DEVELOPMENT OF AN ENERGY STORAGE TANK MODEL

By

Robert Christopher Buckley

Approved:

DEVELOPMENT OF AN ENERGY STORAGE TANK MODEL

By

Robert Christopher Buckley

A Thesis
Submitted to the Faculty of the
University of Tennessee at Chattanooga
in Partial Fulfillment of the Requirement
for the Degree of Master of Science
in Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

October 2012

ABSTRACT

A linearized, one-dimensional finite difference model employing an implicit finite difference method for energy storage tanks is developed, programmed with MATLAB, and demonstrated for different applications.

A set of nodal energy equations is developed by considering the energy interactions on a small control volume. The general method of solving these equations is described as are other features of the simulation program. Two modeling applications are presented: the first using a hot water storage tank with a solar collector and an absorption chiller to cool a building in the summer, the second using a molten salt storage system with a solar collector and steam power plant to generate electricity. Recommendations for further study as well as all of the source code generated in the project are also provided.

DEDICATION

I would like to dedicate this work to my parents, Lucy Knowles and Martin Buckley, for their love, continuous support, and inexhaustible faith in me, to my brother for the same, and to Sarah for everything.

ACKNOWLEDGEMENTS

The author would like to formally express his sincere gratitude to everyone whose assistance was invaluable to this thesis. I cannot thank Dr. Prakash Dhamshala enough for serving as my academic adviser and thesis committee chairman, for his guidance, and for his dedication to this project. I am also deeply indebted to the other members of my thesis committee, Dr. James W. Hiestand and Dr. Philip M. Kazemersky, for their recommendations and assistance with this thesis. Finally, the author would like to thank Debbie Odom, again, for her patience and for logistical support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CHP – Combined Heat and Power, another term for Cogeneration Systems

COP – Coefficient of Performance

EPRI – Electric Power Research Institute

PCM – Phase Change Material

PCT – Parallel Computing Toolbox

SSPT – Simple Steam Power Toolbox

TABLER – Transient Analysis of Building Loads and Energy Requirements

LIST OF SYMBOLS

A_w	Area of the wall of the control volume; the area perpendicular to the radial direction
A_x	Area of the cross section of the control volume; the area perpendicular to the axial Direction
c_p	Specific Heat
h	Specific Enthalpy
k	Thermal Conductivity
m	Mass
$\dot{m}$	Mass flow rate
n	Node number
P	Pressure
$\dot{Q}_{gen}$	Power generated inside of control volume by heater
s	Specific Entropy
t	Time
T	Temperature
T_{ENV}	Environmental Temperature
U	Thermal Resistance of the storage tank wall
Δx	Distance between centroids of adjacent control volumes; usually the height of the control volume
X	Steam Quality

ρ	Density
`mldivide`	Words written in Courier New type face refer either to MATLAB functions or are lines of MATLAB code.
m_n^t	Mass of node n at time t, For most equations subscripts show node number, superscripts show time step.

CHAPTER I

INTRODUCTION

"Electrical storage is a very difficult proposition, and probably more expensive than generating the electricity." – Spencer Abraham {Abraham, 2010, p 157}

Energy storage is one of the most vexing problems associated with the way that energy is produced and utilized. While tremendous strides are being taken to be more conscientious and deliberate about how energy, at all scales, is generated and consumed, suitable means of storing and retrieving energy will also be necessary in the future.

On a large, utility-scale, there are two rather glaring and obvious applications for energy storage: taking some of the variability and unpredictability out of renewable energy resources like solar and wind, and load leveling or load shifting when the base electric load is provided by relatively non-variable sources like coal fired plants or nuclear reactors.

Renewable energy portfolios will require some sort of energy storage platform if they are to be implemented on a base load-scale. Energy sources like wind and solar are too mercurial to be depended on without either storage, alternate means of generation like hydroelectric or natural gas fired plants when there is no wind or sunshine, or both. Recently constructed concentrated solar plants in Spain, as well as those currently being constructed in the western United States, are being designed with large integrated energy storage systems, in the form of thermal energy storage tanks. {Flueckiger 2012} These plants, using either parabolic trough collectors or a large array of heliostats, concentrate large amounts of solar energy on a point where a heat transfer fluid, typically a heat transfer oil or a molten salt blend, is pumped and then the heat transfer fluid is pumped on to a tank. This heated fluid is then diverted from the tank to a heat exchanger to generate steam for a Rankine cycle power plant or some other useful thermal load as needed.

One problem that electric power providers find themselves with is that they have to be able to generate enough electricity to meet demand at any time without the guarantee that it will be used. To meet peak loading conditions, often they have to either bring more expensive modes of generation online or buy power from a neighboring utility, often at a premium. Meanwhile, the same utilities often have base load power during the non-peak hours that goes unused.

In many areas, the daily commercial electric demand follows a rough sinusoid pattern, with the peak occurring around midday and the trough occurring late at night. One common method of providing for this demand profile is to have a base load that provides a constant amount of power that will more than adequately provide for most of the day's demand and then to have some secondary generation come online to provide an extra amount during the hours of peak loading. This secondary, short-term source is often something comparatively expensive; this presents a common situation where during low demand inexpensive excess base load electricity is dumped because it cannot be used and then mere hours later short term generation, which are often the most expensive generation units for the utility to operate, are brought online to meet the peak demand. Richard Baxter notes that"[l]arge scale energy storage facilities can help increase the utilization of coal facilities or other units with excess capacity by acting as an energy sink during energy-usage off-peak periods." {Baxter, 2006, p 7} If the off-peak surplus power could be stored and then recovered during the peak loading hours, not only could utilities lower their overall emissions but they could eventually even see a cost savings as energy storage solutions become more cost-effective.

While energy storage for large scale power operations is convenient, pragmatic, and often of economic importance, for small scale electric power operations it is often a matter of necessity. In remote communities, where electrical generation options are limited, where fuel is nearly prohibitively expensive, or where renewable resources like wind power or solar power provide a large portion of the area's electricity, some form of energy storage is vital to providing power as needed.

Energy storage is also an important consideration in many building designs, {Parameshwaran et al., 2012} especially in the rising field of zero net-energy buildings and buildings intentionally built unconnected from the electrical grid. The use of a solar collector and an absorption chiller to provide the heating and cooling needs of a building requires thermal energy storage to continue addressing these needs after the sun has set. Small combined heat and power (CHP) systems and cogeneration facilities can often see significantly higher effective thermal efficiencies when used in conjunction with an energy storage system. As building energy management systems are designed and implemented with a more deliberate approach to energy consumption, energy storage systems will become increasingly important parts of many of these larger building energy management systems.

The suitability for a particular energy storage system with a given application is primarily dictated by the form of the energy to be stored and the form in which it is to be recovered. Battery storage has traditionally been one of the most widely used energy storage methods due in great part to electricity being considered more worth storing in many applications than other forms of energy, like kinetic or thermal. Thermal energy storage has become an increasingly viable energy storage method, for many of the reasons described above. The aim of this project is to develop a generic computational model of a thermal energy storage tank.

Many of the current high temperature thermal energy storage designs, especially those to be used in conjunction with concentrated solar power facilities, call for an aggregate of large filler material like quartz; these filler bed materials being chosen more for their price than their thermal storage properties. There are several papers {Flueckiger et al., 2011 & 2012} {Xu et al., 2012} {Yang et al., 2010a & 2010b} that discuss the modeling of thermal storage tanks containing packed beds. While a packed bed style tank has the ability to lower the initial cost of filling the tank, because packed bed materials are less expensive than their heat transfer fluid, in most cases these filler materials also reduce the amount of energy that can be effectively stored and recovered from inside the tank. Indeed, the use of a packed bed within a storage tank is only seriously considered when the primary thermal energy storage medium is considered to be expensive. {Flueckiger et al., 2012}

Adding a packed bed to this simulation would have added undesirable levels of complexity in order to accommodate only a few of the potential applications of the energy storage model. Quantities like bed medium size, geometry, porosity, volume, and operating temperature limits for the bed medium would all need to be accounted for. Incorporating packed beds into the thermal energy storage tank is an interesting problem, one to which several authors have given thoughtful consideration recently, but it is beyond the scope of this simulation.

The use of phase change materials or latent heat energy storage is another very exciting area of research. The amount of energy required to melt a solid or freeze a liquid is often quite substantial, and the energy density of such storage systems is potentially very high. From a modeling perspective, however, faithfully representing a material as it is either changing phases or merely existing in a mixture of phases concurrently is a particularly daunting task. There are also several pragmatic concerns about storing a material at or near its freezing temperature;

material could solidify within piping or at orifices with the effect of limiting material flow rates or heat transfer rates. Material could also solidify and then settle in a part of the tank and become unavailable to the charging or discharging systems. Indeed, as more material solidifies, it can become increasingly difficult to return the contents of the tank to a liquid form. A phase change material is worthy of its own investigation and energy storage systems that incorporate PCMs are beyond the scope of this project.

In short, this project models a sensible heat energy storage tank with a single storage medium. The tank is modeled with a load, something that draws off hot fluid and returns in to the tank at a lower temperature, like a heat exchanger for use with a heating system or even a Rankine cycle. The tank also has a model for a process that draws off cool fluid and returns it at a higher temperature, like a solar collector, an external process heater, or some other heat recovery device. While the model is provided with a single load and a single collector, additional loads or collectors can be added by merely copying and pasting a few lines of code and renaming a couple of variables. There is currently no restriction on which part of the tank the load draws from or where in the tank the fluid that was sent to the collector is reinserted, although such restrictions can be easily added to the code so as to better represent the system being modeled. Finally, the tank model also accommodates a built in heater which can be used both as a means of storing off-peak electricity as well as an emergency provision to prevent the heat transfer fluid inside the storage tank from solidifying. A schematic of the tank system with a solar collector and a generic load are shown in Figure 1.1.

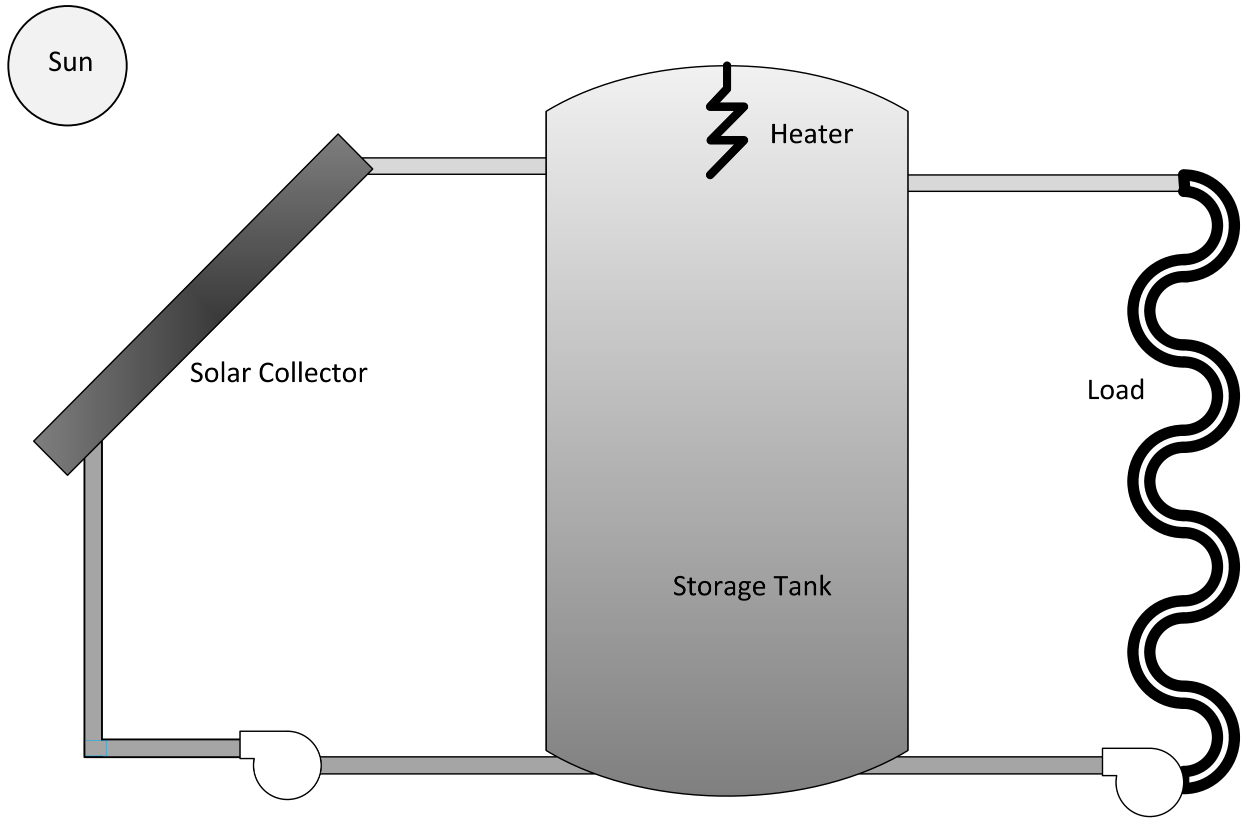


Figure 1.1 Schematic of an Energy Storage System Described by the Model

Prior Methods

There have been a variety of computational models for simulating energy storage tanks already published. Many of these models, however, sacrifice at least one energy interaction that is of potential interest to the modeler for the sake of computational simplicity.

One simulation method of note is the stratified storage tank model presented by Duffie and Beckman in their Solar Engineering of Thermal Processes text. {Duffie, 2006} Their model has a simplicity and an elegance; it models a hot water storage tank with only a handful of nodes and it accounts for heat losses through the wall of the tank. One fault of their model, however, is that it does not account for heat transfer within the fluid column; it does not show that a stagnant, stratified tank will eventually become de-stratified. Their model also assumes that the storage medium has a constant density and specific heat, which may be an appropriate assumption for certain storage media in certain applications, but for applications where the energy storage medium can be stored at a wide range of temperatures this assumption may no longer adequately describe the system.

Another method is the one employed by Ghaddar and Al-Marafie {Ghaddar, 1989} which was published with experimental data to support it. Even simpler than the method presented by Duffie and Beckman, it was developed in a way that could be solved analytically. Their formulation assumed a single flow throughout the tank, not separate inlets and outlets for collectors or loads. Their model also assumed that the tank was perfectly insulated; by neglecting convective losses to the environment they were able to keep their differential equation homogeneous. While computationally simple, this method may not adequately describe many potential energy storage tank situations.

Since computational power is much easier to access at present than it was when these models were devised, the model proposed in this thesis attempts to deal more thoroughly with the energy interactions within the storage tank than these earlier models. Instead of using a simple differential equation like Ghaddar et al., this model is built around an energy equation evaluated on a control volume using the method described by Özisik {Özisik, 1994, p 30-37} which allows the model to accommodate a wider variety of energy sources and interactions like multiple inlet and outlet flows, heaters within the tank, and conductive heat transfer within the tank.

Models developed by Flueckiger, Yang, and Garimella {Flueckiger et al., 2011 & 2012} {Yang et al., 2010a & 2012b} apply a thorough two dimensional finite element model to the consideration of packed bed storage tanks utilizing molten salt as the heat transfer fluid. Their models consider fluid velocity profiles within the tank, effects of external variables like ambient air temperature on the fluid dynamics within the tank, and other important short-term effects on tank dynamics.

Another model worth noting is the TRNSYS model developed by B. Newton {Newton, 1995}. He observed that the tank model that was packaged with the TRNSYS suite was inadequate at the time and wrote a thorough replacement. The model was developed specifically for water storage tanks to be used in conjunction with solar collectors, with great care given to the treatment of the geometry of internal auxiliary heaters and their effects on tank flow dynamics. His program also addresses the potential of multiple tanks and non-cylindrical tanks.

CHAPTER II

PROBLEM FORMULATION AND DEVELOPMENT OF THE ENERGY EQUATIONS

Methodology

One who is interested in modeling a thermal storage tank could use a variety of methods. A finite element model could be used to examine fluid dynamics and interactions within the tank during certain specific conditions. This level of detail makes a finite element method very appealing for some analyses, like Yang and Garimella's examination of environmental boundary conditions {Yang et al., 2010b} but it also makes it less practical for analyses of long-term storage dynamics. For long term storage applications, it might be enough to observe the temperature distribution in the tank over time and hourly energy additions, subtractions, and losses. For these applications, a finite difference model would probably be most appropriate.

There are two primary methods for solving finite difference problems: an explicit method and an implicit method. {Bergman, 2011} Using an explicit method, the temperature at a node at a time t+1 is described as a function of the temperature of the node and the surrounding nodes at time t. The explicit method equations are generally simple to solve but the explicit method has a stability criterion. The stability criterion places an upper limit on the size of the time step; often the time step is small enough that relatively trivial heat transfer problems take a fairly long time to solve computationally. Using the implicit method, the temperature of the node at a time t+1 is a function of the temperatures of the adjacent nodes at time t+1. This presents a system of equations that, for a single time step, requires much more effort to solve than the equations for the explicit method; typically it present n equations with n unknowns. The implicit method is not constrained by a stability criterion, however, so the computational losses in the solution of a single time step when compared to the same problem solved explicitly can be overcome by solving the problem with larger, and therefore fewer, time steps. While there is no minimum time step solution size for the implicit method, care should be taken when selecting the time step; very large time steps will represent the original system less accurately.

For this simulation, an implicit solution method was chosen for several reasons. For energy storage applications, it may be important to simulate the storage system for a period of weeks or months, which could take a very long time to simulate using an explicit method. Also, since the main aspect of this simulation is the storage tank, a system that tends to be if not somewhat static then typically not very quickly changing, larger time step sizes make sense from a modeling perspective. A nodal diagram for a twenty node tank is shown in Figure 2.1.

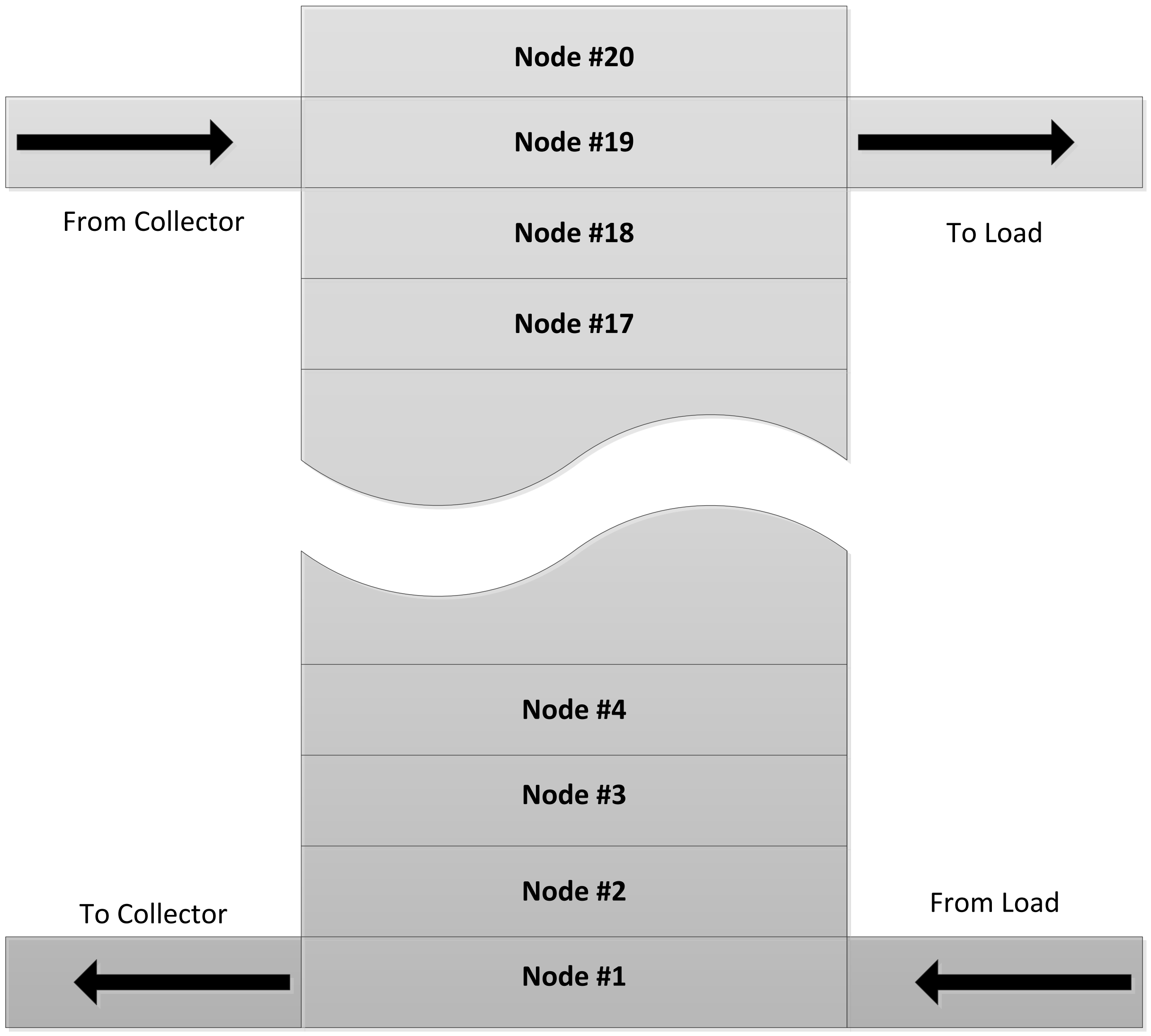


Figure 2.1 Tank Schematic with Nodal Arrangement for a Tank Modeled with Twenty Nodes

If the material properties of the working fluid in the tank, like density and specific heat, are modeled as functions of temperature, then the systems of equations that are generated are inherently non-linear. One method of linearizing these equations, as mentioned by Özisik {Özisik, 1994, p 250-252}, is to “lag” these values, or in essence to assume that the density at time t can be used to represent the density at time t+1. For systems that change slowly or with small time steps, these linearization methods are reasonable, and they are what this simulation employs.

The General Condition of the Energy Equation

Starting from a simple energy balance on an individual node, the change in the energy of the node per time is the difference between the sum of the rates of energy entering and the sum of the rates of energy leaving the node.

$$\frac{dE}{dt} = \sum \dot{E}_{in} - \sum \dot{E}_{out}$$

Sources of energy coming into the node are bulk transport into the node, conductive heat transfer from an adjacent node at a higher temperature, and if the heater is located within the node, energy from the heater. Energy leaves the tank by bulk transport, conductive heat transfer to an adjacent node at a lower temperature, and assuming that the temperature of the node is greater than the temperature of the ambient air outside the tank, heat transfer through the walls of the tank. A nodal diagram with energy balances is presented in Figure 2.2.

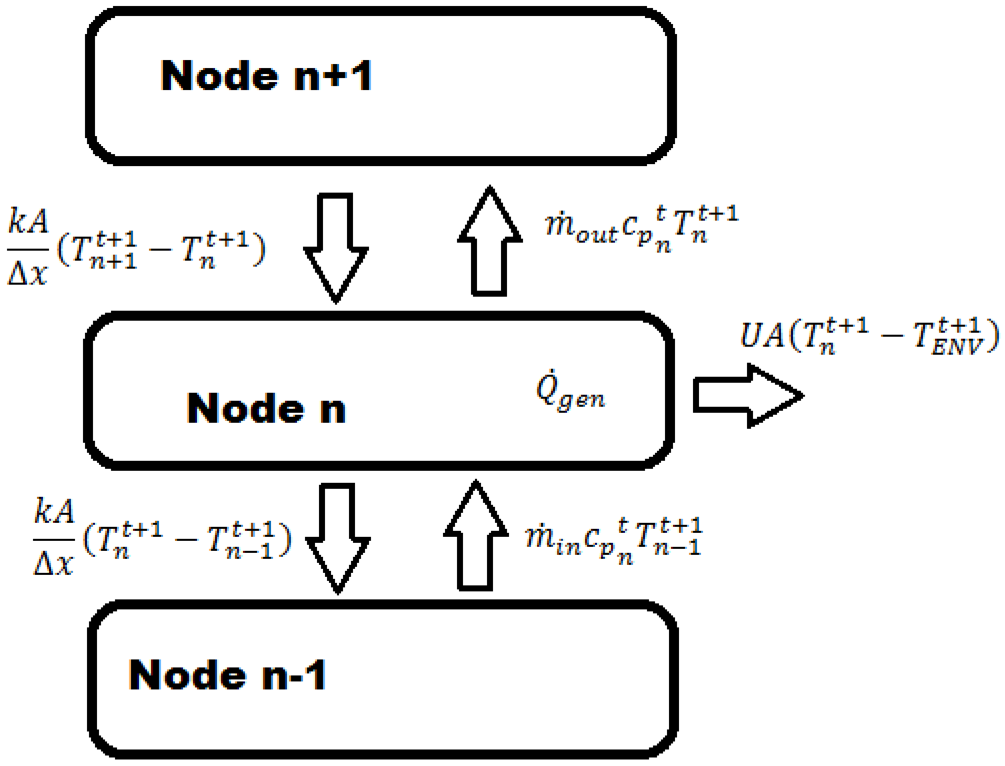

Node n+1
$\frac{kA}{\Delta x}(T_{n+1}^{t+1} - T_n^{t+1})$
$\dot{m}_{out} c_{p_n}^{\ t} T_n^{t+1}$
$UA(T_n^{t+1} - T_{ENV}^{t+1})$
Node n
$\dot{Q}_{gen}$
$\frac{kA}{\Delta x}(T_n^{t+1} - T_{n-1}^{t+1})$
$\dot{m}_{in} c_{p_n}^{\ t} T_{n-1}^{t+1}$
Node n-1

Figure 2.2 Energy Interactions on a Generic Node n Inside the Energy Storage Tank

First the general middle node will be considered, which is any node that has a full node on either side of it. This is case #2 in the set of node types shown in Figure 2.3. Combining the energy sources listed above with the general energy equation, we get the following energy equation for the general node condition:

$$\frac{dm_n\, c_{p_n} T_n}{dt} = \dot{m}_{in} c_{p_{in}} T_{in} + f(n_{HXC}) \dot{m}_{HXC} c_{p_{HXC}} T_{HXC} - \dot{m}_{out} c_{p_{out}} T_{out}$$
$$- f(n_{HXL}) \dot{m}_{HXL} c_{p_n} T_n - \frac{kA_x}{\Delta x}(T_n - T_{n-1}) + \frac{kA_x}{\Delta x}(T_{n+1} - T_n)$$
$$- UA_w(T_n - T_{ENV}) + \dot{Q}_{heater}$$

In the previous equation, $\dot{m}_{in}$ is the mass flow rate across the bottom of the node and $\dot{m}_{out}$ is the mass flow rate across the top of the node. Because the return flow from the load is reinserted at the bottom of the tank, the sign convention is for upward flow to be positive. Therefore, if $\dot{m}_{in}$ is positive, $T_{in} = T_{n-1}$ and if $\dot{m}_{in}$ is negative, $T_{in} = T_n$.

Similarly $for\ \dot{m}_{out} > 0,\ T_{out} = T_n\ \ and\ for\ \dot{m}_{out} < 0,\ T_{out} = T_{n+1}$

The subscripts HXC and HXL refer to flows returning from the collector and flows leaving for the load, respectively. We define the functions $f(n_{HXC})$ and $f(n_{HXL})$ as:

$f(n_{HXC}) = 1$ if node n is the node where the flow through the collector is reinserted and $f(n_{HXC}) = 0$ otherwise. Similarly, $f(n_{HXL}) = 1$ if node n is the node that the fluid sent to the load is drawn from and $f(n_{HXL}) = 0$ otherwise.

To keep the problem linear, it will be assumed for the sake of the energy equation and only the energy equation that the mass present in the node is constant, or rather that the density change in a single time step will be negligible as far as the energy balance equation is concerned. Expressing this as an implicit finite difference problem:

$$m_n^{t+1} {c_p}_n^{t+1} \frac{T_n^{t+1} - T_n^t}{\Delta t}$$

$$= \dot{m}_{in} {c_p}_{n-1}^{t+1} T_{n-1}^{t+1} + f(n_{HXC}) \dot{m}_{HXC} c_{p_{HXC}} T_{HXC} - \dot{m}_{out} {c_p}_n^{t+1} T_n^{t+1}$$

$$- f(n_{HXL}) \dot{m}_{HXL} c_{p_n} T_n \ - \frac{kA_x}{\Delta x}(T_n^{t+1} - T_{n-1}^{t+1}) + \frac{kA_x}{\Delta x}(T_{n+1}^{t+1} - T_n^{t+1})$$

$$- UA_w(T_n^{t+1} - T_{ENV}^{t+1}) + \dot{Q}_{heater}$$

Since the density and the specific heat of the material are being treated as functions of temperature, the above equation is still non-linear. It is also necessary to lag material properties that are functions of temperature by a time step. Doing so produces:

$$m_n^t {c_p}_n^t \frac{T_n^{t+1} - T_n^t}{\Delta t}$$

$$= \dot{m}_{in} {c_p}_{n-1}^t T_{n-1}^{t+1} + f(n_{HXC}) \dot{m}_{HXC} {c_p}_{HXC} T_{HXC} - \dot{m}_{out} {c_p}_n^t T_n^{t+1}$$

$$- f(n_{HXL}) \dot{m}_{HXL} c_{p_n} T_n \; - \frac{kA_x}{\Delta x} (T_n^{t+1} - T_{n-1}^{t+1}) + \frac{kA_x}{\Delta x} (T_{n+1}^{t+1} - T_n^{t+1})$$

$$- UA_w (T_n^{t+1} - T_{ENV}^{t+1}) + \dot{Q}_{heater}$$

The mass balance is considered as follows: after the system of equations is solved for the temperature distribution at time t+1, the mass at each node at time t+1 is calculated based on the density at the temperature at t+1 and the volume of the node. The mass flow rate leaving the node in the above equations is defined as

$$\dot{m}_{out} = \dot{m}_{in} + f(n_{HXC}) \dot{m}_{HXC} - f(n_{HXL}) \dot{m}_{HXL} + \frac{m_n^{t-1} - m_n^t}{\Delta t}$$

So, the changes in mass in the control volume due to density shifts are accounted for, but like the other properties that are functions of temperature, they lag the rest of the simulation by a time step.

It should also be noted that the flow rate of mass leaving the upper edge of node n is the same as the flow rate entering the bottom of node n+1, or:

$$\dot{m}_{out_n} = \dot{m}_{in_{n+1}}$$

For the case where the net mass flow is upward, the energy equation becomes:

$$T_{n-1}^{t+1}\left(-\frac{kA_x}{\Delta x}-\dot{m}_{in}{c_p}_{n-1}^{t+1}\right)+T_n^{t+1}\left(\frac{m_n^t {c_p}_n^t}{\Delta t}+\dot{m}_{out}{c_p}_n^t+f(n_{HXL})\dot{m}_{HXL}c_{p_n}+\frac{2kA_x}{\Delta x}+UA_w\right)$$

$$+T_{n+1}^{t+1}\left(-\frac{kA_x}{\Delta x}\right)$$

$$= m_n^t {c_p}_n^t\frac{T_n^t}{\Delta t}+f(n_{HXC})\dot{m}_{HXC}c_{p_{HXC}}T_{HXC}+UA_wT_{ENV}^{t+1}+\dot{Q}_{heater}$$

The values in the parentheses are the coefficients for this equation in the matrix [A] and the whole left side of the equation is the value of the element in the column matrix [C] corresponding to this node, or:

$$A(n,n-1)=\left(-\frac{kA_x}{\Delta x}-\dot{m}_{in}{c_p}_{n-1}^{t+1}\right)$$

$$A(n,n)=\left(\frac{m_n^t {c_p}_n^t}{\Delta t}+\dot{m}_{out}{c_p}_n^t+f(n_{HXL})\dot{m}_{HXL}c_{p_n}+\frac{2kA_x}{\Delta x}+UA_w\right)$$

$$A(n,n+1)=\left(-\frac{kA_x}{\Delta x}\right)$$

$$C(n,1)=m_n^t {c_p}_n^t\frac{T_n^t}{\Delta t}+f(n_{HXC})\dot{m}_{HXC}c_{p_{HXC}}T_{HXC}+UA_wT_{ENV}^{t+1}+\dot{Q}_{heater}$$

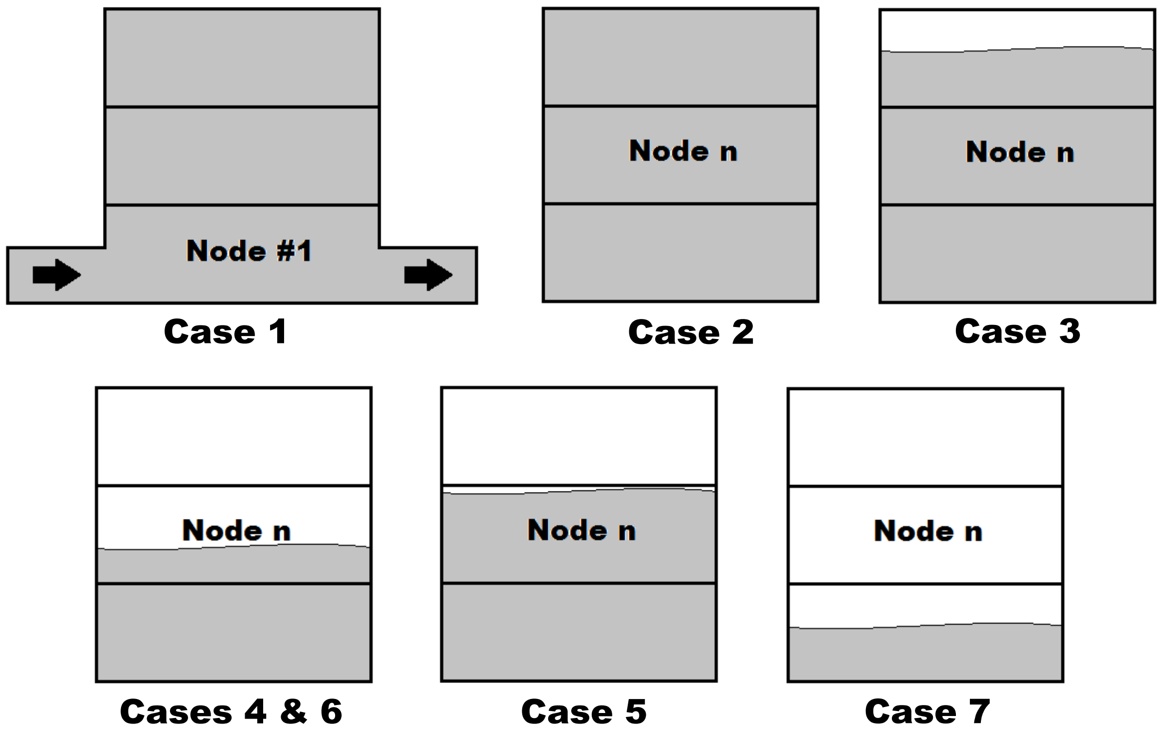


Figure 2.3 The Seven Different Node Types

Other Cases

The node at the bottom of the tank, node number one, is referred to as case #1 in Figure 2.3. The program currently inserts the return flow from the load into this node and extracts the flow to the collector from this node. While it might not be sound engineering practice to do so, the default model does not preclude one from reinserting the flow from the collector back into the first node. If we use our standard assumption that the net mass flow across the interface between nodes one and two is upward, then the energy balance for the bottom node takes the form:

$$T_n^{t+1}\left(\frac{m_n^t {c_p}_n^t}{\Delta t} + \dot{m}_{out} {c_p}_n^t + \dot{m}_{HXC}^{t+1} {c_p}_n^t + \frac{kA_x}{\Delta x} + UA_w\right) + T_{n+1}^{t+1}\left(-\frac{kA_x}{\Delta x}\right)$$

$$= m_n^t {c_p}_n^t \frac{T_n^t}{\Delta t} + \dot{m}_{in} c_{p_{in}} T_{in} + f(n_{HXC}) \dot{m}_{HXC}^t {c_p}_{HXC}^t T_{HXC}^t + UA_w T_{ENV}^{t+1} + \dot{Q}_{heater}$$

Where $\dot{m}_{in}$ and T_{in} refer to the flow returning from the load.

The penultimate node containing fluid, or case #3 from Figure 2.3, has almost exactly the same energy equation as case #2. Since the node above it is not entirely full, the distance between centroids for the upper conduction equation is no longer the same as the thickness of a single node, Δx. For this we define a new thickness, Δx_2:

$$\Delta x_2 = \frac{\Delta x + \Delta x_{n+1}}{2}$$

So, for the penultimate node containing fluid, the energy equation becomes:

$$T_{n-1}^{t+1}\left(-\frac{kA_x}{\Delta x} - \dot{m}_{in} {c_p}_{n-1}^{t+1}\right)$$

$$+ T_n^{t+1}\left(\frac{m_n^t {c_p}_n^t}{\Delta t} + \dot{m}_{out} {c_p}_n^t + f(n_{HXL}) \dot{m}_{HXL} c_{p_n} + \frac{kA_x}{\Delta x} + \frac{kA_x}{\Delta x_2} + UA_w\right)$$

$$+ T_{n+1}^{t+1}\left(-\frac{kA_x}{\Delta x_2}\right)$$

$$= m_n^t {c_p}_n^t \frac{T_n^t}{\Delta t} + f(n_{HXC}) \dot{m}_{HXC} c_{p_{HXC}} T_{HXC} + UA_w T_{ENV}^{t+1} + \dot{Q}_{heater}$$

There are two separate cases for the last node with fluid in it. The first of these two cases is the case when the node n+1 is empty at both time t and time t+1, or when there is not enough fluid entering the node during the time step to cause it to spill into the node above it. This case is case #4 in Figure 2.3. Similarly to case #3, we will need to define a distance between the centroid of node n and node n-1 as:

$$\Delta x_2 = \frac{\Delta x + \Delta x_n}{2}$$

Using this, and noting that the surface area of the wall for node n is $A_{w_n} = \pi D \Delta x_n$, the energy balance equation for this node becomes:

$$T_{n-1}^{t+1}\left(-\frac{kA_x}{\Delta x} - \dot{m}_{in} {c_p}_{n-1}^{t+1}\right) + T_n^{t+1}\left(\frac{m_n^t {c_p}_n^t}{\Delta t} + f(n_{HXL})\dot{m}_{HXL} c_{p_n} + \frac{kA_x}{\Delta x_2} + UA_{w_n}\right)$$

$$= m_n^t {c_p}_n^t \frac{T_n^t}{\Delta t} + f(n_{HXC})\dot{m}_{HXC} {c_p}_{HXC} T_{HXC} + UA_{w_n} T_{ENV}^{t+1} + \dot{Q}_{heater}$$

For the other case of the last node containing fluid, the node is almost full of fluid at time t, and the amount of fluid being added to the node during the time interval is large enough that the node is filled and it passes mass to the node above it, n+1, which was empty at time t. This is referred to in Figure 2.3 as case #5. Since the node n is almost full at time t, and it becomes full during the time step, for the nodal conductive and convective equations it is assumed to be full during the time step. The temperature of fluid in the new node is defined as $T_{n+1}^{t+1} = T_n^{t+1}$. Since at the beginning of the time step there is no fluid in node n+1, and there is presumably a small amount at time t+1, conduction to the upper node is neglected. So the energy equation for this case ends up being very similar to the case of a full node, and becomes:

$$T_{n-1}^{t+1}\left(-\frac{kA_x}{\Delta x} - \dot{m}_{in} {c_p}_{n-1}^{t+1}\right) + T_n^{t+1}\left(\frac{m_n^t {c_p}_n^t}{\Delta t} + \dot{m}_{out} {c_p}_n^t + f(n_{HXL})\dot{m}_{HXL} c_{p_n} + \frac{kA_x}{\Delta x} + UA_w\right)$$

$$= m_n^t {c_p}_n^t \frac{T_n^t}{\Delta t} + f(n_{HXC})\dot{m}_{HXC} {c_p}_{HXC} T_{HXC} + UA_w T_{ENV}^{t+1} + \dot{Q}_{heater}$$

And for the matrix coefficients for node n+1 at time t+1:

$$A(n+1, n+1) = 1, \qquad A(n+1, n) = -1, \qquad C(n+1, 1) = 0$$

The sixth case, the case when there is some mass in the final node, is actually the same as the partly full node case, case #3.

The seventh case is the case when a node is empty at time t and will continue to be empty at time t+1. It is a trivial case but included here because it is part of the calculations. It very simply sets the coefficients in the matrices equal to zero.

$A(n, n-1) = 0,\ A(n, n) = 0,\ A(n, n+1) = 0,\ C(n, 1) = 0$

CHAPTER III

SOLUTION IMPLEMENTATION AND PROGRAM OVERVIEW

Implementation

The program was written as a series of MATLAB functions in the MATLAB m-code format. This will allow the program to be incorporated into larger MATLAB system simulations, allow users familiar with MATLAB programming to tailor the code to fit their needs, and it allows the program to be used with Simulink system models. The source code for the main simulation function is provided in Appendix A, source code for the initializing functions are provided in Appendix B, and source code for the ancillary functions like the one that describes the collector are provided in Appendix C.

With the finite difference method described in Chapter two, the system of equations takes the matrix form [A] [T] = [C], where [A] is an n by n matrix containing the coefficients of T at time t+1, [T] is an n by 1 column of the temperatures of each node at time t+1, [C] is an n by 1 column containing the right hand side of the system of equations, and n is the number of nodes in the simulation. Solving for [T], the simulation uses a MATLAB function called `mldivide` which is similar to a matrix inversion method except that it has been optimized to be run much more efficiently. {Mathworks, 2012b}

If one or more of the upper nodes of the tank is empty, `mldivide` would either return a trivial solution or no solution at all for that time step. Since having being able to vary the mass and the volume of the working fluid inside the tank is highly desirable, a simple script was implemented to handle this difficulty. Just before `mldivide` is called, the script counts the number of zero values in the diagonal of the [A] matrix and then removes that many rows and columns from the end of [A] and that many rows from the end of [C]. After `mldivide` is called, the [T] column has zeroes concatenated onto its end until it has as many elements as there are nodes in the simulation. Note that values of zero in the temperature column are used to describe empty nodes; if this program is to be employed in an application where the heat transfer material is used at or around zero degrees Celsius, the program should be converted to work in Kelvin. This is merely an issue of redefining material properties and a few simulation parameters in the simulation loader file.

General Assumptions of the Simulation

For the sake of the energy equation, the amount of mass present in each node is assumed to not change, unless it is one of the upper two nodes, during an individual time step. The change of mass in a single node over a single time step is accounted for, and a further explanation of how the mass balances are treated will be given, but for the energy balance, this small change in mass makes a negligible contribution to the total energy of the node while making the energy balance equation non-linear.

The simulation also assumes that the temperature distribution can be treated as one dimensional; while there radial losses through the side of the tank to the environment, all points at the same height, and within the same node for that matter, are assumed to be at the same temperature. For well-insulated tanks with large inlets and outlets, this assumption is reasonable, and in reviewing the prior research, only a handful of models did not make this assumption, and they all used finite element methods to study fluid dynamics within the tank over short periods of time. {Yang, 2010b}

Another assumption that this simulation is built upon is the assumption that the top layer of the storage medium has no heat transfer interactions with the air pocket at the top of the tank; the top empty pocket of the tank is treated as a vacuum. If the tank is modeled as being well-insulated, this assumption should not have a significant impact on the simulation.

Flow mixing is also simplified within the model. It is assumed that flow returning from either the collector or load immediately displaces the same amount of mass from the node it is entering, and then it becomes perfectly mixed with the fluid remaining inside that node. While this will not affect the overall energy balance of the entire system, if time step sizes are large or if node sizes that are very small are selected, the temperature distribution in the nodes near the tank inlets in the model might deviate from how it would appear in the actual system.

There is a great amount of flexibility with the number of nodes the user assigns to the simulation. The number of nodes should be selected on how thick each node would be; it is up to the user to judiciously select an appropriate node thickness. The node should be thin enough that the nodal temperature, which is an average temperature for the entire node, can be considered representative of all of the material within the node.

The default model of the load and the collector assume that the flows returning to the tank do so at constant temperatures. The default collector model also assumes that a constant amount of energy is added to the system whenever the collector is being utilized. Using these two criteria, the program calculates the mass flow rates through the load and through the collector.

While theoretically there is no upward limit to the number of nodes in the tank, the MATLAB function `xlswrite` that exports the data to Microsoft Excel can only write 256 columns at one time. {Mathworks, 2012a} Currently the same instance of the `xlswrite` function also exports the other recorded values like the energy lost through the wall of the tank at each hour, so assigning more than 236 nodes to the simulation will likely cause the output function to fail. Assigning that many nodes for a relatively simple finite difference problem is not suggestible regardless and if one needed that level of refinement a finite element model in at least two dimensions would be required anyways.

Program Overview

The energy storage system model was built as a collection of MATLAB functions. The first function to be called by the user, the loader function, stores and initializes all the variables that the simulations needs. To adjust tank design parameters like the size, thermal resistance, size of the heater or the load, and the material properties of the heat transfer fluid, one would alter the values inside the loader function. The loader function also opens the Microsoft Excel spreadsheet file that contains the information pertinent to the specific simulation run, including hourly data on whether the collector, load, or internal heater are being used, weather data, and the initial temperature distribution inside of the tank. The loader function contains code built to find this information inside the Excel spreadsheet file and then import that data into the base MATLAB workspace. A call to the loader function might appear in the MATLAB command window as

```
LOADER('Condition1.xlsx');
```

The simulation continues when the user calls the tank function. The tank function contains the bulk of the simulation; it takes the data imported by the loader function and evaluates the energy balance equations at each period of time in every part of the tank. The workings and process of the tank function will be described in further detail in the following sections. After the simulation is complete, the tank function exports the relevant data like the hourly energy losses through the tank walls, the energy deposited in the tank, work done by the load, and the temperature distribution inside the tank at every hour. It saves these data in a new sheet in the same Excel spreadsheet that the loader function opened and the title of this new sheet is the date and time that this particular instance of the tank function was run.

If, for example, the operating conditions were stored in an Excel file titled Simulation2.xlsx, an entire simulation might appear in the MATLAB command window as:

```
>> LOADER('Simulation2.xlsx')
Variable Initializing Complete.
>> TANK
Tank Simulation Complete.
```

There may also appear an error in the MATLAB command window that reads “Warning: Added Specific Worksheet” followed by four other lines. This is normal; `xlswrite`, the MATLAB function that exports data to a Microsoft Excel spreadsheet, looks for an existing sheet with the name of the sheet in its call. The tank function calls `xlswrite` twice: once at the beginning of the function to create a new sheet and to format that sheet and then again at the end of the function to write all the simulation data to the sheet. The first time the tank function calls `xlswrite`, it asks it to write the data to a sheet whose name is the current time. If no sheet exists with that name, `xlswrite` will create a new one but it also displays this warning to let the user know that a new sheet was created. {Mathworks, 2012a}

There are also a few small ancillary functions that the tank function calls. One of these describes the functionality and control system for the collector and another does the same for the heater. This allows the user to make multiple copies of these small functions if, for instance, the user wanted to see how different collectors or different controllers affected tank dynamics. Additionally, there are a few inline functions that are defined in the loader file. These are simple functions that get called frequently within the main tank function and the other ancillary functions. These functions include those that define material properties like density and specific heat as functions of temperature. These are defined as global inline functions because several other functions call them but they are both simulation specific and simple enough to be contained within a single line and therefore don’t require their own separate function files.

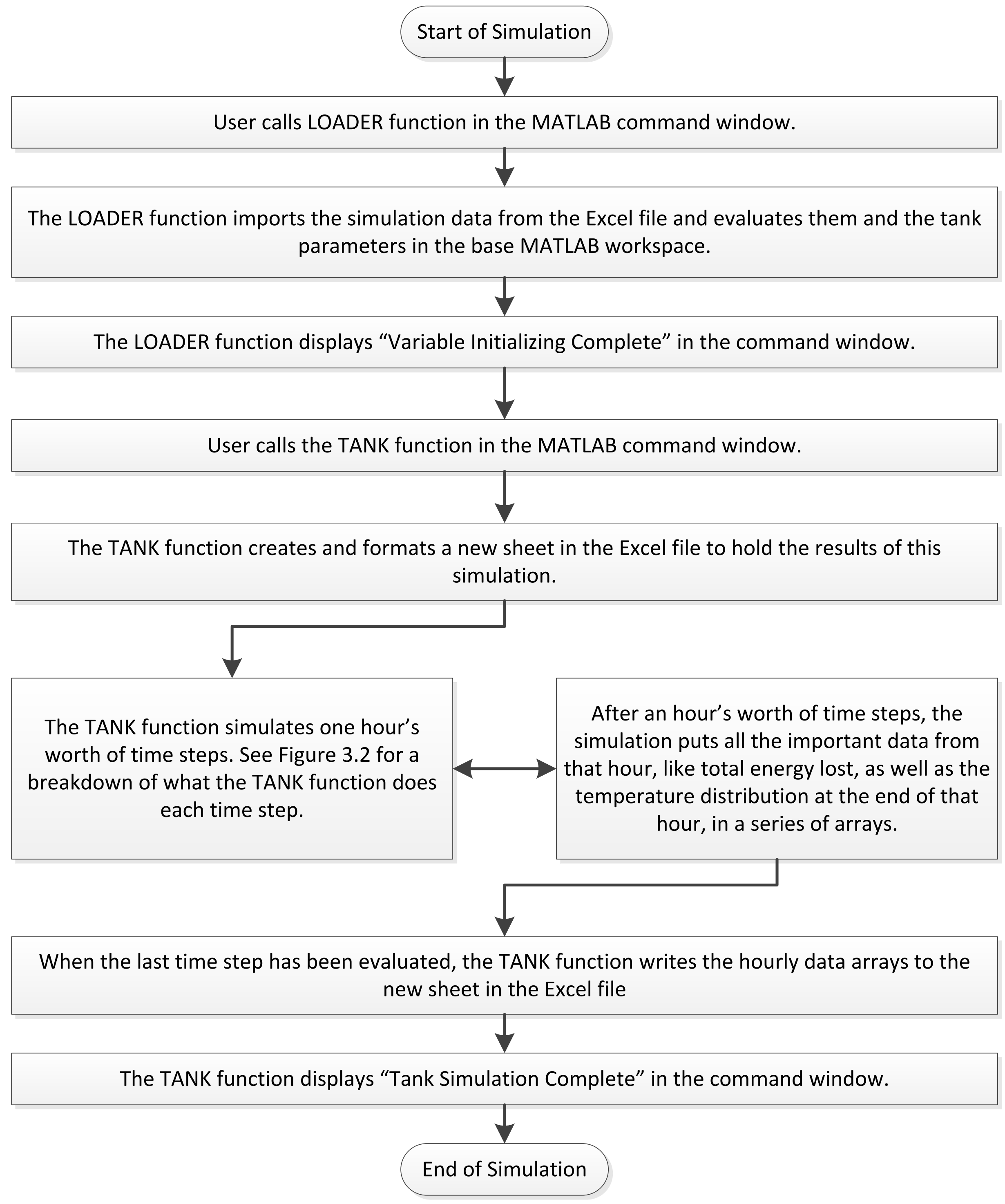

Start of Simulation
User calls LOADER function in the MATLAB command window.
The LOADER function imports the simulation data from the Excel file and evaluates them and the tank parameters in the base MATLAB workspace.
The LOADER function displays “Variable Initializing Complete” in the command window.
User calls the TANK function in the MATLAB command window.
The TANK function creates and formats a new sheet in the Excel file to hold the results of this simulation.
The TANK function simulates one hour’s worth of time steps. See Figure 3.2 for a breakdown of what the TANK function does each time step.
After an hour’s worth of time steps, the simulation puts all the important data from that hour, like total energy lost, as well as the temperature distribution at the end of that hour, in a series of arrays.
When the last time step has been evaluated, the TANK function writes the hourly data arrays to the new sheet in the Excel file
The TANK function displays “Tank Simulation Complete” in the command window.
End of Simulation

Figure 3.1 Simulation Overview Flowchart

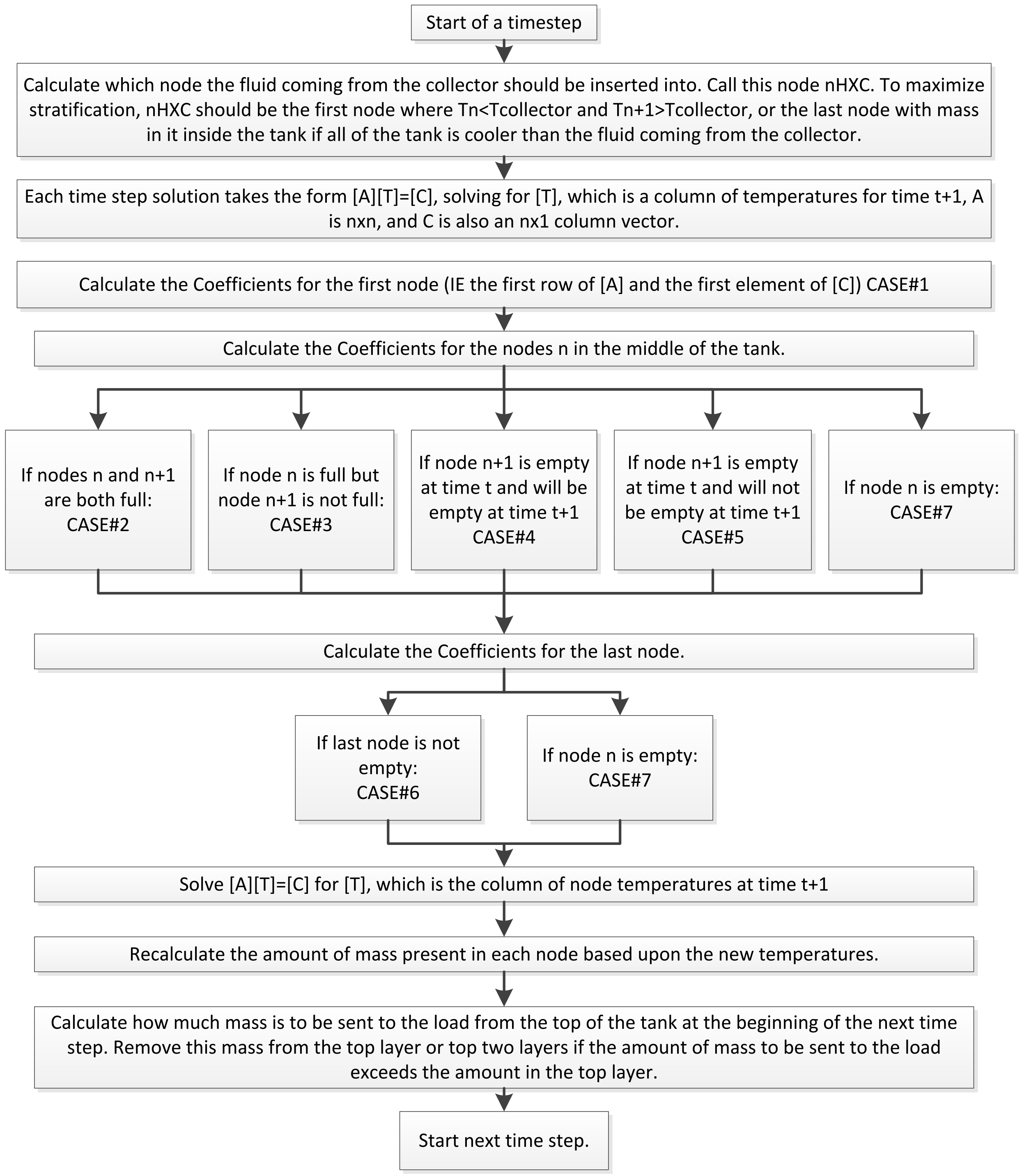

Start of a timestep
Calculate which node the fluid coming from the collector should be inserted into. Call this node nHXC. To maximize stratification, nHXC should be the first node where Tn<Tcollector and Tn+1>Tcollector, or the last node with mass in it inside the tank if all of the tank is cooler than the fluid coming from the collector.
Each time step solution takes the form [A][T]=[C], solving for [T], which is a column of temperatures for time t+1, A is nxn, and C is also an nx1 column vector.
Calculate the Coefficients for the first node (IE the first row of [A] and the first element of [C]) CASE#1
Calculate the Coefficients for the nodes n in the middle of the tank.
If nodes n and n+1 are both full: CASE#2
If node n is full but node n+1 is not full: CASE#3
If node n+1 is empty at time t and will be empty at time t+1 CASE#4
If node n+1 is empty at time t and will not be empty at time t+1 CASE#5
If node n is empty: CASE#7
Calculate the Coefficients for the last node.
If last node is not empty: CASE#6
If node n is empty: CASE#7
Solve [A][T]=[C] for [T], which is the column of node temperatures at time t+1
Recalculate the amount of mass present in each node based upon the new temperatures.
Calculate how much mass is to be sent to the load from the top of the tank at the beginning of the next time step. Remove this mass from the top layer or top two layers if the amount of mass to be sent to the load exceeds the amount in the top layer.
Start next time step.

Figure 3.2 Flowchart of Simulation Process During a Single Time Step

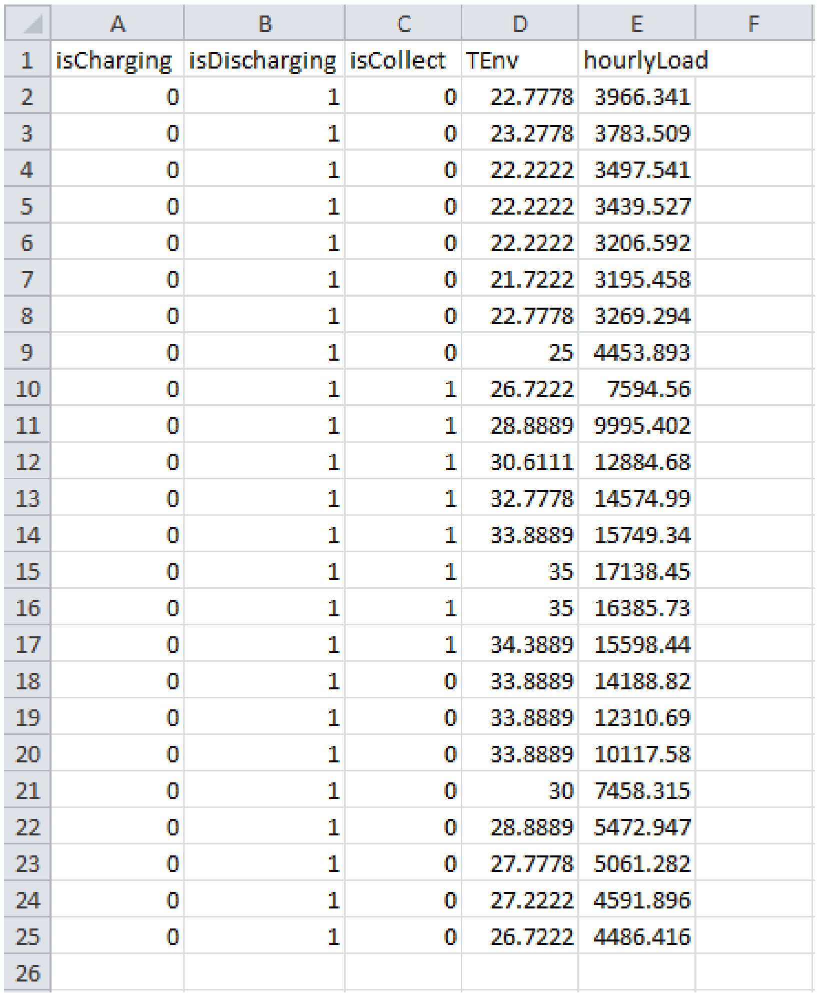
Program Features

To make the program easier to use, an input/output system was written taking advantage of some of MATLAB’s built-in functions. To make it easier to import data, manipulate the input and output data, and to allow those who either don’t have access to a MATLAB license or those who would prefer to use other means to analyze the data, the program uses a Microsoft Excel spreadsheet file as its means of input and output. The program can read and write to either the contemporary .xlsx, .xlsb, and .xlsm file extensions or the older .xls extension, but the current builds of MATLAB require Microsoft Excel 2007 or later installed on the computer to read or write to the newer formats.

The LOADER function scans the Excel file for certain key words, and all numerical values within the same column are assumed to be under that heading and are imported into the MATLAB workspace. These column titles that the loader function looks for are listed in Table 3.1. Note that the first three input titles, the Booleans, are required by the LOADER function, and all three columns need to be the same length. It is by measuring the length of these columns that the program knows how many hours it is simulating. A sample input sheet is shown in Figure 3.3.

Table 3.1 Column Titles for Simulation Input Parameters

Column Title	**Input Function**
isCharging	Tells the program whether the internal heater is scheduled to run that hour, Boolean
isDischarging	Tells the program whether the load is applied in that hour, Boolean
isCollect	Tells the program whether the collector is used in that hour, Boolean
TEnv	Ambient temperature outside the tank in Celsius; if absent from the Excel file, the loader function will use a default ambient temperature.
hourlyLoad	Power absorbed by the load that hour, in watts. isDischarging must be 1 that hour to be used. If absent from Excel file, the function will use a default load power whenever the load is applied.
initialTankTemps	Initial temperature in each node, starting with node number one (bottom of the tank) in degrees Celsius. If absent from the Excel file, the loader function will assign a default initial temperature distribution to the simulation.



	A	B	C	D	E	F
1	isCharging	isDischarging	isCollect	TEnv	hourlyLoad	
2	0	1	0	22.7778	3966.341	
3	0	1	0	23.2778	3783.509	
4	0	1	0	22.2222	3497.541	
5	0	1	0	22.2222	3439.527	
6	0	1	0	22.2222	3206.592	
7	0	1	0	21.7222	3195.458	
8	0	1	0	22.7778	3269.294	
9	0	1	0	25	4453.893	
10	0	1	1	26.7222	7594.56	
11	0	1	1	28.8889	9995.402	
12	0	1	1	30.6111	12884.68	
13	0	1	1	32.7778	14574.99	
14	0	1	1	33.8889	15749.34	
15	0	1	1	35	17138.45	
16	0	1	1	35	16385.73	
17	0	1	1	34.3889	15598.44	
18	0	1	0	33.8889	14188.82	
19	0	1	0	33.8889	12310.69	
20	0	1	0	33.8889	10117.58	
21	0	1	0	30	7458.315	
22	0	1	0	28.8889	5472.947	
23	0	1	0	27.7778	5061.282	
24	0	1	0	27.2222	4591.896	
25	0	1	0	26.7222	4486.416	
26						

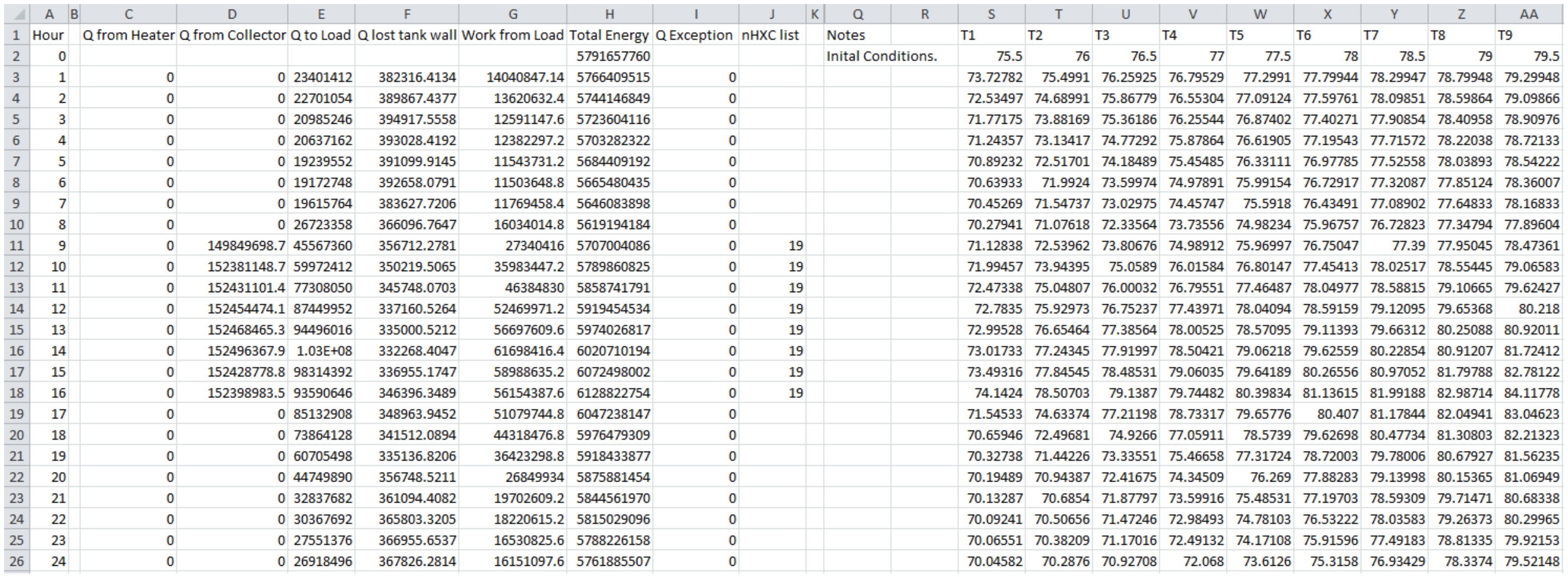
Figure 3.3 Sample Input Sheet in the Microsoft Excel Spreadsheet Used in the Simulation

When the tank function is called, the first thing it does after taking the global variables is it marks the time at that moment, and then keeps it as a formatted string. It then creates a new sheet with that string as its name and then prints the column titles shown in Table 3.2 to the top of the sheet. At the very end of the tank simulation, it prints the rest of the output data to the sheet as well. A sample output sheet is shown partially in Figure 3.4 with nodal temperatures through node number nine.

Table 3.2 Column Titles for Simulation Outputs

Column Title	**Output Definition**
Hour	An hour number to help the user keep track of simulation events, with hour zero showing the initial conditions
Q from Heater	Energy added to the tank by operating the internal heater that hour, joules
Q from Collector	Energy added to the tank by operating the collector that hour, joules
Q to Load	Energy removed from the tank by applying the load that hour, joules
Q lost tank Wall	Energy lost through heat transfer to the surroundings that hour, joules
Work from Load	Total work done by the load that hour, joules. If the load is a power plant, that would be shaft work; if the load was a heating or cooling system, it would be energy transferred to the building being heated or cooled.
Total Energy	Total amount of energy present in the tank at the end of the hour, joules
Q Exception	Net energy change in the tank due to application of the last node exception, joules
nHXC list	List of the nodes that fluid returning from the collector was deposited in
Notes	Lists important events that occurred during the hour.
T1, T2, etc.	Temperatures of each node at the end of the hour, degrees Celsius.

Printing all of the output data at once has both benefits and negative effects as well. If there is an error, and for some reason the program has to quit prematurely, no data is recorded. This can be frustrating when trying to seek out the cause of the problem; often the error in the MATLAB command window will only display the line number of the code that caused the simulation to fail. On the other hand, having fewer calls to the `xlswrite` function speeds up the program significantly. The `xlswrite` function also requires that the spreadsheet be unused by any other programs at the moment it is called, yet `xlswrite` can tie up the spreadsheet for a relatively long time while it writes data to it; if the program tried to write the output to the spreadsheet every hour in the simulation, a single call to `xlswrite` could cause the entire simulation to crash because the last call to `xlswrite` was still using the spreadsheet. There are a few other ways to remedy this situation, but having the function write all the output data at once was the simplest and easiest to execute.



	A	B	C	D	E	F	G	H	I	J	K	Q	R	S	T	U	V	W	X	Y	Z	AA
1	Hour		Q from Heater	Q from Collector	Q to Load	Q lost tank wall	Work from Load	Total Energy	Q Exception	nHXC list		Notes		T1	T2	T3	T4	T5	T6	T7	T8	T9
2	0							5791657760				Inital Conditions.		75.5	76	76.5	77	77.5	78	78.5	79	79.5
3	1		0	0	23401412	382316.4134	14040847.14	5766409515	0					73.72782	75.4991	76.25925	76.79529	77.2991	77.79944	78.29947	78.79948	79.29948
4	2		0	0	22701054	389867.4377	13620632.4	5744146849	0					72.53497	74.68991	75.86779	76.55304	77.09124	77.59761	78.09851	78.59864	79.09866
5	3		0	0	20985246	394917.5558	12591147.6	5723604116	0					71.77175	73.88169	75.36186	76.25544	76.87402	77.40271	77.90854	78.40958	78.90976
6	4		0	0	20637162	393028.4192	12382297.2	5703282322	0					71.24357	73.13417	74.77292	75.87864	76.61905	77.19543	77.71572	78.22038	78.72133
7	5		0	0	19239552	391099.9145	11543731.2	5684409192	0					70.89232	72.51701	74.18489	75.45485	76.33111	76.97785	77.52558	78.03893	78.54222
8	6		0	0	19172748	392658.0791	11503648.8	5665480435	0					70.63933	71.9924	73.59974	74.97891	75.99154	76.72917	77.32087	77.85124	78.36007
9	7		0	0	19615764	383627.7206	11769458.4	5646083898	0					70.45269	71.54737	73.02975	74.45747	75.5918	76.43491	77.08902	77.64833	78.16833
10	8		0	0	26723358	366096.7647	16034014.8	5619194184	0					70.27941	71.07618	72.33564	73.73556	74.98234	75.96757	76.72823	77.34794	77.89604
11	9		0	149849698.7	45567360	356712.2781	27340416	5707004086	0	19				71.12838	72.53962	73.80676	74.98912	75.96997	76.75047	77.39	77.95045	78.47361
12	10		0	152381148.7	59972412	350219.5065	35983447.2	5789860825	0	19				71.99457	73.94395	75.0589	76.01584	76.80147	77.45413	78.02517	78.55445	79.06583
13	11		0	152431101.4	77308050	345748.0703	46384830	5858741791	0	19				72.47338	75.04807	76.00032	76.79551	77.46487	78.04977	78.58815	79.10665	79.62427
14	12		0	152454474.1	87449952	337160.5264	52469971.2	5919454534	0	19				72.7835	75.92973	76.75237	77.43971	78.04094	78.59159	79.12095	79.65368	80.218
15	13		0	152468465.3	94496016	335000.5212	56697609.6	5974026817	0	19				72.99528	76.65464	77.38564	78.00525	78.57095	79.11393	79.66312	80.25088	80.92011
16	14		0	152496367.9	1.03E+08	332268.4047	61698416.4	6020710194	0	19				73.01733	77.24345	77.91997	78.50421	79.06218	79.62559	80.22854	80.91207	81.72412
17	15		0	152428778.8	98314392	336955.1747	58988635.2	6072498002	0	19				73.49316	77.84545	78.48531	79.06035	79.64189	80.26556	80.97052	81.79788	82.78122
18	16		0	152398983.5	93590646	346396.3489	56154387.6	6128822754	0	19				74.1424	78.50703	79.1387	79.74482	80.39834	81.13615	81.99188	82.98714	84.11778
19	17		0	0	85132908	348963.9452	51079744.8	6047238147	0					71.54533	74.63374	77.21198	78.73317	79.65776	80.407	81.17844	82.04941	83.04623
20	18		0	0	73864128	341512.0894	44318476.8	5976479309	0					70.65946	72.49681	74.9266	77.05911	78.5739	79.62698	80.47734	81.30803	82.21323
21	19		0	0	60705498	335136.8206	36423298.8	5918433877	0					70.32738	71.44226	73.33551	75.46658	77.31724	78.72003	79.78006	80.67927	81.56235
22	20		0	0	44749890	356748.5211	26849934	5875881454	0					70.19489	70.94387	72.41675	74.34509	76.269	77.88283	79.13998	80.15365	81.06949
23	21		0	0	32837682	361094.4082	19702609.2	5844561970	0					70.13287	70.6854	71.87797	73.59916	75.48531	77.19703	78.59309	79.71471	80.68338
24	22		0	0	30367692	365803.3205	18220615.2	5815029096	0					70.09241	70.50656	71.47246	72.98493	74.78103	76.53222	78.03583	79.26373	80.29965
25	23		0	0	27551376	366955.6537	16530825.6	5788226158	0					70.06551	70.38209	71.17016	72.49132	74.17108	75.91596	77.49183	78.81335	79.92153
26	24		0	0	26918496	367826.2814	16151097.6	5761885507	0					70.04582	70.2876	70.92708	72.068	73.6126	75.3158	76.93429	78.3374	79.52148

Figure 3.4 Sample Output Sheet from the Microsoft Excel Spreadsheet Used in the Simulation

Another important part of the output functionalities is the Notes column. In this column of the output sheet, all the important events of the hour are recorded so that the user can quickly identify sources of error in the results. Currently, the program has an output note for when the volume of the mass inside the tank exceeds the volume of the tank itself; the output notes that the tank effectively overflowed and that the program removed the excessive mass from the system. If the collector or the heater are scheduled to be used, but the temperatures inside the tank are too high to need it, there are notes to signify that the collector or heater wasn't used. If the temperature of a node drops below a critical temperature and the heater is employed even though it was not scheduled to be employed, it is noted. When the temperature at the top of the tank is too low for the load to be used and the load is scheduled to be used, it is turned off and the load shutdown is noted. There is a note for when the last node exception, which is explained in further detail in chapter 6, is employed.

CHAPTER IV

APPLICATION FOR BUILDING-SCALE ENERGY STORAGE

Arguably, the area for the largest and most rapid deployment of thermal energy storage systems is in providing for the heating, cooling, and electrical needs of buildings. As smart grid systems get more widely applied to the electrical power grid and varying electrical pricing structures based on time-of-use are introduced, building managers, even in buildings that don't employ some sort of solar collector or CHP system, will find energy storage systems to be increasingly economically advantageous.

To demonstrate the energy storage tank simulation for this application, a small building with four occupants was imagined. The building's cooling needs would be met by employing an absorption chiller fed by a hot water storage tank, and the storage tank would in turn be charged by a solar collector. For this particular simulation the Transient Analysis of Building Loads and Energy Requirements (TABLER) program {Dhamshala, 2012} was employed to develop an hourly cooling load requirement for a hypothetical building located in Chattanooga, TN on its day of peak cooling. The full input and output of this program is listed in Appendix D.

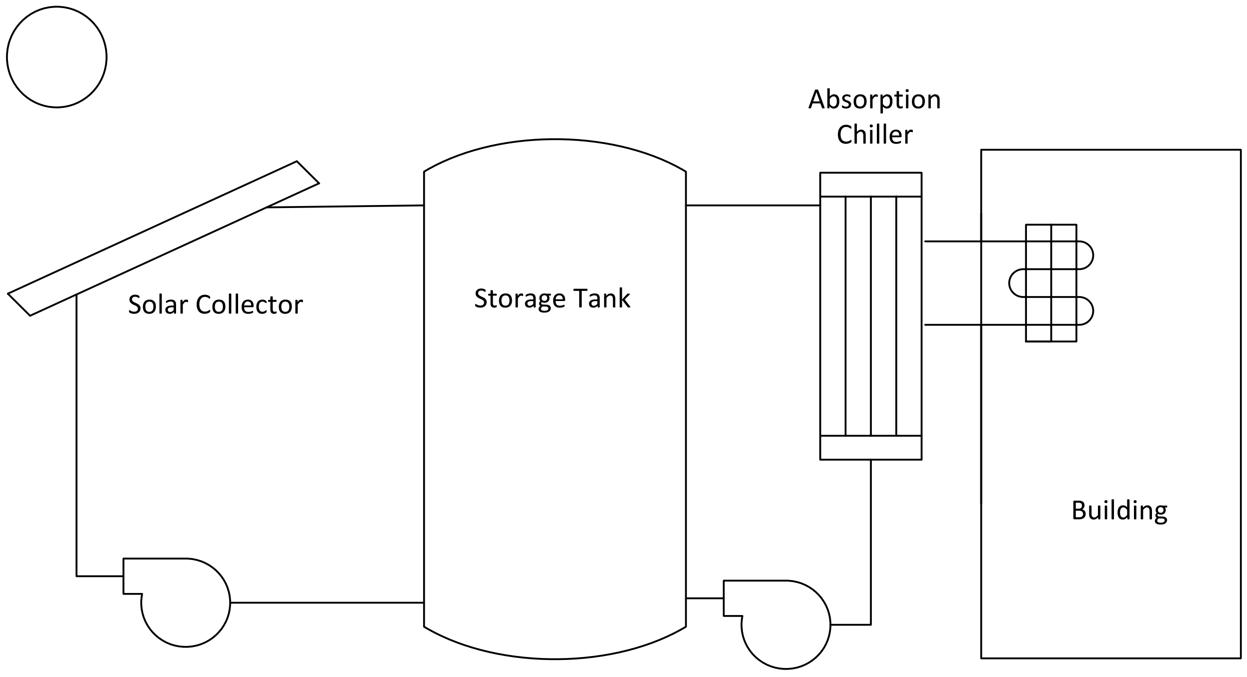


Figure 4.1 Schematic of Solar Collector, Storage Tank, and Absorption Chiller System Described in this Simulation

Simulation Parameters

The following equations, {Florides et al., 2003} were used to model the thermal properties of water in the loader file.

Specific Heat, in J/kg K : $c_p = (3.216145833 * 10^{-6})T^4 - (7.98668982 * 10^{-4})T^3 + (7.80295139 * 10^{-2})T^2$ -3.0481614T + 4217.7377

Density, in kg/m^3 : $\rho = (1.5451 * 10^{-5})T^3 - (5.9003 * 10^{-3})T^2 - (1.9075 * 10^{-2})T + 1002.3052$

The thermal conductivity was kept at a constant 0.558 W/m K.

For this test case, the absorption chiller was modeled as having a constant COP of 0.6 and a constant return temperature of 70 degrees Celsius for the fluid returning to the tank; this is a significant simplification but it should be adequate for demonstrating the energy storage tank simulation. A simplified solar collector model was used as well; the solar collector had a constant thermal output of 42.36 kW and was employed for eight hours, from 9:00 A.M. to 5:00 P.M. The collector size was dictated by the peak cooling load; 42.36 kW over eight hours ought to be able to provide for all twenty four hours of the peak cooling day.

Allowing for a drop of 15 degrees Celsius in temperature for the fluid leaving the tank and passing through the absorption chiller's heat exchanger, the minimum volume of the storage tank that would be able to provide enough energy to power the absorption chiller was calculated to be 12.9 cubic meters. The dimensions of the tank used in this simulation were 2.5 meters in diameter and 4 meters high, which provides about 50% more storage than the calculated minimum required. The thermal resistance of the tank wall was modeled as 0.436 W/m^{2}K, based on a two inch thick layer of polyurethane foam insulation, common for this kind of tank, and average convective coefficients.

Results

In the first simulation run using this model, the tank and absorption chiller system was modeled for twenty four hours, using the cooling load demands predicted by TABLER. A graph of the temperature distribution for this simulation is shown in Figure 4.2. By observing the initial and final condition curves, one can see that the area under the two, which is roughly proportional to the amount of energy present in the system, is equivalent. Indeed, by examining the output file for the simulation, the total energy contained within the tank initially was 5.792 GJ, at its highest point was in excess of 6.128 GJ, and was 5.762 GJ after twenty four hours. This close return to the initial amount of energy within the tank means, that for this preliminary simulation, the size of the solar collector was indeed well selected for the purpose.

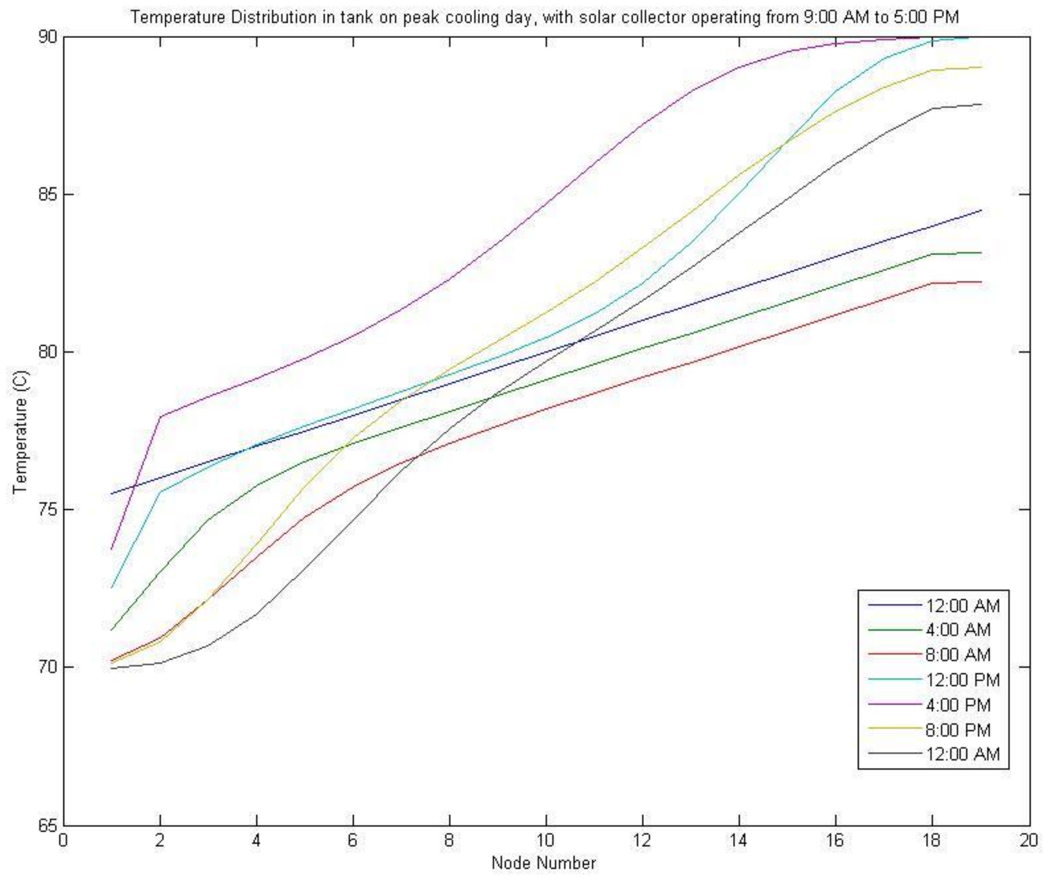


Figure 4.2 Temperature Distribution Inside Water Tank Every Four Hours for the Peak Cooling Load Day

In Figure 4.2, one can also observe stratification within the tank. Notice how the upper nodes at 4:00 A.M. and 8:00 A.M. retain a linear temperature distribution; the temperature distribution in the upper part of the tank is mostly unaffected by the cooler fluid being returned at the bottom of the tank. While the chart shows the temperature at the upper nodes dropping during this time period, this is mostly due to the fact that the load is being operated and the collector is not. When the load is operating and the collector is not, the net flow inside the tank is upward because fluid is being drawn off the top of the tank and reinserted at the bottom. The fluid higher in the tank is displaced upward by the fluid below it.

Another observation from Figure 4.2 is how the temperatures at the top and the bottom nodes change as operating conditions change. It is immediately apparent that the temperature at the top of the tank increases when the collector is in use and that it decreases when it is not in use. Perhaps less obvious is that the temperature of the bottom node does as well. Since the temperature change across the inlet and outlets of both the collector and the load are about the same, the mass flow rate through the collector is, on average, roughly three times the flow rate of the load since it is operated one-third as often. So, when the collector is operating, the net motion of the fluid inside the tank is downward, and when the collector is not operating, the net flow within the tank is upward.

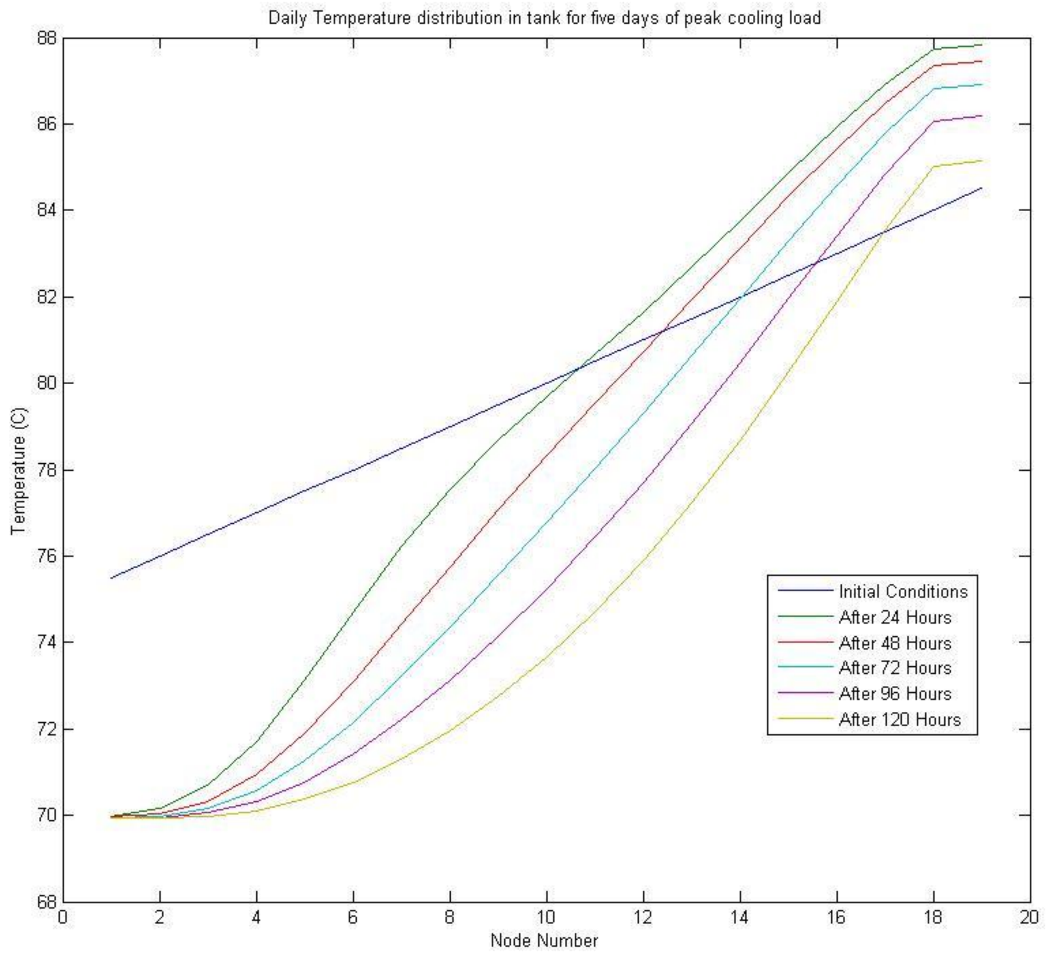


Figure 4.3 Daily Temperature Distribution in Water Tank over Five Days

Figure 4.3 shows a temperature distribution at the same time within the tank over a five day period with the maximum cooling load, as calculated by TABLER, applied to each day. Upon examination, an obvious trend emerges. The purely diagonal line is the initial temperature distribution, but the temperature distributions for the last four days follow similar curves. The temperature of the middles nodes drop a little as time progresses; this is because the solar collector used in the simulation was modeled as matching the peak cooling day load without accounting for energy losses through the walls of the insulated tank.

This simulation platform should allow someone to test different collector sizes and different weather conditions to help design an actual solar collector installation. This idealized collector was sufficient to provide enough energy to the system for its peak cooling day, but the peak cooling load does not necessarily need to be totally met by the system if the initial temperatures in the tank are higher. One can run simulations with lower cooling load days in advance of the peak cooling load day and examine the storage dynamics, potentially allowing for a much smaller solar collector to functionally meet most of the building's cooling loads.

One can also examine how overcast days affect the storage system; if the system needs to be able to handle overcast periods without relying on the internal heater or some other auxiliary method to provide the building's cooling load, a larger collector would likely be required. The model can be used to run a simulation of weeks or months with varying load and weather conditions to help the user find an effective solution for meeting the cooling and heating requirements of a building.

CHAPTER V

APPLICATION FOR ELECTRIC POWER-SCALE ENERGY STORAGE

The second application considered for this program was for a large molten salt energy storage tank for use with either a large solar collector or as in a potential load leveling application. The working fluid for this model is HITEC brand heat transfer salt, a mixture sold by Coastal Chemical that is by weight 7% $NaNO_3$, 40% $NaNO_2$, and 53% KNO_3. Molten salts are of interest for these applications because they remain in a liquid state at temperatures in the steam power regime, so molten salt can be diverted from a tank and through a heat exchanger where it generates high pressure steam for a Rankine cycle plant. HITEC salt specifically remains in a stable liquid form at temperatures as high as 450 degrees Celsius, and in fact remain in a liquid form at temperatures as high as 530 degrees Celsius, but above 450 degrees Celsius the nitrite starts to decompose.

The material properties and the equations that represent the material properties as function of temperature were found in the Coastal Chemical MSDS and Sohal et al. {Sohal, 2010}

Specific Heat, in J/kg K : $c_p = (7.2412 * 10^{-3})T^2 - 6.879\mathrm{T} + 3388.3$

Density, in kg/m^3 : $\rho = -0.7497T + 2088.93$

The thermal conductivity was kept at a constant 0.176 W/m K.

The tank sizing was based on sizing data gathered by the Electric Power Research Institute. {EPRI, 2010} One part of this report was the finding that, based on typical soil bearing loads and California state construction codes with regards to earthquakes, a packed bed tank with quartz as the bed medium and molten salt as the heat transfer fluid, tank height should not exceed 39 feet, so a height of 10 meters, or about 32.8 feet, was selected for this model. A tank diameter of 26 meters was selected because it fell roughly in the middle of the range of tank sizes considered by the EPRI report.

There are at least two compelling uses for this salt tank model. The most obvious way to implement this model is using it to model storage dynamics of concentrated solar power plants like those recently built in Spain and those currently under construction in the western United States by SolarReserve. These plants rely on their storage capabilities to be able to provide power throughout the night and during times with low solar output; a thorough understanding of how they react to different weather conditions is essential for their successful installation and for effective operation. A demonstration of this application is given later in this chapter.

The second compelling use of this model is an investigation of whether a molten salt system could be used for large scale load leveling or for storing energy generated by means other than concentrated solar. There is potential that, by employing large scale electric resistance heaters, off-peak power or excess generation by intermittent sources like large wind farms could be stored in a molten salt tank and then later recovered through a generator connected to a steam power plant fed by the tank. This might be one of the more interesting potential applications of this model, and one this author hopes to continue to explore.

Several promising-looking simulations have been run using the salt tank model, but so far they have not been verifiable. Different combinations of load application, collector application, and utilization of the internal heater all produced temperature distributions that follow the general trends that might be expected. One model is of a concentrated solar plant with a constant output of 5 MWe and with an assumed steam plant thermal efficiency of 32%. To provide for this load, the tank was modeled with a solar collector that provided a constant 46.9 MWt for eight hours, from 9:00 A.M. until 5:00 P.M. A temperature distribution for 24 hours of this system is shown in Figure 5.1.

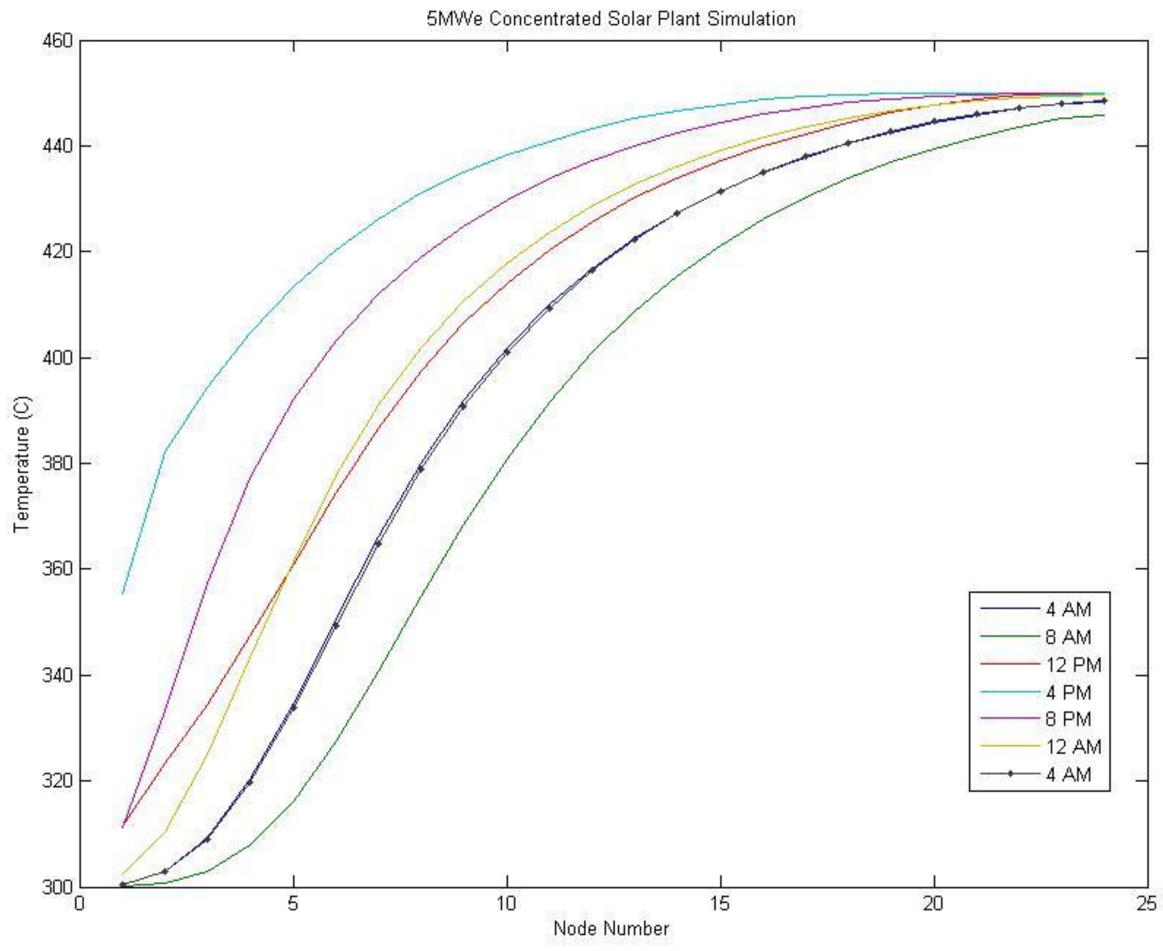


Figure 5.1 Molten Salt Tank Temperature Distribution

There are several promising things that can be observed in Figure 5.1. The most striking is that the temperature profiles at the beginning and at the end of the 24 hour period are nearly identical; indeed, a dot marker was needed on the last time (4:00 AM) curve to help distinguish it from the curve from twenty four hours earlier. This signifies that not only does the collector seem correctly sized for the application chosen, but there also seems to be a general periodicity in the temperature distribution when the tank operates steadily. The general shape of the temperature distribution curves, especially when compared to those in the water storage tank in the previous chapter, also suggests that the tank is relatively oversized for the application. This should allow the facility to continue operating during periods of low sunlight.

CHAPTER VI

CONCLUSIONS

After considering prior methods employed to simulate thermal energy storage tanks, a model was developed attempting to strike a balance between thoroughly describing the system and also producing a program that took a reasonable amount of computational time to run. A system of equations to describe the temperature distribution within the tank over time was developed as a function of the tank's operating parameters and based on a control volume analysis and then a program was developed to solve that series of equations and output the temperature distribution as well as other operational characteristics of the tank. While it is hoped that this program can be of assistance in simulating many different energy storage applications, there are still aspects of the code that can be improved upon; recommendations for further research will be present in the next chapter.

The program was also demonstrated in a couple different applications. Although the results of the sample simulations contained within this thesis are so far unverified, they seem incredibly plausible. Many storage tank dynamics that weren't specifically hard-coded into the model program, like stratification within the fluid column, can be readily observed within the temperature distributions in the simulations.

The default version of the model is generic enough to be used to simulate a wide range of energy storage applications, including those that the author has not considered in this thesis, but it can also be refined to model a specific system in greater detail with minimum effort; the

MATLAB programming language was chosen to build this model in part because one can learn how to program with it relatively quickly, it is well-documented, at time of writing it has an extensive support community online, and its built-in editor catches logical and semantic errors on the fly.

This model ought to be useful in determining the appropriate size and operating parameters for an energy storage system, and it has the potential to assist building managers and plant operators with forecasting and planning for the storage dynamics of their energy storage systems. The author hopes to continue refining the model, making it even more user-friendly, more computationally efficient, and more readily incorporated into other projects.

CHAPTER VII

RECOMMENDATIONS FOR FURTHER RESEARCH

Last Node Exception

With the current mass balance treatment in this simulation, if the last node with any mass in it has a very small amount of mass, the recalculating of masses in each node based on the density at the nodal temperatures at time t+1 can wreak havoc on the energy equation for the last node. It might only be an issue for a couple of time steps, but during those few time steps the temperature in the last node of interest can climb or plummet to temperatures that the actual system could not achieve. This is a result of the methods used to linearize the energy equations; the last node containing mass bears the brunt of the mass recalculations, absorbing any changes from the nodes below it. A correction of this was implemented; at the end of each time step the program compares the temperature at the last node with mass in it to the node below it, and if it deviates by a certain percentage, it is set to the same temperature as the node below it. The percentage it is allowed to deviate is one of the tank parameters; for a tank where water is the working fluid, a higher percentage is allowable than in tanks where the working fluid is molten salt, for instance. To change the percentage, adjust the value of the decimal parameter `percentageLNE` in the loader file.

Notification of this last node exception is programmed into the output function; every hour in which it is employed is noted in the simulation run notes column of the Excel file and the energy change in the node as a result of employing the last node exception is also recorded. In practice, the program does not need to employ the last node exception very often, but having it available has prevented several test simulations from returning temperature distributions that are physically unattainable.

At time of writing, the problem with the last node seems to be resolved; several recent test cases have been run under different parameters and none of them required the last node exception code. Should this be a happy coincidence, however, the last node exception code has been left inside the program. Finding an alternate solution to rare mass errors in the last node from the current last node exception should be one of the first courses of action if this program is to be improved upon.

Density Issues

While small changes in density from time step to time step are accounted for with a slight addition or subtraction to the mass flow rate upward through the tank, the tank model does not compare the densities of the nodes to one another. The general tank dynamic lends itself to a good level of stratification; cold fluid is inserted at the bottom of the tank and hot fluid is inserted higher in the tank. The default code treats the issue of where the return fluid from the collector is inserted the same way that the model presented by Duffie and Beckman {Duffie, 2006} does; it assumes that the fluid enters at the top of the tank and immediately sinks to the first node with a lower temperature than it, and that any mixing or other energy interactions with any nodes it passes through are neglected. If this assumption is not adequate for a given simulation or if the number of nodes that the fluid returning from the collector is allowed to be reinserted into is limited, opportunities for a denser mass of fluid to exist inside the tank at a higher node than a less dense mass increase. This has not caused any issues in the models simulated thus far because when the collector has been used in these models, it has been able to deposit the hot fluid in the upper node of the tank. For the sake of other potential applications, however, this ought to be corrected quickly.

Further Documentation

While this thesis aims to explain the development of this model, it is by no means an operating manual for the simulation programs it describes. To facilitate future researchers hoping to make use of these programs, a more thorough explanation of some of the more intricate parts of the code should and will be prepared.

Test Bed

One thing this simulation program is sorely missing is external validation based on empirical observations. My advisor, Dr. Prakash Dhamshala, has had several ideas for how a test bed could be assembled and used both to verify this and other simulations as well as to advance the research initiatives at the University of Tennessee at Chattanooga in the fields of energy storage and of low and zero net-energy buildings. One idea was that, should a solar collector be unavailable or impractical to use with the test bed, a conventional residential water heater could probably be reasonably used in its place. Another was that any thermistor or thermocouple array used to measure the temperature distribution in the tank should be positioned with some care; it should neither be in a position or of a geometry where it would noticeably affect the flow of the water in the tank, nor should it be too close to the inlet ports. By incorporating a data acquisition and control board like a National Instruments LABView system, flow from collector or water heater could be controlled from a computer, based on the current tank temperature profile as read by the thermocouple array. One potential schematic of such a test bed is presented in Figure 7.1.

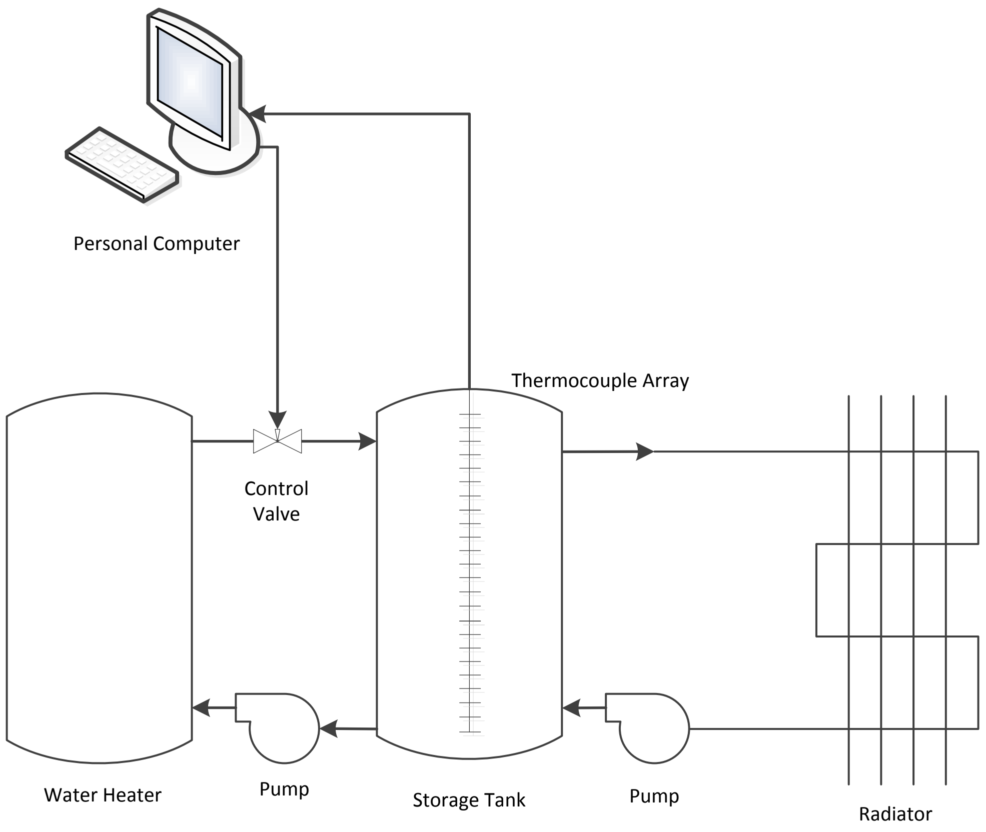


Figure 7.1 Schematic for Potential Test Bed

Multi-Threading

One of the less desirable aspects of this model can be the time that it takes a normal desktop computer to perform the simulation; on a desktop machine with a 2.8GHz processor and 4GB of RAM, it takes almost a minute to run a twenty four hour simulation with a time step size of one minute and the tank divided into twenty nodes; a simulation of tank operations for a year with those conditions would take hours to run. If one wanted to decrease the computational time required by the simulation without reducing either the number of nodes in the finite difference problem or reducing the time step size, using multiple computational threads, or multi-threading, might be advised.

Recent builds of MATLAB can come with MATLAB's multi-threading functionalities in the Parallel Computing Toolbox or PCT. This allows MATLAB to distribute computational responsibilities for portions of code that can be performed in parallel to multiple processor cores or to multiple threads within the same processor, allowing for a faster execution of the code. Unfortunately, much of the code in this model, because of the nature of this model, must be executed serially, and so the uses for the PCT in this model are limited.

One application for the PCT would be the generation of the elements of the [A] and [C] matrices before the solution of the [T] matrix. Under the current parameters, the [A] matrix cannot be assembled in parallel; since the mass flow entering each node is dependent on the mass interactions of the nodes below it, each row of the [A] matrix needs to be assembled one at a time. If the density of the working fluid in the tank were assumed to be constant, this restriction would be lifted, although reducing the complexity and verisimilitude of the model simply to reduce computational time should be given careful consideration and is generally not advised in this application. There is nothing that prevents the [C] matrix from being assembled in parallel, although to do so would require a separate `parfor` loop, and gains in speed might not be appreciable.

MATLAB's `mldivide` function, the function that solves the systems of equations for the temperature at the next time step, can benefit from using the Parallel Computing Toolbox. {Mathworks, 2012b} This might be the most noticeable application of the PCT for this model with respect to computational time saving. As the number of nodes in the finite difference problem increases, the time savings become more apparent. It should be noted, however, that if the number of nodes is very large, the simulation might better benefit from one of several MATLAB functions that use an iterative solution method. {Mathworks, 2012b}

Time Step Size

Another functionality that would be helpful as this project is continued would be either a separate program or a functionality within the tank program that decides what an appropriate time step size for the simulation would be. Currently, the default time step size is set to one minute, but the only limitation on the time step size is that there needs to be a whole number of steps every hour; twenty minutes and thirty minutes would be allowable time step sizes, but twenty five minutes would not be.

If increasing the simulation time step size from one minute to ten minutes for a given model did not noticeably alter the output, it would be desirable because it would decrease the computational time of the simulation by about a factor of ten. Code that estimates the largest time step size that still accurately captures the dynamics of the tank might be a tremendous benefit to this program and perhaps ought to be implemented after some of the more pressing issues have been addressed.

Incorporating SSPT

To more accurately and flexibly model the load, a set of Simulink blocks called the Simple Steam Power Toolbox was developed in conjunction with this simulation. The SSTP is described in further detail in Appendix E, but in short it allowed for quickly assembling and modeling potentially complicated steam power plant models in Simulink by dragging and connecting model blocks. The SSTP was intended to be used to model the load in this simulation. Instead, a much more encapsulated model of the load was used, described by the power output of the plant, the thermal efficiency of the plant, and the effectiveness of the heat exchanger. While incorporating the SSPT would have allowed for a more dynamic and flexible model of the load, it would have greatly increased the computational time of the model.

One possible source of improvement in future studies would be to incorporate a less encapsulated load model, either by incorporating the Simple Steam Power Toolbox or a similar program.

Better Collector Function

The collector function included with the simulation, which could be used to model a solar collector, a heat recovery device, or some other sort of external heater, is currently rather simplistic. It currently adds a set amount of energy to the tank every hour that it is called with a constant outlet temperature; for a heat recovery system, this model might be wholly adequate. If the system is to be used to model a system with a solar collector, however, a more refined model will probably be required.

Towards a Cleaner Code

A final, minor source for improvement would be a simplifying of the program's code. While there is nothing technically wrong with having the program assign a large number of variables as global variables or with having the simulation comprised of half a dozen or so separate files, it is perhaps in bad taste, or at least bad style. A single MATLAB file, built with nested sub-functions and composed with a firmer grasp of variable scope, might integrate with other systems more readily and would certainly be more elegant.

REFERENCES

Abraham, S. (2010). Lights Out. Ten myths about (and real solutions to) America's energy Crisis, St. Martin's Press.

Baxter, R. (2006). Energy Storage. A Nontechnical Guide. Tulsa OK, PennWell.

Bergman, T. L., A. S. Lavine, F. P. Incropera and D. P. DeWitt (2011). Fundamentals of Heat and Mass Transfer, Wiley.

Coastal Chemical Co. LLC. HITEC Heat Transfer Salt. MSDS. http://www.coal2nuclear.com/MSR%20-%20HITEC%20Heat%20Transfer%20Salt.pdf

Dhamshala, P. (2012). Transient Analysis of Building Loads and Energy Requirements (TABLER) [Computer Program]

Duffie, J. A. and W. A. Beckman (2006). Solar Engineering of Thermal Processes, Wiley.

EPRI (2010). Demonstration Development Project: Solar Thermocline Storage Systems: Preliminary Design Study.

Florides, G. A., S. A. Kalogirou, S. A. Tassou and L. C. Wrobel (2003). Design and Construction of a LiBr-water absorption machine, Energy Conversion and Management. **44:** 2483-2508.

Flueckiger, S., Z. Yang and S. V. Garimella (2011). An integrated thermal and mechanical investigation of molten-salt thermocline energy storage, Journal of Applied Energy. **88:** 2098-2105.

Flueckiger, S. M. and S. V. Garimella (2012). Second-law analysis of molten-salt thermal energy storage in thermoclines, Solar Energy. **86:** 1621 - 1631.

Ghaddar, N. K., A. M. Al-Marafie and A. Al-Kandari (1989). Numerical Simulation of Stratification Behavior in Thermal Storage Tanks, Journal of Applied Energy. **32:** 225-239.

Mathworks (2012a). xlswrite: Write Microsoft Excel spreadsheet file. MATLAB Documentation Center. http://www.mathworks.com/help/matlab/ref/xlswrite.html

Mathworks (2012b). Systems of Linear Equations. MATLAB Documentation Center. http://www.mathworks.com/help/matlab/math/systems-of-linear-equations.html

Newton, B. J. (1995). Modeling of Solar Storage Tanks. Master's Thesis, University of Wisconsin-Madison.

Özisik, M. N. (1994). Finite Difference Methods in Heat Transfer, CRC Press.

Parameshwaran, R., S. Kalaiselvam, S. Harikrishnan and A. Elayaperumal (2012). Sustainable thermal energy storage technologies for buildings: A review, Renewable and Sustainable Energy Reviews. **16:** 2394-2433.

Sohal, M. S., M. A. Ebner, P. Sabharwall and P. Sharpe (2010). Engineering Database of Liquid Salt Thermophysicial and Thermochemical Properties, Idaho National Laboratory.

Xu, C., Z. Wang, Y. He, X. Li and F. Bai (2012). Sensitivity analysis of the numerical study on the thermal performance of a packed-bed molten salt thermocline thermal storage system., Journal of Applied Energy. **92:** 65-75.

Yang, Z. and S. V. Garimella (2010a). Thermal analysis of solar thermal energy storage in a molten-salt thermocline, Journal of Solar Energy. **84:** 974-985.

Yang, Z. and S. V. Garimella (2010b). Molten-salt thermal energy storage in thermoclines under different environmental boundary conditions, Journal of Applied Energy. **87:** 3322-3329.

APPENDIX A

TANK FUNCTION SOURCE CODE

```
function [ ] = TANK(  )
% Main simulation function for the energy storage tank model.
%   Robert Buckley, UTC Chattanooga, October 9th 2012
%   After a LOADER function is called to initialize the variables for the
%   simulation, this function is called; it contains the bulk of the
%   simulation. It builds the output sheet in the Excel file, runs the
%   simulation, and then outputs the relevant data to the Excel Sheet

global tankRadius;
global tankHeight;
global tankResistance;
global tankNodeCount;
global massFlowLoad;
global initialTankTemps;
global stepSize;
global inputStepSize;
global kHTF;
global Cp_HTF;
global RHO_HTF;
global T_Env;
global T_HXL;
global T_HXC;
global totalHTFMass;
global del_H;
global del_V;
global isCharging;
global isDischarging;
global isCollecting;
global stepsPerHour;
global runNotes;
global inputRow;
global steamPlantSize;
global steamPlantEfficiency;
global fileName;
global pastMasses;
global T;
global collectorFail;
global getMassFail;
global qGenFail;
global qGenFail2;
global qToSteamPlant;
global containerRunNotes;
global KA;
global UpiD;
global percentageLNE;
global containerT;

% The following records the time that the run starts at and then formats
% the time and uses it as the sheet name to store the output in the excel
% file. It Then prints the column headings for the output sheet.
runStartTime=clock;
sheetName=num2str(runStartTime);
sheetName=regexprep(sheetName,'\s+','-');
```

```
firstTitles=[{'Hour'} {''} {'Q from Heater'} {'Q from Collector'} {'Q to Load
HX'} {'Q lost tank wall'} {'Work from Load'} {'Total Energy in Tank'} {'Q
Exception'} {'nHXC list'} {''} {''} {''} {''} {''} {''} {'Notes'}];
xlswrite(fileName,firstTitles,sheetName,'A1');

% The following is a loop to generated the temperature node titles
% (T1 T2 T3 etc) for the output file.
nodeTempTitles={};
for n=1:tankNodeCount
    nodeTempTitles(n)={['T' num2str(n)]};
end

% The following removes the initialTankTemps terms for the nodes that have
% no mass, and then calculates the total amount of energy initially
% presesnt in the tank and prints it to the excel sheet.
initialMass=GETMASS(initialTankTemps',0);
initialTankTemps(initialMass==0)=0;
initialTotalEnergyTank=sum(initialMass.*Cp_HTF(initialTankTemps').*initialTan
kTemps');

% The following writes nodeTempTitles to the output sheet starting in cell S1
xlswrite(fileName,nodeTempTitles,sheetName,'S1');
xlswrite(fileName,initialTankTemps,sheetName,'S2');
xlswrite(fileName,initialTotalEnergyTank,sheetName,'H2');
xlswrite(fileName,{'Inital Conditions. '},sheetName,'Q2');
hoursForExcel=0:numel(isCharging);
xlswrite(fileName,hoursForExcel',sheetName,'A2');

% In the following, steamPlantSize is in W, qToSteamPlant is in J,
% qToSteamPlant is the amount of energy per time step that needs to be sent
% to the steam plant in order to generate the load.
qToSteamPlant=steamPlantSize(1)*stepSize/steamPlantEfficiency;

%The tank solution will take the form [A][T]=[C], solving for [T]
A=zeros(tankNodeCount);
C=[1:tankNodeCount]';
TEMPLIST=[];

% To keep the finite difference problem linear, we will use the current
% time step and the one before it to decide changes in density and any
% other variable thermal property in the NEXT time step. To do this, we
% will store the last set of temperatures in an array "pastTemps", where
% the first column contains temperatures at t. Since we don't have a
% condition before the initial condition, "pastTemps" will be initialized
% set to the initial conditions for the first iteration.
pastTemps=[initialTankTemps'];
pastMasses=[initialMass initialMass];

% nHXCLIST is a list of every node that the flow returning from the
% collector heat exchanger is deposited into during an hour. Should
% probably not have more than a couple elements in it.
nHXCLIST=[];

% cc is the timestep counter, driving the while loop that is each timestep.
```

```
cc=1;

% inputRow is the row number of the input variables we are drawing from
% so if the simulation is in the first hour, inputRow=1. this will be
% used to pull T_ENV, chargeState, etc. from their respective input
% tables.
inputRow=1;

T_ENV=T_Env(inputRow);
chargeState=isCharging(inputRow);
dischargeState=isDischarging(inputRow);
collectorState=isCollecting(inputRow);
Q_from_heater=0;
Q_lost_conv=0;
Q_LOAD=0;
POWER_LOAD=0;
Q_COLLECTOR=0;
Q_Exception=0;
q_LOSS=0;
mHXC=[0 0];
collectorFail=0;
getMassFail=0;
qGenFail=0;
qGenFail2=0;
dischargeFail=0;
lastNodeFail=0;
containerT=zeros(numel(isCharging),tankNodeCount);
containerQ_from_heater=zeros(numel(isCharging),1);
containerQ_COLLECTOR=zeros(numel(isCharging),1);
containerQ_Exception=zeros(numel(isCharging),1);
containerQ_LOAD=zeros(numel(isCharging),1);
containerQ_lost_conv=zeros(numel(isCharging),1);
containerPOWER_LOAD=zeros(numel(isCharging),1);
containerTotalEnergyTank=zeros(numel(isCharging),1);
containerNHXCLIST=cell(numel(isCharging),1);
containerRunNotes=cell(numel(isCharging),1);

% We need to initialize T as the initial conditions
T=initialTankTemps';

%The massFlowLoad, or the mass flow from the tank to the load heat
%exchanger, is updated at the end of each iteration. It needs a value for
%that first iteration. For simplicity, let's assume during that first small
%step, the load has not been engaged and no flow goes to it.
massFlowLoad=0;

while cc<numel(isCharging)*stepsPerHour+1

    % The following if-statement checks to make sure that inputStepSize is
    % a whole multiple of stepSize. If it isn't, it breaks the whole while
    % loop.

    if mod(stepsPerHour,1)>0
```

```
        disp('ERROR - inputStepSize is not a multiple of stepSize. correct
this.');
        break
    end

    % The following if statement checks to see if the current time step is
    % the begining of a new hour. If it is, it records values of interest
    % to the container and advances the inputRow by one.

    if mod(cc,stepsPerHour)==0

totalEnergyTank=sum(pastMasses(:,2).*Cp_HTF(pastTemps(:)).*pastTemps(:));
        containerT(inputRow,:)=T';
        containerQ_from_heater(inputRow,:)=Q_from_heater;
        containerQ_COLLECTOR(inputRow,:)=Q_COLLECTOR;
        containerQ_LOAD(inputRow,:)=Q_LOAD;
        containerQ_lost_conv(inputRow,:)=Q_lost_conv;
        containerQ_Exception(inputRow,:)=Q_Exception;
        containerPOWER_LOAD(inputRow,:)=POWER_LOAD;
        containerTotalEnergyTank(inputRow,:)=totalEnergyTank;
        containerNHXCLIST(inputRow,:)={num2str(unique(nHXCLIST))};
        containerRunNotes(inputRow,:)={runNotes};
        inputRow=inputRow+1;
        runNotes=[];
        Q_from_heater=0;
        Q_lost_conv=0;
        Q_LOAD=0;
        POWER_LOAD=0;
        Q_COLLECTOR=0;
        Q_Exception=0;
        collectorFail=0;
        getMassFail=0;
        qGenFail=0;
        qGenFail2=0;
        lastNodeFail=0;
        dischargeFail=0;
        nHXCLIST=[];
        % The following if statement allows the program to run results for
        % the last hour by suppressing the following calls.
        if cc< numel(isCharging)*stepsPerHour;
            T_ENV=T_Env(inputRow);
            chargeState=isCharging(inputRow);
            dischargeState=isDischarging(inputRow);
            collectorState=isCollecting(inputRow);
            if numel(steamPlantSize)>1

qToSteamPlant=steamPlantSize(inputRow)*stepSize/steamPlantEfficiency;
            end
        end
    end
    mHXC(1)=mHXC(2);
    [mHXC(2) T_HXC]=COLLECTOR(pastTemps(1,1));
    massIn=massFlowLoad;

    % The following if-statement calculates which node the flow from the
    % collector is dumped into, if there is a flow from the collector
    % during that time step. It does this by finding the first node whose
    % temperature is greater than the return temperature from the
    % collector, and then inserts it into the node below that one. If the
    % first node has a temperature greater than T_HXC, then the first sub
    % if-statement tells the program to insert the mass from the collector
    % into node 1 instead of node 0, because there is no node 0. If this
    % first assignment results in an empty variable( IE if none of the
    % nodes have a temperature greater than T_HXC) then it chooses the last
    % node in the tank that is not empty.
    if collectorState==1
        Q_COLLECTOR=Q_COLLECTOR +mHXC(1)*stepSize*(Cp_HTF(T_HXC)*T_HXC -
Cp_HTF(pastTemps(1,1))*pastTemps(1,1));
        nHXC=find(pastTemps>T_HXC,1)-1;
        if nHXC==0
            nHXC=1;
        elseif isempty(nHXC)==true
            nHXC=find(pastMasses(:,2),1,'last');
        end
        if ismember(nHXC,nHXCLIST)==0
            nHXCLIST=[nHXCLIST nHXC];
        end
    else
        nHXC=0;
    end

    q_Gen=QGEN(pastTemps,chargeState);
    % Q_from_heater is the total amount of energy deposited into the tank
    % over the course of the hour by the heater
    Q_from_heater=Q_from_heater+sum(q_Gen);
    Q_lost_conv=Q_lost_conv + sum(q_LOSS);
    q_LOSS=zeros(tankNodeCount,1);

    A=zeros(tankNodeCount);
    % Begin Generating matrix coefficients for node #1
    n=1;

    massOut=massIn-mHXC(2) +(pastMasses(n,1)-pastMasses(n,2))/stepSize;
    if nHXC==1
        massOut=massOut +mHXC(1);
    end
    W1=pastMasses(n,2)*Cp_HTF(pastTemps(n,1))/stepSize;
    W2=mHXC(2)*Cp_HTF(pastTemps(n,1));
    W3=massOut*Cp_HTF(pastTemps(n,1));
    W4=massOut*Cp_HTF(pastTemps(n+1,1));
    W5=massIn*Cp_HTF(T_HXL)*T_HXL;
    A(n,n)= W1 +W2 +KA/del_H +UpiD*del_H;
    A(n,n+1)=-KA/del_H;
    C(n,n)=pastTemps(n,1)*W1 + W5 +UpiD*del_H*T_ENV + q_Gen(n);
    if massOut>0
        A(1,1)=A(1,1)+W3;
    else
        A(1,2)=A(1,2)+W4;
    end
    if nHXC==1
        C(n,1)=C(n,1)+ mHXC(1)*Cp_HTF(T_HXC)*T_HXC;
    end
    q_LOSS(n,1)=UpiD*del_H*stepSize*(pastTemps(n,1)-T_ENV);
    massIn=massOut;

    %Now to generate the coefficients for the middle volumes
    n=2;
    doneGenerating=false;
    while n<tankNodeCount

        % Now to generate the values of A(n,:) for n(2:end-1) under the
        % cases listed on the flowchart

        % CASE #7 - node is empty
        if doneGenerating==true
            A(n,n-1)=0;
            A(n,n)=0;
            A(n,n+1)=0;
            C(n,1)=0;
            q_LOSS(n,1)=0;
        end

        % CASE #3 / CASE #8
        if pastMasses(n+1,1) > 0.99*del_V*RHO_HTF(pastTemps(n+1,1))

            massOut=massIn +(pastMasses(n,1)-pastMasses(n,2))/stepSize;
            if nHXC==n
                massOut=massOut +mHXC(1);
            end
            W1=pastMasses(n,2)*Cp_HTF(pastTemps(n,1))/stepSize;
            W2=massOut*Cp_HTF(pastTemps(n,1));
            W3=massOut*Cp_HTF(pastTemps(n+1,1));
            W4=massIn*Cp_HTF(pastTemps(n,1));
            W5=massIn*Cp_HTF(pastTemps(n-1,1));
            A(n,n-1)=-KA/del_H;
            A(n,n)= W1 + 2*KA/del_H + UpiD*del_H;
            A(n,n+1)=-KA/del_H;
            if massIn>0
                A(n,n-1)=A(n,n-1)-W5;
            else
                A(n,n)=A(n,n)-W4;
            end
            if massOut>0
                A(n,n)=A(n,n)+W2;
            else
                A(n,n+1)=A(n,n+1)+W3;
            end
            C(n,1)=pastTemps(n,1)*W1 +UpiD*del_H*T_ENV + q_Gen(n);
            if n==nHXC
                C(n,1)=C(n,1)+ mHXC(1)*Cp_HTF(T_HXC)*T_HXC;
            end
            q_LOSS(n,1)=UpiD*del_H*stepSize*(pastTemps(n,1)-T_ENV);
            massIn=massOut;

        % CASE #4 / CASE #9
```

```
        elseif pastMasses(n+1,1) > 0

            massOut=massIn +(pastMasses(n,1)-pastMasses(n,2))/stepSize;
            if nHXC==n
                massOut=massOut +mHXC(1);
            end
            W1=pastMasses(n,2)*Cp_HTF(pastTemps(n,1))/stepSize;
            W2=massOut*Cp_HTF(pastTemps(n,1));
            W3=massOut*Cp_HTF(pastTemps(n+1,1));
            W4=massIn*Cp_HTF(pastTemps(n,1));
            W5=massIn*Cp_HTF(pastTemps(n-1,1));
            dx1=0.5*del_H
+(0.5*pastMasses(n+1,2))/(RHO_HTF(pastTemps(n+1,1))*pi*tankRadius^2);
            A(n,n-1)=-KA/del_H;
            A(n,n)= W1 + KA/del_H +KA/dx1 + UpiD*del_H;
            A(n,n+1)=-KA/dx1;
            if massIn>0
                A(n,n-1)=A(n,n-1)-W5;
            else
                A(n,n)=A(n,n)-W4;
            end
            if massOut>0
                A(n,n)=A(n,n)+W2;
            else
                A(n,n+1)=A(n,n+1)+W3;
            end
            C(n,1)=pastTemps(n,1)*W1 +UpiD*del_H*T_ENV + q_Gen(n);
            if n==nHXC
                C(n,1)=C(n,1)+ mHXC(1)*Cp_HTF(T_HXC)*T_HXC;
            end
            q_LOSS(n,1)=UpiD*del_H*stepSize*(pastTemps(n,1)-T_ENV);
            massIn=massOut;

        % CASE #5 / CASE #10
        elseif del_V*RHO_HTF(pastTemps(n,1)) >=
pastMasses(n,2)+(massIn+mHXC*isequal(n,nHXC))*stepSize

            % In this section, m2 is a mass, not a mass flow rate.
            m2=pastMasses(n,2)+massIn*stepSize;
            if nHXC==n
                m2=m2 +mHXC(1)*stepSize;
            end
            dx2=(pastMasses(n,2) +
m2)/(RHO_HTF(pastTemps(n,1))*2*pi*tankRadius^2);
            dx3=0.5*(del_H+dx2);
            W1=m2*Cp_HTF(pastTemps(n,1))/stepSize;
            W2=pastMasses(n,2)*Cp_HTF(pastTemps(n,1))/stepSize;
            W3=massIn*Cp_HTF(pastTemps(n,1));
            W4=massIn*Cp_HTF(pastTemps(n-1,1));
            A(n,n-1)=-KA/dx3;
            A(n,n)= W1 + KA/dx3 + UpiD*dx2;
            A(n,n+1)=0;
            if massIn>0
                A(n,n-1)=A(n,n-1)-W4;
            else
```

```
                A(n,n)=A(n,n)-W3;
            end
            C(n,1)=pastTemps(n,1)*W2 +UpiD*dx2*T_ENV + q_Gen(n);
            if n==nHXC
                C(n,1)=C(n,1)+ mHXC(1)*Cp_HTF(T_HXC)*T_HXC;
            end
            q_LOSS(n,1)=UpiD*dx2*stepSize*(pastTemps(n,1)-T_ENV);
            doneGenerating=true;

        %CASE #6 / CASE #11
        elseif del_V*RHO_HTF(pastTemps(n,1)) <
pastMasses(n,2)+(massIn+mHXC*isequal(n,nHXC))*stepSize

            massOut=massIn +(pastMasses(n,2)-
RHO_HTF(pastTemps(n,1))*del_V)/stepSize;
            if nHXC==n
                massOut=massOut +mHXC(1);
            end
            W1=RHO_HTF(pastTemps(n,1))*del_V*Cp_HTF(pastTemps(n,1))/stepSize;

W2=pastMasses(n,2)*pastTemps(n,1)*Cp_HTF(pastTemps(n,1))/stepSize;
            W3=massIn*Cp_HTF(pastTemps(n,1));
            W4=massIn*Cp_HTF(pastTemps(n-1,1));
            W5=massOut*Cp_HTF(pastTemps(n,1));
            A(n,n-1)=-KA/del_H;
            A(n,n)= W1 + W5+ KA/del_H + UpiD*del_H;
            A(n,n+1)=-KA/del_H;
            A(n+1,n+1)=-1;
            A(n+1,n)=1;
            C(n,1)=W2+ UpiD*del_H*T_ENV + q_Gen(n);
            C(n+1,1)=0;
            if massIn>0
                A(n,n-1)=A(n,n-1)-W4;
            else
                A(n,n)=A(n,n)-W3;
            end
            if n==nHXC
                C(n,1)=C(n,1)+ mHXC(1)*Cp_HTF(T_HXC)*T_HXC;
            end
            q_LOSS(n,1)=UpiD*del_H*stepSize*(pastTemps(n,1)-T_ENV);
            doneGenerating=true;

            %CASE #7 (again, just in case)
        else
            A(n,n-1)=0;
            A(n,n)=0;
            A(n,n+1)=0;
            C(n,1)=0;
            q_LOSS(n,1)=0;
        end
        n=n+1;
    end

    % Now to generate the last part of the tank, the uppermost differential
    % volume. First we'll check to see if there is a need for an upper
    % volume.
    if sum(pastMasses(1:end-1,2)) < 0.998*totalHTFMass
        dx2=del_H*pastMasses(n,2)/RHO_HTF(pastTemps(n,1));
        dx3=(del_H+dx2)*0.5;
        Q1=massIn/pastMasses(n,2);

Q6=(kHTF*stepSize*pi*tankRadius^2)/(pastMasses(n,2)*Cp_HTF(pastTemps(n,1))*dx
3);

Q8=(tankResistance*2*pi*tankRadius*dx2*stepSize)/(pastMasses(n,2)*Cp_HTF(past
Temps(n,1)));
        Q9=(q_Gen(n)*stepSize)/(pastMasses(n,2)*Cp_HTF(pastTemps(n,1)));
        A(end,end-1)=-Q1-Q6;
        A(end,end)=1+Q6+Q8;
        C(end,1)=pastTemps(end,1) +T_ENV*Q8+Q9;

q_LOSS(end,1)=tankResistance*2*pi*tankRadius*dx2*stepSize*(pastTemps(n,1)-
T_ENV);
    else
        A(end,end-1)=0;
        A(end,end)=0;
        C(end,1)=0;
        q_LOSS(end,1)=0;
    end

    % Since the tank can have empty layers, we can't use a simple matrix
    % inversion to solve this as is. What we can do is knock the zeros off
    % the bottom of the 'C' column, resize A accordingly, and then solve
    % for T =A\C and then re-tack on the missing empty values if needed.
    % otter is the number of non-zero values in C
    otter=numel(find(C));
    Cprime = C(1:otter);
    Aprime= A(1:otter,1:otter);
    T=Aprime\Cprime;
    while numel(T)<tankNodeCount
        T=vertcat(T,0);
    end
    pastMasses(:,1)=[];
    recalcMasses=GETMASS(T,mHXC(2));
    pastMasses=[pastMasses recalcMasses];
    T(recalcMasses==0)=0;
    lastNode=find(T,1,'last');

    % The following is the Last Node Exception code. If the temperature of
    % the last node containing mass starts to deviate wildly from the node
    % below it, this will catch it, set the temperature equal to the node
    % below it, note what was done and record to the output sheet what the
    % energy change to the system as a result was.
   if abs((T(lastNode)-T(lastNode-1))/T(lastNode))>percentageLNE
       lastNodeAdjustment=pastMasses(lastNode,2)*(Cp_HTF(T(lastNode-
1,1))*T(lastNode-1,1) - Cp_HTF(T(lastNode,1))*T(lastNode,1))/stepSize;
       Q_Exception=Q_Exception + lastNodeAdjustment;
       T(lastNode)=T(lastNode -1);
      if lastNodeFail==0
          lastNodeFail=1;
```

```
           runNotes=[runNotes 'Last node exception applied. '];
       end
   end

    pastTemps=T;

    dischargeTempRatio=pastTemps(lastNode-1,1)/T_HXL;
    if dischargeState==1
        if dischargeTempRatio>1.1
            delhSalt=pastTemps(lastNode-1,1)*Cp_HTF(pastTemps(lastNode-1,1))-
T_HXL*Cp_HTF(T_HXL);
            massToLoad=qToSteamPlant/delhSalt;
            Q_LOAD=Q_LOAD+qToSteamPlant;
            POWER_LOAD=POWER_LOAD+qToSteamPlant*steamPlantEfficiency;
            if pastMasses(lastNode,2)>massToLoad
                pastMasses(lastNode,2)=pastMasses(lastNode,2)-massToLoad;
            else
                panther=pastMasses(lastNode-1,2)+pastMasses(lastNode,2)-
massToLoad;
                pastMasses(lastNode,2)=0;
                pastMasses(lastNode - 1,2)=panther;
                pastTemps(lastNode,1)=0;
            end
            massFlowLoad=massToLoad/stepSize;
       elseif dischargeFail==0
       dischargeFail=1;
       runNotes=[runNotes 'Load was not applied for part of this hour; tank
temperature too low. '];
       massToLoad=0;
       massFlowLoad=0;
        else
       massToLoad=0;
       massFlowLoad=0;
        end
    end
    cc=cc+1;

end %end of main loop

    %Start of Output to Excel code
        xlswrite(fileName,containerT,sheetName,'S3');
        xlswrite(fileName,containerQ_from_heater,sheetName,'C3');
        xlswrite(fileName,containerQ_COLLECTOR,sheetName,'D3');
        xlswrite(fileName,containerQ_LOAD,sheetName,'E3');
        xlswrite(fileName,containerQ_lost_conv,sheetName,'F3');
        xlswrite(fileName,containerPOWER_LOAD,sheetName,'G3');
        xlswrite(fileName,containerTotalEnergyTank,sheetName,'H3');
        xlswrite(fileName,containerQ_Exception,sheetName,'I3');
        xlswrite(fileName,containerNHXCLIST,sheetName,'J3');
        xlswrite(fileName,containerRunNotes,sheetName,'Q3');

disp('Tank Simulation Complete.');
end %TANK end
```

APPENDIX B

WATER AND SALT LOADER FUNCTIONS SOURCE CODE

```
function [] = WATERLOADER(excelFileName)
%Initializes global variables and loads input data for water storage project
%   Robert Buckley, UT Chattanooga, August 27th, 2012
%   excelFileName is the name of the Excel file that we are loading from

%TANK PARAMETERS
evalin('base','global tankRadius;');
evalin('base','global tankHeight;');
evalin('base','global tankResistance;');
evalin('base','global tankNodeCount;');
evalin('base','global chargePerHour;');
evalin('base','tankRadius = 1.25;');
evalin('base','tankHeight = 4;');
evalin('base','tankResistance = 0.438;');
evalin('base','tankNodeCount = 20;');
evalin('base','chargePerHour=50000;');
global effectivenessHXL;
effectivenessHXL=1;
evalin('base','global effectivenessHXL;');
global steamPlantEfficiency;
steamPlantEfficiency=0.6;
evalin('base','global steamPlantEfficiency;');
global steamPlantSize;
steamPlantSize=8434;
evalin('base','global steamPlantSize;');

%SIMULATION PARAMETERS
evalin('base','global stepSize;');
evalin('base','global recorderStepSize;');
evalin('base','global inputStepSize;');
evalin('base','stepSize=60;');
evalin('base','recorderStepSize=60;');
evalin('base','inputStepSize=3600;');
evalin('base','global runNotes;');
evalin('base','global inputRow;');
global inputStepSize;
global stepSize;
global stepsPerHour;
stepsPerHour=inputStepSize/stepSize;
evalin('base','global stepsPerHour;');
global fileName
fileName=excelFileName;
evalin('base','global fileName;');
global RHO_HTF;
RHO_HTF=inline('0.000015451*T.^3 -0.0059003*T.^2-0.019075*T + 1002.3052');
evalin('base','global RHO_HTF;');
global Cp_HTF;
Cp_HTF=inline('(0.000003216145833*T.^4 -0.000798668982*T.^3
+0.0780295139*T.^2 -3.0481614*T + 4217.7377)');
evalin('base','global Cp_HTF;');
% thermal conductivity is assumed constant over the temperature range with
% units of w/mK
evalin('base','global kHTF;');
global kHTF;
kHTF=0.558;
```

```
evalin('base','global totalHTFMass;');
evalin('base','totalHTFMass= 17300;');
evalin('base','global critTempLow;');
evalin('base','critTempLow=60;');
evalin('base','global critTempHigh;');
evalin('base','critTempHigh=98;');
evalin('base','global overrideTempLow;');
evalin('base','overrideTempLow=55;');

%Now to open the Excel File and build the load tables
[excelNums excelTxt excelRaw]=xlsread(excelFileName);

%This finds the heading "isCharging" in the excel file, then builds a
%column vector in MATLAB with the data under that heading, and then
%declares it as global in the base workspace so it can be used in any
%part of the simulation. isCharging is true (1) if the heater is in use,
%isDischarging is true if the load (turbine) is in use, isCollecting is
%true if the collector is in use.
isChargingColumn = ismember(excelTxt, 'isCharging')==1;
global isCharging;
isCharging = excelNums(:,isChargingColumn);
evalin('base','global isCharging;');

isDischargingColumn = ismember(excelTxt, 'isDischarging')==1;
global isDischarging;
isDischarging = excelNums(:,isDischargingColumn);
evalin('base','global isDischarging;');

if length(isCharging)~= length(isDischarging)
    disp('Error - Charging data different length than Discharging data.');
end
    %makes sure that the two columns are the same length
isCollectColumn = ismember(excelTxt, 'isCollect')==1;
global isCollecting;
isCollecting = excelNums(:,isCollectColumn);
evalin('base','global isCollecting;');

hourlyLoadColumn = ismember(excelTxt, 'hourlyLoad')==1;
if sum(hourlyLoadColumn)>0
    steamPlantSize=excelNums(:,hourlyLoadColumn)';
end

isCollecting = excelNums(:,isCollectColumn);
evalin('base','global isCollecting;');
T_EnvColumn = ismember(excelTxt, 'TEnv')==1;
global T_Env;
T_Env = excelNums(:,T_EnvColumn);
evalin('base','global T_Env;');

%This defines a default set of initial tank temperatures in a HORIZONTAL
%array, where the first element of the array represents the bottom of the
% tank then searches the text of the excel file for the term
%"initialTankTemps". If it finds it, it sets the array of initial tank
%temperatures to the data from the excel sheet, if not it uses the default.
```

```
%Then it erases any zeros from the array and declares it as global in the
%base workspace.

global tankNodeCount;

%gives an even distribution of temperatures from ~75 to 85
defaultInitialTankTemps = 75+[1:tankNodeCount]*(10/tankNodeCount);
global initialTankTemps;
initialTankTempsColumn = ismember(excelTxt, 'initialTankTemps')==1;
if sum(initialTankTempsColumn)==0
    initialTankTemps=defaultInitialTankTemps;
else
    initialTankTemps=excelNums(:,initialTankTempsColumn)';
end
initialTankTemps(initialTankTemps==0)=[];
evalin('base','global initialTankTemps;');

defaultCondTemp=20;
global condenserTemperature;
condTempsColumn = ismember(excelTxt, 'condenserTemperature')==1;
if sum(condTempsColumn)==0
    condenserTemperature=defaultCondTemp;
else
    condenserTemperature=excelNums(:,condTempsColumn)';
end
evalin('base','global condenserTemperature;');

% del_H is the height of a single control volume within the tank, del_V is
% the volume.
global tankHeight;
global tankRadius;
global del_H;
global del_V;
del_H= tankHeight / tankNodeCount;
del_V=del_H*pi*tankRadius^2;
global T_HXL;
T_HXL=70;
global KA;
KA=kHTF*pi*tankRadius*tankRadius;
global UpiD;
global tankResistance;
UpiD=tankResistance*pi*2*tankRadius;
global qCollector;
global THXCdefault;
qCollector=152500000;
THXCdefault=90;
evalin('base','global qCollector;');
evalin('base','global THXCdefault;');
global percentageLNE
percentageLNE=0.18;
disp('Variable Initializing Complete.');
end % End of Water Loader Function
```

```
function [] = SALTLOADER(excelFileName)
%Initializes global variables and loads input data for molten salt project
%   Robert Buckley, UT Chattanooga, August 27th, 2012
%   excelFileName is the name of the Excel file that we are loading from

%TANK PARAMETERS
evalin('base','global tankRadius;');
evalin('base','global tankHeight;');
evalin('base','global tankResistance;');
evalin('base','global tankNodeCount;');
evalin('base','global chargePerHour;');
evalin('base','tankRadius = 13;');
evalin('base','tankHeight = 10;');
evalin('base','tankResistance = 0.0678;');
evalin('base','tankNodeCount = 25;');
evalin('base','chargePerHour=900000000;');
global effectivenessHXL;
effectivenessHXL=0.8;
evalin('base','global effectivenessHXL;');
global steamPlantEfficiency;
steamPlantEfficiency=0.32;
evalin('base','global steamPlantEfficiency;');
global steamPlantSize;
steamPlantSize=5000000;
evalin('base','global steamPlantSize;');

%SIMULATION PARAMETERS

evalin('base','global stepSize;');
evalin('base','global recorderStepSize;');
evalin('base','global inputStepSize;');
evalin('base','stepSize=60;');
evalin('base','recorderStepSize=60;');
evalin('base','inputStepSize=3600;');
evalin('base','global runNotes;');
evalin('base','global inputRow;');
global inputStepSize;
global stepSize;
global stepsPerHour;
stepsPerHour=inputStepSize/stepSize;
evalin('base','global stepsPerHour;');

global fileName
fileName=excelFileName;
evalin('base','global fileName;');
global RHO_HTF;
RHO_HTF=inline('2088.93 - 0.7497*T');
evalin('base','global RHO_HTF;');
global Cp_HTF;
Cp_HTF=inline('3388.3 - 6.879*T + 0.0072412*T.^2');
evalin('base','global Cp_HTF;');
% thermal conductivity is assumed constant over the temperature range with
% units of w/mK
evalin('base','global kHTF;');
global kHTF;
```

```
kHTF=0.176;
evalin('base','global totalHTFMass;');
evalin('base','totalHTFMass=9.00e+06;');
evalin('base','global critTempLow;');
evalin('base','critTempLow=285;');
evalin('base','global critTempHigh;');
evalin('base','critTempHigh=520;');
evalin('base','global overrideTempLow;');
evalin('base','overrideTempLow=260;');

%Now to open the Excel File and build the load tables
[excelNums excelTxt excelRaw]=xlsread(excelFileName);

%This finds the heading "isCharging" in the excel file, then builds a
%column vector in MATLAB with the data under that heading, and then
%declares it as global in the base workspace so it can be used in any
%part of the simulation. isCharging is true (1) if the heater is in use,
%isDischarging is true if the load (turbine) is in use, isCollecting is
%true if the collector is in use.

isChargingColumn = ismember(excelTxt, 'isCharging')==1;
global isCharging;
isCharging = excelNums(:,isChargingColumn);
evalin('base','global isCharging;');

isDischargingColumn = ismember(excelTxt, 'isDischarging')==1;
global isDischarging;
isDischarging = excelNums(:,isDischargingColumn);
evalin('base','global isDischarging;');

if length(isCharging)~= length(isDischarging)
    disp('Error - Charging data different length than Discharging data.');
end
    %makes sure that the two columns are the same length

isCollectColumn = ismember(excelTxt, 'isCollect')==1;
global isCollecting;
isCollecting = excelNums(:,isCollectColumn);
evalin('base','global isCollecting;');

T_EnvColumn = ismember(excelTxt, 'TEnv')==1;
global T_Env;
T_Env = excelNums(:,T_EnvColumn);
evalin('base','global T_Env;');

global tankNodeCount;

%This defines a default set of initial tank temperatures in a HORIZONTAL
%array, where the first element of the array represents the bottom of the
% tank then searches the text of the excel file for the term
%"initialTankTemps". If it finds it, it sets the array of initial tank
%temperatures to the data from the excel sheet, if not it uses the default.
%Then it erases any zeros from the array and declares it as global in the
%base workspace.
```

```
%gives an even distribution of temperatures from ~350 to 450
defaultInitialTankTemps = 350+[1:tankNodeCount]*(100/tankNodeCount);

global initialTankTemps;
initialTankTempsColumn = ismember(excelTxt, 'initialTankTemps')==1;
if sum(initialTankTempsColumn)==0
    initialTankTemps=defaultInitialTankTemps;
else
    initialTankTemps=excelNums(:,initialTankTempsColumn)';
end
initialTankTemps(initialTankTemps==0)=[];
evalin('base','global initialTankTemps;');

defaultCondTemp=20;
global condenserTemperature;
condTempsColumn = ismember(excelTxt, 'condenserTemperature')==1;
if sum(condTempsColumn)==0
    condenserTemperature=defaultCondTemp;
else
    condenserTemperature=excelNums(:,condTempsColumn)';
end
evalin('base','global condenserTemperature;');

% del_H is the height of a single control volume within the tank, del_V is
% the volume.
global tankHeight;
global tankRadius;
global del_H;
global del_V;
del_H= tankHeight / tankNodeCount;
del_V=del_H*pi*tankRadius^2;
global reasonableSaltMass;
reasonableSaltMass=sum(del_V.*RHO_HTF(initialTankTemps(1:end)))*0.955;
global T_HXL;
T_HXL=300;
global KA;
KA=kHTF*pi*tankRadius*tankRadius;
global UpiD;
global tankResistance;
UpiD=tankResistance*pi*2*tankRadius;
global qCollector;
global THXCdefault;
qCollector=168900000000;
THXCdefault=450;
evalin('base','global qCollector;');
evalin('base','global THXCdefault;');
global percentageLNE
percentageLNE=0.02;
disp('Variable Initializing Complete.');
end % End of Salt Loader Function
```

APPENDIX C

ANCILLARY FUNCTIONS SOURCE CODE

```
function [ mHXC THXC ] = COLLECTOR( tempBottomOfTank )
%COLLECTOR To be used to simulate an energy collection system to be used
%with the storage tank program.
%   Robert Buckley, UT Chattanooga, October 9th 2012
%   This is a very simple solar collector model to be used to test the
%   system and as a template for more sophisticated models.

global inputRow;
global isCollecting;
global stepsPerHour;
global Cp_HTF;
global runNotes;
global collectorFail;
global qCollector;
global THXCdefault;
global stepSize;

qInPerHour=qCollector;
THXC=THXCdefault;
tempRatio=THXC/tempBottomOfTank;

% Currently a very simple collector model. Assumes a constant outlet
% temperature and a constant rate of heat addition when the collector is in
% use.

if inputRow<=numel(isCollecting)
    if isCollecting(inputRow)==1
        qInStep = qInPerHour/stepsPerHour;
        CPdelT=Cp_HTF(THXC)*THXC - Cp_HTF(tempBottomOfTank)*tempBottomOfTank;
        mHXC= qInStep / CPdelT;
        if tempRatio<1.035
            if collectorFail==0
               collectorFail=1;
                runNotes=[runNotes 'Collector deemed ineffective and not used
during part of this hour. '];
            end
        mHXC=0;
        end
    else
        mHXC=0;
    end
else
    if isCollecting(end)==1
        qInStep = qInPerHour/stepsPerHour;
        CPdelT=Cp_HTF(THXC)*THXC - Cp_HTF(tempBottomOfTank)*tempBottomOfTank;
        mHXC= qInStep / CPdelT;
        if tempRatio<1.035
            if collectorFail==0
                collectorFail=1;
                runNotes=[runNotes 'Collector deemed ineffective and not used
during part of this hour. '];
            end
        mHXC=0;
        end
    else
```

```
        mHXC=0;
    end
end

if mHXC<0
    if collectorFail==0
        collectorFail=1;
        runNotes=[runNotes 'Collector deemed ineffective and not used during
part of this hour. '];
    end
    mHXC=0;
end

%The tank funcion expects COLLECTOR to return a mass flow rate; the
%following converts the mass to an average mass flow rate
mHXC=mHXC/stepSize;
end %end of COLLECTOR
```

```
function [ massCol ] = GETMASS( tempsCol, mHXC2 )

%GETMASS calculates the mass present in each control volume
%   Robert Buckley, UT Chattanooga, October 9th

global tankNodeCount;
global del_V;
global totalHTFMass;
global RHO_HTF;
global getMassFail;
global runNotes;
global stepSize

massCol=del_V.*RHO_HTF(tempsCol);
massHXC2=mHXC2*stepSize;

% The next while loop makes sure that the mass in the tank does not exceed
% the maximum mass in the system minus whatever amount might be in the
% collector. It does this by comparing the mass if every volume was full to
% the total mass, and then sets the upper tanks as empty until the two
% equal out.
cc=tankNodeCount;
while sum(massCol)>totalHTFMass-mHXC2 && cc>0
    differ=sum(massCol)-totalHTFMass+mHXC2;
    if differ>massCol(cc)
        massCol(cc)=0;
    else
        massCol(cc)=massCol(cc)-differ;
    end
    cc=cc-1;
end

% The following tests to make sure that the volume that the mass ought to
% take up does not exceed the volume of the tank; if it does, it removes
% some mass and puts a note in that hour that there was an overflow.
if sum(massCol)<totalHTFMass
   if getMassFail==0
       getMassFail=1;
       runNotes=[runNotes 'Error: volume of heat transfer fluid exceeds tank
storage volume. '];
   end
   totalHTFMass=sum(massCol);
end
end %end of GETMASS
```

```
function [Qout ] = QGEN( pastTemps, chargeState )
% QGEN A control function to decide when and how the tank is charged by the
% internal heater.
%   Robert Buckley, UT Chattanooga, October 9th 2012

global tankNodeCount;
global critTempHigh;
global critTempLow;
global chargePerHour;
global overrideTempLow
global stepsPerHour;
global runNotes;
global qGenFail;
global qGenFail2;

% critTempHigh is the critical temperature which the tank should not be
% charged above. if Tn>critTempHigh, there will be no energy sent to the
% heater in that node.
% critTempLow is the critical temperature at which the tank will stop
% evenly distributing the heating load and concentrate it only on the nodes
% below this temperature.
% overrideTempLow is the temperature at which the heater will turn itself
% on even if the tank is not in charging mode.

% The following removes the last node from consideration for the QGEN
% function.
pastTemps(find(pastTemps,1,'last'))=0;

numNodes=numel(find(pastTemps));
lowNodes=find(pastTemps<critTempLow & pastTemps>0);
numLowNodes=numel(lowNodes);
overrideNodes=find(pastTemps<overrideTempLow & pastTemps>0);
numOverrideNodes=numel(overrideNodes);
highNodes=find(pastTemps>critTempHigh);
numHighNodes=numel(highNodes);

% Charging conditions:
% (1): Override conditions
% (2): There are nodes below critLowTemp and tank is charging.
% (3): There are no nodes below critLowTemp but there are some below
% critHighTemp and tank is charging.
% (4): Entire tank is above critHighTemp but tank is charging.
% (5): Tank is not charging.

if numOverrideNodes>0
    chargePerNode=chargePerHour/(stepsPerHour*numOverrideNodes);
    Qout(overrideNodes)=chargePerNode;
    if qGenFail==0
        qGenFail=1;
        runNotes=[runNotes 'Tank temperature dropped below the override
temperature in at least one node; internal heater employed. '];
    end
elseif numLowNodes>0 && chargeState==1
    chargePerNode=chargePerHour/(stepsPerHour*numLowNodes);
    Qout(lowNodes)=chargePerNode;
```

```
elseif numHighNodes<numNodes && chargeState==1
    chargableNodes=find(pastTemps<critTempHigh);
    numChargable=numel(chargableNodes);
    chargePerNode=chargePerHour/(stepsPerHour*numChargable);
    Qout(chargableNodes)=chargePerNode;

elseif numHighNodes==numNodes && chargeState==1
    Qout=zeros(tankNodeCount,1);
    if qGenFail2==0
        qGenFail2=1;
        runNotes=[runNotes 'Heater not used during part of this hour; tank
temperature deemed too high. '];
    end

elseif chargeState==0
   Qout=zeros(tankNodeCount,1);

end
end %end of QGEN
```

APPENDIX D

INPUT AND OUTPUT FROM TABLER

```
CHATTANOOGA                    TN          35          2          85          12          75
Time 12:18:03 PM Date: 10/5/2012
                       (TABLER)Transient Analysis of Building Loads and Energy Requirements   Zone:1
Envelop:                  Roof   East Wall  South Wall  West Wall  North Wall  Floor    NE Wall  SE Wall  SW Wall  NW Wall
Area(ft^2):               2500   500        300         500        200         2500     0        0        0        0
Glass Area (ft^2):        0      0          200         0          200         0        0        0        0        0
U-Factor(Btu/hr.ft^2.F):  0.043  0.042      0.042       0.042      0.042       0.08
Type:                     4      9          9           9          9           6

No of Occupants (day) = 4  (night) = 0   (holidays) = 0     PV System:Off  TIAC:  Life(yrs):  PVeff:1  Cap Cost:600  Payback =

Equip Elec Load, kW (day) = 0.15  (night) = 0   (holidays) = 0    No hrs/yr:  Heat Eqpmt on: 0 Cool Eqpmt on:  0 Eqpmt Off:  0

Elec. Lights,W/ft^2 = 0.2    (night) = 0.1     (holidays) = 0.1     DCV System:Off      Zc =        Energy Recovery System: Off  Sen.Eff: 0.85  Lat.Eff: 0.85

Infiltration,fraction of ACH = 0.3 Ventilation,cfm/person = 15  Hot Water Cons,gal/person/day = 2  U- of Glass(Btu/hr.ft^2.F) = 0.8

S.C of Glass = 0.85   Elec Power Cost,cents/kWh = 12    Gas Fuel Cost, cents/therm = 95   Economiser: Off   Elec kW limit: 10   kW cost:0

Wint:Therm set = 72 F N. Setback = 72 F Throtl Range = 1F         :Sum: Thrm set = 76 F  N.Setback = 76 F Throtl Range = 1 F

Pk H/lat.Load,tons = -2.5/0.5 in Month = 1 on Day = 9 at hr = 8       :  Peak C/maxll, tons = 4.58/0.8 in month = 7 on Day = 17 at hr = 14

Approx Recommended Cap of Heating Eqpmt, tons =  0                    :    Approx Recommended Cap of Cooling Eqpmnt, tons =  0
                  On the Peak Heating Day  :                                                 On the Peak Cooling Day
  Time   A.Temp    S.Load     H.Added    S.Flux     S.Temp          Time   A.Temp   S.Load    Heat Extr   S.Temp   S.Flux
  hr     (deg F)   (Btu/hr)   (Btu/hr)   (Btuh/sft) (deg F)          hr    (deg F)  (Btu/hr)  (Btu/hr)    (deg F)  Btuh/sft
  1       12.92    -26465     -26447.1   0          72.1             1     73       14945     13537       75.2     0
  2       12.02    -27061     -26666.7   0          72.1             2     73.9     14299     12913       75.2     0
  3       10.94    -27736     -27075.8   0          72.1             3     72       13404     11937       75.3     0
  4       10.04    -28301     -27447.1   0          72.1             4     72       12792     11739       75.3     0
  5       10.04    -28767     -27779     0          72.1             5     72       11737     10944       75.3     0
  6       8.96     -29433     -28390.5   0          72.1             6     71.1     11477     10906       75.3     0
  7       8.06     -30038     -29011.6   0          72.1             7     73       11511     11158       75.3     0
  8       8.06     -28872     -27728.2   0          72.1             8     77       14951     15201       75.2     0
  9       10       -27124     -25669     0          72               9     80.1     24556     25920       75       0
  10      10.9     -17957     -15720     0          71.8             10    84       31935     34114       74.8     0
  11      15.1     -7955      -4926      0          71.6             11    87.1     41051     43975       74.6     0
  12      17.1     -1118      0          0          71.9             12    91       46553     49744       74.5     0
  13      19.9      3673      7591       0          71.3             13    93       50385     53752       74.4     0
  14      23        6329      9820       0          71.3             14    95       54973     58493       74.3     0
  15      25        6860      10131      0          71.3             15    95       53019     55924       74.3     0
  16      26.1      5087      7943       0          71.3             16    93.9     51120     53237       74.4     0
  17      27        831       2802       0          71.4             17    93       46982     48426       74.5     0
  18      25       -5859      -4943      0          71.6             18    93       41326     42016       74.6     0
  19      25       -16670     -16903     0          71.9             19    93       34777     34531       74.8     0
  20      21.9     -20807     -20981     0          71.9             20    86       26450     25455       75       0
  21      19.9     -22897     -23317     0          72               21    84       20048     18679       75.1     0
  22      19       -23442     -23600     0          72               22    82       18646     17274       75.1     0
  23      19       -23765     -23730     0          72               23    81       17231     15672       75.2     0
  24      18       -24499     -24359     0          72               24    80.1     16566     15312       75.2     0
                                                          Utility Cost Analysis
         Jan      Feb      March   April   May     June    July    Aug     Sept    Oct     Nov     Dec    t.year
 ac:     12       21       64      117     221     312     364     376     246     118     47      14     1912
 eqp:    4        4        4       4       4       4       4       4       3       4       4       3      46
 aux:    24       19       18      17      19      23      25      26      16      17      17      22     243
 lit:    29       26       29      28      29      28      29      29      27      29      28      28     339
 edmd:   0        0        0       0       0       0       0       0       0       0       0       0      0
 wat:    3        2        2       2       2       2       2       2       2       2       2       2      25
 telc:   72       71       116     168     275     369     423     437     294     170     97      69     2561
 erht:   0        0        0       0       0       0       0       0       0       0       0       0      0
 heat:   116      81       52      24      7       1       0       0       2       34      66      104    487
 cheat:  0        0        0       0       0       0       0       0       0       0       0       0      0
 cexet:  0        0        0       0       0       0       0       0       0       0       0       0      0
 cPVp:   0        0        0       0       0       0       0       0       0       0       0       0      0
 tuty:   188      152      168     192     282     370     423     437     296     204     163     173    3048

 Pk kW Dem:  1.7      kWh. Conp/yr: 21355    kWh Cost/yr,$: 2563    therms/yr: 995    GasCost,$:509    E.D.Cost,$/yr:0

tcfuel =  0    tmaintc =  0    tcded =  0    tdpoc =  0    taddc =  0    cedmd =  0    acostchp =  1967
```

APPENDIX E

SIMPLE STEAM POWER TOOLBOX

The Simple Steam Power Toolbox is a set of Simulink blocks that model the functions of the components of a steam power plant. It makes use of `XSteam`, Magnus Holmgren's wonderful encoding of the International Association for Properties of Water and Steam Industrial Formulation 1997 equations for MATLAB and many of the block symbols are from "Symbol Sourcebook" by Henry Dreyfuss. The Simple Steam Power Toolbox was developed to be used with the energy storage tank simulation, but it has many other potential applications.

The toolbox currently contains sixteen blocks. There are blocks to model turbines, pumps, condensers, feedwater heaters both open and closed, steam bleed lines, throttling valves, and heaters. The rest of the blocks allow the user to set or view properties of the steam at a given point in the system, like observing the enthalpy at the outlet of a component. The blocks are set up as masked subsystem blocks, which means it is very easy for a user to later look under the mask and either change some of the internal working of the block or to see how the block was constructed and make new blocks on their own.

Simulink models pass a signal from one block to the next, so part of the process of developing the SSPT was defining how a "steam signal" should look and behave. While the state of a steam flow might be adequately defined by four dimensions: pressure, temperature, mass flow rate, and then either enthalpy, entropy, or steam quality, it was decided that the signal should be redundant, so all six values are used. The steam signal is a vector defined as:

$$steam = < P, T, h, s, X, \dot{m} >$$

Where P is the pressure, T is the temperature, h is the specific enthalpy, s is the specific entropy, X is the steam quality (0 for liquid, 1 for vapor), and $\dot{m}$ is the mass flow rate. The unit conventions follow those of `XSteam`, so currently pressure is measured in bar, temperature in Celsius, specific enthalpy in kJ/kg, specific entropy in kJ/kg K, and mass flow rate in kg/s. The

SSPT is currently set up in metric units; if one needed to use it in US units, one would merely need to go under the mask in each block and replace each call to `XSteam` with a call to `XSteamUS`.

After the completion of this project, it is my ambition to finish documenting the Simple Steam Power Toolbox and to make it available for download through MATLAB's File Exchange website so that anyone can download it and use it.

An early test of the Simple Steam Power Toolbox is displayed in Figure E.1. The expected value of the thermal efficiency of the system, determined by solving the problem by hand, was 40.3%, and the SSPT displays a thermal efficiency of 40.45%. This difference is negligible and most likely due to small differences between the steam tables used to solve the problem by hand and the equations built into `XSteam`.

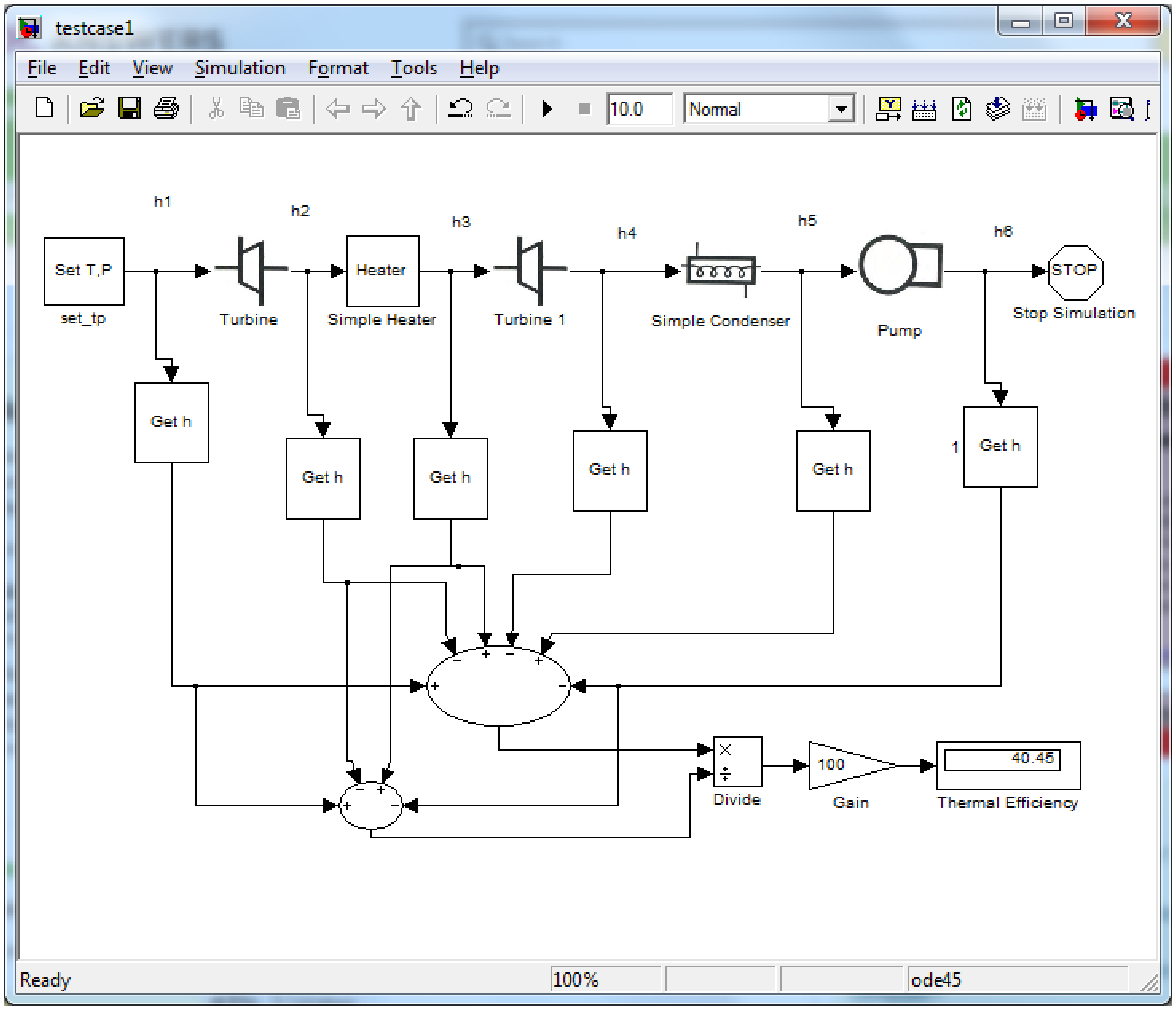


Figure E.1 An Early Test of the Simple Steam Power Toolbox

VITA

Robert Christopher Knowles Buckley was born in Aiken, South Carolina, to parents Lucy Knowles and Martin Buckley. He has a younger brother. After graduating from the Asheville School in Asheville, NC, Robert studied Mechanical Engineering at Vanderbilt University, graduating in the spring of 2008 with a Bachelors of Engineering in Mechanical Engineering. Robert accepted a graduate research assistantship at the University of Tennessee, Chattanooga, and graduated with a Masters of Science degree in Mechanical Engineering with an energy focus in December of 2012. Robert currently resides in Chattanooga, TN.