

University of Kentucky

UKnowledge

Theses and Dissertations--Civil Engineering

Civil Engineering


2023

Implementation of Digital Twins for Small Water Systems

Aidan Gill

University of Kentucky, apgi227@uky.edu

Author ORCID Identifier:

 <https://orcid.org/0000-0002-4623-4502>

Digital Object Identifier: <https://doi.org/10.13023/etd.2023.489>

[Right click to open a feedback form in a new tab to let us know how this document benefits you.](#)

Recommended Citation

Gill, Aidan, "Implementation of Digital Twins for Small Water Systems" (2023). *Theses and Dissertations--Civil Engineering*. 140.

https://uknowledge.uky.edu/ce_etds/140

This Master's Thesis is brought to you for free and open access by the Civil Engineering at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Civil Engineering by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Aidan Gill, Student

Dr. Lindell Ormsbee, Major Professor

Dr. Mei Chen, Director of Graduate Studies

IMPLEMENTATION OF DIGITAL TWINS FOR SMALL WATER SYSTEMS

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By

Aidan Patrick Gill

Lexington, Kentucky

Director: Dr. Lindell E. Ormsbee, Professor of Civil Engineering

Lexington, Kentucky

2023

Copyright © Aidan Gill 2023
<https://orcid.org/0000-0002-4623-4502>

ABSTRACT OF THESIS

IMPLEMENTATION OF DIGITAL TWINS FOR SMALL WATER SYSTEMS

The main objective of this thesis is to develop a working digital twin for a small water system in central Kentucky which will serve as a general format for other similar systems in the region wishing to implement digital twins for operator support. While the benefit of having a calibrated hydraulic and water quality model is widely understood, small distribution systems tend to not have the same financial and economic means to properly support these tools. Creation of a digital twin using this methodology provides a means for operators to predict pressure, flows, chlorine residuals, and total trihalomethane (TTHM) concentrations within their system with little to no cost and maintenance.

The application is developed using the MATLAB app development toolkit and is then linked with the EPANET hydraulic and water quality engine via the EPANET-MATLAB toolkit. The application provides simple user inputs such as initial tank levels, pump scheduling, demand scenarios, and mapping capabilities for results.

Reliability of the digital twin output is rooted in the extended period simulation (EPS) calibration steps which ensure the variation of demands both spatially and temporally accurately reflect conditions seen in the system. Both the Box-Complex (multi pressure zone systems) and the bisection method (two zone systems) were used in the processing of tank telemetry and meter data to create representative demand factors.

The creation of a useful digital twin is highly reliant on both the programming capability of the developer and familiarity with the many nuances of hydraulic and water quality calibration which are necessary foundations upon which accurate predictions of key parameters are accomplished. While outputs given in the MATLAB interface are simple, accurate, and robust against failure, there is much to be desired by way of interactive mapping. Python offers a broader range of available libraries capable of supporting mapping which will make inputting parameters and viewing results much simpler for operators. Additionally, the tools provided in this digital twin use historical data for hydraulic calibration (demand factors) and testing which are useful based solely on operator understanding of which demand scenarios in the past will most accurately reflect what they will see in the present. Extension of these historical patterns into forecasted demands using machine learning or time series analysis will greatly improve the usefulness of the model and overall operator experience.

KEYWORDS: [Water Distribution System, Network Analysis, Digital Twins, Hydraulic Calibration, Demand Prediction, EPANET-MATLAB toolkit]

Aidan Patrick Gill

12/08/2023

Date

IMPLEMENTATION OF DIGITAL TWINS FOR SMALL WATER SYSTEMS

By
Aidan Patrick Gill

Dr. Lindell E. Ormsbee

Director of Thesis

Dr. Mei Chen

Director of Graduate Studies

12/08/2023

Date

Dedicated to my Lord and Savior Jesus Christ, through whom all my worldly accomplishments find their sole responsibility in.

A.M.D.G.

ACKNOWLEDGMENTS

I would like to first acknowledge my advisor, Dr. Ormsbee. When I met Dr. O back in Fall of 2021, I had no idea the profound impact he would have on my life. While studying under one of the finest professors and foremost leaders in the water industry has been the experience of a lifetime, I am deeply indebted to him as a mentor and role model. The life he lives as a Christian man and leader of young people has been truly inspiring. I am grateful to count myself among the students who will try their best to carry on a legacy that in many ways is greater than the sum of all its parts.

I would also like to acknowledge my rugby coaches Sam, Vince, and most especially Gary. I would never be at the University of Kentucky without him, and my efforts in every endeavor here at school have been a small payment on the debt of gratitude I owe. To the three of you, thank you for pushing me and for making my time at UKY so endearing.

To the Gluten Free Council, you know exactly who you are. You are my brothers for life and probably don't want to read any more than a few more lines of this soppiness. Stay friendly and clean.

Lastly, and most importantly, I would like to acknowledge my family. My mom, dad, and two brothers mean more to me than anything in this world. I know at the very least if I have nothing else, I have them. Their support through this has brought me further than they could ever imagine, and I love them all the more for it.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	iii
LIST OF FIGURES	vi
LIST OF TABLES	vii
CHAPTER 1. INTRODUCTION	1
1.1 Background	1
1.2 Research Motivation	2
1.3 Research Hypothesis	3
1.5 Utility of Interest	6
1.6 Organization of Thesis	7
CHAPTER 2. LITERATURE REVIEW	10
2.1 Hydraulic Modeling	10
2.1.1 Darcy-Weisbach Equation	12
2.1.2 Hazen-Williams Equation	14
2.1.3 Network Algorithms	15
2.1.4 Hydraulic Calibration	24
Pipe Roughness Calibration	25
Demand Factor Calibration	26
2.2 Digital Twins	28
CHAPTER 3. DIGITAL TWIN DEVELOPMENT	31
3.1 Lebanon Water Works	31
3.1.1 Understanding System Needs and Relevant Data	33
3.2 Modeling Methodology and Tools Used	34
CHAPTER 4. HYDRAULIC CALIBRATION OF DIGITAL TWIN (LEBANON)	35
4.1 Demand Management Areas (Pressure Zones)	36
4.1.1 Bisection Algorithm (Two Zone System)	39
4.1.2 Box-Complex Algorithm (Multi Zonal System)	44
4.2 Hydraulic Calibration of LWW Digital Twin	49
CHAPTER 5. DIGITAL TWIN APPLICATION	59

5.1	Creation of the Graphical User Interface (GUI)	59
CHAPTER 6. DISCUSSION OF RESULTS AND CONCLUSION		75
6.1	Interface Selection and the Underlying Hydraulic Model	75
6.2	Data in the Hydraulic Calibration Process.....	78
6.3	Algorithms and the Hydraulic Calibration Process	79
6.4	General Conclusions	80
7. RECOMMENDATIONS.....		83
7.1	Engineering Significance	83
7.2	Limitations of Approach.....	84
7.3	Need for Future Research	84
7.4	Recommendations.....	85
APPENDICES		87
APPENDIX A. Demand Factors From Bisection Algorithm		87
APPENDIX B. Code For Hydraulic Simulation And Demand Factor Calibration... ..		117
APPENDIX C. User’s Manual for Digital Twin (Lebanon).....		174
APPENDIX D. Example Box-Complex Method Code For Four Zone System		188
REFERENCES		192
VITA.....		196

LIST OF FIGURES

Figure 2-1: Conservation of Mass and Energy Example	11
Figure 2-2: Moody Diagram (Ormsbee and Walski, 2016).....	13
Figure 2-3: Example Pipe Network	15
Figure 2-4: Generalized Newton-Raphson Method (Bhave and Gupta, 2013)	16
Figure 2-5: Example Setup for a C-Factor Test.....	26
Figure 3-1: Hydraulic Model of Lebanon Water Works	32
Figure 4-1: Pressure Zone Delineation within an Example Network.....	37
Figure 4-2: Example Network with Open Valve and Unintended Pressures	38
Figure 4-3: Example of Objective Function In Equation 4-1	41
Figure 4-4: Example of Initializing the Bisection Method	42
Figure 4-5: Example of Improved Solution Using the Bisection Method	43
Figure 4-6: General Structure of Bisection Method	43
Figure 4-7: Example Expansion Using the Box-Complex Method.....	48
Figure 4-8: Example Contraction Using the Box-Complex Method.....	48
Figure 4-9: General Algorithm for Box-Complex Method	49
Figure 4-10: LWW North and South Pressure Zone Delineation.....	50
Figure 4-11: Example of Noise in Tank Data (Walski et al., 2012).....	53
Figure 4-12: Flow Error Using Separate Time Intervals (Walski et al., 2012)	54
Figure 5-1: Home Screen for the Digital Twin.....	60
Figure 5-2: Time and Water Quality Screen.....	61
Figure 5-3: Time and Water Quality Screen.....	62
Figure 5-4: Time and Water Quality Screen.....	63
Figure 5-5: Demand Setting Section.....	64
Figure 5-6: Example Graph Given for Demand Pattern In Zone	64
Figure 5-7: Results Page from EPS	66
Figure 5-8: Example of Tank Level Output.....	67
Figure 5-9: Tabulated Results Given Selection of Tank Level	67
Figure 5-10: Example Junction Output.....	68
Figure 5-11: Example Pump Output.....	68
Figure 5-12: Example of Output Table Given Inputs for Each of the Graphs.....	69
Figure 5-13: Buttons for Graphing Results.....	70
Figure 5-14: Generated Interactive Map of LWW Showing Pipe Diameters.....	71
Figure 5-15: Tools for Map Visualization	72
Figure 5-16: Discovery Tool for Map Visualization	73
Figure 5-17: Map Results Detailing Pressure, Flows, and “Discovery” Results.....	74

LIST OF TABLES

Table 6-1: Summary of Digital Twin Creation Process	82
---	----

CHAPTER 1. INTRODUCTION

1.1 Background

Technology in the water industry has continuously shaped the way that operators interact with their systems and how they provide clean, safe drinking water to the communities that they support. As early as 700 B.C. in northern Iraq and Greece, sloping channels called qanats were being carved into the hillsides to provide irrigation for ancient farmers (Sedlak, 2014). A few hundred years later, Romans began mastering the art of creating aqueducts which could bring up to 1.13 million cubic meters of water per day to the empire at its peak (De Feo et al., 2013). Similar advancements in distribution practice were made by the Mayans in modern day southern Mexico with the introduction of the first pressurized piping systems around 250 AD (French and Duffy 2010). These advancements, as well as many others not mentioned in this paper, contributed to the modern drinking water systems we see today.

The first of these “modern” utilities appeared in the U.S. in the year 1652 when Boston employed its water works for fire-fighting and domestic use (Ormsbee, 2006). Since then, new challenges have arisen due to the complexity of water distribution systems as pressurized underground piping proliferated. Because of the inherent difficulties in understanding the nature of flows and pressures in piping networks, the 20th century saw a boom in research literature focused on the topic of “water distribution network analysis.” (Ormsbee, 2006).

Hardy Cross, a structural engineer at the University of Illinois at Urbana-Champaign was the first to create a numerical methodology for solving networks of pipes. Using the Hazen-Williams equation for losses and an iterated adjustment factor for solving the

continuity equation around loops, solutions could be found that were satisfactory in most cases but were however time consuming. Further, the methodology was limited to systems with only a few loops and without other system components (pumps, regulating valves, etc.). The dawn of the computer age and improved methods for solving the conservation of mass and energy equations allowed for methods that far surpassed the original Hardy-Cross method such as the simultaneous node, loop, pipe, and gradient methods (Ormsbee, 2006).

By the new millennium, several software packages such as KYPIPE (KYPIPE LLC, 2022) and EPANET (Rossman et al., 2020) became commercially available and gave operators a leg up in understanding the physical characteristics of their utilities. By calibrating for parameters such as pipe roughness, chlorine concentration, tank elevations, and pump operations; models gave a picture of the current state of the system (steady state analysis) as well as a confident understanding of what the system might look like in the near future (extended period simulation).

1.2 Research Motivation

There are many advantages afforded to water distribution operators who frequently consult water models for guiding their day-to-day decision making. Common uses of these models include predicting flows, pressure, velocity, and head loss but there are other advantages as well. Forecasting demands and hazardous chemicals like DBP's (disinfectant by products), simulating emergency scenarios, and planning for capital improvement projects are a few other applications of importance to engineers and operators alike (Huang, 2019).

While the advantages of consistent use and upkeep of a model are many, it is essential for operators to trust and understand the results of a hydraulic analysis (Huang, 2019). It follows then that calibration of a model is of the utmost importance for any analysis to be considered trustworthy. As Savic notes “regardless of the methodology used and parameters calibrated another general conclusion can be drawn, such as that a large amount of ‘good’ observation data is needed for estimating calibration parameters with sufficient confidence.” (Savic et al., 2009). To double down on this notion, water quality calibration depends not only on good data but also on the accuracy of the hydraulic model as well (Savic et al., 2009). These issues pose a sizeable challenge to operators, especially in smaller distribution systems who are unlikely to have the resources necessary to maintain models like this.

Because it is incredibly difficult for small systems to maintain a high-quality working model of their system, optimizing the triple bottom line (social, environmental, and economic considerations) is nearly impossible. As technology continues to advance however, new horizons are being discovered that allow utilities to address the issues associated with current modeling practices.

1.3 Research Hypothesis

In recent years, many water utilities have begun to incorporate the use of “Digital Twins” into their daily operations. The concept of digital twins first developed by Michael Grieves (referred to at the time as the “mirrored spaces” model) is a method through which the physical characteristics of a system are closely “mirrored” through a digital representation of a physical asset (Grieves and Vickers, 2016). In other words, digital twins

are developed to use continuous or near continuous data streams that allow for models to automatically calibrate and represent the system they were built for. Digital twins are defined by the context in which they are used and how they are applied across numerous industries. James Cooper, Global Director of Water Optimization at Arcadis notes that digital twins can be “a software application, a way of working, or a process” and states that “twins can also vary in complexity and maturity” (Cooper, 2021). These levels of complexity are referred to as states and vary from digital twin ready (modeled system) to live data integration and analysis.

Larger utilities like Las Vegas (Cooper, 2021) and Houston (Tripathi et al., 2021) have been leveraging the digital twin concept for almost two decades now with incredible success. After having worked through the several states of Digital twin operation (from Digital twin ready to using live data feeds) the Las Vegas Valley Water District (LVVWD) saw significant improvements in controlling DBP formation, substantial savings in energy consumption, and even had better response to emergency shutdowns (Cooper et al, 2022). The question remains of how to integrate these benefits into a smaller system while still considering the limited resources and the triple bottom line. As a result, the basic hypothesis of this thesis is that many of the operational benefits afforded by digital twin technologies can be extended to smaller systems, although it is recognized that some basic amount of system data will be needed as well as some level of cooperation by the partner utility. Part of this research will seek to identify what minimum baseline of information is needed.

1.4 Goals and Objectives

The purpose of this paper is to explore the development of a digital twin for smaller systems with limited data. The research integrates several existing software packages such as KYPIPE, EPANET, and MATLAB in an environment that is easy to use and capable of performing many of the previously described functionality of a traditional digital twin. Ultimately, this work will serve as a guideline for implementing a low cost, low maintenance tool for accurately predicting tank levels, pressures, chlorine residuals, and disinfectant by-product (DBP) formation for small systems. Additionally, this work seeks to investigate possible pitfalls in creating accurate and reliable digital twin solutions for small utilities. These goals are met through the successful pursuit of the following objectives:

1. The first objective of this work is to review other relevant scientific literature as it relates to distribution system analysis and digital twins.
2. The next objective is to investigate potential methodologies for use in developing digital twins for real world application. This will include utility feedback, specific modelling tools, and the appropriate modelling template.
3. The third objective for this work is to research means of developing realistic demand scenarios for use in evaluating alternative operational policies for the modelled system. This process will involve assessing the viability and potential limitations associated with available telemetry data and how to translate this data into reliable historical water demand time series. It is anticipated that some type of automated methodology or software will be needed. In this case,

alternative algorithms will be explored and the best among these will be chosen and applied.

4. Following the development of reliable demand forecasts, a digital twin will be developed which will allow for an operator to select from among these forecasts a representative demand pattern. The system will then be tested using these patterns for the purpose of optimizing daily operations and planning for the selected utility.
5. The final objective of this work is to determine the next steps the utilities of interest can take to advance their digital twin efforts. In addition to this objective, the research team seeks to come up with a generalized step by step process where other similar utilities may take advantage of their available data and develop their own digital twins at minimal cost and maintenance to their systems.

1.5 Utility of Interest

Initially, this research began looking at the possibility of developing a digital twin model for the Whitesburg Water Utility in Letcher County, Kentucky. Unfortunately, another system had to be considered because of continuing problems gaining access to critical data about the system because of a severe regional flood that occurred in the area, which forced a change in utility priorities. Instead of having time to focus on partnering with UK to develop a digital twin for their water system, they operators were more focused on flood recovery activities and just keeping the water utility open in support of the rest of the community. Initial work prior to the flood also identified significant data inconsistencies with the provided network topology and historical telemetry data which

raised questions about the feasibility of developing a baseline water distribution model for the system. This led to the first general observation about the feasibility of developing a digital twin for a small water utility. There obviously needs to be a minimum level of data and access before such a process can be undertaken, and in the case of Whitesburg, this proved to be infeasible. Thus, digital twins may not be universally feasible for all small water systems.

As a result of problems with the Whitesburg system, the focus of the research changed to an alternative system, namely the Lebanon Water Works (LWW) system in central Kentucky. The Lebanon Water Works (LWW) has a serviceable population of less than 21,000 people. This “small” system represents a well-run, progressive utility which has sufficient quantities of reliable data as well as having the fiscal and operational capabilities of integrating a digital twin model. The nature of this system is conducive to the testing and development of a framework through which digital twins may be applied to other small utilities, particularly utilities in the eastern Kentucky region.

1.6 Organization of Thesis

This thesis is divided into the following chapters:

Chapter 1. Introduction: this chapter details the importance of digital twin technology as a support and decision tool for operators.

Chapter 2. Literature Review: this chapter reviews the types of models available for modeling flows and pressures in water distribution systems, including both the Darcy-Weisbach and Hazen-Williams equations as well as broader network

algorithms for water distribution modeling. This chapter also reviews the steps necessary to calibrate such models using actual field data.

Chapter 3. Digital Twin Development: this chapter examines the needs within the Lebanon Water Works (LWW) system which drove the development of the digital twin model. Additionally, a general framework for the development of the digital twin is proposed.

Chapter 4. System Demand Forecast Scenarios. This chapter explains the development of potential algorithms for use in developing demand forecast scenarios for use in application in the proposed digital twin. Two separate algorithms were investigated, a Bisection method for application to two tank systems, and the Box Complex method for multiple tank applications.

Chapter 5. Digital Twin Application: this chapter details the application of the digital twin model within the Lebanon Water Works (LWW) using the general methodology built in chapter 3 and relevant equations described in chapter 4. Outputs from the hydraulic calibration process as well as app development are displayed in this chapter.

Chapter 6. Discussion of Results and Conclusion: this chapter discusses the summary of research with its conclusions.

Chapter 7. Recommendations for Future Research.

Appendix A, this section contains computed demand factors for Lebanon, Kentucky for the period between June 20th 2023 through July 3rd 2023

Appendix B, this section contains the code used to create the digital twin as well as relevant functions for hydraulic and water quality modelling.

Appendix C, this section contains a user manual for the digital twin application.

Appendix D, this section contains example code on the development of the Box-Complex method for a four-zone system

CHAPTER 2. LITERATURE REVIEW

Many decision-making processes for water distribution networks (WDN) are fundamentally rooted in having a working hydraulic model. Maintaining proper tank levels, optimizing pump scheduling, and ensuring proper fire protection are just a few of the applications within WDN's that rely on having an accurate, working model. Once a hydraulic model is properly implemented, the model may be extended to incorporate water quality features which assist in predicting chlorine residual and disinfection by-product (DBP) formation. Models can be extended even further by employing "Digital Twin" concepts as another tool in the operating and management process. Using digital twins allow operators and engineers to move from static to a more dynamic understanding of the current state of the WDN.

This section explores hydraulic models and digital twins in detail and provides a general understanding of the modelling processes used throughout this thesis.

2.1 Hydraulic Modeling

At its core, network analysis methods were developed in order to address the complexities of reliably delivering water in the growing number of municipal utilities that were springing up throughout the 20th century. Solutions to these networks, however, remains to this day a very non-trivial process. Every algorithm used to predict flows and pressures in WDN's are based on 1) the conservation of mass and 2) the conservation of energy equations (i.e., equations 2-1, and 2-2).

$$\sum_j q_{ij} - D_i = 0 \quad (2-1)$$

Where $\sum_j q_{ij}$ is the sum of flows at all junctions j connected to junction i (flow into a node is taken as positive) and D_i is the demand at junction i . The conservation of energy equation for each pipe segment is given by:

$$h_i - h_j = h_{Lij}(q_{ij}) \quad (\text{Rossman et al, 2020}) \quad (2-2)$$

Where h_i is the head at junction i , h_j is the head at junction j , and $h_{Lij}(q_{ij})$ is the head loss in the pipe that connects nodes i and j as a function of flow. Figure 2-1 shows a schematic for the relevant parameters as they appear for a several connected pipes.

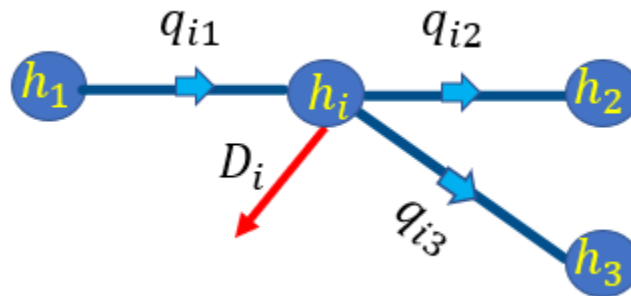


Figure 2-1: Conservation of Mass and Energy Example

While the conservation of mass equation is linear in terms of q and thus can be solved explicitly, much of the complexity of network analysis is due to the nonlinear nature of the energy equation (which can be seen in the equations of the following subsections) and the implicit calculation of its solution that is required. The approximate headloss discharge relationship itself may be evaluated using either the 1) The Darcy-Weisbach, or the 2) Hazen-Williams equations (2-3 and 2-5).

2.1.1 Darcy-Weisbach Equation

The Darcy-Weisbach equation, formulated as a result of the compiled works of Weisbach and later Prony and Darcy (Ormsbee and Walski, 2016) is:

$$h_L = \frac{fLV^2}{2gD} \quad (2-3)$$

Where h_L is the head loss term, D is the internal diameter of the pipe (ft. or m.), g is the gravitational constant ($32.17 \frac{ft}{sec^2}$ or $9.81 \frac{m}{sec^2}$), V is the fluid velocity (ft/sec), and f is the friction factor. Essential to the proper calculation of the head loss in a pipe using the Darcy-Weisbach equation is finding the friction factor which can be expressed as a function of the pipe diameter, roughness, and Reynolds number (which is a function of the pipe diameter, velocity, and fluid viscosity).

As originally formulated, the friction factor was obtained using a graph (typically referred to as the Moody Diagram – see Figure 2-2) which integrated previous experimental results of several researchers into a graph relating the friction factor to the Reynolds number and the ratio of the physical pipe roughness and the pipe diameter. Because of the lack of a closed form equation to represent the relationships displayed by the graph, computational applications of the Darcy Weisbach equation were initially limited, although several authors developed graphical solutions for small networks, (Ormsbee and Walski, 2016).

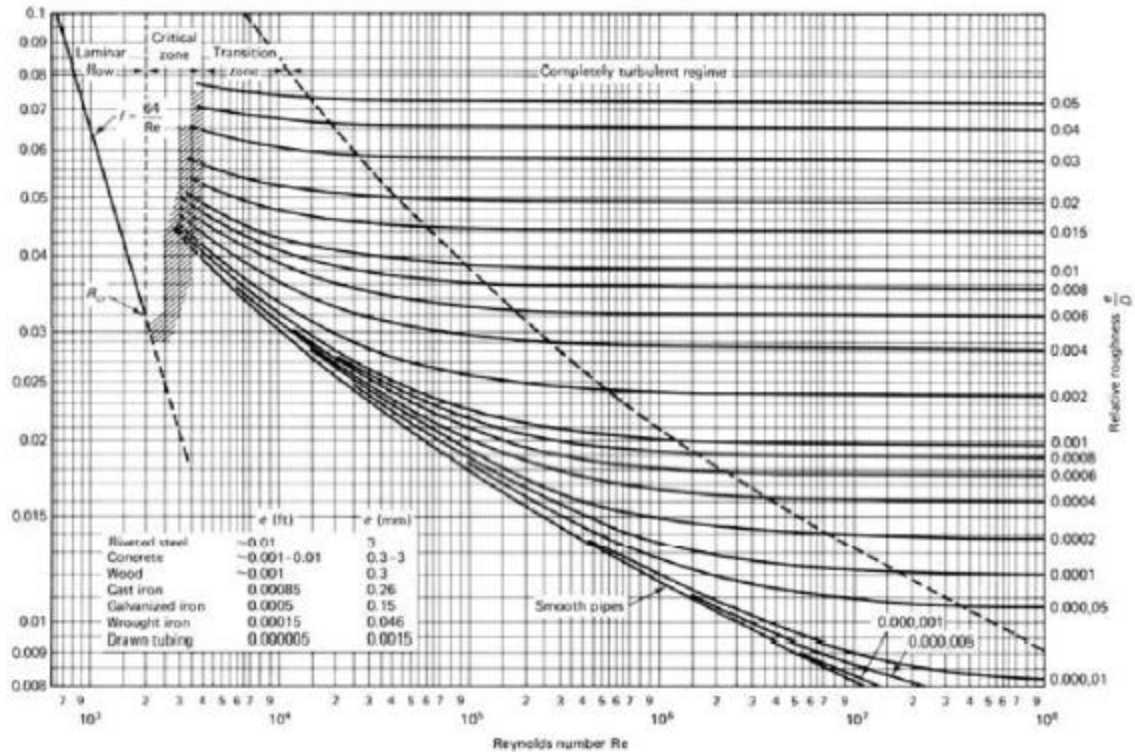


Figure 2-2: Moody Diagram (Ormsbee and Walski, 2016)

Eventually, the Colebrook-White equation was developed as an approximation to the friction factor in the Moody diagram but still required an iterative process to solve. Swamee and Jain resolved this issue by approximating the Colebrook-White equation which provided a tool for explicitly evaluating f , resulting in computerized solutions of head loss (2-4).

$$f = \frac{0.25}{\left[\log\left(\frac{\epsilon}{3.7D} + \frac{5.74}{Re^{0.9}} \right) \right]^2} \quad (\text{Ormsbee and Walski, 2016}) \quad (2-4)$$

Where ϵ is the pipe roughness (ft/ft or m/m), D is the pipe diameter (ft or m), and Re is the Reynolds number (dimensionless).

2.1.2 Hazen-Williams Equation

The Hazen-Williams equation is the less theoretically correct alternative to the Darcy-Weisbach equation and is given by:

$$h_L = \frac{4.72 LQ^{1.852}}{C^{1.852}D^{4.87}} \quad (2-5)$$

Where L is the length of pipe, Q is the flow rate ($\frac{ft^3}{sec}$), C is the Hazen-Williams C factor, and D is the internal diameter of the pipe (ft).

Where the Darcy-Weisbach equation applies to most flow regimes, roughness's, and fluids, the Hazen-Williams equation is only applicable to water under specific conditions. These conditions however are rarely violated under normal conditions in distribution systems and do not make the Hazen-Williams ineffective in the analysis of WDN's (Ormsbee and Walski, 2016). While computer models are quite capable of handling the explicit formulation of the Darcy-Weisbach equation, it is still much more common for modelers and engineers to employ the use of the Hazen-Williams equation due to its extensive use in the water industry and the common use of the Hazen-Williams C factor to characterize pipe roughness. The use of the Hazen-Williams equation is further supported by its ability to provide estimates of flowrate and diameter directly. In this research we will use the Hazen-Williams equation due to its familiarity with the system operators.

2.1.3 Network Algorithms

The network algorithms used in solving large systems of hydraulic equations come in many distinct forms and have been continuously developed and improved over the last 70 years. Each method seeks a robust formulation of the conservation of mass and conservation of energy equations that leads to the most efficient computerized solutions to modelled pipe networks. This section examines two of the more popular methods, specifically, the Newton-Raphson Method (NRM) for Pipes (NR-P), and the Global Gradient Algorithm (GGA) which are the main engines used in KYPIPE and EPANET respectively. For the sake of clarity, examples of these algorithms will exclude pumps, check valves, pressure reducing valve's (PRV), and other similar components. The application of each method to a typical water distribution network is illustrated using the example system shown in Figure 2-3.

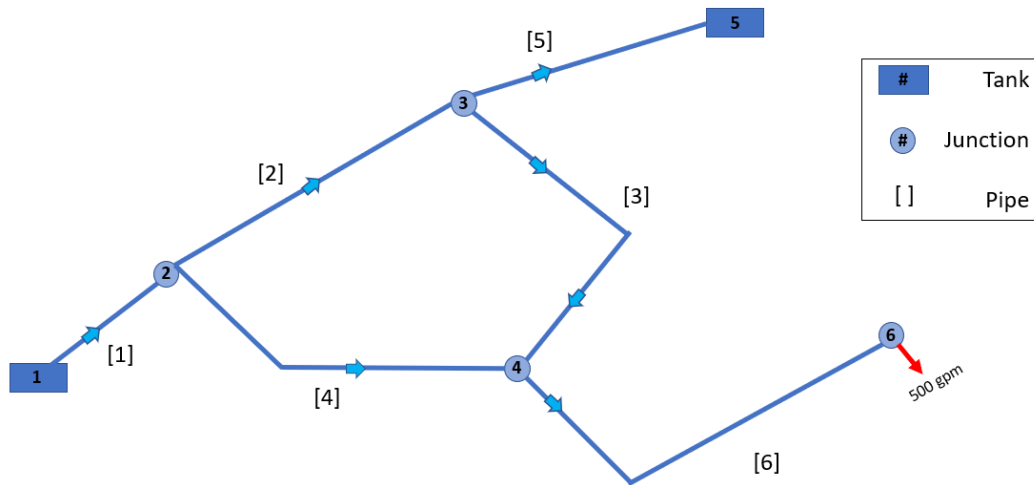


Figure 2-3: Example Pipe Network

Newton-Raphson Method for Pipes

In order to solve the nonlinear conservation of energy equations, they must first be approximated using a truncated Taylor series which can then be solved iteratively using the Newton-Raphson method. Technically, the Newton-Raphson method is a multi-dimensional version of the classical Newton's method for determining the root of a single nonlinear equation. Newton's method is illustrated in Figure 2-4.

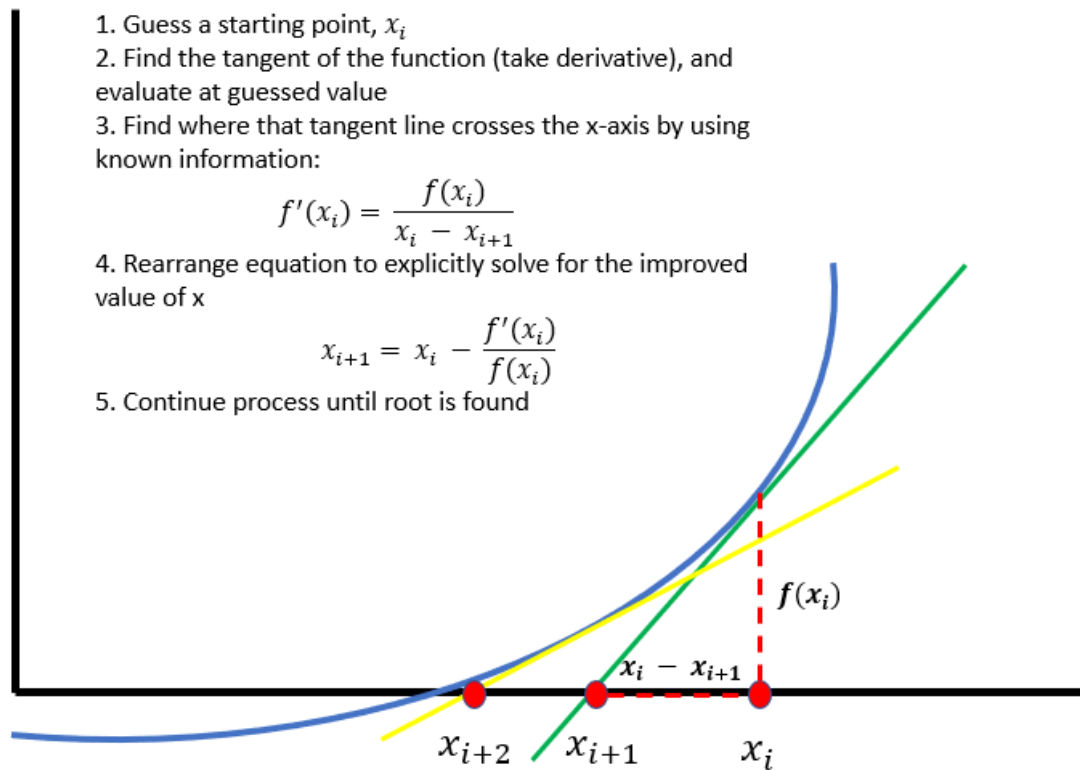


Figure 2-4: Generalized Newton-Raphson Method (Bhave and Gupta, 2013)

The Newton-Raphson method was first used to solve the conservation of mass and energy equations (expressed in terms of nodal heads) by Martin and Peters in 1963 and has subsequently been used in many other solution algorithms since. This section looks specifically at the NRM for pipes which constitutes the engine used in KYPIPE.

The pipe flows algorithm here is uniquely distinct from the previous NRM for nodes (Martin and Peters, 1963) and NRM for loops (Epp and Fowler, 1970) in that it is solving explicitly for the updated flow value as opposed to the change in nodal heads or flows associated with a loop or flow path. It is noted in (Wood and Rayes, 1981) that the total number of equations needed to solve the NRM for pipes is:

$$j + l + f - 1 = \# \text{ of pipes} = \# \text{ of equations to solve} \quad (2-6)$$

Where j = the number of junction nodes, l = the number of distinct loops, f = the number of fixed grade nodes (i.e., reservoirs or tanks where the hydraulic grade is known or specified), and p = the total number of pipes.

Applying this identity to the example system in Figure 2-3 reveals a total of six pipes, or six unknown pipe flows. As a result, six equations will be required. This will involve j (or 4) conservation of mass equations and $l+f-1$ (or 2) conservation of energy equations. For the example system, the four conservation of mass equations can be expressed as:

$$Q_1 - Q_2 - Q_4 = 0 \quad (2-7)$$

$$Q_2 - Q_5 - Q_3 = 0 \quad (2-8)$$

$$Q_6 - Q_3 - Q_4 = 0 \quad (2-9)$$

$$Q_6 - D_6 = 0 \quad (2-10)$$

Where Q_i is the unknown flow in pipe i and d_6 is the demand at node 6 (i.e., 500 gpm). For the example system, one conservation of energy equation can be written for the only pipe loop (i.e., involving pipes 2, 3, and 4) and another energy equation can be written

connecting the two fixed grade nodes (i.e., involving 1, 2, and 5), Mathematically, these two equations can be expressed as:

$$FGN_5 - FGN_1 = -h_{L1} - h_{L2} - h_{L5} \quad (2-11)$$

$$h_{L4} - h_{L3} - h_{L2} = 0 \quad (2-12)$$

Where FGN_5 and FGN_1 represent the water levels in each fixed grade node, and h_{Li} represents the head loss in pipe i .

Since equations 2-11 and 2-12 are nonlinear in terms of Q (i.e., equation 2-5), they must first be linearized before they can be solved. This can be accomplished using Newton's method, where:

$$Q_{i+1} = Q_i - \frac{H(Q_i)}{H'(Q_i)} \quad (2-13)$$

By approximating each $H(Q_i)$ in equations (2-11) and (2-12) using equation (2-13), each energy equation can be expressed as:

$$[H'(Q_i)\{Q_{i+1}\} = \{-H(Q_i) + H'(Q_i)(Q_i)\} + \Delta E] \quad (2-14)$$

Where H , H' , and Q , are all vectors and ΔE is a scalar (in this case the difference in elevation between the two tanks). Now that the energy equations have been linearized, they may be combined with the conservation of mass equations to yield six equations in terms of six unknowns: ($Q_{1+1} \dots Q_{6+1}$) as shown in equations 2-15a and 2-15b (broken into two equations for visibility).

$$\begin{bmatrix} 1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & G_L(Q_2)_i & G_L(Q_3)_i & G_L(Q_4)_i & 0 & 0 \\ G_L(Q_1)_i & G_L(Q_2)_i & 0 & 0 & G_L(Q_5)_i & 0 \end{bmatrix} \begin{bmatrix} Q_{1(i+1)} \\ Q_{2(i+1)} \\ Q_{3(i+1)} \\ Q_{4(i+1)} \\ Q_{5(i+1)} \\ Q_{6(i+1)} \end{bmatrix} = X \quad (2-15a)$$

$$X = \begin{bmatrix} 0 \\ 0 \\ 0 \\ q_6 \\ -H_{Loop}(Q_i) + G_{Loop}(Q_i)Q_i \\ -H_{path}(Q_i) + G_{path}(Q_i)Q_i + \Delta E \end{bmatrix} \quad (2-15b)$$

Where $G_L(Q_i)_i$ is the gradient of the headloss term for a specific pipe (i.e., $G_L(Q_i) = H'_L(Q_i)$), and where the matrix coefficients are all scalar quantities while the last two terms in the right-hand side vector are vector quantities e.g., $H_{Loop}(Q_i) = H_{L4}(Q_4) - H_{L3}(Q_3) - H_{L2}(Q_2)$. The algorithm is initiated with initial guesses for each Q_i (which is typically done assuming an initial velocity of 5 fps in each pipe). These Q's are then used to solve for the various G_L and H_L coefficients which are then loaded into the matrix and the right-hand side vector. Once populated, the system of equations is then solved for each Q_{i+1} . Once determined, these are used to update each Q_i , where $Q_i = Q_{i+1}$, and the process is then repeated until the Q's all converge to a stable solution.

Global Gradient Algorithm

The gradient method originally developed by Todini and Pilati (1987) solves the NRM directly for flows and heads simultaneously within the EPANET software. For networks with known pipe resistances, in this case they are assumed to be known, we may formulate the energy equations as follows (Bhave and Gupta, 2013):

$$(H_i + \Delta H_i) - (H_j + \Delta H_j) = R_x Q_x^n + nR_x |Q_x|^{n-1} \Delta Q_x \quad (2-16)$$

Where H_i is the known or assumed head at node i, H_j is the known or assumed head at node j, ΔH is the change in head for each respective node, R_x is the resistance constant in the pipe x (specific to which loss equation and units being used) which connects nodes i and j, Q_x is flow in a pipe x, and ΔQ_x is the change in flow in pipe x between iterations.

The Taylor series expansion and subsequent derivation of the NRM results in the right-hand side of equation 2-16. The left-hand side of the equation represents the updated heads at nodes i and j given the solution to the NRM on the RHS. By simplifying, $(H_i + \Delta H_i)$ and $(H_j + \Delta H_j)$ in equation 2-16 may be replaced by $H_{(i+1)}$ and $H_{(j+1)}$. By moving $nR_x|Q_x|^{n-1}\Delta Q_x$ to the LHS and subtracting $nR_xQ_x^n$ from both sides, the equation now becomes:

$$H_{(i+1)} - H_{(j+1)} - (nR_x|Q_x|^{n-1})Q_{x+1} = (1 - n)R_xQ_x^n \quad (2-17)$$

Where Q_{x+1} is the updated flow in the pipe. What this allows for is the unknowns (updated heads and flows) to be expressed on the LHS and the knowns to be expressed on the RHS of the equation. In the case where a FGN (known head) is a starting or terminal node, move $H_{(i+1)}$ or $H_{(j+1)}$ over to the RHS respectively. The total number of equations needed to solve the system is given as:

$$J + P = \# \text{ of equations} \quad (2-18)$$

Where j is the number of junctions and p is the number of pipes (see equation 2-6 for help identifying the number of pipes). Using figure 2-3, there are 4 junctions and 6 pipes making the system of equations to solve equal to 10. The continuity equations remain the same as in the NRM for pipes whereas the energy equations reflect the GGA methodology. Please note that when formulating the following equations, $H_{(i+1)}$ represents

the source node, $H_{(j+1)}$ represents the terminal node. These terms are multiplied by 1 if flow proceeds from i to j and -1 if it is from j to i.

$$H_{j_2} + G_{p_1}(Q_1)Q_{1(i+1)} = H_{FGN1} + (n - 1)H_{L1}(Q_1) \quad (2-19)$$

$$-H_{j_2} + H_{j_3} + G_{p_2}(Q_2)Q_{2(i+1)} = (n - 1)H_{L2}(Q_2) \quad (2-20)$$

$$-H_{j_3} + H_{j_4} + G_{p_3}(Q_3)Q_{3(i+1)} = (n - 1)H_{L3}(Q_3) \quad (2-21)$$

$$-H_{j_2} + H_{j_4} + G_{p_4}(Q_4)Q_{4(i+1)} = (n - 1)H_{L4}(Q_4) \quad (2-22)$$

$$-H_{j_3} + G_{p_5}(Q_5)Q_{5(i+1)} = -H_{FGN2} + (n - 1)H_{L5}(Q_5) \quad (2-23)$$

$$-H_{j_4} + H_{j_6} + G_{p_6}(Q_6)Q_{6(i+1)} = (n - 1)H_{L6}(Q_6) \quad (2-24)$$

Where H_j represents the head at a specified junction, H_{FGN} is the head at a fixed grade node (separate labeling for j and FGN unnecessary and used only for clarity), $G_p(Q_p)$ is the gradient in the pipe with respect to the pipes flow ($nR_x|Q_x|^{n-1}$), $Q_{(i+1)}$ is the updated flow term, and $H_L(Q_p)$ is the headloss in the pipe with respect to the pipes flow. The following is the matrix representation of the above equations:

$$\begin{bmatrix}
G_{p1}(Q_1) & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & G_{p2}(Q_2) & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & 0 & G_{p3}(Q_3) & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & G_{p4}(Q_4) & 0 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & G_{p5}(Q_5) & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & G_{p6}(Q_6) & 0 & 0 & -1 & 1 \\
1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
Q_{1(i+1)} \\
Q_{2(i+1)} \\
Q_{3(i+1)} \\
Q_{4(i+1)} \\
Q_{5(i+1)} \\
Q_{6(i+1)} \\
H_{j2} \\
H_{j3} \\
H_{j4} \\
H_{j6}
\end{bmatrix}
= X \tag{2-25a}$$

22

$$X = \begin{bmatrix}
H_{FGN1} + (n-1)H_{L1}(Q_1) \\
(n-1)H_{L2}(Q_2) \\
(n-1)H_{L3}(Q_3) \\
(n-1)H_{L4}(Q_4) \\
-H_{FGN2} + (n-1)H_{L5}(Q_5) \\
(n-1)H_{L6}(Q_6) \\
0 \\
0 \\
0 \\
q_6
\end{bmatrix} \tag{2-25b}$$

Important to note from this section is the understanding that this methodology allows for the explicit solution of heads and flows at each iteration which vastly increases its speed relative to other algorithms. This thesis will continue to develop the framework of hydraulic modeling assuming that the equations being solved follow the EPANET GGA because of its use in the digital twin model (more on digital twins in section 2.3).

Extended Period Simulation

The process through which hydraulic models are expanded from static (steady state) simulations to their more dynamic, temporally varying counterparts is called extended period simulation (EPS). EPS can be captured through modeling the change in all the tank volumes over time which may be seen in equation 2-26 (Rossman et. al., 2020).

$$\frac{dV_s}{dt} = Q_{s,net} \quad (2-26)$$

Where $\frac{dV_s}{dt}$ is the change in tank volume over time and $Q_{s,net}$ is the net flow into or out of the tank. Continuing with the methodology developed by Rossman in the EPANET 2.2 manual (Rossman et. al., 2020), a second equation is then needed that relates the head at the surface level of the tank to the volume of the tank (equation 2-27).

$$H_s = E_s + Y(V_s) \quad (2-27)$$

Where H_s is the tank's elevation head, E_s is the tank's bottom elevation and $Y(V_s)$ is the relative tank water level as a function of volume. Solving the network using the GGA algorithm results in the flows into or out of the tanks ($Q_{s,net}$) which can then be used to solve 2-28 and 2-29. By solving equations 2-28 and 2-29 the time step may be accurately

advanced and hydraulic conditions known within the system.

$$V_s(t + \Delta t) = V_s(t) + Q_{s,net}(t)\Delta t \quad (2-28)$$

$$H_s(t + \Delta t) = E_s + Y(V_s(t + \Delta t)) \quad (2-29)$$

This is the generalized process behind extended period simulations. It should be noted that any changes such as pumps turning off and on, tanks completely empty or full, or other similar scenarios will also trigger this process.

2.1.4 Hydraulic Calibration

Another complexity of hydraulic modeling involves the process of ensuring that the model outputs accurately reflect conditions seen in the field such as pressures, flows, and tank levels. This process is referred to as hydraulic calibration and it is typically broken down into 1) pipe roughness calibration using (steady state) and 2) demand calibration using (extended period simulation). These two parameters typically have the highest degree of uncertainty and because of this, they are the most in need of high-quality field observations and data. Calibration for pipe roughness and demands are carried out by adjusting the Hazen-Williams C factor and demand factors respectively to obtain a useful model (Ormsbee and Lingireddy, 1997).

Model calibration is accomplished using seven basic steps outlined by Ormsbee and Lingireddy (1997), 1) identify the intended use of the model, 2) determine initial estimates of model parameters, 3) collect calibration data, 4) evaluate the model results, 5) perform the macro-level calibration, 6) perform the sensitivity analysis, and 7) perform the micro-level calibration. For the purpose of understanding the calibration process, only step

7 (micro-level calibration) will be highlighted in the following subsections which focuses on pipe roughness and demand calibration.

Pipe Roughness Calibration

It is an industry standard to provide pipe roughness factors for pipes supplied by manufacturers and can be found online in a plethora of handbooks. The Hazen-Williams C-factors are typically populated into uncalibrated hydraulic models using such typical values and some programs like KYPIPE will automatically assign default C-factors depending on what type of material is specified. However, such values might not be totally accurate since C-factors can decrease over time as a function of water age and water quality. For example, a 60-year-old cast iron transmission main may have a significantly reduced C-factor due to the formation of tuberculation.

Because C-factors may also be used in a model to compensate for several other factors such as fitting losses and system skeletonization, field testing is critical even for relatively newer pipes (Ormsbee and Lingireddy, 1997). The procedure for conducting these field tests are as follows (figure 2-5): 1) select a straight-line section of pipe containing a minimum of 3 fire hydrants, 2) isolate the pipe by closing the downstream valve, 3) attach pressure gauges to the first two hydrants (called residual hydrants), 4) measure the elevation differences between the first two hydrants and their respective pressures, and then flow and record the flow rate at the 3rd hydrant. Finally, calculate the head loss in the pipe using equation 2-30.

$$h_L = \frac{(P_2 - P_1)}{32.2} + (Z_2 - Z_1) \quad (2-30)$$

Where h_L is the headloss in the pipe (ft), 32.2 is the gravitational constant ($\frac{ft}{sec^2}$), P_i is the pressure measured at the hydrants (convert from psi to psf), and Z_i is the elevation of the hydrants (ft). Once this is completed, the Hazen-Williams C-factor can be determined by rearranging equation 2-5 and solving for C.

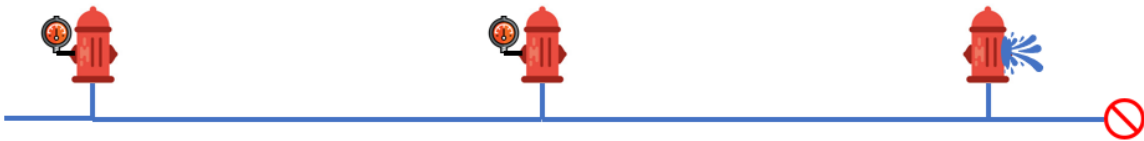


Figure 2-5: Example Setup for a C-Factor Test

In almost every case, it is economically infeasible to perform this test on all pipes in a WDN. Because of this, C-factor tests are usually taken at locations that are generally representative of the system at hand. If a neighborhood, for example, is known to have been constructed at a certain time, it is likely that all the pipes in that area are the same and will be subject to the same C-factor values. These tests should be conducted regularly so that hydraulic model performance can be maintained or improve over time.

Demand Factor Calibration

Calibration of demand factors is an inherently complicated process since they are subject to variation both spatially and temporally. Models are always populated with “base demands” which in most cases are chosen as the average demand seen at that point in the system over a selected period. Errors at this step (steady state) may be found given a peak flow scenario where the HGL is drastically affected by losses encountered by pipes, fittings, PRV’s, etc. (Walski, 2017). After calibrating pipe roughness, If the HGL is still

well above or below what has been measured in the field and that error is somewhat evenly distributed across a pressure zone, it can be safely assumed that the base demands need adjusting.

When considering temporal differences in data, it is important to note that differences between any two days will likely be significant (Walski, 2017). Using the region surrounding the University of Kentucky campus, for example, demands may be altered by home football games, seasonal breaks, fires, etc. Because of this, operators and engineers must exercise extreme caution when evaluating the intended use of their models and what data has been used to calibrate the “demand factors”. In most situations, variations in demand may be accounted for by scaling up and or down these factors, depending on a water treatment plant’s daily production. However, in the case of fires, this would not be enough due to the concentrated nature of such a demand (Walski, 2017).

In conclusion, there are numerous factors and methods used to calibrate hydraulic models which in many cases can overwhelm the operator or engineer attempting to use a hydraulic model to guide operational decisions. There are hundreds of papers dedicated to this subject alone, many of which are outside the scope of this paper. When describing their 7-step calibration procedure, Ormsbee and Lingireddy (1997) make note that “one of the most difficult steps in the process has been the final adjustment of pipe roughness values and nodal demands”. Because of this, later sections of this paper focus heavily upon the process of demand factor calibration due to its inherent difficulty and the challenges it poses to operators and engineers alike.

2.2 Digital Twins

Digital twins are a concept that first originated with a presentation given by Dr. Michael Grieves in a 2002 presentation at the University of Michigan centered around the creation of a “Product Lifecycle Management Center.” The name was not explicitly that of “digital twins”, but it had all of the relevant elements: “real space, virtual space, the link for data flow from real space to virtual space, the link for information flow from virtual space to real space and virtual subspaces” (Grieves and Vickers, 2016).

At its core, this concept is rooted in the successful integration of high-quality data transmitted in real time, to a working model that is capable of producing outputs that may be seen in the field. Consider that hydraulic models are calibrated and used based on historical sources of information whereas digital twins are based on current or near current data streams. The paradigm shift here is equivalent to having a picture of something as it was in the past (and assuming how it might look and function in the future), or having the real thing being effectively “mirrored” as it is right now.

The water industry in the past has been slow to adopt digital solutions such as this but several case studies in places such as Las Vegas have demonstrated the significant return on investment that many utilities are looking for. In addition, COVID 19 prompted an accelerated integration of these digital solutions which ultimately thrust the industry into this relatively newfound territory (Cooper, 2021).

Digital twins foundationally may seem simple enough to understand given the definition by Grieves, however its application in the water industry is consistently met with confusion due to lack of consensus on a uniform definition. Some considered digital twins as simply a hydraulic model, others feel the necessity of live data feeds, still others think

of it as a supervisory control and data acquisition system (SCADA) (Cooper, 2021). Questions also surrounded whether application of digital twins change with respect to utility size, cost, and overall purpose.

The need for consensus on the topic in the water industry prompted the formation of the “Digital Twins Committee” (DTC) in AWWA. What resulted from this is the formal definition of digital twins (within the water industry) as: “A digital, dynamic system of real-world entities and their behaviors using models with static and dynamic data that enable insights and interactions to drive actionable and optimized outcomes” (Saša, T., et al. 2022).

The definition, while targeted and well formulated, still leaves room for several different “levels” of digital twins which vary from utility to utility. The AWWA-DTC defines these as levels zero through three. Level zero, also referred to as “digital twin ready” are any systems that have gone through the process of collecting historical data of their systems or even a hydraulic model that can be doing much more than they are currently doing (Cooper et al., 2022). Many utilities, especially those with fewer resources find themselves at this stage.

Level one digital twins are called “informational twins” and use the virtual representations (typically hydraulic models) built during level zero and incorporate historical data to improve the performance of the model (Cooper et al., 2022). For example, historical information may come from sources such as SCADA or GIS that will allow for more accurate prediction of day-to-day operations and planning.

Level two digital twins are defined as “Operational Twins” and are different from level one twins in that they incorporate live data streams. This version of a digital twin most closely fits the definition originally proposed by Grieves and Vickers (2016) and therefore is the most traditional when considering other industry standards. If created properly, these twins give significant advantages to their respective utilities. For example, Houston Water Planning, which serves over 5 million residents, has created a functional digital twin and has met great success in the process. By integrating SCADA and GIS information in real time, they have been able to identify valves which had been assumed open but were closed due to line breaks that occurred several years ago. This correction in the model vastly improved hydraulic modeling for the system and overall management decision making (Tripathi, et al., 2021).

The most advanced form of a digital twin is defined as “Connective Twins” and are the end goal for all utilities operating under this framework. Connective twins in simple terms are a digital twin that communicates with other digital twins. For example, there may be a digital twin that has real time information on electrical rates, usage, and forecast demands that integrates with a hydraulic digital twin performing similar functions. By having the two twins communicate, electrical usage at the water utility may be optimized to reduce cost and overall energy consumption.

While these definitions are useful, it is important to understand that a utility typically will find itself somewhere in between these levels and digital twin transformation may be viewed on a spectrum of readiness, implementation, and use.

CHAPTER 3. DIGITAL TWIN DEVELOPMENT

In section 2.2, it was noted that digital twin applications may look different between any two utilities depending on their specific system needs and current levels of data management and integration. Using Lebanon Water Works (LWW) as a case study representing small water systems; hydraulic and water quality data were collected, and a digital twin was built for this WDN. This section details the processes undertaken to understand the specific needs of this utility, data collection methods, and challenges unique to this system that may cause the success or failure of producing a useful digital twin for distribution operators.

3.1 Lebanon Water Works

Lebanon Water Works (LWW), located in Marion County, Kentucky, provides drinking water to the city of Lebanon with a directly serviceable population of 6,412 and an indirectly serviced population of 14,006 (WRIS, 2023). Water is mostly drawn from the Rolling Fork River and occasionally from Marion County Lake as a reserve. The system currently maintains 3 tanks. In the northern portion of the system, there is an elevated tank that has a capacity of 250,000 gallons. The other two tanks, which are identical in geometry and are located adjacent to one another, are in the central part of the system and have a combined storage of 188,000 gallons. There is a booster pump located near the western side of the system that pumps to the Springfield Road tank in the north. The water treatment plant in the very southern end of the system delivers approximately 2.6 MGD and has a design capacity of 5.2 MGD. LWW sells most of its water (approximately 62% of last

year's annual volume) through 10 key points throughout the WDN. All of these attributes are detailed in figure 3-1.

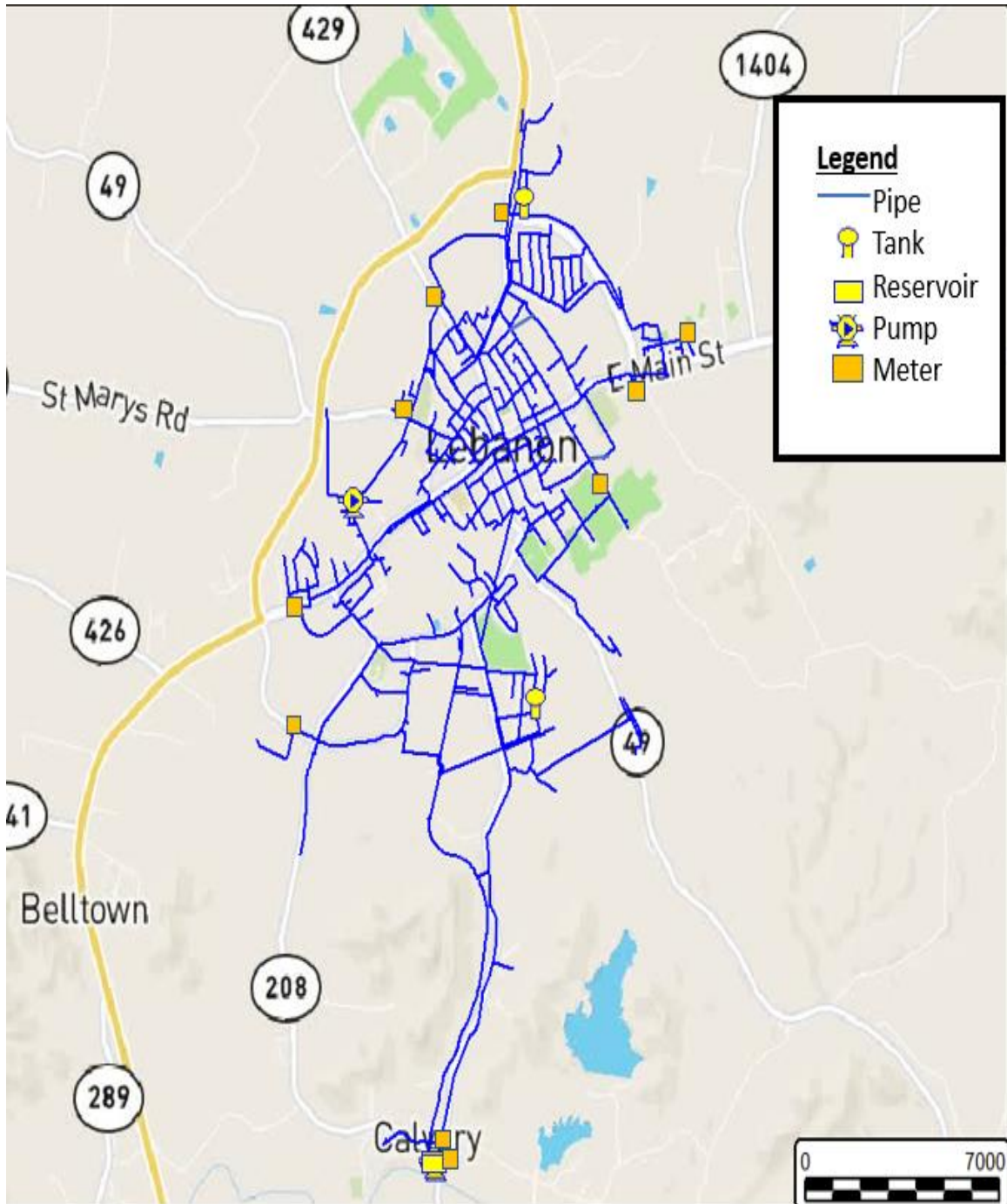


Figure 3-1: Hydraulic Model of Lebanon Water Works

3.1.1 Understanding System Needs and Relevant Data

The original model for this research was provided by Jim Thompson of Kentucky Engineering Group located in Versailles, Kentucky. The model elevations, pump curves, tank elevations, and overall system topology were validated through coordinated efforts between Mr. Thompson, LWW staff, and the author. Master meter data for all 10 selling points were provided by the utility for dates from 6/19/2023 through 7/20/2023 at a data resolution of 1 hour. Data for tank levels, pump intake and discharge pressure, water treatment plant (WTP) flow rate, and free chlorine residual were provided from 6/1/2023 through 7/24/2023 with a data resolution of 2 minutes. Master meter, tank, and pump information were all provided by LWW.

Given the quantity and quality of available data, discussions around building a digital twin for LWW were very different than those of communities in rural Eastern Kentucky. LWW identified the following as the key outcomes they were seeking to obtain through the development and use of a digital twin: 1) capability for evaluating the placement of a new tank and a new 16" transmission main, 2) prediction of pressures at key junctions, 3) prediction of water age in tanks, and 4) prediction of tank levels over a 24 hour period, 5) prediction of free chlorine residual at key junctions, and 6) prediction of DBP indicators such as HAA5. The utility also expressed a desire to be able to access such information through an easy-to-understand graphical user interface (GUI) that is accessible to distribution operators who are assumed to have little to no knowledge of hydraulic modeling.

3.2 Modeling Methodology and Tools Used

The main tool used in processing the hydraulic data for this application is EPANET. EPANET is an open source hydraulic and water quality engine developed by the United States Environmental Protection Agency (U.S. EPA) which is capable of producing results for both steady state and extended period simulations at incredible speeds (section 2.1.3). By taking the original model developed by Jim Thompson and the Kentucky Engineering Group in KYPIPE, the data was first exported from KYPIPE as a “.inp” file and stored for processing.

The GUI development platform selected for this application is MATLAB which offers extremely simple dashboard development and user elements. Through the EPANET-MATLAB toolkit (Eliades et al., 2016), the MATLAB GUI elements can directly interact with the EPANET engine and display useful results to the operator.

For this thesis, elements available for user input were limited to, 1) operational information associated with the WTP and booster pump operations, 2) initial tank levels, 3) free chlorine concentration at the WTP, 4) bulk and wall decay coefficients (preset), 5) basic demand patterns, and 6) total time and time step of simulation.

Outputs in this application do not meet all the expected outcomes put forth by LWW due to the time limitations of this work. Outputs developed as a part of this research include: 1) prediction of pressure at key junctions, 2) prediction of free chlorine at key junctions, 3) prediction of DBP formation at key junctions, 4) tank water age, and 5) tank levels over a 24-hour period. Other outputs include a simple graphical display which will give the user the ability to quickly and simply visualize all of these parameters which can then be used to drive actionable decisions.

CHAPTER 4. HYDRAULIC CALIBRATION OF DIGITAL TWIN (LEBANON)

Hydraulic calibration is the foundation upon which useful information may be extracted from water distribution models. This process, highlighted in section 2.1.4, can prove to be extremely difficult depending on the quality and quantity of available data. LWW was selected to pilot this study due to the progressive nature of the utility and its ability to provide high quality system information that most nearly mimics the live data streams that are characteristic of digital twins.

Assumptions made relative to the hydraulic calibration process for the digital twin are as follows: 1) pipe roughness' given in original model are sufficiently calibrated, 2) pump curves are sufficiently calibrated, 3) nodal base demands are sufficiently calibrated, 4) nodal elevations are correct, 5) overall system topology is correct within reason (no outstanding errors), 6) sensor data at tanks and pumps are accurate, and 7) all information given is reflective of standard conditions within the system.

The purpose of this thesis is to present ways in which a digital twin can be developed efficiently as a decision support tool for operators. While other calibration steps are critical to the development of a useful model, the parameter that drives much of the error and variation of results for extended period simulations are the spatial and temporal distribution of demand factors.

Typically, modelers approach this step in the calibration process by ensuring that their modeled tank level changes in their respective models are tracking with observed telemetry data. While this provides a “snapshot” of how the utility might operate on a typical day, this information is susceptible to significant error in the actual hydraulics (tank

levels, demands, flow rates, etc.) seen in the system. Demand variation from one day to the next may be as high as 20% or more and raises the concern of calibrating a model from a single days' worth of information (Walski et al., 2012).

While the benefits of having a calibrated model are known, the cost for most small utilities is prohibitive due to the time it takes to calibrate and validate extended period simulations which is a highly iterative and time intensive process. In addition, most operators of these systems do not have engineering backgrounds and typically lack great understanding of how and when models are calibrated to meet their needs. In this study, the primary focus was on determining the correct temporal variations of nodal demands for actual observed days. Since individual customer demands were not readily available, several surrogate measures (i.e., pump discharges, master meters, and water tank levels) were used to calibrate the associated temporal demand distributions. The final calibration process was then performed by coupling the associated EPANET model with a traditional nonlinear optimization algorithm. To facilitate this process, the system was first broken down into two different demand management areas.

4.1 Demand Management Areas (Pressure Zones)

The calibration of demands for a water distribution system can be facilitated by either taking advantage existing zone separations (i.e., using pressure zones) or by creating artificial water demand zones through the installation and closing of isolations valves which are then connected by water meters. Pressure zones are delineated by closing a series of valves that isolate one region of a distribution system from another. By creating these zones, portions of a WDN that exist at higher elevations can benefit from increased

pressures from booster pump stations and storage tanks while protecting assets at lower elevations which would likely be damaged from these excessive pressures. To demonstrate the concept of pressure zones, a valve is closed in the example system in figure 4-1 that causes two independent zones to be created. When opening that same valve in figure 4-2, assets in the south are subject to excessive pressure from the booster pump whereas service in the northern portion of the system will likely be inadequate due to low pressures.

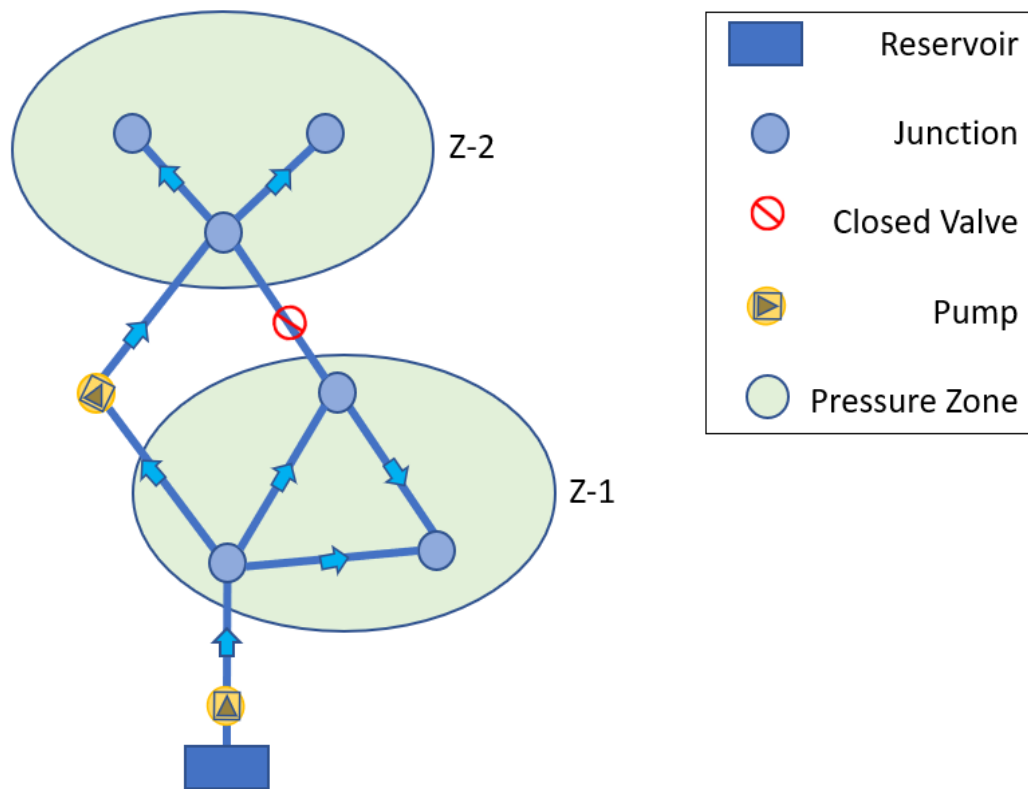


Figure 4-1: Pressure Zone Delineation within an Example Network

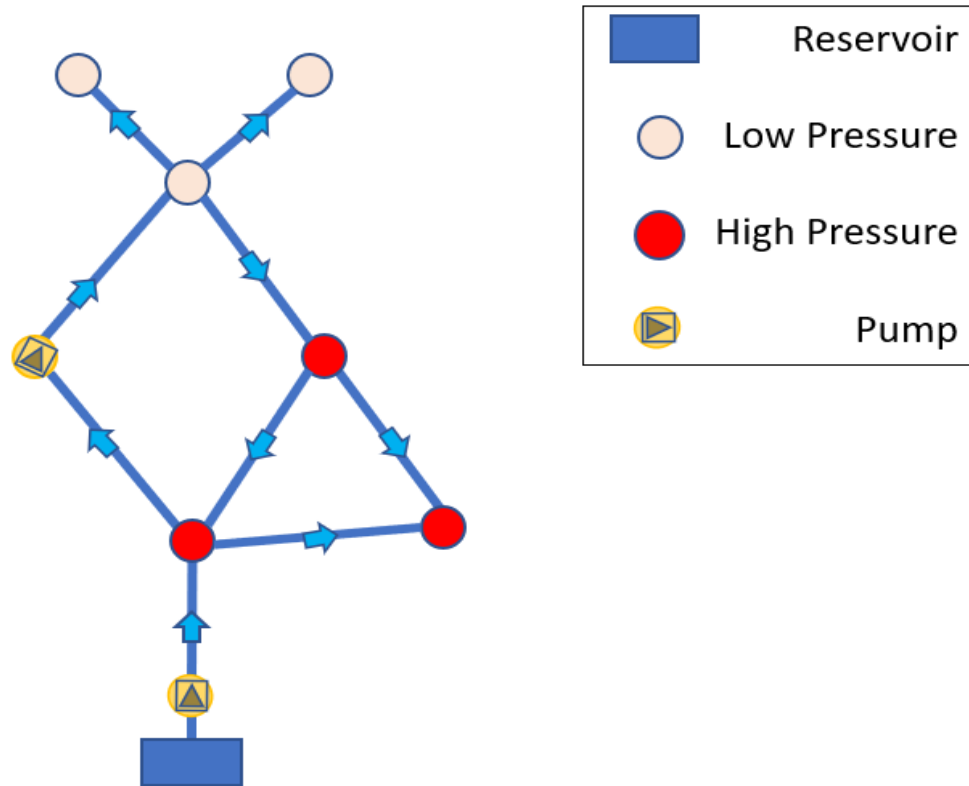


Figure 4-2: Example Network with Open Valve and Unintended Pressures

The ultimate number of pressure zones to be modeled can impact the best choice of an optimization algorithm. In general, a simple bisection method was found sufficient for two zone systems, while the Box-Complex Method (Box, 1965) was found sufficient for multi-zone systems with more than 2 pressure zones.

4.1.1 Bisection Algorithm (Two Zone System)

In some cases, especially with smaller systems, a WDN may be conveniently divided into two distinct pressure zones or demand management zones. To find the optimized demand factors for these zones, an objective function must first be defined. The objective function is simply the function whose value is minimized by the chosen algorithm. In the case of the Lebanon system, the following objective function was used:

$$f(DF_{z1,t}) = (T_{m,z1,t} - T_{r,z1,t}) \quad (4-1)$$

Where $f(DF_{z,t})$ is the function to be minimized where $DF_{z,t}$ represents the demand factor for zone z and time t , $T_{m,z1}$ is the modeled tank level in zone 1, $T_{r,z1}$ is the observed tank level in zone 1 and whose values are a function of the demand in zone 1. These values of the decision variables (i.e., the $DF_{z,t}$) are further constrained to not violate conservation of mass across the system for a given total demand associated with time t (i.e., TD_t).

$$Q_{plant,t_i} + Q_{tank,t_i} - Q_{sold,t_i} = TD_t \quad (4-2)$$

Where t is the time step, Q_{plant} is the flow from the plant, Q_{tank} is the flow from the tanks (where flow out of tanks is taken as positive), Q_{sold} is the flow out of the system at its boundary (water sold to other utilities, these are given from master meter data), and TD_t is the total demand. Total demand may be expanded to include the two zones within the system as they are typically modeled in hydraulic software:

$$TD = (DF_1 * (\Sigma BD_1)) + (DF_2 * (\Sigma BD_2)) \quad (4-3)$$

DF_1 is the demand factor for zone 1, ΣBD_1 is the sum of the nodal base demands in zone 1, DF_2 is the demand factor for zone 2, and ΣBD_2 is the sum of the base demands for zone 2. By combining equation 4-2 and 4-3, the system constraint becomes:

$$(DF_{1,t_i} * (\Sigma BD_1)) + (DF_{2,t_i} * (\Sigma BD_2)) = Q_{plant,t_i} + Q_{tank,t_i} - Q_{sold,t_i} \quad (4-4)$$

Because plant flow rate and the flow rate of water sold to neighboring utilities are known parameters (metered data) whose exact values may be input to the model as fixed boundary conditions, the objective function simply becomes a problem of minimizing the difference between the model tank levels and the real (known) tank levels given these constraints.

Since the analysis and optimization of demand factors is not necessarily a straightforward process (see section 2.1) this task would take an experienced engineer many iterations with a hydraulic model to achieve values that match field conditions. This process however is easily automated using the bisection method.

The bisection method is a simple, but very powerful algorithm that allows for the “bracketing” of a solution to non-linear objective functions. Consider figure 4-3, a non-linear representation of the difference between real and model tank levels within a single zone as a function of that zone’s respective demand factors.

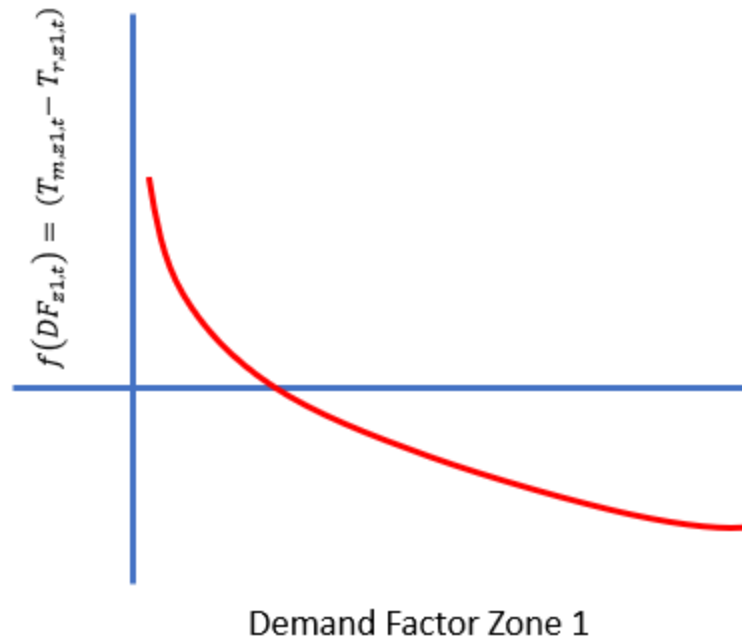


Figure 4-3: Example of Objective Function In Equation 4-1

The bisection method is performed by applying the following steps to the objective function in figure 4-3:

- 1) Select two possible solutions (a high demand factor and a low one) that satisfy the problem constraint (i.e., equation 4-4) and then evaluate the objective function.
- 2) Observe the results from equation 4-1, the goal being to minimize the difference between model and real tank levels.
- 3) The small demand factor will yield a positive result for the objective function and the high demand factor will be negative.
- 4) The solution is somewhere in the middle (also known as a bracketed solution), by testing a 3rd point that lies directly between the first two, we begin to “squeeze” the solution.

5) if the 3rd point is positive, remove the old demand factor that resulted in a positive objective function. If negative, remove the old demand factor and repeat this process until a solution is found.

6) Plug the optimized value of DF_1 into equation 4-4 and solve for DF_2

By utilizing step 6 at every iteration, the bisection method needs only to be applied to a single demand factor, in this case DF_1 . As the objective function in equation 4-1 is minimized, the mass balance in equation 4-4 also yields a minimized difference in real and model tank levels for zone 2. Figure(s) 4-4 and 4-5 give a graphical representation of the bisection method while figure 4-6 shows a flow chart representing the general process of the Bisection method.

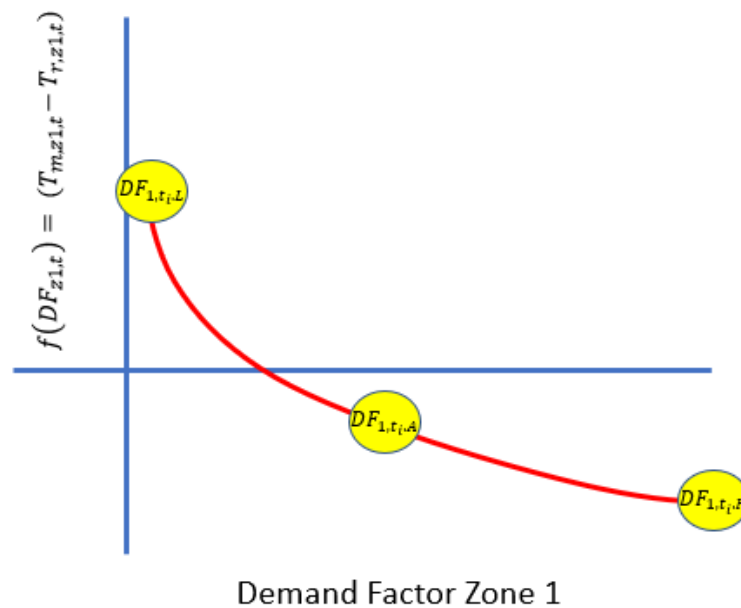


Figure 4-4: Example of Initializing the Bisection Method

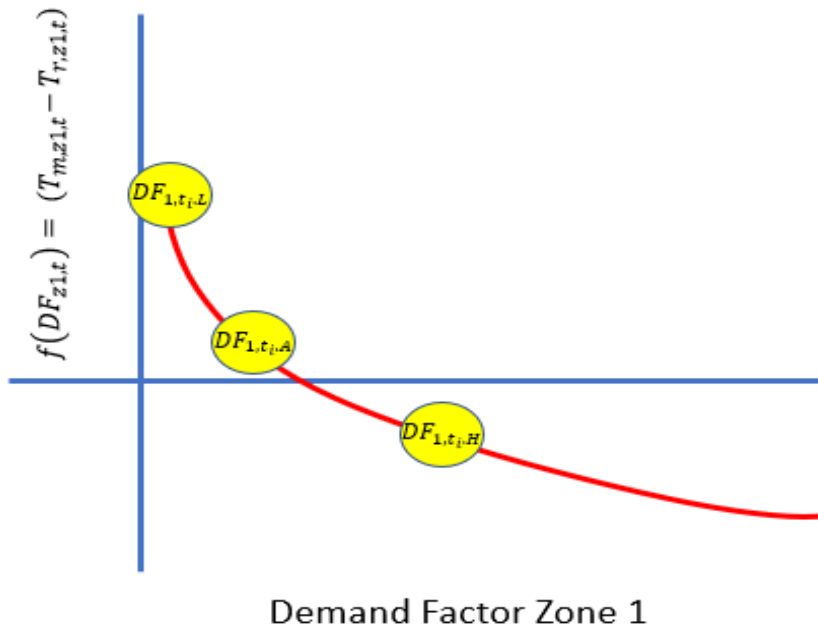


Figure 4-5: Example of Improved Solution Using the Bisection Method

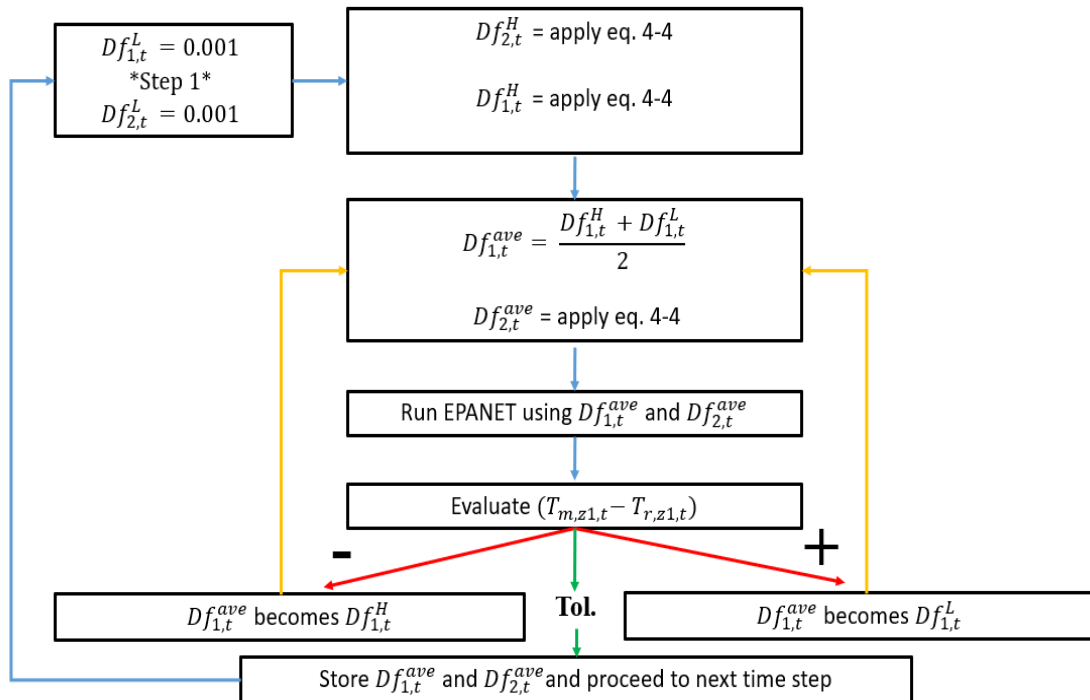


Figure 4-6: General Structure of Bisection Method

4.1.2 Box-Complex Algorithm (Multi Zonal System)

When solving for systems with more than two demand management areas, algorithms which can handle a solution space in multiple dimensions become necessary. Equation 4-4 for multi zonal systems becomes:

$$\Sigma(DF_{n,t_i} * (\Sigma BD_n)) = Q_{plant,t_i} + Q_{tank,t_i} - Q_{sold,t_i} \quad (4-5)$$

Where n is the number of zones (dimensions) that will be used within the Box-Complex algorithm. Because solving for one factor does not guarantee an improved solution for other factors given systems with more than two pressure zones, equation 4-1 becomes:

$$f(DF_{z1,t} \dots DF_{zn,t}) = \Sigma(T_{m,zn,t} - T_{r,zn,t})^2 \quad (4-6)$$

Where $\Sigma(T_{m,zn,t} - T_{r,zn,t})^2$ is the sum of the squared difference between real and model tank levels for all tanks within a given system and $DF_{z1,t} \dots DF_{zn,t}$ describes all associated demand factors for a system with multiple pressure zones.

The Box-Complex method is the constrained form of the Simplex algorithm first introduced by Spendley, Hext and Himsworth (Ormsbee, 1979). A simplex is a geometrical figure consisting of N dimensions, N+1 vertices, and all their connecting sides (Press et al, 2007). Constraints for this algorithm are formulated as either explicit or implicit. Explicit constraints provide explicit bounds on the values that the decision variable can assume. Implicit constraints consist of other equations expressed in terms of the decision variables whose values are also constrained to be either equal to a value (i.e., 0), or greater than or less than a non-zero value. The Box-Complex method is especially suited for nonlinear

optimization problems involving both explicit and implicit inequality constraints. In some cases, implicit equality constraints can be enforced by a separate simulation model which is then linked with the Box-Complex optimization algorithm.

The following generalized steps for the Box-Complex method were referenced from Dr. Lindell Ormsbee's original master's degree thesis "Optimization of Hydraulic Networks Using the Box-Complex Optimization Technique and the Linear Method of Hydraulic Analysis" (1979) and have been slightly changed for this unique application.

- 1) Generate $k \geq n + 1$ points, where n is the number of function variables. Each point contains the necessary number of demand factors depending on the number of pressure zones in the WDN. All of the points are randomly generated with a standard randomizer within programs such as MATLAB and are bounded by equation 4-5. These points are also bounded by an explicit constraint which requires all demand factors to be greater than 0 (homeowners will not discharge into the distribution system).
- 2) Each point is then evaluated given the objective function from equation 4-6. The point with the highest value is deemed "worst" and will be used to then generate a new point in the opposite direction using following steps.
- 3) Reflect the worst point through the centroid of the remaining points:

$$P^* = (1 + \alpha)\bar{P} - \alpha Ph \quad (4-7)$$

Where P^* is the new point, α is an expansion coefficient, \bar{P} is the centroid of the remaining points (all points excluding the current worst point), and Ph is the worst point.

4) Once P^* is generated it is first checked to ensure it satisfies the explicit constraint (i.e., $P^* > 0$). If not, it is contracted halfway back toward the center using equation 4-8 until a feasible point is found. If P^* yields an objective function value less than Ph , then we keep this value and discard Ph . If the new point is worse than Ph , (i.e., has a larger value of the objective function than Ph), then the new point is again contracted back towards the centroid using:

$$P^{**} = \omega Ph + (1 - \omega)\bar{P} \quad (4-8)$$

Where P^{**} is the new point generated and ω is the contraction coefficient. This process is continued until a new point is generated which yields an objective function value less than the current Ph . Once this point is found, it then replaces Ph in the complex, and the process is repeated. Assuming the solution space is convex relative to the objective function, the algorithm should converge to a solution.

For this formulation of the Box-Complex algorithm, it is recommended that the expansion coefficient remain relatively small (anywhere between 1-2 depending on application). By using an expansion factor greater than 1 the simplex is allowed to “search” different regions of the solution space. While values greater than 2 are also acceptable, they may lead to slower convergence within this specific application. Contraction coefficients may be anywhere from 0-1 where 0 will result in the centroid and 1 will result in the old worst point respectively. Additionally, not all demand factors (dimensions in a complex) for the WDN are used. In order to satisfy conservation of mass, $(n-1)$ factors are

manipulated within the complex (where n is the total number of pressure zones), after which the n^{th} factor is generated by solving equation 4-9:

$$DF_{n,t_i} = \frac{Q_{plant,t_i} +/- \sum_{x=1}^{\# \text{ of Tanks}} (Q_{tank x,t}) - Q_{sold,t_i} - \sum_{x=1}^{(n-1)} (DF_{x,t} * (\Sigma BD_x))}{\Sigma BD_n} \quad (4-9)$$

Where Q_{plant,t_i} is the flow into the system from the WTP, $\sum_{x=1}^{\# \text{ of Tanks}} (Q_{tank x,t})$ is the sum of all the flows from every tank in the system at a specific time step (where leaving the tanks are considered positive), Q_{sold,t_i} is the water being sold, and $\sum_{x=1}^{(n-1)} (DF_{x,t} * (\Sigma BD_x))$ are the demands in every zone with the exception of the n^{th} zone.

An example of the expansion and contraction process is demonstrated in figures 4-7 and 4-8 below (yellow circle is the solution). Given the dimensionality of the solution space in these figures, the implied number of demand factors and therefore their respective pressure zones are three. The Box-Complex method first determines the demand factor for the first two zones (DF_1 and DF_2) while the demand factor for zone 3 (DF_3) is solved using equation 4-9 and is dependent upon DF_1 and DF_2 . This same approach can be extended to problems involving additional demand factors (i.e., > 3). The general structure of the algorithm is provided in Figure 4-9. Example code for optimizing a single hour for a four-zone system is given in Appendix D

It should be noted that the Box-Complex method does not guarantee the global maximum or minimum. However, if the algorithm is repeatedly run with a different set of initial demand factors and it continues to converge to the same solution, that would suggest that a global optimum has been achieved.

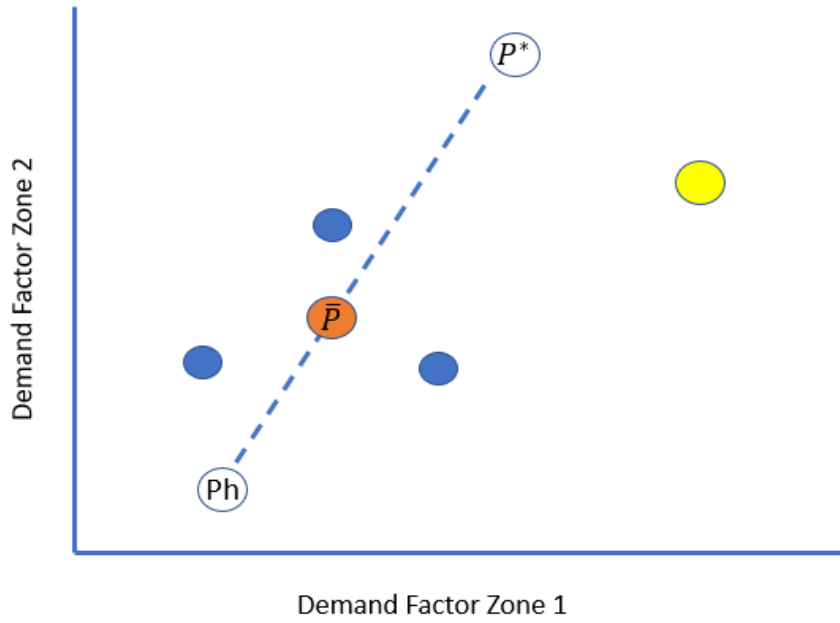


Figure 4-7: Example Expansion Using the Box-Complex Method

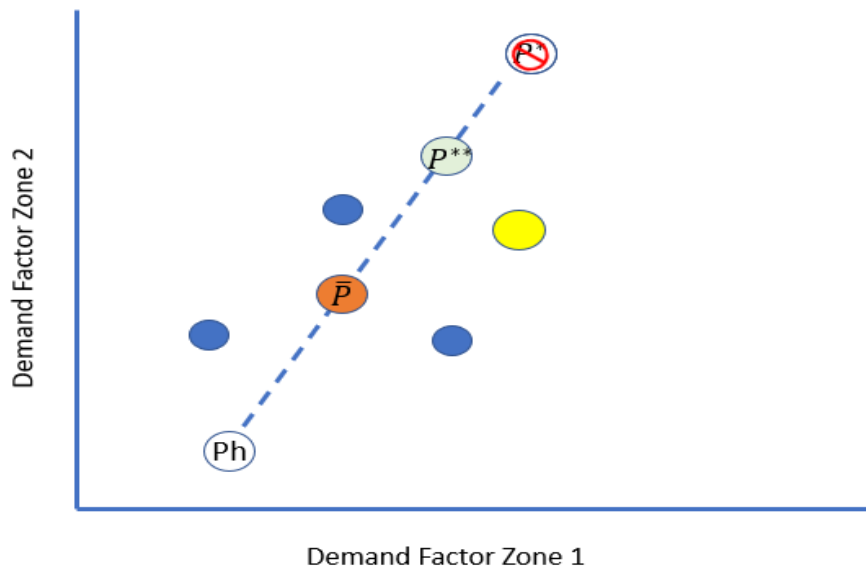


Figure 4-8: Example Contraction Using the Box-Complex Method

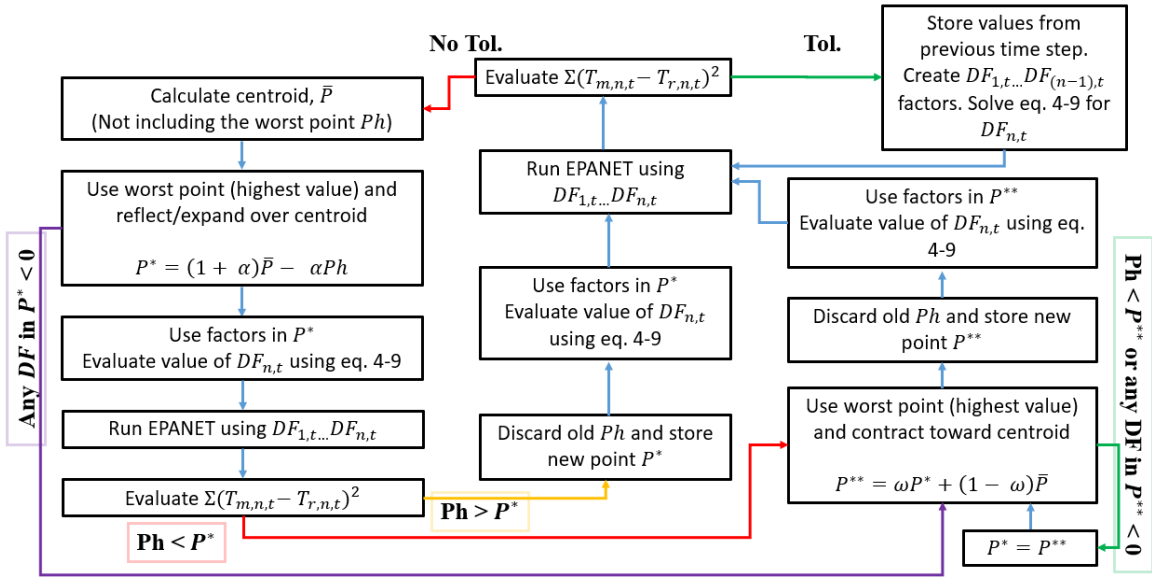


Figure 4-9: General Algorithm for Box-Complex Method

4.2 Hydraulic Calibration of LWW Digital Twin

The hydraulic model underlying the digital twin was originally provided by the Kentucky Engineering Group while consulting for the LWW system. This model, given as a “.p2k” KYPIPE file, was converted into an EPANET “.inp” format and checked to ensure no information was lost in the export process.

The nodal elevations and C-factors for the pipes given from this model are assumed satisfactory for the desired outcomes and have not been altered from the original file. Other topographical information in the model was confirmed using information provided by the LWW system. In particular, the Springfield Tank, which has a non-cylindrical geometry, required updating so that it accurately reflected discharge rates as a function of changing elevation. Similarly, the two pumps within the model, the water treatment plant (WTP) and booster pump stations have been confirmed as accurately populated within the model (per meeting with Kentucky Engineering Group) and have not been altered in any way.

Calibration of the model therefore encompasses the accurate creation of demand factors which comprise the extended period simulation (EPS). The first step taken towards accomplishing this was to delineate the pressure zones within the system itself. Figure 4-10 identifies the valves that were closed in the actual system (as well as in the model) which allowed for the isolation of “Zone 1” and “Zone 2”. Zone 1, which is associated with Cavalry Tanks and the water treatment plant, is in the southern portion of the system. Zone 2, which is associated with the Springfield Tank and booster pump, is in the northern portion of the system.

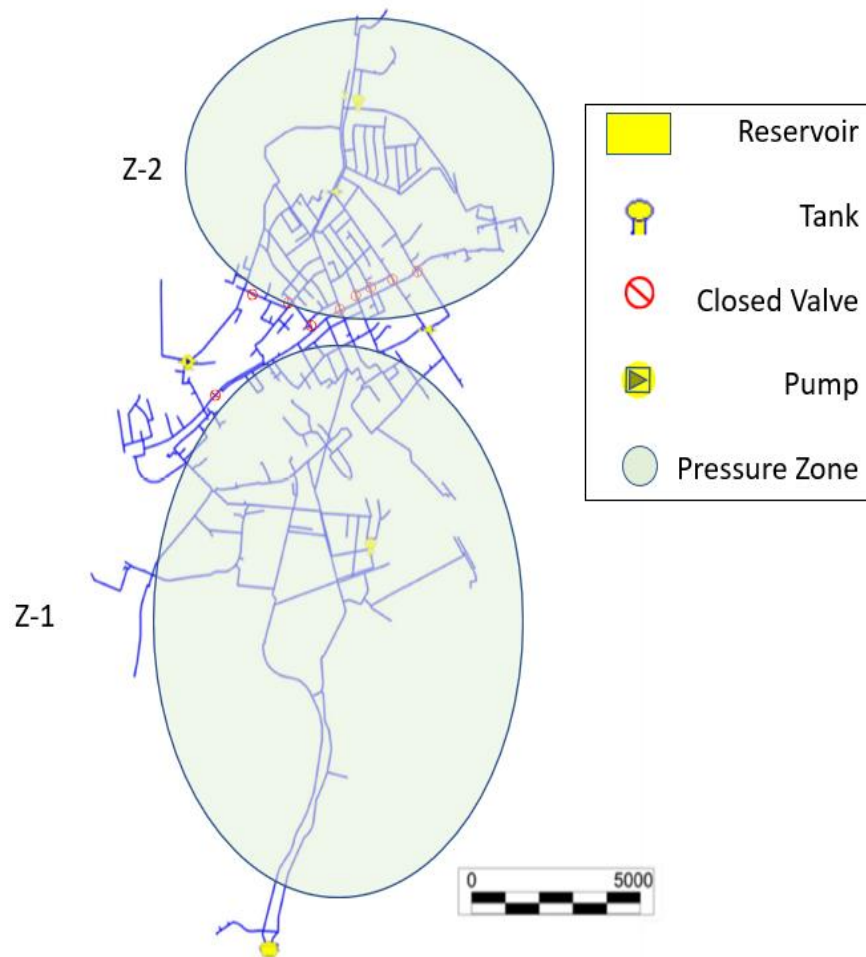


Figure 4-10: LWW North and South Pressure Zone Delineation

After splitting the system into two clearly defined zones, the total base demand for each zone (ΣBD in equation 4.4) is found by summing the base demands of junctions within zones 1 and 2 respectively:

$$\Sigma BD_1 = 254.99 \text{ gpm}$$

$$\Sigma BD_2 = 116.35 \text{ gpm}$$

These values are unique to the closed valve locations which, if changed, will require redistribution of the base demands to zone 1 and 2 respectively. Having found these values, equation 4.4 becomes:

$$(DF_{1,t_i} * (254.99 \text{ gpm})) + (DF_{2,t_i} * (116.35)) = Q_{plant,t_i} + Q_{tank,t_i} - Q_{sold,t_i}$$

Now that the relevant model parameters on the left-hand side of the equation have been found, we move to the right-hand side dealing with real model data given from LWW.

The parameters on the right-hand side including plant flow rate, and tank discharge (which are a function of tank geometry and level over time) were given for the time period of June 1st, 2023, through July 24th, 2023, with data points given every two minutes. The last parameter, the sum of the master meter demands, were given for the time period of June 19th, 2023, through July 20th, 2023, with information on these demands given every hour. With the master meter demands constraining the period for which we can create the proper demand factors, a two-week period was chosen starting on June 20th and running through July 3rd.

The next step in this process is to understand how we can appropriately use this data to find demand factors for the system. Throughout the entirety of this project, there were a plethora of data errors that were encountered and cleaned which took place before the implementation of the algorithm. While a more comprehensive exploration of pitfalls arising from flawed data can be found in Walski et. al (2012), this discussion specifically addresses the issues of “data latching” and managing “noise”.

Data latching occurs when the reporting interval for the SCADA system is much more frequent than the data being sent to it from the transmitter itself. In the case where signal is lost from the tank, the SCADA system will continue to report the same data point until the signal is found again. In observing the raw data set from LWW, there are several instances where the value of the tank levels does not appear to change for several minutes at a time. This is highly unlikely to reflect reality and it is concluded to be a result of data latching.

Tank sensors are also susceptible to noise, which is defined as the “random variations of sensor output unrelated to the variation in sensor input” (Masi, 2020). In the case of telemetry data, electronic sensors may only be expected to be accurate to within one-tenth of a foot (Walski et al., 2012). Because tank discharge is calculated as a function of the change in tank level between two time periods, noise may wreak havoc in cases where the period is sufficiently small (figure 4-11).

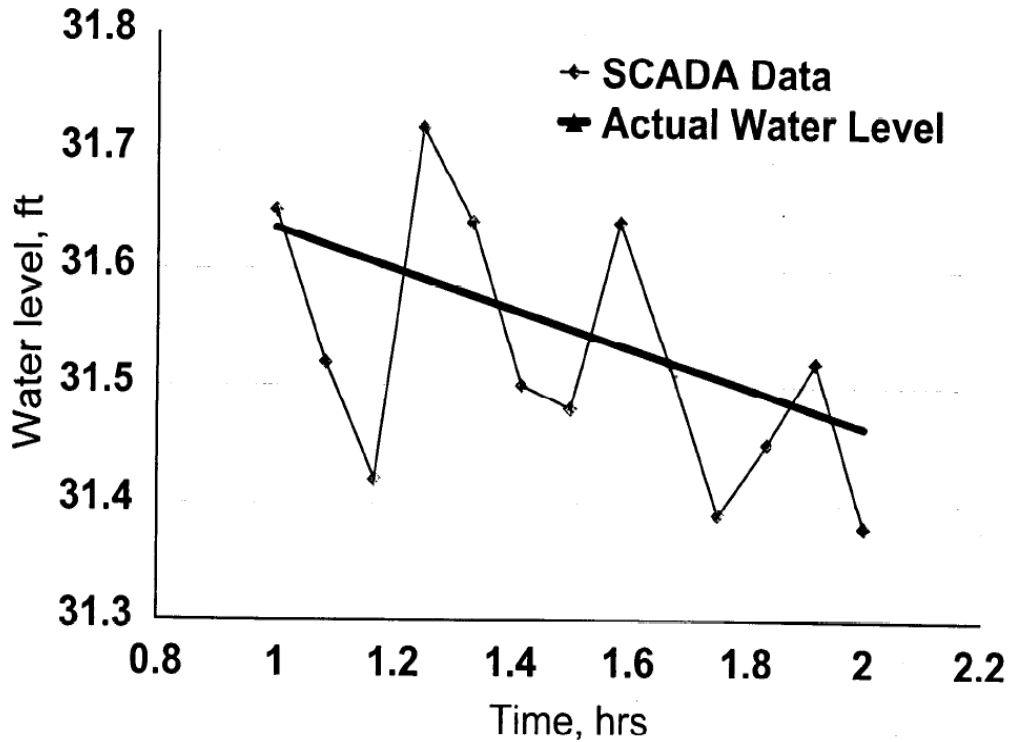


Figure 4-11: Example of Noise in Tank Data (Walski et al., 2012)

In the first formulation of the bisection method for this digital twin, time intervals of 2-minutes were used for the tanks while pump data was disaggregated from their original 1-hour frequency into 10-minute frequencies. Errors that propagated were negative demands which were necessary to satisfy equation 4.4 but were obviously not representative of reality and occurred in approximately 21% of the 720 computed demand factors. By choosing a time interval of 1 hour, equation 4.4 resulted in demand values that were almost never in error (outside of significant periods of data latching) and accounted for only 2% of the 672 computed demand factors. This is consistent with expected error in tank flow rates using a 1-hour interval given the error in tank level (figure 4-12).

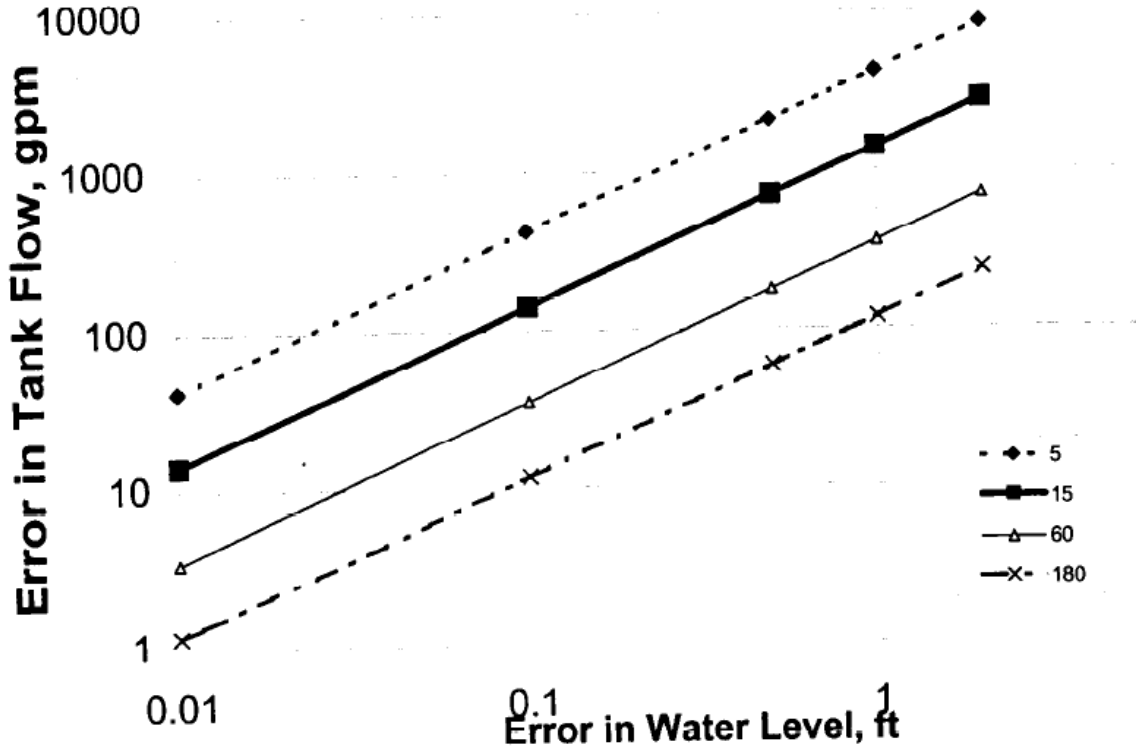


Figure 4-12: Flow Error Using Separate Time Intervals (Walski et al., 2012)

The function created in MATLAB to calculate these demand factors is called “demandButtonTestPushed” (Appendix B) and follows these general steps:

- 1) Initialize the hydraulic simulation with starting tank levels, water treatment plant flow rates, booster pump on/ off times, and master meter demands.
- 2) Create an array where the first row contains demand factor values for zone 1 which are very low (.001) and produce demand factors for zone 2 in row two by plugging DF_1 into equation 4.4 and solving for DF_2 .
- 3) Create a second array making the demand factor for zone 2 very small (.001) and produce demand factors for zone 1 that are constrained by equation 4.4

(with this array also having the factors for zone 1 and 2 occupying rows 1 and 2 respectively).

- 4) By doing this, we have two arrays that contain or “sandwich” the solution (being that each zone contains factors which are as high and as low as they may be when constrained by equation 4.4).
- 5) A third array is generated by taking the first rows of the first two arrays, summing them, and then dividing them by two. Once this is done, its complimentary demand factor in zone 2 is calculated again by using equation 4.4.
- 6) This third point is the one which will be tested and updated by the bisection algorithm. After running the simulation, if the error value is negative, the tank level in the model is too low and the demand factor needs to become smaller. This is accomplished by removing the 1st array which contained the high demand factor for zone 1 and keeping the other two arrays. A new third array is created, and the process continues until sufficient convergence (which is arbitrarily defined as when the model tank level for zone 1 is within .005 feet of the real tank level.)
- 7) In the cases where tank levels cannot converge on their real-world values, an error adjustment is made. This adjustment takes the total flow needed to either be discharged or added to the tanks on the pervious iteration and adds that volume to the total demand (right hand side of equation 4.4) onto the current iteration. This will cause the current iteration to produce demand factors that are not technically representative of reality, however, it will realign the tank

levels to where they are supposed to be at a given time step and will allow for the following demand factors to be calculated properly.

An example of this process using data from June 20th for a single hour is calculated as follows:

The starting tank levels are 1009.40' and 966.07' for Springfield Road and Calvary Tanks respectively. There is no flow coming from the water treatment plant at midnight on the 20th of June, the booster pump is not on in that first hour, and the total master meter demand over this period is found to be 489.01 gpm. After 1 hour the tank levels are 1007.37' and 965.30' which results in a discharge from both tanks of 345.15 gpm and 352.69 gpm. Equation 4.4 therefore becomes:

$$(DF_1 * (254.99)) + (DF_2 * (116.35)) = 0 + 697.84 - 489.02$$

To create the first array (in this case it will just be the single demand factor for the first hour) the value of DF_1 is set to .001 and DF_2 is calculated as 1.79. The same is done for array 2 by setting DF_2 to .001 and solving for DF_1 which is calculated as 0.82. Array 3 is calculated as the average of row 1 of both arrays and is found to be 0.41. Row 2 of array 3 is again constrained by equation 4.4 and found to be 0.89.

$$Array\ 1 = \begin{bmatrix} 0.001 & DF_{1,t2} & DF_{1,tn} \\ 1.79 & DF_{2,t2} & DF_{2,tn} \end{bmatrix}$$

$$Array\ 2 = \begin{bmatrix} 0.82 & DF_{1,t2} & DF_{1,tn} \\ 0.001 & DF_{2,t2} & DF_{2,tn} \end{bmatrix}$$

$$Array\ 3 = \begin{bmatrix} 0.41 & DF_{1,t2} & DF_{1,tn} \\ 0.89 & DF_{2,t2} & DF_{2,tn} \end{bmatrix}$$

Testing the 3rd array yields an error of -0.06' for the cavalry tank. Because this is negative, array 2 is deleted and a new third point is created between the remaining two arrays.

$$Array\ 1 = \begin{bmatrix} 0.001 & DF_{1,t2} & DF_{1,tn} \\ 1.79 & DF_{2,t2} & DF_{2,tn} \end{bmatrix}$$

$$Array\ 2 = \begin{bmatrix} 0.41 & DF_{1,t2} & DF_{1,tn} \\ 0.89 & DF_{2,t2} & DF_{2,tn} \end{bmatrix}$$

$$Array\ 3 = \begin{bmatrix} 0.21 & DF_{1,t2} & DF_{1,tn} \\ 1.33 & DF_{2,t2} & DF_{2,tn} \end{bmatrix}$$

The error value with the new 3rd array is now .05 for the Cavalry Tank. This process is continued until the tank in zone 1 is within the .005' tolerance. Because this equation is perfectly constrained by equation 4.4, the demand factor in zone 2 will result in tank levels at or near the tolerance specified for zone 1.

While convergence in this case and most of the other cases is not an issue, for the sake of demonstration let's assume that the levels in both tanks are still higher than they are supposed to be at this time step using the optimized demand factors. If the total volume of flow in the model that needs to be drained to meet the real-world tank levels is 5000 gallons, that value will be added to the right-hand side of equation 4.4 on the next iteration. The demand factors in the next iteration will then compensate for the error from the previous iteration by being slightly higher than they would've been had the error not been there. With the tank levels back to where they should be at the end of hour 2, the demand factors for hour 3 may now be comfortably calculated.

This approach was used to generate a series of daily demand patterns (using a 1-hour time step) for each day between Jun 20th and July 3rd, 2022 (see Appendix A). These patterns thus provide the operators with a library of actual system demand patterns for individual days of the week including weekends (i.e., Saturday and Sunday). This leaves the operator with the option of providing an estimate of the projected total demand for the next day along with one of the available demand patterns which will then be scaled up or down to match the projected demand, and thus provide a projected hourly demand pattern for the next day.

CHAPTER 5. DIGITAL TWIN APPLICATION

Combining the system needs and general methodology in Chapter 3 with the calibration steps noted in Chapter 4, this chapter details the specific steps taken to develop a digital twin model for the Lebanon Water Works (LWW) system. In addition, details are provided on the process of creating the graphical user interface as well as validating model outputs.

5.1 Creation of the Graphical User Interface (GUI)

MATLAB version R2022A (MathWorks, 2022) was selected as the development platform for creating a GUI for the Lebanon digital twin. MATLAB was chosen for this task because of the existing link between the EPANET engine through the EPANET-MATLAB toolkit in addition to the “App Developer” toolkit existing within the MATLAB framework. The app developer toolkit (MathWorks, 2022) allows for simple drag and drop interactive elements where functions may be coded which tie EPANET functionality to the button itself.

The first step in this process is the development of the home page, where operators first engage with the digital twin (figure 5-1). Here we can see all the elements with which users can interact and where information can be placed to initialize their EPS simulation.

SIM TIMES AND WATER QUALITY

General (EPS + Water Quality)		Water Quality	
Hydraulic Time Step	1	Bulk	-0.08
Water Quality Time Step	1	Wall	-0.009
Total Time	24	Chlorine (mg/l)	1.71

TANKS

Springfield Road Tank		Calvary Tanks	
Initial Tank level	19.9	Initial Tank level	59
<input type="checkbox"/> Use		<input type="checkbox"/> Use	
Springfield Road Pump		WTP	
On When Below	14	On When Below	45
Off When Above	19	Off When Above	59

PUMPS (Use If Not Specified In Tanks Section)

WTP		Springfield Road Pump	
ON AT TIME		ON AT TIME	
OFF AT TIME	0:00	OFF AT TIME	0:00
ON AT TIME		ON AT TIME	
OFF AT TIME		OFF AT TIME	
ON AT TIME		ON AT TIME	
OFF AT TIME		OFF AT TIME	
ON AT TIME		ON AT TIME	
OFF AT TIME		OFF AT TIME	
ON AT TIME		ON AT TIME	
OFF AT TIME		OFF AT TIME	
ON AT TIME		ON AT TIME	
OFF AT TIME		OFF AT TIME	

SPECIFY USING MILITARY TIME
Ex. 0:00 or 20:00

DEMAND PATTERNS

Demand Settings	
Zone 1 Pattern	Average Weekday
Zone 2 Pattern	Average Weekday
Scale Zone 1 Pattern By:	0
Scale Zone 2 Pattern By:	0
Leave Scale 0 If You Do Not Want to Scale Demand Pattern	

Demand Pattern Zone 1

Demand Pattern Zone 2

Lebanon Water **RUN EPS!** UK

Figure 5-1: Home Screen for the Digital Twin

In the top left corner, users encounter the “Sim Times and Water Quality” section. Here the length of the simulation, its time and water quality steps (which control the precision of the results), the bulk and wall decay rates, and chlorine concentration in (mg/l) are specified (figure 5-2). The hydraulic and water quality time steps are given as options to the operator in the event that refinement in report results is deemed appropriate. The bulk and wall decay rates for this system are given as preset values from research done at the University of Kentucky (Gautam and Ormsbee, 2023). While operators are discouraged from changing this value, they are given this as an option because these values allow for the refinement of observed chlorine concentration values in the distribution system to match what is seen in the model. Because chlorine residuals are highly impacted by seasonality, giving this as an option to operators allows for the relatively simple calibration of outputs to match inevitable changes in residuals over time.

SIM TIMES AND WATER QUALITY	
General (EPS + Water Quality)	Water Quality
Hydraulic Time Step	<input type="text" value="1"/>
Water Quality Time Step	<input type="text" value="1"/>
Total Time	<input type="text" value="24"/>
	Bulk <input type="text" value="-0.08"/>
	Wall <input type="text" value="-0.009"/>
	Chlorine (mg/l) <input type="text" value="1.71"/>

Figure 5-2: Time and Water Quality Screen

Just below the Sim Times and Water Quality menu is the “Tanks” section which allows the operator to input initial tank levels as well as “control statements” for pumps associated with the tanks (figure 5-3). Control statements are statements within EPANET that dictate conditions under which pumps turn on and off. In this case the conditions are made relative to the tanks and are enabled in pressing the “use” switch under each respective tank in the Lebanon system. For example, by specifying “use” for the Springfield Road Tank, putting the number 14 in the “On When Below” text box, and the number 19 in the “Off When Above” box, a control statement is sent to EPANET that turns the booster pump on when the Springfield Road Tank is below 14’ and off when the tank is above 19’.

The screenshot shows a software interface titled "TANKS" with a yellow header. It is divided into two main sections: "Springfield Road Tank" and "Calvary Tanks".

Springfield Road Tank	Calvary Tanks
Initial Tank level: <input type="text" value="19.9"/>	Initial Tank level: <input type="text" value="59"/>
<input checked="" type="checkbox"/> Use	<input checked="" type="checkbox"/> Use
Springfield Road Pump	WTP
On When Below: <input type="text" value="14"/>	On When Below: <input type="text" value="45"/>
Off When Above: <input type="text" value="19"/>	Off When Above: <input type="text" value="59"/>

Figure 5-3: Time and Water Quality Screen

Pump operations may also be specified by time. Immediately to the right of the “Sim Times and Water Quality” and “Tanks” menus is the “Pumps” menu. Here users are given the option to specify (in military time) when the water treatment plant and booster pumps turn on and off (figure 5-4). The program has been created such that if either of the pump conditions in the tanks section are specified as “use”, all the inputs in the pumps

section will be ignored. The reason for this is to avoid confusion around the complex switching on and off of pumps that occur when several conditions are sent to EPANET. Because the goal of the digital twin is to promote simplicity, access to the background EPANET output file is not given. However, because of this, if several different types of conditional statements are given, it might be difficult to determine how EPANET has interpreted them and troubleshoot any noticeable errors.

PUMPS (Use If Not Specified In Tanks Section)

WTP		Springfield Road Pump	
ON AT TIME	<input type="text"/>	ON AT TIME	<input type="text" value="0:00"/>
OFF AT TIME	<input type="text" value="0:00"/>	OFF AT TIME	<input type="text" value="1:00"/>
ON AT TIME	<input type="text" value="1:50"/>	ON AT TIME	<input type="text" value="2:00"/>
OFF AT TIME	<input type="text" value="6:00"/>	OFF AT TIME	<input type="text" value="5:00"/>
ON AT TIME	<input type="text" value="12:30"/>	ON AT TIME	<input type="text" value="16:00"/>
OFF AT TIME	<input type="text" value="14:50"/>	OFF AT TIME	<input type="text" value="17:43"/>
ON AT TIME	<input type="text" value="23:25"/>	ON AT TIME	<input type="text" value="19:21"/>
OFF AT TIME	<input type="text" value="24:00"/>	OFF AT TIME	<input type="text" value="21:00"/>
ON AT TIME	<input type="text"/>	ON AT TIME	<input type="text"/>
OFF AT TIME	<input type="text"/>	OFF AT TIME	<input type="text"/>
ON AT TIME	<input type="text"/>	ON AT TIME	<input type="text"/>
OFF AT TIME	<input type="text"/>	OFF AT TIME	<input type="text"/>
ON AT TIME	<input type="text"/>	ON AT TIME	<input type="text"/>
OFF AT TIME	<input type="text"/>	OFF AT TIME	<input type="text"/>
ON AT TIME	<input type="text"/>	ON AT TIME	<input type="text"/>
OFF AT TIME	<input type="text"/>	OFF AT TIME	<input type="text"/>

SPECIFY USING MILITARY TIME
Ex. 0:00 or 20:00

Figure 5-4: Time and Water Quality Screen

At the bottom of the page is the final component relative to the available options for extended period simulation within the digital twin model, i.e., “Demand Patterns”. The intention for this data menu is to allow operators the ability to incorporate pre-processed demand factors (appendix A) from data derived directly from the LWW system which were created using the demand calibration methods discussed in section 4.2. The demand factors for the two zones represent data from June 20th through July 3rd, 2023 and allow the operator to pick any days between them or the average weekday or weekend for that time period. If the operator deems it appropriate, they may also scale the demand patterns up and down through a simple scaling factor text box like what is found in the pumps section

(figure 5-5). To provide the operator clarity on what the factors for each zone look like, a graph is also provided which will reflect any changes made in the demand options (figure 5-6).

Demand Settings

Zone 1 Pattern Zone 2 Pattern

Scale Zone 1 Pattern By:

Scale Zone 2 Pattern By:

Leave Scale 0 If You Do Not Want to Scale Demand Pattern

Figure 5-5: Demand Setting Section

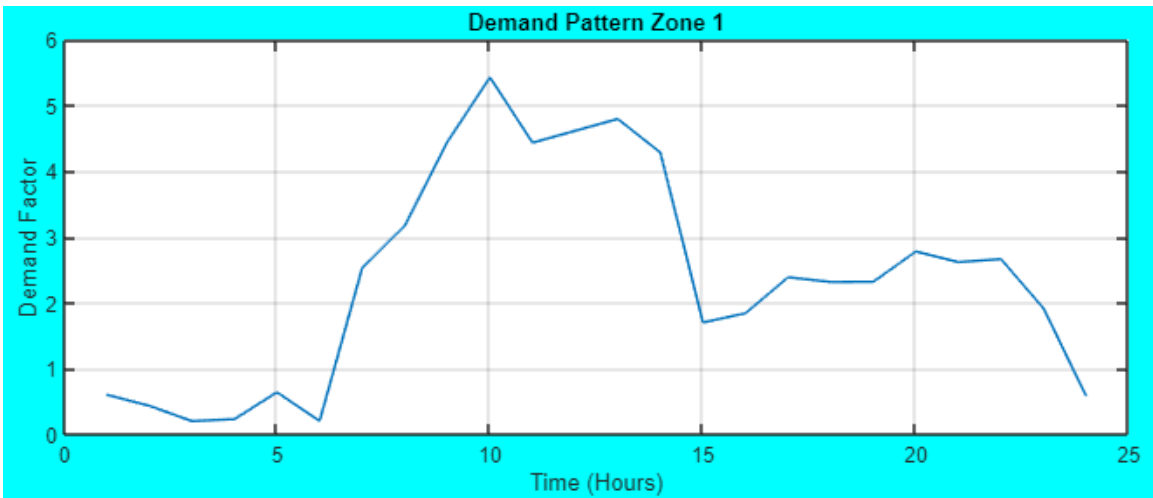


Figure 5-6: Example Graph Given for Demand Pattern In Zone

Once the operator has specified all initial inputs, MATLAB stores this information within the function associated with the “Run EPS!” button and then sends this data to the “ExtendedPeriodV2” function in MATLAB (see “RUNEPSButtonPushed” and ExtendedPeriodV2 functions in appendix B). The ExtendedPeriodV2 function uses options in the EPANET-MATLAB toolkit to signal for an EPS to be run and returns output values that may be used for visualization in the application.

The outputs may be visualized in the “new results” tab (figure 5-7) where the operator may view several parameters at select junctions, pumps, and tanks throughout the system. In the “Tanks” section, the user may select the drop down and specify whether they want to graph water age or the tank levels as a function of time. Depending on what input is specified in either of the graphs, the output table in that section will reflect the tabulated results of the selected parameter (figures 5-8 and 5-9).

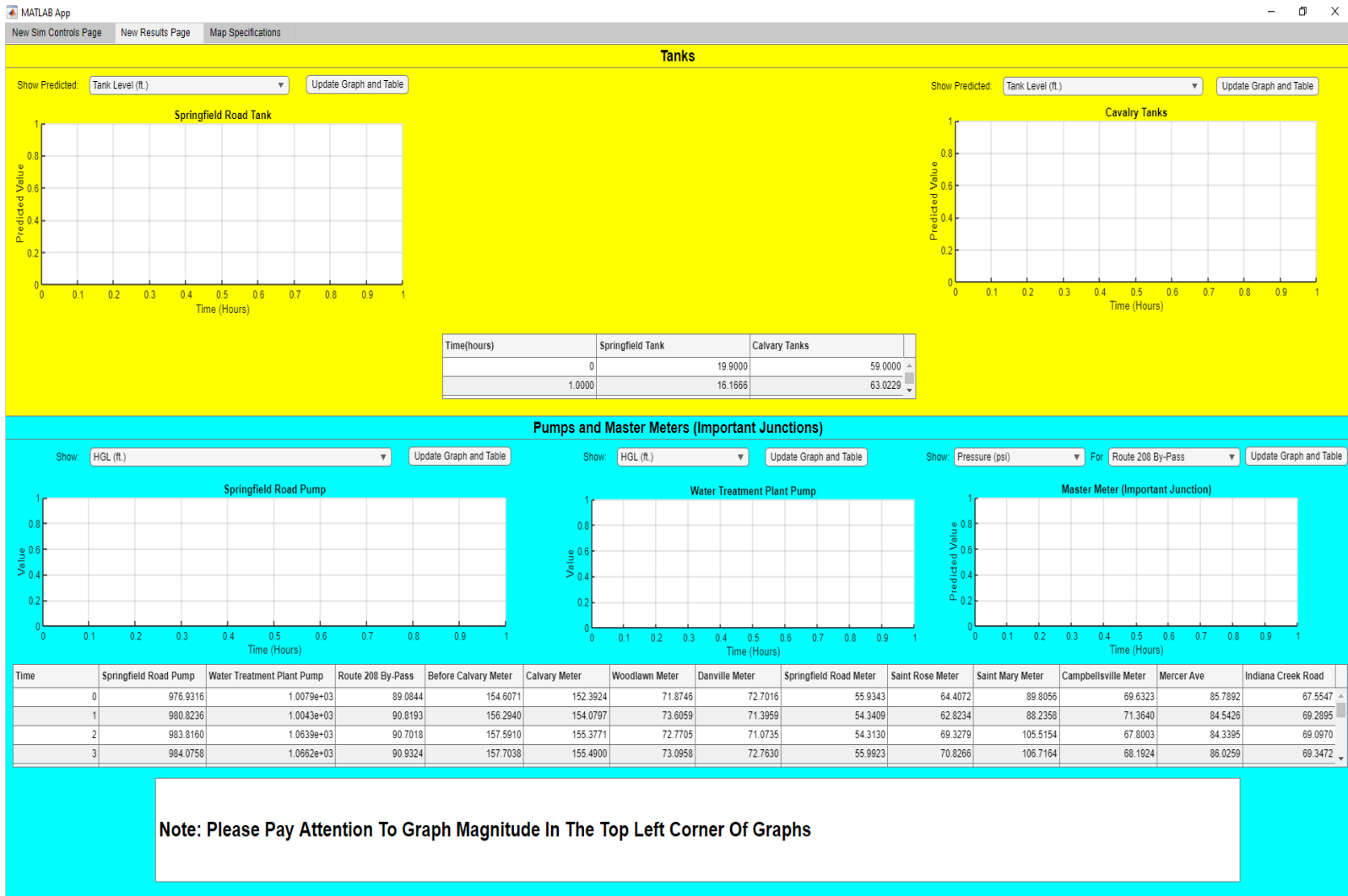


Figure 5-7: Results Page from EPS

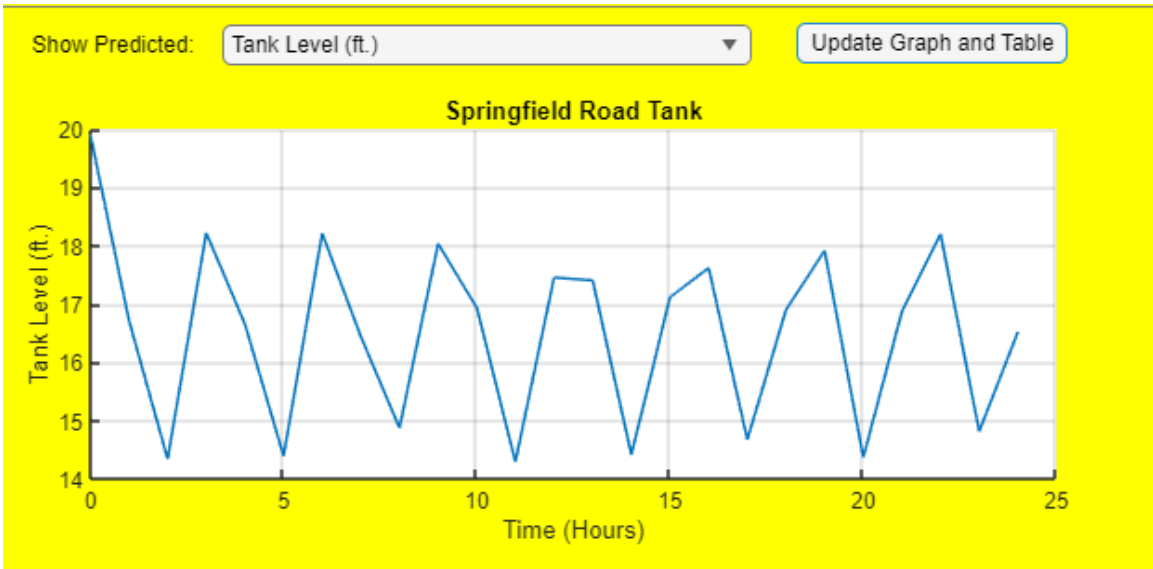


Figure 5-8: Example of Tank Level Output

Time(hours)	Springfield Tank	Calvary Tanks
0	19.9000	59.0000 ▲
1.0000	16.7173	62.8047 ▼

Figure 5-9: Tabulated Results Given Selection of Tank Level

In the “Pumps and Master Meters (Important Junctions)” section, the user can choose parameters from the drop-down menu relative to either the Springfield Road (booster) pump, the water treatment plant pump, or master meters. The pumps allow for the selection of plotting either the hydraulic grade line (HGL) in feet or the flow rate (gpm) while the last graph takes two inputs, the parameter sought after and the specific junction with which we are interested. The parameters available in the last graph include plotting pressure (psi), demand (gpm), chlorine (mg/l), and total trihalomethane (TTHM) (mg/l). Like the Tanks section, once the graphs are updated, the table will reflect the parameters chosen detailing the data shown in the graphs.

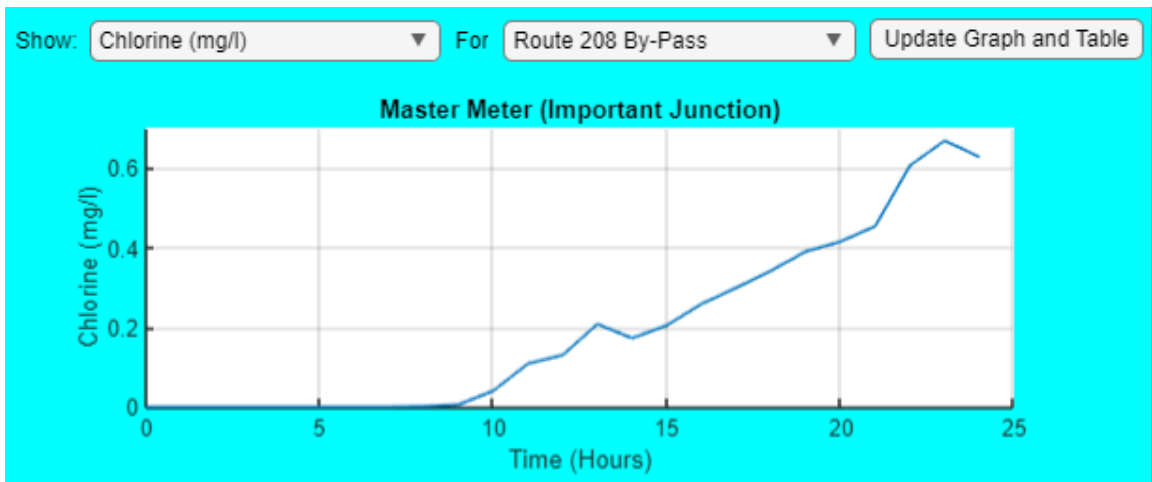


Figure 5-10: Example Junction Output

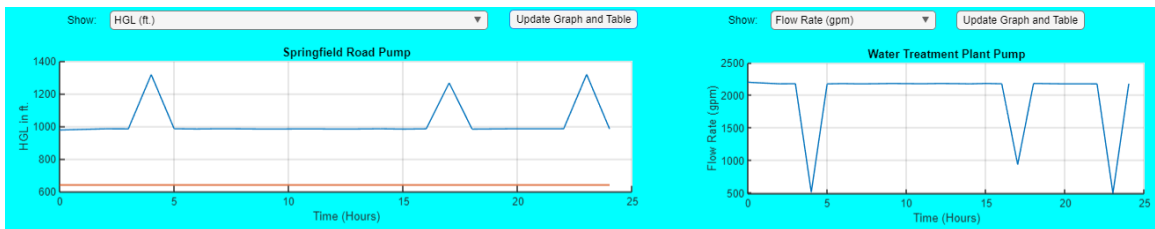


Figure 5-11: Example Pump Output

Time	Springfield Road Pump	Water Treatment Plant Pump	Route 208 By-Pass	Before Calvary Meter	Calvary Meter
0	976.8995	2.1979e+03	0	0	0
1	980.6006	2.1857e+03	9.4913e-07	0.1641	1.4280
2	984.0333	2.1743e+03	3.5597e-09	1.6945	1.7079
3	983.7931	2.1751e+03	5.1820e-05	1.6971	1.7089

Figure 5-12: Example of Output Table Given Inputs for Each of the Graphs

To make the data accessible for the operator, the menu allows the user to only select a small portion of the total system junctions (all of which are master meters except for two which were arbitrarily selected to represent the northern and southern portions of the system). One of the parameters available for selection for the “Master Meter (Important Junctions)” graph is TTHM concentrations. This calculation is based upon a simple linear relationship from which chlorine demand is related to TTHM based on data specific to the LWW system (Gautam and Ormsbee, 2023). This relationship is modeled as follows:

$$Chlorine_{Demand} \left(\frac{mg}{l} \right) = Chlorine_{Plant} \left(\frac{mg}{l} \right) - Chlorine_{Junction} \left(\frac{mg}{l} \right)$$

$$TTHM \left(\frac{mg}{l} \right) = 0.0508 (Chlorine_{Demand})$$

As a result, the EPANET model is used to calculate the chlorine demand at the selected junction nodes from which the TTHM concentration is then determined and displayed.

The “Map Specifications” tab is the last element of the application and allows for visualization of outputs as they vary both spatially and temporally. Once the user presses the “Generate Generic Map” button, the “GenerateGenericMapButtonPushed” function is

run in MATLAB which then generates a window detailing the pipe diameters of the LWW system (figures 5-13 and 5-14).

Once the generic map has been created, the user may proceed to the right half of the tab where pressure, flow, and chlorine specifications may be made; each of these having a check box for extended period simulation and a corresponding slider for selecting the period of interest (figure 5-15). The user has the choice to plot the pressures and flows or chlorine residual by simply pressing the “Generate Nodal Pressure and Pipe Flows Map” or “Generate Nodal Chlorine Residuals Map” buttons.

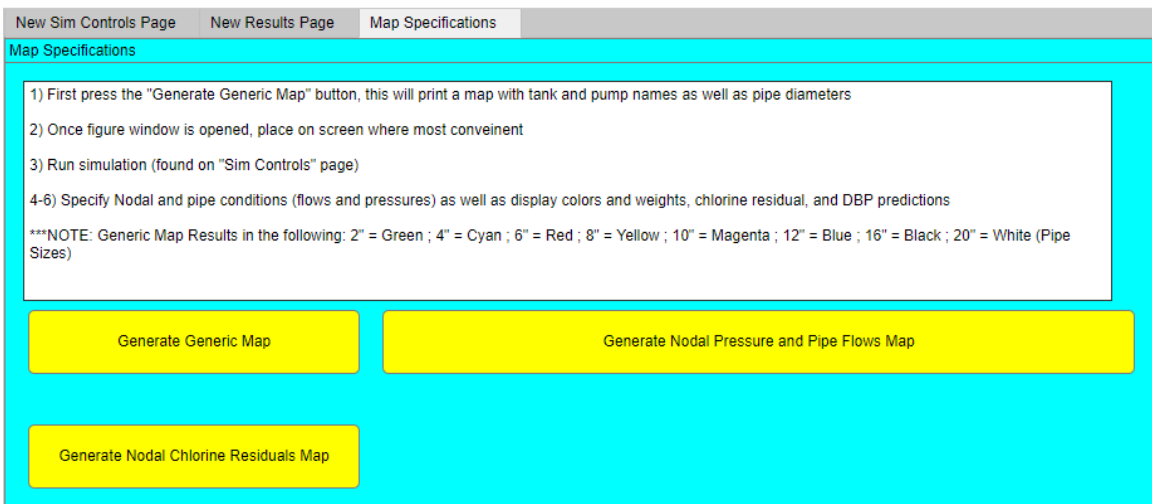


Figure 5-13: Buttons for Graphing Results



Figure 5-14: Generated Interactive Map of LWW Showing Pipe Diameters

Nodal (Junction) Pressure

High Pressure (psi)	80	Color 1	gl ▼	<input type="checkbox"/> EPS?	EPS (Hour) <input style="width: 100%; height: 15px;" type="text"/>
Medium Pressure (psi)	60	Color 2	bl ▼		
Low Pressure (psi)	20	Color 3	yt ▼		
		Color 4	re ▼		

Above high pressure value will plot as color 1
 Above medium pressure value will plot as color 2
 Above low pressure values will plot as color 3
 Below low pressure value will plot as color 4

Link (Pipe) Flow

High Flow (cfs)	8	Color 5	gl ▼	<input type="checkbox"/> EPS?	EPS (Hour) <input style="width: 100%; height: 15px;" type="text"/>
Low Flow (cfs)	2	Color 6	yt ▼		
		Color 7	re ▼		

Above high flow value will plot as color 5
 Above low flow value will plot as color 6
 below low flow values will plot as color 7

Nodal (Junction) Chlorine

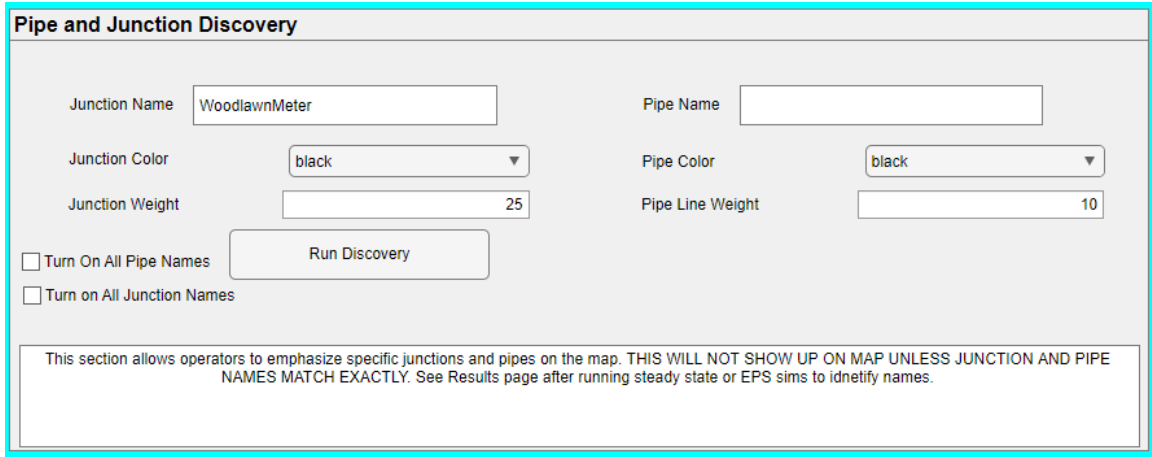
High Concentration (mg/l)	1.5	Color 8	gl ▼	<input type="checkbox"/> EPS?	EPS (Hour) <input style="width: 100%; height: 15px;" type="text"/>
Low Concentration (mg/l)	0.4	Color 9	yt ▼		
		Color 10	re ▼		

Above high concentration value will plot as color 8
 Above low concentration value will plot as color 9
 below low concentration values will plot as color 10

Figure 5-15: Tools for Map Visualization

If the user is unsure of where a specific pipe or junction is located within the system, the “Pipe and Junction Discovery” field may be populated. The user may enter specific pipe or junction names within the text fields and specify what color and size they want that element to appear as in the map window. In addition, the user may also select the “Turn On All Pipe Names” or the “Turn On All Junction Names” check boxes and they will also appear in the map window (figures 5-16 and 5-17). This functionality is all controlled by the “RunDiscoveryButtonPushed” function and may be found in Appendix B. Note: The

names, structure, and logic of each of the functions tied to the buttons are found in appendix B with the appended term “ButtonPushed” at the end.



Pipe and Junction Discovery

Junction Name Pipe Name

Junction Color Pipe Color

Junction Weight Pipe Line Weight

Turn On All Pipe Names

Turn on All Junction Names

This section allows operators to emphasize specific junctions and pipes on the map. THIS WILL NOT SHOW UP ON MAP UNLESS JUNCTION AND PIPE NAMES MATCH EXACTLY. See Results page after running steady state or EPS sims to identify names.

Figure 5-16: Discovery Tool for Map Visualization

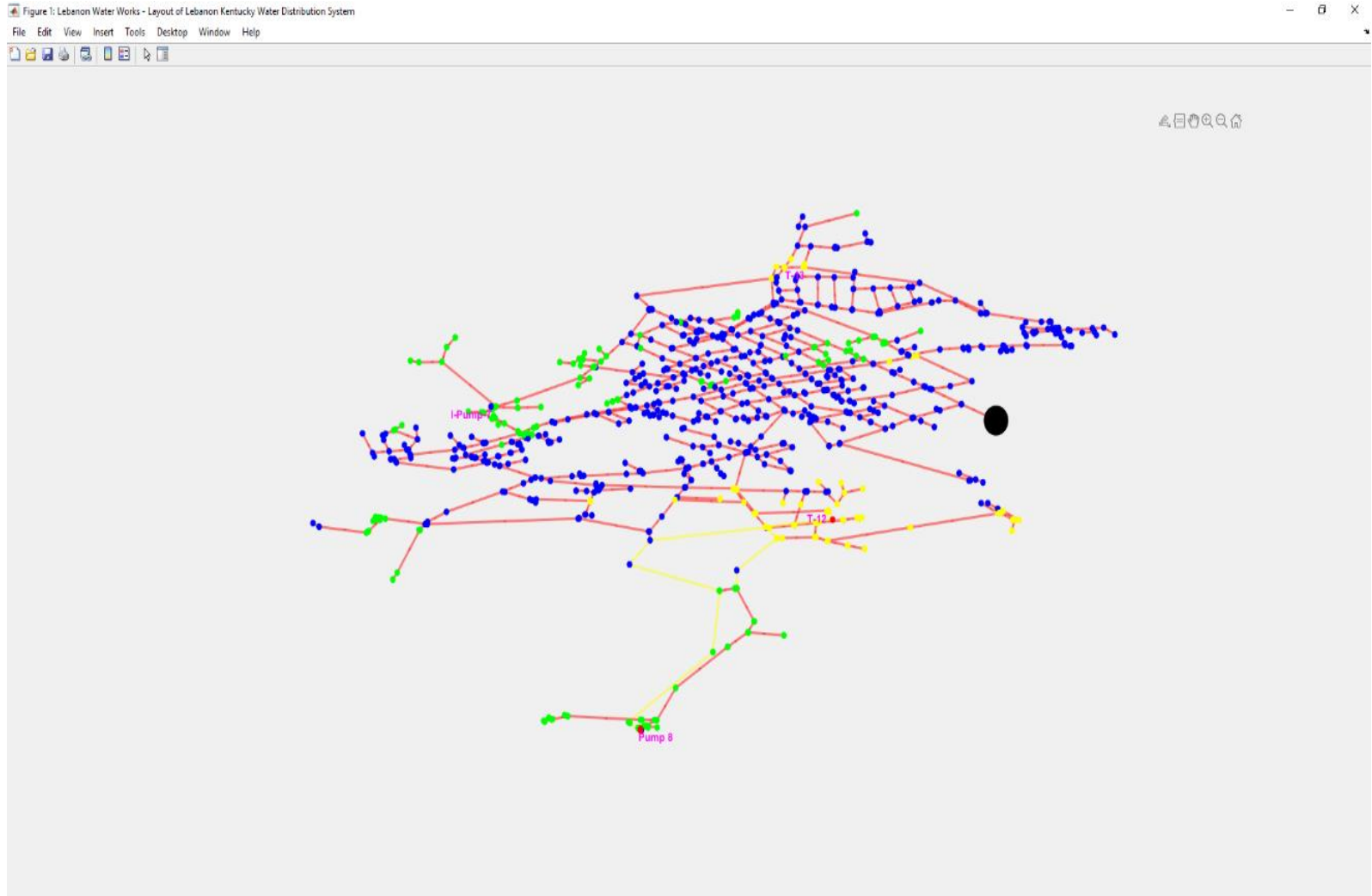


Figure 5-17: Map Results Detailing Pressure, Flows, and “Discovery” Results

CHAPTER 6. DISCUSSION OF RESULTS AND CONCLUSION

The main objective of this research was to investigate the feasibility of being able to create “digital twin” model for water distribution operators in small systems who lack the resources and time needed to purchase or create such applications themselves. In striving to meet these objectives, there were several lessons learned and relevant outcomes relating to the process.

The process of implementation proved to be significant in highlighting the most appropriate framework and methodologies for use in the development of digital twin applications for small utilities. This proved to be an iterative process and involving the testing of several interface configurations, optimization algorithms, and data collection strategies in order to determine the most feasible and effective strategy.

6.1 Interface Selection and the Underlying Hydraulic Model

It is the authors experience that a significant amount of time may be spent not only on the development and incorporation of traditional digital twin functionality, but also on troubleshooting the underlying hydraulic model data and working within the constraints of certain programming environments.

With regard to selecting proper programming environments, it is valuable to first take the suggestions and objectives of the respective water utility and map those concepts to available functionalities within each language. For example, when creating the digital twin for the Whitesburg Water System, a link between EPANET, Excel, and MATLAB was used and tested for suitability. The reason for connecting these three software packages was due to the assumed familiarity and comfortability operators may have with the Excel

interface while also exploiting the computational power of MATLAB for efficient hydraulic computation times.

While the EPANET-MATLAB-Excel platform proved to be capable of achieving the objectives of this work, other methods were found to be faster and offer user interface options more conducive to simple operator interaction. The interface detailed in this work is proof of that process; by using the EPANET-MATLAB toolkit in combination with the app developer toolkit available within MATLAB, the deliverables outlined in Chapter 1 were further optimized.

It is estimated however that a significant amount of time might have been saved throughout this work if a more robust investigation into other available programming languages had been continued prior to the eventual selection and development of the MATLAB environment. MATLAB was selected because of the authors familiarity with the language and the existence of an open-source toolkit capable of connecting the EPANET hydraulic engine to the MATLAB programming language. This was done because of time constraints which ultimately prevented a more robust investigation. In the end, it was concluded that Python offers a better platform than MATLAB for achieving this objective and future development work should consider using it versus MATLAB, especially when considering the development of digital twins for smaller systems.

In addition to proper software selection, the process of digital twin development would have been made more efficient if ample work was spent initially on validating system topologies within the system under question. This is a lesson that was carried on to the creation of the digital twin for Lebanon and is due in large part to the experiences working in the Whitesburg system. While working in Whitesburg, several months were

spent on attempting to calibrate their hydraulic model using a single days' worth of telemetry data. After repeatedly testing and retesting many algorithms for demand calibration and failing to get the model to agree with real world conditions, it was assumed that there was something wrong with the approach being taken.

Upon revisiting the original demand allocation algorithm developed for Whitesburg (which was based on the Box Complex Method), it was found that there seemed to be no issue at all with the algorithms themselves but was more likely due to underlying errors in the system topology and inherent noise in their telemetry data. Because of the impacts of the regional flood, there was never an opportunity to meet with the operators to resolve these issues. As a result, we pivoted to work with the Lebanon system, which has a much more reliable baseline hydraulic model and more much more reliable telemetry data. Therefore, the developed demand allocation model was able to produce reliable demand scenarios, after minor adjusted were made to noise issues in some of the tank telemetry data. This allowed for seamless integration of the bisection method into the digital twin framework and highlighted the fact that a critical first step in this process is ensuring that 1) the physical elements of our system (pipes, tanks, pumps, etc) are modeled to the best understanding of the utility and 2) telemetry data is equally as reflective of the conditions seen in the system.

This conclusion was subsequently validated by assuming the demands and the tank telemetry data in the Whitesburg model were actually known and then seeing if the Box-Complex model could recover them from the data assuming that the underlying model topology was correct. Thus, by explicitly specifying the system demands in the model, running EPANET to then produce the “actual” tank telemetry data, topological and

telemetry errors are controlled for. The Box-Complex method was then used to see if it could replicate these artificial demands which it was able to do effortlessly.

6.2 Data in the Hydraulic Calibration Process

In contrast to the previous section, while having too little system information is certainly an obstacle in creating a useful hydraulic model (and therefore a digital twin), too much information also creates a unique set of problems. For a digital twin to produce useful outputs from which an operator can make decisions, incoming data must be “cleaned” in a way that captures the true physical characteristics of the WDN in real time.

It was initially tempting to look at the available telemetry and meter data from the Lebanon Water Works (LWW) system, given that it was very abundant, and assume that this data would also be highly accurate. A cursory glance of the data set does in fact confirm this assumption; however, it ignores reality in that no matter how sophisticated the practices and equipment may be, errors attributed to the capture of real-world data seem to be unavoidable.

The “quirks” associated with the LWW system telemetry and meter data are a function of many factors which are made manifest in the data latching and noise characteristics described in section 4.2. Cleaning the data in order that they do not impact the resulting outputs, requires a strong understanding of 1) when the data might be in error 2) why the data is flawed so that future instances might be predicted (human, equipment, etc.), and 3) how to systematically remove these instances when running the demand calibration algorithms.

In the case of the LWW system, a simple change in the time used in analyzing the telemetry data resulted in demand factor errors only occurring 2% (improved from 21%) of the time. Further investigation into other error sources may improve this value, however that is likely unnecessary and will not significantly impact the resulting demand factors because of the procedure adapted in this research, namely, carrying the small residual demand error into the next time step.

It is in the authors experience that real world data in this context is as unique as any two individuals. From monitoring equipment to the entire SCADA system and all the way to how information is reported, data is characteristic of the system from which it was gathered. Because system data inherently has its own patterns and tendencies that cause it to vary from reality, a basic knowledge of each system is fundamental for creating the programs and algorithms that will help operators plan and manage their respective utilities.

6.3 Algorithms and the Hydraulic Calibration Process

The main complication in choosing and subsequently implementing the appropriate algorithm (in the context of demand calibration) was one of speed and robustness of a given solution. For example, the Box Complex and bisection methods were chosen due to their simplicity and because they both were able to calculate demand factors quickly and without error relative to the algorithms themselves.

Because the bisection method is most suited for accurately calculating two demand factors (section 2.1.4), the LWW system is a unique use case for this algorithm. Additionally, even though the Box Complex Method is extendable to systems of any size, increasing dimensionality may increase computational burden and impact the accuracy of

a given solution. This is due to the way the algorithm searches a solution space which can become localized. By increasing dimensionality, the solution space is inherently being increased and the chances that the algorithm will be able to capture an optimal solution can be somewhat reduced. Nonetheless, the algorithm is simple to program, generally robust, and should be considered for future applications to multi-tank systems.

Thus, as long as the physical characteristics of the system are properly reflected in the algorithm and the data is sufficiently cleaned, the resulting demand factors are expected to capture the real demands of the WDN in question. Given this foundation, the LWW digital twin for example, may employ its digital twin as it currently stands to evaluate real time telemetry and meter data for the creation of predictive demand factors.

6.4 General Conclusions

The final version of the digital twin for Lebanon, Kentucky does meet most of the intended objectives of this thesis. While the MATLAB GUI display may be further improved (e.g., using Python), the amount of time spent on hydraulic calibration allows for the useful prediction of pressure, demand, and chlorine concentration. The inputs and outputs are intentionally simple to leave the operators little room for confusion which are likely barriers to other software programs such as EPANET or KYPIPE.

Perhaps the most important finding of the research was that the success of digital twins for small systems is especially depending upon the availability of the utility staff to answer questions, access to reliable physical and operational data, and the existence of (or the ability to create) a calibrated hydraulic model of the network. Fortunately, all three

were available for the Lebanon system, and hence the measured success, while none of the three were initially available for the Whitesburg system.

The research was also able to show that realistic demand scenarios can be derived from historic tank telemetry data along with a calibrated network model by employing either a bisection or Box Complex algorithm. Both algorithms are thus available for applications to other systems.

Unfortunately, time and logistical constraints involving the partner utility (i.e., Lebanon) prevented a full testing of the Lebanon digital twin by the utility staff and operators. Consequently, completion of the second part of objective 4 (see section 1.4) will have to await subsequent future research.

A final benefit of this research is that the methodologies employed (as well as the demand calibration algorithms) are easily reproducible for other systems and may be extended to incorporate other features. By following the summary steps in table 6-1, it would not be unreasonable to expect the creation of a digital twin for a new relatively small system to take no more than a few weeks (given the readiness of the system for the incorporation of a digital twin).

Table 6-1: Summary of Digital Twin Creation Process

1) Create EPANET file (.inp) of system.
2) Perform macro level calibration (Ormsbee and Lingireddy, 1997).
3) Delineate pressure zones.
4) Adjust the "ExtendedPeriodV2" function within the MATLAB code to reflect new file (junction names, number of patterns, etc.)
5) Adjust the "RunEPSButtonPushed" function to reflect new file (new tanks, pumps, etc.)
6) Create a unique Box Complex or bisection method solver for the system using the example code in the appendices, I and then implement the resulting demand scenarios within the application.

In conclusion, adopting digital twin models for small systems does require significant knowledge of a particular system in combination with strong hydraulic modeling skills, accurate data (both topologic and telemetry) a willingness to work closely with utilities, a demand calibration algorithm, and the ability to translate all that information into programmable language. However, through the foundation created within this thesis, extension of the template employed in creating the Lebanon digital twin should prove to be a simple and cost-effective alternative to other custom digital twin applications currently available to distribution operators.

7. RECOMMENDATIONS

7.1 Engineering Significance

In most cases, smaller utilities in Appalachia lack the resources to create and maintain even a simple hydraulic model. While many utilities around the country are experiencing the effects of the current digital revolution, small systems in this region may come to rely on open-source applications like the digital twin presented in this research in order to stay competitive in the rapidly changing climate of water distribution system management.

Given that much of the national attention at the moment is focused on revamping deteriorating infrastructure throughout the country, developing tools like this for small utilities may prove to be a significant step in maximizing the benefits of many new assets that are expected to be implemented throughout the United States. Small utilities in Appalachia are likely to benefit from continued research into digital twin solutions due to the low number of operators available to manage such systems and to meet the needs of their communities.

Additionally, this work provides an initial investigation into some of the common pitfalls of employing digital twin solutions for water distribution systems. Lessons learned throughout this process can facilitate the potential expedient employment of digital twins for small water systems while also ensuring that the process of creating the application result in an accurate and robust application for distribution operators.

7.2 Limitations of Approach

While creating the platform through which data is effectively expressed is a relatively straight forward process and depends only on the needs of a utility; cleaning and incorporating that data may pose a challenge.

The frequency of data, noise, and where that data is being stored are just a few factors that complicate the effective use of such system information. It takes a large amount of time to begin to understand the unique characteristics of each system including the performance and reliability of individual sensors. This may impact the speed of implementation for these types of tools.

7.3 Need for Future Research

While the current foundation laid within this report provides a strong base upon which other digital twins may be developed, improvements may need to be made to improve the rapidity in which the application is implemented for each individual system under this framework. As the capabilities of artificial intelligence and machine learning continue to grow at unprecedented rates, it is not unreasonable to envision these types of technologies having a significant role in developing digital twins and smoothing out the data noise problem as well as the implementation process.

For example, large language modelling (LLM) (i.e., similar to what is found in ChatGPT) may be incorporated and trained on a wide range of relevant information from operator experiences around the country. One can imagine an application that functions in the same way that J.A.R.V.I.S. does in Marvels' Iron Man series. Through continuous training and integration of both text based and quantitative data, a LLM could be trained

to quickly produce its own hydraulic models which could then be used by an operator. For example, if a future weather forecast included such information such as “there is rain in the forecast and the parade will take place on main street around 3pm” a LLM like this could produce demand scenarios that would have occurred in the past under different scenarios. Simply put, this would be a model of models, the ultimate support tool for operators.

Further research on the simple automation of demand forecasting might also help support the implementation of digital twin models such as this one. While historical data will undoubtedly aid in prediction of daily operations, investigating work done with time series modelling as well as several machine learning algorithms could be useful in implementing a more reliable forecasting strategy for this digital twin.

7.4 Recommendations

The continued improvement of this digital twin for the LWW system will rely upon further integration and cleaning of data to ensure its accuracy and precision. Additionally, this is not yet a fully evolved digital twin. Live data stream integration and predictive modelling should be included in future versions of this digital twin to approach the quality of other services that are currently available on the market but are typically out of reach for smaller utilities.

It is also recommended that future applications consider the use of Python programming instead of MATLAB. Python is a high-level programming language like MATLAB and has a minimal learning curve. Python also offers a larger library of resources for development of features like interactive mapping relative to MATLAB.

Once this digital twin format has been translated into Python, functionality should be created for simple element additions via the aforementioned interactive mapping features. Explicit demand forecasting should also be included in addition to the historical demand information currently populating the model. Packages within Python support simple integration of forecasting tools and would be of great use for creating demand scenarios more reflective of real system information.

While the current model GUI has been developed to accommodate water quality parameters such as chlorine and TTHM prediction, the associated functional models to drive these predictions have not yet been calibrated and linked with the EPANET and GUI interface. Once the final water quality models have been calibrated, this linkage should be completed.

Lastly, the model should continue to be tested and integrated within the LWW daily operations to validate the usefulness of the digital twin application. It is likely that in utilizing the digital twin, operators will offer valuable feedback which will greatly improve the usefulness of the model. It is therefore a final recommendation that the model be occasionally updated to reflect the changing needs of the utility it is supporting.

APPENDICES

APPENDIX A. Demand Factors From Bisection Algorithm

Table A 1: Uncleaned Demand Factors for Zone 1 of LWW

	Date	20-Jun	21-Jun	22-Jun	23-Jun	24-Jun	25-Jun	26-Jun	27-Jun	28-Jun	29-Jun	30-Jun	1-Jul	2-Jul	3-Jul
Hour															
1		0.295	0.255	0.423	0.403	0.001	0.614	0.634	0.687	0.256	0.678	0.220	2.029	0.133	0.435
2		0.492	0.298	0.577	0.291	0.001	0.001	0.001	0.082	0.405	0.984	0.297	1.432	0.001	0.474
3		0.151	0.282	0.037	0.136	0.107	-8.197	0.001	0.001	0.042	0.480	0.204	1.308	0.001	0.249
4		0.001	0.430	0.001	0.156	0.001	-0.845	0.001	0.001	0.001	0.087	0.001	1.288	0.001	0.001
5		0.001	0.001	0.001	0.429	0.001	-0.198	0.001	0.001	0.001	0.316	0.001	1.062	0.001	0.001
6		0.075	0.001	0.001	0.136	0.001	-0.512	0.001	0.001	0.001	0.556	0.001	1.060	0.001	0.001
7		1.184	1.571	1.741	1.685	0.987	-0.460	0.001	-7.725	0.001	0.883	0.069	1.149	0.001	0.001
8		2.334	2.246	1.223	2.113	1.724	17.095	0.450	0.377	0.482	0.378	0.789	1.957	-14.972	1.442
9		2.495	2.685	0.580	2.962	2.629	1.780	0.138	16.981	1.016	1.301	0.663	2.345	16.743	2.875
10		2.714	2.959	0.540	3.615	2.736	2.387	0.907	2.935	0.700	2.857	0.502	2.284	0.620	3.243
11		3.514	3.285	0.819	2.953	2.298	2.150	1.493	2.929	1.583	1.501	0.796	2.430	1.002	3.209
12		3.294	2.545	0.750	5.433	2.312	1.960	1.419	2.068	1.500	0.955	0.332	2.081	2.632	4.715
13		3.397	2.712	1.878	3.194	2.914	1.764	1.417	2.067	1.369	1.097	0.459	2.807	2.430	3.629
14		2.701	3.174	0.759	2.854	2.662	1.559	2.147	2.039	1.969	1.693	0.324	2.439	2.440	3.898
15		2.654	2.987	1.317	1.133	1.008	1.220	1.376	1.662	3.398	1.364	0.316	1.974	2.004	2.832
16		2.820	2.677	2.984	1.227	1.183	1.391	1.520	1.377	2.975	1.482	0.644	2.535	2.094	2.877
17		3.125	2.630	2.762	1.591	1.037	1.407	1.297	1.923	2.894	1.382	2.538	2.461	2.229	3.464
18		2.751	2.583	2.896	1.544	0.636	1.278	1.797	1.587	1.567	1.955	2.750	2.435	2.243	1.868
19		2.884	2.723	0.484	1.545	1.041	1.156	2.652	1.487	1.793	1.697	2.471	1.706	1.626	2.623
20		2.506	1.667	1.676	1.852	1.129	1.061	2.621	1.484	2.065	1.066	2.713	2.463	1.311	2.499
21		1.683	2.949	0.001	1.745	1.994	0.001	1.897	1.253	3.221	0.001	2.772	2.008	1.206	2.282
22		2.869	1.977	0.738	1.775	1.076	0.126	1.631	1.566	2.911	1.138	2.079	0.001	0.783	1.963
23		1.063	0.298	1.429	1.275	0.531	0.909	0.949	1.036	0.623	1.278	1.741	-7.637	0.782	2.572
24		0.211	0.586	0.281	0.388	0.344	0.712	0.562	0.295	-7.350	0.312	0.982	-0.120	0.508	1.012

Table A 2: Uncleaned Demand Factors for Zone 2 of LWW

	Date	20-Jun	21-Jun	22-Jun	23-Jun	24-Jun	25-Jun	26-Jun	27-Jun	28-Jun	29-Jun	30-Jun	1-Jul	2-Jul	3-Jul
Hour															
1		1.149	1.321	1.190	1.570	0.459	0.412	0.463	0.589	1.229	1.485	1.138	0.419	0.226	0.839
2		0.691	0.838	0.495	0.994	0.816	-0.732	0.053	0.773	0.249	0.145	0.652	0.001	-0.267	0.240
3		0.150	0.902	1.169	0.756	0.388	-17.967	0.329	0.626	0.001	0.001	0.001	0.001	-0.198	0.001
4		0.233	0.218	0.476	1.470	-0.729	0.001	0.218	0.452	-0.148	0.315	-0.068	0.404	-0.740	0.039
5		0.828	0.438	0.793	0.427	0.162	0.001	0.915	0.669	0.747	0.889	0.161	0.457	-0.549	0.490
6		1.325	1.331	1.506	0.178	0.401	0.001	0.562	-0.377	0.409	0.833	1.026	0.593	-0.377	0.714
7		0.977	1.741	1.734	1.336	1.629	0.001	2.131	-16.933	1.491	0.879	1.431	0.582	1.428	0.933
8		0.434	1.379	0.751	1.181	0.785	2.250	1.440	0.001	1.445	1.376	0.970	0.001	0.001	1.544
9		0.672	1.058	0.666	1.311	0.284	0.598	2.912	4.802	0.838	1.951	0.761	0.001	1.766	0.449
10		1.743	0.797	1.674	0.783	0.001	0.001	2.554	0.952	1.050	1.025	1.244	0.001	0.314	0.057
11		0.001	0.001	1.814	0.347	0.772	0.038	1.281	0.429	0.721	0.471	1.082	0.655	0.794	1.420
12		0.995	0.525	0.694	0.001	1.111	0.703	1.217	1.511	1.286	0.758	0.381	0.795	0.046	1.583
13		0.736	1.443	0.203	0.001	0.692	0.062	1.926	0.890	0.556	1.491	0.644	0.001	0.001	1.096
14		0.343	0.465	2.350	0.001	0.746	0.001	0.721	0.878	1.688	0.946	0.001	0.001	0.001	0.312
15		1.592	0.408	0.963	0.166	0.072	0.618	0.924	0.343	1.299	0.200	1.011	0.490	0.912	0.150
16		1.310	1.087	0.212	1.094	0.371	0.704	0.477	2.064	0.966	0.674	1.325	1.282	0.752	0.001
17		0.222	0.544	0.122	0.499	0.113	0.320	0.948	0.646	0.718	1.642	0.422	0.043	0.001	0.016
18		0.350	0.001	0.001	0.167	0.730	0.001	1.856	0.763	0.675	0.404	0.001	0.001	0.001	1.044
19		0.001	0.218	2.508	0.139	1.560	0.710	1.446	0.534	0.816	0.532	0.460	0.819	0.700	0.410
20		0.319	2.050	1.551	0.132	1.296	1.490	0.488	0.568	2.020	1.140	1.201	0.717	0.698	0.132
21		1.129	0.731	6.989	1.496	1.061	4.627	0.725	1.760	1.156	4.583	0.197	0.018	0.325	0.471
22		0.392	0.449	1.617	1.191	0.290	2.640	0.371	1.051	0.722	2.493	0.844	1.762	1.612	1.097
23		0.912	1.664	0.001	0.783	0.698	1.650	0.878	0.821	38.424	0.188	1.326	-16.739	0.837	0.001
24		0.530	1.325	0.685	0.893	0.785	1.575	1.274	0.710	-16.442	0.748	2.185	1.005	1.124	2.265

Table A 3: Cleaned Demand Factors for Zone 1 of LWW

	Date	20-Jun	21-Jun	22-Jun	23-Jun	24-Jun	25-Jun	26-Jun	27-Jun	28-Jun	29-Jun	30-Jun	1-Jul	2-Jul	3-Jul
Hour															
1		0.295	0.255	0.423	0.403	0.001	0.614	0.634	0.687	0.256	0.678	0.220	2.029	0.133	0.435
2		0.492	0.298	0.577	0.291	0.001	0.001	0.001	0.082	0.405	0.984	0.297	1.432	0.001	0.474
3		0.151	0.282	0.037	0.136	0.107	0.001	0.001	0.001	0.042	0.480	0.204	1.308	0.001	0.249
4		0.001	0.430	0.001	0.156	0.001	0.001	0.001	0.001	0.001	0.087	0.001	1.288	0.001	0.001
5		0.001	0.001	0.001	0.429	0.001	0.001	0.001	0.001	0.001	0.316	0.001	1.062	0.001	0.001
6		0.075	0.001	0.001	0.136	0.001	0.001	0.001	0.001	0.001	0.556	0.001	1.060	0.001	0.001
7		1.184	1.571	1.741	1.685	0.987	0.001	0.001	0.001	0.001	0.883	0.069	1.149	0.001	0.001
8		2.334	2.246	1.223	2.113	1.724	0.891	0.450	0.377	0.482	0.378	0.789	1.957	0.001	1.442
9		2.495	2.685	0.580	2.962	2.629	1.780	0.138	1.656	1.016	1.301	0.663	2.345	0.311	2.875
10		2.714	2.959	0.540	3.615	2.736	2.387	0.907	2.935	0.700	2.857	0.502	2.284	0.620	3.243
11		3.514	3.285	0.819	2.953	2.298	2.150	1.493	2.929	1.583	1.501	0.796	2.430	1.002	3.209
12		3.294	2.545	0.750	3.073	2.312	1.960	1.419	2.068	1.500	0.955	0.332	2.081	2.632	4.715
13		3.397	2.712	1.878	3.194	2.914	1.764	1.417	2.067	1.369	1.097	0.459	2.807	2.430	3.629
14		2.701	3.174	0.759	2.854	2.662	1.559	2.147	2.039	1.969	1.693	0.324	2.439	2.440	3.898
15		2.654	2.987	1.317	1.133	1.008	1.220	1.376	1.662	3.398	1.364	0.316	1.974	2.004	2.832
16		2.820	2.677	2.984	1.227	1.183	1.391	1.520	1.377	2.975	1.482	0.644	2.535	2.094	2.877
17		3.125	2.630	2.762	1.591	1.037	1.407	1.297	1.923	2.894	1.382	2.538	2.461	2.229	3.464
18		2.751	2.583	2.896	1.544	0.636	1.278	1.797	1.587	1.567	1.955	2.750	2.435	2.243	1.868
19		2.884	2.723	0.484	1.545	1.041	1.156	2.652	1.487	1.793	1.697	2.471	1.706	1.626	2.623
20		2.506	1.667	1.676	1.852	1.129	1.061	2.621	1.484	2.065	1.066	2.713	2.463	1.311	2.499
21		1.683	2.949	0.001	1.745	1.994	0.001	1.897	1.253	3.221	0.001	2.772	2.008	1.206	2.282
22		2.869	1.977	0.738	1.775	1.076	1.126	1.631	1.566	2.911	1.138	2.079	0.001	0.783	1.963
23		1.063	0.298	1.429	1.275	0.531	0.909	0.949	1.036	0.623	1.278	1.741	0.001	0.782	2.572
24		0.211	0.586	0.281	0.388	0.344	0.712	0.562	0.295	0.001	0.312	0.982	0.001	0.508	1.012

Table A 4: Cleaned Demand Factors for Zone 2 of LWW

	Date	20-Jun	21-Jun	22-Jun	23-Jun	24-Jun	25-Jun	26-Jun	27-Jun	28-Jun	29-Jun	30-Jun	1-Jul	2-Jul	3-Jul
Hour															
1		1.149	1.321	1.190	1.570	0.459	0.412	0.463	0.589	1.229	1.485	1.138	0.419	0.226	0.839
2		0.691	0.838	0.495	0.994	0.816	0.001	0.053	0.773	0.249	1.145	0.652	0.001	0.001	0.240
3		0.150	0.902	1.169	0.756	0.388	0.001	0.329	0.626	0.001	0.001	0.001	0.001	0.001	0.001
4		0.233	0.218	0.476	1.470	0.275	0.001	0.218	0.452	0.001	0.315	0.001	0.404	0.001	0.039
5		0.828	0.438	0.793	0.427	0.162	0.001	0.915	0.669	0.747	0.889	0.161	0.457	0.001	0.490
6		1.325	1.331	1.506	0.178	0.401	0.001	0.562	0.001	0.409	0.833	1.026	0.593	0.001	0.714
7		0.977	1.741	1.734	1.336	1.629	0.001	2.131	0.001	1.491	0.879	1.431	0.582	1.428	0.933
8		0.434	1.379	0.751	1.181	0.785	2.250	1.440	0.001	1.445	1.376	0.970	0.001	0.001	1.544
9		0.672	1.058	0.666	1.311	0.284	0.598	2.912	0.477	0.838	1.951	0.761	0.001	1.766	0.449
10		1.743	0.797	1.674	0.783	0.001	0.001	2.554	0.952	1.050	1.025	1.244	0.001	0.314	0.057
11		0.001	0.001	1.814	0.347	0.772	0.038	1.281	0.429	0.721	1.082	1.082	0.655	0.794	1.420
12		0.995	0.525	0.694	0.001	1.111	0.703	1.217	1.511	1.286	0.758	0.381	0.795	0.046	1.583
13		0.736	1.443	0.203	0.001	0.692	0.062	1.926	0.890	0.556	1.491	0.644	0.001	0.001	1.096
14		0.343	0.465	2.350	0.001	0.746	0.001	0.721	0.878	1.688	0.946	0.001	0.001	0.001	0.312
15		1.592	0.408	0.963	0.166	0.072	0.618	0.924	0.343	1.299	0.200	1.011	0.490	0.912	0.150
16		1.310	1.087	0.212	1.094	0.371	0.704	0.477	2.064	0.966	0.674	1.325	1.282	0.752	0.001
17		0.222	0.544	0.122	0.499	0.113	0.320	0.948	0.646	0.718	1.642	0.422	0.043	0.001	0.016
18		0.350	0.001	0.001	0.167	0.730	0.001	1.856	0.763	0.675	0.404	0.001	0.001	0.001	1.044
19		0.001	0.218	2.508	0.139	1.560	0.710	1.446	0.534	0.816	0.532	0.460	0.819	0.700	0.410
20		0.319	2.050	1.551	0.132	1.296	1.490	0.488	0.568	2.020	1.140	1.201	0.717	0.698	0.132
21		1.129	0.731	1.584	1.496	1.061	2.065	0.725	1.760	1.156	4.583	0.197	0.018	0.325	0.471
22		0.392	0.449	1.617	1.191	0.290	2.640	0.371	1.051	0.722	2.493	0.844	1.762	1.612	1.097
23		0.912	1.664	0.001	0.783	0.698	1.650	0.878	0.821	0.362	0.188	1.326	0.001	0.837	0.001
24		0.530	1.325	0.685	0.893	0.785	1.575	1.274	0.710	0.001	0.748	2.185	1.005	1.124	2.265

Figure A 1: Uncleaned Demand Factors for Zone 1 of LWW (6/20/2023)

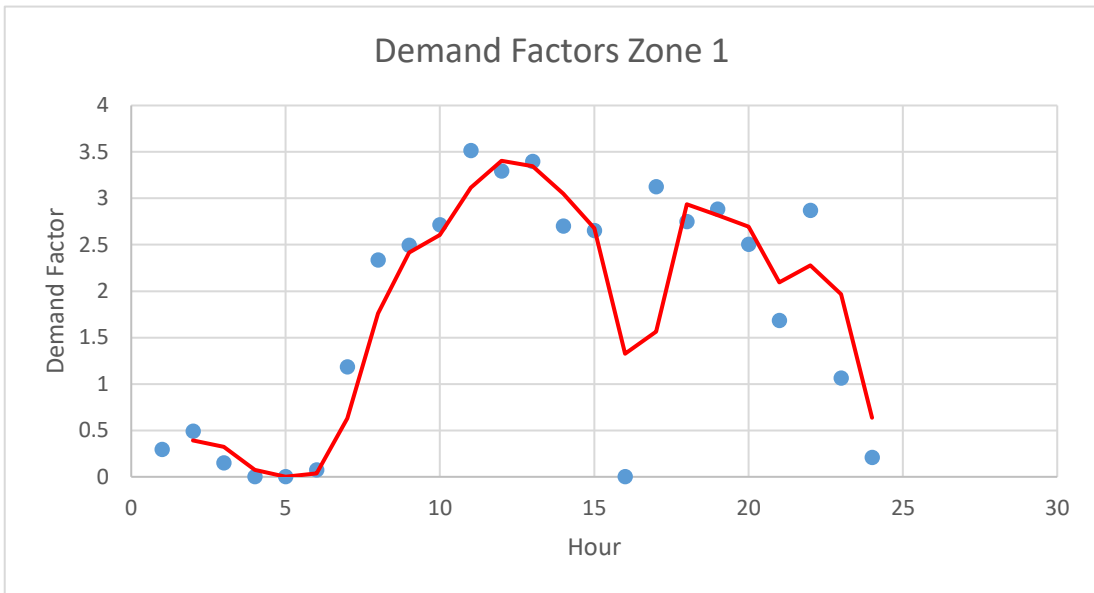


Figure A 2: Uncleaned Demand Factors for Zone 2 of LWW (6/20/2023)

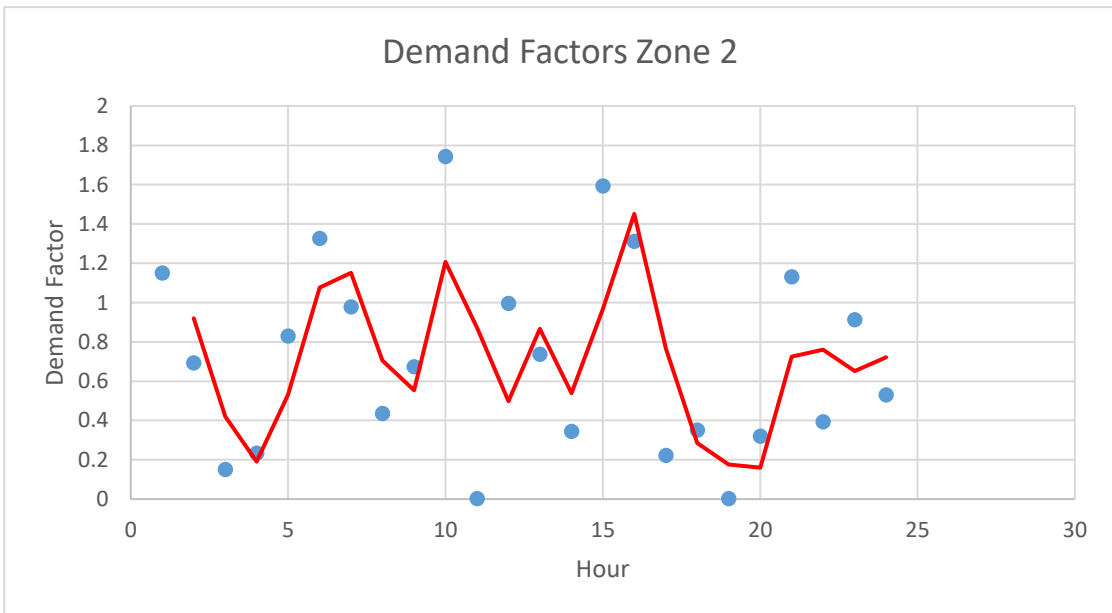


Figure A 3: Uncleaned Demand Factors for Zone 1 of LWW (6/21/2023)

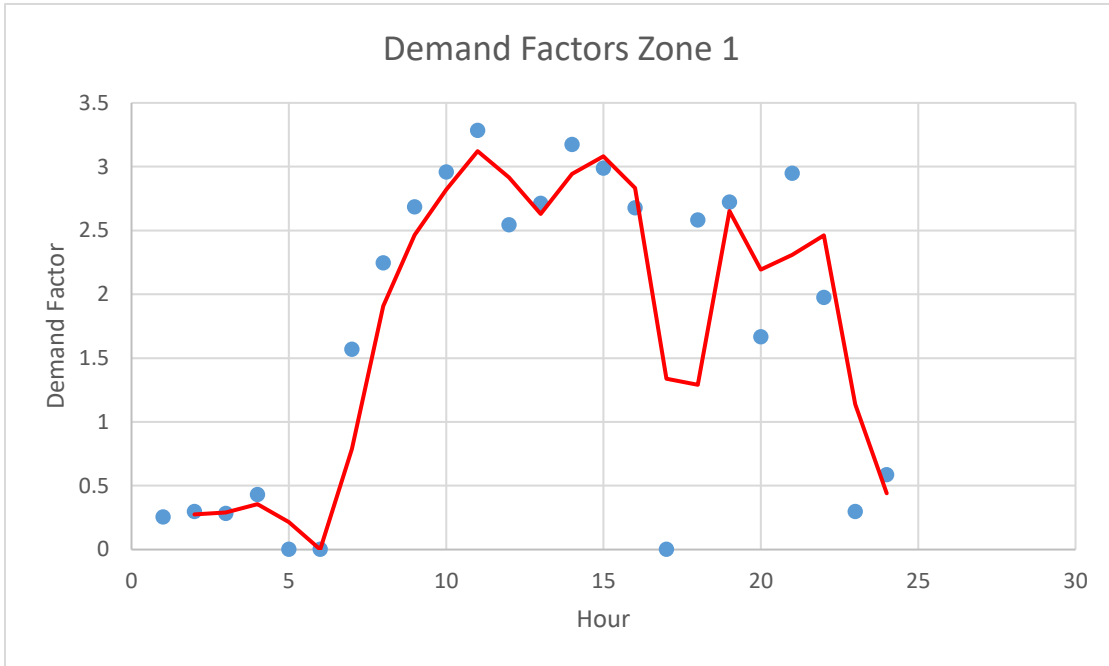


Figure A 4: Uncleaned Demand Factors for Zone 2 of LWW (6/21/2023)

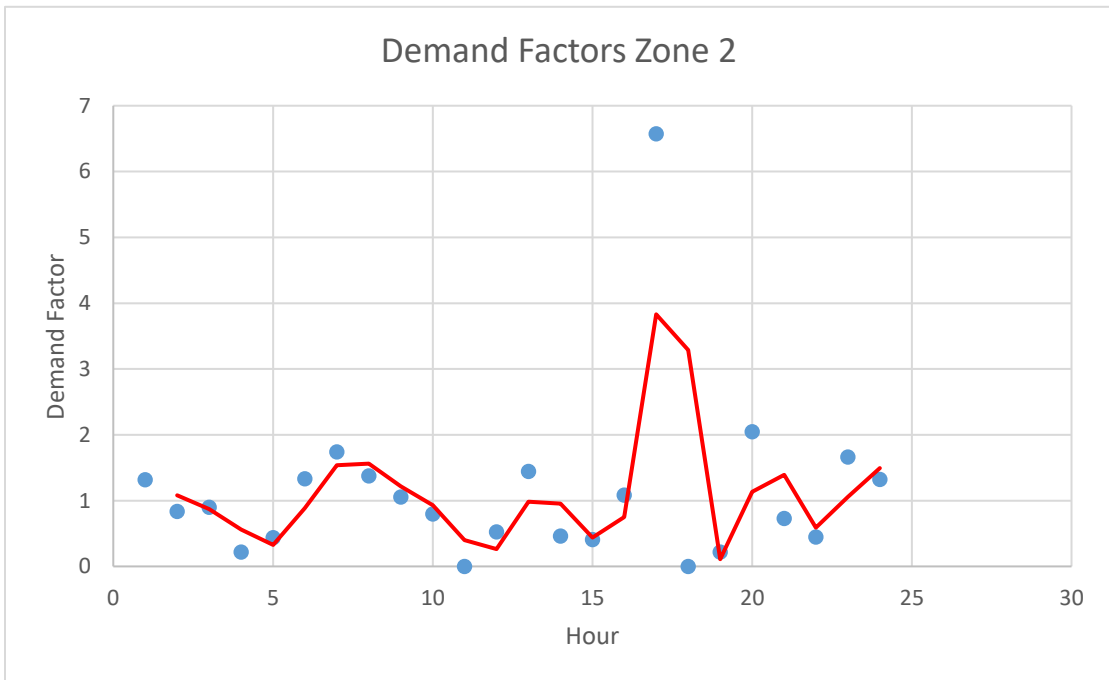


Figure A 5: Uncleaned Demand Factors for Zone 1 of LWW (6/22/2023)

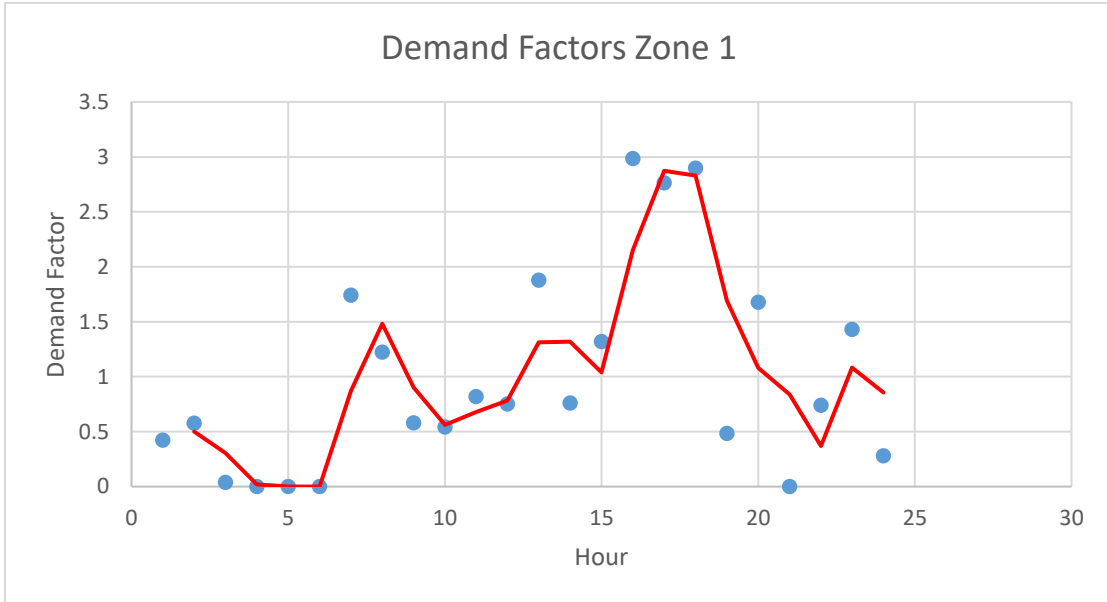


Figure A 6: Uncleaned Demand Factors for Zone 2 of LWW (6/22/2023)

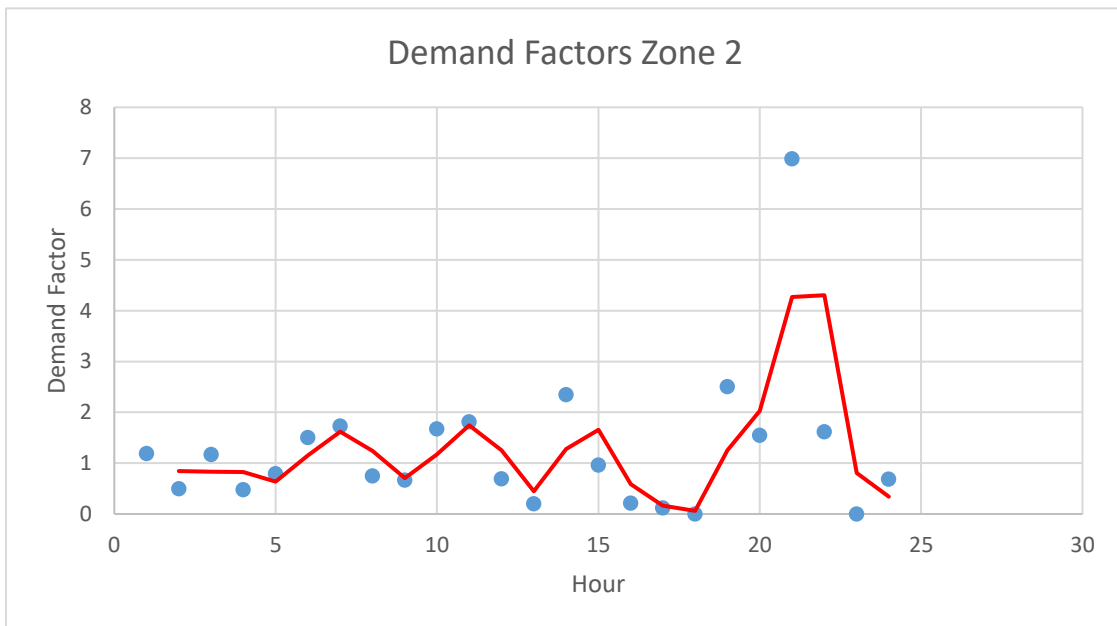


Figure A 7: Uncleaned Demand Factors for Zone 1 of LWW (6/23/2023)

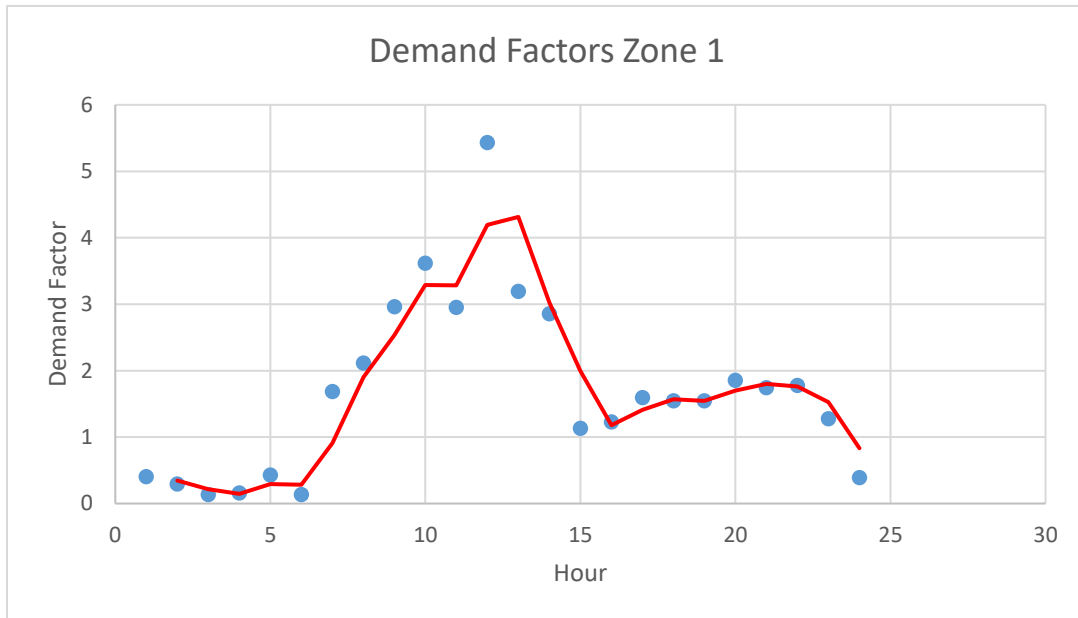


Figure A 8: Uncleaned Demand Factors for Zone 2 of LWW (6/23/2023)

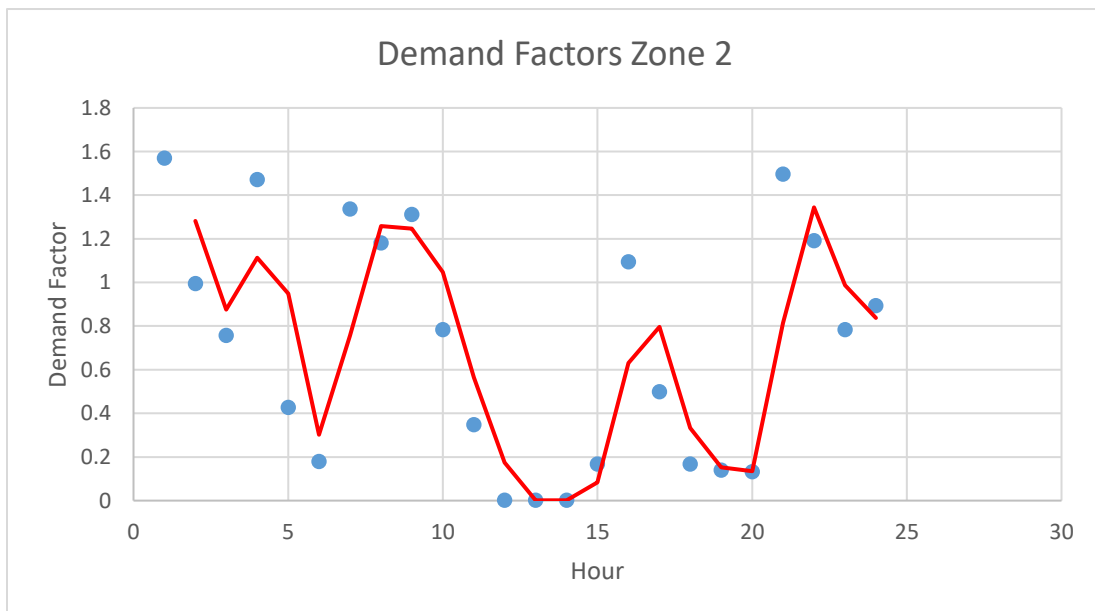


Figure A 9: Uncleaned Demand Factors for Zone 1 of LWW (6/24/2023)

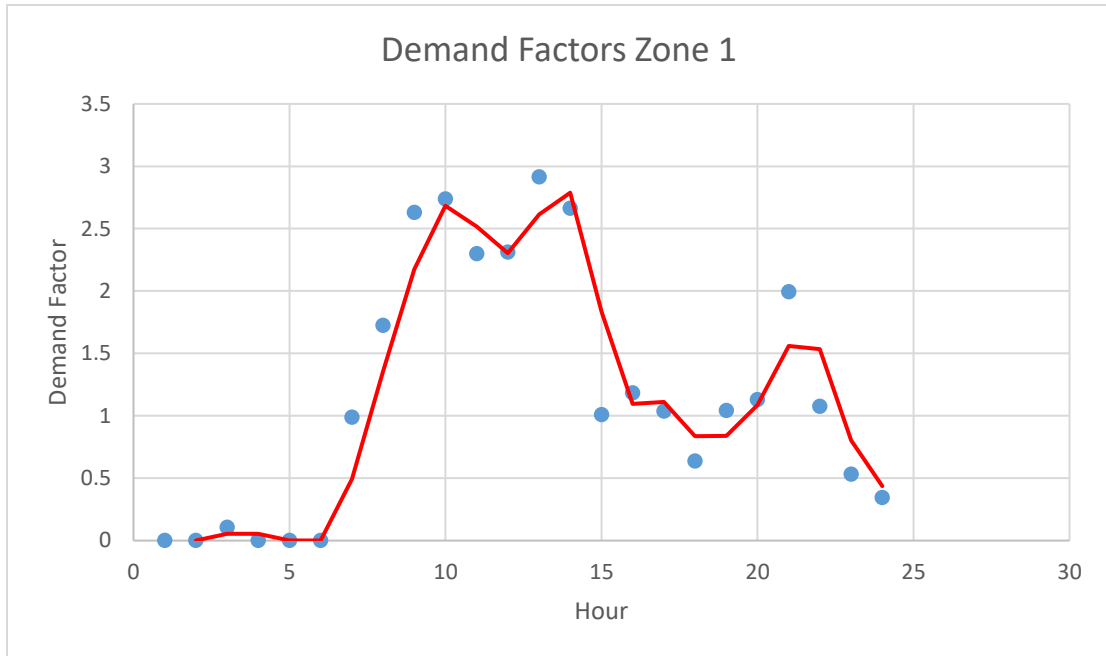


Figure A 10: Uncleaned Demand Factors for Zone 2 of LWW (6/24/2023)

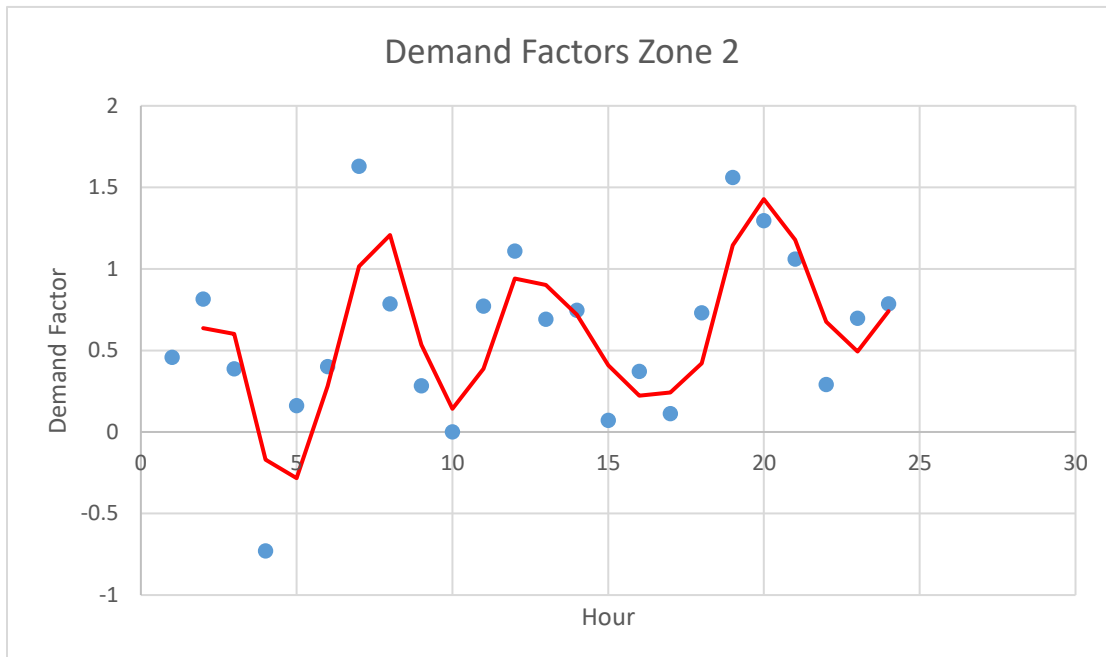


Figure A 11: Uncleaned Demand Factors for Zone 1 of LWW (6/25/2023)

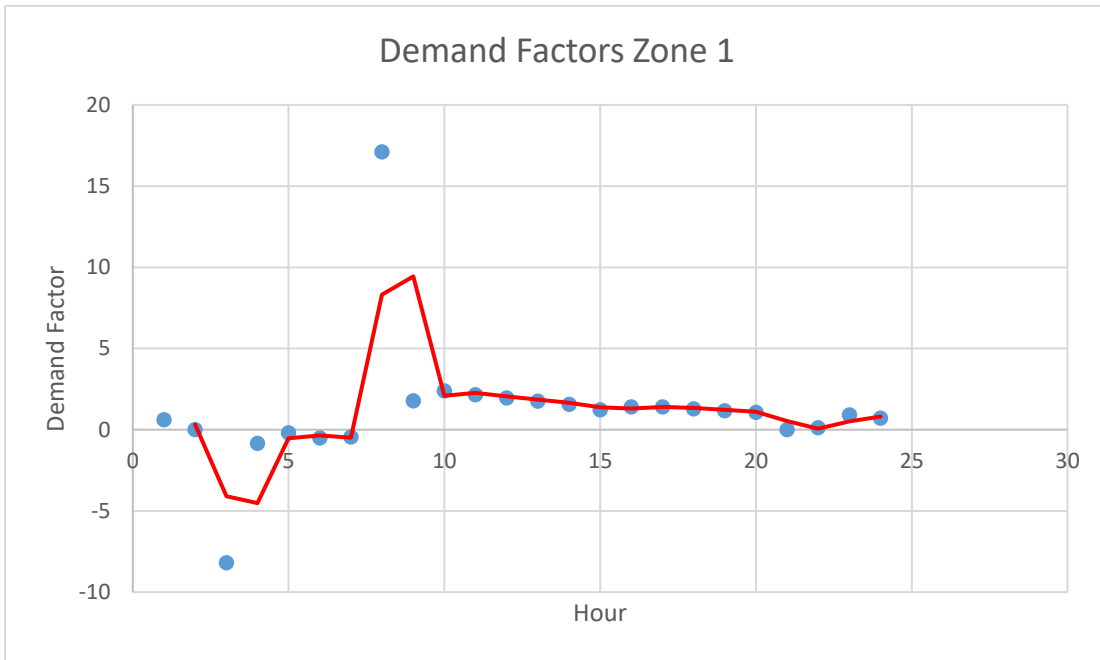


Figure A 12: Uncleaned Demand Factors for Zone 2 of LWW (6/25/2023)

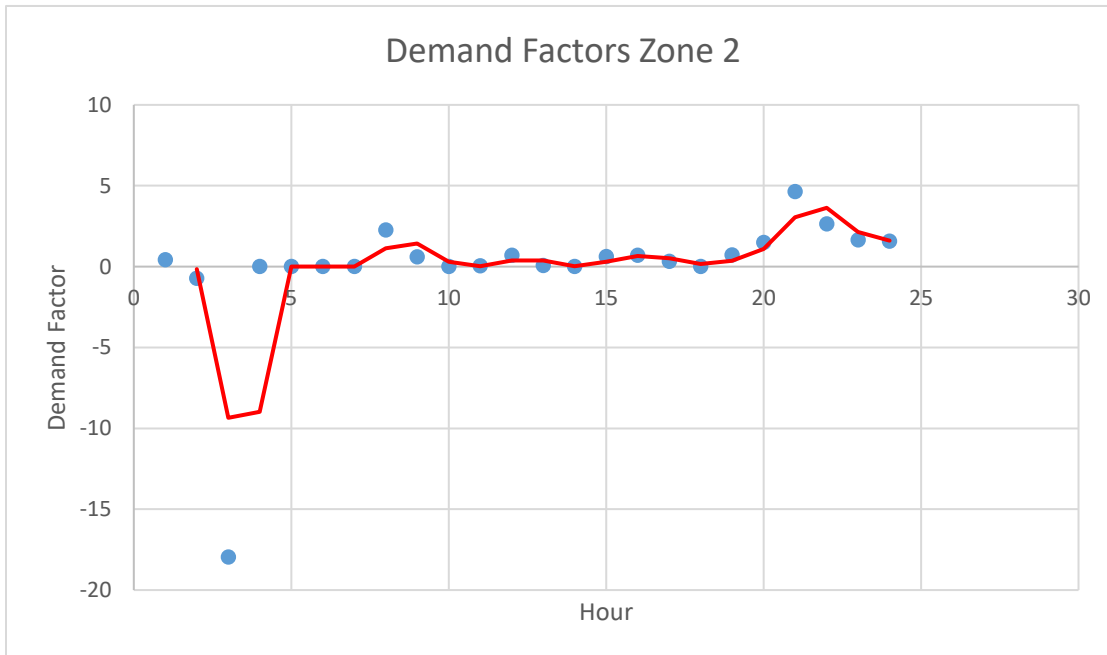


Figure A 13: Uncleaned Demand Factors for Zone 1 of LWW (6/26/2023)

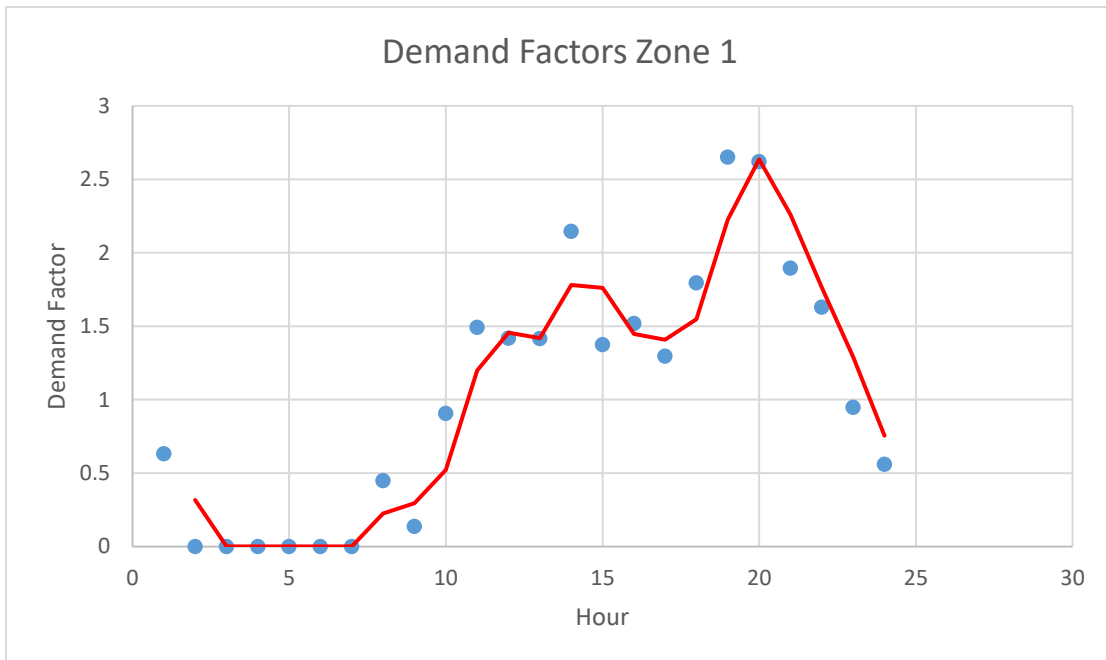


Figure A 14: Uncleaned Demand Factors for Zone 2 of LWW (6/26/2023)

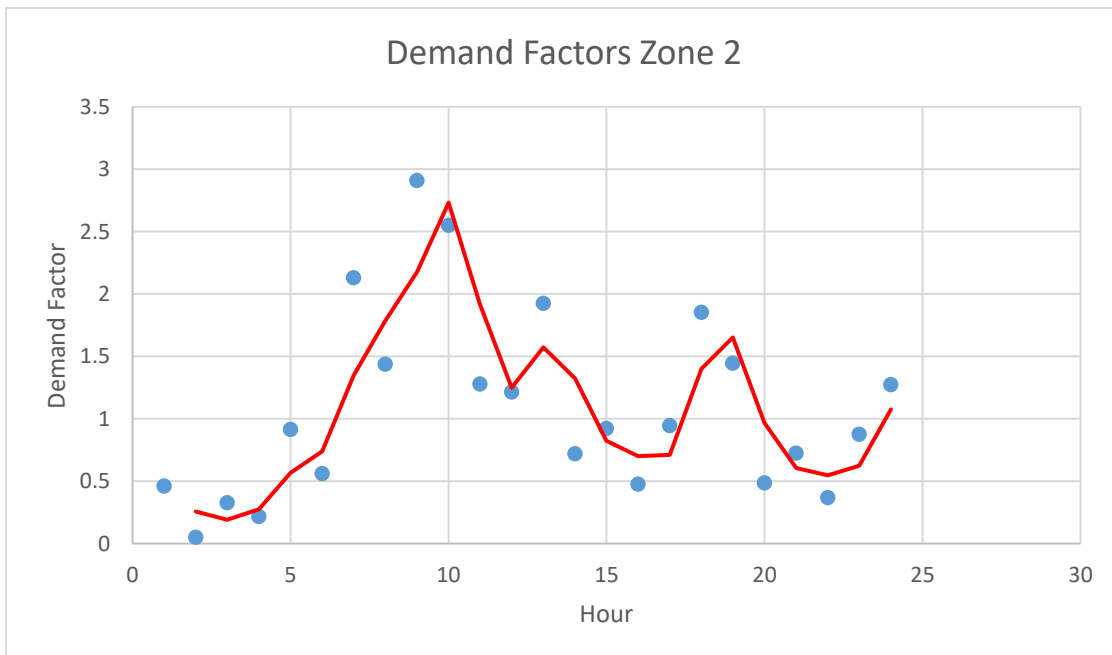


Figure A 15: Uncleaned Demand Factors for Zone 1 of LWW (6/27/2023)

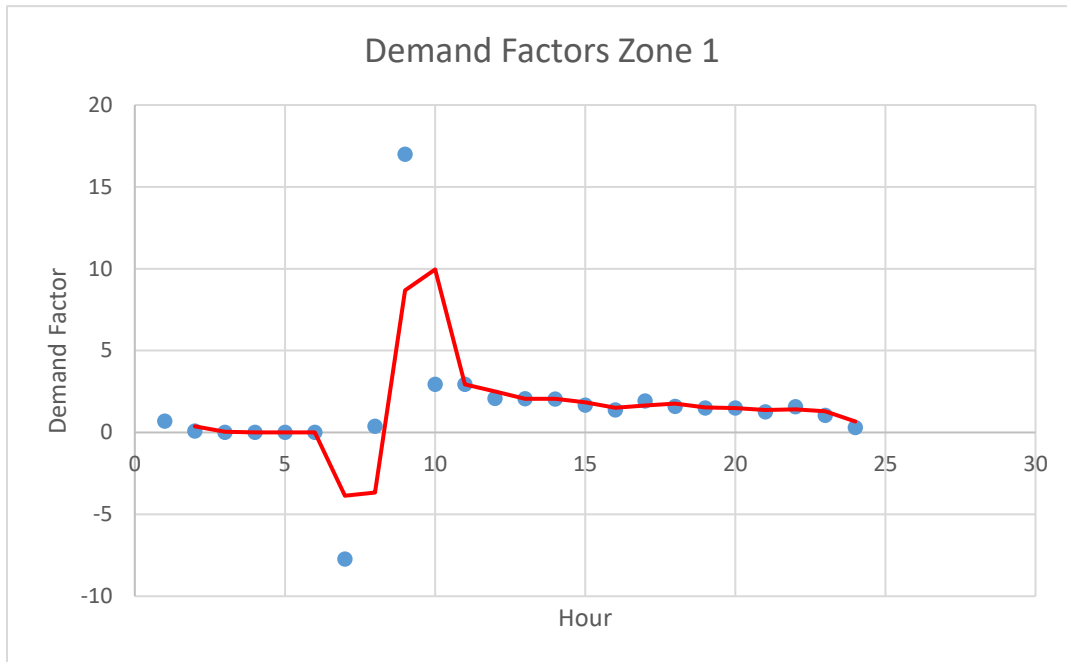


Figure A 16: Uncleaned Demand Factors for Zone 2 of LWW (6/27/2023)

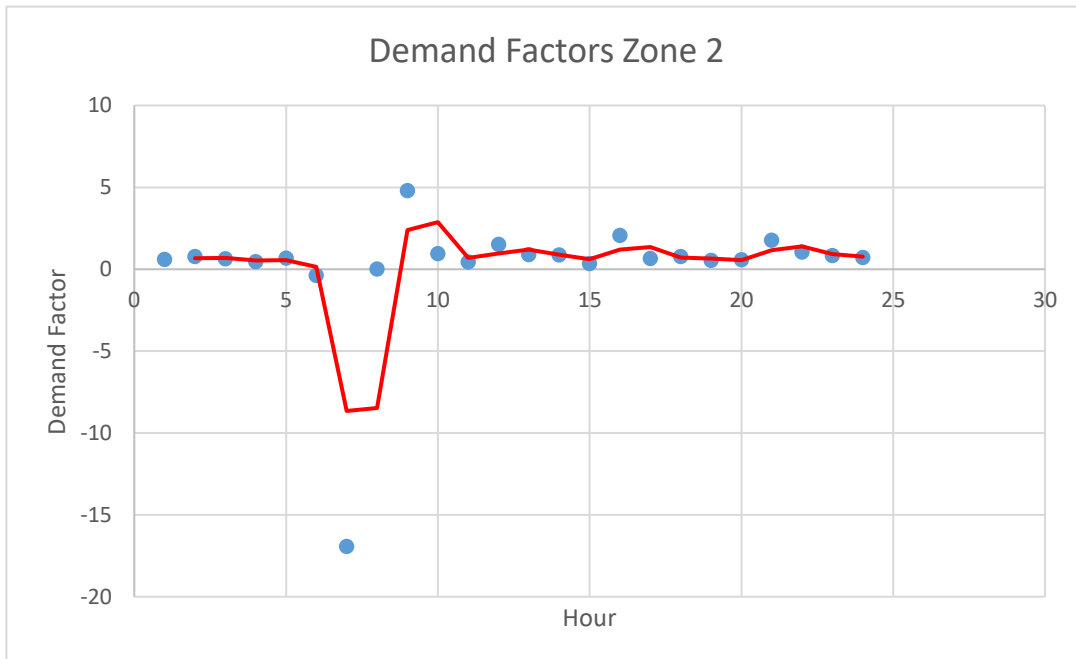


Figure A 17: Uncleaned Demand Factors for Zone 1 of LWW (6/28/2023)

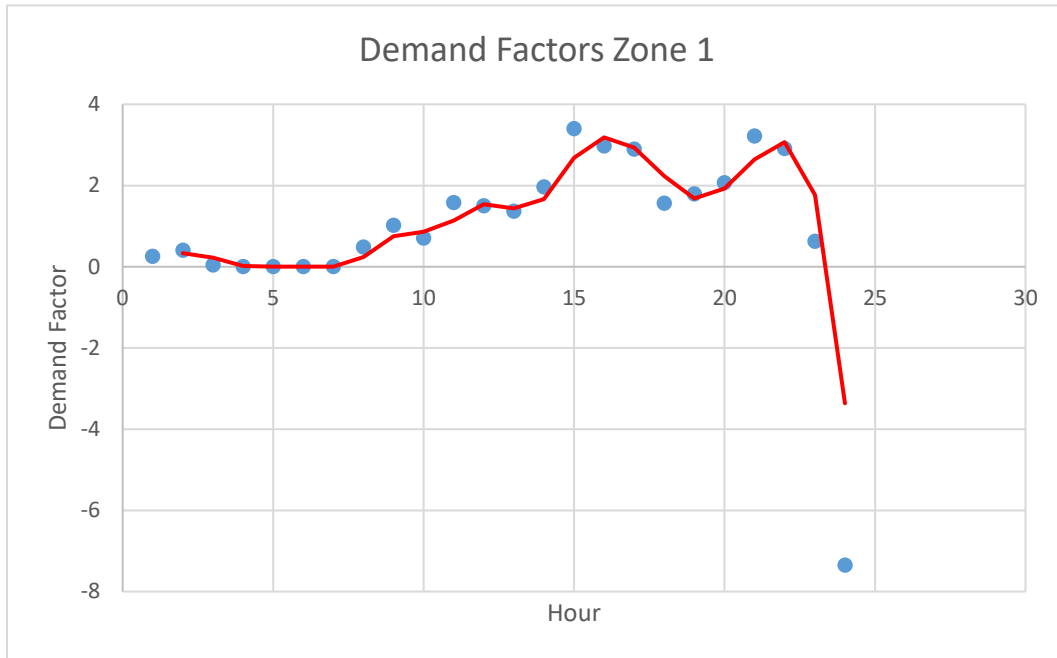


Figure A 18: Uncleaned Demand Factors for Zone 2 of LWW (6/28/2023)

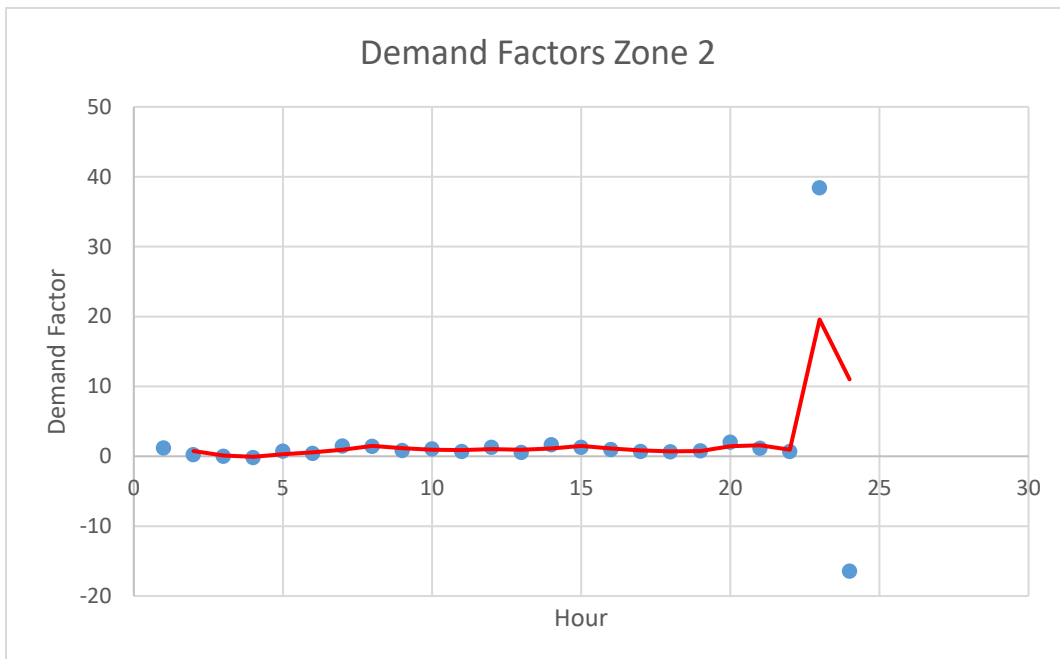


Figure A 19: Uncleaned Demand Factors for Zone 1 of LWW (6/29/2023)

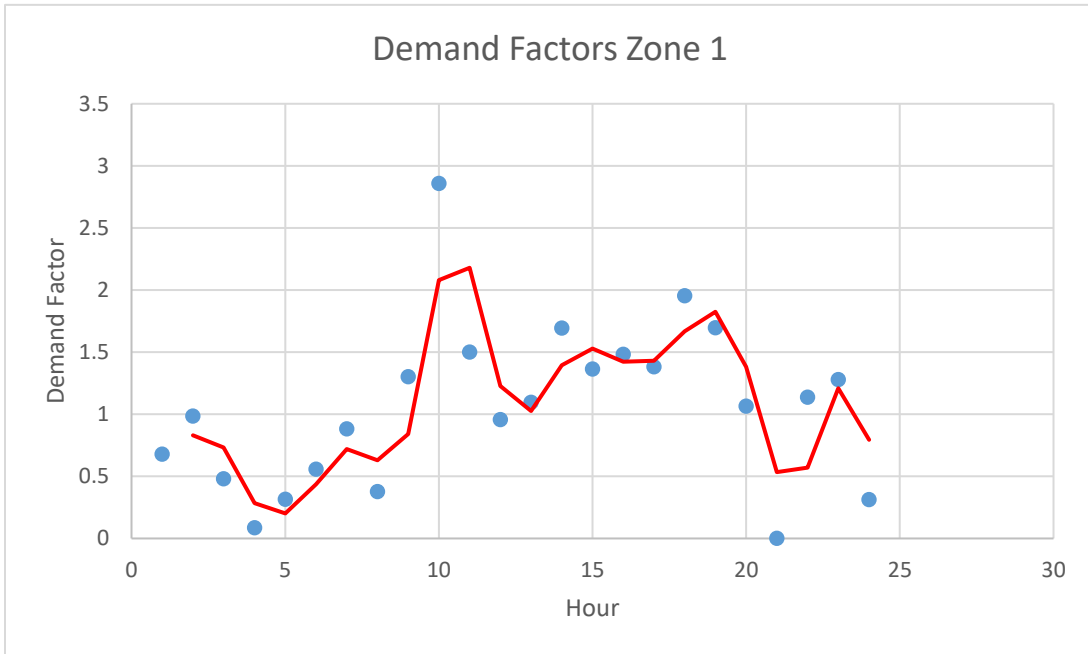


Figure A 20: Uncleaned Demand Factors for Zone 2 of LWW (6/29/2023)

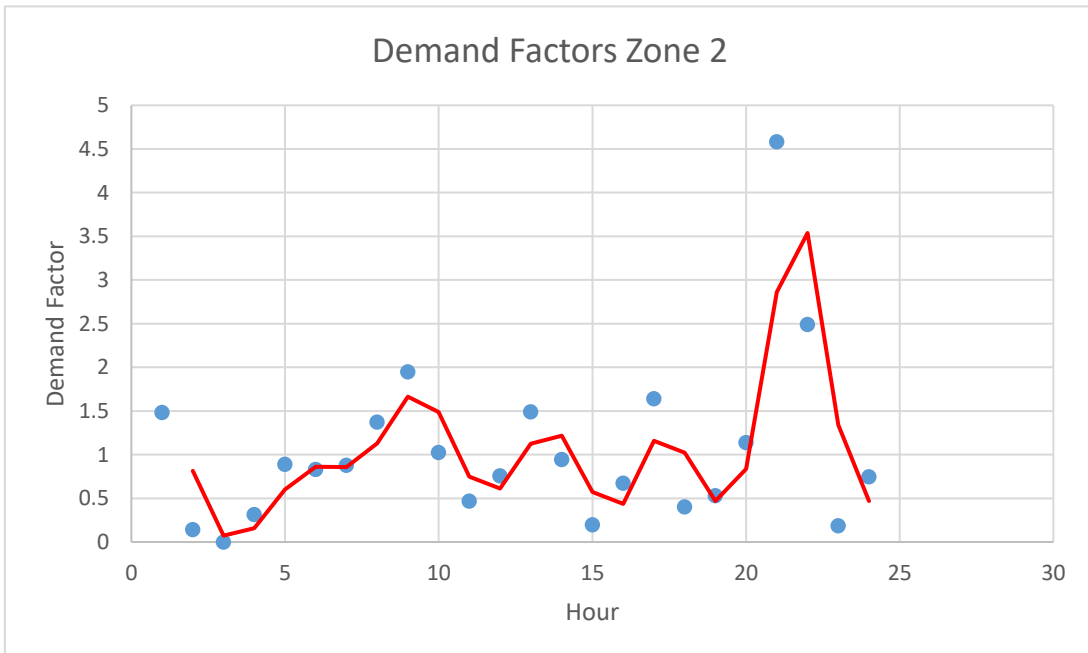


Figure A 21: Uncleaned Demand Factors for Zone 1 of LWW (6/30/2023)

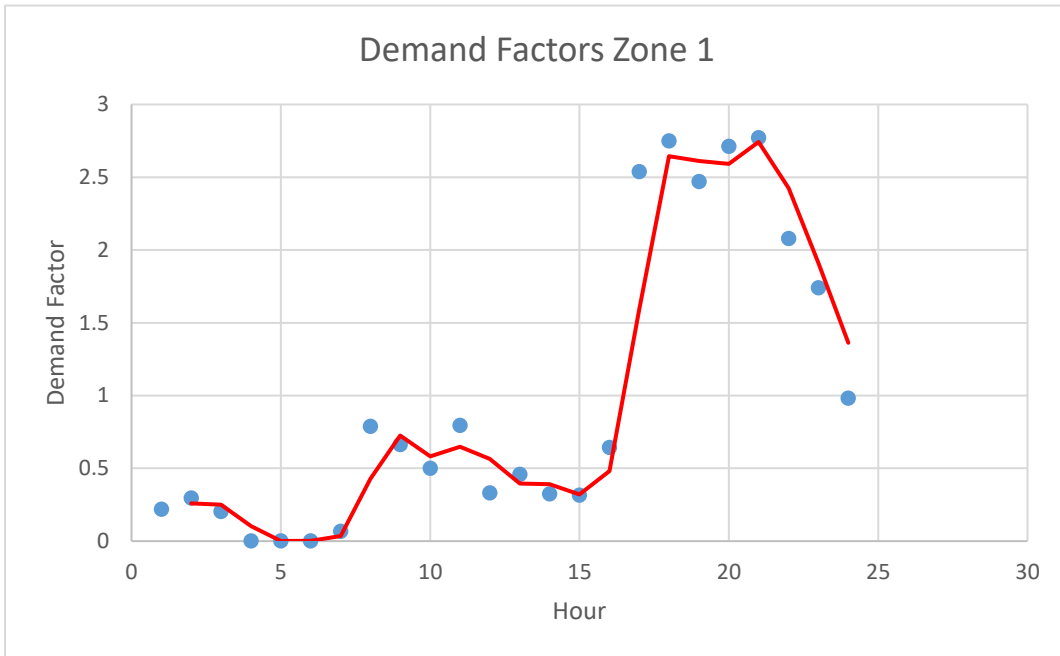


Figure A 22: Uncleaned Demand Factors for Zone 2 of LWW (6/30/2023)

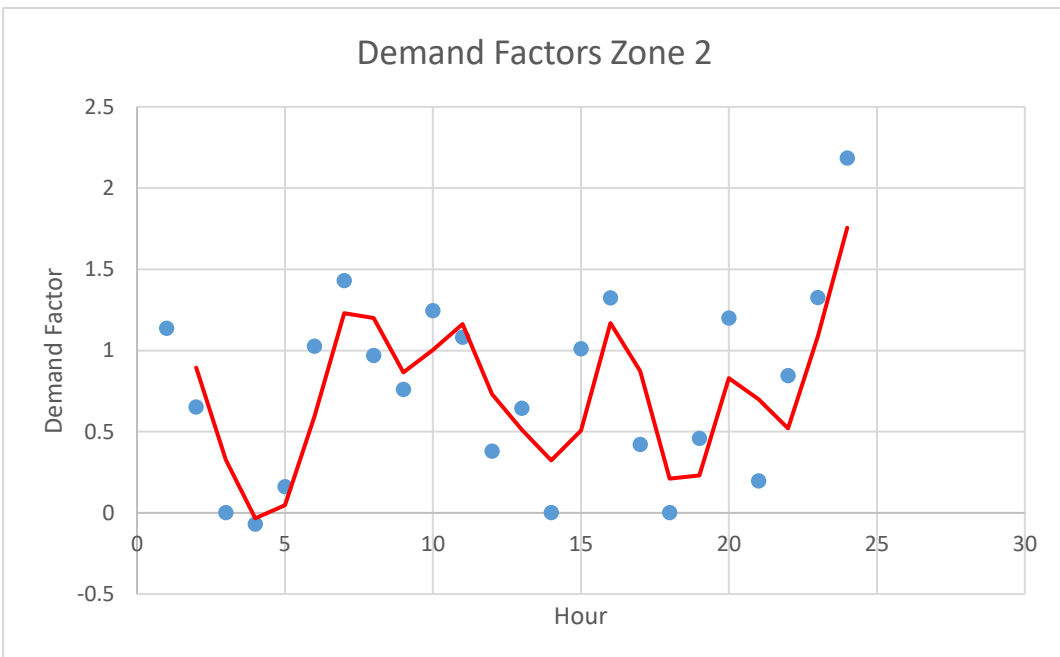


Figure A 23: Uncleaned Demand Factors for Zone 1 ofLWW (7/1/2023)

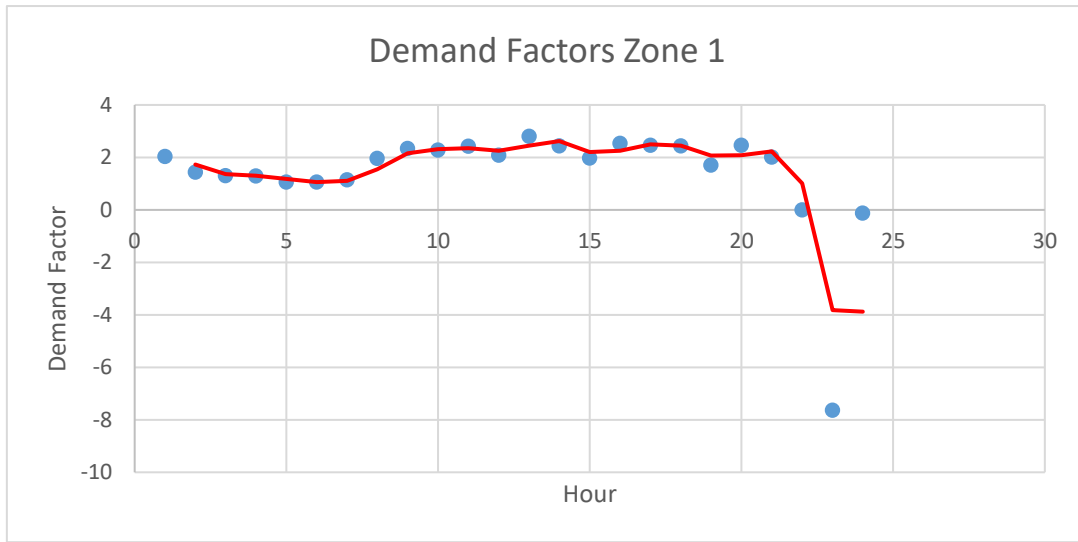


Figure A 24: Uncleaned Demand Factors for Zone 2 ofLWW (7/1/2023)

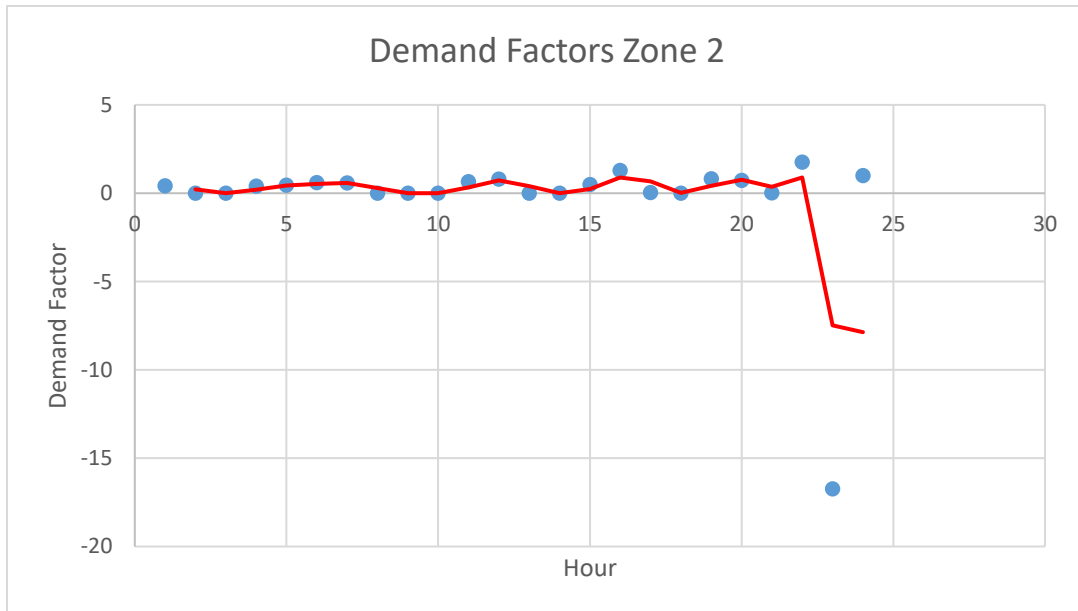


Figure A 25: Uncleaned Demand Factors for Zone 1 of LWW (7/2/2023)

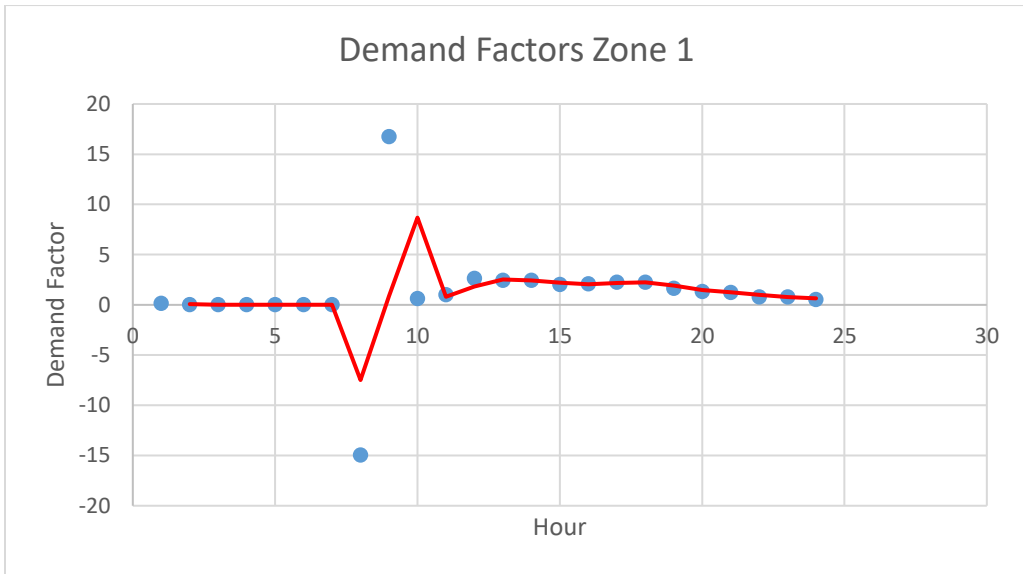


Figure A 26: Uncleaned Demand Factors for Zone 2 of LWW (7/2/2023)

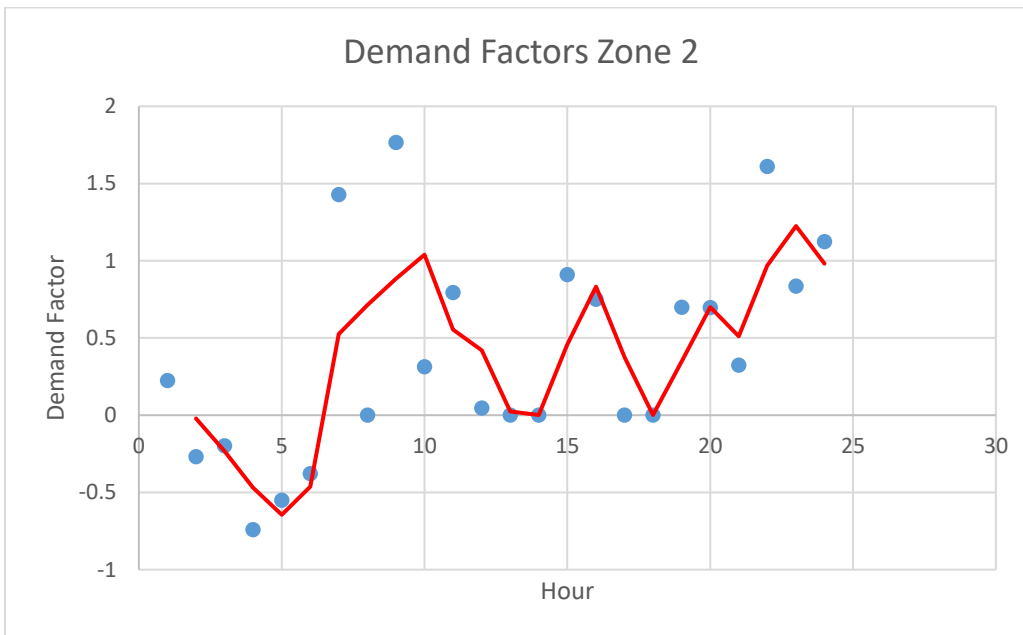


Figure A 27: Uncleaned Demand Factors for Zone 1 of LWW (7/3/2023)

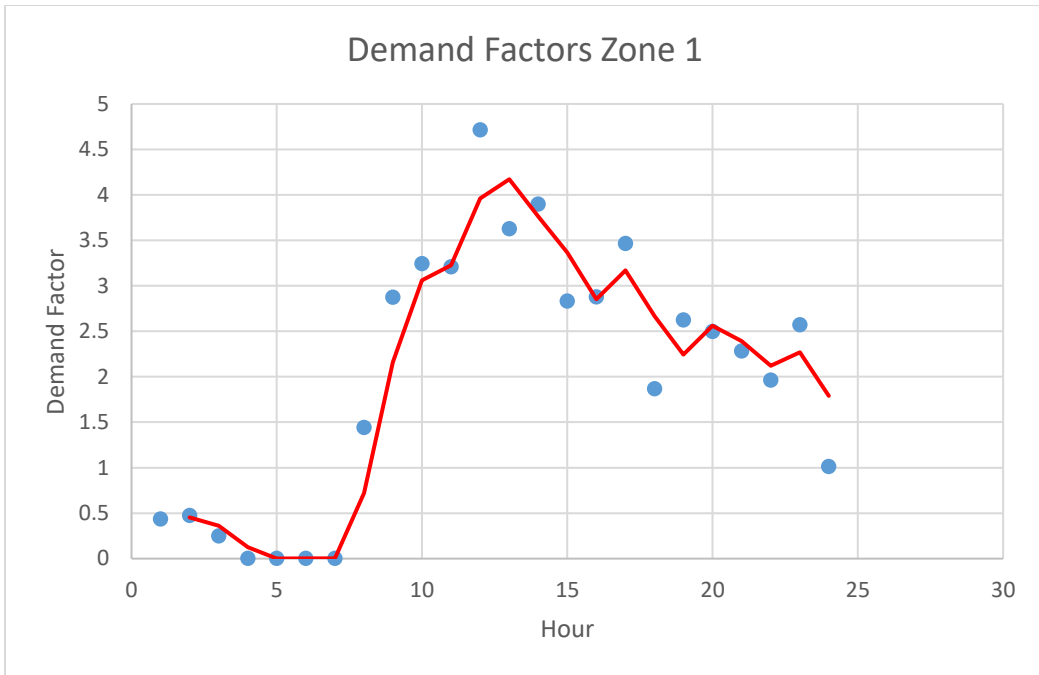


Figure A 28: Uncleaned Demand Factors for Zone 2 of LWW (7/3/2023)

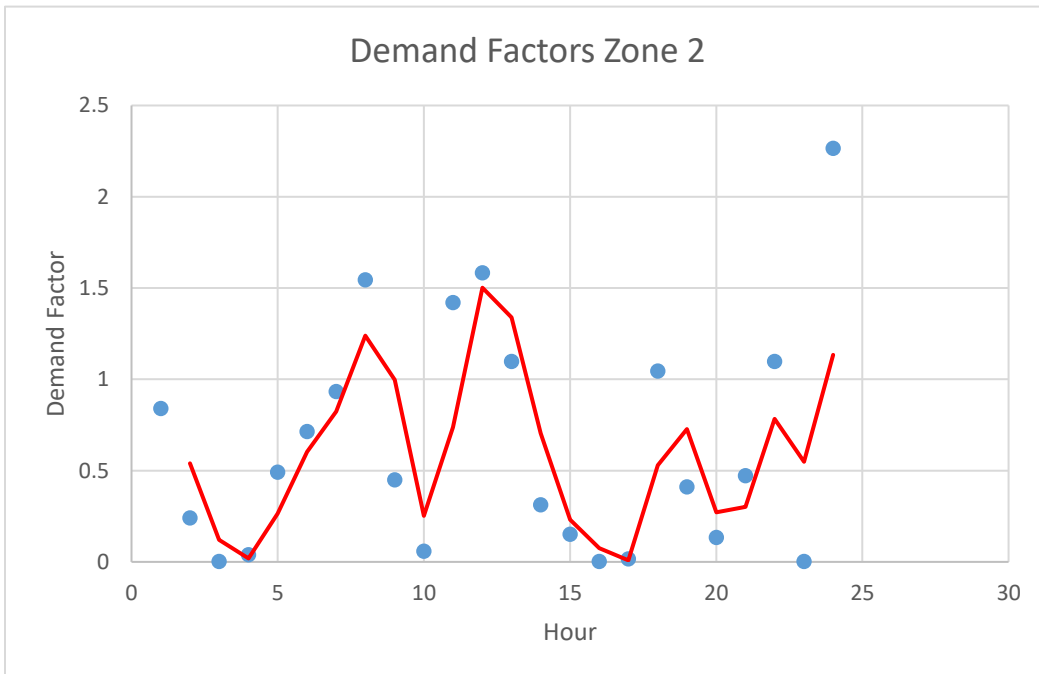


Figure A 29: Cleaned Demand Factors for Zone 1 of LWW (6/20/2023)

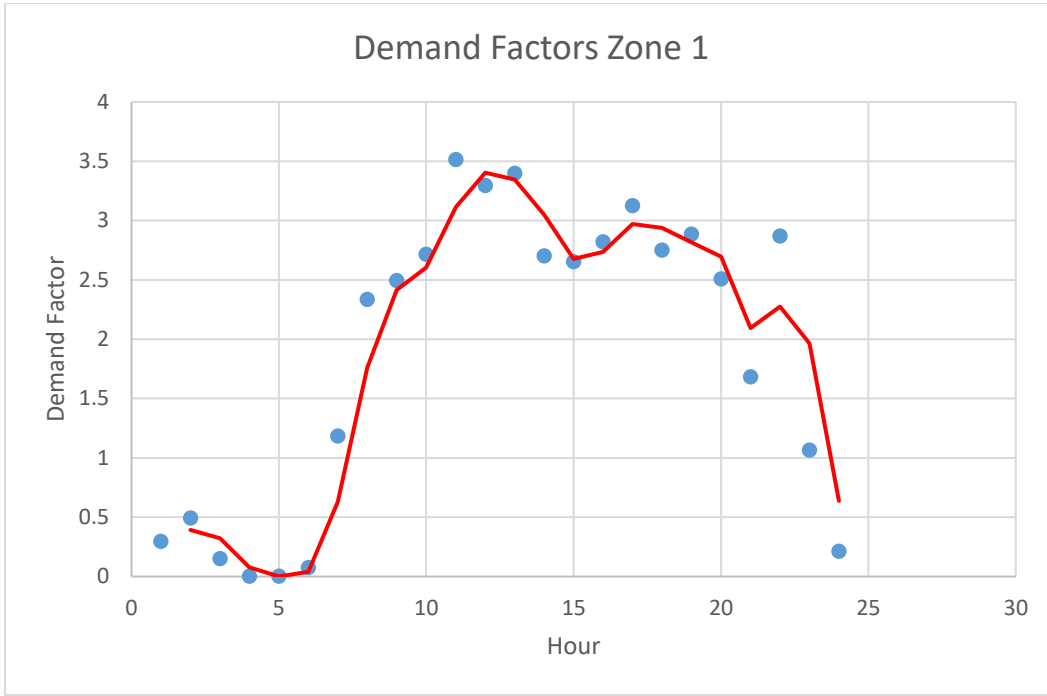


Figure A 30: Cleaned Demand Factors for Zone 2 of LWW (6/20/2023)

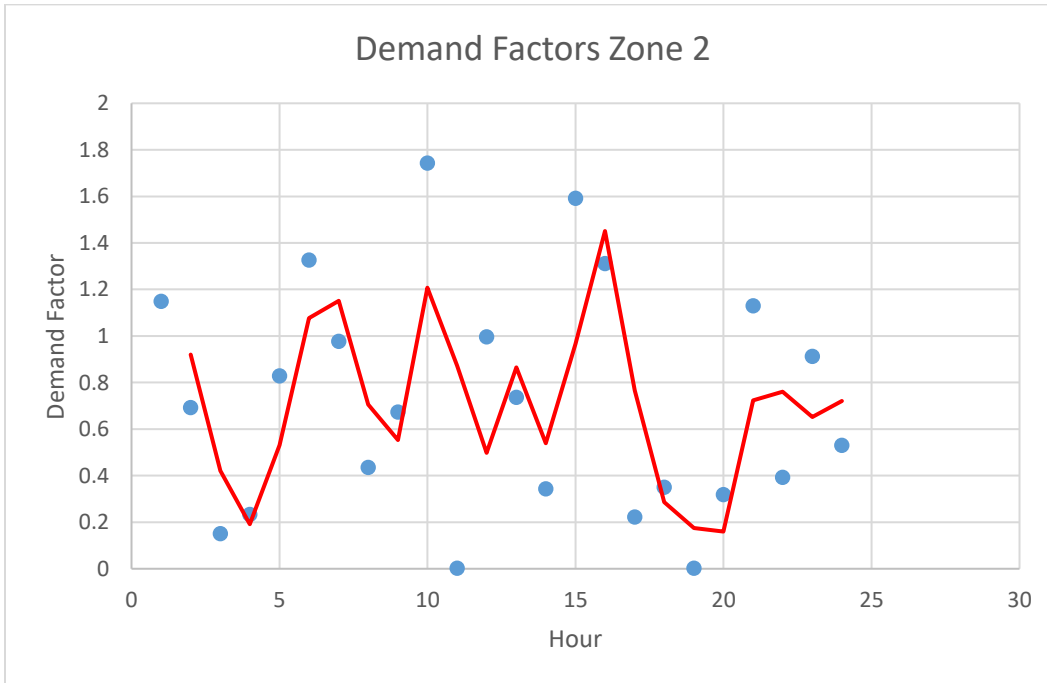


Figure A 31: Cleaned Demand Factors for Zone 1 of LWW (6/21/2023)

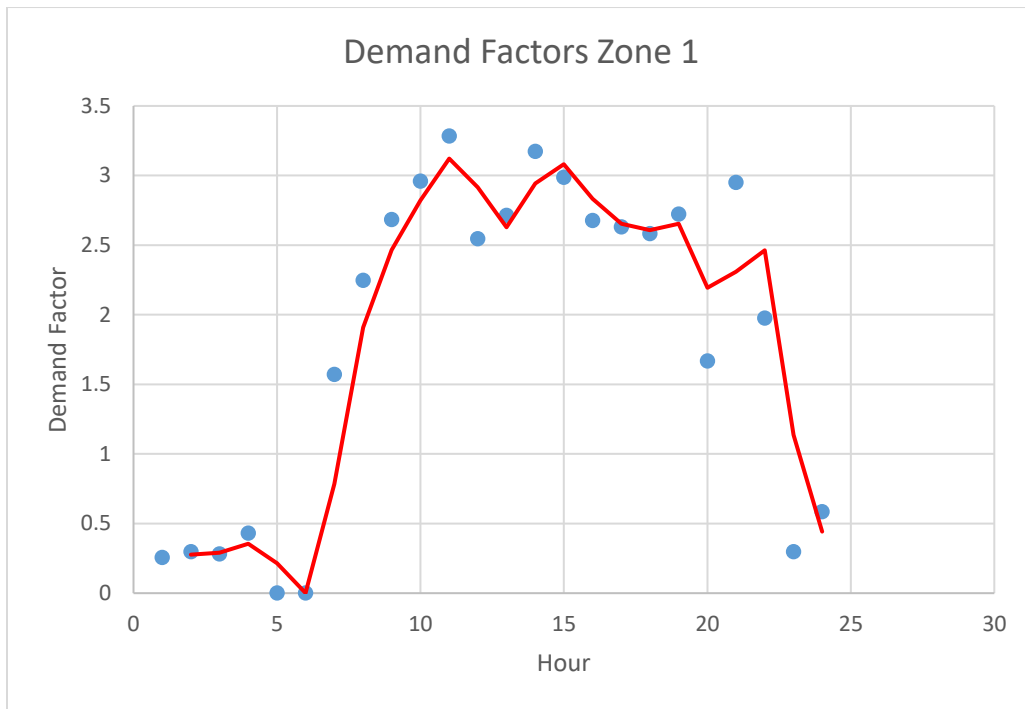


Figure A 32: Cleaned Demand Factors for Zone 2 of LWW (6/21/2023)

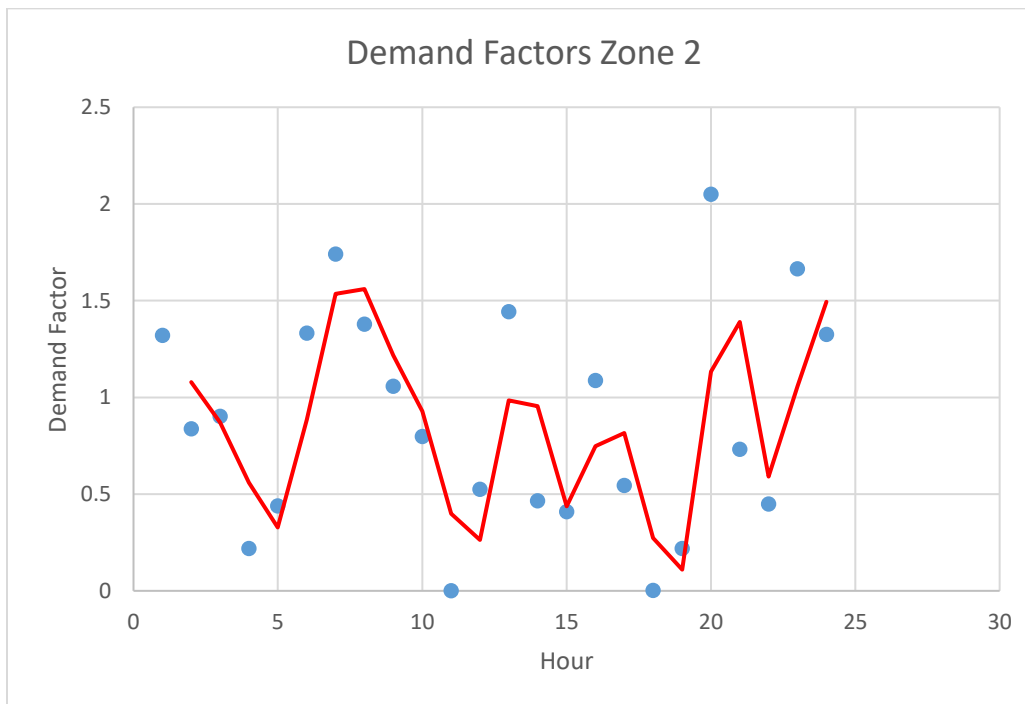


Figure A 33: Cleaned Demand Factors for Zone 1 of LWW (6/22/2023)

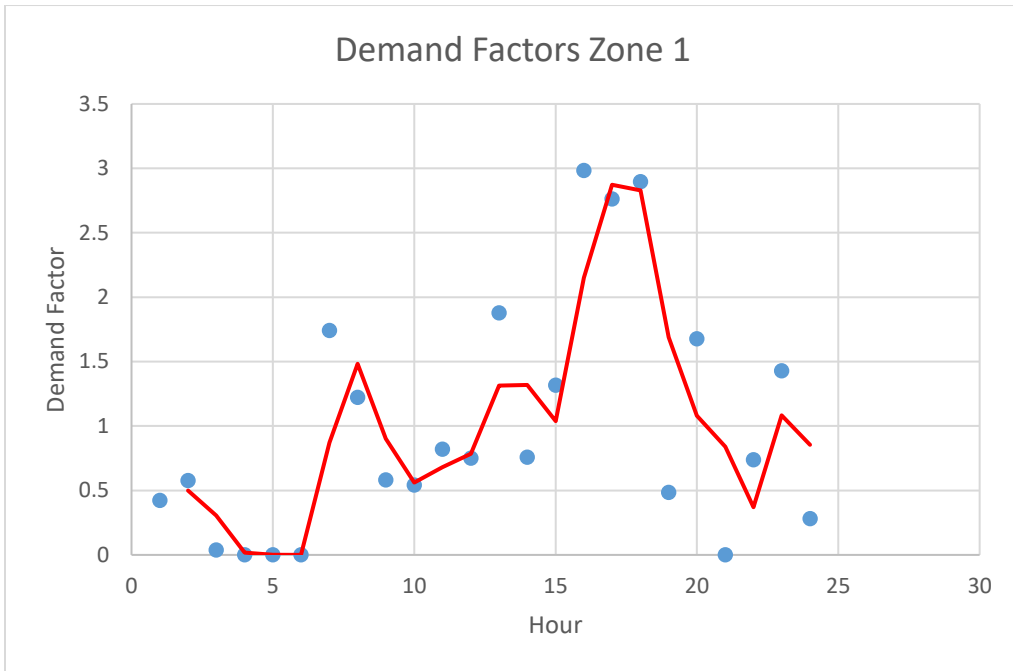


Figure A 34: Cleaned Demand Factors for Zone 2 of LWW (6/22/2023)

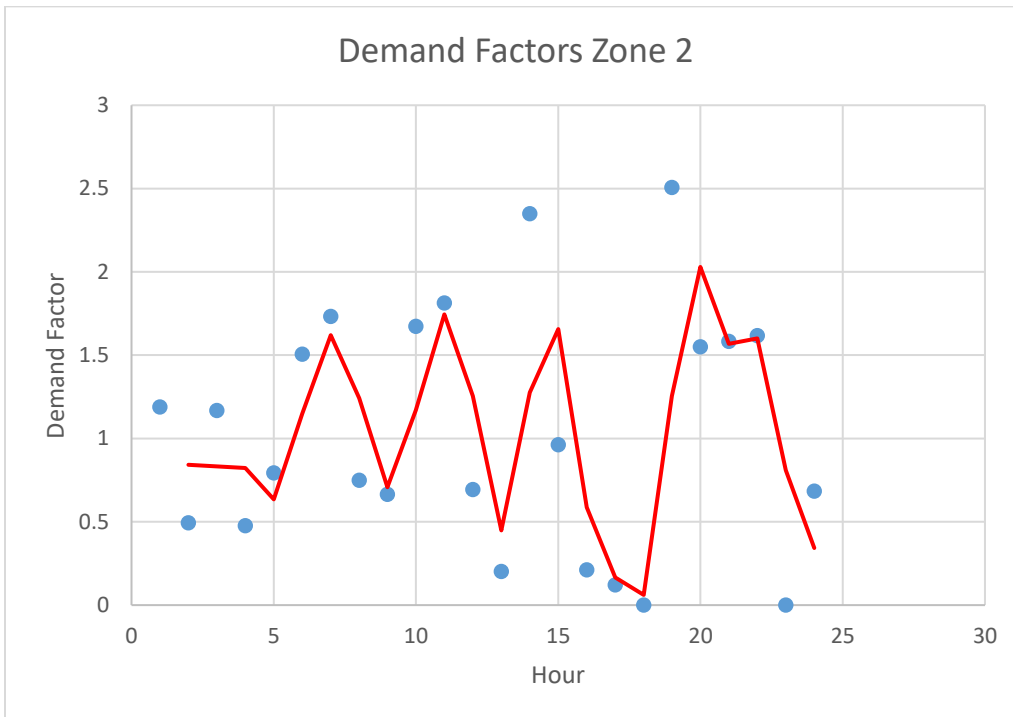


Figure A 35: Cleaned Demand Factors for Zone 1 of LWW (6/23/2023)

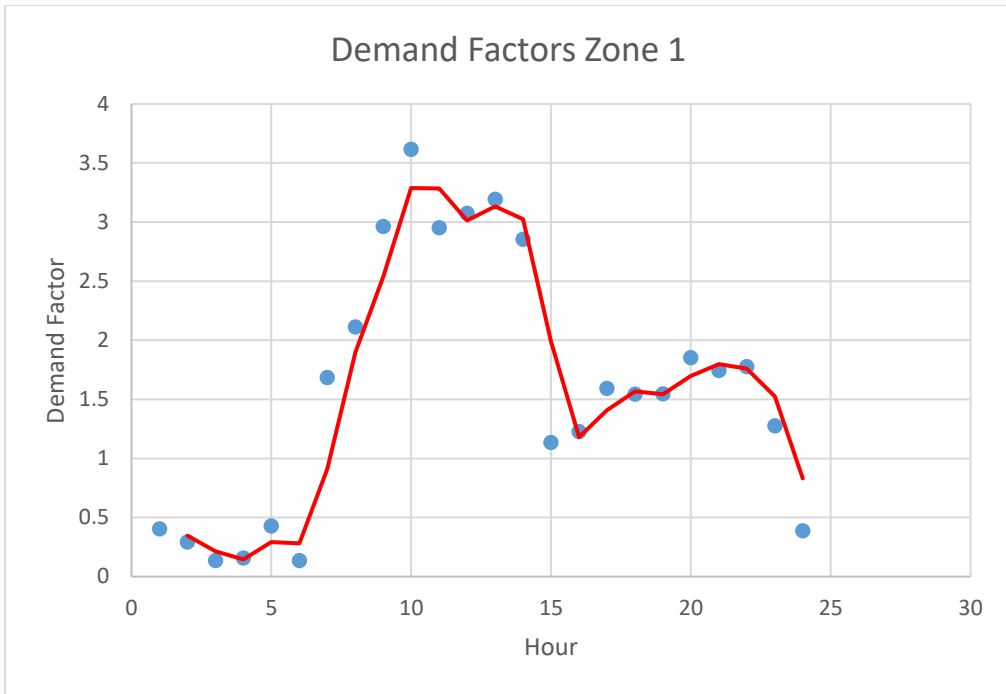


Figure A 36: Cleaned Demand Factors for Zone 2 of LWW (6/23/2023)

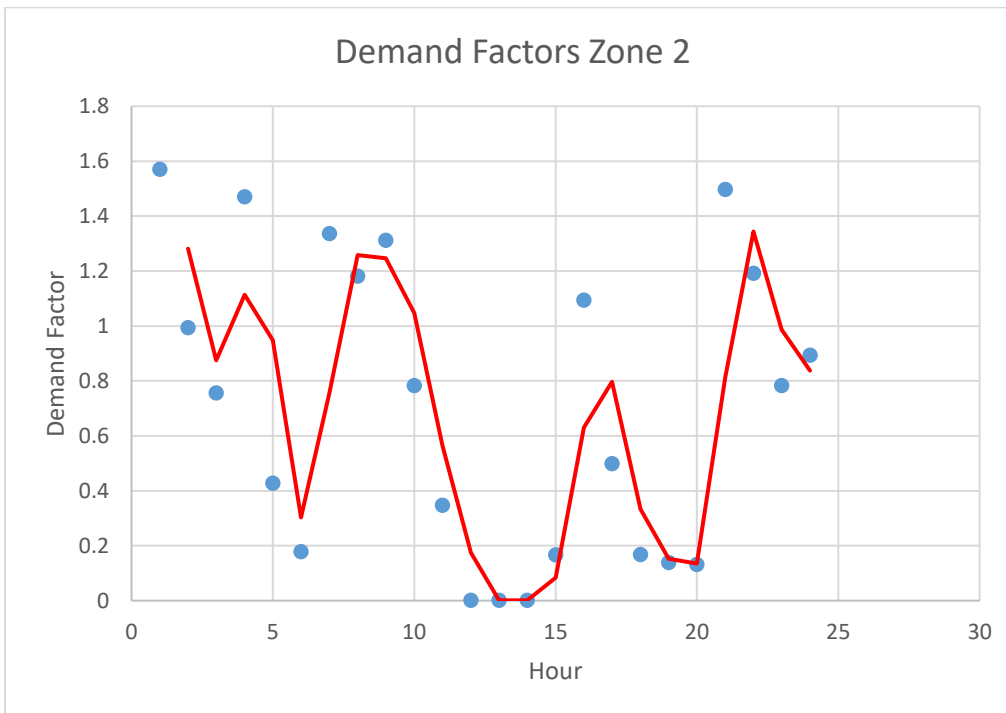


Figure A 37: Cleaned Demand Factors for Zone 1 of LWW (6/24/2023)

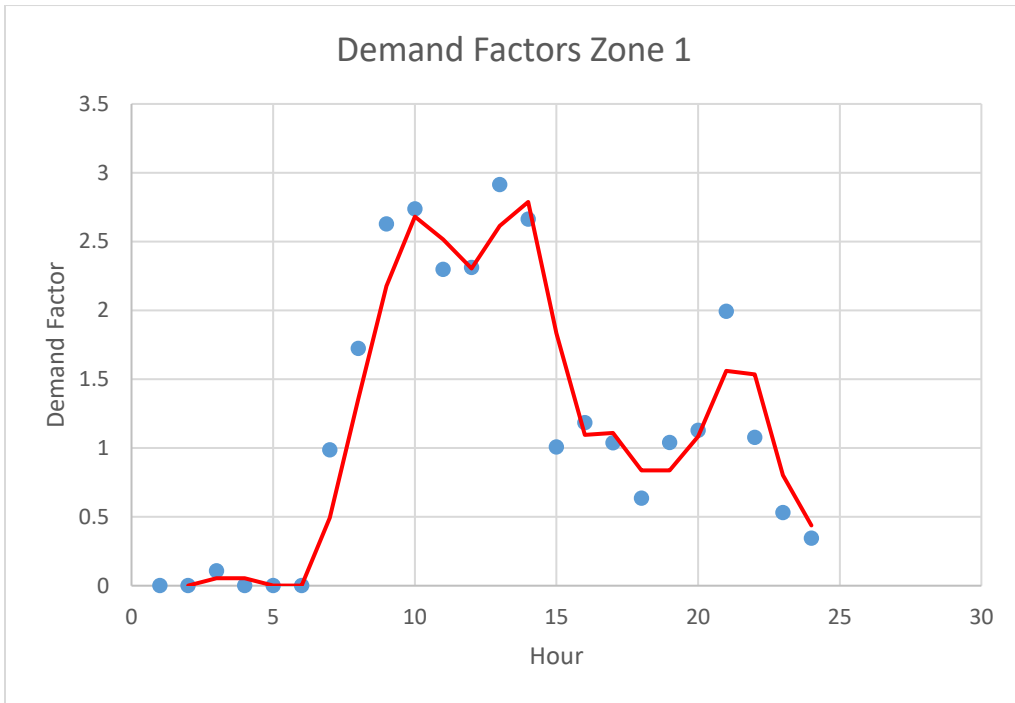


Figure A 38: Cleaned Demand Factors for Zone 2 of LWW (6/24/2023)

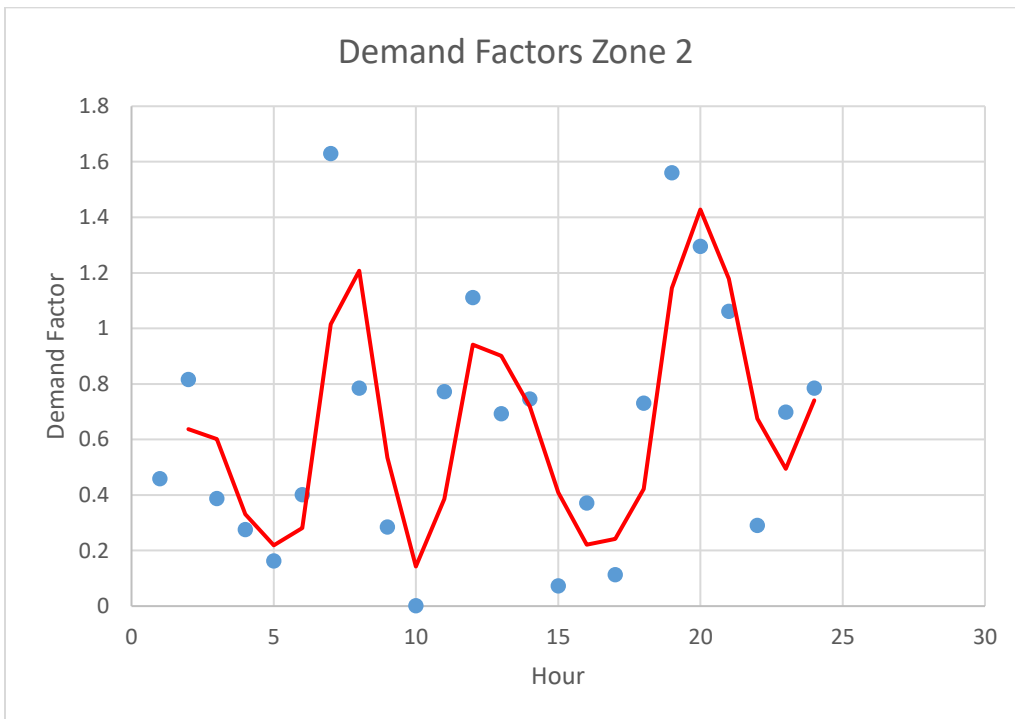


Figure A 39: Cleaned Demand Factors for Zone 1 of LWW (6/25/2023)

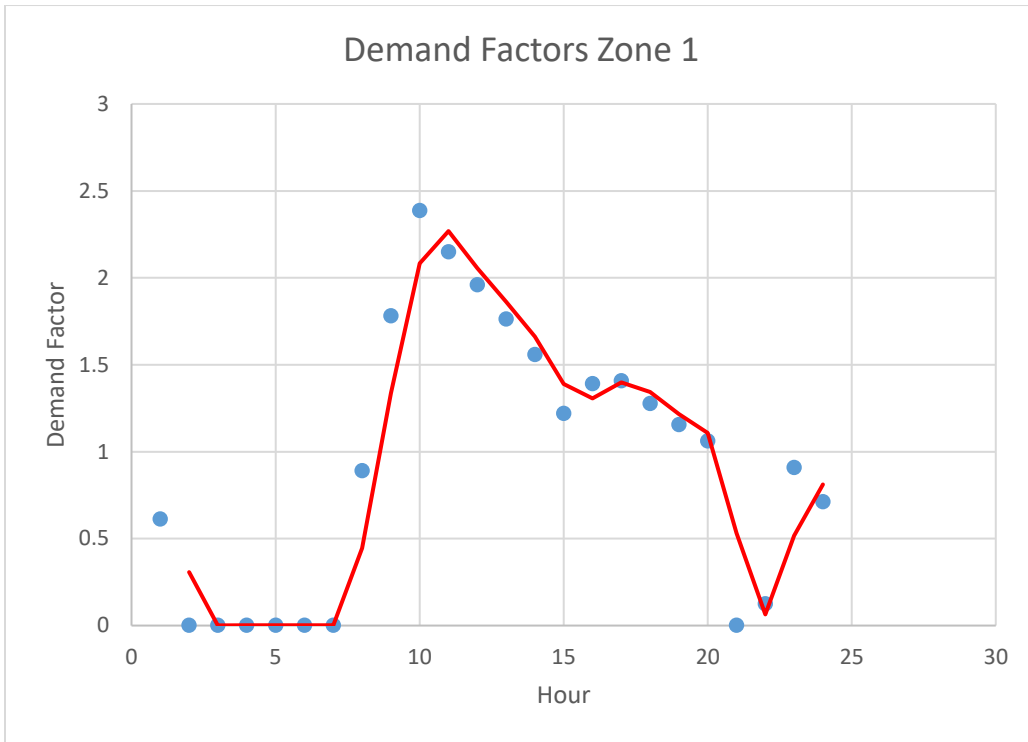


Figure A 40: Cleaned Demand Factors for Zone 2 of LWW (6/25/2023)

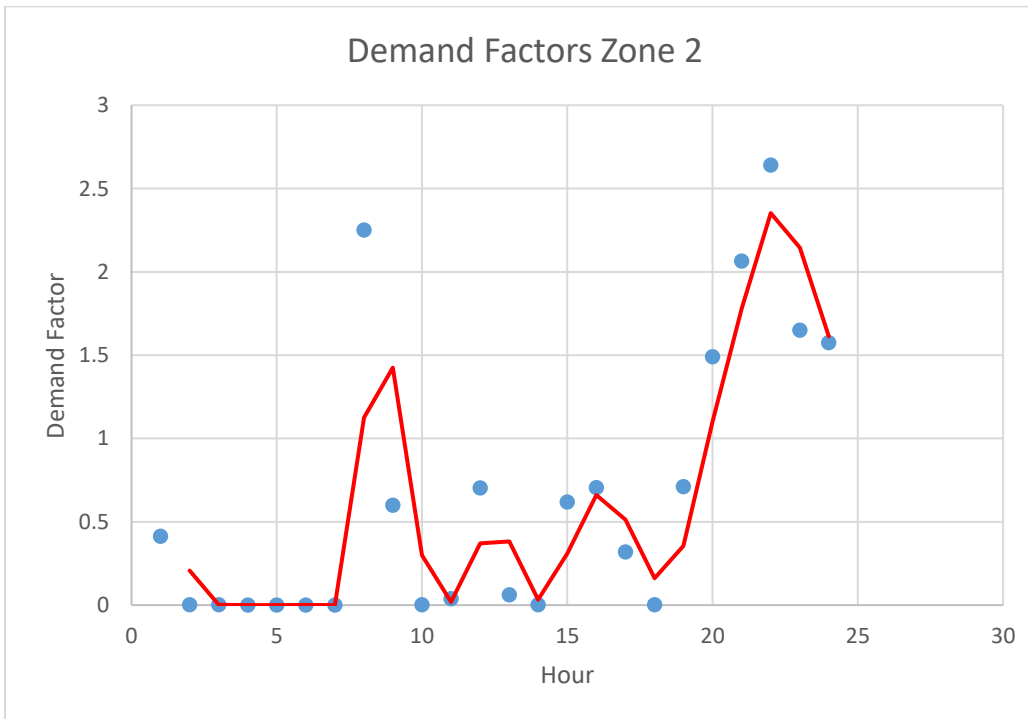


Figure A 41: Cleaned Demand Factors for Zone 1 of LWW (6/26/2023)

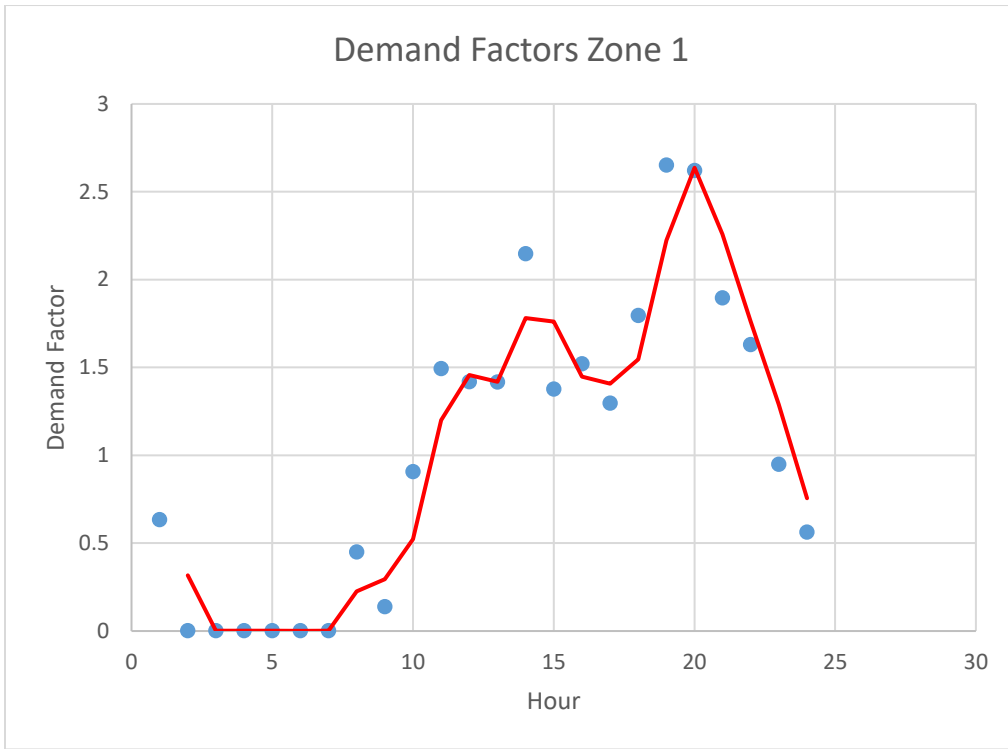


Figure A 42: Cleaned Demand Factors for Zone 2 of LWW (6/26/2023)

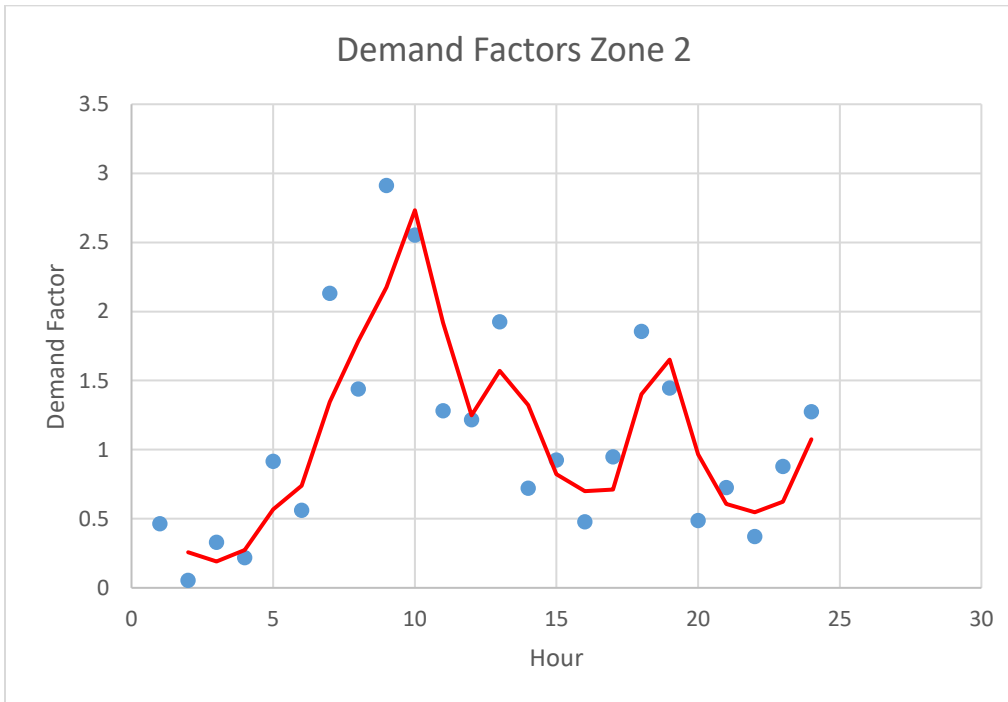


Figure A 43: Cleaned Demand Factors for Zone 1 of LWW (6/27/2023)

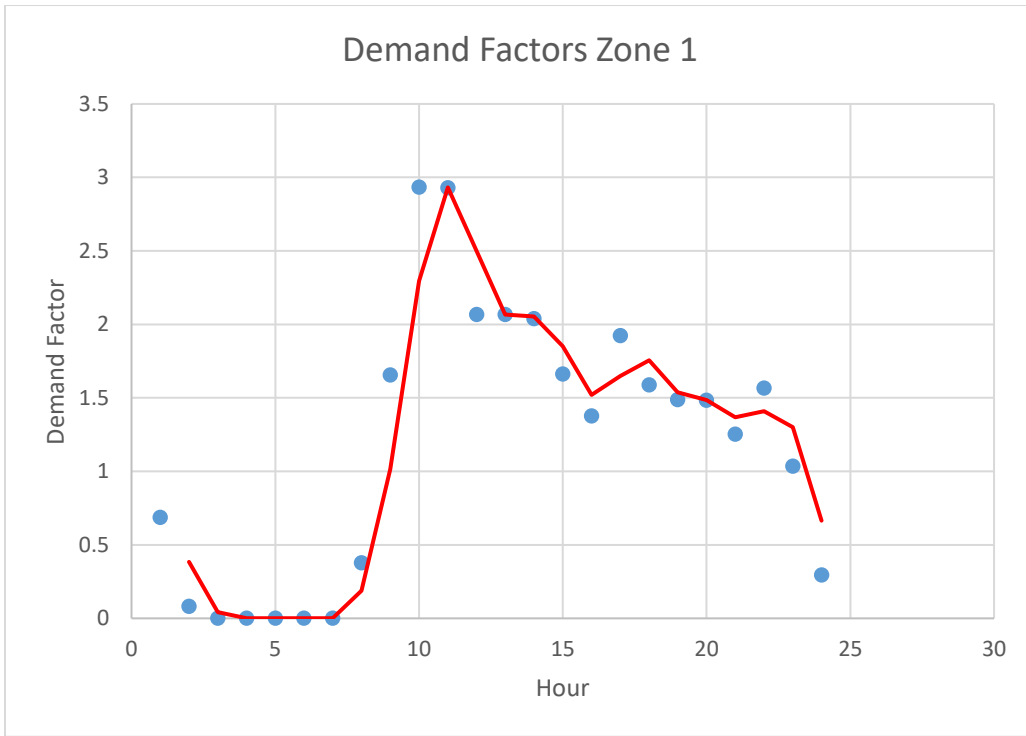


Figure A 44: Cleaned Demand Factors for Zone 2 of LWW (6/27/2023)

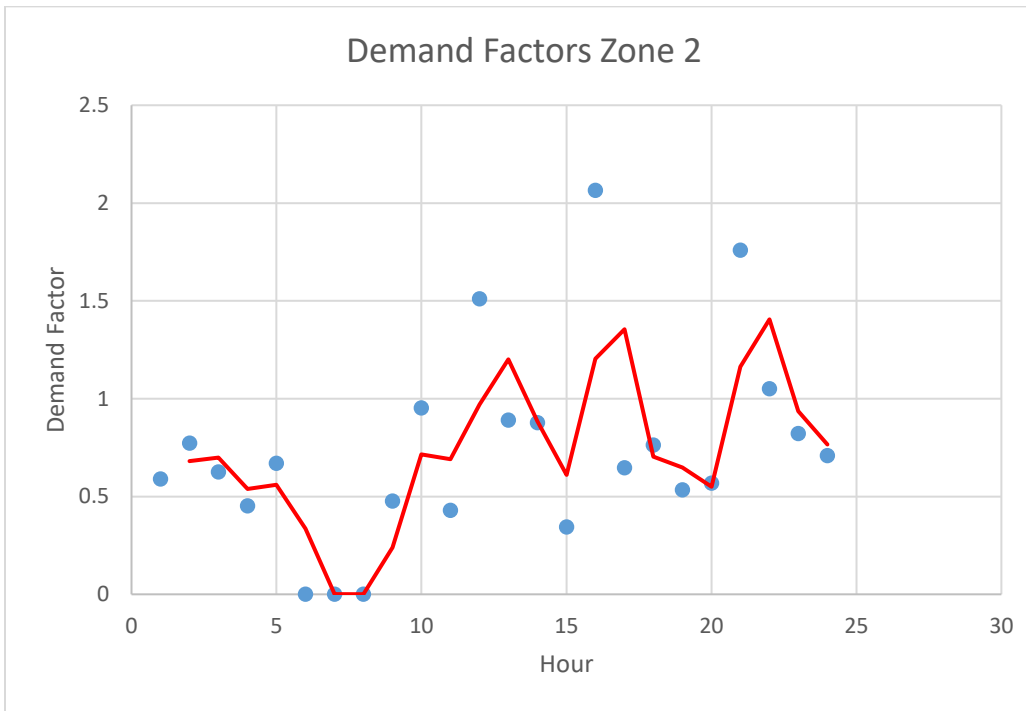


Figure A 45: Cleaned Demand Factors for Zone 1 of LWW (6/28/2023)

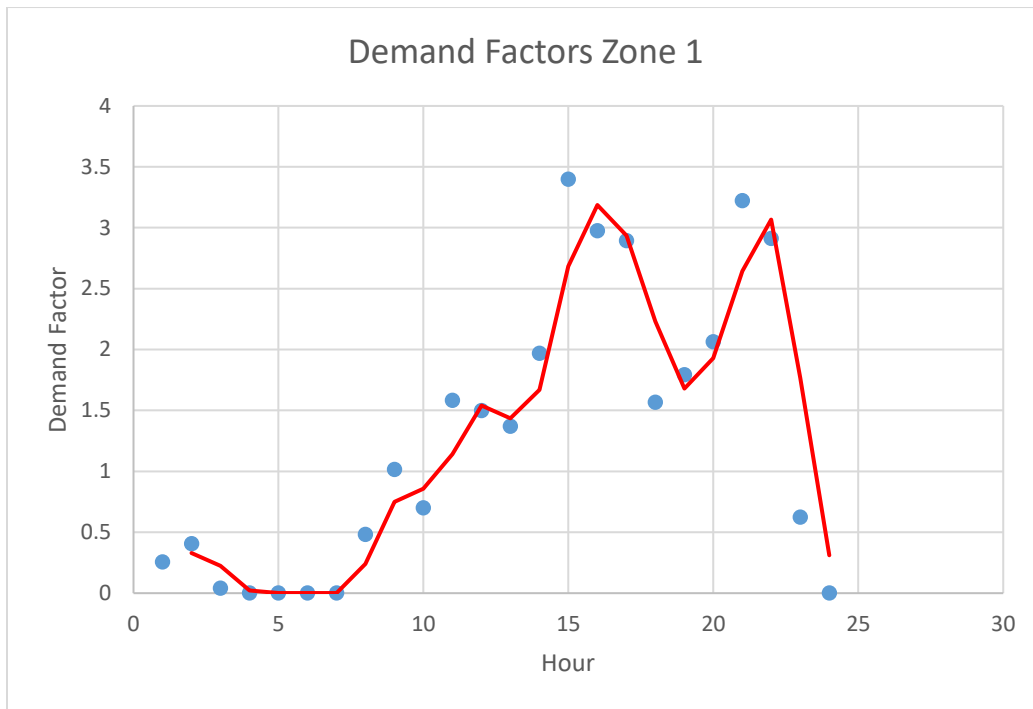


Figure A 46: Cleaned Demand Factors for Zone 2 of LWW (6/28/2023)

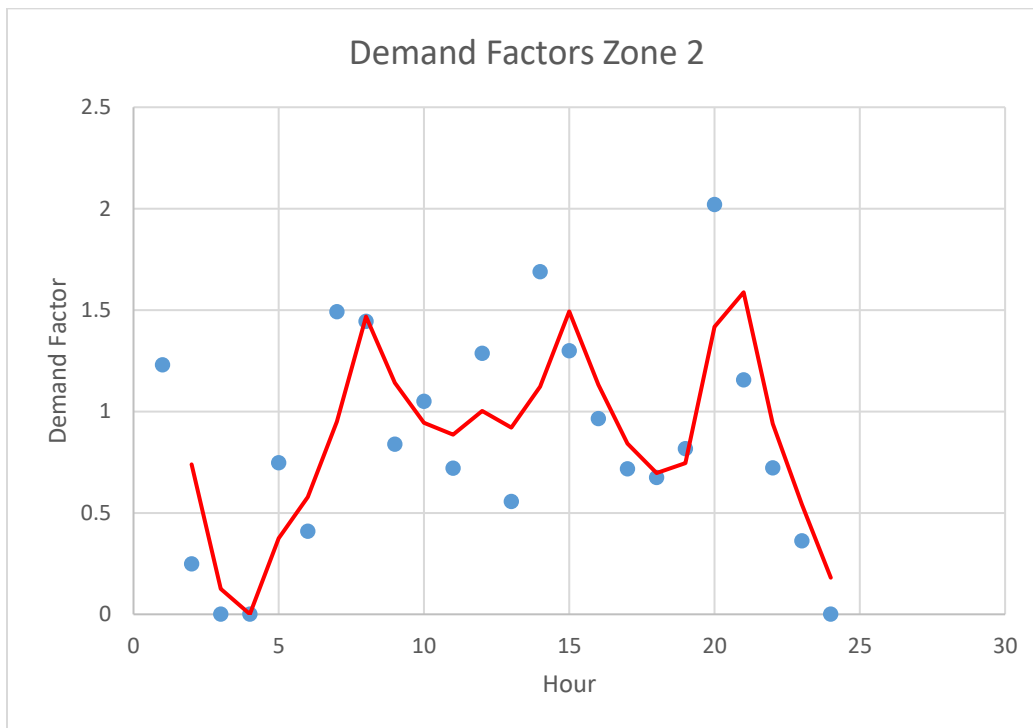


Figure A 47: Cleaned Demand Factors for Zone 1 of LWW (6/29/2023)

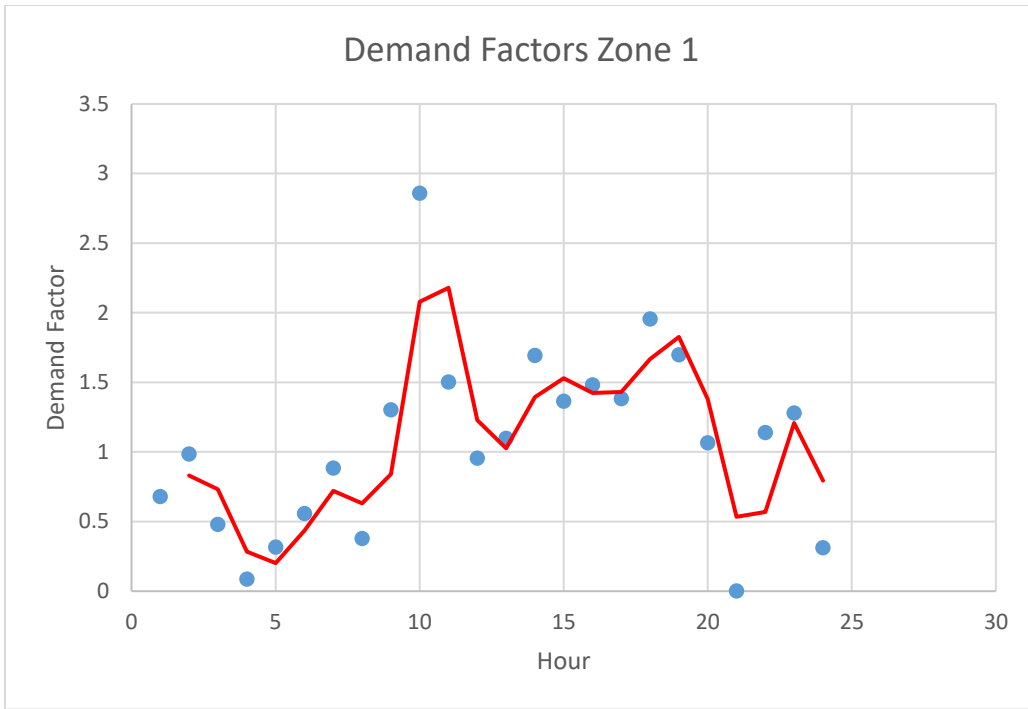


Figure A 48: Cleaned Demand Factors for Zone 2 of LWW (6/29/2023)

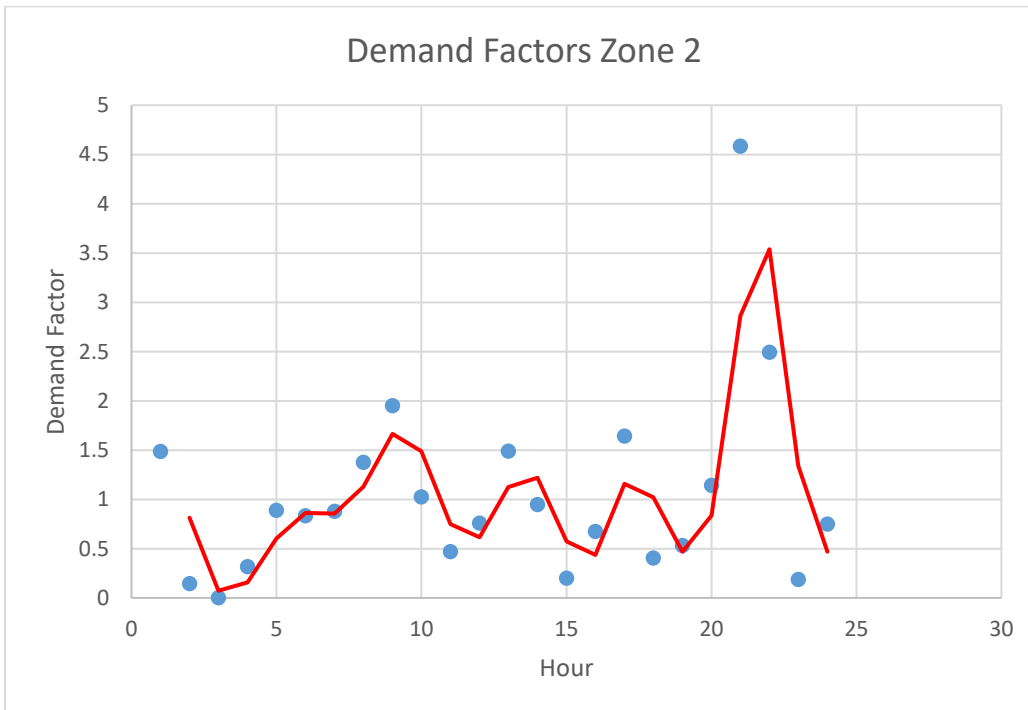


Figure A 49: Cleaned Demand Factors for Zone 1 of LWW (6/30/2023)

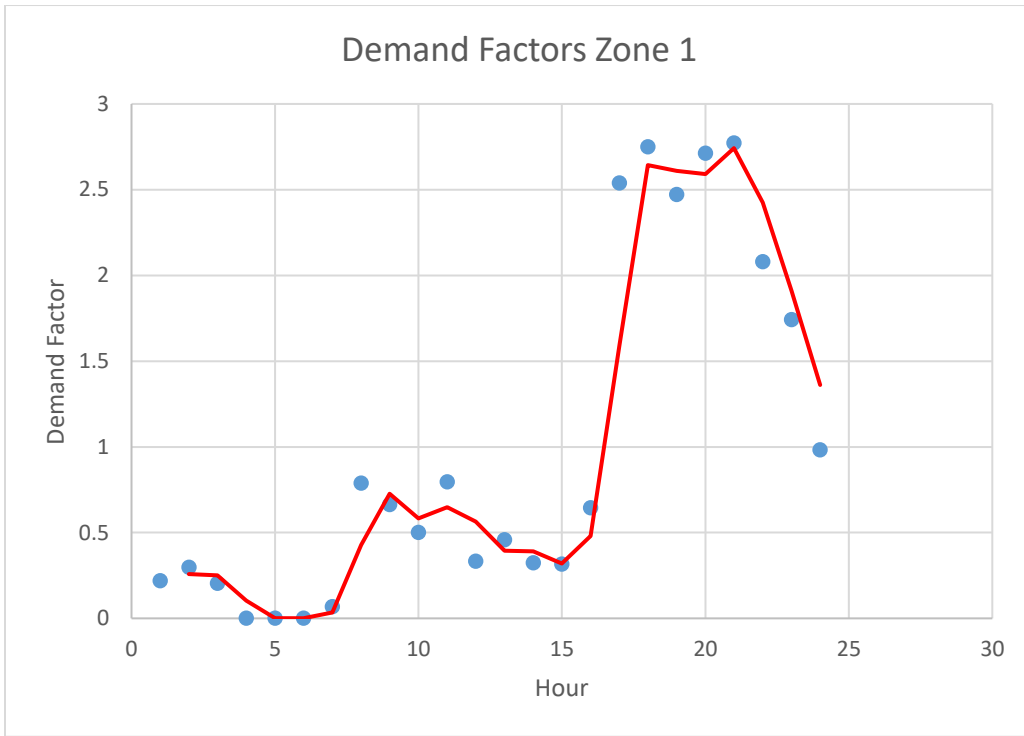


Figure A 50: Cleaned Demand Factors for Zone 2 of LWW (6/30/2023)

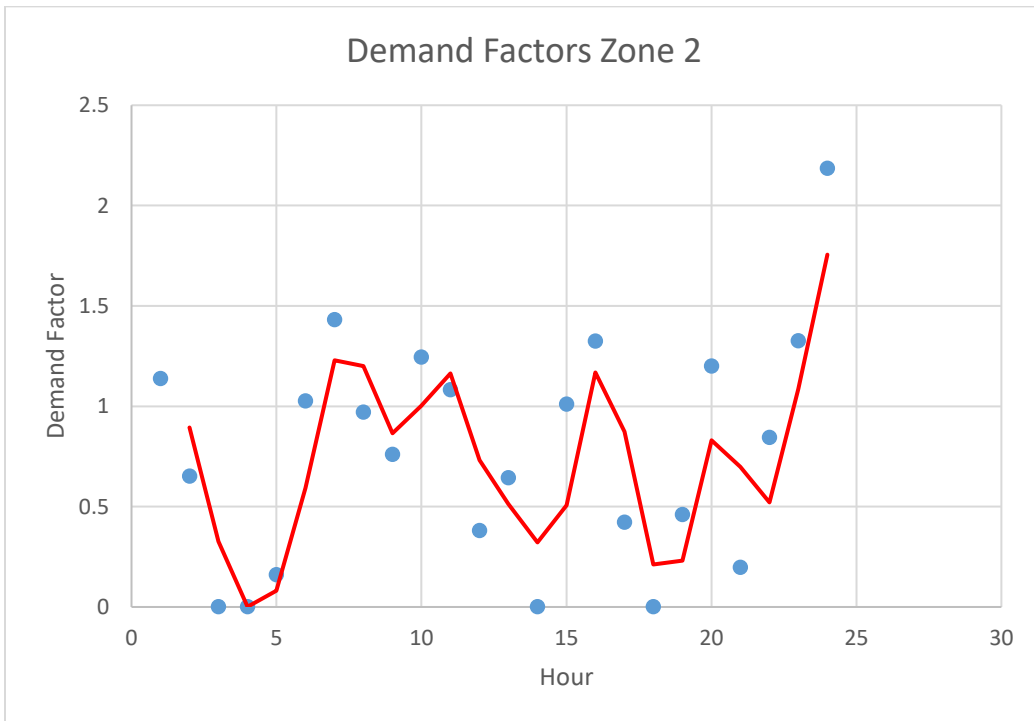


Figure A 51: Cleaned Demand Factors for Zone 1 of LWW (7/1/2023)

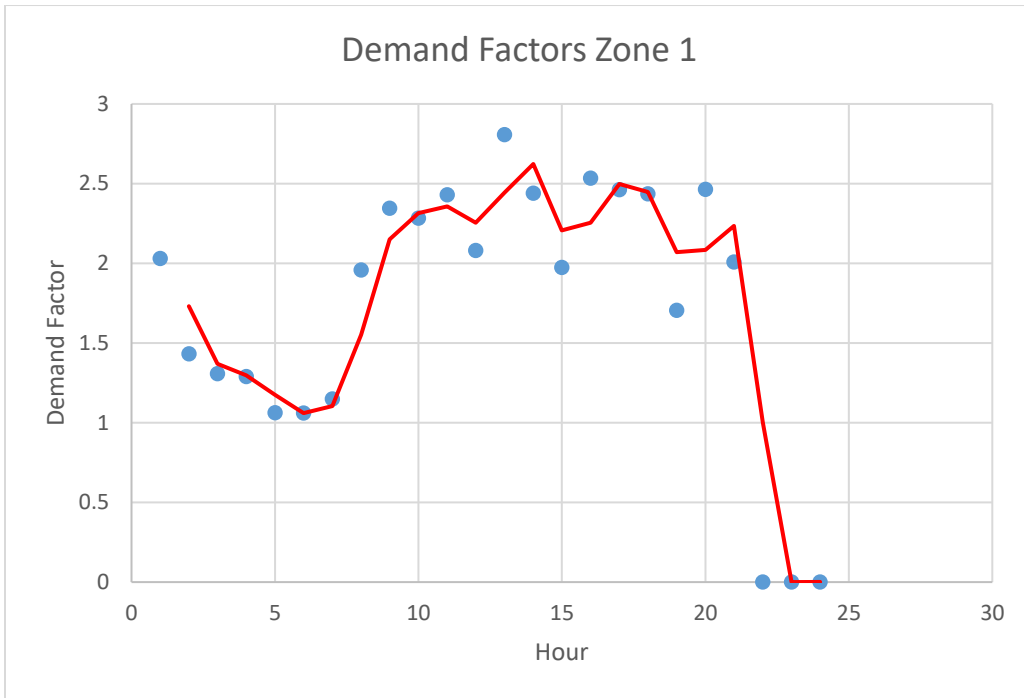


Figure A 52: Cleaned Demand Factors for Zone 2 of LWW (7/1/2023)

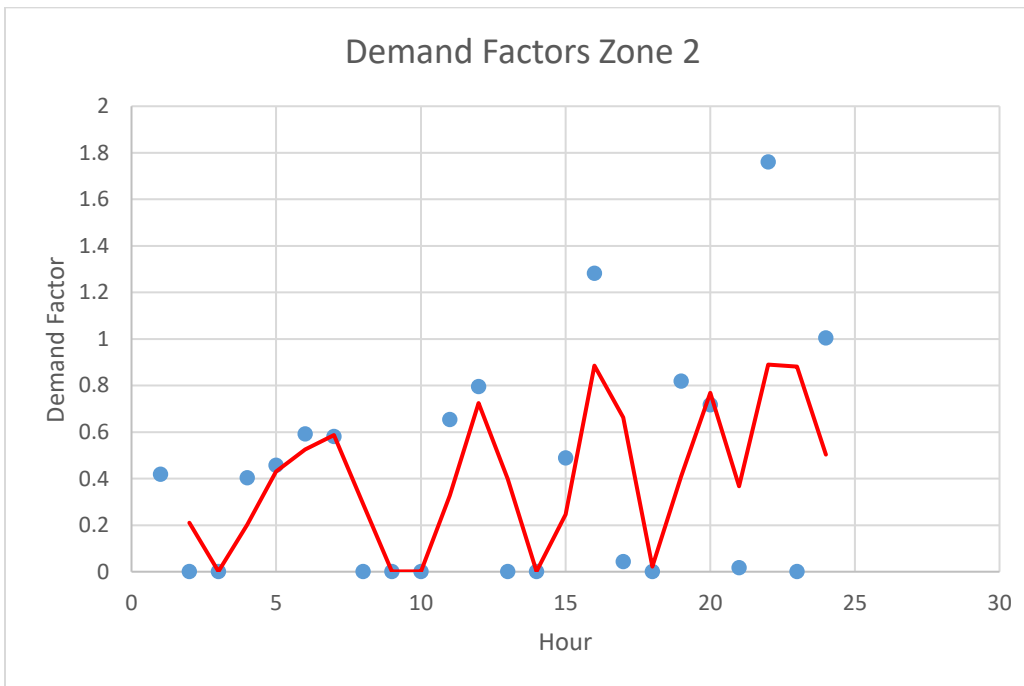


Figure A 53: Cleaned Demand Factors for Zone 1 of LWW (7/2/2023)

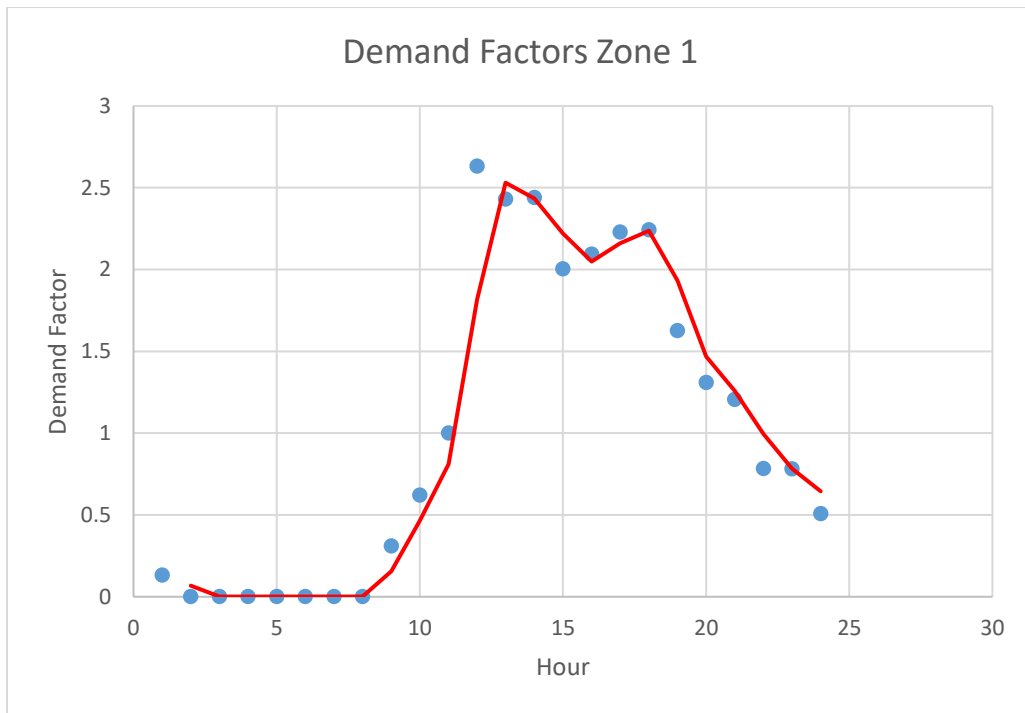


Figure A 54: Cleaned Demand Factors for Zone 2 of LWW (7/2/2023)

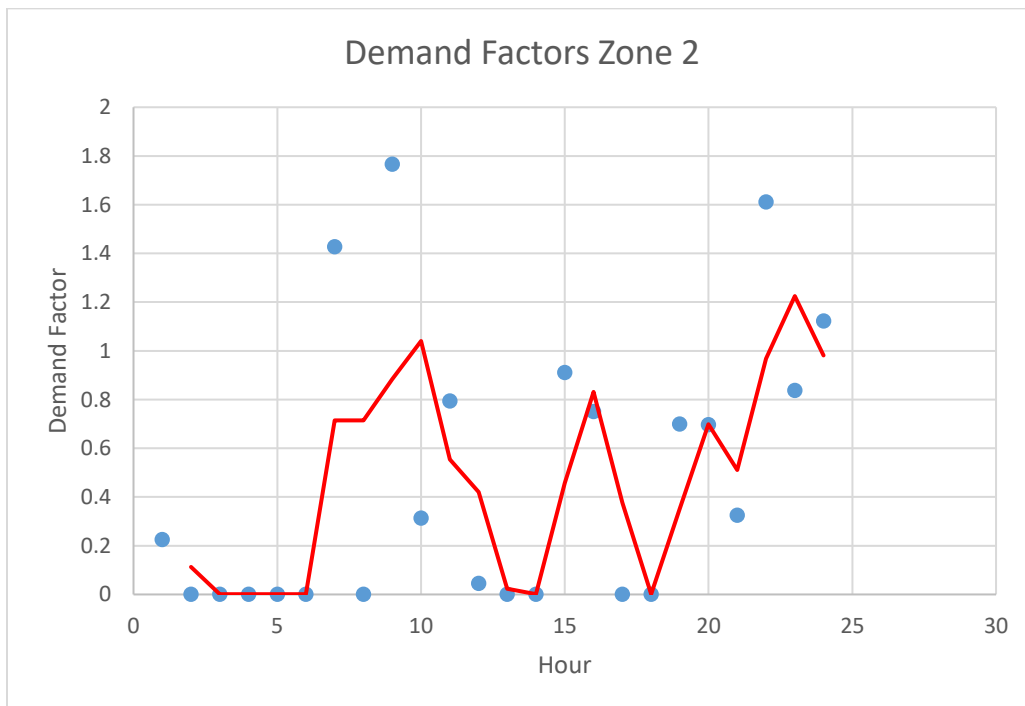


Figure A 55: Cleaned Demand Factors for Zone 1 of LWW (7/3/2023)

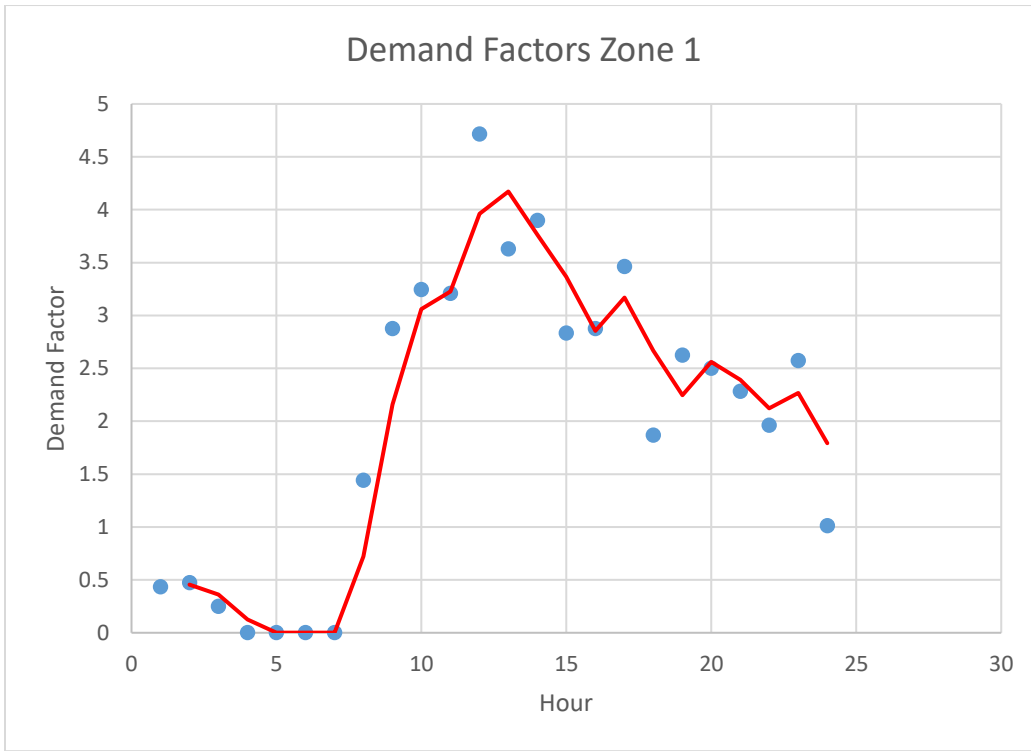
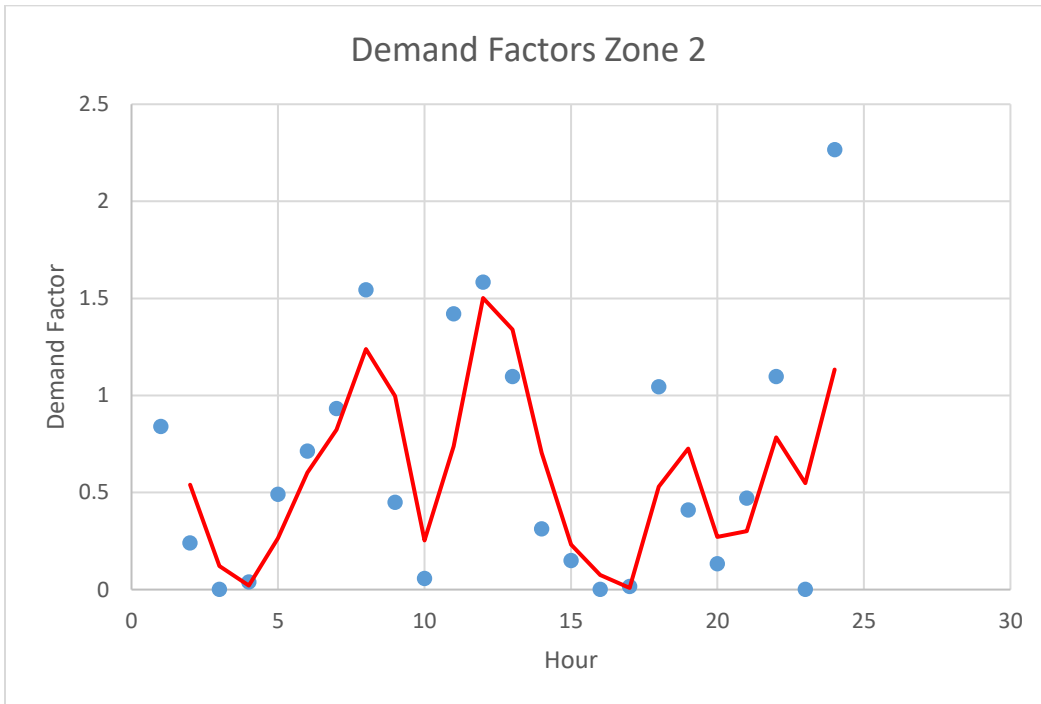


Figure A 56: Cleaned Demand Factors for Zone 2 of LWW (7/3/2023)



APPENDIX B. Code For Hydraulic Simulation And Demand Factor Calibration

```
%Final extended period simulation code (for result output in the app)

function [PressureComp, FlowComp, QualityComp, TankLevels, TankWaterAge, PumpHGL,
PumpFlowRate, JunctionPressure, JunctionDemand, JunctionChlorine, JunctionTTHM, Time, d]
= ExtendedPeriodV2(varargin)

%The first argument into the function will be whether or not the input
%file is the current system in Lebanon or the new projected system with
%the replaced tank and new 16" line
whatFile = varargin{1};

if whatFile == 0
    d = epanet('LebanonCurrent_July2023Testable.inp', 'LoadFile');
elseif whatFile ==1
    d = epanet('LebanonNew_July2023.inp', 'LoadFile');
end

NodeNames = d.getNodeNameID;
Targets = ["ByPass", "BeforeCalvaryMeter", "CalvaryMeter", "woodlawnMeter",
"DanvilleHighwayMeter", "SpringfieldRoadMeter", "StRoseMeter", "StMaryMeter",
"CampbellsvilleMeter", "598", "808"];
[~,whereNames] = ismember(Targets, NodeNames);

%Set the initial tank levels
tankInfo = d.getNodeTankData;
tankMinLevel(1,1:2) = tankInfo.Minimum_Water_Level;

%this is important because EPANET interprets tank level as the values
%between minimum and maximum. Therefore we need the relative tank level (user input) and
the minimum
%tank level and then add them together to place and the initial level
%input.
tankLevelSpecs = varargin{2};
tankInitialLevel = varargin{2} + tankMinLevel;

    if tankInitialLevel > tankMinLevel
        %Do nothing
    elseif tankInitialLevel < tankMinLevel
        %Give the user an error if the initial level is too low
        ErrorMessage = sprintf('One of the tank input values is lower than the
mimimum tank level.\n Please retry with different initial inputs.');
```

```
        msgbox(ErrorMessage, 'Input Error', 'error');
    end

d.setNodeTankInitialLevel(tankInitialLevel);

%Set the initial Pump Controls - Note, these need to be cleaned up and
```



```

%Formatted correctly in order to be implemented into the analysis, this is
%done on the front end in the application itself

PumpControls = varargin{3};
%Remove all of the original controls
d.deleteControls;
%add the new controls
d.addControls(PumpControls);

%Hydraulically speaking we have the initial tank levels and now the pump
%conditions as well. Other than adding in the specified demands we are good
%to go onto our chlorine analysis

%Chlorine residual from the plant
Chlorine = varargin{4};

%relic from using two files, leave for now
if whatFile == 0

%Set the chlorine concentration coming from the treatment plant
d.setNodeInitialQuality(986, Chlorine);
d.setNodeSourceQuality(986, Chlorine);

%Set the reaction coefficients
Bulk = varargin{5};
wall = varargin{6};

d.setLinkBulkReactionCoeff((0*d.getLinkBulkReactionCoeff) + Bulk);
d.setLinkWallReactionCoeff((0*d.getLinkWallReactionCoeff)+wall);

%Set the time specifications
Time4 = varargin{7};
Time5 = varargin{8};
Time6 = varargin{9};

d.setTimePatternStep(Time4 * 3600)
d.setTimeReportingStep(Time4 * 3600)
d.setTimeHydraulicStep(Time4 * 3600);
d.setTimeQualityStep(Time5 *3600);
d.setTimeSimulationDuration(Time6 * 3600);

%Set the demand patterns
DemandFactors = varargin{10};
DZ1 = DemandFactors(:,1)';
DZ2 = DemandFactors(:,2)';

%pattern for first high pressure zone (WTP)
d.setPattern(3, DZ1(1,:));
%pattern for Springfield Road Pressure Zone
d.setPattern(4, DZ2(1, :));

```

```

%we need to run this twice in order to get water age as well as the
%chlorine info, they are seperate analysis in EPANET

for i = 1:2

    %on the second iteration change the quality analysis to chlorine
    if i ==2
        d.setQualityType('chem', 'CHLORINE')
    end
    %relatively simple to run the analysis, here is the code for that.
    %Open, initialize (never understood why we needed that step), run and
    %store outputs, close the analysis (temporary file no longer accepting
    %values from future inputs (allows to be re-initialized I believe).
    d.openHydraulicAnalysis;
    d.initializeHydraulicAnalysis;
    %Run and close analysis
    Series = d.getComputedTimeSeries;
    d.closeHydraulicAnalysis
    if i ==1
        TankWaterAge = Series.NodeQuality(:,988:989);
    end
    if i ==2
        QualityComp = Series.NodeQuality
        ChlorineResidual = Series.NodeQuality(:,whereNames);
    end
end

%Time comes in seconds, change to hours for output
Time = (Series.Time) ./ 3600;

%First compute tank heads (comes as HGL, I want relative tank levels
%though)
TankHeads = Series.Head;
Elevations = d.getNodeElevations;
CalvaryTankLevel = TankHeads(:,988) - (tankMinLevel(1,1) + Elevations(1,988));
SpringfieldRoadTankLevel = TankHeads(:,989) - (tankMinLevel(1,2) + Elevations(1,989));

TankLevels = [CalvaryTankLevel, SpringfieldRoadTankLevel];

%Grab the relevant chlorine values

JunctionChlorine = ChlorineResidual;

%Grab the relevant pressure values
PressureComp = Series.Pressure;
JunctionPressure = PressureComp(:,whereNames);

%Lets get some pump information

%Springfield Road Pump Flow and head
%Convert the flows to GPM from CFS by multiplying by 448.83
FlowComp = (Series.Flow);
PumpFlowRate = FlowComp(:,1060:1061);

```

```

%Get the pump HGL

Heads = Series.Head;
PumpHeadIn = Heads(:, [986, 983]);
PumpHeadOut = Heads(:, [864, 979]);

PumpHGL = horzcat(PumpHeadIn, PumpHeadOut);

%Get the releveant junction demands

Demands = Series.Demand;
JunctionDemand = Demands(:, whereNames);

%TTHM, HAA5, and DBP formation in Lebanon, Kentucky

%modelling using chlorine demand (the difference between the chlorine
%concentration at a junction and the chlorine demand leaving the plant) The
%equation used may be found in Yogesh's thesis paper

JunctionChlorineDemand = Chlorine - JunctionChlorine;
JunctionTTHM = 0.0508 .* JunctionChlorineDemand;

%updating due to several file changes. Easiest way to do this would be to
%copy the above code, take new file (new tank and line) and update to
%account for these changes.

%{

elseif whatFile ==1
tankInfo = d.getNodeTankData;
tankMinLevel = [0,0,0];
tankMinLevel(1,1:2) = tankInfo.Minimum_Water_Level;
tankMinLevel(1,3) = tankMinLevel(1,1);
tankLevelSpecs = varargin{2};

tankInitialLevel = tankLevelSpecs + tankMinLevel;
pawn1 = tankInitialLevel;
tankInitialLevel(1,2) = pawn1(1,3);
tankInitialLevel(1,3) = pawn1(1,2);

d.setNodeTankInitialLevel(tankInitialLevel(1,2:3));

NodeNames = d.getNodeNameID;
Targets = ["244", "229", "618", "962", "1365", "1432", "J-135", "J-136", "J-137", "J-166"];
[~, whereNames] = ismember(Targets, NodeNames);

d.setNodeInitialQuality(993, Chlorine);
d.setNodeSourceQuality(993, Chlorine);

%Set the reaction coefficients

```

```

Bulk = varargin{5};
wall = varargin{6};

d.setLinkBulkReactionCoeff((0*d.getLinkBulkReactionCoeff) + Bulk);
d.setLinkWallReactionCoeff((0*d.getLinkWallReactionCoeff)+wall);

%Now set some time limitations

%Set the time specifications
Time4 = varargin{7};
Time5 = varargin{8};
Time6 = varargin{9};

d.setTimePatternStep(Time4 * 3600)
d.setTimeReportingStep(Time4 * 3600)
d.setTimeHydraulicStep(Time4 * 3600);
d.setTimeQualityStep(Time5 * 3600);
d.setTimeSimulationDuration(Time6 * 3600);

%% Run the extended Period Simulation Given the Above Info

%we need to run this twice in order to get water age as well as the
%chlorine info, they are seperate analysis in EPANET

for i = 1:2

    %on the second iteration change the quality analysis to chlorine
    if i ==2
        d.setQualityType('chem', 'Chlorine')
    end
    d.openHydraulicAnalysis;
    d.initializeHydraulicAnalysis;
    %Run and close analysis
    Series = d.getComputedTimeSeries;
    d.closeHydraulicAnalysis
    if i ==1
        TankWaterAge = Series.NodeQuality(:,995:996);
    end
    if i ==2
        ChlorineResidual = Series.NodeQuality(:,whereNames);
    end
end

Time = (Series.Time) ./ 3600;
%First compute tank heads

TankHeads = Series.Head;
Elevations = d.getNodeElevations;
CalvaryTankLevel = TankHeads(:,995) - (tankMinLevel(1,1) + Elevations(1,995));
SpringfieldRoadTankLevel = TankHeads(:,996) - (tankMinLevel(1,2) + Elevations(1,996));

TankLevels = [CalvaryTankLevel, SpringfieldRoadTankLevel];

```

```

%Grab the relevant chlorine values

JunctionChlorine = ChlorineResidual;

%Grab the relevant pressure values
JunctionPressure = Series.Pressure;
JunctionPressure = JunctionPressure(:,WhereNames);

%Lets get some pump information

%Springfield Road Pump Flow and head
%Convert the flows to GPM from CFS by multiplying by 448.83
Flows = 448.83.*(Series.Flow);
PumpFlowRate = Flows(:,1066:1067);

%Get the pump HGL

Heads = Series.Head;
PumpHeadIn = Heads(:,[988,982]);
PumpHeadOut = Heads(:,[981,989]);

PumpHGL = horzcat(PumpHeadIn, PumpHeadOut);

%Get the releveant junction demands

Demands = Series.Demand;
JunctionDemand = Demands(:, WhereNames);

%TTHM, HAA5, and DBP formation in Lebanon, Kentucky

%modelling using chlorine demand (the difference between the chlorine
%concentration at a junction and the chlorine demand leaving the plant) The
%equation used may be found in Yogesh's thesis paper

JunctionChlorineDemand = Chlorine - JunctionChlorine;
JunctionTTHM = 0.0508 .* JunctionChlorineDemand;
%}

end

% Code that executes after component creation
function startupFcn(app)
start_Toolkit;

end

function PlotButtonPushed(app, event)
    %Tells me which column to look in for the data we are seeking
    [~, InColumn] = ismember(app.EnterJunctionPressureEditField.Value,
app.UITable_2.ColumnName);
    %Plots the relevant data for pressure

```

```

x1 = app.UITable_2.Data(:,1);
y1 = app.UITable_2.Data(:,InColumn);
plot(app.UIAxes,x1,y1)

%Tells me which column to look in for the data we are seeking
[~, InColumn2] = ismember(app.EnterJunctionChlorineEditField.Value,
app.UITable_3.ColumnName);
%Plots the relevant data for chlorine residual
x2 = app.UITable_3.Data(:,1);
y2 = app.UITable_3.Data(:,InColumn2);
plot(app.UIAxes_2,x2,y2)

%Tells me which column to look in for the data we are seeking
[~, InColumn3] = ismember(app.EnterTankHeadEditField.Value,
app.UITable_4.ColumnName);
%Plots the relevant data for tank levels
x3 = app.UITable_4.Data(:,1);
y3 = app.UITable_4.Data(:,InColumn3);
plot(app.UIAxes_3,x3,y3)

end

% Button pushed function: GenerateGenericMapButton
function GenerateGenericMapButtonPushed(app, event)

%open EPANET and load Lebanon file
d = epanet('LebanonCurrent_July2023Testable.inp', 'loadfile');

NodeName = d.getNodeNameID;

%create figure template and change the name to match the system

fig = figure('Name','Lebanon water works - Layout of Lebanon Kentucky Water
Distribution System');
%plot the graph
[EdgesandNodes,fig] = d.plotDiGraph;

assignin("base","fig", fig);

%send this variable to the matlab workspace so it can be used
%later for mapping
assignin('base', 'EdgesandNodes', EdgesandNodes);

%give the figure a name

%Change the weight of the lines so we can see them a little
%better

fig.Linewidth = 2;

%Highlight the tanks and the pumps in the system and label them

```

```

    TankIndex = [988,989];

TankIndex = [988,989];
highlight(fig, TankIndex, "Marker", "s", "MarkerSize", 10, "NodeColor", "red");
PumpIndex = [864, 983];
highlight(fig, PumpIndex, "Marker", "<", "MarkerSize", 10, "NodeColor", "red");

text(fig.XData(983), fig.YData(983), NodeName(983), 'HorizontalAlignment', 'right',
'VerticalAlignment', 'top', 'Fontweight','bold', 'Color','magenta');
%This is for pump 8
text(fig.XData(864), fig.YData(864), "Pump 8", 'HorizontalAlignment', 'left',
'VerticalAlignment', 'top', 'Fontweight','bold', 'Color','magenta');
text(fig.XData(TankIndex), fig.YData(TankIndex), NodeName(TankIndex),
'HorizontalAlignment', 'right', 'VerticalAlignment', 'top',
'Fontweight','bold','Color','magenta');

%place all of the sizes of the pipes into the image as well
Diameter = d.getLinkDiameter;

%I don't think this is being used but won't delete for now until sure
%LinkNames = d.getLinkNameID;

%I am only doing the following two lines because edgesandnodes is storing
%two columns worth of data as 1 column in a table and I want them seperated
%for access
EdgesandNodes= EdgesandNodes.Edges(:,1);
Edges = splitvars(EdgesandNodes, 1);
Edges = table2array(Edges);

%because the plotdi graph is kind of a pain in the butt is gives the proper
%nodal connections (the right start and end nodes) but it doesn't give them
%as indexed values with the pipes, they are just sorted in ascending order
%in the firs column as apposed to associating them with their respective
%pipes. The following corrects that

ConnectDiameter = d.getLinkNodesIndex;

%Connect is indexed with the pipes but not proper order of start and end
%nodes, edges is the opposite. Therefore we can just sort them in ascending
%order in the columns, compare where sortConnect and sortEdges are
%(Ordering the Edges variable to be properly indexed with pipes) and use
%those values to change the highlight for the digrpah

sortConnect = sort(ConnectDiameter, 2, 'ascend');
sortEdges = sort(Edges, 2, 'ascend');

[~,index] = ismember(sortConnect, sortEdges, 'rows');

```

```

%assign in the matlab workspace for other map processing purposes. Here
%this is for indexing Link names so we can label them on the map.

assignin('base', 'index',index);

EdgesNew = Edges(index,:);
%assign in the matlab workspace for other map processing purposes. Here
%this is for labeling pipes.
assignin('base','EdgesNew',EdgesNew);

%Highlight for different sizes

twoInch = find(Diameter ==2);
fourInch = find(Diameter ==4);
sixInch = find(Diameter ==6);
eightInch = find(Diameter ==8);
tenInch = find(Diameter ==10);
twelveInch = find(Diameter ==12);
sixteenInch = find(Diameter ==16);
twentyInch = find(Diameter ==20);

two = EdgesNew(twoInch(1,:),:);
four = EdgesNew(fourInch(1,:),:);
six = EdgesNew(sixInch(1,:),:);
eight = EdgesNew(eightInch(1,:),:);
ten = EdgesNew(tenInch(1,:),:);
twelve = EdgesNew(twelveInch(1,:),:);
sixteen = EdgesNew(sixteenInch(1,:),:);
twenty = EdgesNew(twentyInch(1,:),:);

fig.Linewidth = 2;

%change the colors of the pipes to be properly matched
highlight(fig, two(:,1), two(:,2), "EdgeColor", "green");

highlight(fig, four(:,1), four(:,2), "EdgeColor", "cyan");

highlight(fig, six(:,1), six(:,2), "EdgeColor", "red");

highlight(fig, eight(:,1), eight(:,2), "EdgeColor", "yellow");

highlight(fig, ten(:,1), ten(:,2), "EdgeColor", "magenta");

highlight(fig, twelve(:,1), twelve(:,2), "EdgeColor", "blue");

highlight(fig, sixteen(:,1), sixteen(:,2), "EdgeColor", "black");

```



```

highlight(fig, twenty(:,1), twenty(:,2), "EdgeColor", "white");

end

% Button pushed function:
% GenerateNodalPressureandPipeFlowsMapButton
function GenerateNodalPressureandPipeFlowsMapButtonPushed(app, event)

%first we need to pull in some of the results from functions
%that were already ran

%pull the workspace variables in for processing in the map
PressureComp = evalin('base', 'PressureComp');
FlowComp = evalin('base', 'FlowComp');
EdgesandNodes = evalin('base', 'EdgesandNodes');
fig = evalin('base', 'fig');
d = evalin('base', 'd');

%first define the thresholds

Threshold1 = app.HighPressurepsiEditField.Value;
Threshold2 = app.MediumPressurepsiEditField.Value;
Threshold3 = app.LowPressurepsiEditField.Value;

%now define how we want these beauties to be colored

junctionColor1 = string(app.Color1DropDown.Value);
junctionColor2 = string(app.Color2DropDown.Value);
junctionColor3 = string(app.Color3DropDown.Value);
junctionColor4 = string(app.Color4DropDown.Value);

%was EPS selected?

if app.EPSCheckBox.Value ==0

%EPS was not selected so we are running our colors as "steady
%state
pressures = PressureComp(1,:);

    HPressure = find(pressures > Threshold1)';
    MPressure = find(pressures <= Threshold1 & pressures > Threshold2)';
    LPressure = find(pressures <= Threshold2 & pressures > Threshold3)';
    IllegalPressure = find(pressures <= Threshold3)';

    highlight(fig, HPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor1);

    highlight(fig, MPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",

```

```

junctionColor2);

    highlight(fig, LPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor3);

    highlight(fig, IllegalPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor4);

elseif app.EPSCheckBox.Value ==1
    %EPS was selected
    Time = (app.EPSHourSlider.Value)+1;
    Time = round(Time);
    pressures = PressureComp(Time,:);

    HPressure = find(pressures > Threshold1)';
    MPressure = find(pressures <= Threshold1 & pressures > Threshold2)';
    LPressure = find(pressures <= Threshold2 & pressures > Threshold3)';
    IllegalPressure = find(pressures <= Threshold3)';

    highlight(fig, HPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor1);

    highlight(fig, MPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor2);

    highlight(fig, LPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor3);

    highlight(fig, IllegalPressure, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor4);

end

```

```

PipeColor5 = string(app.Color5DropDown.Value);
PipeColor6 = string(app.Color6DropDown.Value);
PipeColor7 = string(app.Color7DropDown.Value);

```

```

EdgesandNodes= EdgesandNodes.Edges(:,1);
Edges = splitvars(EdgesandNodes, 1);
Edges = table2array(Edges);

```

```

%because the plotdi graph is kind of a pain in the butt is gives the proper
%nodal connections (the right start and end nodes) but it doesn't give them
%as indexed values with the pipes, they are just sorted in ascending order
%in the first column as apposed to associating them with their respective
%pipes. The following corrects that

```

```

ConnectDiameter = d.getLinkNodesIndex;

```

```

%Connect is indexed with the pipes but not proper order of start and end
%nodes, edges is the opposite. Therefore we can just sort them in ascending
%order in the columns, compare where sortConnect and sortEdges are

```

```

%(ordering the Edges variable to be properly indexed with pipes) and use
%those values to change the highlight for the digrph

sortConnect = sort(ConnectDiameter, 2, 'ascend');
sortEdges = sort(Edges, 2, 'ascend');

[~,index] = ismember(sortConnect, sortEdges, 'rows');

EdgesNew = Edges(index,:);

%for map processing purposes, pass this value into the base workspace
assignin('base', 'index',index);

%Highlight for different flow regimes

Threshold4 = app.HighFlowcfsEditField_2.Value;
Threshold5 = app.LowFlowcfsEditField_2.Value;

if app.EPSCheckBox_2.Value ==0
FlowComp = abs(FlowComp ./ 448.8);
HFlow = find(FlowComp(1,:) > Threshold4);
MFlow = find(FlowComp(1,:) <= Threshold4 & FlowComp(1,:) > Threshold5);
LFlow = find(FlowComp(1,:) <= Threshold5);

try
LowFlow = EdgesNew(LFlow(1,:),:);
MediumFlow = EdgesNew(MFlow(1,:),:);
HighFlow = EdgesNew(HFlow(1,:),:);
catch
end

%change the colors of the pipes to be properly matched
try
highlight(fig, LowFlow(:,1), LowFlow(:,2), "EdgeColor", PipeColor7);
highlight(fig, MediumFlow(:,1), MediumFlow(:,2), "EdgeColor", PipeColor6);
highlight(fig, HighFlow(:,1), HighFlow(:,2), "EdgeColor", PipeColor5);

catch
end

end

if app.EPSCheckBox_2.Value ==1
Time2 = (app.EPSHourSlider_2.Value)+1;
Time2 = round(Time2);

%Need to convert flow to CFS
FlowComp = abs(FlowComp ./ 448.8);
HFlow = find(FlowComp(Time2,:) > Threshold4);
MFlow = find(FlowComp(Time2,:) <= Threshold4 & FlowComp(Time2,:) > Threshold5);
LFlow = find(FlowComp(Time2,:) <= Threshold5);

try

```

```

LowFlow = EdgesNew(LFlow(1,:),:);
MediumFlow = EdgesNew(MFlow(1,:),:);
HighFlow = EdgesNew(HFlow(1,:),:);
catch
end

%change the colors of the pipes to be properly matched
try
    highlight(fig, LowFlow(:,1), LowFlow(:,2), "EdgeColor", PipeColor7);
    highlight(fig, MediumFlow(:,1), MediumFlow(:,2), "EdgeColor", PipeColor6);
    highlight(fig, HighFlow(:,1), HighFlow(:,2), "EdgeColor", PipeColor5);

catch
end

end

end

% Button pushed function: GenerateNodalChlorineResidualsMapButton
function GenerateNodalChlorineResidualsMapButtonPushed(app, event)

    %Pull in the relevant chlorine data and figure
    qualityComp = evalin('base', 'QualityComp');
    fig = evalin('base', 'fig');
    %define thresholds from the main map page
    Threshold7 = app.HighConcentrationmgIEditField.Value;
    Threshold8 = app.LowConcentrationmgIEditField.Value;
    %define some of the colors we will be using
    junctionColor7 = string(app.Color8DropDown.Value);
    junctionColor8 = string(app.Color9DropDown.Value);
    junctionColor9 = string(app.Color10DropDown.Value);

    if app.EPSCheckBox_3.Value ==0

        %EPS was not selected so we are running our colors as "steady
        %state
        quality = QualityComp(1,:);
        HResidual = find(quality > Threshold7)';
        MResidual = find(quality <= Threshold7 & quality > Threshold8)';
        LResidual = find(quality <= Threshold8)';

        try
            highlight(fig, HResidual, "Marker", "o", "MarkerSize", 5, "NodeColor",
            junctionColor7);

            highlight(fig, MResidual, "Marker", "o", "MarkerSize", 5, "NodeColor",
            junctionColor8);

            highlight(fig, LResidual, "Marker", "o", "MarkerSize", 5, "NodeColor",
            junctionColor9);
        catch
        end

        elseif app.EPSCheckBox_3.Value ==1

```

```

        %EPS was selected
        Time3 = (app.EPSHourSlider_3.Value)+1;
        Time3 = round(Time3);
        quality = QualityComp(Time3,:);

        HResidual = find(quality > Threshold7)';
        MResidual = find(quality <= Threshold7 & quality > Threshold8)';
        LResidual = find(quality <= Threshold8)';

    try
        highlight(fig, HResidual, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor7);

        highlight(fig, MResidual, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor8);

        highlight(fig, LResidual, "Marker", "o", "MarkerSize", 5, "NodeColor",
junctionColor9);
    catch
    end

    end

end

% Button pushed function: RunDiscoveryButton
function RunDiscoveryButtonPushed(app, event)

    %bring in the figure from matlab 'base' workspace

    fig = evalin('base', 'fig');
    index = evalin('base', 'index');
    d = evalin('base', 'd');
    EdgesNew = evalin('base', 'EdgesNew')

    NodeName = d.getNodeNameID;
    LinkName = d.getLinkNameID;
    LinkNew = LinkName;

    JunctionName = string(app.JunctionNameEditField.Value);
    PipeName = string(app.PipeNameEditField.Value);

    JunctionIndex = find(strcmp(NodeName, JunctionName))

    %lets get the junctions all squared away
    if isempty(app.JunctionNameEditField.Value) ==1
        %means that there is no text
        fig.NodeLabel = [];
    elseif isempty(app.JunctionNameEditField.Value) ==0
        %means that someone put in some text. This will error if
        %the text is not exactly right. I will not fix that at the
        %moment

```

```

        %we also need the weight and color wanted for the junction
        %index

        highlight(fig, JunctionIndex, "Marker", "o", "MarkerSize",
app.JunctionWeightEditField.Value, "NodeColor", string(app.JunctionColorDropDown.Value));
    end

    %Now lets figure out the pipe labeling

    %first index it
    PipeIndex = find(strcmp(LinkName, PipeName))

    %now use the reordered edges to find what you are looking for

    WhatEdge = EdgesNew(PipeIndex,:);

    %now run the what if statement

    if isempty(app.PipeNameEditField.Value) ==1
        %means that there is no text
        fig.EdgeLabel = [];
    elseif isempty(app.PipeNameEditField.Value) ==0
        %means that someone put in some text. This will error if
        %the text is not exactly right. I will not fix that at the
        %moment
        highlight(fig, WhatEdge(:,1), WhatEdge(:,2), "EdgeColor",
string(app.PipeColorDropDown.Value), "Linewidth", app.PipeLineweightEditField.Value);
    end

    %Now reorder the LinkName to match the indexing necessary for
    %proper pipe labeling

    LinkNew(1,index) = LinkName;

    %Turn on all of the junction names if that option is selected,
    %turn on all pipe names if that option is selected

    if app.TurnOnAllPipeNamesCheckBox.Value ==1
        fig.EdgeLabel = LinkNew;
    elseif app.TurnOnAllPipeNamesCheckBox.Value ==0
        fig.EdgeLabel = [];
    end

    if app.TurnonAllJunctionNamesCheckBox.Value ==1
        fig.NodeLabel = NodeName;
    end

```

```

elseif app.TurnonAllJunctionNamesCheckBox.Value ==0
    fig.NodeLabel = [];
end

end

% Callback function
function GetCurrentFilePipeDescriptionButtonPushed(app, event)
    d = evalin('base', 'd');
    LinkInfo = d.getLinksInfo;
    LinkName = d.getLinkNameID;

    %if the table is empty fill these puppies in

    if isempty(app.UITable2.Data) == 1
        app.UITable2.Data(:,2) = LinkInfo.LinkLength;
        app.UITable2.Data(:,3) = LinkInfo.LinkDiameter;
        app.UITable2.Data(:,4) = LinkInfo.LinkRoughnessCoeff;
        app.UITable2.Data(:,7) = LinkInfo.NodesConnectingLinksIndex(:,1);
        app.UITable2.Data(:,8) = LinkInfo.NodesConnectingLinksIndex(:,2);
        app.UITable2.Data = num2cell(app.UITable2.Data);

        app.UITable2.Data(:,1) = LinkName;

    else
        %its already populated and you just need to replace the roughness
        %coefficients
        Pawn = num2cell(LinkInfo.LinkRoughnessCoeff);
        app.UITable2.Data(:,4) = Pawn;

    end

end

end

% FUNCTION NOT IN CURRENT APP VERSION
function ShowCurrentPumpCurvesButtonPushed(app, event)
    %%All of this is found in the KYPIPE "BASIC Computer Program for the
    %%Analysis of Pressure and Flow in Pipe Distribution Systems Including
    %%Extended Period Simulations" By Don wood and is the same formulation for
    %%the pump curves in EPANET

    %This is the pump calibration function and will be used to help visualize,
    %and optimize the use of their pumps

    d = evalin('base', 'd');
    Pump = d.getCurveValue;

    %Pull in some of the operating point data relevant for plotting later

```

```

%Get Pressure Data (Index)
app.UITable_2.ColumnName
wherePump6PO = find(strcmp('O-Pump-6', app.UITable_2.ColumnName));
wherePump7PO = find(strcmp('O-Pump-7', app.UITable_2.ColumnName));
wherePump8PO = find(strcmp('O-Pump-8', app.UITable_2.ColumnName));

wherePump6PI = find(strcmp('I-Pump-6', app.UITable_2.ColumnName));
wherePump7PI = find(strcmp('I-Pump-7', app.UITable_2.ColumnName));
wherePump8PI = find(strcmp('I-Pump-8', app.UITable_2.ColumnName));
%Get Flow Data (Index)
wherePump6F = find(strcmp('~@Pump-6', app.UITable_5.ColumnName));
wherePump7F = find(strcmp('~@Pump-7', app.UITable_5.ColumnName));
wherePump8F = find(strcmp('~@Pump-8', app.UITable_5.ColumnName));

%Now we can gather that data

%This is for the Heads of the pumps at the Inlet and outlet (convert to head from
%psi)

Pump6HeadO = app.UITable_2.Data(:,wherePump6PO) .*2.30725;
Pump7HeadO = app.UITable_2.Data(:,wherePump7PO) .*2.30725;
Pump8HeadO = app.UITable_2.Data(:,wherePump8PO) .*2.30725;

Pump6HeadI = app.UITable_2.Data(:,wherePump6PI) .*2.30725;
Pump7HeadI = app.UITable_2.Data(:,wherePump7PI) .*2.30725;
Pump8HeadI = app.UITable_2.Data(:,wherePump8PI) .*2.30725;

%Because we are not solving the system head curve explicitly we need to
%actually solve for the amount of head that the pump is adding to the
%system to determine its operating point.

Pump6HeadAdded = Pump6HeadO - Pump6HeadI;
Pump7HeadAdded = Pump7HeadO - Pump7HeadI;
Pump8HeadAdded = Pump8HeadO - Pump8HeadI;

%This is for the flows (convert back to GPM from CFS)
Pump6Flow = (app.UITable_5.Data(:,wherePump6F)) .* (1/.00222802);
Pump7Flow = (app.UITable_5.Data(:,wherePump7F)) .* (1/.00222802);
Pump8Flow = (app.UITable_5.Data(:,wherePump8F)) .* (1/.00222802);

%This is to show whether or not we are using the slider. We add a plus 1
%because the row that we search in for hour 1 values is actually in row 2
%(this is because there is an hour 0 reported value.

if app.OperatingPointSlider_5.value > 0
Time3 = (app.OperatingPointSlider_5.Value)+1; %Pump 8
Time3 = round(Time3);
else
    Time3 = 1;
end
if app.OperatingPointSlider_3.value >0
Time4 = (app.OperatingPointSlider_3.Value)+1; %Pump 6
Time4 = round(Time4);
else

```



```

    Time4 = 1;
end
if app.OperatingPointSlider_4.Value >0
Time5 = (app.OperatingPointSlider_4.Value)+1; %Pump 7
Time5 = round(Time5);
else
    Time5 = 1;
end

%%This is for the Lake Pump
PumpCurve1 = cell2mat(Pump(1,1));

%plot the first portion of the line
n = (log( (PumpCurve1(1,2) - PumpCurve1(3,2)) / (PumpCurve1(1,2) -PumpCurve1(2,2)) ))
/(log(PumpCurve1(3,1) / PumpCurve1(2,1)));

C = (PumpCurve1(1,2) - PumpCurve1(2,2)) / (PumpCurve1(2,1)^n);

Q = linspace(0,PumpCurve1(3,1), 150);

EP = PumpCurve1(1,2) - (C * (Q.^n));

%plot the last extension of the line (the straight line portion coming
%after Q3.

S = (-n) * C * (PumpCurve1(3,1)^(n));
A = PumpCurve1(3,2) - (S * PumpCurve1(3,1));

EP2 = 50;

%end when the plotted curve ends up running into the x-axis
add = (PumpCurve1(3,1) * .10);
Q2 = PumpCurve1(3,1);

%This tells me when (maybe inefficiently) the flows will result in 0 head
while EP2 > 0
Q2 = Q2 + add;
EP2 = A + (S .* Q2);
end

%I can then linearly interpolate these flows to give me a line between the
%last known flow and the flow we predict will result in 0 head and then
%plot them.

QA11 = horzcat(Q, Q2(1,2:end));
EPA11 = horzcat(EP, EP2(1,2:end));

plot(app.UIAxes2_7, QA11, EPA11);
hold(app.UIAxes2_7, 'on');
scatter(app.UIAxes2_7, Pump6Flow(Time4, 1), Pump6HeadAdded(Time4,1), 'filled');
hold(app.UIAxes2_7, 'off');
%%This is for the River Pump

```

```

PumpCurve1 = cell2mat(Pump(1,2));

%plot the first portion of the line
n = (log( (PumpCurve1(1,2) - PumpCurve1(3,2)) / (PumpCurve1(1,2) -PumpCurve1(2,2)) ))
/(log(PumpCurve1(3,1) / PumpCurve1(2,1)));

C = (PumpCurve1(1,2) - PumpCurve1(2,2)) / (PumpCurve1(2,1)^n);

Q = linspace(0,PumpCurve1(3,1), 150);

EP = PumpCurve1(1,2) - (C * (Q.^n));

%plot the last extension of the line (the straight line portion coming
%after Q3.

S = (-n) * C * (PumpCurve1(3,1)^(n-1));
A = PumpCurve1(3,2) - (S * PumpCurve1(3,1));

EP2 = 50;

%end when the plotted curve ends up running into the x-axis
add = (PumpCurve1(3,1) * .10);
Q2 = PumpCurve1(3,1);

%This tells me when (maybe inefficiently) the flows will result in 0 head
while EP2 > 0
Q2 = Q2 + add;
EP2 = A + (S .* Q2);
end

%I can then linearly interpolate these flows to give me a line between the
%last known flow and the flow we predict will result in 0 head and then
%plot them.

QA11 = horzcat(Q, Q2(1,2:end));
EPA11 = horzcat(EP, EP2(1,2:end));

plot(app.UIAxes2_4, QA11, EPA11);
hold(app.UIAxes2_4, 'on');
scatter(app.UIAxes2_4, Pump8Flow(Time3, 1), Pump8HeadAdded(Time3,1), 'filled');
hold(app.UIAxes2_4, 'off');
%%This is for the North Tank Pump
PumpCurve1 = cell2mat(Pump(1,3));

%plot the first portion of the line
n = (log( (PumpCurve1(1,2) - PumpCurve1(3,2)) / (PumpCurve1(1,2) -PumpCurve1(2,2)) ))
/(log(PumpCurve1(3,1) / PumpCurve1(2,1)));

C = (PumpCurve1(1,2) - PumpCurve1(2,2)) / (PumpCurve1(2,1)^n);

Q = linspace(0,PumpCurve1(3,1), 150);

```

```

EP = PumpCurve1(1,2) - (C * (Q.^n));

%plot the last extension of the line (the straight line portion coming
%after Q3.

S = (-n) * C * (PumpCurve1(3,1)^(n-1));
A = PumpCurve1(3,2) - (S * PumpCurve1(3,1));

EP2 = 50;

%end when the plotted curve ends up running into the x-axis
add = (PumpCurve1(3,1) * .10);
Q2 = PumpCurve1(3,1);

%This tells me when (maybe inefficiently) the flows will result in 0 head
while EP2 > 0
Q2 = Q2 + add;
EP2 = A + (S .* Q2);
end

%I can then linearly interpolate these flows to give me a line between the
%last known flow and the flow we predict will result in 0 head and then
%plot them.

QA11 = horzcat(Q, Q2(1,2:end));
EPA11 = horzcat(EP, EP2(1,2:end));

plot(app.UIAxes2_6, QA11, EPA11);
hold(app.UIAxes2_6, 'on');
scatter(app.UIAxes2_6, Pump7Flow(Time5, 1), Pump7HeadAdded(Time5,1), 'filled');
hold(app.UIAxes2_6, 'off');

end

% FUNCTION NOT IN CURRENT APP VERSION
function PlaceFieldDataInGraphButtonPushed(app, event)

FieldData = evalin('base', 'fieldData1');

HeadData = app.EnterFieldDataHeadFeetEditField.Value;
FlowData = app.EnterFieldDataFlowGPMeditField.Value;
NewData = horzcat(HeadData, FlowData);

FieldData = vertcat(FieldData, NewData);
fieldData1 = FieldData;

assignin('base','fieldData1', fieldData1)

```

```

        hold(app.UIAxes2_4, 'on');
S1 = scatter(app.UIAxes2_4, fieldData1(:,2), fieldData1(:,1), 'magenta');
assignin('base','S1', S1)
hold(app.UIAxes2_4, 'off');

    end

% FUNCTION NOT IN CURRENT APP VERSION
function PlaceFieldDatainGraphButton_2Pushed(app, event)
    FieldData = evalin('base', 'fieldData2');

    HeadData = app.EnterFieldDataHeadFeetEditField_2.Value;
    FlowData = app.EnterFieldDataFlowGPMeditField_2.Value;
    NewData = horzcat(HeadData, FlowData);

    FieldData = vertcat(FieldData, NewData);
    fieldData2 = FieldData;

    assignin('base','fieldData2', fieldData2)

    hold(app.UIAxes2_7, 'on');
S2 = scatter(app.UIAxes2_7, fieldData2(:,2), fieldData2(:,1), 'magenta');
assignin('base','S2', S2)
hold(app.UIAxes2_7, 'off');
    end

% FUNCTION NOT IN CURRENT APP VERSION
function PlaceFieldDatainGraphButton_3Pushed(app, event)
    FieldData = evalin('base', 'fieldData3');

    HeadData = app.EnterFieldDataHeadFeetEditField_3.Value;
    FlowData = app.EnterFieldDataFlowGPMeditField_3.Value;
    NewData = horzcat(HeadData, FlowData);

    FieldData = vertcat(FieldData, NewData);
    fieldData3 = FieldData;

    assignin('base','fieldData3', fieldData3)

    hold(app.UIAxes2_6, 'on');
S3 = scatter(app.UIAxes2_6, fieldData3(:,2), fieldData3(:,1), 'magenta');
    assignin('base','S3', S3)
hold(app.UIAxes2_6, 'off');
    end

% FUNCTION NOT IN CURRENT APP VERSION
function RemoveFieldDatafromGraphButtonPushed(app, event)

    delete(findobj(app.UIAxes2_4,'type', 'scatter'))
    fieldData1 = [];
    assignin('base', 'fieldData1', fieldData1);

end

```

```

% FUNCTION NOT IN CURRENT APP VERSION
function RemoveFieldDatafromGraphButton_2Pushed(app, event)

    delete(findobj(app.UIAxes2_7,'type', 'scatter'))
    fieldData2 = [];
    assignin('base', 'fieldData2', fieldData2);
end

% Callback function
function RemoveFieldDatafromGraphButton_3Pushed(app, event)

    delete(findobj(app.UIAxes2_6,'type', 'scatter'))
    fieldData3 = [];
    assignin('base', 'fieldData3', fieldData3);
end

% FUNCTION NOT IN CURRENT APP VERSION
function ChangeRoughnessforAllPipesButtonPushed(app, event)
    NewRoughnessValues = num2cell(app.CFactorEditField_3.Value);

    app.UITable2.Data(:,4) = NewRoughnessValues;
end

% FUNCTION NOT IN CURRENT APP VERSION
function ChangeRoughnessforPipesofSameDiameterButtonPushed(app, event)
    SizeOfPipeSpecified = app.SizeofPipeEditField.Value;
    NewRoughness = num2cell(app.CFactorEditField.Value);

    DataFromTable = cell2mat(app.UITable2.Data(:,3));

    WhatRows = find(DataFromTable == SizeOfPipeSpecified);
    app.UITable2.Data(WhatRows,4) = NewRoughness;
end

% FUNCTION NOT IN CURRENT APP VERSION
function Button_3Pushed(app, event)
    %Read in the EPANET file
    d = evalin('base', 'd');

    %Find where the names of the rows match
    NameEnteredByUser = string(app.ReferencePipeNameEditField.Value);

    WhatRows = find(strcmp(app.UITable2.Data(:,1), NameEnteredByUser));

    %Now that we know what row contains the value of the reference
    %pipe we can go ahead and find out what the starting node is of
    %that pipe

    WhatNode = app.UITable2.Data(WhatRows, 7);

    WhatNode = round(cell2mat(WhatNode));

    %Now we can go and grab the relevant coordinates for the node

```

```

NodeCoordinates = d.getNodeCoordinates;

NodeXCoordinates = NodeCoordinates{1,1};
NodeYCoordinates = NodeCoordinates{1,2};

XCoordsMyNode = NodeXCoordinates(1,WhatNode);
YCoordsMyNode = NodeYCoordinates(1,WhatNode);

%Now we need to find any pipes that are within the specified
%radius of the node (here we are loosely defining radius, this
%is actually generating a box)

terminalValue = app.RadiusftEditField.Value;

XCoordLeft = XCoordsMyNode - terminalValue;
XCoordRight = XCoordsMyNode + terminalValue;

YCoordUp = YCoordsMyNode + terminalValue;
YCoordsDown = YCoordsMyNode - terminalValue;

%Now we can find all of the starting nodes where the
%coordinates match the specified conditions

WhatStartNodes = find(NodeXCoordinates < XCoordRight & NodeXCoordinates >
XCoordLeft & NodeYCoordinates > YCoordsDown & NodeYCoordinates < YCoordUp);

%We know now which starting nodes qualify for change of
%roughness value, we just need to do so
StartNodes = round(cell2mat(app.UITable2.Data(:,7)));

%Find the places where StartNodes is contained within
%WhatStartNodes, Index by finding where the
%WherePipeFromStartNode variable is not equal to 0

[~,WherePipeFromStartNode] = ismember(StartNodes, WhatStartNodes);

IndicestoUpdate = find(WherePipeFromStartNode ~=0);
NewRoughness = num2cell(app.CFactorEditField_2.Value);
app.UITable2.Data(IndicestoUpdate,4) = NewRoughness;

end

% FUNCTION NOT IN CURRENT APP VERSION
function Button_4Pushed(app, event)
    %Find where the names of the rows match
    NameEnteredByUser = string(app.PipeNameEditField_2.Value);

```

```

        WhatRows = find(strcmp(app.UITable2.Data(:,1), NameEnteredByUser));
        NewRoughness = num2cell(app.CFactorEditField_4.Value);
        app.UITable2.Data(WhatRows,4) = NewRoughness;
    end

% FUNCTION NOT IN CURRENT APP VERSION
function RunEPSwithTrialRoughnessValuesButtonPushed(app, event)

%First we have to identify where some of the changes came from,
%we do this by comparing the original roughness values with
%the new values placed into the table

d = evalin('base', 'd');

LinkInfo = d.getLinksInfo;
%Get the roughness values from the original file
LinkRoughOrig = LinkInfo.LinkRoughnessCoeff';
UserInput = cell2mat(app.UITable2.Data(:,4));

%find where the values are equal to each other (where there
%have been no changes)
Index = (LinkRoughOrig == UserInput);

%Now that we have this we can actually find the index of where
%the changes have happened

IndexedChangeInRough = find(Index == 0);

%we have this and now we can look for the roughness values of
%the change

ChangedRough = UserInput(IndexedChangeInRough,1);

%Package these together as a neat input

NewRoughness = horzcat(IndexedChangeInRough, ChangedRough);

tic;

%find the row that you are wanting to concatenate and create a
%cell array. Join them and remove redundant spaces. Once this
%is done for all 12 rows we combine them into a another
%completed cell array with the combined strings. Side note, the
%deblank function removes the trailing whitespace produced and
%the regexp function searches for any whitespace that is
%over 1 space long and replaces it with whitespace that is in
%fact 1 space long

row1 = {app.DropDown.Value,
app.DropDown_3.Value, app.DropDown_4.Value, app.DropDown_5.Value, app.D
ropDown_6.Value, app.D
ropDown_7.Value, app.DropDown_8.Value, app.EditField.Value, app.DropDown_86.Value};
Newr1 = strjoin(row1, " ");

```

%This expression takes the joined string "Newr1" and finds where
%the joined string has 1 or more spaces (' +') and replaces it
%with a single space. deblank removes any trailing space.

```
Newer1 = deblank(regexprep(Newr1, ' +', ' '));
```

```
row2 = {app.DropDown_9.Value,  
app.DropDown_10.Value,app.DropDown_11.Value,app.DropDown_12.Value,app.DropDown_13.Value,  
pp.DropDown_14.Value,app.DropDown_15.Value,app.EditField_2.Value, app.DropDown_87.Value};  
Newr2 = strjoin(row2, " ");  
Newer2 = deblank(regexprep(Newr2, ' +', ' '));
```

```
row3 = {app.DropDown_16.Value,  
app.DropDown_17.Value,app.DropDown_18.Value,app.DropDown_19.Value,app.DropDown_20.Value,  
pp.DropDown_21.Value,app.DropDown_22.Value,app.EditField_3.Value, app.DropDown_88.Value};  
Newr3 = strjoin(row3, " ");  
Newer3 = deblank(regexprep(Newr3, ' +', ' '));
```

```
row4 = {app.DropDown_23.Value,  
app.DropDown_24.Value,app.DropDown_25.Value,app.DropDown_26.Value,app.DropDown_27.Value,  
pp.DropDown_28.Value,app.DropDown_29.Value,app.EditField_4.Value, app.DropDown_89.Value};  
Newr4 = strjoin(row4, " ");  
Newer4 = deblank(regexprep(Newr4, ' +', ' '));
```

```
row5 = {app.DropDown_30.Value,  
app.DropDown_31.Value,app.DropDown_32.Value,app.DropDown_33.Value,app.DropDown_34.Value,  
pp.DropDown_35.Value,app.DropDown_36.Value,app.EditField_5.Value, app.DropDown_90.Value};  
Newr5 = strjoin(row5, " ");  
Newer5 = deblank(regexprep(Newr5, ' +', ' '));
```

```
row6 = {app.DropDown_37.Value,  
app.DropDown_38.Value,app.DropDown_39.Value,app.DropDown_40.Value,app.DropDown_41.Value,  
pp.DropDown_42.Value,app.DropDown_43.Value,app.EditField_6.Value, app.DropDown_91.Value};  
Newr6 = strjoin(row6, " ");  
Newer6 = deblank(regexprep(Newr6, ' +', ' '));
```

```
row7 = {app.DropDown_44.Value,  
app.DropDown_45.Value,app.DropDown_46.Value,app.DropDown_47.Value,app.DropDown_48.Value,  
pp.DropDown_49.Value,app.DropDown_50.Value,app.EditField_7.Value, app.DropDown_92.Value};  
Newr7 = strjoin(row7, " ");  
Newer7 = deblank(regexprep(Newr7, ' +', ' '));
```

```
row8 = {app.DropDown_51.Value,  
app.DropDown_52.Value,app.DropDown_53.Value,app.DropDown_54.Value,app.DropDown_55.Value,  
pp.DropDown_56.Value,app.DropDown_57.Value,app.EditField_8.Value, app.DropDown_93.Value};  
Newr8 = strjoin(row8, " ");  
Newer8 = deblank(regexprep(Newr8, ' +', ' '));
```

```
row9 = {app.DropDown_58.Value,  
app.DropDown_59.Value,app.DropDown_60.Value,app.DropDown_61.Value,app.DropDown_62.Value,  
pp.DropDown_63.Value,app.DropDown_64.Value,app.EditField_9.Value, app.DropDown_94.Value};  
Newr9 = strjoin(row9, " ");  
Newer9 = deblank(regexprep(Newr9, ' +', ' '));
```

```
row10 = {app.DropDown_65.Value,
```



```

app.DropDown_66.Value,app.DropDown_67.Value,app.DropDown_68.Value,app.DropDown_69.Value,
app.DropDown_70.Value,app.DropDown_71.Value,app.EditField_10.Value,
app.DropDown_95.Value};
    Newr10 = strjoin(row10, " ");
    Newer10 = deblank(regexprep(Newr10, ' +', ' '));

    row11 = {app.DropDown_72.Value,
app.DropDown_73.Value,app.DropDown_74.Value,app.DropDown_75.Value,app.DropDown_76.Value,
app.DropDown_77.Value,app.DropDown_78.Value,app.EditField_11.Value,
app.DropDown_96.Value};
    Newr11 = strjoin(row11, " ");
    Newer11 = deblank(regexprep(Newr11, ' +', ' '));

    row12 = {app.DropDown_79.Value,
app.DropDown_80.Value,app.DropDown_81.Value,app.DropDown_82.Value,app.DropDown_83.Value,
app.DropDown_84.Value,app.DropDown_85.Value,app.EditField_12.Value,
app.DropDown_97.Value};
    Newr12 = strjoin(row12, " ");
    Newer12 = deblank(regexprep(Newr12, ' +', ' '));

    %Finally, bring them altogether

    NextArray = {Newer1;Newer2; Newer3; Newer4;Newer5; Newer6; Newer7;Newer8;
Newer9; Newer10;Newer11; Newer12};

    %uses cell function to apply the isempty function to every row in the cell
    %array. Once it identifies the rows that are empty in Next Array it deletes
    %them. Found on Mathworks open forums. Essentially, all of the inputs we
    %are specifying here will be sent to the extended period function as a cell
    %value and then inside of the Extended period function we will convert to
    %the proper format so that the epanet-matlab toolkit can process correctly.
    %If this is not done we will be thrown errors.
    NextArray(cellfun('isempty',NextArray)) = cellstr('NULL');

    % now that we have the control statements, we need to add the other
    % factors that go into the simulation

    %InitialTankLevels
    out13 = app.T_1.Value;
    out14 = app.T_2.Value;

    TankLevels = {out13; out14};

    %Times
    out15 = app.TotalTimeEditField.Value; %Total Time
    out16 = app.HydraulicTimeStepEditField.Value; %Hydraulic Time Step
    out17 = app.WaterQualityTimeStepEditField.Value; %Water Quality Time Step

    Time1 = {out15};
    Time2 = {out16};
    Time3 = {out17};

    %Decay Rates
    out18 = app.BulkEditField.Value;
    out19 = app.WallEditField.Value;

```

```

Decay = {out18; out19};

%Chlorine value
out20 = app.ChlorinemglEditField.Value;
Chlorine = {out20};

%Combine all of them
Quality2 = vertcat(NextArray, TankLevels, Time1,Time2, Time3, Decay, Chlorine);

[d, QualityComp,PressureComp, FlowComp,pressure, quality, flow,Headfinal, PressureName,
QualityName, FlowName, Message] = ExtendedPeriod_Lebanon(Quality2, NewRoughness);

%so that the mapping can use these variables later
assignin('base', 'PressureComp', PressureComp);
assignin('base', 'FlowComp', FlowComp);
assignin('base', 'QualityComp', QualityComp);

%This code can probably be made simpler (with for loops for example with
%the row information. I did not think this was necessary because the system
%is rather small and am only writing a few control statements. Will have to
%check the data limitations of for looping, I didn't like how it wrote and
%rewrote the matrix for every iteration, that seemed slow. Maybe my method
%is slow too here.

app.UITable_2.ColumnName = horzcat('Time (Hours)',PressureName);
app.UITable_2.Data = pressure;

app.UITable_3.ColumnName = horzcat('Time (Hours)',QualityName);
app.UITable_3.Data = quality;

app.UITable_4.ColumnName = {'Time (Hours)', 'T-1', 'T-2'};
app.UITable_4.Data = Headfinal;

app.UITable_5.ColumnName = horzcat('Time (Hours)',FlowName);
app.UITable_5.Data = flow;

app.TextArea_5.Value = Message;

stop = toc;

app.EditField_13.Value = stop;

%This is all of the stuff I want assigned into the base workspace for
%several other functions. This will be overwritten if I run this function
%again which is good! If they want a different sim to be ran then I would
%want a lot of this stuff to change

%assign the epanet file into the workspace to be used for other functions
%in this application

```

```

assignin("base", "d", d);

%This is for pump calibration
fieldData1 = [];
fieldData2 = [];
fieldData3 = [];

assignin('base', 'fieldData1', fieldData1);
assignin('base', 'fieldData2', fieldData2);
assignin('base', 'fieldData3', fieldData3);

end

% FUNCTION NOT IN CURRENT APP VERSION
function SaveRoughnessChangesButtonPushed(app, event)
    filename = ('lebanon_May23.inp');
    d.saveInputFile(filename)
end

% FUNCTION NOT IN CURRENT APP VERSION
function RUNTRIALSIMWITHFIELDDATAButtonPushed(app, event)

%%First we need to bring in the field data as well as the matlab file
d = evalin('base', 'd');

FieldData1 = evalin('base', 'fieldData1');
FieldData2 = evalin('base', 'fieldData2');
FieldData3 = evalin('base', 'fieldData3');

if FieldData1 == [];
    FieldData1 = [0,0];
end

if FieldData2 == [];
    FieldData1 = [0,0];
end

if FieldData3 == [];
    FieldData1 = [0,0];
end

%Sort the data

FieldData1 = sortrows(FieldData1,1,'descend');
FieldData2 = sortrows(FieldData2,1,'descend');
FieldData3 = sortrows(FieldData3,1,'descend');

%Preallocate some curves to save on some processing speed. These will be
%populated and thrown into the extended period simulation

Curve1 = zeros(3,2);
Curve2 = zeros(3,2);

```

```

Curve3 = zeros(3,2);

%how many data points do we have total?
HowManyPoints1 = 1 + size(find(FieldData1(2:end,1) ~= 0),1);
HowManyPoints2 = 1 + size(find(FieldData2(2:end,1) ~= 0),1);
HowManyPoints3 = 1 + size(find(FieldData3(2:end,1) ~= 0),1);

%now place some logic in here to tell the button to change the curves if
%there is more than 3 points of field data, if there are not, do nothing.

%Also, I made a mistake, some of these are based on the head being in the
%first column and some are based on it being in the second column. This is
%kind of a caveman fix but we are just going to have to deal with it for
%now

pawn1 = FieldData1;
pawn2 = FieldData2;
pawn3 = FieldData3;

FieldData1(:,1) = pawn1(:,2);
FieldData1(:,2) = pawn1(:,1);

FieldData2(:,1) = pawn2(:,2);
FieldData2(:,2) = pawn2(:,1);

FieldData3(:,1) = pawn3(:,2);
FieldData3(:,2) = pawn3(:,1);

%also we need to remove any trailing 0's (from GPT)

% Find rows containing all zeros
rowsToRemove = all(FieldData1(4:end,:) == 0, 2);
% Remove the zero rows at the end
FieldData1 = FieldData1(1:end - sum(rowsToRemove), :);

% Find rows containing all zeros
rowsToRemove = all(FieldData2(4:end,:) == 0, 2);
% Remove the zero rows at the end
FieldData2 = FieldData2(1:end - sum(rowsToRemove), :);

% Find rows containing all zeros
rowsToRemove = all(FieldData3(4:end,:) == 0, 2);
% Remove the zero rows at the end
FieldData3 = FieldData3(1:end - sum(rowsToRemove), :);

if HowManyPoints1 >= 3

%now we can create many fitted lines in the data using Dr. Woods equation
%for pump curves and average all of these values then find the point that
%best fits that line and use it in setting the new pump curve

HowManyLines1 = HowManyPoints1 - 2;

```

```

Q = linspace(0,FieldData1(HowManyPoints1,1), 150);
i = 1;
EP = [];
while i <= HowManyLines1
n = (log( (FieldData1(1,2) - FieldData1(HowManyPoints1,2)) / (FieldData1(1,2) -
FieldData1((i+1),2)) )) / (log(FieldData1(HowManyPoints1,1) / FieldData1((i+1),1)));

C = (FieldData1(1,2) - FieldData1((i+1),2)) / (FieldData1((i+1),1)^n);

EPNew = FieldData1(1,2) - (C * (Q.^n));
EP = vertcat(EP, EPNew);

i = i+1;
end

%Now we get the average curve from the above run and turn it into a column
%vector

Average = mean(EP, 1)';

j =1;
closestPoint =[];
while j <= HowManyLines1

% Calculate distance between points and curve
distances = sqrt(sum((Average(j,1) - FieldData1((j+1),1)).^2, 2));

closestPoint = vertcat(closestPoint, distances)
j=j+1;
end

%which point is the keeper
[minClosestPoint, minCPIndex] = min(closestPoint);

%Now we can go ahead and fill some one of the curves

Curve1 = [FieldData1(1,1), FieldData1(1,2); FieldData1((minCPIndex+1),1),
FieldData1((minCPIndex+1),2); FieldData1((HowManyPoints1),1),
FieldData1((HowManyPoints1),2) ];

elseif HowManyPoints1 < 3
%do nothing
end

%do the same thing for the lake pump

if HowManyPoints2 >= 3

```

```

%now we can create many fitted lines in the data using Dr. Woods equation
%for pump curves and average all of these values then find the point that
%best fits that line and use it in setting the new pump curve

HowManyLines2 = HowManyPoints2 - 2;
Q = linspace(0,FieldData2(HowManyPoints2,1), 150);
i = 1;
EP = [];
while i <= HowManyLines1
n = (log( (FieldData2(2,2) - FieldData2(HowManyPoints2,2)) / (FieldData2(1,2) -
FieldData2((i+1),2)) )) / (log(FieldData2(HowManyPoints2,1) / FieldData2((i+1),1)));

C = (FieldData2(1,2) - FieldData2((i+1),2)) / (FieldData2((i+1),1)^n);

EPNew = FieldData2(1,2) - (C * (Q.^n));
EP = vertcat(EP, EPNew);

i = i+1;
end

Average = mean(EP, 1)';

j =1;
closestPoint =[];
while j <= HowManyLines2

distances = sqrt(sum((Average(j,1) - FieldData2((j+1),1)).^2, 2));

closestPoint = vertcat(closestPoint, distances)
j=j+1;
end

[minclosestPoint, minCPIndex] = min(closestPoint);

Curve2 = [FieldData2(1,1), FieldData2(1,2); FieldData1((minCPIndex+1),1),
FieldData2((minCPIndex+1),2); FieldData2((HowManyPoints2),1),
FieldData2((HowManyPoints2),2) ];

elseif HowManyPoints2 < 3

end

%now we can do the same for the north tank pump

if HowManyPoints3 >= 3

%now we can create many fitted lines in the data using Dr. Woods equation
%for pump curves and average all of these values then find the point that
%best fits that line and use it in setting the new pump curve

```

```

HowManyLines3 = HowManyPoints3 - 2;
Q = linspace(0,FieldData3(HowManyPoints3,1), 150);
i = 1;
EP = [];
while i <= HowManyLines3
n = (log( (FieldData3(2,2) - FieldData3(HowManyPoints3,2)) / (FieldData3(1,2) -
FieldData3((i+1),2)) )) / (log(FieldData3(HowManyPoints3,1) / FieldData3((i+1),1)));

C = (FieldData3(1,2) - FieldData3((i+1),2)) / (FieldData3((i+1),1)^n);

EPNew = FieldData3(1,2) - (C * (Q.^n));
EP = vertcat(EP, EPNew);

i = i+1;
end

Average = mean(EP, 1)';

j =1;
closestPoint =[];
while j <= HowManyLines3

distances = sqrt(sum((Average(j,1) - FieldData3((j+1),1)).^2, 2));

closestPoint = vertcat(closestPoint, distances)
j=j+1;
end

[minClosestPoint, minCPIndex] = min(closestPoint);

Curve3 = [FieldData3(1,1), FieldData3(1,2); FieldData3((minCPIndex+1),1),
FieldData3((minCPIndex+1),2); FieldData3((HowManyPoints3),1),
FieldData3((HowManyPoints3),2) ];

elseif HowManyPoints3 < 3

end

%Now we need to run an extended period simulation with these points

AllCurves = horzcat(Curve1, Curve2, Curve3);

tic;

```

```

%find the row that you are wanting to concatenate and create a
%cell array. Join them and remove redundant spaces. Once this
%is done for all 12 rows we combine them into a another
%completed cell array with the combined strings. Side note, the
%deblank function removes the trailing whitespace produced and
%the regexprep function searches for any whitespace that is
%over 1 space long and replaces it with whitespace that is in

```

`%fact 1 space long`

```
row1 = {app.DropDown.Value,  
app.DropDown_3.Value,app.DropDown_4.Value,app.DropDown_5.Value,app.DropDown_6.Value,app.D  
ropDown_7.Value,app.DropDown_8.Value,app.EditField.Value, app.DropDown_86.Value};  
Newr1 = strjoin(row1, " ");  
%This expression takes the joined string "Newr1" and finds where  
%the joined string has 1 or more spaces ( ' +') and replaces it  
%with a single space. deblank removes any trailing space.  
Newer1 = deblank(regexprep(Newr1, ' +', ' '));  
  
row2 = {app.DropDown_9.Value,  
app.DropDown_10.Value,app.DropDown_11.Value,app.DropDown_12.Value,app.DropDown_13.Value,a  
pp.DropDown_14.Value,app.DropDown_15.Value,app.EditField_2.Value, app.DropDown_87.Value};  
Newr2 = strjoin(row2, " ");  
Newer2 = deblank(regexprep(Newr2, ' +', ' '));  
  
row3 = {app.DropDown_16.Value,  
app.DropDown_17.Value,app.DropDown_18.Value,app.DropDown_19.Value,app.DropDown_20.Value,a  
pp.DropDown_21.Value,app.DropDown_22.Value,app.EditField_3.Value, app.DropDown_88.Value};  
Newr3 = strjoin(row3, " ");  
Newer3 = deblank(regexprep(Newr3, ' +', ' '));  
  
row4 = {app.DropDown_23.Value,  
app.DropDown_24.Value,app.DropDown_25.Value,app.DropDown_26.Value,app.DropDown_27.Value,a  
pp.DropDown_28.Value,app.DropDown_29.Value,app.EditField_4.Value, app.DropDown_89.Value};  
Newr4 = strjoin(row4, " ");  
Newer4 = deblank(regexprep(Newr4, ' +', ' '));  
  
row5 = {app.DropDown_30.Value,  
app.DropDown_31.Value,app.DropDown_32.Value,app.DropDown_33.Value,app.DropDown_34.Value,a  
pp.DropDown_35.Value,app.DropDown_36.Value,app.EditField_5.Value, app.DropDown_90.Value};  
Newr5 = strjoin(row5, " ");  
Newer5 = deblank(regexprep(Newr5, ' +', ' '));  
  
row6 = {app.DropDown_37.Value,  
app.DropDown_38.Value,app.DropDown_39.Value,app.DropDown_40.Value,app.DropDown_41.Value,a  
pp.DropDown_42.Value,app.DropDown_43.Value,app.EditField_6.Value, app.DropDown_91.Value};  
Newr6 = strjoin(row6, " ");  
Newer6 = deblank(regexprep(Newr6, ' +', ' '));  
  
row7 = {app.DropDown_44.Value,  
app.DropDown_45.Value,app.DropDown_46.Value,app.DropDown_47.Value,app.DropDown_48.Value,a  
pp.DropDown_49.Value,app.DropDown_50.Value,app.EditField_7.Value, app.DropDown_92.Value};  
Newr7 = strjoin(row7, " ");  
Newer7 = deblank(regexprep(Newr7, ' +', ' '));  
  
row8 = {app.DropDown_51.Value,  
app.DropDown_52.Value,app.DropDown_53.Value,app.DropDown_54.Value,app.DropDown_55.Value,a  
pp.DropDown_56.Value,app.DropDown_57.Value,app.EditField_8.Value, app.DropDown_93.Value};  
Newr8 = strjoin(row8, " ");  
Newer8 = deblank(regexprep(Newr8, ' +', ' '));  
  
row9 = {app.DropDown_58.Value,
```



```

app.DropDown_59.Value,app.DropDown_60.Value,app.DropDown_61.Value,app.DropDown_62.Value,app.DropDown_63.Value,app.DropDown_64.Value,app.EditField_9.Value, app.DropDown_94.Value};
Newr9 = strjoin(row9, " ");
Newer9 = deblank(regexprep(Newr9, ' +', ' '));

row10 = {app.DropDown_65.Value,
app.DropDown_66.Value,app.DropDown_67.Value,app.DropDown_68.Value,app.DropDown_69.Value,app.DropDown_70.Value,app.DropDown_71.Value,app.EditField_10.Value,
app.DropDown_95.Value};
Newr10 = strjoin(row10, " ");
Newer10 = deblank(regexprep(Newr10, ' +', ' '));

row11 = {app.DropDown_72.Value,
app.DropDown_73.Value,app.DropDown_74.Value,app.DropDown_75.Value,app.DropDown_76.Value,app.DropDown_77.Value,app.DropDown_78.Value,app.EditField_11.Value,
app.DropDown_96.Value};
Newr11 = strjoin(row11, " ");
Newer11 = deblank(regexprep(Newr11, ' +', ' '));

row12 = {app.DropDown_79.Value,
app.DropDown_80.Value,app.DropDown_81.Value,app.DropDown_82.Value,app.DropDown_83.Value,app.DropDown_84.Value,app.DropDown_85.Value,app.EditField_12.Value,
app.DropDown_97.Value};
Newr12 = strjoin(row12, " ");
Newer12 = deblank(regexprep(Newr12, ' +', ' '));

%Finally, bring them altogether

NextArray = {Newer1;Newer2; Newer3; Newer4;Newer5; Newer6; Newer7;Newer8;
Newer9; Newer10;Newer11; Newer12};

%uses cell function to apply the isempty function to every row in the cell
%array. Once it identifies the rows that are empty in Next Array it deletes
%them. Found on Mathworks open forums. Essentially, all of the inputs we
%are specifying here will be sent to the extended period function as a cell
%value and then inside of the Extended period function we will convert to
%the proper format so that the epanet-matlab toolkit can process correctly.
%If this is not done we will be thrown errors.
NextArray(cellfun('isempty',NextArray)) = cellstr('NULL');

% now that we have the control statements, we need to add the other
% factors that go into the simulation

%InitialTankLevels
out13 = app.T_1.Value;
out14 = app.T_2.Value;

TankLevels = {out13; out14};

%Times
out15 = app.TotalTimeEditField.Value; %Total Time
out16 = app.HydraulicTimeStepEditField.Value; %Hydraulic Time Step
out17 = app.WaterQualityTimeStepEditField.Value; %Water Quality Time Step

```

```

Time1 = {out15};
Time2 = {out16};
Time3 = {out17};
%Decay Rates
out18 = app.BulkEditField.Value;
out19 = app.WallEditField.Value;

Decay = {out18; out19};

%Chlorine Value
out20 = app.ChlorinemglEditField.Value;
Chlorine = {out20};

%Combnine all of them
Quality2 = vertcat(NextArray, TankLevels, Time1,Time2, Time3, Decay, Chlorine);

[d, QualityComp,PressureComp, FlowComp,presure, quality, flow,Headfinal, PressureName,
QualityName, FlowName, Message] = ExtendedPeriod_Lebanon(Quality2, AllCurves);

%so that the mapping can use these variables later
assignin('base', 'PressureComp', PressureComp);
assignin('base', 'FlowComp', FlowComp);
assignin('base', 'QualityComp', QualityComp);

%This code can probably be made simpler (with for loops for example with
%the row information. I did not think this was necessary because the system
%is rather small and am only writing a few control statements. Will have to
%check the data limitations of for looping, I didn't like how it wrote and
%rewrote the matrix for every iteration, that seemed slow. Maybe my method
%is slow too here.

app.UITable_2.ColumnName = horzcat('Time (Hours)',PressureName);
app.UITable_2.Data = pressure;

app.UITable_3.ColumnName = horzcat('Time (Hours)',QualityName);
app.UITable_3.Data = quality;

app.UITable_4.ColumnName = {'Time (Hours)', 'T-1', 'T-2'};
app.UITable_4.Data = Headfinal;

app.UITable_5.ColumnName = horzcat('Time (Hours)',FlowName);
app.UITable_5.Data = flow;

app.TextArea_5.Value = Message;

stop = toc;

app.EditField_13.Value = stop;

%This is all of the stuff I want assigned into the base workspace for

```

```

%several other functions. This will be overwritten if I run this function
%again which is good! If they want a different sim to be ran then I would
%want a lot of this stuff to change

%assign the epanet file into the workspace to be used for other functions
%in this application

assignin("base", "d", d);

end

% FUNCTION NOT IN CURRENT APP VERSION
function ReadPumpFieldDataFromExcelFileRecommendedButtonPushed(app, event)
%Read the data in from excel file
rawTable = readtable('Curves.xlsx');
Curve1 = rawTable(:,1:2);
Curve2 = rawTable(:,3:4);
Curve3 = rawTable(:,5:6);

%Turn any NaN's to 0
Curve1(isnan(Curve1))=0;
Curve2(isnan(Curve2))=0;
Curve3(isnan(Curve3))=0;

%For the River Pump
fieldData1 = Curve1;
hold(app.UIAxes2_4, 'on');
S1 = scatter(app.UIAxes2_4, fieldData1(:,2), fieldData1(:,1), 'magenta');
hold(app.UIAxes2_4, 'off');
assignin('base', 'fieldData1', fieldData1);

%For the Lake Pump
fieldData2 = Curve2;
hold(app.UIAxes2_7, 'on');
S2 = scatter(app.UIAxes2_7, fieldData2(:,2), fieldData2(:,1), 'magenta');
hold(app.UIAxes2_7, 'off');
assignin('base', 'fieldData2', fieldData2);

%For the North Tank Pump
fieldData3 = Curve3;
hold(app.UIAxes2_6, 'on');
S3 = scatter(app.UIAxes2_6, fieldData3(:,2), fieldData3(:,1), 'magenta');
hold(app.UIAxes2_6, 'off');
assignin('base', 'fieldData3', fieldData3);
end

% Callback function
function STORENEWPUMPCURVESButtonPushed(app, event)
    filename = ('lebanon_May23.inp');
    d.saveInputFile(filename)
end

% Button pushed function: RUNEPSButton
function RUNEPSButtonPushed(app, event)

```

```
%Clear axes on the results page (necessary if there is data still
%in them from a previous simulation)
```

THIS IS THE MAIN SCRIPT FOR THE EPS RUNNING THE APPLICATION

```
cla(app.UIAxes_4)
cla(app.UIAxes_5)

cla(app.UIAxes_7)
cla(app.UIAxes_8)
cla(app.UIAxes_9)

%Pull in the demand factors so that they can be used in the
%simulation
DZ1 = evalin('base', 'DZ1');
DZ2 = evalin('base', 'DZ2');
DemandFactors = [DZ1, DZ2];

%%Get the initial tank levels

SpringfieldTank = app.InitialTankLevelEditField_9.Value;
CalvaryTanks = app.InitialTankLevelEditField_10.Value;

InitialTankLevels = [CalvaryTanks, SpringfieldTank];

if strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==1 &&
strcmp(app.WTPSwitch.Value, 'Use') ==1
    % Set the WTP to respond to tank levels. Note that this
    % needs to be written in a specific format for EPANET
    % to use. Hence the variable "Row1" etc.

    %Turn on when
    Value1 = (app.OnWhenBelowEditField_2.Value) + 27;
    Row1 = 'LINK ~@Pump-8 OPEN IF NODE T-12 BELOW ';
    combinedString1 = sprintf('%s%d', Row1, Value1);

    %Turn off when
    Value2 = (app.OffWhenAboveEditField_2.Value) + 27;
    Row2 = 'LINK ~@Pump-8 CLOSED IF NODE T-12 ABOVE ';
    combinedString2 = sprintf('%s%d', Row2, Value2);

    % Do the same for the Springfield Road pump

    %Turn on when
    Value3 = (app.OnWhenBelowEditField.Value) + 104.5;
    Row3 = 'LINK ~@Pump-7 OPEN IF NODE T-13 BELOW ';
    combinedString3 = sprintf('%s%d', Row3, Value3);

    %Turn off when
    Value4 = (app.OffWhenAboveEditField.Value) + 104.5;
    Row4 = 'LINK ~@Pump-7 CLOSED IF NODE T-13 ABOVE ';
    combinedString4 = sprintf('%s%d', Row4, Value4);
end
```

```

if strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==0 &&
strcmp(app.WTPSwitch.Value, 'Use') ==0

    %If neither settings were specified as "Use" call
    %"NULL" and delete later when pushed into the extended
    %period simulation

    combinedString1 = 'NULL';
    combinedString2 = 'NULL';
    combinedString3 = 'NULL';
    combinedString4 = 'NULL';
end

if strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==1 &&
strcmp(app.WTPSwitch.Value, 'Use') ~= 1

    %If Springfield Road is used and WTP is not, do the
    %following:

    %Do not use WTP pumps (relative to tank
    %levels)
    combinedString1 = 'NULL';
    combinedString2 = 'NULL';

    %Turn on when
    Value3 = (app.OnWhenBelowEditField.Value) + 104.5;
    Row3 = 'LINK ~@Pump-7 OPEN IF NODE T-13 BELOW ';
    combinedString3 = sprintf('%s%d', Row3, value3);

    %Turn off when
    Value4 = (app.OffWhenAboveEditField.Value) + 104.5;
    Row4 = 'LINK ~@Pump-7 CLOSED IF NODE T-13 ABOVE ';
    combinedString4 = sprintf('%s%d', Row4, value4);
end

if strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ~=1 &&
strcmp(app.WTPSwitch.Value, 'Use') == 1

    %If WTP is used and Springfield Road is not, do the
    %following:

    %Do not use Springfield Road pump (relative to tank
    %levels)
    combinedString3 = 'NULL';
    combinedString4 = 'NULL';

    %Turn on when
    Value1 = (app.OnWhenBelowEditField_2.Value) + 27;
    Row1 = 'LINK ~@Pump-8 OPEN IF NODE T-12 BELOW ';
    combinedString1 = sprintf('%s%d', Row1, value1);

    %Turn off when
    Value2 = (app.OffWhenAboveEditField_2.Value) + 27;

```

```

        Row2 = 'LINK ~@Pump-8 CLOSED IF NODE T-12 ABOVE ';
        combinedString2 = sprintf('%s%d', Row2, value2);
    end

%Initialize the string used for the control statements here because that
%makes inputting them into EPANET (via EPANET-MATLAB Toolkit) much easier.
%At least in my experience.

SpringfieldPumpStringOPEN = 'LINK ~@Pump-7 OPEN AT TIME ';
SpringfieldPumpStringCLOSED = 'LINK ~@Pump-7 CLOSED AT TIME ';
WTPPumpStringOPEN = 'LINK ~@Pump-8 OPEN AT TIME ';
WTPPumpStringCLOSED = 'LINK ~@Pump-8 CLOSED AT TIME ';

%Now read in all of the values

%Springfield Road on times
spring1 = app.ONATTIMEEditField.Value;
spring2 = app.ONATTIMEEditField_5.Value;
spring3 = app.ONATTIMEEditField_4.Value;
spring4 = app.ONATTIMEEditField_3.Value;
spring5 = app.ONATTIMEEditField_6.Value;
spring6 = app.ONATTIMEEditField_7.Value;
spring7 = app.ONATTIMEEditField_8.Value;
spring8 = app.ONATTIMEEditField_9.Value;

%Springfield Road off times
spring9 = app.OFFATTIMEEditField.Value;
spring10 = app.OFFATTIMEEditField_2.Value;
spring11 = app.OFFATTIMEEditField_3.Value;
spring12 = app.OFFATTIMEEditField_4.Value;
spring13 = app.OFFATTIMEEditField_5.Value;
spring14 = app.OFFATTIMEEditField_6.Value;
spring15 = app.OFFATTIMEEditField_7.Value;
spring16 = app.OFFATTIMEEditField_8.Value;

%WTP on times
WTP1 = app.ONATTIMEEditField_10.Value;
WTP2 = app.ONATTIMEEditField_13.Value;
WTP3 = app.ONATTIMEEditField_12.Value;
WTP4 = app.ONATTIMEEditField_11.Value;
WTP5 = app.ONATTIMEEditField_14.Value;
WTP6 = app.ONATTIMEEditField_15.Value;
WTP7 = app.ONATTIMEEditField_16.Value;
WTP8 = app.ONATTIMEEditField_17.Value;

%WTP off times
WTP9 = app.OFFATTIMEEditField_9.Value;
WTP10 = app.OFFATTIMEEditField_12.Value;
WTP11 = app.OFFATTIMEEditField_11.Value;
WTP12 = app.OFFATTIMEEditField_10.Value;
WTP13 = app.OFFATTIMEEditField_13.Value;
WTP14 = app.OFFATTIMEEditField_14.Value;
WTP15 = app.OFFATTIMEEditField_15.Value;

```

```
WTP16 = app.OFFATTIMEEditField_16.Value;
```

```
%create the springfield pump open control statements as they are recognized
```

```
%in EPANET
```

```
combinedString5 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring1);  
combinedString6 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring2);  
combinedString7 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring3);  
combinedString8 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring4);  
combinedString9 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring5);  
combinedString10 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring6);  
combinedString11 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring7);  
combinedString12 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring8);
```

```
%create the springfield pump closed control statements as they are recognized
```

```
%in EPANET
```

```
combinedString13 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring9);  
combinedString14 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring10);  
combinedString15 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring11);  
combinedString16 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring12);  
combinedString17 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring13);  
combinedString18 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring14);  
combinedString19 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring15);  
combinedString20 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring16);
```

```
%create the WTP pump open control statements as they are recognized
```

```
%in EPANET
```

```
combinedString21 = sprintf('%s%s', WTPPumpStringOPEN, WTP1);  
combinedString22 = sprintf('%s%s', WTPPumpStringOPEN, WTP2);  
combinedString23 = sprintf('%s%s', WTPPumpStringOPEN, WTP3);  
combinedString24 = sprintf('%s%s', WTPPumpStringOPEN, WTP4);  
combinedString25 = sprintf('%s%s', WTPPumpStringOPEN, WTP5);  
combinedString26 = sprintf('%s%s', WTPPumpStringOPEN, WTP6);  
combinedString27 = sprintf('%s%s', WTPPumpStringOPEN, WTP7);  
combinedString28 = sprintf('%s%s', WTPPumpStringOPEN, WTP8);
```

```
%create the WTP pump closed control statements as they are recognized
```

```
%in EPANET
```

```
combinedString29 = sprintf('%s%s', WTPPumpStringCLOSED, WTP9);  
combinedString30 = sprintf('%s%s', WTPPumpStringCLOSED, WTP10);  
combinedString31 = sprintf('%s%s', WTPPumpStringCLOSED, WTP11);  
combinedString32 = sprintf('%s%s', WTPPumpStringCLOSED, WTP12);  
combinedString33 = sprintf('%s%s', WTPPumpStringCLOSED, WTP13);  
combinedString34 = sprintf('%s%s', WTPPumpStringCLOSED, WTP14);  
combinedString35 = sprintf('%s%s', WTPPumpStringCLOSED, WTP15);  
combinedString36 = sprintf('%s%s', WTPPumpStringCLOSED, WTP16);
```

```
%Combine all control statements under one variable
```

```
AllSimpleControls = {combinedString1;
```

```
combinedString2;combinedString3;combinedString4;combinedString5;combinedString13;combined  
String6;combinedString14;combinedString7;combinedString15;combinedString8;combinedString1  
6;combinedString9;combinedString17;combinedString10;combinedString18;combinedString11;com  
binedString19;combinedString12;combinedString20;combinedString21;combinedString29;  
combinedString22; combinedString30;combinedString23; combinedString31;combinedString24;  
combinedString32;combinedString25; combinedString33;combinedString26;
```

```
combinedString34;combinedString27; combinedString35;combinedString28; combinedString36};
```

```
% If either of the "USE" buttons are specified, just use the first four  
% control statements
```

```
if strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==1 ||  
strcmp(app.WTPSwitch.Value, 'Use') ==1
```

```
    AllSimpleControls = {combinedString1;  
combinedString2;combinedString3;combinedString4};  
end
```

```
%If a cell is populated with just the string, that means no user input  
%was specified. Here this is tagged and deleted. This is done for all  
%four possible statements.
```

```
Logic1 = strcmp(SpringfieldPumpStringOPEN, AllSimpleControls);  
whereLogic1 = find(Logic1 ==1);  
AllSimpleControls(whereLogic1, :) = [];
```

```
Logic2 = strcmp(SpringfieldPumpStringCLOSED, AllSimpleControls);  
whereLogic2 = find(Logic2 ==1);  
AllSimpleControls(whereLogic2, :) = [];
```

```
Logic3 = strcmp(WTPPumpStringOPEN, AllSimpleControls);  
whereLogic3 = find(Logic3 ==1);  
AllSimpleControls(whereLogic3, :) = [];
```

```
Logic4 = strcmp(WTPPumpStringCLOSED, AllSimpleControls);  
whereLogic4 = find(Logic4 ==1);  
AllSimpleControls(whereLogic4,:) = [];
```

```
%NULL is also deleted
```

```
Logic5 = strcmp('NULL', AllSimpleControls);  
whereLogic5 = find(Logic5 ==1);  
AllSimpleControls(whereLogic5, :) = [];
```

```
[PressureComp, FlowComp, QualityComp, TankHeads, TankWaterAge, PumpHGL, PumpFlowRate,  
JunctionPressure, JunctionDemand, JunctionChlorine, JunctionTTHM, Time, d] =  
ExtendedPeriodV2(0,InitialTankLevels,  
AllSimpleControls,app.ChlorinemglEditField_2.value,app.BulkEditField_2.Value,  
app.wallEditField_2.Value, app.HydraulicTimeStepEditField_2.Value,  
app.waterQualityTimeStepEditField_2.Value, app.TotalTimeEditField_2.Value,  
DemandFactors);
```

```
%Tank names accidentally reversed relative to elements to be filled in the  
%application!
```

```
Pawn1 = TankHeads;  
TankHeads(:,1) = Pawn1(:,2);  
TankHeads(:,2) = Pawn1(:,1);
```



```

app.UITable4.Data = horzcat(Time,TankHeads);

%Set the column name for the table
app.UITable4.ColumnName = ["Time(hours)","Springfield Tank","Calvary Tanks"];

```

User can specify which parameter they are looking for

```

%This is for the Springfield Road Pump
if strcmp(app.ShowDropDown.Value, 'HGL (ft.)') ==1
    Pump1Info = PumpHGL(:,3);
else
    Pump1Info = PumpFlowRate(:,1);
end
%for the WTP Pump
if strcmp(app.ShowDropDown_2.Value, 'HGL (ft.)') ==1
    Pump2Info = PumpHGL(:,4);
else
    Pump2Info = PumpFlowRate(:,2);
end

%for the junctions themselves
if strcmp(app.ShowPredictedDropDown_8.Value, 'Pressure (psi)') ==1
    JunctionInfo = JunctionPressure;
elseif strcmp(app.ShowPredictedDropDown_8.Value, 'Demand (gpm)') ==1
    JunctionInfo = JunctionDemand;
elseif strcmp(app.ShowPredictedDropDown_8.Value, 'Chlorine (mg/l)') ==1
    JunctionInfo = JunctionChlorine;
else
    JunctionInfo = JunctionTTHM;
end

%This has been packaged nicely now into the tables. The tables are the
%elements referenced for the graphs to be populated

app.UITable4_2.Data = horzcat(Time, Pump1Info, Pump2Info, JunctionInfo);

%Place results in workspace so that the graphs may be used
assignin('base', 'TankHeads', TankHeads);
assignin('base', 'TankWaterAge', TankWaterAge);
assignin('base', 'PumpHGL', PumpHGL);
assignin('base', 'PumpFlowRate', PumpFlowRate);
assignin('base', 'JunctionPressure', JunctionPressure);
assignin('base', 'JunctionDemand', JunctionDemand);
assignin('base', 'JunctionChlorine', JunctionChlorine);
assignin('base', 'JunctionTTHM', JunctionTTHM);
assignin('base', 'Time', Time);

%Set the column name for the table that all of the graphs base themselves
%off of

app.UITable4_2.ColumnName = {'Time'; 'Springfield Road Pump'; 'Water Treatment Plant
Pump'; 'Route 208 By-Pass'; 'Before Calvary Meter'; 'Calvary Meter'; 'Woodlawn
Meter'; 'Danville Meter'; 'Springfield Road Meter'; 'Saint Rose Meter'; 'Saint Mary Meter';

```

```

'Campbellsville Meter';'Mercer Ave';'Indiana Creek Road'};

%For mapping later
%so that the mapping can use these variables later
assignin('base', 'PressureComp', PressureComp);
assignin('base', 'FlowComp', FlowComp);
assignin('base', 'QualityComp', QualityComp);
assignin('base', 'd', d);

%Use a message box to tell the user that the simulation has been run and
%that they may proceed to other evaluations
msgbox("EPS Successfully Ran, User May Proceed to Functionality On Next Page",
"Success");

```

```
end
```

```

% Button pushed function: UpdateGraphandTableButton
function UpdateGraphandTableButtonPushed(app, event)

```

```
    %Call in the tank values from the workspace
```

```

    TankHeads = evalin('base', 'TankHeads');
    Pawn1 = TankHeads;
    TankHeads(:,1) = Pawn1(:,2);
    TankHeads(:,2) = Pawn1(:,1);

```

```

    TankWaterAge = evalin('base', 'TankWaterAge');
    Time = evalin('base', 'Time');

```

```
    if strcmp(app.ShowPredictedDropDown.Value, 'Tank Level (ft.)') ==1
```

```

        %Set all of the information for the Springfield Road Tank
        app.UIAxes_4.YLabel.String = 'Tank Level (ft.)';
        plot(app.UIAxes_4, Time,TankHeads(:,2))

```

```
        app.UITable4.Data(:,2) = TankHeads(:,2);
```

```
    else
```

```

        %Set all of the information for the Springfield Road Tank
        app.UIAxes_4.YLabel.String = 'Water Age (Hours)';
        plot(app.UIAxes_4, Time,TankWaterAge(:,2))
        app.UITable4.Data(:,2) = TankWaterAge(:,2);

```

```
    end
```

```
end
```

```

% Button pushed function: UpdateGraphandTableButton_2
function UpdateGraphandTableButton_2Pushed(app, event)

```

```
    %Call in the tank values from the workspace
```

```

    TankHeads = evalin('base', 'TankHeads');
    Pawn1 = TankHeads;

```

```

    TankHeads(:,1) = Pawn1(:,2);
    TankHeads(:,2) = Pawn1(:,1);

    TankwaterAge = evalin('base', 'TankwaterAge');
    Time = evalin('base', 'Time');

    %Set all the information for the Calvary Tanks and the New tanks
    if strcmp(app.ShowPredictedDropDown_4.Value, 'Tank Level (ft.)') ==1

        %This means that we are using the old tanks here
        app.UIAxes_5.YLabel.String = 'Tank Level (ft.)';
        plot(app.UIAxes_5, Time, TankHeads(:,1))

        app.UITable4.Data(:,3) = TankHeads(:,1);

    else

        %Set all the information for the Calvary Tanks
        app.UIAxes_5.YLabel.String = 'Water Age (Hours)';
        plot(app.UIAxes_5, Time, TankwaterAge(:,1))

        app.UITable4.Data(:,3) = TankwaterAge(:,1);
    end

end

% Callback function

% Button pushed function: UpdateGraphandTableButton_4
function UpdateGraphandTableButton_4Pushed(app, event)
    PumpFlowRate = evalin('base', 'PumpFlowRate');
    PumpHGL = evalin('base', 'PumpHGL');
    Time = evalin('base', 'Time');
    %Now we need to update the springfield road information

    if strcmp(app.ShowDropDown.Value, 'HGL (ft.)')
        app.UITable4_2.Data(:,2) = PumpHGL(:,3);

        app.UIAxes_7.YLabel.String = 'HGL in ft.';
        plot(app.UIAxes_7, Time, PumpHGL(:,3))
        hold(app.UIAxes_7, 'on')
        plot(app.UIAxes_7, Time, PumpHGL(:,1))
        hold(app.UIAxes_7, 'off')

    else
        app.UIAxes_7.YLabel.String = 'Flow Rate (gpm)';
        app.UITable4_2.Data(:,2) = PumpFlowRate(:,1);
        plot(app.UIAxes_7, Time, PumpFlowRate(:,1))
    end
end

```

```

end

end

% Button pushed function: UpdateGraphandTableButton_5
function UpdateGraphandTableButton_5Pushed(app, event)
    PumpFlowRate = evalin('base', 'PumpFlowRate');
    PumpHGL = evalin('base', 'PumpHGL');
    Time = evalin('base', 'Time');
    %Now we need to update the springfield road information

    if strcmp(app.ShowDropDown_2.Value, 'HGL (ft.)')
        app.UITable4_2.Data(:,3) = PumpHGL(:,4);

        app.UIAxes_8.YLabel.String = 'HGL in ft.';
        plot(app.UIAxes_8, Time, PumpHGL(:,4))
        hold(app.UIAxes_8, 'on')
        plot(app.UIAxes_8, Time, PumpHGL(:,2))
        hold(app.UIAxes_8, 'off')

    else
        app.UIAxes_8.YLabel.String = 'Flow Rate (gpm)';
        app.UITable4_2.Data(:,3) = PumpFlowRate(:,2);
        plot(app.UIAxes_8, Time, PumpFlowRate(:,2))

    end

end

% Button pushed function: UpdateGraphandTableButton_6
function UpdateGraphandTableButton_6Pushed(app, event)

    %Read in all of the processed data from the simulation

    JunctionPressure = evalin('base', 'JunctionPressure');
    JunctionDemand = evalin('base', 'JunctionDemand');
    JunctionChlorine = evalin('base', 'JunctionChlorine');
    JunctionTTHM = evalin('base', 'JunctionTTHM');
    Time = evalin('base', 'Time');

    %first we must figure out what junction and parameter we are looking at

    Junction = app.AtJunctionDropDown.Value;
    Parameter = app.ShowPredictedDropDown_8.Value;

    %first we will update the table that represents the data

    if strcmp(Parameter, 'Pressure (psi)')
        app.UITable4_2.Data(:,4:14) = JunctionPressure;
    elseif strcmp(Parameter, 'Demand (gpm)')

```

```

        app.UITable4_2.Data(:,4:14) = JunctionDemand;
    elseif strcmp(Parameter, 'Chlorine (mg/l)')
        app.UITable4_2.Data(:,4:14) = JunctionChlorine;
    else
        app.UITable4_2.Data(:,4:14) = JunctionTTHM;
    end

    %Now we can access this data to update the UIAxes (index the
    %names to access the correct column)

    NameOfJunctions = ["Route 208 By-Pass","Before Calvary Meter", "Calvary
    Meter","Woodlawn Meter","Danville Meter","Springfield Road Meter", "Saint Rose Meter",
    "Saint Mary Meter", "Campbellsville Meter","Mercer Ave","Indiana Creek Road"];

    [~, whereISNameOfJunc] = ismember(Junction, NameOfJunctions);

    DataForSpecificJunction = app.UITable4_2.Data(:,(whereISNameOfJunc+3));

    app.UIAxes_9.YLabel.String = sprintf('%s', Parameter);
    plot(app.UIAxes_9, Time, DataForSpecificJunction);

end

```

```

% FUNCTION USED FOR DEMAND CALIBRATION PROCESS

```

```

%Lines made here to draw readers attention

```

```

function DemandButtonTestPushed(app, event)

```

```

%%Gather the initial tank levels

```

```

SpringfieldTank = app.InitialTanklevelEditField_9.Value;
CalvaryTanks = app.InitialTanklevelEditField_10.Value;
NewTank = app.InitialTanklevelEditField_11.Value;

```

```

InitialTankLevels = [CalvaryTanks, SpringfieldTank,NewTank];

```

```

%properly

```

```

%Look at the pump operations specific to the tanks
%for springfield road, cavalry tanks, and new tank respectively

```

```

%Not really sure why && is working instead of what I thought it
%should be which is ||. Weird

```

```

    if strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==0 &&
    strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==1
        elseif strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==0

```

```

        combinedString1 = 'NULL';
        combinedString2 = 'NULL';
elseif strcmp(app.SpringfieldRoadPumpSwitch.Value, 'Use') ==1
    Value1 = (app.OnwhenBelowEditField.Value) + 104.5;
    Row1 = 'LINK ~@Pump-7 OPEN IF NODE T-13 BELOW ';
    combinedString1 = sprintf('%s%d', Row1, Value1);

    Value2 = (app.OffwhenAboveEditField.Value) + 104.5;
    Row2 = 'LINK ~@Pump-7 CLOSED IF NODE T-13 ABOVE ';
    combinedString2 = sprintf('%s%d', Row2, Value2);
end

```

%These are the conditions specified for Cavalry Tank

```

        if strcmp(app.WTPumpSwitch_4.Value, 'Use') ==1 &&
        strcmp(app.WTPumpSwitch_4.Value, 'Use') ==0

elseif strcmp(app.WTPumpSwitch_4.Value, 'Use') ==0
    combinedString3 = 'NULL';
    combinedString4 = 'NULL';
elseif strcmp(app.WTPumpSwitch_4.Value, 'Use') ==1
    Value1 = (app.OnwhenBelowEditField_2.Value) +27;
    Row1 = 'LINK ~@Pump-7 OPEN IF NODE T-12 BELOW ';
    combinedString3 = sprintf('%s%d', Row1, Value1);

    Value2 = (app.OffwhenAboveEditField_2.Value) +27;
    Row2 = 'Link ~@Pump-7 CLOSED IF NODE T-12 ABOVE ';
    combinedString4 = sprintf('%s%d', Row2, Value2);

end

```

%These are the conditions specified for New Tank

```

        if strcmp(app.SpringfieldRoadPumpSwitch_3.Value, 'Use') ==1 &&
        strcmp(app.SpringfieldRoadPumpSwitch_3.Value, 'Use') ==0

elseif strcmp(app.SpringfieldRoadPumpSwitch_3.Value, 'Use') ==0
    combinedString5 = 'NULL';
    combinedString6 = 'NULL';
elseif strcmp(app.SpringfieldRoadPumpSwitch_3.Value, 'Use') ==1
    Value1 = (app.OnwhenBelowEditField_3.Value)+104.5;
    Row1 = 'LINK ~@Pump-7 OPEN IF NODE T-12 BELOW ';
    combinedString5 = sprintf('%s%d', Row1, Value1);

    Value2 = (app.OffwhenAboveEditField_3.Value) +104.5;
    Row2 = 'Link ~@Pump-7 CLOSED IF NODE T-12 ABOVE ';
    combinedString6 = sprintf('%s%d', Row2, Value2);

end

```

%now that we have the above groupings of pump protocols, we
%can go to the time specified settings

%This is for the springfield road pump

```
SpringfieldPumpStringOPEN = 'LINK ~@Pump-7 OPEN AT TIME ';  
SpringfieldPumpStringCLOSED = 'LINK ~@Pump-7 CLOSED AT TIME ';  
WTPPumpStringOPEN = 'LINK ~@Pump-8 OPEN AT TIME ';  
WTPPumpStringCLOSED = 'LINK ~@Pump-8 CLOSED AT TIME ';
```

%Now read in all of the values

```
spring1 = app.ONATTIMEEditField.Value;  
spring2 = app.ONATTIMEEditField_3.Value;  
spring3 = app.ONATTIMEEditField_4.Value;  
spring4 = app.ONATTIMEEditField_5.Value;  
spring5 = app.OFFATTIMEEditField.Value;  
spring6 = app.OFFATTIMEEditField_2.Value;  
spring7 = app.OFFATTIMEEditField_3.Value;  
spring8 = app.OFFATTIMEEditField_4.Value;  
spring9 = app.ONATTIMEEditField_6.Value;  
spring10 = app.OFFATTIMEEditField_5.Value;  
spring11 = app.ONATTIMEEditField_7.Value;  
spring12 = app.OFFATTIMEEditField_6.Value;
```

```
combinedString7 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring1);  
combinedString8 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring2);  
combinedString9 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring3);  
combinedString10 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring4);
```

```
combinedString11 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring5);  
combinedString12 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring6);  
combinedString13 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring7);  
combinedString14 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring8);
```

```
combinedString15 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring9);  
combinedString16 = sprintf('%s%s', SpringfieldPumpStringOPEN, spring11);
```

```
combinedString17 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring10);  
combinedString18 = sprintf('%s%s', SpringfieldPumpStringCLOSED, spring12);
```

```
AllSimpleControls = {combinedString1;  
combinedString2;combinedString3;combinedString4;combinedString5;combinedString6;combinedString7;  
combinedString11;combinedString10;combinedString12;combinedString9;combinedString13;  
combinedString8;combinedString14;combinedString15;combinedString17;combinedString16;combinedString18};
```

```
%% The below describes how I placed a junction in EPANET instead of a pump so I can
control the outflow exactly. This is actually done when trying to generate system pump
curves as well.
```

```
%This is also the unique new way we will be placing the WTP pump data into
%the simulation. We are doing this because we want to force the inflow to
%be a certain value (because when we are looking at historical data we know
%the inflow into the system.
```

```
WTPPumpData = [app.CurrentFlowgpmEditField.Value,
app.Hour1gpmEditField.Value, app.Hour2gpmEditField.Value, app.Hour3gpmEditField.Value, app.H
our4gpmEditField.Value, app.Hour5gpmEditField.Value, app.Hour6gpmEditField.Value, app.Hour7g
pmEditField.Value, app.Hour8gpmEditField.Value, app.Hour9gpmEditField.Value, app.Hour10gpmEd
itField.Value, app.Hour11gpmEditField.Value, app.Hour12gpmEditField.Value, app.Hour13gpmEdit
Field.Value, app.Hour14gpmEditField_2.Value, app.Hour15gpmEditField_2.Value, app.Hour16gpmEd
itField_2.Value, app.Hour17gpmEditField_2.Value, app.Hour18gpmEditField_2.Value, app.Hour19g
pmEditField_2.Value, app.Hour20gpmEditField_2.Value, app.Hour21gpmEditField_2.Value, app.Hou
r22gpmEditField_2.Value, app.Hour23gpmEditField_2.Value];
```

```
%Because there are only 24 entries (and we are working at a time scale of
%around 10 minute reporting period) we need to make the inputs for an hour
%over 5 entries
```

```
PumpPattern = ones(1,144);
```

```
try
```

```
RealTotalDemand = evalin('base', 'TotalDemand');
RealTotalDemand = RealTotalDemand(1,2:25);
RealTankLevels = evalin('base', 'TankLevels');
RealTankFlow = evalin('base', 'TankFlow');
FlowData = evalin('base', 'FlowData');
```

```
%read in the flow meter data
```

```
BeforeCavMeter = evalin('base', 'BeforeCavMeter');
CavMeter = evalin('base', 'CavMeter');
woodlawnMeter = evalin('base', 'WoodlawnMeter');
DanvilleMeter = evalin('base', 'DanvilleMeter');
SpringfieldRoadMeter = evalin('base', 'SpringfieldRoadMeter');
SaintRoseMeter = evalin('base', 'SaintRoseMeter');
SaintMaryMeter = evalin('base', 'SaintMaryMeter');
CampMeter = evalin('base', 'CampMeter');
ByPassMeter = evalin('base', 'ByPassMeter');
```

```
TotalMeterDemand = evalin('base', 'TotalMeterDemand');
```

```
%The goal below was to clear the data from the workspace after it has been
%used because I wanted to give the user the option to specify pump
```



```
%operations (the try function above will use data from the excel and not
%from user input if it sees any data still in the workspace). However, this
%hasn't worked the way I intended anyways and I will probably just remove
%because this function will likely not be available to the user and it
%solely going to be used for the purpose of generating demand factors that
%will be stored for the sake of user usage.
```

```
PumpPattern = FlowData(2:25,1)';
```

```
clear TotalDemand
```

```
clear TankLevels
```

```
clear TankFlow
```

```
catch
```

```
%If there is an error that means we have not populated the workspace
```

```
%with a pump pattern. Use the user inputted pump pattern instead
```

```
PumpPattern(1,1:6) = PumpPattern(1,1:6) .* WTPPumpData(1,1);
PumpPattern(1,7:12) = PumpPattern(1,7:12) .* WTPPumpData(1,2);
PumpPattern(1,13:18) = PumpPattern(1,13:18) .* WTPPumpData(1,3);
PumpPattern(1,19:24) = PumpPattern(1,19:24) .* WTPPumpData(1,4);
PumpPattern(1,25:30) = PumpPattern(1,25:30) .* WTPPumpData(1,5);
PumpPattern(1,31:36) = PumpPattern(1,31:36) .* WTPPumpData(1,6);
PumpPattern(1,37:42) = PumpPattern(1,37:42) .* WTPPumpData(1,7);
PumpPattern(1,43:48) = PumpPattern(1,43:48) .* WTPPumpData(1,8);
PumpPattern(1,49:54) = PumpPattern(1,49:54) .* WTPPumpData(1,9);
PumpPattern(1,55:60) = PumpPattern(1,55:60) .* WTPPumpData(1,10);
PumpPattern(1,61:66) = PumpPattern(1,61:66) .* WTPPumpData(1,11);
PumpPattern(1,67:72) = PumpPattern(1,67:72) .* WTPPumpData(1,12);
PumpPattern(1,73:78) = PumpPattern(1,73:78) .* WTPPumpData(1,13);
PumpPattern(1,79:84) = PumpPattern(1,79:84) .* WTPPumpData(1,14);
PumpPattern(1,85:90) = PumpPattern(1,85:90) .* WTPPumpData(1,15);
PumpPattern(1,91:96) = PumpPattern(1,91:96) .* WTPPumpData(1,16);
PumpPattern(1,97:102) = PumpPattern(1,97:102) .* WTPPumpData(1,17);
PumpPattern(1,103:108) = PumpPattern(1,103:108) .* WTPPumpData(1,18);
PumpPattern(1,109:114) = PumpPattern(1,109:114) .* WTPPumpData(1,19);
PumpPattern(1,115:120) = PumpPattern(1,115:120) .* WTPPumpData(1,20);
PumpPattern(1,121:126) = PumpPattern(1,121:126) .* WTPPumpData(1,21);
PumpPattern(1,127:132) = PumpPattern(1,127:132) .* WTPPumpData(1,22);
PumpPattern(1,133:138) = PumpPattern(1,133:138) .* WTPPumpData(1,23);
PumpPattern(1,139:144) = PumpPattern(1,139:144) .* WTPPumpData(1,24);
```

```
end
```

```
Logic1 = strcmp(SpringfieldPumpStringOPEN, AllSimpleControls);
```

```
whereLogic1 = find(Logic1 ==1);
```

```
AllSimpleControls(whereLogic1, :) = [];
```

```
Logic2 = strcmp(SpringfieldPumpStringCLOSED, AllSimpleControls);
```

```
whereLogic2 = find(Logic2 ==1);
```

```
AllSimpleControls(whereLogic2, :) = [];
```

```

Logic5 = strcmp('NULL', AllSimpleControls);
whereLogic5 = find(Logic5 ==1);
AllSimpleControls(whereLogic5, :) = [];

%Get Outputs from the simulation

%test some initial demand points these are all constrained by the demand
%equation built out by getting the total base demands for the different
%pressure zones. This will change if the zones or individual base demands
%themsevles are altered. ExtendedPeriodDemandCalibrator

%These are named in reference to zone 1. Demand pattern high is a high
%demand pattern for zone 1 and a low one for zone 2. Demand pattern low is
%a low demand pattern for zone 1 and a high one for zone 2. 254.6 is the
%base demand in zone 1 and 116.3467 is the base demand in zone 2

demandPatternLow = ones(1, 24).*0.001;
demandPatternLowComp = ((RealTotalDemand)- (demandPatternLow * 254.9906)) ./ 116.3467;

demandPatternHighComp = ones(1, 24).*0.001;
demandPatternHigh = ((RealTotalDemand) - (demandPatternHighComp * 116.3467)) ./ 254.9906;

demandPatternLow = vertcat(demandPatternLow, demandPatternLowComp);
demandPatternHigh = vertcat(demandPatternHigh, demandPatternHighComp);

demandPatternsNew = (demandPatternHigh(1,:) + demandPatternLow(1,:)) ./ 2;
demandPatternsNewComp = ((RealTotalDemand)- (demandPatternsNew * 254.9906)) ./ 116.3467;
demandPatternsNew = vertcat(demandPatternsNew, demandPatternsNewComp);

%what File are we using

whatFile = app.RunwithNewTankCheckBox.Value;

if whatFile == 0
    d = epanet('LebanonCurrent_July2023.inp', 'LoadFile');
elseif whatFile ==1
    d = epanet('LebanonNew_July2023.inp', 'LoadFile');
end
assignin('base', 'd', d);

%initialize some of the variables for the bi-section method

[TankLevelsNew] = ExtendedPeriodDemandCalibrator(InitialTankLevels,
AllSimpleControls,3600,86400, PumpPattern, demandPatternsNew, BeforeCavMeter(1,2:25),
CavMeter(1,2:25), woodlawnMeter(1,2:25), DanvilleMeter(1,2:25),
SpringfieldRoadMeter(1,2:25), SaintRoseMeter(1,2:25), SaintMaryMeter(1,2:25),
CampMeter(1,2:25), ByPassMeter(1,2:25));

```

```

I =1;
CavalryCount = 0;
SpringCount = 1;
Tank1Error = 1;

%24 hours in a day
while I < 24

%Now here with the bi-section method. I've had to re-do this a bunch of
%times. Yes this is simple but word to the wise, make sure you look
%critically at your data before you try to work with garbage (GIGO
%principle abused here before)

    %lets check the error
    %Cav Tank
    Tank1Error = RealTankLevels(I+1, 2) - TankLevelsNew(I+1,2);
    %Cav Tank
    Tank2Error = RealTankLevels(I+1, 1) - TankLevelsNew(I+1,1);

    %if the tank level is close to relity, advance to the next step
    if abs(Tank1Error) <.005
        I =I+1

        %make some adjustments to the objective function to account for
        %the difference between the model tank levels and the real ones
        RealTotalDemand(1,I) = (2*((TankLevelsNew((I),2) - RealTankLevels((I+1),2)) *
(3.1415/4) * (48^2) * (7.48 / 60))) +
(sprinfeldRoadTankFunction(TankLevelsNew(I,1),RealTankLevels((I+1),1))) +
FlowData(I+1,1) - TotalMeterDemand(1,I+1) ;
        demandPatternLow(2,I) = ((RealTotalDemand(1,I))- (demandPatternLow(1,I) *
254.9906)) ./ 116.3467;
        demandPatternHigh(1,I) = ((RealTotalDemand(1,I)) -
(demandPatternHighComp(1,I) * 116.3467)) ./ 254.9906;
        demandPatternsNew(1,I) = (demandPatternHigh(1,I) + demandPatternLow(1,I)) ./
2;

    elseif abs(Tank1Error) >=.005
        if abs(demandPatternHigh(1,I) - demandPatternsNew(1, I)) <= .0001
            I=I+1
            %make some adjustments to the objective function to account for
            %the difference between the model tank levels and the real ones
            RealTotalDemand(1,I) = (2*((TankLevelsNew((I),2) - RealTankLevels((I+1),2)) *
(3.1415/4) * (48^2) * (7.48 / 60))) +
(sprinfeldRoadTankFunction(TankLevelsNew(I,1),RealTankLevels((I+1),1))) +
FlowData(I+1,1) - TotalMeterDemand(1,I+1) ;
            demandPatternLow(2,I) = ((RealTotalDemand(1,I))- (demandPatternLow(1,I) *
254.9906)) ./ 116.3467;
            demandPatternHigh(1,I) = ((RealTotalDemand(1,I)) -
(demandPatternHighComp(1,I) * 116.3467)) ./ 254.9906;
            demandPatternsNew(1,I) = (demandPatternHigh(1,I) + demandPatternLow(1,I)) ./
2;

```

```

elseif abs(demandPatternLow(1,I) - demandPatternsNew(1,I)) <= .0001
    I = I+1
    %make some adjustments to the objective function to account for
    %the difference between the model tank levels and the real ones
    RealTotalDemand(1,I) = (2*((TankLevelsNew((I),2) - RealTankLevels((I+1),2)) *
(3.1415/4) * (48^2) * (7.48 / 60))) +
(springfieldRoadTankFunction(TankLevelsNew(I,1),RealTankLevels((I+1),1))) +
FlowData(I+1,1) - TotalMeterDemand(1,I+1) ;
    demandPatternLow(2,I) = ((RealTotalDemand(1,I))- (demandPatternLow(1,I) *
254.9906)) ./ 116.3467;
    demandPatternHigh(1,I) = ((RealTotalDemand(1,I)) -
(demandPatternHighComp(1,I) * 116.3467)) ./ 254.9906;
    demandPatternsNew(1,I) = (demandPatternHigh(1,I) + demandPatternLow(1,I)) ./
2;

%or if we are still improving, enter the script below
elseif demandPatternHigh(1,I) ~= demandPatternsNew(1, I) || demandPatternLow(1,I)
~= demandPatternsNew(1,I)
    %If the error is greater than 1 for the cav tank that means the demand is
    %too low and needs to come up
    if Tank1Error >0
        demandPatternHigh(1,I) = demandPatternsNew(1,I);
        demandPatternsNew(1,I) = (demandPatternHigh(1,I) +
demandPatternLow(1,I)) ./ 2;
        demandPatternsNewComp = ((RealTotalDemand) - (demandPatternsNew(1,:)
* 254.9906)) ./ 116.3467;
        demandPatternsNew(2,I) = demandPatternsNewComp(1, I);
    end
    %If the error is less than 1 for the Cav tank, that means that the demand
    %is too high in the model and needs to come down
    if Tank1Error <0
        demandPatternLow(1,I) = demandPatternsNew(1,I);
        demandPatternsNew(1,I) = (demandPatternHigh(1,I) +
demandPatternLow(1,I)) ./ 2;
        demandPatternsNewComp = ((RealTotalDemand) - (demandPatternsNew(1,:)
* 254.9906)) ./ 116.3467;
        demandPatternsNew(2,I) = demandPatternsNewComp(1, I);
    end

    end

[TankLevelsNew] = ExtendedPeriodDemandCalibrator(InitialTankLevels,
AllSimpleControls,3600,86400, PumpPattern, demandPatternsNew, BeforeCavMeter(1,2:25),
CavMeter(1,2:25), woodlawnMeter(1,2:25), DanvilleMeter(1,2:25),
SpringfieldRoadMeter(1,2:25), SaintRoseMeter(1,2:25), SaintMaryMeter(1,2:25),
CampMeter(1,2:25), Meter(1,2:25));
    end
end
if I > 23
    pinchme = 1;
end
end
end

```

```

end

% Callback function
function FillUsingExcelDataButtonPushed(app, event)
    T = readtable('RealTankDataTest.xlsx');
    TotalDemand = T(:,20);
    FlowData = T(:,8);
    TankLevels = T(:,21:22);
    TankFlow = T(:,5:6);

    %Flow Information at the Meters (periphery of the system)
    Before_Cavalry = T(:,9);
    Calvary_Meter = T(:,10);
    WoodLawn_Meter = T(:,11);
    DanvilleHighway_Meter = T(:,12);
    SpringfieldRoad_Meter = T(:,13);
    SaintRose_Meter = T(:,14);
    SaintMary_Meter = T(:,15);
    Campbellsville_Meter = T(:,16);
    ByPass_Meter = T(:,17);
    TotalMeterDemand = T(:, 19);

    assignin('base','FlowData', FlowData);
    assignin('base','TotalDemand', TotalDemand);
    assignin('base','TankLevels', TankLevels);
    assignin('base','TankFlow', TankFlow);

    %Assign the meter data into the base workspace in MATLAB for
    %use in the demand creation function

    assignin('base','BeforeCavMeter', Before_Cavalry);
    assignin('base','CavMeter', Calvary_Meter);
    assignin('base','WoodlawnMeter', woodLawn_Meter);
    assignin('base','DanvilleMeter', DanvilleHighway_Meter);
    assignin('base','SpringfieldRoadMeter', SpringfieldRoad_Meter);
    assignin('base','SaintRoseMeter', SaintRose_Meter);
    assignin('base','SaintMaryMeter', SaintMary_Meter);
    assignin('base','CampMeter', Campbellsville_Meter);
    assignin('base','ByPassMeter', ByPass_Meter);

    assignin('base', 'TotalMeterDemand', TotalMeterDemand);

end

% Value changed function: Zone1PatternDropDown
function Zone1PatternDropDownValueChanged(app, event)
    value = app.Zone1PatternDropDown.Value;

    % Read in the demand patterns from excel file

```

```

tDemand = readtable('DemandFactors.xlsx');

%Turn that data (which is in cell format) into a num variable

demandsZone1 = horzcat(tDemand{:,1:14}, tDemand{:,29:30});

%index dates and find which values to extract
dateNames = ["June 20", "June 21","June 22","June 23","June 24","June
25","June 26","June 27","June 28","June 29","June 30","July 1","July 2","July 3","Average
weekday","Average weekend"];
[~, whereDate] = ismember(value, dateNames);

%Now the user can specify which demand pattern he/ she wants to
%use and it will populate within the simulation
DZ1 = demandsZone1(:,whereDate);

%Operators allowed to scale
if app.ScaleZone1PatternByEditField.Value ~=0
    DZ1 = DZ1 *app.ScaleZone1PatternByEditField.Value
end

%Take the demand factor and assign in workspace so that it can
%be used in the EPS simulation
assignin('base', 'DZ1', DZ1);

Time = [1:24];
%Take this and populate the graph next to the user input
plot(app.UIAxes_10, Time,DZ1)

end

% Value changed function: Zone2PatternDropDown
function Zone2PatternDropDownValueChanged(app, event)
    value = app.Zone2PatternDropDown.Value;
    % Read in the demand patterns from excel file
    tDemand = readtable('DemandFactors.xlsx');

    %Turn that data (which is in cell format) into a num variable

    demandsZone2 = horzcat(tDemand{:,15:28}, tDemand{:,31:32});

    %index dates and find which values to extract
    dateNames = ["June 20", "June 21","June 22","June 23","June 24","June
25","June 26","June 27","June 28","June 29","June 30","July 1","July 2","July 3","Average
weekday","Average weekend"];
    [~, whereDate] = ismember(value, dateNames);

    %Now the user can specify which demand pattern he/ she wants to
    %use and it will populate within the simulation
    DZ2 = demandsZone2(:,whereDate);

    %Operators allowed to scale
    if app.ScaleZone2PatternByEditField.Value ~=0
        DZ2 = DZ2 *app.ScaleZone2PatternByEditField.Value
    end

```

```

end

%Take the demand factor and assign in workspace so that it can
%be used in the EPS simulation
assignin('base', 'DZ2', DZ2);

Time = [1:24];
%Take this and populate the graph next to the user input
plot(app.UIAxes_11, Time,DZ2)
end

% Value changed function: ScaleZone1PatternByEditField
function ScaleZone1PatternByEditFieldValueChanged(app, event)
    value = app.ScaleZone1PatternByEditField.Value;

    % Read in the demand patterns from excel file
    tDemand = readtable('DemandFactors.xlsx');

    %Turn that data (which is in cell format) into a num variable

    demandsZone1 = horzcat(tDemand{:,1:14}, tDemand{:,29:30});

    %index dates and find which values to extract
    dateNames = ["June 20", "June 21","June 22","June 23","June 24","June
25","June 26","June 27","June 28","June 29","June 30","July 1","July 2","July 3","Average
weekday","Average weekend"];
    [~, whereDate] = ismember(app.Zone1PatternDropDown.Value, dateNames);

    %Now the user can specify which demand pattern he/ she wants to
    %use and it will populate within the simulation
    DZ1 = demandsZone1(:,whereDate);

    %Operators allowed to scale
    if value ~=0
        DZ1 = DZ1 *app.ScaleZone1PatternByEditField.Value
    end

    %Take the demand factor and assign in workspace so that it can
    %be used in the EPS simulation
    assignin('base', 'DZ1', DZ1);

    Time = [1:24];
    %Take this and populate the graph next to the user input
    plot(app.UIAxes_10, Time,DZ1)

end

% Value changed function: ScaleZone2PatternByEditField
function ScaleZone2PatternByEditFieldValueChanged(app, event)
    value = app.ScaleZone2PatternByEditField.Value;

    % Read in the demand patterns from excel file
    tDemand = readtable('DemandFactors.xlsx');

```

```

%Turn that data (which is in cell format) into a num variable

demandsZone2 = horzcat(tDemand{:,15:28}, tDemand{:,31:32});

%index dates and find which values to extract
dateNames = ["June 20", "June 21","June 22","June 23","June 24","June
25","June 26","June 27","June 28","June 29","June 30","July 1","July 2","July 3","Average
weekday","Average weekend"];
[~, whereDate] = ismember(app.Zone2PatternDropDown.Value, dateNames);

%Now the user can specify which demand pattern he/ she wants to
%use and it will populate within the simulation
DZ2 = demandsZone2(:,whereDate);

%Operators allowed to scale
if value ~=0
    DZ2 = DZ2 *app.ScaleZone2PatternByEditField.Value;
end

%Take the demand factor and assign in workspace so that it can
%be used in the EPS simulation
assignin('base', 'DZ2', DZ2);

Time = [1:24];
%Take this and populate the graph next to the user input
plot(app.UIAxes_11, Time,DZ2)

end
end

```


APPENDIX C. User's Manual for Digital Twin (Lebanon)

- 1) Install MATLAB software (version R2022A used but other version within a few years give or take will likely work).

(<https://www.mathworks.com/help/install/ug/install-products-with-internet-connection.html>) Downloading and license registration is free at many Universities

and a purchased license is likely not necessary.

- 2) Install EPANET (Version 2.2)

<https://www.epa.gov/water-research/epanet>

- 3) Install the EPANET-MATLAB Toolkit (Eliades et al, 2016):

<https://www.mathworks.com/matlabcentral/fileexchange/25100-openwateranalytics-epanet-matlab-toolkit>

- 4) Ensure that the toolkit is pointed in the correct directory.

Place the downloaded toolkit within your MATLAB folder and unzip it in that location. Once this is accomplished, edit the “Start_Toolkit.m” file to specify where exactly the toolkit is located.

The screenshot shows a Windows File Explorer window titled 'Documents'. The address bar indicates the current location is 'This PC > Documents'. The ribbon at the top includes 'File', 'Home', 'Share', and 'View' tabs, with various icons for file management operations like Copy, Paste, Delete, and Rename. The main pane displays a list of files and folders with columns for Name, Status, Date modified, Type, and Size. The 'MATLAB' folder is highlighted in yellow.

Name	Status	Date modified	Type	Size
ArcGIS		12/14/2022 4:38 PM	File folder	
ArcGIS 10.8.2		9/14/2022 10:44 AM	File folder	
ArcGIS Pro 2.9		9/14/2022 10:43 AM	File folder	
ArcGIS Pro 3.0		9/14/2022 10:44 AM	File folder	
CE 667		11/2/2023 11:14 AM	File folder	
CE 672 - 001 Landfill design		9/14/2022 1:23 PM	File folder	
Custom Office Templates		9/14/2022 10:43 AM	File folder	
DevelopingSystemCurves		8/15/2023 3:43 PM	File folder	
epanet_matlab_toolkit		9/14/2022 10:44 AM	File folder	
Enka's Functions		9/22/2022 11:33 AM	File folder	
Fax		8/14/2023 2:21 PM	File folder	
features		9/14/2022 10:43 AM	File folder	
GILL_Fall2022		9/13/2023 11:19 AM	File folder	
GILL_Spring2023		11/20/2023 1:41 PM	File folder	
GILL_Summer2022		4/25/2023 8:17 AM	File folder	
GILL_Summer2023		5/17/2023 2:47 PM	File folder	
GILL_Summer2023		10/13/2023 9:33 PM	File folder	
GitHub		11/27/2023 7:07 PM	File folder	
KYValvedSystems		9/22/2022 11:32 AM	File folder	
MATLAB		10/21/2023 11:42 AM	File folder	
MatlabJunk		11/7/2023 2:15 PM	File folder	
Misc		12/14/2022 3:15 PM	File folder	
Network Monitor 3		9/14/2022 10:43 AM	File folder	
New folder		12/14/2022 3:11 PM	File folder	
Python Scripts		8/25/2023 11:43 AM	File folder	
RAS		11/2/2023 11:14 AM	File folder	
Scanned Documents		11/10/2023 11:17 AM	File folder	
USB Drive Stuff		9/14/2022 10:44 AM	File folder	
Water Quality Testing		6/5/2023 1:30 PM	File folder	
WhitesburgProjectFiles		9/13/2023 11:18 AM	File folder	
Zoom		10/18/2023 10:46 PM	File folder	
DataCleaning2		11/17/2022 3:34 PM	Microsoft Excel W...	13 KB
Desktop - Shortcut		12/8/2022 10:24 AM	Shortcut	1 KB
Envision Checklist for Report III-GILL		4/20/2022 4:59 PM	Microsoft Word D...	12 KB
envision_checklist-GILL		4/20/2022 4:59 PM	Microsoft Excel W...	580 KB
ErrorFromEPANETGeneration		10/12/2022 10:05 AM	Adobe Acrobat D...	194 KB
EWRIConferenceProgram		8/3/2023 9:55 AM	JPG File	99 KB
excllink		12/14/2016 1:32 PM	Microsoft Excel A...	109 KB

MATLAB

File Home Share View

Pin to Quick access Copy Paste Cut Copy path Move to Copy to Delete Rename New folder New Item Easy access Properties Edit History Select all Select none Invert selection Select

Clipboard Organize New Open Select

← → ↑ ↓ > This PC > Documents > MATLAB

	Name	Status	Date modified	Type	Size
★ Quick access					
📁 Documents	epanet_matlab_toolkit		6/23/2023 3:44 PM	File folder	
📁 Downloads	EPANET-Matlab-Toolkit-2.2.3		10/18/2022 6:01 AM	File folder	
📁 Pictures	Examples		6/19/2023 3:04 PM	File folder	
📁 assignment-14-nets	html		11/8/2023 11:06 AM	File folder	
📁 MATLAB	Apr24_Gill_Box_temp		4/26/2023 3:43 PM	BIN File	30 KB
📁 RoughDraftofThesis	Apr24_Gill_Box		4/25/2023 9:26 AM	INP File	205 KB
📁 THESIS	DecemberEditsWhitesburg		11/8/2023 9:15 AM	INP File	195 KB
	DecemberEditsWhitesburg_temp		11/10/2023 4:54 PM	INP File	255 KB
	LebanonCurrent_July2023		9/16/2023 11:54 AM	INP File	375 KB
📁 Creative Cloud Files	LebanonCurrent_July2023Testable		10/18/2023 5:00 PM	INP File	374 KB
📁 OneDrive - University	LebanonCurrent_July2023Testable_temp		11/15/2023 1:27 PM	INP File	614 KB
📁 Apps	LebanonNew_July2023		7/25/2023 2:43 PM	INP File	375 KB
📁 Attachments	mcwata		11/21/2023 11:40 AM	INP File	707 KB
📁 Desktop	mcwata_temp		11/21/2023 11:42 AM	INP File	707 KB
📁 Documents	Net3		11/7/2023 4:00 PM	INP File	30 KB
📁 Office Lens	arrow		7/26/2023 12:13 PM	JPG File	9 KB
📁 Pictures	LWW Logo		7/19/2023 1:21 PM	JPG File	15 KB
📁 ResearchPapers	UKVLogo2		1/2/2023 3:17 PM	JPG File	678 KB
📁 Week1_Example_Gl:	OperatorDashboard_Lebanon_Final		10/19/2023 1:04 PM	MATLAB App	422 KB
	OperatorDashboardV1		1/9/2023 2:33 PM	MATLAB App	298 KB
	OperatorDashboardV1_Lebanon		10/17/2023 9:21 AM	MATLAB App	993 KB
📁 This PC	BoxComplexTest		11/7/2023 6:14 PM	MATLAB Code	5 KB
📁 3D Objects	ExtendedPeriod		1/7/2023 1:44 PM	MATLAB Code	6 KB
📁 Desktop	ExtendedPeriod_Lebanon		7/13/2023 1:29 PM	MATLAB Code	8 KB
📁 Documents	ExtendedPeriodDemandCalibrator		9/15/2023 6:52 PM	MATLAB Code	4 KB
📁 Downloads	ExtendedPeriodV2		10/18/2023 8:46 PM	MATLAB Code	10 KB
📁 Music	March21		4/3/2023 6:27 PM	MATLAB Code	21 KB
📁 Pictures	oldBisection		9/15/2023 9:56 PM	MATLAB Code	24 KB
📁 Videos	OperatorDashboard_Lebanon_Final_expo...		10/19/2023 11:11 AM	MATLAB Code	214 KB
	OperatorDashboard_Lebanon_FinalV2_ex...		10/19/2023 1:04 PM	MATLAB Code	208 KB
📁 Windows (C:)	OptimSimTest		5/9/2023 1:32 PM	MATLAB Code	8 KB
	PlotObjectiveFun		5/9/2023 1:32 PM	MATLAB Code	8 KB
📁 Network	PumpCurveCal		7/3/2023 1:27 PM	MATLAB Code	2 KB
	sprinfliedRoadTankFunction		9/15/2023 9:05 PM	MATLAB Code	3 KB
	Start_Toolkit		12/9/2022 11:48 AM	MATLAB Code	1 KB
	SteadyMap		6/28/2023 8:13 AM	MATLAB Code	4 KB
	SteadyState		1/6/2023 12:10 PM	MATLAB Code	3 KB
	SteadyState_Lebanon		6/27/2023 9:34 AM	MATLAB Code	3 KB

```

1 function start_toolkit()
2 %START_TOOLKIT Loads all the EPANET-MATLAB Toolkit folder paths in MATLAB.
3 %Run this function before calling 'epanet.m' and the Matlab modules.
4 %
5 % Syntax: start_toolkit
6 %
7 % Inputs:
8 % none
9 %
10 % Outputs:
11 % none
12 %
13 % Example:
14 % start_toolkit
15 %
16 % Other m-files required: none
17 % Subfunctions: none
18 % MAT-files required: none
19 %
20 % See also: none
21
22 % Author : Demetrios G. Eliades, Marios Kyriakou
23 % Work address : KIOS Research Center, University of Cyprus
24 % email : eldemet@ucy.ac.cy
25 % Website : http://www.kios.ucy.ac.cy
26 % Last revision : September 2016
27
28 %----- BEGIN CODE -----
29 addpath(genpath('C:\Users\apgi227\OneDrive - University of Kentucky\Documents\MATLAB\epanet_matlab_toolkit'));
30 disp('EPANET-MATLAB Toolkit Paths Loaded.');
```

5) Take “OperatorDashboard_Lebanon_FinalV2_exported.m” and run the script by hitting the play button in the upper middle ribbon.

PUBLISH VIEW

Refactor Analyze Section Break Run Run and Advance Run to End

CODE ANALYZE SECTION RUN

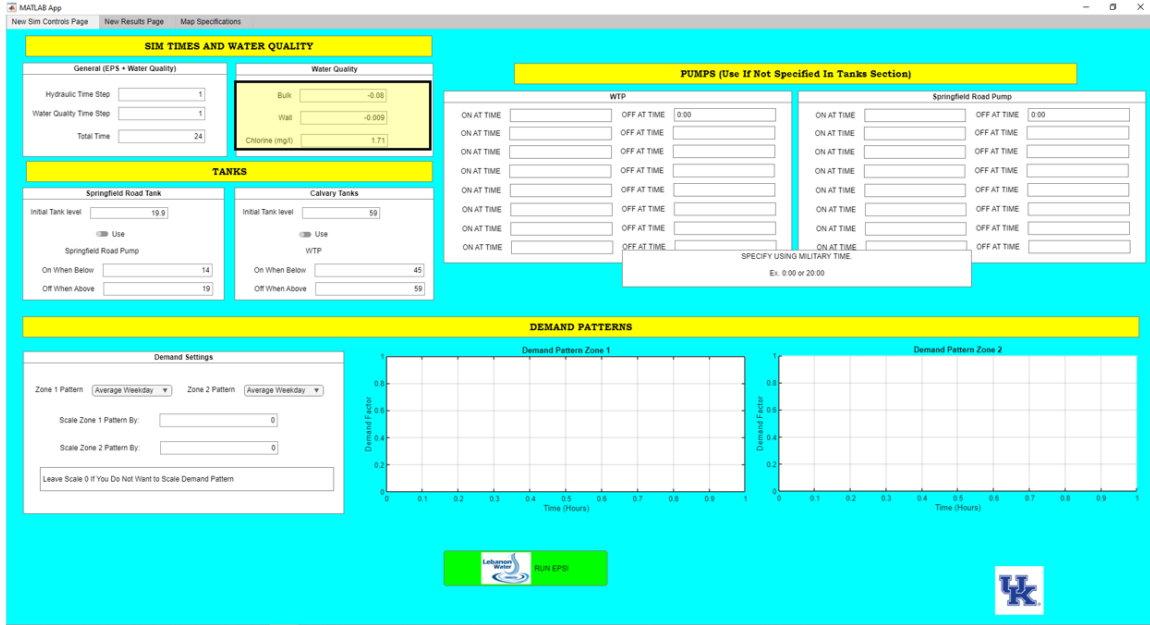
```

1 classdef OperatorDashboard_Lebanon_FinalV2_exported < matlab.apps.AppBase
2
3 % Properties that correspond to app components
4 properties (Access = public)
5     UIFigure
6     TabGroup
7     NewSimControlsPageTab
8     EditField_25
9     EditField_24
10    EditField_23
11    EditField_22
12    TextArea_16
13    WTPPanel
14    OFFATTIMEEditField_16
15    OFFATTIMEEditField_16Label
16    OFFATTIMEEditField_15
17    OFFATTIMEEditField_15Label
18    OFFATTIMEEditField_14
19    OFFATTIMEEditField_14Label
20    OFFATTIMEEditField_13
21    OFFATTIMEEditField_13Label
22    OFFATTIMEEditField_12
23    OFFATTIMEEditField_12Label
24    OFFATTIMEEditField_11
25    OFFATTIMEEditField_11Label
26    OFFATTIMEEditField_10
27    OFFATTIMEEditField_10Label
28    OFFATTIMEEditField_9
29    OFFATTIMEEditField_9Label
30    ONATTIMEEditField_17
31    ONATTIMEEditField_17Label
```

6) The main interface should now be visible. The next steps will go through the process of using inputs. Firstly, the user may the hydraulic and water quality time steps to refine results but this is not necessary. Total time is preset at 24 hours for the extended period sim (EPS) but this may be changed to longer or shorter time steps.

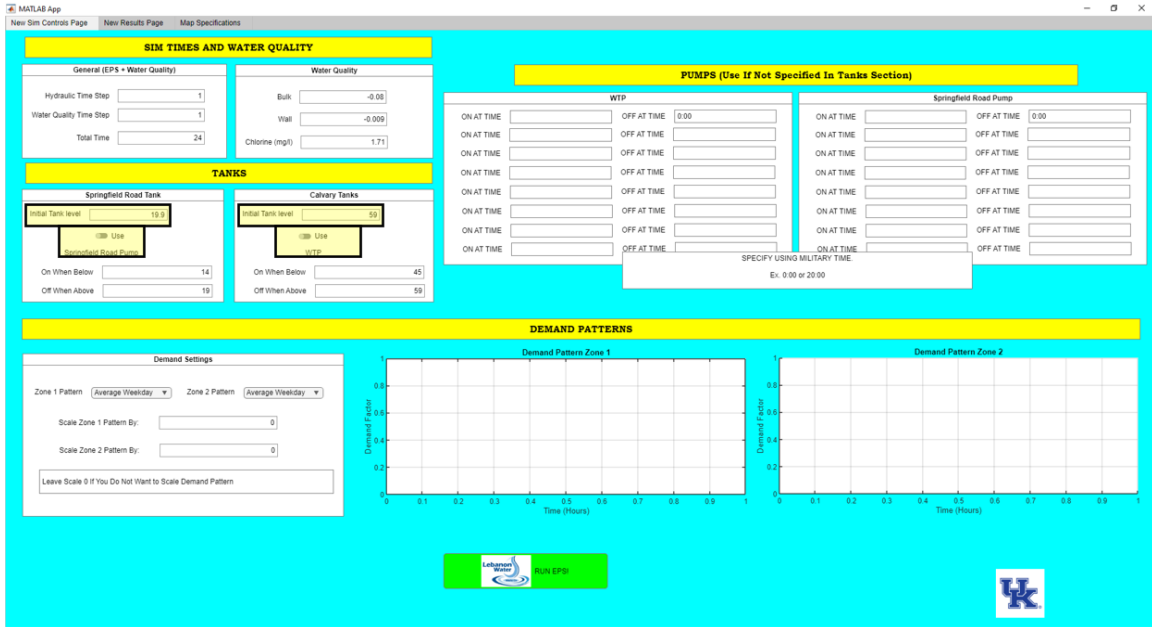


7) Water quality parameters may also be specified but are not necessary. Bulk and wall reaction rates should remain as they are unless an experienced user clearly understands the complex chemistry of their system. Chlorine is the concentration of chlorine at the WTP being pumped out into the system.



8) Initial tank levels are to be specified in the “tanks” section. It is important to know what your MAX tank levels are because the way the software is currently set up, an error may not be thrown if an initial input is over these values.

Also relevant to this section is the “Use” button. The use button allows the user to specify when the pumps will turn on and off dependent on associated tank levels. If “Use” is turned on, the program will ignore inputs in the “PUMPS” section.



9) After specifying the tanks levels in the program, if the user has not used either of the “use” switches for the pumps, the dashboard will use any times specified in the “PUMPS” section as control switches for the pump. These times must be military time (0:00, 20:00, etc.) and should lie within the specified total time for the simulation.



10) Now the user is encouraged to explore the demand section. The demands have been pre-processed for their suitability within the LWW. There is an associated excel file relative to this data that should have been downloaded with the other files sent with this program. Please do not alter this file unless there is a clear understanding of demand factors and how the program is bringing these factors in.

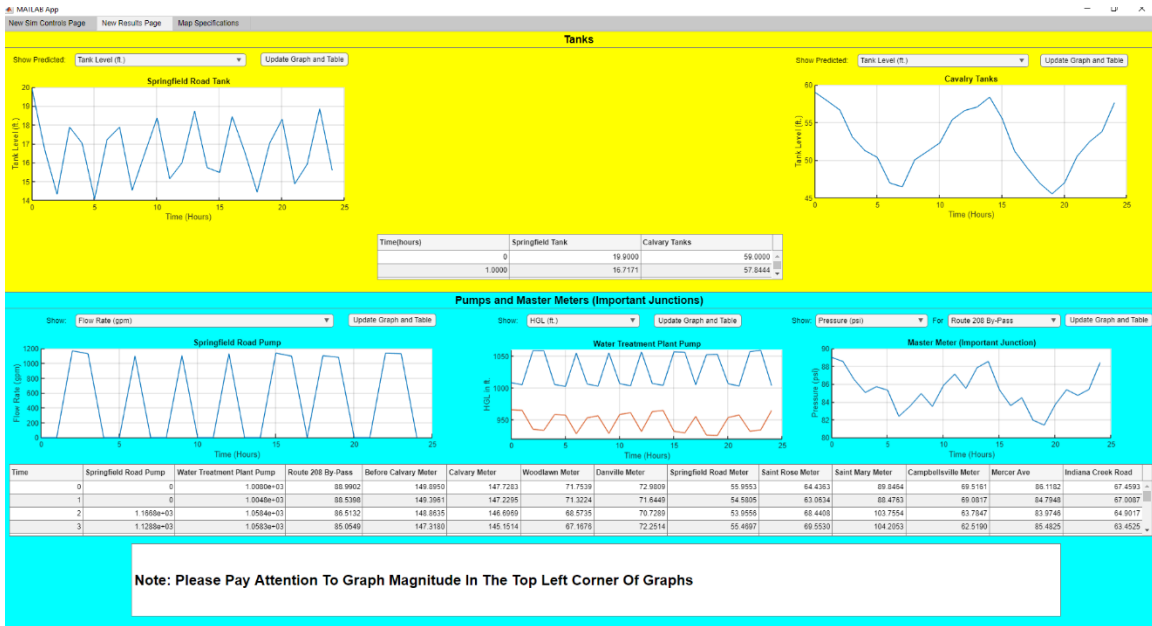
The range of factors is representative of the system between the June 20th and July 3rd 2023. User may use any of these or an average of them as specified by “average weekend” and “average weekday”. If other date ranges are required, please reach out to the University of Kentucky with meter, tank, and pump information as well as their relevant time stamps and they will be processed and sent back.

Factors will conveniently appear in the graphs as soon as they are selected and they may be scaled up and down to increase flexibility within the factors.

Once this has all been specified, hit “Run EPS!” and see results on the next page.



11) In the “Tanks section” on the New Results Page, the user may specify viewing either the water age or the tank levels as a function of time which will be plotted in the top two graphs. In the “Pumps and Master Meters (Important Junctions)” section, the user can specify pump parameters for the first two graphs (HGL and flow rate) and can look at several parameters for the junctions (pressure, demand, chlorine residual, and TTHM concentrations). As soon as a parameter is picked, select “Update Graph and Table” and the graphs and tables will reflect specified parameters.



12) Mapping may also be accomplished in the “Map Specifications Page” within the program. First, the user MUST select “Generate Generic Map”.

Map Specifications

- 1) First press the “Generate Generic Map” button, this will print a map with tank and pump names as well as pipe diameters
- 2) Once figure window is opened, place on screen where most convenient
- 3) Run simulation (found on “Sim Controls” page)
- 4) Specify Nodal and pipe conditions (flows and pressures) as well as display colors and weights, chlorine residual, and DBP predictors

***NOTE: Generic Map Results in the following: 2" = Green, 4" = Cyan, 6" = Red, 8" = Yellow, 10" = Magenta, 12" = Blue, 16" = Black, 20" = White (Pipe Sizes)

Generate Generic Map Generate Nodal Pressure and Pipe Flow Map

Generate Nodal Chlorine Residual Map

Pipe and Junction Discovery

Junction Name: Pipe Name:

Junction Color: Pipe Color:

Junction Weight: Pipe Line Weight:

Turn On All Pipe Names

Turn on All Junction Names

This section allows operators to emphasize specific junctions and pipes on the map. THIS WILL NOT SHOW UP ON MAP UNLESS JUNCTION AND PIPE NAMES MATCH EXACTLY. See Results page after running steady state or EPS sims to identify names.

Nodal (Junction) Pressure

High Pressure (psi): 80 Color 1: EPS?

Medium Pressure (psi): 60 Color 2: EPS?

Low Pressure (psi): 20 Color 3: EPS?

Color 4: EPS?

Above high pressure value will plot as color 1
Above medium pressure value will plot as color 2
Above low pressure values will plot as color 3
Below low pressure value will plot as color 4

Link (Pipe) Flow

High Flow (cfs): 8 Color 5: EPS?

Low Flow (cfs): 2 Color 6: EPS?

Color 7: EPS?

Above high flow value will plot as color 5
Above low flow value will plot as color 6
Below low flow values will plot as color 7

Nodal (Junction) Chlorine

High Concentration (mg/l): 1.5 Color 8: EPS?

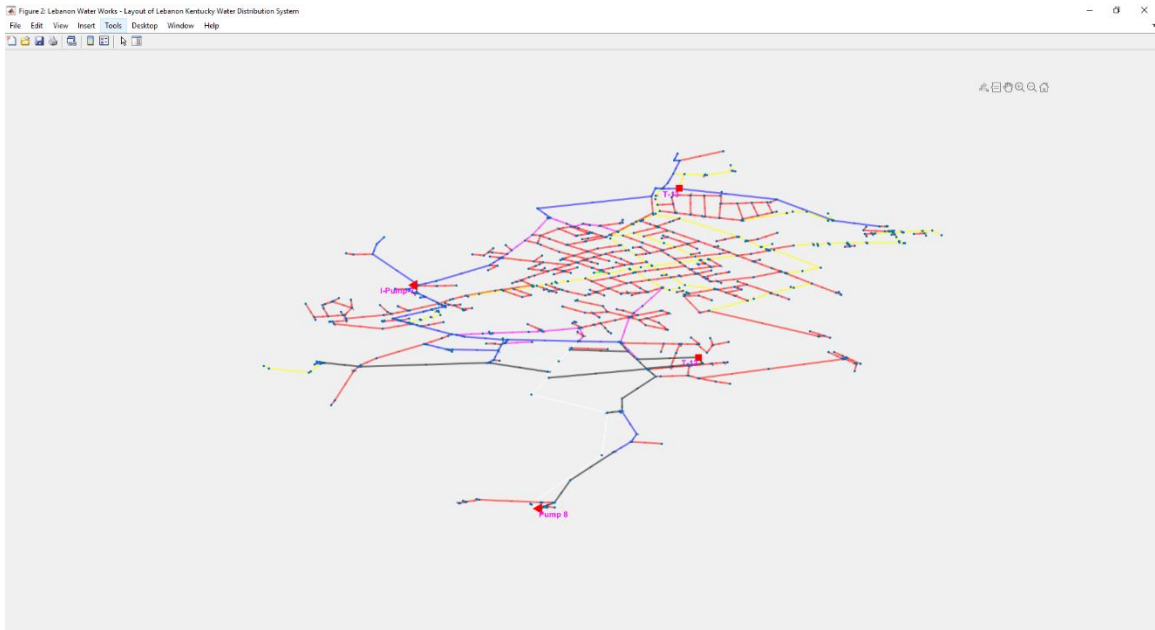
Low Concentration (mg/l): 0.4 Color 9: EPS?

Color 10: EPS?

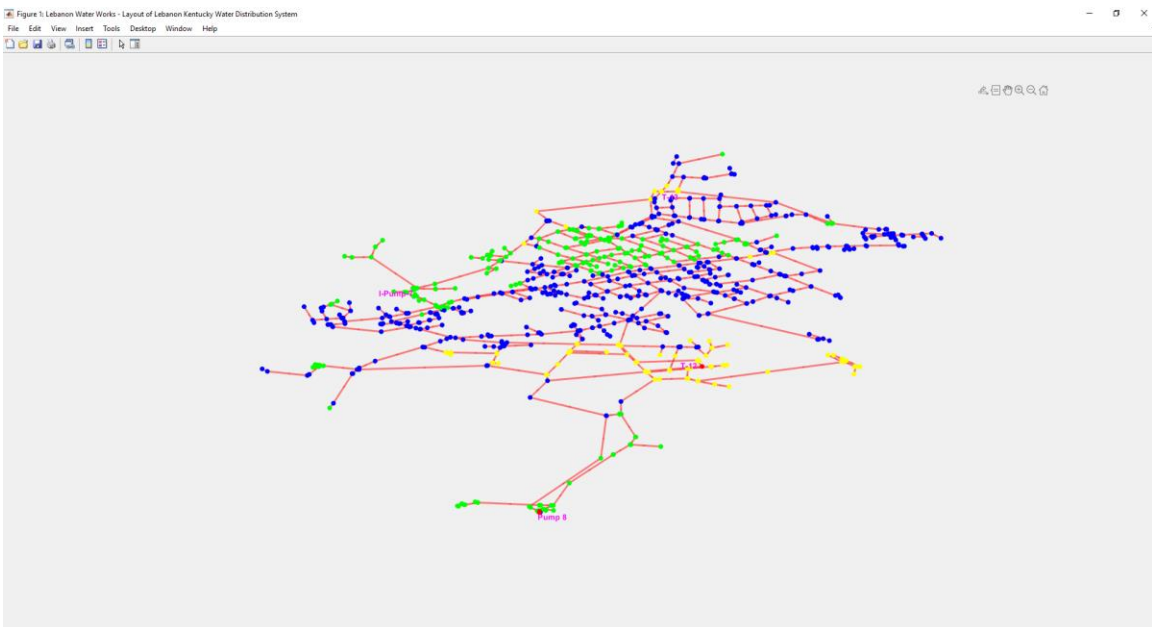
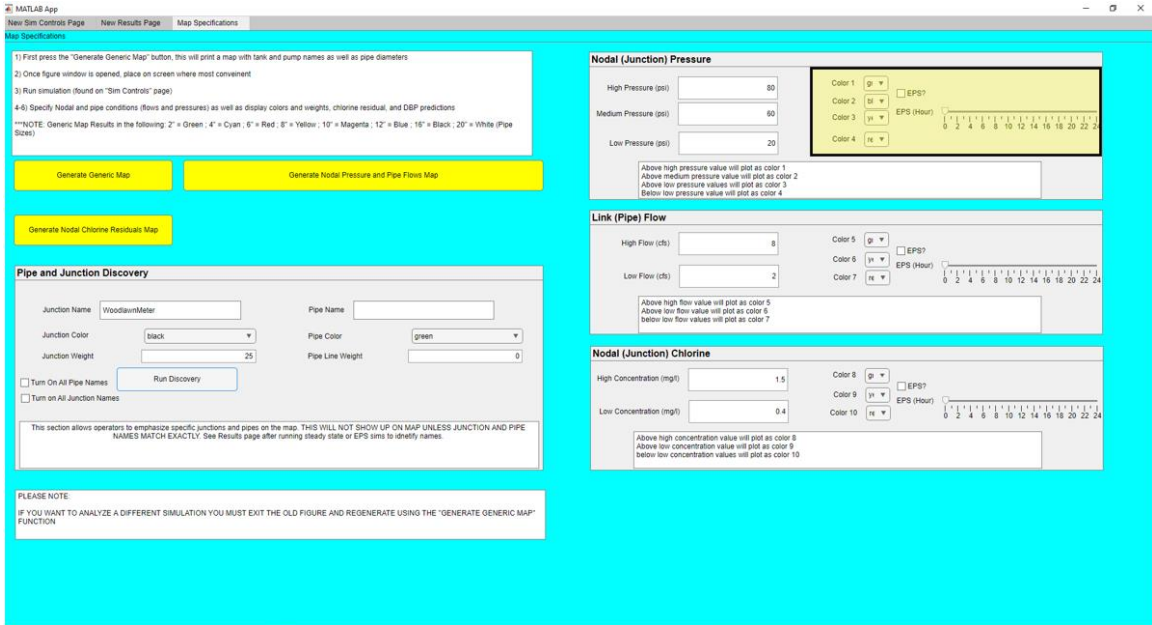
Above high concentration value will plot as color 8
Above low concentration value will plot as color 9
Below low concentration values will plot as color 10

PLEASE NOTE:
IF YOU WANT TO ANALYZE A DIFFERENT SIMULATION YOU MUST EXIT THE OLD FIGURE AND REGENERATE USING THE “GENERATE GENERIC MAP” FUNCTION

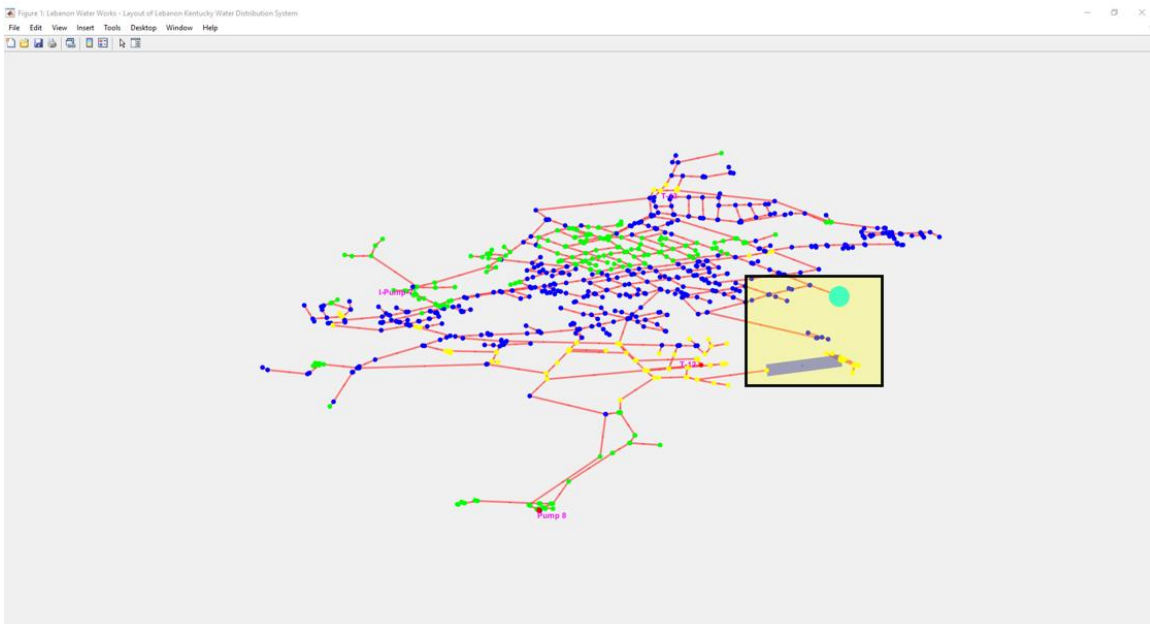
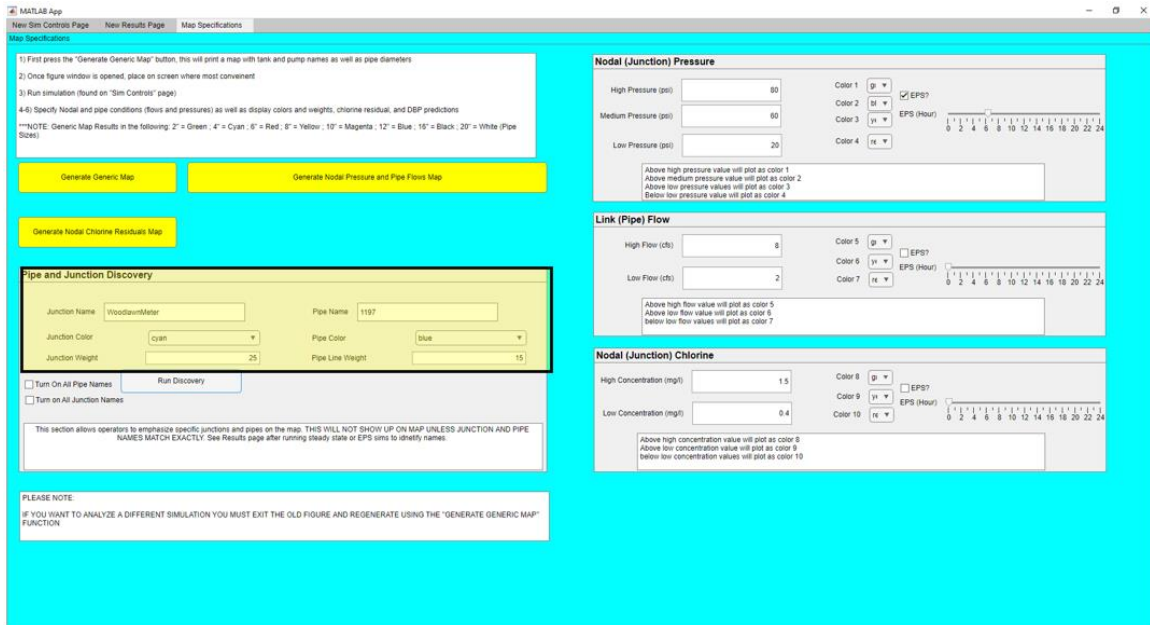
13) A MATLAB figure will be created, and it is suggested that it be placed on another monitor if there are more than one available. The map will initially color all of the pipes based on their nominal diameter.



14) Within each of the categories, the user may specify tolerances on parameters and plot them on the map. For example, I may set the high pressure to 80, medium pressure to 60 and the low pressure to 20. This will then look to each of the colors that I have specified and if the pressure at a node is above 80 it will plot as color 1, in between 80 and 60 will plot as color 2, and so on. User may also check the EPS box and move the slider to specify times to observe. If this is not checked a steady state sim will be shown. This is the same methodology for all of the parameters in this section, after changing the specifications, the user may “Generate Nodal Pressure and Pipe Flow Map”, or “Generate Nodal Chlorine Residuals Map”.



15) In addition, user may also view the names of the pipes and junctions in the “Pipe and Junction Discovery” tab. By typing a name directly as it is viewed in the program and specifying a weight, we can view certain pipes and junctions. This is the most sensitive to user input error and careful attention to the case and spelling of pipes and junctions will give satisfactory results.



16) If the name of a junction or pipe is unknown, the user may check either the “Turn On All Pipe Names” or “Turn On All Junction Names” feature, select “Run Discovery” again, and the names will appear and can be found by interactively zooming in and out of the map. This can be turned off by deselecting the check boxes and hitting the “Run Discovery” button.

MATLAB App

New Sim Controls Page New Results Page Map Specifications

Map Specifications

- 1) First press the "Generate Generic Map" button, this will print a map with tank and pump names as well as pipe diameters
- 2) Once figure window is opened, place on screen where most convenient
- 3) Run simulation (found on "Sim Controls" page)
- 4-6) Specify Nodal and pipe conditions (flows and pressures) as well as display colors and weights, chlorine residual, and DEP predictions

**NOTE: Generic Map Results in the following: 2' = Green, 4' = Cyan, 6' = Red, 8' = Yellow, 10' = Magenta, 12' = Blue, 18' = Black, 20' = White (Pipe Sizes)

Generate Generic Map Generate Nodal Pressure and Pipe Flows Map

Generate Nodal Chlorine Residual Map

Pipe and Junction Discovery

Junction Name: Pipe Name:

Junction Color: Pipe Color:

Junction Weight: Pipe Line Weight:

Turn On All Pipe Names Run Discovery

Turn on All Junction Names

This section allows operators to emphasize specific junctions and pipes on the map. THIS WILL NOT SHOW UP ON MAP UNLESS JUNCTION AND PIPE NAMES MATCH EXACTLY! See Results page after running steady state or EPS Sims to identify names.

Nodal (Junction) Pressure

High Pressure (psi): Color 1: EPS?

Medium Pressure (psi): Color 2: EPS (Hour):

Low Pressure (psi): Color 3: Color 4:

Above high pressure value will plot as color 1
Above medium pressure value will plot as color 2
Above low pressure value will plot as color 3
Below low pressure value will plot as color 4

Link (Pipe) Flow

High Flow (cfs): Color 5: EPS?

Low Flow (cfs): Color 6: EPS (Hour):

Color 7: EPS?

Above high flow value will plot as color 5
Above low flow value will plot as color 6
Below low flow values will plot as color 7

Nodal (Junction) Chlorine

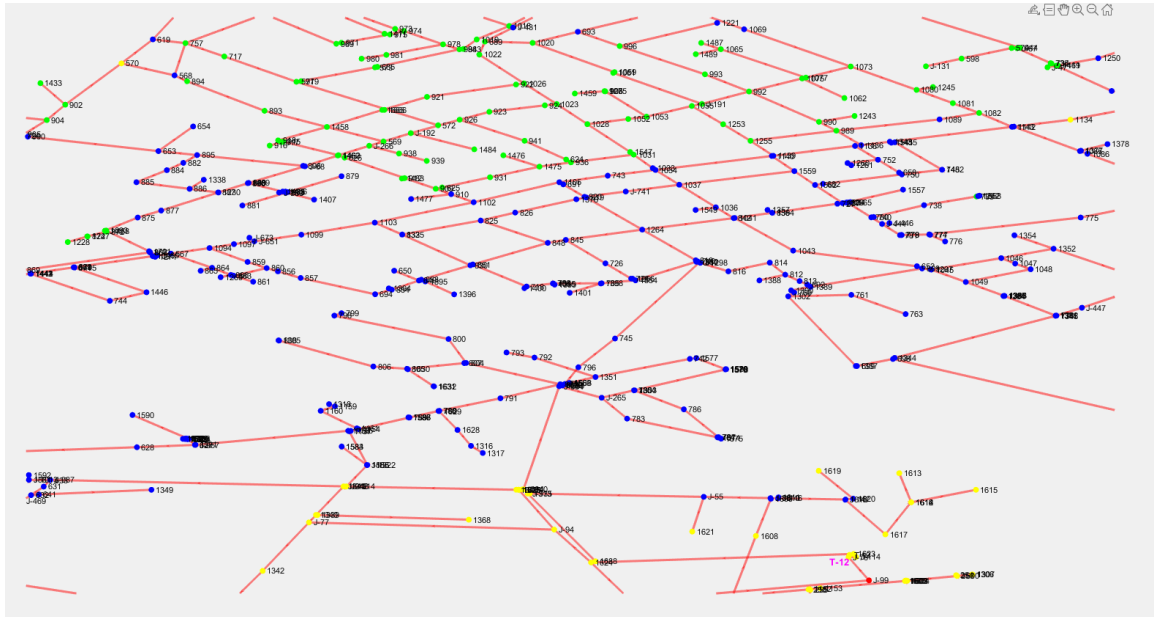
High Concentration (mg/l): Color 8: EPS?

Low Concentration (mg/l): Color 9: EPS (Hour):

Color 10: EPS?

Above high concentration value will plot as color 8
Above low concentration value will plot as color 9
Below low concentration values will plot as color 10

PLEASE NOTE:
IF YOU WANT TO ANALYZE A DIFFERENT SIMULATION YOU MUST EXIT THE OLD FIGURE AND REGENERATE USING THE "GENERATE GENERIC MAP" FUNCTION



APPENDIX D. Example Box-Complex Method Code For Four Zone System

```
%Pull in relevant EPANET file
d = epanet('DecemberEditsWhitesburg.inp', 'LoadFile');
%Completely Random Demand Factors
demandFactors = [2,0.5,3;4,1,2;3,1.5,0.5;5,1.2,1];

%Because I am only testing one hour and epanet requires demand factors
for
%all 24 hours I will create 23 dummy factors
dummyPattern = ones(1,23);

%Now here are the three functions (tanks) that we need to optimize

RealTank1 = 1484.90;
RealTank2 = 1411.49;
RealTank3 = 1469.02;

%Intialize the error matrix (four points in box complex and their
%respective error)
pointsInSimplex = [1,1,1,1];

%This starts the box-complex, here I specify that all of the points need
to
%have a very small error term before it is finished. Essentially,
converge
%on the solution. However this may run for a long time and lossening the
%tolerance will allow for quicker runs

while pointsInSimplex(1,1) >=0.0000001 || pointsInSimplex(1,2) >=0.0000001
|| pointsInSimplex(1,3) <= 0.0000001 || pointsInSimplex(1,4) >=0.0000001

    %evaluation of error for each of the four points in the simplex
    for i = 1:4

        %create the demand patterns to place in the simulation
        pattern1 = demandFactors(i,1);
        pattern2 = demandFactors(i,2);
        pattern3 = demandFactors(i,3);

        %The fourth demand factor is determined by the first three - this is
the
        %total demand function. This is the mass balance in the system which
        %may be found by taking the basedemand for each of the points that
are
```

```

    %specified by a certain demand pattern - multiplying those base
demands
    %by that factor, and summing them altogether. This gives total
demand.
    %To satisfy conservation of mass, the fourth demand factor is
    %determined by the other three. 3,226, and 48 represent the summed
base
    %demands for their respective zones. 391 represents the total demand
of
    %that specific hour.
    pattern4 = (391 - (3 * pattern1) - (226*pattern2) - (48 *
pattern3))/18;

    %Run an initial sim to initialize the box complex

    d.setPattern(1, horzcat(pattern1, dummyPattern));
    d.setPattern(2, horzcat(pattern2, dummyPattern));
    d.setPattern(3, horzcat(pattern3, dummyPattern));
    d.setPattern(4, horzcat(pattern4, dummyPattern));

    d.openHydraulicAnalysis;
    d.initializeHydraulicAnalysis;
    %Run and close analysis
    Series = d.getComputedTimeSeries;
    d.closeHydraulicAnalysis

    ModelTank1 = Series.Head(2,303);
    ModelTank2 = Series.Head(2,308);
    ModelTank3 = Series.Head(2,305);

    Error = ((ModelTank1 - RealTank1)^2) + ((ModelTank2 - RealTank2)^2)
+((ModelTank3 - RealTank3)^2);

    pointsInSimplex(1,i) = Error;
    end
    if pointsInSimplex(1,1) <= 0.000001 || pointsInSimplex(1,2) <= 0.000001 ||
pointsInSimplex(1,3) <= 0.000001 || pointsInSimplex(1,3) <= 0.000001
        break
    else
        %find the worst point and create the centroid of the remaining points
        [maxVal, whereMax] = max(pointsInSimplex);
        logical = find(pointsInSimplex ~= maxVal);
        ph = demandFactors(whereMax,:);
    end
end

```



```

centroid = (demandFactors(logical(1,1),:) +
demandFactors(logical(1,2),:) + demandFactors(logical(1,3),:))./3 ;
%expand the worst point over the centroid of the remaining points

newPoint = (2.5.*centroid) - (1.5 .*ph);

%Contract if the New Point is less than 0
while newPoint(1,1) < 0 || newPoint(1,2) < 0 || newPoint(1,3) < 0
    newPoint = (0.5.*newPoint) + (0.5 .* centroid);
end

%Now evaluate the new point to see if it is better than the old point

pattern1 = newPoint(1,1);
pattern2 = newPoint(1,2);
pattern3 = newPoint(1,3);

pattern4 = (391 - (3 * pattern1) - (226*pattern2) - (48 *
pattern3))/18;

d.setPattern(1, horzcat(pattern1, dummyPattern));
d.setPattern(2, horzcat(pattern2, dummyPattern));
d.setPattern(3, horzcat(pattern3, dummyPattern));
d.setPattern(4, horzcat(pattern4, dummyPattern));

d.openHydraulicAnalysis;
d.initializeHydraulicAnalysis;
%Run and close analysis
Series = d.getComputedTimeSeries;
d.closeHydraulicAnalysis

ModelTank1 = Series.Head(2,303);
ModelTank2 = Series.Head(2,308);
ModelTank3 = Series.Head(2,305);

Error = ((ModelTank1 - RealTank1)^2) + ((ModelTank2 - RealTank2)^2)
+((ModelTank3 - RealTank3)^2);

%if the new error is less than the old store the value
if Error <maxVal
    demandFactors(whereMax,:) = [pattern1,pattern2,pattern3];
    pointsInSimplex(whereMax) = Error;
%if the error is worse than the new point, contract the worst point
%towards the centroid
else

```

```

newPoint = (0.5 .* ph) + (0.5.*centroid);
%evaluate the new point
pattern1 = newPoint(1,1);
pattern2 = newPoint(1,2);
pattern3 = newPoint(1,3);

pattern4 = (391 - (3 * pattern1) - (226*pattern2) - (48 *
pattern3))/18;

d.setPattern(1, horzcat(pattern1, dummyPattern));
d.setPattern(2, horzcat(pattern2, dummyPattern));
d.setPattern(3, horzcat(pattern3, dummyPattern));
d.setPattern(4, horzcat(pattern4, dummyPattern));

d.openHydraulicAnalysis;
d.initializeHydraulicAnalysis;
%Run and close analysis
Series = d.getComputedTimeSeries;
d.closeHydraulicAnalysis

ModelTank1 = Series.Head(2,303);
ModelTank2 = Series.Head(2,308);
ModelTank3 = Series.Head(2,305);

Error = ((ModelTank1 - RealTank1)^2) + ((ModelTank2 -
RealTank2)^2) +((ModelTank3 - RealTank3)^2);

%store some of these new values
demandFactors(whereMax,:) = [pattern1,pattern2, pattern3];
pointsInSimplex(whereMax) = Error;

end

end
end

```

REFERENCES

- Box, M.J. (1965). A New Method of Constrained Optimization and a Comparison With Other Methods. *Comput. J.*, 8, 42-52.
- Bhave, P. R., & Gupta, R. (2013). Analysis of Water Distribution Networks. Narosa Publishing House.
- Cooper, James P. (2021). "Let's discuss Digital Twins." *Journal AWWA*, vol. 113, no. 7, pp. 66–68, <https://doi.org/10.1002/awwa.1769>.
- Cooper, J. P., Jackson, S., Kamojjala, S., Owens, G., Szana, K., & Tomić, S. (2022). Demystifying digital twins: Definitions, applications, and benefits. *Journal AWWA*, 114(5), 58–65. <https://doi.org/10.1002/awwa.1922>
- De Feo, Giovanni, et al. (2013). "Historical and technical notes on aqueducts from prehistoric to medieval times." *Water*, vol. 5, no. 4, pp. 1996–2025, <https://doi.org/10.3390/w5041996>.
- Eliades D.G., Kyriakou, M., Vrachimis S., Polycarpou, M.M. (2016). "EPANET-MATLAB Toolkit: An Open-Source Software for Interfacing EPANET with MATLAB", in *Proc. 14th International Conference on Computing and Control for the Water Industry (CCWI)*, The Netherlands, p.8. (doi:10.5281/zenodo.831493)
- Epp, R., & Fowler, A. G. (1970). Efficient code for steady-state flows in networks. *Journal of the Hydraulics Division*, 96(1), 43–56. <https://doi.org/10.1061/jycejaj.0002316>
- French, K.D., & Duffy, C.J. (2010). Prehispanic water pressure: A New World first. *Journal of Archaeological Science*, 37, 1027-1032.
- Gautam and Ormsbee. (2023). Prediction of Chlorine and Disinfection Byproduct Concentration In Water Distribution Systems Using KYPIPE and TTHM Regression Models: Application to Two Systems In Kentucky. *Theses and Dissertations--Civil Engineering*. 134.
- Georgiev, G. (2019). "'all Models Are Wrong' Does Not Mean What You Think It Means." *Medium*, The Startup, medium.com/swlh/all-models-are-wrong-does-not-mean-what-you-think-it-means-610390c40c9c.

- Grieves, M.W., & Vickers, J.H. (2016). Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems.
- Huang, M. (2019). Consider the value of hydraulic modeling for operators. *Opflow*, 45(7), 16–17. <https://doi.org/10.1002/opfl.1216>
- KYPipe LLC. (2022). *KYPIPE*. KYPIPE.
- Lagarias, J.C., Reeds, J.A., Wright, M.H., & Wright, P.E. (1998). Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. *SIAM J. Optim.*, 9, 112–147.
- Larson, T. E., & Skold, R. V. (1957). Corrosion and tuberculation of Cast Iron. *Journal AWWA*, 49(10), 1294–1302. <https://doi.org/10.1002/j.1551-8833.1957.tb16946.x>
- Masi, C. (2020). What is the cause behind measurement noise in sensors?. <https://www.researchgate.net/post/What-is-the-cause-behind-measurement-noise-in-sensors>.
- Martin, D. W., & Peters, G. (1963). The application of Newton’s method to network analysis by digital computer. *J. Inst. Water Eng*, 17(2), 115-129.
- Ormsbee, L. E., & Lingireddy, S. (1997). Calibrating hydraulic network models. *Journal AWWA*, 89(2), 42–50. <https://doi.org/10.1002/j.1551-8833.1997.tb08177.x>
- Ormsbee, L., & Walski, T. (2016). Darcy-Weisbach versus Hazen-Williams: No calm in West Palm. *World Environmental and Water Resources Congress 2016*. <https://doi.org/10.1061/9780784479865.048>
- Ormsbee, L.E. (1979). Optimization of Hydraulic Networks Using the Box-Complex Optimization Technique and the Linear Method of Hydraulic Analysis.
- Ormsbee, L. E. (2006). The history of Water Distribution Network Analysis: The computer age. *Water Distribution Systems Analysis Symposium 2006*. [https://doi.org/10.1061/40941\(247\)3](https://doi.org/10.1061/40941(247)3)
- Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Numerical recipes: The art of scientific computing. Cambridge University Press.

Qatium. Qatium.app. (2023). <https://qatium.app/>

Rossman, L., Woo, H., Tryby, M., Shang, F., Janke, R., & Haxton, T. (2020). *EPANET 2.2 User Manual*. U.S. Environmental Protection Agency.

Savić, D.A., Kapelan, Z., & Jonkergouw, P.M. (2009). Quo vadis water distribution model calibration? *Urban Water Journal*, 6, 22 – 3

Sedlak, D. L. (2014). *Water 4.0: the past, present, and future of the world's most vital resource*. New Haven, Yale University Press

The MathWorks Inc. (2022). MATLAB version: 9.13.0 (R2022b), Natick, Massachusetts: The MathWorks Inc. <https://www.mathworks.com>

Todini, E., & Rossman, L. A. (2013). Unified Framework for deriving simultaneous equation algorithms for water distribution networks. *Journal of Hydraulic Engineering*, 139(5), 511–526. [https://doi.org/10.1061/\(asce\)hy.1943-7900.0000703](https://doi.org/10.1061/(asce)hy.1943-7900.0000703)

Todini, E., and Pilati, S. (1987). A Gradient Method for the Analysis of Pipe Networks. International Conference on Computer Applications for Water Supply and Distribution, Hertfordshire, England.

Saša, T., et al. (2022). “Digital Twins: Case Studies in Water Distribution Management.” *Journal AWWA*, vol. 114, no. 8, pp. 44–56, <https://doi.org/10.1002/awwa.1979>.

Tripathi, S., Mack, M., Byland, A., Chamberlain, L., & Shumate, C. (2021). Houston Public Works’ Journey Toward a Digital Twin. *Journal AWWA*, 113(8), 80–84. <https://doi.org/10.1002/awwa.1793>

Walski, T. M., Lowry, S., & Rhee, H. (2012). Pitfalls in calibrating an EPS model. *Building Partnerships*. [https://doi.org/10.1061/40517\(2000\)198](https://doi.org/10.1061/40517(2000)198)

Walski, T. (2017). Procedure for hydraulic model calibration. *Journal AWWA*, 109(6), 55–61. <https://doi.org/10.5942/jawwa.2017.109.0075>

Wood, D.J., & Charles, C.O. (1972). Hydraulic Network Analysis Using Linear Theory. *Journal of Hydraulic Engineering*, 98, 1157-1170.

Wood, D.J., & Rayes, A. (1981). Reliability of Algorithms for Pipe Network Analysis. *Journal of Hydraulic Engineering*, 107, 1145-1

WRIS Portal-Kentucky Infrastructure Authority. (2023). <https://wris.ky.gov/portal/DwSysData/KY0780241>

VITA

Aidan Gill

EDUCATION

Bachelor's degree in Civil Engineering at the University of Kentucky, Lexington, Kentucky, May 2022.

PROFESSIONAL EXPERIENCE

Graduate Research Assistant, May 2022-Present
Department of Civil Engineering – University of Kentucky
Lexington, Kentucky

Student Summer Intern, May 2021 – August 2021
Cleveland Division of Water
Cleveland, Ohio

Quality Assurance and Environmental Engineering Intern, May 2019 – August 2019
Kinetico Water Systems
Newbury, Ohio

ORGANIZATIONS

Water Professionals Student Chapter at University of Kentucky (WPSC)
Treasurer, August 2022-December 2023

American Water Works Association (AWWA)
Student Member, August 2022-Present

University of Kentucky Mens's Rugby
Treasurer, Vice President, Captain (roles varying by year)
August 2018-December 2022