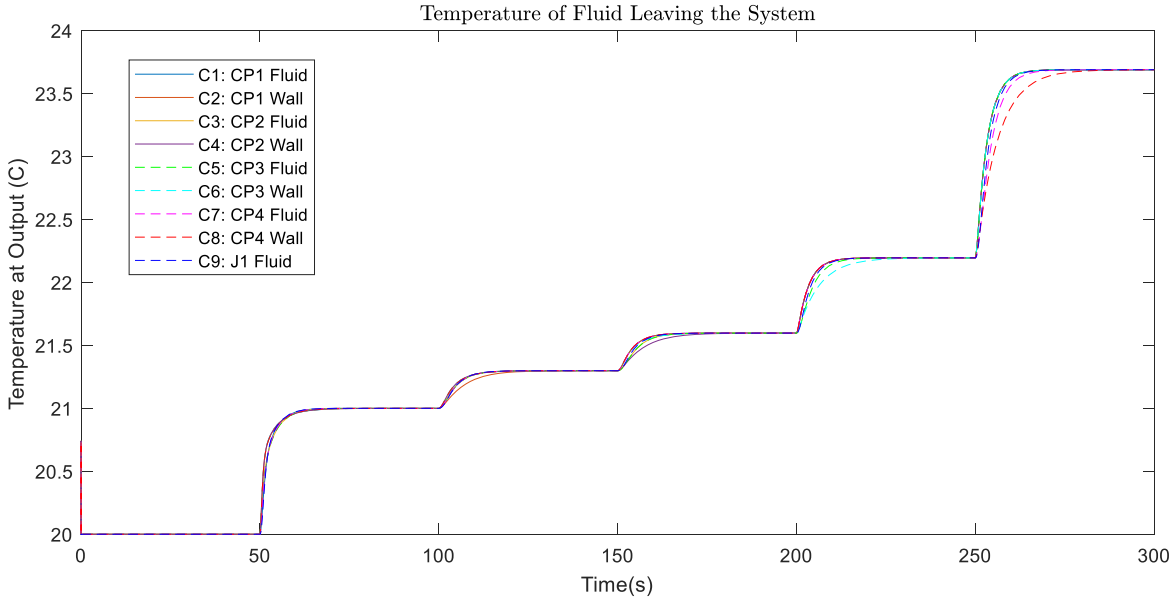

Figure 7.2: Series of Step Inputs on Example 2.

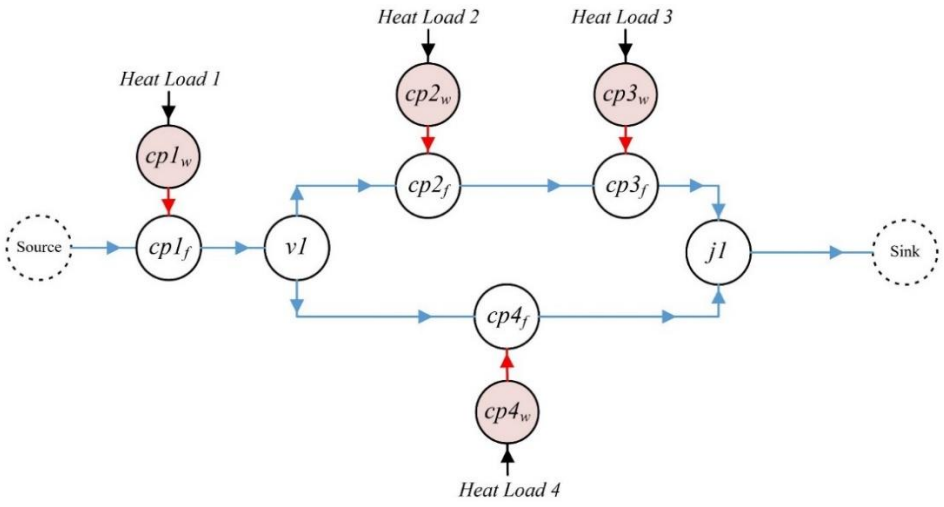

Figure 7.3: Graph-Based Model of Example 2.

The most influential parameters are the ones which result in the dynamics that are furthest from the nominal case. Since the nominal case has the quickest dynamics, the parameters which slow down the system response the most are the most influential for that input signal. For the first step, when the source temperature instantly steps to 21 °C, the parameters seem to be similarly influential. However, for the heat load input signals, the most influential parameter is the associated wall capacitance. For example, the temperature of the fluid when Heat Load 1 is a step input is most sensitive to parameter C_2, which is the wall capacitance of Cold Plate 1, the cold plate which Heat Load 1 is applied to. This matches up with the sensitivity analysis shown again in Fig. 7.4. Notice how the most influential parameter for the largest frequency range of the heat load input sensitivity bode plots is the cold plate wall which the heat load is applied to. For the equation of the output with respect to Heat Load 1, the wall capacitance of Cold Plate 1 is the most influential. Simiarly, the equations of the outputs with respect to Heat Load 2, Heat Load 3, and Heat Load 4 are most sensitive to the wall capacitance of Cold Plate 2, Cold Plate 3, and Cold Plate 4, respecitvely.

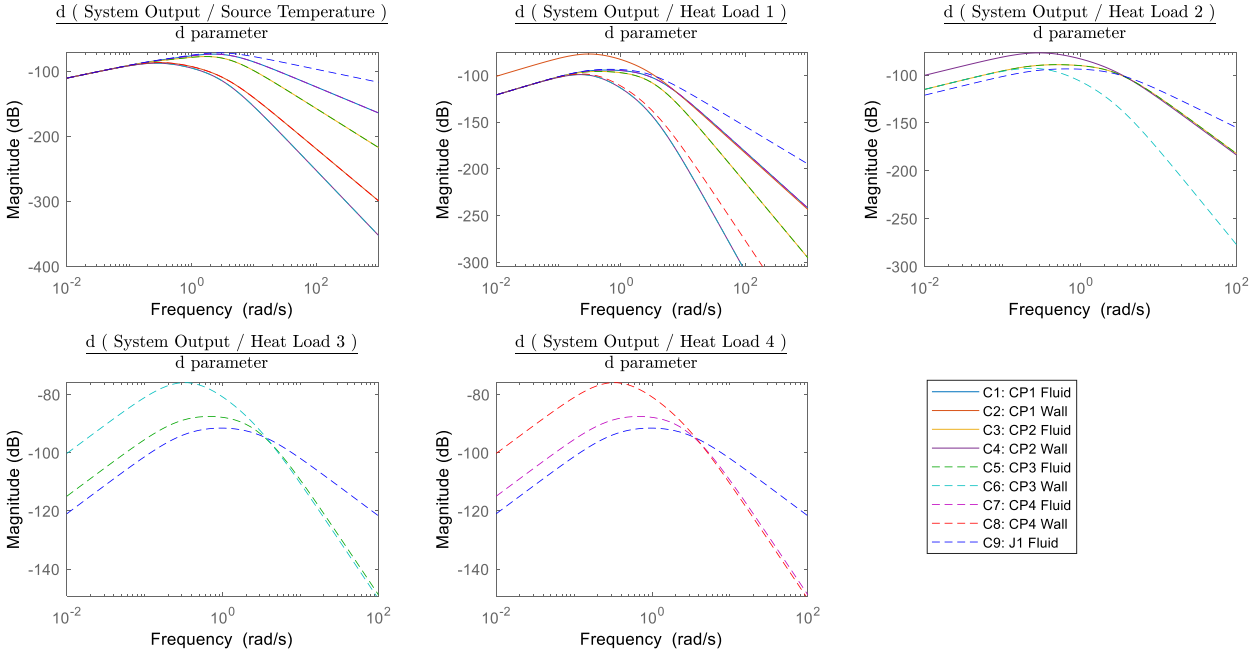

Figure 7.4: Sensitivity Bode Plots of Example 2.

The individual step responses of the system to each of the isolated inputs are plotted in Fig. 7.5 for better comparison. This makes it easier to see that the most influential parameter for Input

Signals 2-5 is the wall capacitance for that heat load. It is also easier to see that the next most influential parameters match the sensitivity bode plot rankings from Fig. 7.4, especially apparent for Input Signals 3-5.

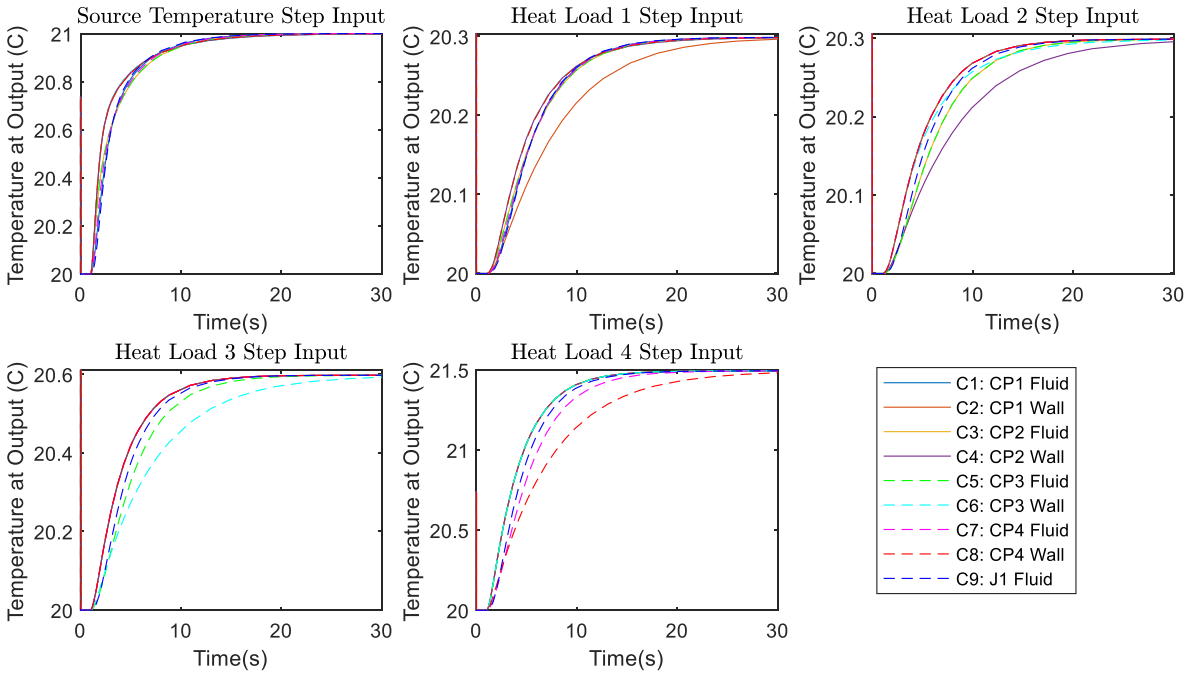

Figure 7.5: Parameter Effects on Step Responses for Each Input Signal.

A step function in the time domain can be represented by the entire spectrum of frequencies in the frequency domain. Therefore, a step response is useful in revealing how a system would respond to a range of frequencies. Fig. 7.6 shows the Heat Load 4 step response from Fig. 7.5 zoomed in to the first 4 seconds.

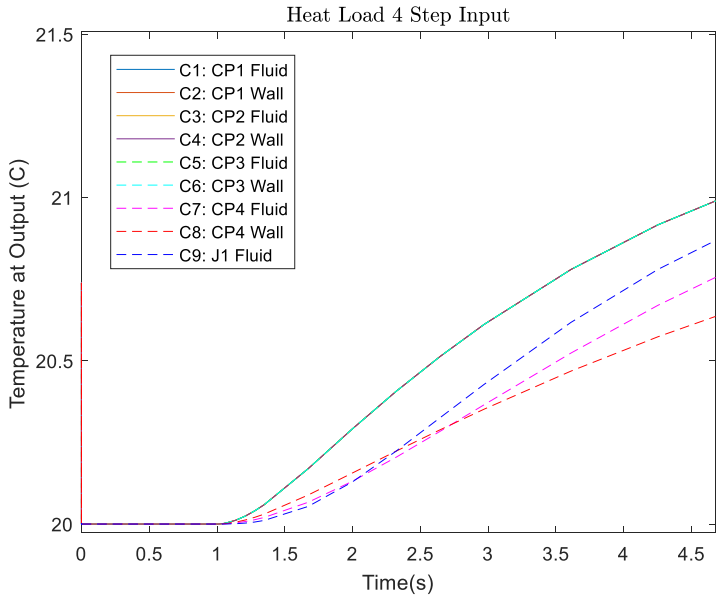

Figure 7.6: Heat Load 4 Step Response.

Notice how in the first 2 seconds, parameter C_9 is the most influential, followed by C_7 and then by C_8. This matches the sensitivity bode plot for Heat Load 4 in Fig. 7.4 at frequencies greater than 4 rads/s, which shows parameter C_9 as the most influential, followed by C_7 and then by C_8. Then at around 3 rad/s on the sensitivity bode plot, and 3 seconds on the step response, the ranking in both plots show that C_8 is the most influential, followed by C_7 and then C_9 for the remaining frequencies or time. There is clearly some correlation between the higher frequencies on a bode plot and the first few seconds of a step response.

The transfer function sensitivity analysis method allows for more than one input signal to be applied to the system at once. Due to the superposition principle, the sensitivities of a few equations which relate the output to several inputs can be combined to determine to total sensitivity to any parameter. The sensitivities at specific input signal frequencies can be identified and multiplied by the input signal amplitude. Note that this analysis is only valid for a linear system or a system that has been linearized about an operating point. If looking at a linearized system, the analysis is only valid around that operating point.

Fig. 7.7 shows the step response of the system in Example 2 to simultaneous step inputs for Heat Load 1 and Heat Load 2, both from 0 kW to 1 kW.

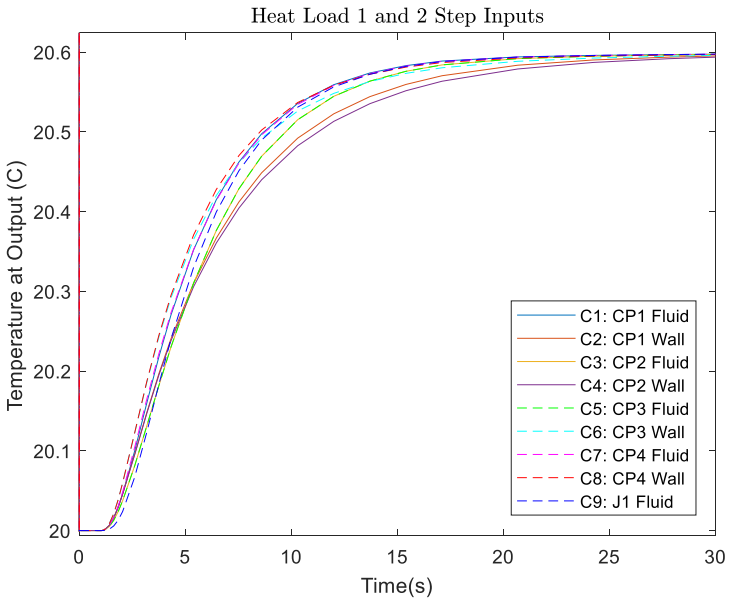

Figure 7.7: Simultaneous Heat Load 1 and Heat Load 2 Step Responses.

The ranking of the parameters matches the sensitivity plot from Scenario 3 in Chapter 5, where Heat Load 1 and 2 had amplitudes of 1 kW at frequencies of 1 and 0.5 rad/s, respectively. This sensitivity plot is included again as Fig. 7.8. Recall that to generate the sensitivity plot, the sensitivities of the output temperature over the input temperature with respect to the parameters were recorded at the specific frequency of the input signal and then multiplied by the input signal amplitude. Then the contributions to the output temperature amplitude from both input signals at different frequencies were added together.

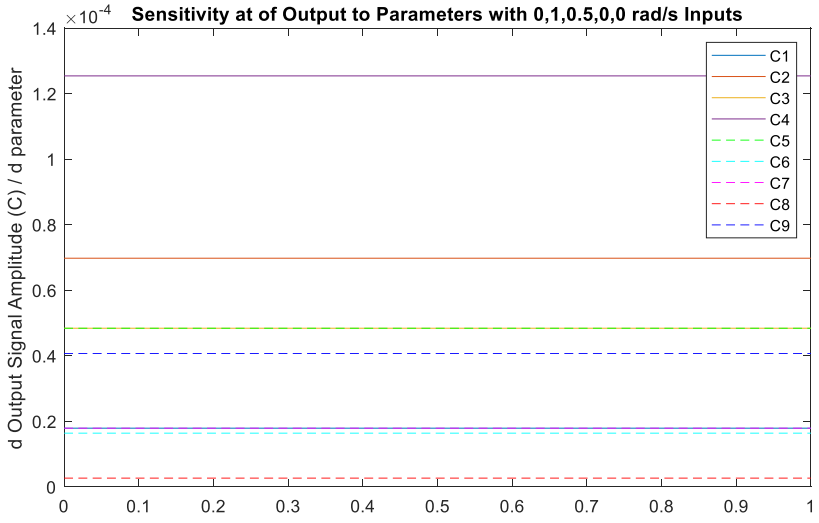

Figure 7.8: Sensitivity Plot of Scenario 3 from Chapter 5.

However, the input signals shown in Fig. 7.7 are not at different frequencies and go through the frequency spectrum at the same time. Since the amplitudes of both input signals are equal and the step functions for both inputs occur at the same moment, perhaps their sensitivities can be summed. Fig. 7.9 below shows the sum of the sensitivity bode plots for both Heat Load 1 and Heat Load 2 from Fig. 7.4 on the same plot. Notice in Fig. 7.4, C_2 was the most influential parameter for the Heat Load 1 transfer function and C_4 was the most influential parameter for the Heat Load 2 transfer function, with very comparable peak values of -77.2 dB and -76.8 dB, respectively. When the two sensitivity bode plots are added together, the highest point of C_4 climbs to -76.4 dB since C_4 appears in both transfer functions, and C_2 maintains its peak at -77.2 dB. This is demonstrated in Fig. 7.9, which also shows at all other frequencies, the added sensitivities of the transfer functions relating the output temperature to the two heat loads. Notice how the parameter influence ranking at the higher frequencies seems to match the numerical simulation in Fig. 7.7 for the first 3 seconds after the step functions. Then the step responses when each of the parameters are perturbed from seconds 5-25 seem to match the frequencies lower than 2 rad/s on the sensitivity bode plot in Fig. 7.9.

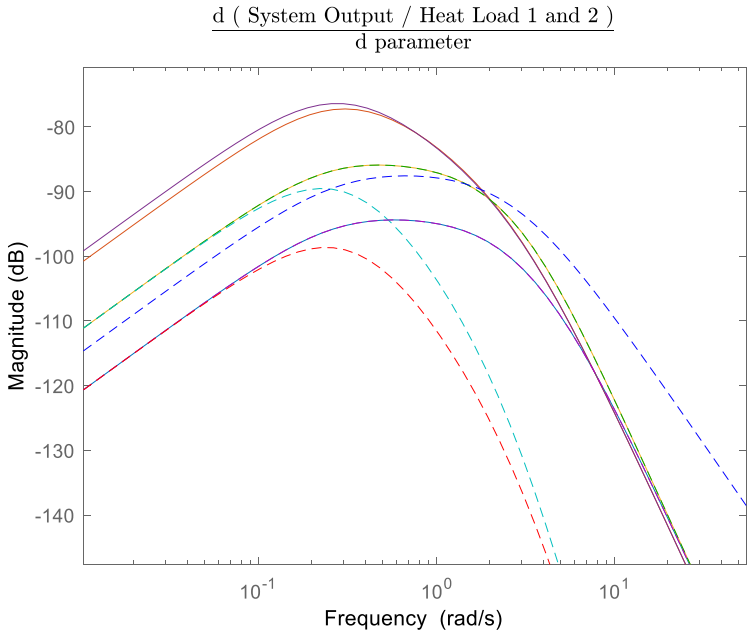

Figure 7.9: Summed Sensitivity Bode Plots for Inputs Heat Load 1 and 2.

There appears to be a correlation between the partial derivative of the system equation with respect to a parameter, or the sensitivity bode plot, and the step response of the system when that

parameter is perturbed. This would be a very interesting area to explore and perhaps derive a way to map the transient response in the time domain to portions of the sensitivity bode plot, which describes the ranking of influence each parameter has for a given input frequency. The connection between time constants and cutoff frequencies might be a good starting point for this future work. If this connection could be further understood and applied to signals which are neither sinusoidal nor periodic, then these sensitivity analysis and co-design methods could be applied to a wider variety of systems.

The following two figures are also included as a further example. Suppose the step input for Heat Load 1 has a value of 2 kW and Heat Load 2 remains at 1 kW. Then C_2, which was more influential to the transfer function relating the output temperature to Heat Load 1, is now proportionally more influential to the system now that the value of Heat Load 1 has increased in proportion. The numerical simulation showing the step response of the system when each of the parameters are perturbed is shown in Fig. 7.10.

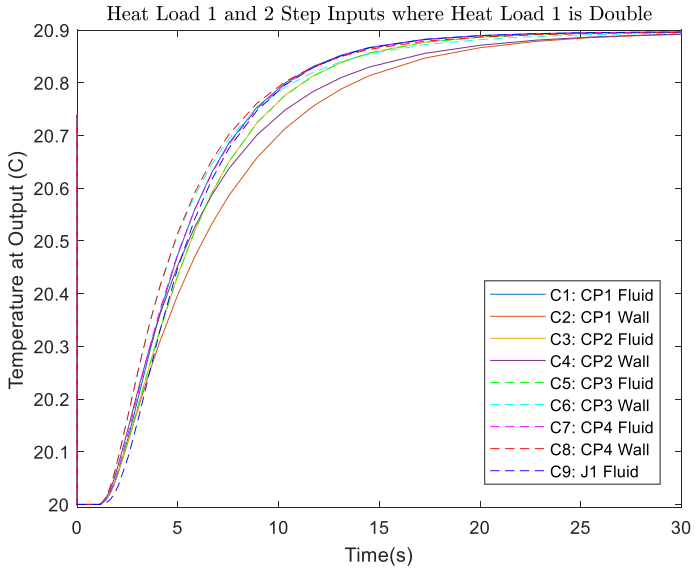

Figure 7.10: Step Responses of Output Temperature when Heat Load 1 is Doubled.

For Fig. 7.9, the sensitivity bode plots for the transfer functions pertaining to the inputs Heat Load 1 and Heat Load 2 were added directly since the inputs shared the same value. Now, the sensitivity bode plot for Heat Load 1 should be doubled before summing with the sensitivity bode plot for Heat Load 2, as shown in Fig. 7.11. Notice how the output temperature is now most sensitive to parameter C_2 instead of to C_4, as it was in the scenario shown in Fig. 7.9.

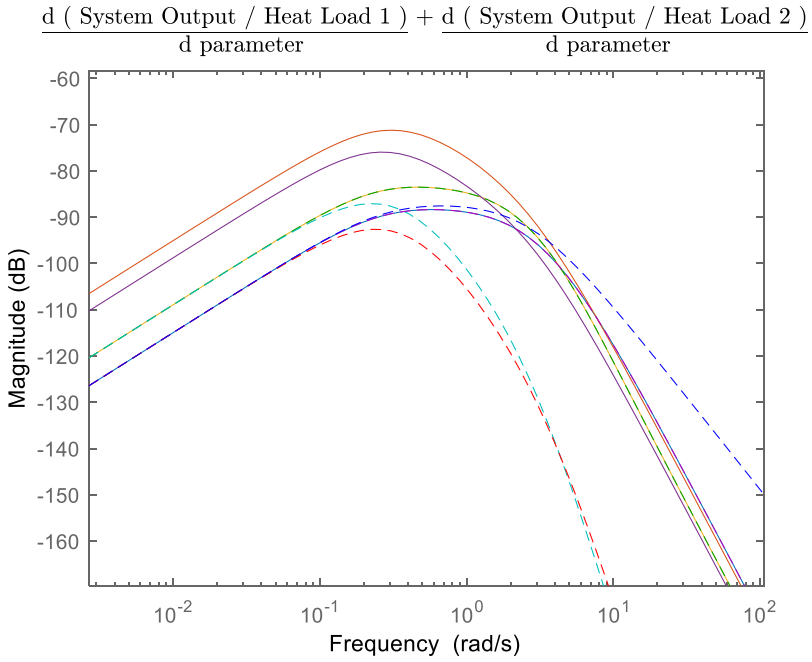

Figure 7.11: Summed Sensitivity Bode Plots when Input Heat Load 1 is Doubled.

Looking forward, perhaps input signals in the time domain could be converted to the frequency domain via a Fourier Transform. Then for the duration of time at which the frequencies of the input signal are known, the total sensitivity of the output to each of the parameters can be calculated based on the percentage the input signal was at each frequency. For example, if the input signal was known to be at a frequency of 1 rad/s for 70% of the time and at a frequency of 0.5 rad/s for the remaining 30% of the time, then the sensitivities of the output to each of the parameters at 1 rad/s can be multiplied by 0.7 and added to the sensitivities at 0.5 rad/s multiplied by 0.3. Note that this is merely a suggestion of what could be attempted next and is in no way guaranteed to yield viable results.

In conclusion, there seems to be a strong correlation between step responses and the sensitivity bode plots introduced in this work. It would be worthwhile to explore this idea further and perhaps map parts of the step response to specific frequencies on the sensitivity bode plots. A potential future goal for this work is to identify the overall top parameters the chosen output is most sensitive to for a non-periodic input signal. These parameters would be optimized alongside the controller to design the optimal plant and controller pair quickly and for low computational costs.

References

[1] H. C. Frey and S. R. Patil, “Identification and review of sensitivity analysis methods,” *Risk Anal.*, vol. 22, no. 3, pp. 553–578, 2002.

[2] G. Sin, K. V. Gernaey, and A. E. Lantz, “Good modeling practice for PAT applications: Propagation of input uncertainty and sensitivity analysis,” *Biotechnol. Prog.*, vol. 25, no. 4, pp. 1043–1053, 2009.

[3] T. Turányi, “Applications of sensitivity analysis to combustion chemistry,” *Reliab. Eng. Syst. Saf.*, vol. 57, no. 1, pp. 41–48, 1997.

[4] E. Rozenwasser and R. Yusupov, *Sensitivity of Automatic Control Systems*. 2019.

[5] R. Precup and S. Preitl, “Stability and Sensitivity Analysis of Fuzzy Control Systems . Mechatronics Applications,” *Control*, vol. 3, no. 1, pp. 61–76, 2006.

[6] S. S. Lin, S. C. Horng, and C. H. Lin, “Embedding sensitivity theory in ordinal optimization for decentralized optimal power flow control,” *Int. J. Electr. Power Energy Syst.*, vol. 34, no. 1, pp. 145–153, 2012.

[7] M. Brenna, E. De Berardinis, F. Foiadelli, G. Sapienza, and D. Zaninelli, “Voltage Control in Smart Grids: An Approach Based on Sensitivity Theory,” *J. Electromagn. Anal. Appl.*, vol. 02, no. 08, pp. 467–474, 2010.

[8] B. Bakhshideh Zad, H. Hasanvand, J. Lobry, and F. Vallée, “Optimal reactive power control of DGs for voltage regulation of MV distribution systems using sensitivity analysis method and PSO algorithm,” *Int. J. Electr. Power Energy Syst.*, vol. 68, pp. 52–60, 2015.

[9] M. Caliano, N. Bianco, G. Graditi, and L. Mongibello, “Design optimization and sensitivity analysis of a biomass-fired combined cooling, heating and power system with thermal energy storage systems,” *Energy Convers. Manag.*, vol. 149, no. July, pp. 631–645, 2017.

[10] S. Cho and J. Y. Choi, “Efficient topology optimization of thermo-elasticity problems using coupled field adjoint sensitivity analysis method,” *Finite Elem. Anal. Des.*, vol. 41, no. 15, pp. 1481–1495, 2005.

[11] H. Xu, W. Li, M. Li, C. Hu, S. Zhang, and X. Wang, “Multidisciplinary robust design optimization based on time-varying sensitivity analysis,” *J. Mech. Sci. Technol.*, vol. 32, no. 3, pp. 1195–1207, 2018.

[12] J. Kim, M. J. Realff, and J. H. Lee, “Optimal design and global sensitivity analysis of biomass supply chain networks for biofuels under uncertainty,” *Comput. Chem. Eng.*, vol. 35, no. 9, pp. 1738–1751, 2011.

[13] K. Dykes, A. Ning, R. King, P. Graf, G. Scot, and P. Veers, “Sensitivity analysis of wind

plant performance to key turbine design parameters: A systems engineering approach," *32nd ASME Wind Energy Symp.*, no. January, pp. 1–26, 2014.

[14] P. Seferlis, "Sensitivity Analysis F O R C H E M I C a L Process," vol. 20, no. 10, pp. 1177–1200, 1996.

[15] P. Abolmoali, S. S. Patnaik, T. Deppen, and M. Boyd, "Sensitivity analysis for the design of aircraft thermal management and power systems," *AIAA Scitech 2021 Forum*, no. 88, pp. 1–20, 2021.

[16] R. L. Iman and S. C. Hora, "A Robust Measure of Uncertainty Importance for Use in Fault Tree System Analysis," *Risk Anal.*, vol. 10, no. 3, pp. 401–406, 1990.

[17] M. S. Eldred *et al.*, "DAKOTA, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis (SAND 2001-3514)," 2014.

[18] J. T. Allison, "Engineering system co-design with limited plant redesign," *Eng. Optim.*, vol. 46, no. 2, pp. 200–217, 2014.

[19] H. C. Pangborn, "Hierarchical Control for Multi-Domain Coordination of Vehicle Energy Systems with Switched Dynamics," 2019.

[20] H. C. Pangborn, J. P. Koeln, M. A. Williams, and A. G. Alleyne, "Experimental validation of graph-based hierarchical control for thermal management," *J. Dyn. Syst. Meas. Control. Trans. ASME*, vol. 140, no. 10, pp. 1–16, 2018.

[21] M. A. Williams, J. P. Koeln, H. C. Pangborn, and A. G. Allenye, "Dynamical Graph Models of Aircraft Electrical, Thermal, and Turbomachinery Components," *J. Dyn. Syst. Meas. Control. Trans. ASME*, vol. 140, no. 4, pp. 1–17, 2018.

[22] C. T. Aksland and A. G. Alleyne, "Experimental Model and Controller Validation for a Series Hybrid Unmanned Aerial Vehicle," *Proc. Am. Control Conf.*, vol. 2020-July, pp. 4154–4160, 2020.

[23] C. Laird, D. J. Docimo, C. T. Aksland, and A. G. Alleyne, "DSCC2020-3 233," pp. 1–9, 2020.

[24] C. Aksland, "Modular Modeling and Control of a Hybrid Unmanned Aerial," 2019.

[25] J. P. Koeln, H. C. Pangborn, M. A. Williams, M. L. Kawamura, and A. G. Alleyne, "Hierarchical Control of Aircraft Electro-Thermal Systems," *IEEE Trans. Control Syst. Technol.*, vol. PP, pp. 1–15, 2019.

[26] D. M. Hamby, "A Review of Techniques for Parameter Sensitivity," *Environ. Monit. Assess.*, vol. 32, no. c, pp. 135–154, 1994.

[27] C. C. Margossian, "A review of automatic differentiation and its efficient implementation," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 9, no. 4, pp. 1–32, 2019.

[28] W. Borutzky, *Bond Graph Modelling of Engineering Systems*. 2011.

[29] Y. Ou and J. B. Dugan, "Sensitivity Analysis of Modular Dynamic Fault Trees p t I Prob i j t = @ R R p , R q Q p , Q q I Prob i j t I Rate i j t I Coverage i j t I Type C m j t q t I Prob

i j t = @ Q p," no. 1.

[30] X. Cao and H. Chen, "Perturbation Realization , Potentials , and," *October*, vol. 42, no. 10, pp. 1382–1393, 1997.

[31] W. Borutzky, "Bond graph modelling and simulation of mechatronic systems an introduction into the methodology," *20th Eur. Conf. Model. Simul. Model. Methodol. Simul. Key Technol. Acad. Ind. ECMS 2006*, pp. 17–28, 2006.

[32] W. Borutzky and J. Granda, "Bond graph based frequency domain sensitivity analysis of multidisciplinary systems," *Proc. Inst. Mech. Eng. Part I J. Syst. Control Eng.*, vol. 216, no. 1, pp. 85–99, 2002.

[33] J. M. Cabanellas, J. F. Félez, and C. Vera, "A formulation of the sensitivity analysis for dynamic systems optimization based on pseudo bond graphs.," *Simul. Ser. 27*, 1994.

[34] P. J. Gawthrop, "Sensitivity bond graphs," *J. Franklin Inst.*, vol. 337, no. 7, pp. 907–922, 2000.

[35] D. J. W. De Pauw and P. A. Vanrolleghem, "Practical aspects of sensitivity function approximation for dynamic models," *Math. Comput. Model. Dyn. Syst.*, vol. 12, no. 5, pp. 395–414, 2006.

[36] J. Banerjee, "Graph-Theoretic Sensitivity Analysis of Dynamic Systems," 2013.

Appendix A: Code

A.1 Overview

Once you have built your Simulink model out of DAEMOT components, you can run the script "Batch_Assign_Params." This renames all the component parameters to the correct variable names. Next run "Setup_Component_Params" to assign values to each of these component parameters in the workspace. Now you can run the Simulink Model.

To build a graph-based model you first need to set it up in "Setup_Graph_Therm." Do not run it yet, because you also need to set up "Gen_Graph_Therm" which actually generates the graph and is the last line of "Setup_Graph_Therm." Now run the "Setup_Graph_Therm" script. Now in Simulink you will need to build a function which reads in the mass flow rates, source power flows, sink states, vertex states from the previous iteration, and tank heights if you have tanks and are using the new heights to update tank capacitances in real time. However, updating capacitances does not work for the existing sensitivity analysis. If you want to use a tank, consider it to be flooded and a cold plate can represent the same interactions with the wall and heat loads from ambient. You can get these models from a current student, or you can find an example in my folder in a Simulink model called "SimpleCh2Example." The corresponding scripts for this model are called "Setup_Graph_Therm_Sense8", "Gen_Graph_Therm_Sense_7", "Setup_Component_Params_Sensitivity7", and "Sys2_Therm1_Mindy."

I've included a few scripts below which were used to generate the results in this thesis. First, I will include the script to complete the sensitivity analysis for Example System 1 (ExampleSys1SensAnalysis), followed by Example System 2 (ExampleSys2SensAnalysis). Then I will include the numerical verificiation script which runs through the different parameter perturbations (Numerical_Verification). I will note that these will only work on my DAEMOT models, since I have made added a "coeff" term to the mask of the components. This term acts like an input to the system and multiplies the fluid capacitance by a coefficient. You can edit the mask

to add this variable as well so that the numerical verification script will actually increment something in the model instead of just in the workspace. Then I will add the numerical verification script which generates the colorful grids in Chapter 6 for the co-design (Numerical_Verification_CoDesign). Finally, I've included two plotting scripts. Once for plotting the control inputs of different cases (Plot_ControlInput) and another for the colorful grids (Plot_CoDesign).

A.2 ExampleSys1SensAnalysis

```
%%Setup Model Equations
%%This is Hard Coded for the Example System 2

%Create symbolic variables for each of the governing equation matrices for
%the components
syms s a1 a2 a3 a4 b1 b2
%Create symbolic variables for each of the parameters so we can take the
%partial derivatives of them
syms C1 C2 C3 C4 E1 E2 E3

%Read in Input Values from model
temp_in = out.source1.data(); %Input 1 (Source Temp, degree C)
heatLoad1 = out.hl1.data(); %Input 2 (Heat Load 1, kW)
heatLoad2 = out.hl2.data(); %Input 3 (Heat Load 2, kW)

%Gather Capacitance Data from Model for Cold Plate 1
Cp = 1000*Fuel_Cp;
U1 = cp1.HTC;
A1 = pi*cp1.D*cp1.L;
Cw1 = 1000*cp1.M_wall*cp1.Cp_wall; %Wall Capacitance
Cf1 = cp1.Ac*cp1.L*Fuel_Rho*Cp; %Fluid Capacitance

%Gather Capacitance Data from Model for Cold Plate 2
U2 = cp2.HTC;
A2 = pi*cp2.D*cp2.L;
Cw2 = 1000*cp2.M_wall*cp2.Cp_wall; %Wall Capacitance
Cf2 = cp2.Ac*cp2.L*Fuel_Rho*Cp; %Fluid Capacitance

%Find mass flow rate steady state value (even if mass flow is sinusoidal)
tenth=floor(length(out.m.data)*0.95);
mflow1=mean(out.m.data(tenth:end))

%number edges and capacitances like the graph numbering for edges and capacitances
C1_v = Cw1; %cp1 wall cap
C2_v = Cf1; %cp1 fluid cap
C3_v = Cw2; %cp2 wall cap
C4_v = Cf2; %cp2 fluid cap

%Edge coefficients
E1_v = U1*A1; %convective edge between fluid and wall
E2_v = mflow1*Cp; %advective edge into cp2 fluid
E3_v = U2*A2; %convective edge between fluid and wall
```

```
%defining transfer function for Cold Plate 1
%Tin --> Eout
CP11_tf = (b1*s -a4*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP12_tf = (a2*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));

%separate numerator from denominator
[N1,D1]=numden(CP11_tf);
[N2,D2]=numden(CP12_tf);


for j=1:2
    %collect coefficients in front of each s term
    eval(['Ncoeffs = collect(N',num2str(j),', s);']);
    eval(['Dcoeffs = collect(D',num2str(j),', s);']);

    %group coefficients into array
    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');

    %Replace a1, a2, a3, a4, b1, b2 terms with edge and capacitance values
    %('v' stands for value)
    Nc = subs(Nc,[a1,a2,a3,a4,b1,b2],[-E2_v/C2_v-E1_v/C2_v,E1_v/C1_v,E1_v/C2_v,-
E1_v/C1_v ,E2_v,1000]);
    Dc = subs(Dc,[a1,a2,a3,a4,b1,b2],[-E2_v/C2_v-E1_v/C2_v,E1_v/C1_v,E1_v/C2_v,-
E1_v/C1_v ,E2_v,1000]);

    %Convert to double and record for each transfer function
    eval(['Nc',num2str(j),' = double(Nc);']);
    eval(['Dc',num2str(j),' = double(Dc);']);
end

%Creating transfer function with real values
CP11_tf_v = tf([Nc1],[Dc1]);
CP12_tf_v = tf([Nc2],[Dc2]);

%Creating transfer function with symbolic variables
%this step is needed for symbolic differentiation
%Replace a1, a2, a3, a4, b1, b2 terms with edge and capacitance symbolic
%variables
CP11_tf = subs(CP11_tf,[a1,a2,a3,a4,b1,b2],[-E1/C2-E2/C2,E1/C1,E1/C2,-E1/C1,E2,1000]);
CP12_tf = subs(CP12_tf,[a1,a2,a3,a4,b1,b2],[-E1/C2-E2/C2,E1/C1,E1/C2,-E1/C1,E2,1000]);

%Plot transfer functions with real values
figure()
eval(['bode(CP11_tf_v);']);
eval(['title("CP11 tf v");']);

figure()
eval(['bode(CP12_tf_v);']);
eval(['title("CP12 tf v");']);

%%
%defining transfer function for Cold Plate 2
%Tin --> Eout
CP33_tf = (b1*s -a4*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP34_tf = (a2*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));

[N1,D1]=numden(CP33_tf);
[N2,D2]=numden(CP34_tf);
```

```
for j=1:2
    eval(['Ncoeffs = collect(N',num2str(j),', s);']);
    eval(['Dcoeffs = collect(D',num2str(j),', s);']);

    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');

    Nc = subs(Nc,[a1,a2,a3,a4,b1,b2],[-E2_v/C4_v-E3_v/C4_v,E3_v/C3_v,E3_v/C4_v,-
E3_v/C3_v ,E2_v,1000]);
    Dc = subs(Dc,[a1,a2,a3,a4,b1,b2],[-E2_v/C4_v-E3_v/C4_v,E3_v/C3_v,E3_v/C4_v,-
E3_v/C3_v ,E2_v,1000]);

    eval(['Nc',num2str(j),' = double(Nc);']);
    eval(['Dc',num2str(j),' = double(Dc);']);
end

%Creating transfer function with real values
CP33_tf_v = tf([Nc1],[Dc1]);
CP34_tf_v = tf([Nc2],[Dc2]);

%Creating transfer function with symbolic variables
%this step is needed for symbolic differentiation
CP33_tf = subs(CP33_tf,[a1,a2,a3,a4,b1,b2],[-E2/C4-E3/C4,E3/C3,E3/C4,-E3/C3,E2,1000]);
CP34_tf = subs(CP34_tf,[a1,a2,a3,a4,b1,b2],[-E2/C4-E3/C4,E3/C3,E3/C4,-E3/C3,E2,1000]);

%%
%Sensitivity Analysis
%%Set up Transfer function equations

%Each of the inputs has their own transfer function equation
loops = ['src';'hl1';'hl2'];

eval([loops(1,:),' = CP11_tf*(1/C2)*CP33_tf*(1/C4)']); %source to out
eval([loops(2,:),' = CP12_tf*(1/C2)*CP33_tf*(1/C4)']); %hl1 to out
eval([loops(3,:),' = CP34_tf*(1/C4)']); %hl2 to out

%Parameters to take the partial derivatives of- you can shorten this to
%just the capacitances
param = {'C1';'C2';'C3';'C4';'E1';'E2';'E3'}

for n=1:length(loops)
    %For input-specific system transfer function separate numerator from
    %denominator
    eval(['[N,D]=numden(',loops(n,:),');']);
    %Collect the s terms (since its in freq domain)
    Ncoeffs = collect(N, s);
    Dcoeffs = collect(D, s);
    %Gather coefficients in correct order to prepare for tf() function
    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');
    %Substitute all parameter symbolic variables for numeric values
    %Right now we are just plotted the TF equation and don't need them to
    %be symbolic since we're not taking the partial derivative yet
    for k=1:length(param)
        eval(['Nc = subs(Nc,[',param{k},'],[',param{k},'_v]);']);
        eval(['Dc = subs(Dc,[',param{k},'],[',param{k},'_v]);']);
    end
    %Convert to double
```

```
    Nc = double(Nc);
    Dc = double(Dc);
    %Create TF for each input-specific TF equation
    eval([loops(n,:),'_v = tf([Nc],[Dc]);']);
end
%%
%%Take partial derivatives

%Note parameters to take the partial derivatives of
edgesofinterest =  param;
paramofedge = linspace(1,length(param),length(param));

for i=1:length(loops)
    for j=1:length(edgesofinterest)
        edgesofinterest{j} %Print parameter name to track progress

        %create copy so you can substitue values in a loop each time you take a
        %partial derivative of a different parameter
        eval([loops(i,:),'2 =',loops(i,:),';']);

        for k=1:paramofedge(j)-1
            %Substitute parameter values up until parameter you are taking the
            %partial derivative of
            eval([loops(i,:),'2 =
subs(',loops(i,:),'2,[',param{k},'],[',param{k},'_v]);']);
        end

        for k=paramofedge(j)+1:length(param)
            %Substitute parameter values for parameters after the one you are taking
the
            %partial derivative of
            eval([loops(i,:),'2 =
subs(',loops(i,:),'2,[',param{k},'],[',param{k},'_v]);']);
        end

        %Simplify resulting equation, getting ready to take partial derivative
        eval([loops(i,:),'2 = simplify(',loops(i,:),'2);']);

        %Take partial derivative of output equation with respect to single parameter
        eval(['TF_',loops(i,:),'_',edgesofinterest{j},' =
diff(',loops(i,:),'2,',edgesofinterest{j},');']);
        %Substitute parameter value back in now that gradient has been calculated
        eval(['TF_',loops(i,:),'_',edgesofinterest{j},' =
subs(TF_',loops(i,:),'_',edgesofinterest{j},',[',edgesofinterest{j},'],[',edgesofinter
est{j},'_v]);']);
        %Simplify this answer
        eval(['TF_',loops(i,:),'_',edgesofinterest{j},' =
simplify(TF_',loops(i,:),'_',edgesofinterest{j},');']);
        %Find numerator and denominator of partial derivative equation
        eval(['[N,D]=numden(TF_',loops(i,:),'_',edgesofinterest{j},');']);

        %Collect s terms you the partial derivative can be plotted as a transfer
        %function in a bode plot
        Ncoeffs = collect(N, s);
        Dcoeffs = collect(D, s);

        %Put coefficients into an array
        Nc = coeffs(Ncoeffs, s, 'All');
        Dc = coeffs(Dcoeffs, s, 'All');
```

```
        %Convert to type double
        Nc = double(Nc);
        Dc = double(Dc);

        %Create transfer function which reperesents the partial derivative of one
        %input-specific transfer function with respect to a single parameter
        eval(['DiffTF_',loops(i,:),'_',edgesofinterest{j},' = tf([Nc],[Dc]);']);
    end
end

%%
%%Sensitivity Bode Plots

%Plot sensitivity bode plots - partial derivatives of input-specific bode plot
for ii=1:length(loops)
    figure()

eval(['bodemag(DiffTF_',loops(ii,:),'_',param{1},',DiffTF_',loops(ii,:),'_',param{2},'
,DiffTF_',loops(ii,:),'_',param{3},',DiffTF_',loops(ii,:),'_',param{4},',"--
",DiffTF_',loops(ii,:),'_',param{5},',DiffTF_',loops(ii,:),'_',param{6},',DiffTF_',loo
ps(ii,:),'_',param{7},')']);
    legend(param)
    eval(['title("Sensitivity Bode Plot for ',loops(ii,:),' Loop")'])
end

title("Sensitivity Bode Plot for Input = Source Temperature")
title("Sensitivity Bode Plot for Input = Heat Load 1")
title("Sensitivity Bode Plot for Input = Heat Load 2")

%% Plot sensitivity bode plots comparing parameter 2 (C2) and 4 (C4)
figure()
subplot(1,3,1)
eval(['bodemag(DiffTF_',loops(1,:),'_',param{2},',DiffTF_',loops(1,:),'_',param{4},',"
--r")']);
title("Sensitivity Bode Plot for Input = Source Temperature")
legend(param{2},param{4})
subplot(1,3,2)
eval(['bodemag(DiffTF_',loops(2,:),'_',param{2},',DiffTF_',loops(2,:),'_',param{4},',"
--r")']);
title("Sensitivity Bode Plot for Input = Heat Load 1")
legend(param{2},param{4})
subplot(1,3,3)
eval(['bodemag(DiffTF_',loops(3,:),'_',param{2},',DiffTF_',loops(3,:),'_',param{4},',"
--r")']);
title("Sensitivity Bode Plot for Input = Heat Load 2")
legend(param{2},param{4})
%%
%%Sensitivity Plot
%Choose parameters to include in Sensitivity Plot
array ={param{1},param{2},param{3},param{4},param{5},param{6},param{7}};
%Note what the frequencies of the different inputs are
freq_des=[1,1,1];
%Note the amplitudes of the inputs
amp = [20,1,2];

magnitude = zeros(3,length(array));
summ = zeros(1,length(array));

for jj=1:3
```

```
    for kk=1:length(array)
        eval(['[mag,phase,wout] = bode(DiffTF_',loops(jj,:),'_',array{kk},');']);

        if freq_des(jj)==0
            magnitude(jj,kk) = 0;
        else
            magnitude(jj,kk) = interp1(wout(:),mag(:),freq_des(jj),'linear');
        end
    end
end

%Add the sensitivities multiplied by the input signal amplitude together
%This combines all the contributions to the output sensitivity of each of the
parameters
%from each input specific TF to yield the total sensitivity of the output to each
parameter
for kk=1:length(array)
        summ(kk)= magnitude(1,kk)*amp(1) + magnitude(2,kk)*amp(2) +
magnitude(3,kk)*amp(3);
end

figure()
xaxis=0:1;
y=[summ(1),summ(2),summ(3),summ(4),summ(5),summ(6),summ(7)];
plot(xaxis,y(1)*ones(size(xaxis)),xaxis,y(2)*ones(size(xaxis)),xaxis,y(3)*ones(size(xa
xis)),xaxis,y(4)*ones(size(xaxis)),"--
b",xaxis,y(5)*ones(size(xaxis)),xaxis,y(6)*ones(size(xaxis)),xaxis,y(7)*ones(size(xaxi
s)));
%plot(xaxis,y(1)*ones(size(xaxis)),xaxis,y(2)*ones(size(xaxis)),xaxis,y(3)*ones(size(x
axis)),xaxis,y(4)*ones(size(xaxis)));
legend(array)
%title("Sensitivity of All Branch TFs at 1, 1, 1 rad/s to Capacitance Parameters")
title("Sensitivity of All Branch TFs at 1, 1, 1 rad/s to All Parameters")
ylabel('d Output Signal Amplitude (C) / d parameter')
%title("Output Amplitude, Given Input Signals at 20,1,2, and 0.5 rad/s")
%title("Sensitivity of Output Amplitude to C2 and C4, Given Input Signals at 20,1,50,
and 0.5 rad/s")
```

A.3 ExampleSys2SensAnalysis

```
%%Setup Model Equations
%%This is Hard Coded for the Example System 2

%Create symbolic variables for each of the governing equation matrices for
%the components
syms s a1 a2 a3 a4 b1 b2
%Create symbolic variables for each of the parameters so we can take the
%partial derivatives of them
syms C1 C2 C3 C4 C5 C6 C7 C8 C9 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12

%Read in Input Values from model
temp_in = out.source1.data(); %Input 1 (Source Temp, degree C)
heatLoad1 = out.hl1.data(); %Input 2 (Heat Load 1, kW)
ambient = out.amb.data(); %Input 3 (Heat Load 2, kW)
heatLoad2 = out.hl2.data(); %Input 4 (Heat Load 3, kW)
heatLoad3 = out.hl3.data(); %Input 5 (Heat Load 4, kW)
```

```
%Gather Capacitance Data from Model for Cold Plate 1
Cp = 1000*Fuel_Cp;
U1 = cp1.HTC;
A1 = pi*cp1.D*cp1.L;
Cw1 = 1000*cp1.M_wall*cp1.Cp_wall; %Wall Capacitance
Cf1 = cp1.Ac*cp1.L*Fuel_Rho*Cp; %Fluid Capacitance

%Gather Capacitance Data from Model for Cold Plate 2
U2 = cp2.HTC;
A2 = pi*cp2.D*cp2.L;
Cw2 = 1000*cp2.M_wall*cp2.Cp_wall; %Wall Capacitance
Cf2 = cp2.Ac*cp2.L*Fuel_Rho*Cp; %Fluid Capacitance

%Gather Capacitance Data from Model for Cold Plate 3
U3 = cp3.HTC;
A3 = pi*cp3.D*cp3.L;
Cw3 = 1000*cp3.M_wall*cp3.Cp_wall; %Wall Capacitance
Cf3 = cp3.Ac*cp3.L*Fuel_Rho*Cp; %Fluid Capacitance

%Gather Capacitance Data from Model for Cold Plate 4
U4 = cp4.HTC;
A4 = pi*cp4.D*cp4.L;
Cw4 = 1000*cp4.M_wall*cp4.Cp_wall; %Wall Capacitance
Cf4 = cp4.Ac*cp4.L*Fuel_Rho*Cp; %Fluid Capacitance

%Gather Capacitance Data from Model for Junction
Cf5 = pi*j5.D^2*0.25*j5.L*Fuel_Rho*Cp; %Fluid Capacitance

%Find mass flow rate steady state value (even if mass flow is sinusoidal)
tenth=floor(length(out.m.data)*0.95)
mflow1=mean(out.m.data(tenth:end))

tenth=floor(length(out.m5.data)*0.95);
mflow2=mean(out.m5.data(tenth:end))

tenth=floor(length(out.m8.data)*0.95)
mflow3=mean(out.m8.data(tenth:end))

tenth=floor(length(out.m6.data)*0.95)
mflow4=mean(out.m6.data(tenth:end))

tenth=floor(length(out.m7.data)*0.95)
mflow5=mean(out.m7.data(tenth:end))

%number edges and capacitances like the graph numbering for edges and capacitances
C1_v = Cf1; %cp1 fluid cap
C2_v = Cw1; %cp1 wall cap
C3_v = Cf2; %cp2 fluid cap
C4_v = Cw2; %cp2 wall cap
C5_v = Cf3; %cp3 fluid cap
C6_v = Cw3; %cp3 wall cap
C7_v = Cf4; %cp4 fluid cap
C8_v = Cw4; %cp4 wall cap
C9_v = Cf5; %junction fluid cap

%Edge coefficients
E1_v = U1*A1; %convective edge between fluid and wall
E2_v = mflow1*Cp; %advective edge out of cp1 fluid
E3_v = mflow2*Cp; %advective edge into cp2 fluid
```

```
E4_v = U2*A2; %convective edge between fluid and wall
E5_v = mflow3*Cp; %advective edge into cp3 fluid
E6_v = U3*A3; %convective edge between fluid and wall
E7_v = mflow3*Cp; %advective edge into junction fluid
E8_v = mflow4*Cp; %advective edge into cp4 fluid
E9_v = U4*A4; %convective edge between fluid and wall
E10_v = mflow4*Cp; %advective edge into junction fluid
E11_v = mflow5*Cp; %advective edge out of system

%defining transfer function for Cold Plate 1
%Tin --> Eout
CP11_tf = (b1*s -a4*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP12_tf = (a2*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP21_tf = (a3*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP22_tf = (b2*s-a1*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));

%separate numerator from denominator
[N1,D1]=numden(CP11_tf);
[N2,D2]=numden(CP12_tf);
[N3,D3]=numden(CP21_tf);
[N4,D4]=numden(CP22_tf);

for j=1:4
    %collect coefficients in front of each s term
    eval(['Ncoeffs = collect(N',num2str(j),', s);']);
    eval(['Dcoeffs = collect(D',num2str(j),', s);']);

    %group coefficients into array
    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');

    %Replace a1, a2, a3, a4, b1, b2 terms with edge and capacitance values
    %('v' stands for value)
    Nc = subs(Nc,[a1,a2,a3,a4,b1,b2],[-E1_v/C1_v-E2_v/C1_v,E1_v/C2_v,E1_v/C1_v,-
E1_v/C2_v ,E2_v,1000]);
    Dc = subs(Dc,[a1,a2,a3,a4,b1,b2],[-E1_v/C1_v-E2_v/C1_v,E1_v/C2_v,E1_v/C1_v,-
E1_v/C2_v ,E2_v,1000]);

    %Convert to double and record for each transfer function
    eval(['Nc',num2str(j),' = double(Nc);']);
    eval(['Dc',num2str(j),' = double(Dc);']);
end

%Creating transfer function with real values
CP11_tf_v = tf([Nc1],[Dc1]);
CP12_tf_v = tf([Nc2],[Dc2]);
CP21_tf_v = tf([Nc3],[Dc3]);
CP22_tf_v = tf([Nc4],[Dc4]);

%Creating transfer function with symbolic variables
%this step is needed for symbolic differentiation
%Replace a1, a2, a3, a4, b1, b2 terms with edge and capacitance symbolic
%variables
CP11_tf = subs(CP11_tf,[a1,a2,a3,a4,b1,b2],[-E1/C1-E2/C1,E1/C2,E1/C1,-E1/C2,E2,1000]);
CP12_tf = subs(CP12_tf,[a1,a2,a3,a4,b1,b2],[-E1/C1-E2/C1,E1/C2,E1/C1,-E1/C2,E2,1000]);
CP21_tf = subs(CP21_tf,[a1,a2,a3,a4,b1,b2],[-E1/C1-E2/C1,E1/C2,E1/C1,-E1/C2,E2,1000]);
CP22_tf = subs(CP22_tf,[a1,a2,a3,a4,b1,b2],[-E1/C1-E2/C1,E1/C2,E1/C1,-E1/C2,E2,1000]);

%%
```

```
%defining transfer function for Cold Plate 2
%Tin --> Eout
R11_tf = (b1*s -a4*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
R12_tf = (a2*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
R21_tf = (a3*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
R22_tf = (b2*s-a1*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));

[N1,D1]=numden(R11_tf);
[N2,D2]=numden(R12_tf);
[N3,D3]=numden(R21_tf);
[N4,D4]=numden(R22_tf);

for j=1:4
    eval(['Ncoeffs = collect(N',num2str(j),', s);']);
    eval(['Dcoeffs = collect(D',num2str(j),', s);']);

    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');

    %These subs for a1..b2 change for each component type
    Nc = subs(Nc,[a1,a2,a3,a4,b1,b2],[-E4_v/C3_v-E3_v/C3_v,E4_v/C4_v,E4_v/C3_v,-
E4_v/C4_v ,E3_v,1000]);
    Dc = subs(Dc,[a1,a2,a3,a4,b1,b2],[-E4_v/C3_v-E3_v/C3_v,E4_v/C4_v,E4_v/C3_v,-
E4_v/C4_v ,E3_v,1000]);

    eval(['Nc',num2str(j),' = double(Nc);']);
    eval(['Dc',num2str(j),' = double(Dc);']);
end

%Creating transfer function with real values
R11_tf_v = tf([Nc1],[Dc1]);
R12_tf_v = tf([Nc2],[Dc2]);
R21_tf_v = tf([Nc3],[Dc3]);
R22_tf_v = tf([Nc4],[Dc4]);

%Creating transfer function with symbolic variables
%this step is needed for symbolic differentiation
R11_tf = subs(R11_tf,[a1,a2,a3,a4,b1,b2],[-E4/C3-E3/C3,E4/C4,E4/C3,-E4/C4,E3,1000]);
R12_tf = subs(R12_tf,[a1,a2,a3,a4,b1,b2],[-E4/C3-E3/C3,E4/C4,E4/C3,-E4/C4,E3,1000]);
R21_tf = subs(R21_tf,[a1,a2,a3,a4,b1,b2],[-E4/C3-E3/C3,E4/C4,E4/C3,-E4/C4,E3,1000]);
R22_tf = subs(R22_tf,[a1,a2,a3,a4,b1,b2],[-E4/C3-E3/C3,E4/C4,E4/C3,-E4/C4,E3,1000]);


%%
%defining transfer function for Cold Plate 3
%Tin --> Eout
CP33_tf = (b1*s -a4*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP34_tf = (a2*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP43_tf = (a3*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP44_tf = (b2*s-a1*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));

[N1,D1]=numden(CP33_tf);
[N2,D2]=numden(CP34_tf);
[N3,D3]=numden(CP43_tf);
[N4,D4]=numden(CP44_tf);

for j=1:4
    eval(['Ncoeffs = collect(N',num2str(j),', s);']);
    eval(['Dcoeffs = collect(D',num2str(j),', s);']);
```

```
    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');

    Nc = subs(Nc,[a1,a2,a3,a4,b1,b2],[-E6_v/C5_v-E3_v/C5_v,E6_v/C6_v,E6_v/C5_v,-
E6_v/C6_v ,E3_v,1000]);
    Dc = subs(Dc,[a1,a2,a3,a4,b1,b2],[-E6_v/C5_v-E3_v/C5_v,E6_v/C6_v,E6_v/C5_v,-
E6_v/C6_v ,E3_v,1000]);

    eval(['Nc',num2str(j),' = double(Nc);']);
    eval(['Dc',num2str(j),' = double(Dc);']);
end

%Creating transfer function with real values
CP33_tf_v = tf([Nc1],[Dc1]);
CP34_tf_v = tf([Nc2],[Dc2]);
CP43_tf_v = tf([Nc3],[Dc3]);
CP44_tf_v = tf([Nc4],[Dc4]);

%Creating transfer function with symbolic variables
%this step is needed for symbolic differentiation
CP33_tf = subs(CP33_tf,[a1,a2,a3,a4,b1,b2],[-E6/C5-E3/C5,E6/C6,E6/C5,-E6/C6,E3,1000]);
CP34_tf = subs(CP34_tf,[a1,a2,a3,a4,b1,b2],[-E6/C5-E3/C5,E6/C6,E6/C5,-E6/C6,E3,1000]);
CP43_tf = subs(CP43_tf,[a1,a2,a3,a4,b1,b2],[-E6/C5-E3/C5,E6/C6,E6/C5,-E6/C6,E3,1000]);
CP44_tf = subs(CP44_tf,[a1,a2,a3,a4,b1,b2],[-E6/C5-E3/C5,E6/C6,E6/C5,-E6/C6,E3,1000]);
%%
%defining transfer function for Cold Plate 4
%Tin --> Eout
CP55_tf = (b1*s -a4*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP56_tf = (a2*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP65_tf = (a3*b1)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));
CP66_tf = (b2*s-a1*b2)/(s^2 -(a1+a4)*s + (a1*a4-a2*a3));

[N1,D1]=numden(CP55_tf);
[N2,D2]=numden(CP56_tf);
[N3,D3]=numden(CP65_tf);
[N4,D4]=numden(CP66_tf);

for j=1:4
    eval(['Ncoeffs = collect(N',num2str(j),', s);']);
    eval(['Dcoeffs = collect(D',num2str(j),', s);']);

    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');

    Nc = subs(Nc,[a1,a2,a3,a4,b1,b2],[-E9_v/C7_v-(E2_v-
E3_v)/C7_v,E9_v/C8_v,E9_v/C7_v,-E9_v/C8_v,(E2_v-E3_v),1000]);
    Dc = subs(Dc,[a1,a2,a3,a4,b1,b2],[-E9_v/C7_v-(E2_v-
E3_v)/C7_v,E9_v/C8_v,E9_v/C7_v,-E9_v/C8_v,(E2_v-E3_v),1000]);

    eval(['Nc',num2str(j),' = double(Nc);']);
    eval(['Dc',num2str(j),' = double(Dc);']);
end

%Creating transfer function with real values
CP55_tf_v = tf([Nc1],[Dc1]);
CP56_tf_v = tf([Nc2],[Dc2]);
CP65_tf_v = tf([Nc3],[Dc3]);
CP66_tf_v = tf([Nc4],[Dc4]);
```

```
%Creating transfer function with symbolic variables
%this step is needed for symbolic differentiation
CP55_tf = subs(CP55_tf,[a1,a2,a3,a4,b1,b2],[-E9/C7-(E2-E3)/C7,E9/C8,E9/C7,-E9/C8,(E2-
E3),1000]);
CP56_tf = subs(CP56_tf,[a1,a2,a3,a4,b1,b2],[-E9/C7-(E2-E3)/C7,E9/C8,E9/C7,-E9/C8,(E2-
E3),1000]);
CP65_tf = subs(CP65_tf,[a1,a2,a3,a4,b1,b2],[-E9/C7-(E2-E3)/C7,E9/C8,E9/C7,-E9/C8,(E2-
E3),1000]);
CP66_tf = subs(CP66_tf,[a1,a2,a3,a4,b1,b2],[-E9/C7-(E2-E3)/C7,E9/C8,E9/C7,-E9/C8,(E2-
E3),1000]);

%%
%defining transfer function for Junction
%Tin --> Eout
J11_tf = (b1)/(s -a1);
J12_tf = (b2)/(s -a1);

[N1,D1]=numden(J11_tf);
[N2,D2]=numden(J12_tf);

for j=1:2
    eval(['Ncoeffs = collect(N',num2str(j),', s);']);
    eval(['Dcoeffs = collect(D',num2str(j),', s);']);

    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');

%ignore sink in tank
    Nc = subs(Nc,[a1,b1,b2],[-E2_v/C9_v,E3_v,E2_v-E3_v]);
    Dc = subs(Dc,[a1,b1,b2],[-E2_v/C9_v,E3_v,E2_v-E3_v]);

    eval(['Nc',num2str(j),' = double(Nc);']);
    eval(['Dc',num2str(j),' = double(Dc);']);
end

%Creating transfer function with real values
J11_tf_v = tf([Nc1],[Dc1]);
J12_tf_v = tf([Nc2],[Dc2]);

%Creating transfer function with symbolic variables
%this step is needed for symbolic differentiation
J11_tf = subs(J11_tf,[a1,b1,b2],[-(E2)/C9,E3,E2-E3]);
J12_tf = subs(J12_tf,[a1,b1,b2],[-(E2)/C9,E3,E2-E3]);

%%
%Sensitivity Analysis
%%Set up Transfer function equations

%Each of the inputs has their own transfer function equation
loops = ['sce';'hl1';'hl2';'hl3';'hl4'];

%Finding Transfer Function equations when the top branch temperature (f5) is the
output
eval([loops(1,:),' = (CP11_tf*(1/C1)*(R11_tf*(1/C3)*CP33_tf*(1/C5)))']); %source to f5
eval([loops(2,:),' = (CP12_tf*(1/C1)*R11_tf*(1/C3)*CP33_tf*(1/C5))']); %hl1 to f5
eval([loops(3,:),' = (R12_tf*(1/C3)*CP33_tf*(1/C5))']); %amb to f5
eval([loops(4,:),' = (CP34_tf*(1/C5))']); %hl2 to f5
% eval([loops(5,:),' = nothing']); %hl3 to f5
```

```
%Finding Transfer Function equations when the bottom branch temperature (f7) is the
output
eval([loops(1,:),' = (CP11_tf*(1/C1)*(CP55_tf*(1/C7)))']); %source to f7
eval([loops(2,:),' = (CP12_tf*(1/C1)*(CP55_tf*(1/C7)))']); %hl1 to f7
% eval([loops(3,:),' = nothing']); %amb to f7
% eval([loops(4,:),' = nothing']); %hl2 to f7
eval([loops(5,:),' = (CP56_tf*(1/C7))']); %hl3 to f7

%Finding Transfer Function equations when the output temperature (f9) is the output
eval([loops(1,:),' =
(CP11_tf*(1/C1)*(R11_tf*(1/C3)*CP33_tf*(1/C5)*J11_tf+CP55_tf*(1/C7)*J12_tf))*(1/C9)'])
; %source to out
eval([loops(2,:),' =
(CP12_tf*(1/C1)*(R11_tf*(1/C3)*CP33_tf*(1/C5)*J11_tf+CP55_tf*(1/C7)*J12_tf))*(1/C9)'])
; %hl1 to out
eval([loops(3,:),' = (R12_tf*(1/C3)*CP33_tf*(1/C5)*J11_tf)*(1/C9)']); %amb to out
eval([loops(4,:),' = (CP34_tf*(1/C5)*J11_tf)*(1/C9)']); %hl2 to out
eval([loops(5,:),' = (CP56_tf*(1/C7)*J12_tf*(1/C9))']); %hl3 to out

%Parameters to take the partial derivatives of- you can shorten this to
%just the capacitances
param =
{'C1';'C2';'C3';'C4';'C5';'C6';'C7';'C8';'C9';'E1';'E2';'E3';'E4';'E5';'E6';'E7';'E8';
'E9';'E10';'E11'}

for n=1:5
    %For input-specific system transfer function separate numerator from
    %denominator
    eval(['[N,D]=numden(',loops(n,:),');']);
    %Collect the s terms (since its in freq domain)
    Ncoeffs = collect(N, s);
    Dcoeffs = collect(D, s);
    %Gather coefficients in correct order to prepare for tf() function
    Nc = coeffs(Ncoeffs, s, 'All');
    Dc = coeffs(Dcoeffs, s, 'All');
    %Substitute all parameter symbolic variables for numeric values
    %Right now we are just plotted the TF equation and don't need them to
    %be symbolic since we're not taking the partial derivative yet
    for k=1:length(param)
        eval(['Nc = subs(Nc,[',param{k},'],[',param{k},'_v]);']);
        eval(['Dc = subs(Dc,[',param{k},'],[',param{k},'_v]);']);
    end
    %Convert to double
    Nc = double(Nc);
    Dc = double(Dc);
    %Create TF for each input-specific TF equation
    eval([loops(n,:),'_v = tf([Nc],[Dc]);']);
    %Plot numeric value of transfer function ('v' stands for value)
    %figure()
    %eval(['bode(',loops(n,:),'_v);']);
end
%%
%%Take partial derivatives

%Note parameters to take the partial derivatives of
edgesofinterest =  param;
paramofedge = linspace(1,length(param),length(param));

for i=1:length(loops)
```

```
    for j=1:length(edgesofinterest)
        edgesofinterest{j} %Print parameter name to track progress

        %create copy so you can substitue values in a loop each time you take a
        %partial derivative of a different parameter
        eval([loops(i,:),'2 =',loops(i,:),';']);

        for k=1:paramofedge(j)-1
            %Substitute parameter values up until parameter you are taking the
            %partial derivative of
            eval([loops(i,:),'2 =
subs(',loops(i,:),'2,[',param{k},'],[',param{k},'_v]);']);
        end

        for k=paramofedge(j)+1:length(param)
            %Substitute parameter values for parameters after the one you are taking
the
            %partial derivative of
            eval([loops(i,:),'2 =
subs(',loops(i,:),'2,[',param{k},'],[',param{k},'_v]);']);
        end

        %Simplify resulting equation, getting ready to take partial derivative
        eval([loops(i,:),'2 = simplify(',loops(i,:),'2);']);

        %Take partial derivative of output equation with respect to single parameter
        eval(['TF_',loops(i,:),'_',edgesofinterest{j},' =
diff(',loops(i,:),'2,',edgesofinterest{j},');']);
        %Substitute parameter value back in now that gradient has been calculated
        eval(['TF_',loops(i,:),'_',edgesofinterest{j},' =
subs(TF_',loops(i,:),'_',edgesofinterest{j},',[',edgesofinterest{j},'],[',edgesofinter
est{j},'_v]);']);
        %Simplify this answer
        eval(['TF_',loops(i,:),'_',edgesofinterest{j},' =
simplify(TF_',loops(i,:),'_',edgesofinterest{j},');']);
        %Find numerator and denominator of partial derivative equation
        eval(['[N,D]=numden(TF_',loops(i,:),'_',edgesofinterest{j},');']);

        %Collect s terms you the partial derivative can be plotted as a transfer
        %function in a bode plot
        Ncoeffs = collect(N, s);
        Dcoeffs = collect(D, s);

        %Put coefficients into an array
        Nc = coeffs(Ncoeffs, s, 'All');
        Dc = coeffs(Dcoeffs, s, 'All');

        %Convert to type double
        Nc = double(Nc);
        Dc = double(Dc);

        %Create transfer function which reperesents the partial derivative of one
        %input-specific transfer function with respect to a single parameter
        eval(['DiffTF_',loops(i,:),'_',edgesofinterest{j},' = tf([Nc],[Dc]);']);
    end
end

%%
%%Input Specific Transfer Functions
```

```
%Plot all input-specific bode plots
for ii=1:5%length(loops)
    figure()
    eval(['bode(',loops(ii,:),'_v);']);
    eval(['title("Transfer Function of System Output to Input ',num2str(ii),'")'])
end

%Plot input-specific bode plots in subplot
figure()
suptitle('Input - Specific System Transfer Functions')
subplot(2,3,1)
ii=1;
eval(['bodemag(',loops(ii,:),'_v);']);
title('System Output / Source Temperature', 'Interpreter', 'latex')
title('Top Branch Temperature / Source Temperature', 'Interpreter', 'latex')
title('Bottom Branch Temperature / Source Temperature', 'Interpreter', 'latex')
subplot(2,3,2)
ii=2;
eval(['bodemag(',loops(ii,:),'_v);']);
title('System Output / Heat Load 1', 'Interpreter', 'latex')
title('Top Branch Temperature / Heat Load 1', 'Interpreter', 'latex')
title('Bottom Branch Temperature / Heat Load 1', 'Interpreter', 'latex')
subplot(2,3,3)
ii=3;
eval(['bodemag(',loops(ii,:),'_v);']);
%eval(['plot(0,0);']);
title('System Output / Heat Load 2', 'Interpreter', 'latex')
title('Top Branch Temperature / Heat Load 2', 'Interpreter', 'latex')
title('Bottom Branch Temperature / Heat Load 2', 'Interpreter', 'latex')
subplot(2,3,4)
ii=4;
eval(['bodemag(',loops(ii,:),'_v);']);
%eval(['plot(0,0);']);
title('System Output / Heat Load 3', 'Interpreter', 'latex')
title('Top Branch Temperature / Heat Load 3', 'Interpreter', 'latex')
title('Bottom Branch Temperature / Heat Load 3', 'Interpreter', 'latex')
subplot(2,3,5)
ii=5;
eval(['bodemag(',loops(ii,:),'_v);']);
%eval(['plot(0,0);']);
title('System Output / Heat Load 4', 'Interpreter', 'latex')
title('Top Branch Temperature / Heat Load 4', 'Interpreter', 'latex')
title('Bottom Branch Temperature / Heat Load 4', 'Interpreter', 'latex')

%%
%%Sensitivity Bode Plot
%Plot sensitivity bode plots - partial derivatives of input-specific bode plot
figure()
%suptitle('Derivatives of Input - Specific System Transfer Functions: Sensitivity Bode
Plots')
subplot(2,3,1)
ii=1;
eval(['bodemag(DiffTF_',loops(ii,:),'_',param{1},',DiffTF_',loops(ii,:),'_',param{2},'
,DiffTF_',loops(ii,:),'_',param{3},',DiffTF_',loops(ii,:),'_',param{4},',DiffTF_',loop
s(ii,:),'_',param{5},',"--g",DiffTF_',loops(ii,:),'_',param{6},',"--
c",DiffTF_',loops(ii,:),'_',param{7},',"--m",DiffTF_',loops(ii,:),'_',param{8},',"--
r",DiffTF_',loops(ii,:),'_',param{9},',"--b");']);
title({'\underline {d ( System Output / Source Temperature )}','d parameter'},
'Interpreter', 'latex')
```

```
title({'\underline {d ( Top Branch Temperature / Source Temperature )}','d
parameter'}, 'Interpreter', 'latex')
title({'\underline {d ( Bottom Branch Temperature / Source Temperature )}','d
parameter'}, 'Interpreter', 'latex')
legend(param{1},param{2},param{3},param{4},param{5},param{6},param{7},param{8},param{9
});
subplot(2,3,2)
ii=2;
eval(['bodemag(DiffTF_',loops(ii,:),'_',param{1},',DiffTF_',loops(ii,:),'_',param{2},'
,DiffTF_',loops(ii,:),'_',param{3},',DiffTF_',loops(ii,:),'_',param{4},',DiffTF_',loop
s(ii,:),'_',param{5},',"--g",DiffTF_',loops(ii,:),'_',param{6},',"--
c",DiffTF_',loops(ii,:),'_',param{7},',"--m",DiffTF_',loops(ii,:),'_',param{8},',"--
r",DiffTF_',loops(ii,:),'_',param{9},',"--b");']);
title({'\underline {d ( Top Branch Temperature / Heat Load 1 )}','d parameter'},
'Interpreter', 'latex')
title({'\underline {d ( Bottom Branch Temperature / Heat Load 1 )}','d parameter'},
'Interpreter', 'latex')
title({'\underline {d ( System Output / Heat Load 1 )}','d parameter'}, 'Interpreter',
'latex')
legend(param{1},param{2},param{3},param{4},param{5},param{6},param{7},param{8},param{9
});
subplot(2,3,3)
ii=3;
eval(['bodemag(DiffTF_',loops(ii,:),'_',param{1},',DiffTF_',loops(ii,:),'_',param{2},'
,DiffTF_',loops(ii,:),'_',param{3},',DiffTF_',loops(ii,:),'_',param{4},',DiffTF_',loop
s(ii,:),'_',param{5},',"--g",DiffTF_',loops(ii,:),'_',param{6},',"--
c",DiffTF_',loops(ii,:),'_',param{7},',"--m",DiffTF_',loops(ii,:),'_',param{8},',"--
r",DiffTF_',loops(ii,:),'_',param{9},',"--b");']);
title({'\underline {d ( System Output / Heat Load 2 )}','d parameter'}, 'Interpreter',
'latex')
title({'\underline {d ( Top Branch Temperature / Heat Load 2 )}','d parameter'},
'Interpreter', 'latex')
title({'\underline {d ( Bottom Branch Temperature / Heat Load 2 )}','d parameter'},
'Interpreter', 'latex')
legend(param{1},param{2},param{3},param{4},param{5},param{6},param{7},param{8},param{9
});
subplot(2,3,4)
ii=4;
eval(['bodemag(DiffTF_',loops(ii,:),'_',param{1},',DiffTF_',loops(ii,:),'_',param{2},'
,DiffTF_',loops(ii,:),'_',param{3},',DiffTF_',loops(ii,:),'_',param{4},',DiffTF_',loop
s(ii,:),'_',param{5},',"--g",DiffTF_',loops(ii,:),'_',param{6},',"--
c",DiffTF_',loops(ii,:),'_',param{7},',"--m",DiffTF_',loops(ii,:),'_',param{8},',"--
r",DiffTF_',loops(ii,:),'_',param{9},',"--b");']);
title({'\underline {d ( System Output / Heat Load 3 )}','d parameter'}, 'Interpreter',
'latex')
title({'\underline {d ( Top Branch Temperature / Heat Load 3 )}','d parameter'},
'Interpreter', 'latex')
title({'\underline {d ( Bottom Branch Temperature / Heat Load 3 )}','d parameter'},
'Interpreter', 'latex')
legend('C1: CP1 Fluid','C2: CP1 Wall','C3: CP2 Fluid','C4: CP2 Wall','C5: CP3
Fluid','C6: CP3 Wall','C7: CP4 Fluid','C8: CP4 Wall','C9: J1 Fluid');
legend(param{1},param{2},param{3},param{4},param{5},param{6},param{7},param{8},param{9
});
subplot(2,3,5)
ii=5;
eval(['bodemag(DiffTF_',loops(ii,:),'_',param{1},',DiffTF_',loops(ii,:),'_',param{2},'
,DiffTF_',loops(ii,:),'_',param{3},',DiffTF_',loops(ii,:),'_',param{4},',DiffTF_',loop
s(ii,:),'_',param{5},',"--g",DiffTF_',loops(ii,:),'_',param{6},',"--
c",DiffTF_',loops(ii,:),'_',param{7},',"--m",DiffTF_',loops(ii,:),'_',param{8},',"--
r",DiffTF_',loops(ii,:),'_',param{9},',"--b");']);
```

```
title({'\underline {d ( System Output / Heat Load 4 )}','d parameter'}, 'Interpreter',
'latex')
title({'\underline {d ( Top Branch Temperature / Heat Load 4 )}','d parameter'},
'Interpreter', 'latex')
title({'\underline {d ( Bottom Branch Temperature / Heat Load 4 )}','d parameter'},
'Interpreter', 'latex')
legend('C1: CP1 Fluid','C2: CP1 Wall','C3: CP2 Fluid','C4: CP2 Wall','C5: CP3
Fluid','C6: CP3 Wall','C7: CP4 Fluid','C8: CP4 Wall','C9: J1 Fluid');
%%
%%Sensitivity Plot
%Choose parameters to include in Sensitivity Plot
array
={param{1},param{2},param{3},param{4},param{5},param{6},param{7},param{8},param{9}};
%Note what the frequencies of the different inputs are
freq_des=[0,1,0.5,0,0];
%Note the amplitudes of the inputs
amp = [20,1,1,2,5];

magnitude = zeros(5,length(array));
summ = zeros(1,length(array));

for jj=1:5
    for kk=1:length(array)
        eval(['[mag,phase,wout] = bode(DiffTF_',loops(jj,:),'_',array{kk},');']);

        if freq_des(jj)==0
            magnitude(jj,kk) = 0;
        else
            magnitude(jj,kk) = interp1(wout(:),mag(:),freq_des(jj),'linear');
        end
    end
end

%Add the sensitivities multiplied by the input signal amplitude together
%This combines all the contributions to the output sensitivity of each of the
parameters
%from each input specific TF to yield the total sensitivity of the output to each
parameter
for kk=1:length(array)
    summ(kk)= magnitude(1,kk)*amp(1) + magnitude(2,kk)*amp(2) + magnitude(3,kk)*amp(3)
+ magnitude(4,kk)*amp(4) + magnitude(5,kk)*amp(5);
end

figure()
xaxis=0:1;
y=[summ(1),summ(2),summ(3),summ(4),summ(5),summ(6),summ(7),summ(8),summ(9)];
plot(xaxis,y(1)*ones(size(xaxis)),xaxis,y(2)*ones(size(xaxis)),xaxis,y(3)*ones(size(xa
xis)),xaxis,y(4)*ones(size(xaxis)),xaxis,y(5)*ones(size(xaxis)),"--
g",xaxis,y(6)*ones(size(xaxis)),"--c",xaxis,y(7)*ones(size(xaxis)),"--
m",xaxis,y(8)*ones(size(xaxis)),"--r",xaxis,y(9)*ones(size(xaxis)),"--b");
legend(param{1},param{2},param{3},param{4},param{5},param{6},param{7},param{8},param{9
})
title("Sensitivity of Bottom Branch to Parameters with 0,1,0.5,0,0 rad/s Inputs")
ylabel('d Output Signal Amplitude (C) / d parameter')
```

A.4 Numerical_Verification

```
%Specify which parameter should be perturbed
%For Example 1
add =
{'cp1.M_wall','cp1.coeff','cp2.M_wall','cp2.coeff','cp1.HTC','sourcemdot','cp2.HTC'}
% add = {'cp1.M_wall','cp2.coeff','cp2.M_wall','cp3.coeff'}

%For Example 2
% add =
{'cp1.coeff','cp1.M_wall','cp2.coeff','cp2.M_wall','cp3.coeff','cp3.M_wall','cp4.coeff
','cp4.M_wall','j5.coeff'}

%Specify what factors should be added to the capacitance of edge values for
perturbation
values = [0,0.02,0.05,0.1,0.5,1,2]

%Specify the name of the Simulink Model
%simName = 'Ch2Example_NoTank'
%simName = 'Ch2Example_Single_Controller'
simName = 'SimpleCh2Example'

%Quick run-through of simulation with nominal values
set_param(simName,'SimulationCommand','Start')

tic %Start time

%Running the simulation in this way will allow it to finish and send data back before
beginning the next one
set_param(simName,'SimulationCommand','Start') %begin simulation
while(~strcmp( get_param(simName,'SimulationStatus') , 'stopped'))
    pause(0.01);
end

toc %End time

%Record Nominal values
%Calculate maximum and minimum values for the last 10% of simulation (steady state
amplitude)
tenth=floor(length(out.systemoutput.data)*0.9) %system output temp
up=max(out.systemoutput.data(tenth:end))
down=min(out.systemoutput.data(tenth:end))

tenth2=floor(length(out.systemoutput2.data)*0.9) %top branch output temp
up2=max(out.systemoutput2.data(tenth:end))
down2=min(out.systemoutput2.data(tenth:end))

tenth3=floor(length(out.systemoutput3.data)*0.9) %bottom branch output temp
up3=max(out.systemoutput3.data(tenth:end))
down3=min(out.systemoutput3.data(tenth:end))

%These are commented out because they only apply to the controller case
%{
nom_cinput_1=sum(abs(out.cinput1.data())); %control input for controller 1
nom_cinput_2=sum(abs(out.cinput2.data())); %control input for controller 2
nom_cinput=nom_cinput_1+nom_cinput_2 %sum of control input (although control input 1
is more relevant)
```

```
nom_cinput_1_data=out.cinput1.data(); %record the data values, not just the summed
integral
nom_cinput_2_data=out.cinput2.data(); %record the data values, not just the summed
integral

nom_amp=(up - down)/2; %system output temp amplitude
nom_amp2=(up2 - down2)/2; %top branch output temp amplitude
nom_amp3=(up3 - down3)/2; %bottom branch output temp amplitude

nom_totalerror1=sum(abs(out.error1.data())); %control error for controller 1
nom_totalerror2=sum(abs(out.error2.data())); %control error for controller 2
%}
%%
%Begin perturbation simulations
figure()

for k=1:length(add) %Go through each parameter

    for j = 1:length(values) %Go through each increment
        run('Setup_Component_Params_Sensitivity7') %Reset parameter values
        clear out; %clear the output from the previous simulation

        eval([add{k},' = ',add{k},'+',num2str(values(j)),';']); %perturb the parameter

        tic %start time

        set_param(simName,'SimulationCommand','Start') %start simulation

        %Running the simulation in this way will allow it to finish and send data back
before beginning the next one
        while(~strcmp( get_param(simName,'SimulationStatus') , 'stopped'))
            pause(0.01);
        end

        toc %end time

        %Record data to sift through later if desired
        eval(['amp_data_',num2str(k),'_',num2str(j),' = out.systemoutput.data();']);

        %Find max and min values of last 10% of simulation
        %Find steady state amplitude of output temp
        tenth=floor(length(out.systemoutput.data)*0.9);
        up=max(out.systemoutput.data(tenth:end));
        down=min(out.systemoutput.data(tenth:end));

        %Find steady state amplitude of top branch temp
        tenth2=floor(length(out.systemoutput2.data)*0.9);
        up2=max(out.systemoutput2.data(tenth:end));
        down2=min(out.systemoutput2.data(tenth:end));

        %Find steady state amplitude of bottom branch temp
        tenth3=floor(length(out.systemoutput3.data)*0.9);
        up3=max(out.systemoutput3.data(tenth:end));
        down3=min(out.systemoutput3.data(tenth:end));

        %Calculate steady state amplitude
        amp=(up - down)/2; %system output temp amplitude
        amp2=(up2 - down2)/2; %top branch temp amplitude
```

```
        amp3=(up3 - down3)/2; %bottom branch temp amplitude
        %Put amplitude in an array for plotting later
        eval(['amp_array_',num2str(k),'(j) = amp;']);
        eval(['amp2_array_',num2str(k),'(j) = amp2;']);
        eval(['amp3_array_',num2str(k),'(j) = amp3;']);

        %These are commented below because they pertain to the controller model
        %{
        cinput_1=sum(abs(out.cinput1.data())); %control input for controller 1
        cinput_2=sum(abs(out.cinput2.data())); %control input for controller 2
        cinput=cinput_1+cinput_2; %sum of control input (although control input 1 is
more relevant)

        totalerror1=sum(abs(out.error1.data())); %sum of error for controller 1 over
simulation
        totalerror2=sum(abs(out.error2.data())); %sum of error for controller 2 over
simulation

        %Find steady state amplitude of control input 1
        ctenth=floor(length(out.cinput1.data)*0.9);
        cup=max(out.cinput1.data(ctenth:end));
        cdown=min(out.cinput1.data(ctenth:end));

        %Find steady state amplitude of control input 2
        ctenth2=floor(length(out.cinput2.data)*0.9);
        cup2=max(out.cinput2.data(ctenth2:end));
        cdown2=min(out.cinput2.data(ctenth2:end));

        %Find steady state amplitude of control error 1
        etenth=floor(length(out.error1.data)*0.9);
        eup=max(out.error1.data(etenth:end));
        edown=min(out.error1.data(etenth:end));

        %Find steady state amplitude of control error 2
        etenth2=floor(length(out.error2.data)*0.9);
        eup2=max(out.error2.data(etenth2:end));
        edown2=min(out.error2.data(etenth2:end));

        camp=(cup - cdown)/2; %control input 1 amplitude
        camp2=(cup2 - cdown2)/2; %control input 2 amplitude

        eamp=(eup - edown)/2; %control error 1 amplitude
        eamp2=(eup2 - edown2)/2; %control error 2 amplitude

        campboth=camp+camp2; %sum control inputs for both controller 1 and 2
        eampboth=eamp+eamp2; %sum control errors for both controller 1 and 2

        %Record values in arrays for plotting
        eval(['cinput_1_array_',num2str(k),'(j) = cinput_1;']);
        eval(['cinput_2_array_',num2str(k),'(j) = cinput_2;']);
        eval(['cinput_array_',num2str(k),'(j) = cinput;']);

        eval(['totalerror1_array_',num2str(k),'(j) = totalerror1;']);
        eval(['totalerror2_array_',num2str(k),'(j) = totalerror2;']);

        eval(['camp_array_',num2str(k),'(j) = camp;']);
        eval(['camp2_array_',num2str(k),'(j) = camp2;']);
```

```
        eval(['eamp_array_',num2str(k),'(j) = eamp;']);
        eval(['eamp2_array_',num2str(k),'(j) = eamp2;']);
        %}
    end
    %Plot how the parameter perturbations affect the steady state amplitude for each
parameter as you go
    eval(['plot(values,amp_array_',num2str(k),');'])

    hold on
end
hold off
run('Setup_Component_Params_Sensitivity7') %reset parameter values
%%
%Plot numerical verification plot
figure()
%Plot steady state amplitudes
plot(values,amp_array_1,values,amp_array_2,values,amp_array_3,values,amp_array_4,value
s,amp_array_5,"--g",values,amp_array_6,"--c",values,amp_array_7,'--
m',values,amp_array_8,'--r',values,amp_array_9,'--b')
%plot(values,amp2_array_1,values,amp2_array_2,values,amp2_array_3,values,amp2_array_4,
values,amp2_array_5,"--g",values,amp2_array_6,"--c",values,amp2_array_7,'--
m',values,amp2_array_8,'--r',values,amp2_array_9,'--b')
%plot(values,amp3_array_1,values,amp3_array_2,values,amp3_array_3,values,amp3_array_4,
values,amp3_array_5,"--g",values,amp3_array_6,"--c",values,amp3_array_7,'--
m',values,amp3_array_8,'--r',values,amp3_array_9,'--b')

%plot(values,camp_array_1,values,camp_array_2,values,camp_array_3,values,camp_array_4,
values,camp_array_5,"--g",values,camp_array_6,"--c",values,camp_array_7,'--
m',values,camp_array_8,'--r',values,camp_array_9,'--b')
%plot(values,camp2_array_1,values,camp2_array_2,values,camp2_array_3,values,camp2_arra
y_4,values,camp2_array_5,"--g",values,camp2_array_6,"--c",values,camp2_array_7,'--
m',values,camp2_array_8,'--r',values,camp2_array_9,'--b')
%plot(values,eamp_array_1,values,eamp_array_2,values,eamp_array_3,values,eamp_array_4,
values,eamp_array_5,"--g",values,eamp_array_6,"--c",values,eamp_array_7,'--
m',values,eamp_array_8,'--r',values,eamp_array_9,'--b')
%plot(values,eamp2_array_1,values,eamp2_array_2,values,eamp2_array_3,values,eamp2_arra
y_4,values,eamp2_array_5,"--g",values,eamp2_array_6,"--c",values,eamp2_array_7,'--
m',values,eamp2_array_8,'--r',values,eamp2_array_9,'--b')

legend('C1: CP1 Fluid','C2: CP1 Wall','C3: CP2 Fluid','C4: CP2 Wall','C5: CP3
Fluid','C6: CP3 Wall','C7: CP4 Fluid','C8: CP4 Wall','C9: J1 Fluid');

%legend('Cp1 Wall Mass','Cp1 Fluid Cap','Cp2 Wall Mass','Cp2 Fluid Cap','Cp1
HTC','Source Mdot In','Cp2 HTC')

title("Steady State Amplitude of Output Signal Temperature With Parameter Perturbation
with 0,1,0.5,0,0 rad/s Inputs")
% title("Steady State Amplitude of Top Branch Temperature With Parameter Perturbation
with 0,1,0.5,0,0 rad/s Inputs")
% title("Steady State Amplitude of Bottom Branch Temperature With Parameter
Perturbation with 0,1,0.5,0,0 rad/s Inputs")

xlabel('Factor added to Vertex Capacitances')

ylabel('Steady State Amplitude of Both Branches')
ylabel('Steady State Amplitude of Top Branch')
ylabel('Steady State Amplitude of Bottom Branch')
```

A.5 Numerical_Verification_CoDesign

```
%Specify which parameters will be perturbed by 1 or by 0.5
bigadd = {'cp1.M_wall','cp2.coeff','cp2.M_wall','cp3.coeff'}
smalladd = {'cp1.M_wall','cp2.coeff','cp2.M_wall','cp3.coeff'}

%Nominal control gains
kp1=3;
kp2=3;

%Simulink Model Name
simName = 'Ch2Example_Single_Controller'

tic %begin time

set_param(simName,'SimulationCommand','Start') %begin simulation

%Running the simulation in this way will allow it to finish and send data back before
beginning the next one
while(~strcmp( get_param(simName,'SimulationStatus') , 'stopped'))
    pause(0.01);
end

toc %end time

%Specify the controller gains to sweep through
gains = [2.5,3,3.5,4,4.5];

%Create matrices for plotting later
errormat=zeros(length(gains)*length(bigadd)); %control error plot
inputmat=zeros(length(gains)*length(bigadd)); %control input plot

for nn=1:length(gains) %sweep through the gains for controller 1
    kp1=gains(nn); %set controller 1 gain
    for mm=1:length(gains) %sweep through the gains for controller 2
        kp2=gains(mm); %set controller 2 gain
        for k=1:length(bigadd) %sweep through large parameter perturbations
            parameters{k} %print to keep track
            %eval(['finalVal_array_',num2str(k),' = ones(1,length(values));']);
            for j = 1:length(smalladd) %sweep through small parameter perturbations

                run('Setup_Component_Params_Sensitivity7') %reset parameter values
                clear out; %clear outputs from previous simulation
                eval([bigadd{k},' = ',bigadd{k},'+1;']); %increment large pertrub
parameter by 1
                eval([smalladd{j},' = ',smalladd{j},'+0.5;']); %increment small
pertrub parameter by 0.5

                tic %begin time
                set_param(simName,'SimulationCommand','Start') %start simulation
                while(~strcmp( get_param(simName,'SimulationStatus') , 'stopped'))
                    pause(0.01);
                end
                toc %end time

                %Find steady state amplitude of output temp
                tenth=floor(length(out.systemoutput.data)*0.9);
                up=max(out.systemoutput.data(tenth:end));
```

```
                down=min(out.systemoutput.data(tenth:end));

                %Find steady state amplitude of top branch temp
                tenth2=floor(length(out.systemoutput2.data)*0.9);
                up2=max(out.systemoutput2.data(tenth:end));
                down2=min(out.systemoutput2.data(tenth:end));

                %Find steady state amplitude of bottom branch temp
                tenth3=floor(length(out.systemoutput3.data)*0.9);
                up3=max(out.systemoutput3.data(tenth:end));
                down3=min(out.systemoutput3.data(tenth:end));

                cinput_1=sum(abs(out.cinput1.data())); %control input for controller 1
                cinput_2=sum(abs(out.cinput2.data())); %control input for controller 2
                cinput=cinput_1+cinput_2; %sum of control input (although control
input 1 is more relevant)

                %Record control input data so it can be referred to and plotted later

eval(['cinput_1_data_kp1_',num2str(nn),'_kp2_',num2str(mm),'_',num2str(k),'_',num2str(
j),' = out.cinput1.data();']);

eval(['cinput_2_data_kp1_',num2str(nn),'_kp2_',num2str(mm),'_',num2str(k),'_',num2str(
j),' = out.cinput2.data();']);

                amp=(up - down)/2; %system output temp amplitude
                amp2=(up2 - down2)/2; %top branch temp amplitude
                amp3=(up3 - down3)/2; %bottom branch temp amplitude

                totalerror1=sum(abs(out.error1.data())); %sum of error for controller
1 over simulation
                totalerror2=sum(abs(out.error2.data())); %sum of error for controller
2 over simulation

                %Find steady state amplitude of control input 1
                ctenth=floor(length(out.cinput1.data)*0.9);
                cup=max(out.cinput1.data(ctenth:end));
                cdown=min(out.cinput1.data(ctenth:end));

                %Find steady state amplitude of control input 2
                ctenth2=floor(length(out.cinput2.data)*0.9);
                cup2=max(out.cinput2.data(ctenth2:end));
                cdown2=min(out.cinput2.data(ctenth2:end));

                %Find steady state amplitude of control error 1
                etenth=floor(length(out.error1.data)*0.9);
                eup=max(out.error1.data(etenth:end));
                edown=min(out.error1.data(etenth:end));

                %Find steady state amplitude of control error 2
                etenth2=floor(length(out.error2.data)*0.9);
                eup2=max(out.error2.data(etenth2:end));
                edown2=min(out.error2.data(etenth2:end));

                camp=(cup - cdown)/2; %control input 1 amplitude
                camp2=(cup2 - cdown2)/2; %control input 2 amplitude

                eamp=(eup - edown)/2; %control error 1 amplitude
                eamp2=(eup2 - edown2)/2; %control error 2 amplitude
```

```
                campboth=camp+camp2; %sum control inputs for both controller 1 and 2
                eampboth=eamp+eamp2; %sum control errors for both controller 1 and 2

                xpos= j+4*(nn-1) %note x axis position for simulation location on grid
plot
                ypos= length(bigadd)*length(gains)-(k-1+4*(mm-1)) %note y axis
position for simulation location on grid plot
                inputmat(ypos,xpos) = campboth; %populate summed control input matrix
                errormat(ypos,xpos) = eampboth; %populate summed control error matrix
                inputmat1(ypos,xpos) = camp; %populate control input 1 matrix
                errormat1(ypos,xpos) = eamp; %populate control error 1 matrix
                inputmat2(ypos,xpos) = camp2; %populate control input 2 matrix
                errormat2(ypos,xpos) = eamp2; %populate control error 2 matrix
            end
        end
    end
end

%reset parameter values
run('Setup_Component_Params_Sensitivity7')

%% Plot how the control input compares for parameter perturbations of CP Wall Masses
timeplot = 1:length(Mwall4_6_cinput_1_data_15_2)

figure()
plot(timeplot,Mwall4_6_cinput_1_data_15_2)
hold on
plot(timeplot,Mwall1_6_cinput_1_data_15_2)
plot(timeplot,Mwall2_6_cinput_1_data_15_2)
plot(timeplot,nom_cinput_1_data)
hold off
```

A.6 Plot_CoDesign

```
figure()
%imagesc(errormat1(13:16,5:8)); %Plot for Nominal Controller

%Normalize control error
max1error=max(max(errormat1(:,:)));
max2error=max(max(errormat2(:,:)));
errormat1plot(:,:)=errormat1(:,:)/max1error;
errormat2plot(:,:)=errormat2(:,:)/max2error;
%Sum of normalized control error
imagesc(errormat1plot(:,:)+errormat2plot(:,:));

%Control input plots
%imagesc(inputmat1(:,:));
%imagesc(inputmat2(:,:));

colormap(jet);
c=colorbar;

%Set colorbar limits for Nominal Controller Plot
%caxis([0.031 0.057]);

c.Label.String = 'Sum of Control Error Amplitudes';
%c.Label.String = 'Control Input 1 Amplitude';
```

```
%c.Label.String = 'Control Input 2 Amplitude';

title('Sum of Steady State Error Amplitudes','FontSize',12);
%title('Sum of Steady State Control Input Amplitude','FontSize',12);
%title('Steady State Control Input 1 Amplitude','FontSize',12);
%title('Steady State Control Error 2 Amplitude','FontSize',12);

for i=1:length(errormat1)
    total=mod(i-1,4)+1;
    reverse=4-(mod(i-1,4));
    text(i-0.3,20,num2str(total));
     %if reverse==3 %Make text white if block is too dark
         %text(1-0.3,i,num2str(reverse),'FontSize',12,'FontWeight','bold','Color','w')
     %else %usually make text black
         text(1-0.3,i,num2str(reverse),'FontSize',12,'FontWeight','bold')
     %end
end

%Text for plot scheme on Nominal Controller Plot
% text(0.7,4,'Large 1','FontSize',14)
% text(1.7,4,{'Large 1','Small 2'},'FontSize',14)
% text(2.7,4,{'Large 1','Small 3'},'FontSize',14)
% text(3.7,4,{'Large 1','Small 4'},'FontSize',14)
%
% text(0.7,3,{'Large 2','Small 1'},'FontSize',14)
% text(1.7,3,'Large 2','FontSize',14)
% text(2.7,3,{'Large 2','Small 3'},'FontSize',14)
% text(3.7,3,{'Large 2','Small 4'},'FontSize',14)
%
% text(0.7,2,{'Large 3','Small 1'},'FontSize',14)
% text(1.7,2,{'Large 3','Small 2'},'FontSize',14)
% text(2.7,2,'Large 3','FontSize',14)
% text(3.7,2,{'Large 3','Small 4'},'FontSize',14)
%
% text(0.7,1,{'Large 4','Small 1'},'FontSize',14)
% text(1.7,1,{'Large 4','Small 2'},'FontSize',14)
% text(2.7,1,{'Large 4','Small 3'},'FontSize',14)
% text(3.7,1,'Large 4','FontSize',14)

set(gca,'xticklabel',[])
set(gca,'yticklabel',[])

% ylabel('Kp2 gain','FontSize',16);
% xlabel('Kp1 gain','FontSize',16);

%%Add grid with separate ticks than axis labels
ax1 = gca;

ax4 = axes('Position',ax1.Position,...
  'XAxisLocation','bottom',...
  'YAxisLocation','left',...
  'Color','none',...
  'Ylim',ax1.YLim,...
  'XLim',ax1.XLim,...
  'TickLength',[0 0],...
  'YTick', [(ax1.YLim(1))+2:4:(ax1.YLim(2))-2], ... % so text aligns with block
  'XTick', [(ax1.XLim(1))+2:4:(ax1.XLim(2))-2],  ... % so text aligns with block
  'YTickLabel', [2.5,3,3.5,4,4.5],  ... %controller 2 gains
  'XTickLabel', [2.5,3,3.5,4,4.5]  ); %controller 1 gains
ax4.GridColor = [0.9 0.9 0.9];
```

```
%ax4.GridColor = [1 1 1];
ax4.GridLineStyle = 'none';
ax4.GridAlpha = 1;

ylabel('Kp2 gain','FontSize',12);
xlabel('Kp1 gain','FontSize',12);

ax1 = gca;
ax2 = axes('Position',ax1.Position,...
  'XAxisLocation','bottom',...
  'YAxisLocation','left',...
  'Color','none',...
  'Ylim',ax1.YLim,...
  'XLim',ax1.XLim,...
  'TickLength',[0 0],...
  'YTick', [(ax1.YLim(1)):1:(ax1.YLim(2))], ...
  'XTick', [(ax1.XLim(1)):1:(ax1.XLim(2))],  ...
  'YTickLabel', [],  ...
  'XTickLabel', []  );
%ax2.GridColor = [0.9 0.9 0.9];
ax2.GridColor = [0 0 0];
ax2.GridLineStyle = '--';
ax2.GridAlpha = 0.5;
grid on

ax3 = axes('Position',ax1.Position,...
  'XAxisLocation','bottom',...
  'YAxisLocation','left',...
  'Color','none',...
  'Ylim',ax1.YLim,...
  'XLim',ax1.XLim,...
  'TickLength',[0 0],...
  'YTick', [(ax1.YLim(1)):4:(ax1.YLim(2))], ...
  'XTick', [(ax1.XLim(1)):4:(ax1.XLim(2))],  ...
  'YTickLabel', [],  ...
  'XTickLabel', []  );
%ax3.GridColor = [0.9 0.9 0.9];
ax3.GridColor = [0 0 0];
ax3.GridLineStyle = '-';
ax3.GridAlpha = 1;
ax3.LineWidth = 2;
grid on

linkaxes([ax2 ax3], 'xy')
grid on
set(gcf,'CurrentAxes',ax1);
```

A.7 Plot_Control_Input

```
%% Plot Control Input 1 (Need to set nominalset=out, suboptimalset=out etc after
simulation run to save that data)
figure()
plot(nominalset.cinput1.time(),nominalset.cinput1.data())
hold on
plot(suboptimalset.cinput1.time(),suboptimalset.cinput1.data())
plot(optimalset.cinput1.time(),optimalset.cinput1.data())
%plot(out.cinput1.time(),out.cinput1.data())
legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Control Input for Controller 1');
```

```
ylabel('Control Input 1');
xlabel('Time (s)');
hold off
%% Plot Control Error 1 (Need to set nominalset=out, suboptimalset=out etc after
simulation run to save that data)
figure()
plot(nominalset.error1.time(),nominalset.error1.data())
hold on
plot(suboptimalset.error1.time(),suboptimalset.error1.data())
plot(optimalset.error1.time(),optimalset.error1.data())
legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Error for Controller 1');
ylabel('Control Error 1');
xlabel('Time (s)');
hold off
%% Plot Control Error 2 (Need to set nominalset=out, suboptimalset=out etc after
simulation run to save that data)
figure()
plot(nominalset.error2.time(),nominalset.error2.data())
hold on
plot(suboptimalset.error2.time(),suboptimalset.error2.data())
plot(optimalset.error2.time(),optimalset.error2.data())
legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Error for Controller 2');
ylabel('Control Error 2');
xlabel('Time (s)');
hold off

%% Plot Control Input 1 Subplot (Need to set nominalset=out, suboptimalset=out etc
after simulation run to save that data)
figure()
subplot(1,2,1)
plot(nominalset.cinput1.time(),nominalset.cinput1.data())
hold on
plot(suboptimalset.cinput1.time(),suboptimalset.cinput1.data())
plot(optimalset.cinput1.time(),optimalset.cinput1.data())
%plot(out.cinput1.time(),out.cinput1.data())
legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Control Input for Controller 1');
ylabel('Control Input 1');
xlabel('Time (s)');
hold off
subplot(1,2,2)
plot(nominalset.cinput1.time(13800:14000),nominalset.cinput1.data(13800:14000))
hold on
plot(suboptimalset.cinput1.time(13800:14000),suboptimalset.cinput1.data(13800:14000))
plot(optimalset.cinput1.time(13800:14000),optimalset.cinput1.data(13800:14000))
%plot(out.cinput1.time(),out.cinput1.data())
%legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Control Input for Controller 1 Zoomed In');
ylabel('Control Input 1');
xlabel('Time (s)');
hold off
%% Plot Control Error 1 Subplot
figure()
subplot(1,2,1)
plot(nominalset.error1.time(),nominalset.error1.data())
hold on
plot(suboptimalset.error1.time(),suboptimalset.error1.data())
plot(optimalset.error1.time(),optimalset.error1.data())
legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
```

```
title('Error for Controller 1');
ylabel('Control Error 1');
xlabel('Time (s)');
hold off
subplot(1,2,2)
plot(nominalset.error1.time(13800:14000),nominalset.error1.data(13800:14000))
hold on
plot(suboptimalset.error1.time(13800:14000),suboptimalset.error1.data(13800:14000))
plot(optimalset.error1.time(13800:14000),optimalset.error1.data(13800:14000))
%legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Error for Controller 1 Zoomed In');
ylabel('Control Error 1');
xlabel('Time (s)');
hold off
%% Plot Control Error 2 Subplot
figure()
subplot(1,2,1)
plot(nominalset.error2.time(),nominalset.error2.data())
hold on
plot(suboptimalset.error2.time(),suboptimalset.error2.data())
plot(optimalset.error2.time(),optimalset.error2.data())
legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Error for Controller 2');
ylabel('Control Error 2');
xlabel('Time (s)');
hold off
subplot(1,2,2)
plot(nominalset.error2.time(13800:14000),nominalset.error2.data(13800:14000))
hold on
plot(suboptimalset.error2.time(13800:14000),suboptimalset.error2.data(13800:14000))
plot(optimalset.error2.time(13800:14000),optimalset.error2.data(13800:14000))
%legend('Nominal Configuration','Lowest Control Input','Lowest Control Error')
title('Error for Controller 2 Zoomed In');
ylabel('Control Error 2');
xlabel('Time (s)');
hold off
%% Plot Control Inputs Subplot
figure()
subplot(2,3,1)
plot(input1step.time(),input1step.data(:,1))
hold on
plot(input1step.time(),input1step.data(:,2))
plot(input1step.time(),input1step.data(:,3))
plot(input1step.time(),input1step.data(:,4))
plot(input1step.time(),input1step.data(:,5),"--g")
plot(input1step.time(),input1step.data(:,6),"--c")
plot(input1step.time(),input1step.data(:,7),"--m")
plot(input1step.time(),input1step.data(:,8),"--r")
plot(input1step.time(),input1step.data(:,9),"--b")
hold off
%title('Heat Load 1 and 2 Step Inputs where Heat Load 1 is Double', 'Interpreter',
'latex')
title('Temperature of Fluid Leaving the System', 'Interpreter', 'latex')
xlabel('Time(s)')
ylabel('Temperature at Output (C)')

subplot(2,3,2)
plot(input2step.time(),input2step.data(:,1))
hold on
plot(input2step.time(),input2step.data(:,2))
plot(input2step.time(),input2step.data(:,3))
```

```
plot(input2step.time(),input2step.data(:,4))
plot(input2step.time(),input2step.data(:,5),"--g")
plot(input2step.time(),input2step.data(:,6),"--c")
plot(input2step.time(),input2step.data(:,7),"--m")
plot(input2step.time(),input2step.data(:,8),"--r")
plot(input2step.time(),input2step.data(:,9),"--b")
hold off
title('Heat Load 1 Step Input', 'Interpreter', 'latex')
xlabel('Time(s)')
ylabel('Temperature at Output (C)')

subplot(2,3,3)
plot(input3step.time(),input3step.data(:,1))
hold on
plot(input3step.time(),input3step.data(:,2))
plot(input3step.time(),input3step.data(:,3))
plot(input3step.time(),input3step.data(:,4))
plot(input3step.time(),input3step.data(:,5),"--g")
plot(input3step.time(),input3step.data(:,6),"--c")
plot(input3step.time(),input3step.data(:,7),"--m")
plot(input3step.time(),input3step.data(:,8),"--r")
plot(input3step.time(),input3step.data(:,9),"--b")
hold off
title('Heat Load 2 Step Input', 'Interpreter', 'latex')
xlabel('Time(s)')
ylabel('Temperature at Output (C)')

subplot(2,3,4)
plot(input4step.time(),input4step.data(:,1))
hold on
plot(input4step.time(),input4step.data(:,2))
plot(input4step.time(),input4step.data(:,3))
plot(input4step.time(),input4step.data(:,4))
plot(input4step.time(),input4step.data(:,5),"--g")
plot(input4step.time(),input4step.data(:,6),"--c")
plot(input4step.time(),input4step.data(:,7),"--m")
plot(input4step.time(),input4step.data(:,8),"--r")
plot(input4step.time(),input4step.data(:,9),"--b")
hold off
title('Heat Load 3 Step Input', 'Interpreter', 'latex')
xlabel('Time(s)')
ylabel('Temperature at Output (C)')

subplot(2,3,5)
plot(input5step.time(),input5step.data(:,1))
hold on
plot(input5step.time(),input5step.data(:,2))
plot(input5step.time(),input5step.data(:,3))
plot(input5step.time(),input5step.data(:,4))
plot(input5step.time(),input5step.data(:,5),"--g")
plot(input5step.time(),input5step.data(:,6),"--c")
plot(input5step.time(),input5step.data(:,7),"--m")
plot(input5step.time(),input5step.data(:,8),"--r")
plot(input5step.time(),input5step.data(:,9),"--b")
hold off
title('Heat Load 4 Step Input', 'Interpreter', 'latex')
xlabel('Time(s)')
ylabel('Temperature at Output (C)')
legend('C1: CP1 Fluid','C2: CP1 Wall','C3: CP2 Fluid','C4: CP2 Wall','C5: CP3
Fluid','C6: CP3 Wall','C7: CP4 Fluid','C8: CP4 Wall','C9: J1 Fluid');
```