Photocopy and Use Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes maybe granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature 

Date 07/23/19

Reactor Transient Classification Using Machine Learning

By

Pedro Arturo Mena

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Nuclear Engineering

Idaho State University

Summer 2019

Committee Approval

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of Pedro Arturo Mena find it satisfactory and recommend that it be accepted.

Dr. Leslie Kerby, Major Advisor

Dr. R.A Borrelli, Committee Member (University of Idaho)

Dr. David Beard, Graduate Faculty Representative

Table of Contents

Abstract

This thesis explores the use of machine learning in identifying nuclear reactor transients. Models were produced using six supervised learning techniques. Due to the nature of nuclear power plants, synthetic data was gathered using a reactor simulator. Data was collected on four different transients and on normal operations. Transients were examined using a combination core life and power output. The Python TPOT package was used to preprocess data, as well as build and validate models. The results of the test showed that the decision tree model produced the best results with an accuracy of 98.6%, as well as high scores on the rest of the validation measurements. The other models also performed well with scores in the mid-90s. These high validation results show that machine learning has potential to be used as a tool to assist reactor operators in diagnosing reactor transients earlier and more accurately. This could aid in accident mitigation and prevention during operations at a facility.

Introduction

Artificial Intelligence (AI) and other similar concepts are a popular area of research in the 2010's. Private industries and governments are looking at using AI to better predict market conditions, equipment status, preventive maintenance and even increase and improve automation processes. An example of this would be Coca-Cola's use of AI in product development and marketing. This AI collects data from soft drink fountains that allow the user to mix a customized drink. This data is then used to determine which mixes may make successful products at a retail store [1]. Machine learning is a part of AI that has recently begun to be explored, improved and implemented in various fields including the nuclear industry.

The purpose of this paper is to explore the ability of machine learning algorithms to identify transient events occurring within a nuclear reactor. In order to perform this experiment it was necessary to gather nuclear reactor data to use in creating machine learning models. Due to the extremely high cost of nuclear reactors, as well as the possible serious health and environmental consequences of a nuclear power plant accident, it is impossible to stage real-life situations where an actual nuclear reactor under goes serious transient events or accident conditions that could be used to gather data. Instead, a reactor simulator will be used in order to gather the synthetic data needed to apply the machine learning algorithms. Once the data is gathered, it will be necessary to modify and format the data in a form where the machine learning algorithms can understand and interpret it. This will be done using data science packages such as Pandas and NumPy within the Python programming language. Once the data has been properly

formatted the data will be applied using machine learning algorithms within two python machine learning packages: TPOT and scikit-learn. This will produce several different machine learning models. Finally, each machine learning model created will then be validated using measurements such as accuracy and precision**.**

Literature Review

Data is becoming more and more central in many industries across the world. Retailers are using data to better plan promotions and target customers, i.e., marketing efforts are more focused on consumer data for targeting key demographics, etc. One industry that has recently begun using data to affect operations is the nuclear industry. Many of these efforts have focused on the use of AI and machine learning to perform a variety of functions within the industry.

One of the most promising data applications within the nuclear industry is the use of AI and machine learning for preventive maintenance. Shutdowns in the nuclear industry can be extremely expensive and time consuming. An unexpended failure can lead to several months of ceased operations, but the hope is that with the use of AI, problems can be found ahead of time, leading to quicker repair times and shorter down times. One such effort to develop this ability is happening at Purdue University. Researchers at the university have proposed using a "deep learning" framework, a form of AI, to identify cracks within a nuclear reactor [2]. The AI is "trained" using data from over 300,000 crack and non-crack patches. Then, the AI uses video images from within the reactor itself to determine if there is a crack forming or not. According to Dr. Mohammad Jahanshahi, of Purdue, the AI was able to identify cracks correctly 98.3% of

the time, a much higher accuracy than other techniques that are currently in use. The full paper on this has been published by IEEE [3]. Figure 1 shows an example of an image examined by the AI.

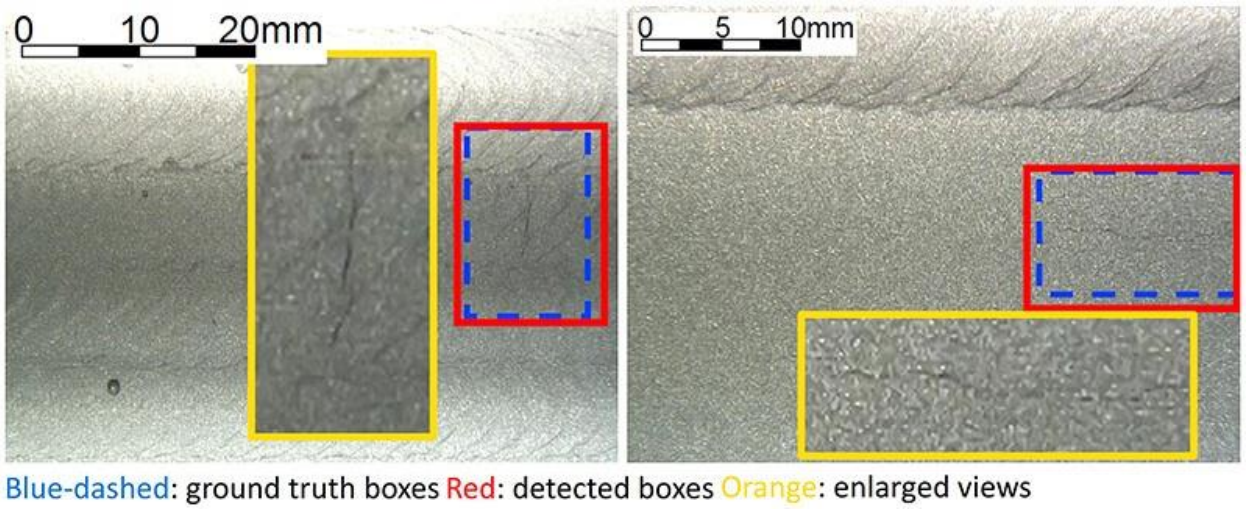


Figure 1: Photo of Crack Analysis Using Machine Learning at Purdue [2]

Other applications for the use of AI and machine learning are being explored across the country. In June of 2018, North Carolina State University (NCSU) was awarded a $3.4 million grant by the United State Department of Energy (DOE) to develop a control system that will rely more on AI. The plan is for AI to use machine learning to analyze the data that is collected from the reactor and alert the operator. It should be stressed that the operator will also have control of reactor functions. TerraPower is hoping to collaborate with NCSU to determine the best areas to place many of the sensors and instruments where they will be most effective to gather data [4]. DOE has listed cost reduction through less staff, as well as better training to be goals for this project [5]. Another project in progress involves collaboration between the University of Wisconsin-Madison and Oak Ridge National Laboratory. A machine

learning program has been designed to analyze and detect damage to materials caused by radiation. Figure 2 shows some examples of the classifications identified during this experiment [6].

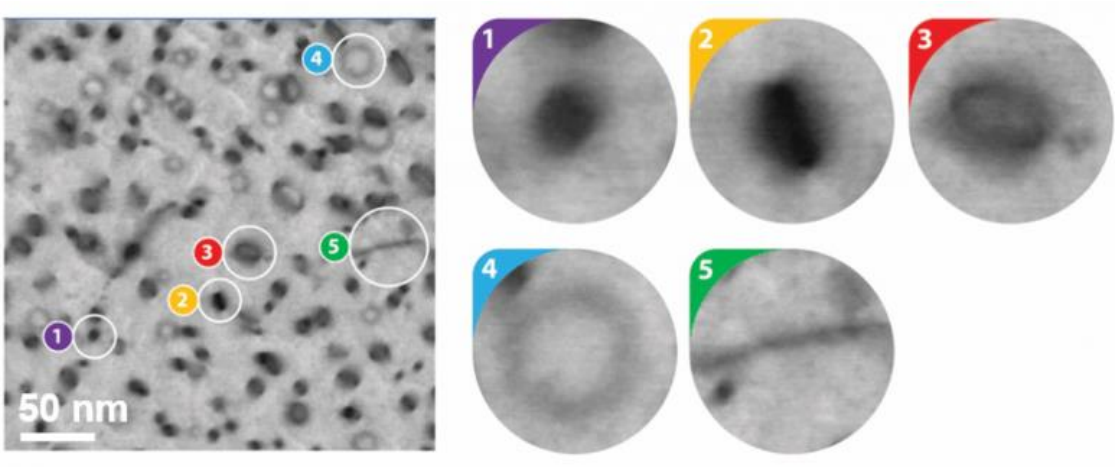


Figure 2: Identification of Material Radiation Damage Using Machine Learning at University of Wisconsin-Madison [6]

One hurdle that these techniques face is the gathering of data. The majority of nuclear power plants in the Unites States predate the introduction of digital technology. As such, the digitalization of existing nuclear power plants is essential in order to properly gather and store the data that will be needed to use machine learning and other AI techniques. To this end, groups such as Nuclear Energy Institute (NEI) have been promoting digitalization and other efficiency improvement measures. The greatest incentive to promote the implementation of digital equipment and use of AI is the increased boost in profit margin. A study done by McKinsey Global Institute found that industries that implemented the use of these technologies, either proactively or partially, saw an increasing profit margin. Figure 3 shows a graphical comparison from the study [7].

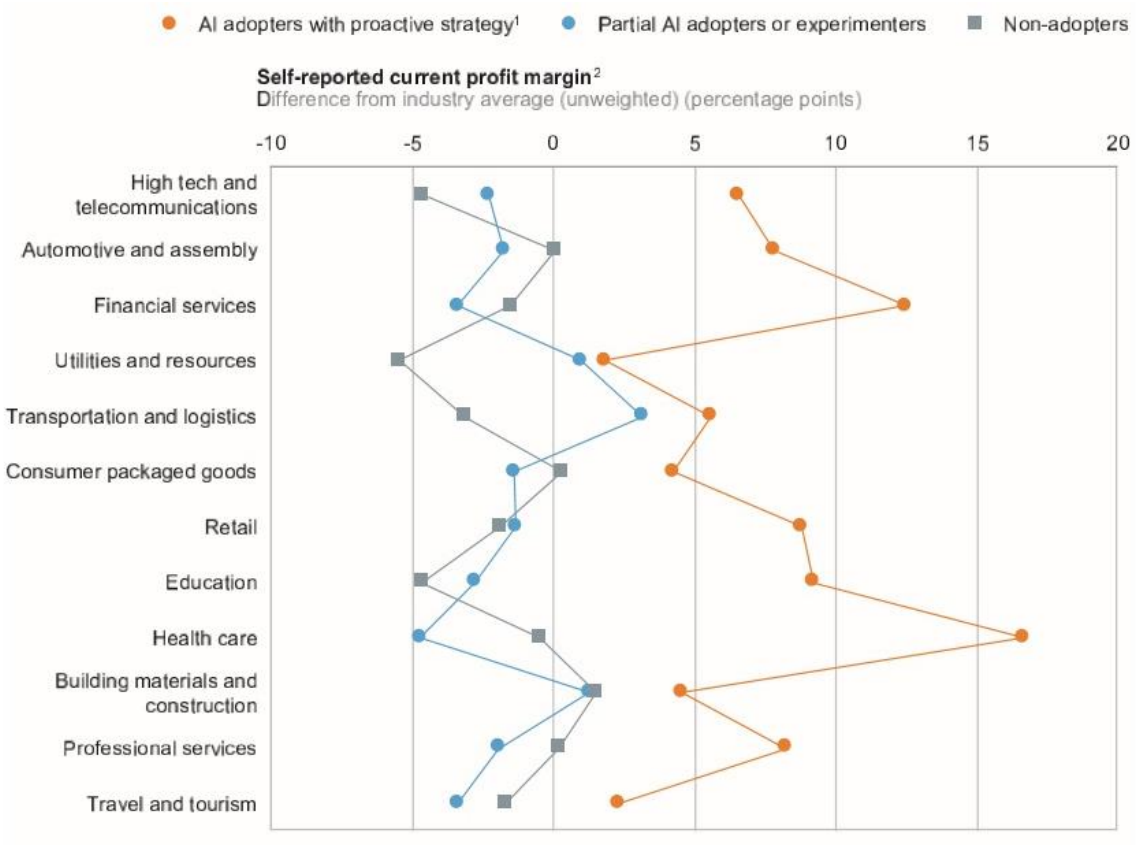


Figure 3: Profit Margin Comparison of Industries that have adopted AI [7]

Companies such as General Electric (GE) have also begun looking into the use of machine learning and AI for their systems. One of the applications of AI and machine learning that GE is looking at is the use of data to better inform management. GE hopes that techniques will result in better information and as a result, better managerial decisions will be made. This will lead to increased performance and better cost efficiency [8].

Background

Machine Learning

Machine learning is an area of study in the field of computer science. The purpose of machine learning is to teach computers to make predictions based on patterns and data that are provided to the computer. This is a critical part of the AI system as it controls the decision-making aspect of a machine. There are many different methods that can be used in machine learning to make predictions. These methods are known as machine learning algorithms or models.

Machine learning systems have several advantages over preprogrammed computer systems. Where a preprogrammed system only performs the functions that were initially put in the system, machine learning systems can make changes and adapt to outside changes by analyzing new data. Machine learning can also be used to find relationships between different types of data as part of a data mining process. This does introduce the possibility of mistakes or errors occurring [9]. Today, a variety of industries make use of machine learning in their business model. These include: marketing, finance, insurance, casinos, retail, manufacturing, research & development, science, transportation, politics, etc. As the world becomes more data driven, machine learning will continue to be used in more and more applications.

Supervised Learning

Machine learning algorithms are built using a variety of methods. This project focused on building models using the supervised learning technique. Supervised learning

is a form of learning by example. This method uses data that already has an outcome associated with it. This is usually in the form of an output or label. An example of this would be a dataset that categorizes a set of features as good, fair, poor, etc. A supervised learning model will develop criteria to make classifications based on the provided labels. This is sometimes referred to as the supervisor. These criteria can be created in a variety of ways, such as using probabilities derived from the training features [10]. An advantage of supervised learning is that it allows the user to specify the desired output labels and the number of labels to train the model on. It should be noted that supervised models have an increased tendency of model overfitting and training these models can be computationally expensive in some cases.

Python Packages

NumPy

NumPy is a Python library that has been designed with several different applications. These include: the ability to integrate C/C++ and FORTRAN code into Python, preform linear algebra functions and easy integration with datasets. NumPy was developed by Dr. Travis Oliphant in 2005 as a successor to Numeric and Numarray. NumPy development is done through the NumFocus Foundation, a nonprofit organization. Today, NumPy is used by many companies and organizations, such as Netflix and NASA [11]. NumPy was designed with scientific computing in mind, but the package also has the ability to help with database construction and manipulation. NumPy is a free open-source package and is included standard in many Python distributions such as Cygwin and Anaconda or the library can be downloaded using Pip. Many of the

packages used in this project, such as TPOT and Pandas, use NumPy arrays to perform numerical operations. At the time of this project, the most current version of NumPy was version 1.17.0, which was released in May of 2019. NumPy is able to operate in a Windows, Linux or Mac OS environments [12].

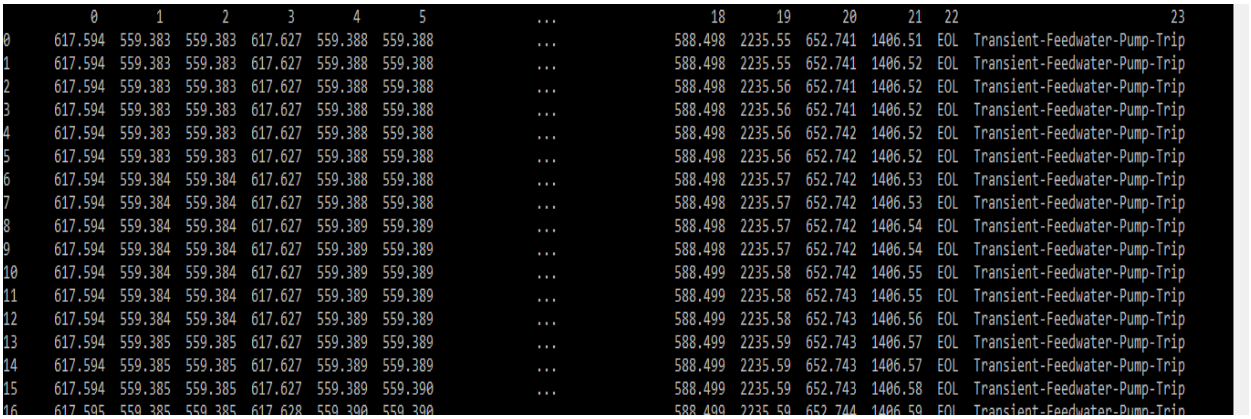
One of the most important features in the NumPy library is the ability to create NumPy arrays also referred to as `ndarrays`. A NumPy array is a container that allows for the storage of several different elements. NumPy arrays, while similar to a Python list, have a few key differences. The first is the ability to operate quicker and take up less memory than a Python list. This is due to better integration with C/C++ which helps mitigate the loss of efficiency that higher-level and easier-to-use languages typically have. Also, NumPy has been optimized for linear algebra operations. NumPy arrays are considered homogenous, meaning that all data is the same size and is processed the same way, regardless of any differences between elements. These elements are described by a `dtype` object that can be built using different data types. Every NumPy array has a `dtype` object associated with it. This can tell the user the descriptive information about the NumPy array, such as type, memory usage, etc. Elements used in code are taken from the array using indexing. The index represents an object scaler which was part of the NumPy development.

Pandas

Pandas is a free, open-source Python package, which aims to help users with data manipulation, modification and analysis. The package can be downloaded on most Python distributions, such as Anaconda or via Pip. The package was initially developed

by Wes McKinney in 2008 in response to a need for better data tools and development is ongoing as of 2019. The project receives funding and support from the University of Paris Saclay Center for Data Science, as well as from Two Sigma. Pandas requires Python version 2.7 or greater but support for all Python 2 versions will be dropped on January 1st of 2020. The most current version, as of May 2019, is version 0.24.1, released in February of 2019. Pandas only requires the NumPy package to operate properly. Pandas is designed to work with Windows, Mac OS and Linux environments [14].

The goal of the Pandas project is to provide data tools to users in Python. In the past, the adaptation of Python in data science and statistics had been slow as users had preferred to use tools such as MATLAB and R. Pandas has the ability to read and convert datasets, typically in a CSV format, into a structured dataset known as a Pandas `DataFrame`. A Pandas `DataFrame` is a 2-dimensional Python list that allows users to store values in a tabular form. An example of a `DataFrame` is shown in Figure 4. Like most Python packages, Pandas is considered high-level and there are tradeoffs in efficiency for ease-of-use. Cython was used to mitigate this issue. Pandas has the ability to help the user identify and manage missing values, a common issue in data science. The package also has the ability to group or sort data by specified user input, such as individual values within the dataset or data types like floats or strings. Other useful Pandas functions include the ability to change large groups of data, perform statistical analysis such as mean, median and standard deviations over the dataset, as well as add, remove and combine parts of different datasets [15].



	0	1	2	3	4	5	...	18	19	20	21	22	23
0	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.55	652.741	1406.51	EOL	Transient-Feedwater-Pump-Trip
1	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.55	652.741	1406.52	EOL	Transient-Feedwater-Pump-Trip
2	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.741	1406.52	EOL	Transient-Feedwater-Pump-Trip
3	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.741	1406.52	EOL	Transient-Feedwater-Pump-Trip
4	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.742	1406.52	EOL	Transient-Feedwater-Pump-Trip
5	617.594	559.383	559.383	617.627	559.388	559.388	...	588.498	2235.56	652.742	1406.52	EOL	Transient-Feedwater-Pump-Trip
6	617.594	559.384	559.384	617.627	559.388	559.388	...	588.498	2235.57	652.742	1406.53	EOL	Transient-Feedwater-Pump-Trip
7	617.594	559.384	559.384	617.627	559.388	559.388	...	588.498	2235.57	652.742	1406.53	EOL	Transient-Feedwater-Pump-Trip
8	617.594	559.384	559.384	617.627	559.389	559.389	...	588.498	2235.57	652.742	1406.54	EOL	Transient-Feedwater-Pump-Trip
9	617.594	559.384	559.384	617.627	559.389	559.389	...	588.498	2235.57	652.742	1406.54	EOL	Transient-Feedwater-Pump-Trip
10	617.594	559.384	559.384	617.627	559.389	559.389	...	588.499	2235.58	652.742	1406.55	EOL	Transient-Feedwater-Pump-Trip
11	617.594	559.384	559.384	617.627	559.389	559.389	...	588.499	2235.58	652.743	1406.55	EOL	Transient-Feedwater-Pump-Trip
12	617.594	559.384	559.384	617.627	559.389	559.389	...	588.499	2235.58	652.743	1406.56	EOL	Transient-Feedwater-Pump-Trip
13	617.594	559.385	559.385	617.627	559.389	559.389	...	588.499	2235.59	652.743	1406.57	EOL	Transient-Feedwater-Pump-Trip
14	617.594	559.385	559.385	617.627	559.389	559.389	...	588.499	2235.59	652.743	1406.57	EOL	Transient-Feedwater-Pump-Trip
15	617.594	559.385	559.385	617.627	559.389	559.390	...	588.499	2235.59	652.743	1406.58	EOL	Transient-Feedwater-Pump-Trip
16	617.595	559.385	559.385	617.628	559.390	559.390	...	588.499	2235.59	652.744	1406.59	EOL	Transient-Feedwater-Pump-Trip

Figure 4: Example of a DataFrame From Project

Scikit-learn

This project makes great use of the scikit-learn Python package. The package is a free, open-source package that can be downloaded through Pip or a Python distribution, such as Anaconda. Scikit-learn was developed by Dr. David Cournapeau in 2007 as a summer project for a Google Summer of Code Project. The purpose behind the project was to design a system that could run complicated machine learning algorithms using Python and maintain a user-friendly intuitive interface. The first public release of scikit-learn was released on February 1st, 2010 and the French Institute for Research in Computer Science and Automation (INRIA) began heading the project. Today, scikit-learn development and research is funded by universities, such as the New York University, University of Sydney, and Columbia University, among others [16]. Many companies use scikit-learn as part of their information system operations, including: JP Morgan, Spotify, Booking.com and Change.org. Some of these applications include predicting user's preference in music, credit and market trend analysis, as well as targeting users with more customized add-ons and specials [17].

Similar to many other Python packages, scikit-learn makes use of many modern C++ libraries using Cython, a programming language designed to help bridge C and Python code. Scikit-learn has been designed to be compatible in both, a Windows or Linux environment. The latest version of scikit-learn 0.21 requires Python version 3.5 or higher. Scikit-learn relies on three Python packages to run, NumPy, SciPy and Joblib, which allows the package to be easily distributed and used. While not required, Pandas is needed in order to take full advantage of the abilities of the scikit-learn package. In the past, the package has focused on remaining easy to use and efficient, rather than adding new features. Though, recently scikit-learn has been updated with new features that assist in data exploration using Pandas. This includes better ways of dealing with missing values with the `SimpleImputer` function. Since Python is a high-level programming language, there are tradeoffs in code efficiency for ease of use. Steps have been taken in order to manage and mitigate most of these issues: the specification of objects through interface rather than inheritance, the use of Cython to increase the efficiency of using C++ libraries within Python, and others [18].

Scikit-learn has the ability to perform several different types of machine learning algorithms: supervised learning methods, such as classification and regression, as well as unsupervised methods, like k-means clustering. Currently, scikit-learn has functions to perform 17 different types of supervised machine learning methods, as well as 9 different unsupervised methods. The package has been designed with functions that help with data preparation, such as splitting datasets for validation purposes. Scikit-learn also has several functions for data preprocessing, such as the standard scaler function and model validation and scoring through measurements, such as accuracy, precision, and goodness

of fit. Finally, scikit-learn can make use of Python's Matplotlib package in order to help users visualize the results of the models generated. This includes clustering graphs, confusion matrices, etc. [19]

TPOT

As is the trend with most industries, the idea of implementing automated processes to improve ease-of-use, results and performance is occurring with machine learning. This is sometimes referred to as AutoML. New packages and software have been developed to automate areas of machine learning that are complex and/or time consuming, such as data preprocessing, feature selection and model selection. While this simplifies the process, it can be computationally expensive. Python packages that are currently available include: Auto-WEKA/auto-sklearn, H_2O and TPOT. Google is also developing its own cloud-based AutoML software, called Cloud AutoML, to try and open machine learning to non-data-scientists. Services available include photo and video analysis/modeling, language translation and data analysis [20].

The supervised learning models for this project were created using the Tree-based Pipeline Optimization Tool (TPOT) package in Python. TPOT was chosen for this project because it is one of the more mature AutoML Python packages available. Also, it is simple to use and can evaluate a number of different machine learning models. TPOT was developed by the Computational Genetics Lab at the University of Pennsylvania with support from the National Institute of Health. Development began in 2011 and the package continues to be developed by Epistasis Lab at the University of Pennsylvania. TPOT is open-source and is available for download from the lab's GitHub repository for

free. TPOT was developed in response to the growing demand and interest in machine learning applications. The process of creating a machine learning model can be complex and time consuming, even if just limited to supervised learning. There are multiple models that can be created, as well as many different methods of data preprocessing. It can be difficult for even an experienced data scientist to develop the best possible model. The purpose of the TPOT package is to simplify and automate parts of the machine learning process, while providing better results due to improved data preprocessing and the use of multiple different supervised learning methods [21].

TPOT is designed to make use of the scikit-learn Python package for both, data preprocessing and model construction. As such, the user is required to have the scikit-learn package installed and imported into the program. TPOT also makes use of NumPy arrays and Pandas `DataFrames` and these packages are required as well. The DEAP, SciPy, tqdm, stopit, and update_checker packages are also needed. These packages are all available for free via download and can be configured with the Anaconda Python distribution using Pip. Other Python distributions can be used if the pywin32 module is used. It should be noted that Epistasis Lab strongly recommends that Python 3 be used rather than Python 2.

TPOT is designed to aid the user in data preparation for supervised learning. This includes automating feature preprocessing, selection and construction. Doing so can dramatically simplify the process of preparing data for use in machine learning algorithms and these steps can be situational, complex and time consuming. It is important to note that the TPOT package requires the user to do data examination on the data to be used in the supervised learning process. The package cannot account for

missing values, qualitative data and incorrectly formatted datasets. TPOT uses 9 different preprocessing methods from the scikit-learn Python package: Binarization, Feature Agglomeration, Maximal Absolute Scaling, Minimum-Maximum Scaling, Normalization, Principal Component Analysis, Robust Scaling, Standard Scaling and RBF kernel sampling. More detail on these methods will be provided in later sections. These methods are all implemented within TPOT using scikit-learn functions.

TABLE 1: TPOT DATA PREPROCESSING/ FEATURE SELECTION METHODS		
Binarization	Feature Agglomeration	Maximal Absolute Scaling
Minimum-Maximum Scaling	Normalization	Principal Component Analysis,
Robust Scaling	Standard Scaling	RBF Kernel Sampling.
Select Family Wise Error	Select Percentile	Variance Threshold Selection

Dictionaries can be created so the user can specify which preprocessing techniques are implemented. In addition to data preprocessing, TPOT makes use of 3 feature selection methods: Select Family Wise Error, Select Percentile and Variance Threshold Selection. These methods also use scikit-learn functions run through TPOT and the user can define which methods will be applied. Again, more detail on these selection methods will be provided in later sections. The entire process of data preprocessing and selection is known as a pipeline. A visualization of a typical TPOT pipeline from the paper *TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning* is shown in Figure 5.

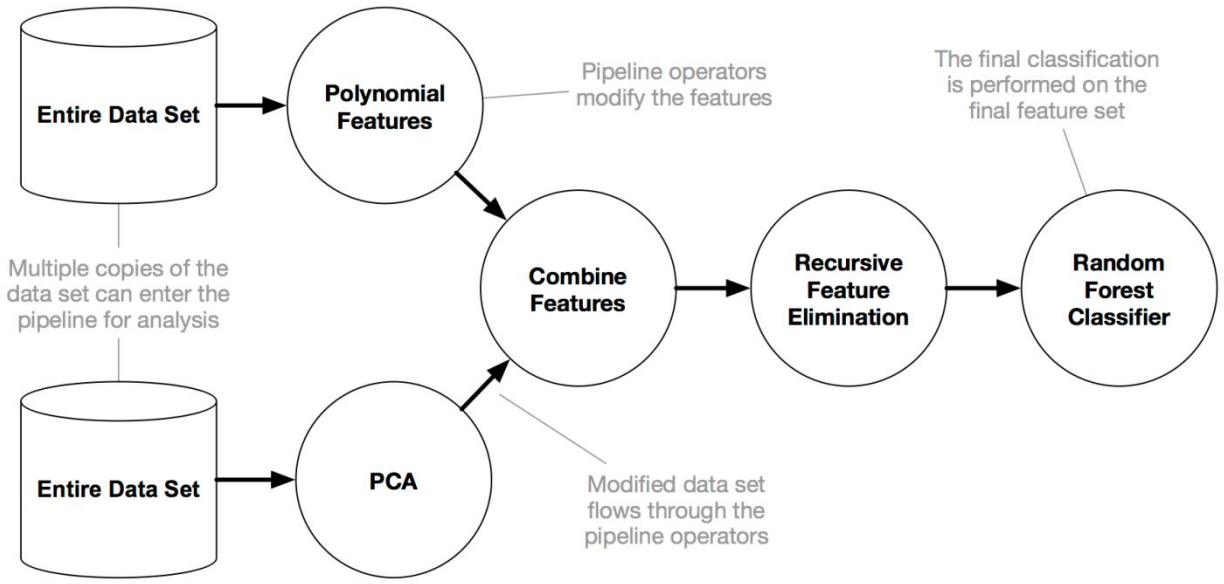


Figure 5: Example of a Typical TPOT Pipeline[22]

Once the data preprocessing has been completed, TPOT then creates and test several different machine learning models. TPOT can perform either a regression analysis, where the user is looking to determine some numerical value using a set of features, or a classification analysis where features are used to identify some value which represents an object, scenario, event, etc. For the purposes of this project the classification method was used. The TPOT Classifier creates and tests models using 6 different techniques: Gaussian naïve, Bayes, Bernoulli naïve Bayes, multinomial naïve Bayes, k-nearest neighbors, decision tree classification and logistic regression.

These methods all use the functions from the scikit-learn package. The user is able to use a TPOT dictionary to specify which methods should be tested when creating the model.

TABLE 2: TPOT MACHINE LEARNING MODELS		
Bernoulli Naïve Bayes	Multinomial Naïve Bayes	Gaussian Naïve Bayes
K-Nearest Neighbors	Decision Tree Classification	Logistic Regression.

The TPOT Classifier also allows the user to define the parameters of the model creation. One important parameter is the number of generations that will be used in the model creation. This is the number of iterations that will be used in the optimization process. Typically, the more generations run, the better the results will be but the process will take longer. Another important parameter that can be specified is the population size used. This number is the number of individual pipelines retained in each generation; again a larger population produces better results but increases the time needed to complete the model. Other parameters that the user can choose include the random number seed, the TPOT dictionary used in the model creation, and the verbosity, which is the amount of information shown during the model creation [22]. This project used: 100 generations, a population size of 100, 10 cross-fold validations and a random state of 5. The TPOT classification code used for this project is shown in Figure 6:

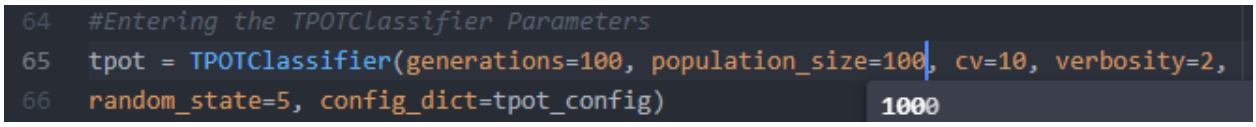


Figure 6: TPOT Classifier Used for Project

Once the parameters of the TPOT Classifier have been set by the user, the program will then take the test and train datasets and determine which model is optimal. This process is time consuming and can take hours to days to run depending on the parameters and dictionaries used. TPOT will output the type of model that is optimal, as well as a single validation measurement. By default, TPOT will provide the accuracy of the optimal model, but other forms of validation measurement, such as precision and recall, can be displayed using the sckit-learn functions. The optimal results can be stored for further use in the program and optimal pipeline can be exported for use at a later time.

It should be noted that the user will need to ensure that the correct file name is used to import the data in the exported pipeline, as the exported file will only use a placeholder for this. Also, depending on data configuration some slight modifications to the optimal model code may be needed to ensure that the data is properly read by the exported file. Figure 7, from the TPOT documentation shows the entire process the package applies in creating a model.

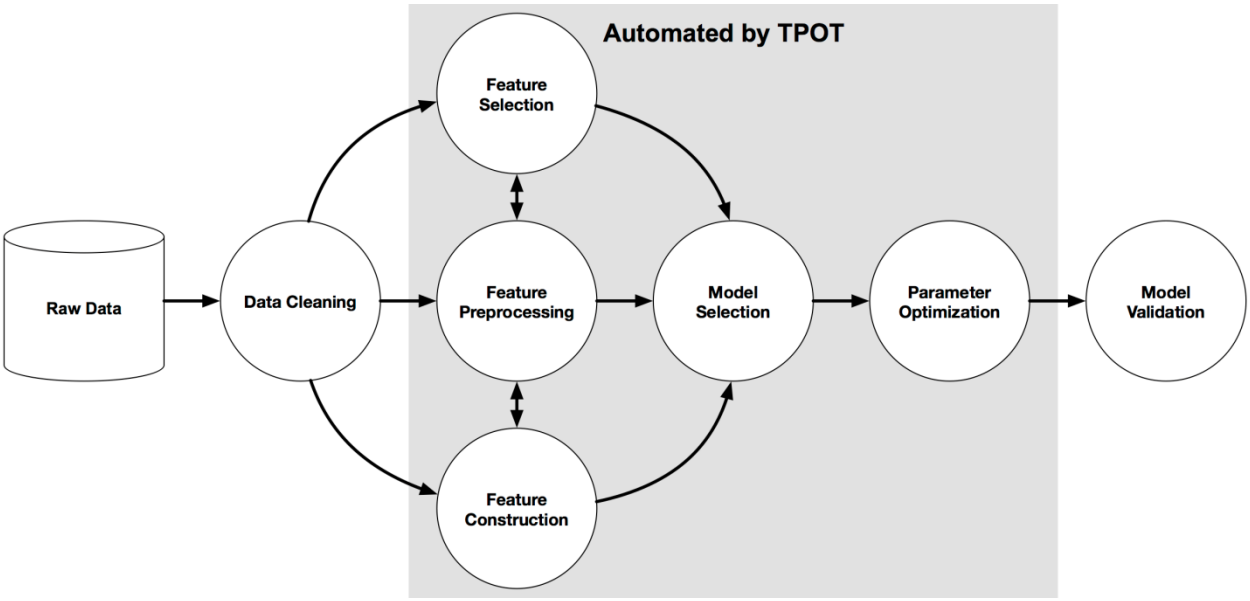


Figure 7: TPOT Machine Learning Process from TPOT Documentation [21]

Data Science Methods Used

Data Preprocessing

Binarization

Binarization is the process of taking numerical data and converting it into binary\boolean form. In machine learning, this method is most commonly used in preparing data for use with the Bernoulli naïve Bayes method, as this method requires data to be distributed in a binary form. There are other cases where this method could be applied. Binarization of data is particularly important in training image analyzers, where pixels are assigned a true or false value, based on the characteristics of the pixel [23].

Binarization relies on determining a threshold for the data that is to be converted. This threshold is the standard that determines if the data being converted is classified as a 1, a true value, or a 0, a false value. The threshold value is dependent on the data and context of the analysis being performed. The scikit-learn has a preprocessing method `.Binarizer()`. This method will convert the user-specified data into binary form. By default, the threshold value is set to 0. As such, a negative or 0 value will be assigned a 0 and a positive value will be assigned a 1. The user is able to change this threshold so that the method better fits the analysis. Figure 8, from the scikit-learn documentation, shows an example of the binarization method [24].

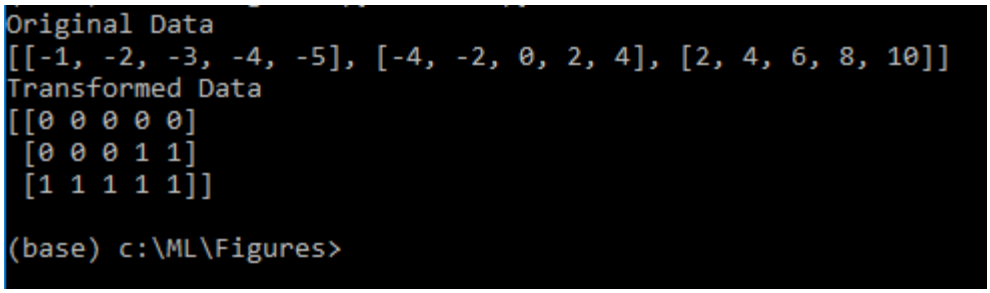


Figure 8: Binarization Example Using Scikit-learn

Standard Scaling

The final data preprocess technique that was used in the TPOT dictionary for this project was Standard Scaling. Standard Scaling removes the mean and then scales the dataset to its unit variance. The scaling of the individual data points, z, is given by the following equation:

$$z = \frac{X - \mu}{S}$$

Where:

μ is the average of the dataset

S is the standard deviation of the dataset

Standard Scaling is necessary for many machine learning algorithms that require centered data. Also, this prevents bias from features that are of different type. The scikit-learn Standard Scaling function `.StandardScaler()`. Figure 9 and 10 show a comparison of unscaled and standard scaled data.

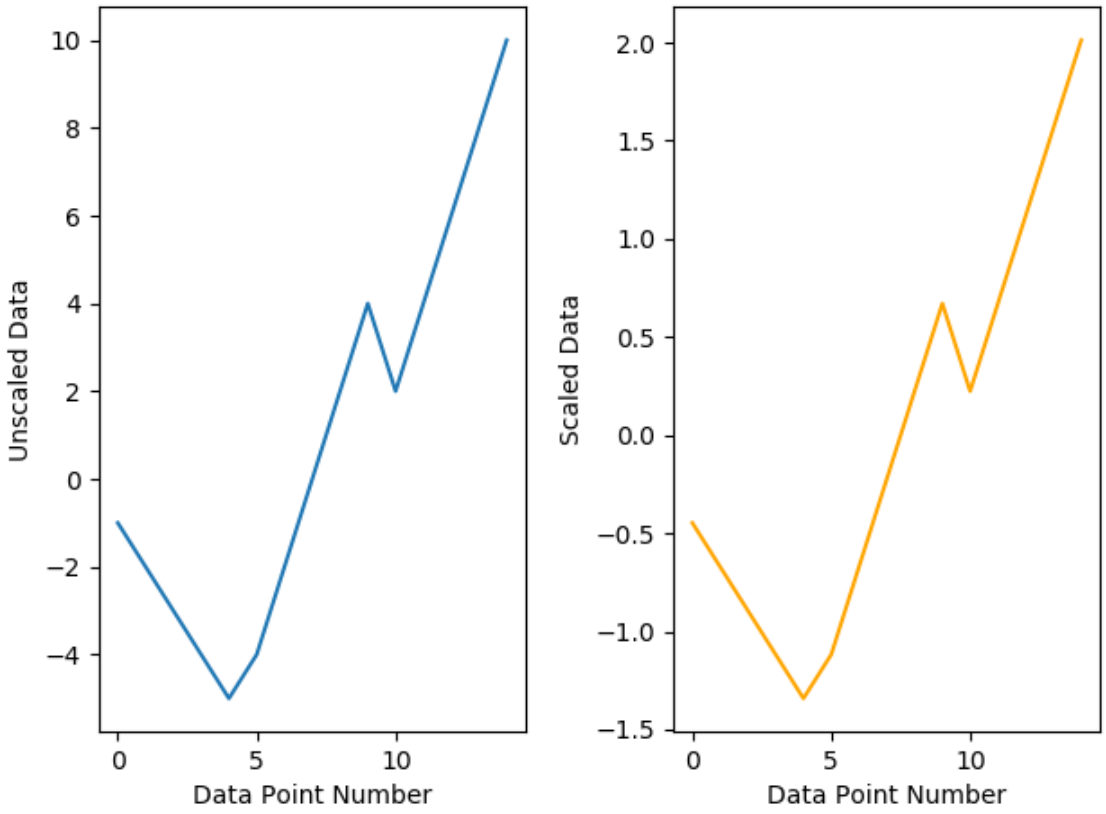


Figure 9: Comparison of Standard Scaled Data and Unscaled Data

```
Unscaled Data              Scaled Data
[[-1]                       [[-0.4472136 ]
 [-2]                       [-0.67082039]
 [-3]                       [-0.89442719]
 [-4]                       [-1.11803399]
 [-5]                       [-1.34164079]
 [-4]                       [-1.11803399]
 [-2]                       [-0.67082039]
 [ 0]                       [-0.2236068 ]
 [ 2]                       [ 0.2236068 ]
 [ 4]                       [ 0.67082039]
 [ 2]                       [ 0.2236068 ]
 [ 4]                       [ 0.67082039]
 [ 6]                       [ 1.11803399]
 [ 8]                       [ 1.56524758]
 [10]]                      [ 2.01246118]]
```

Figure 10: Example of Standard Scaled Data

Robust Scaler

The Robust Scaler method scales features using statistical methods. These methods are the unit variance or the standard deviation of the feature being scaled. These methods are intended to help deal with outlying data points to prevent machine learning models from becoming skewed. This is done by removing and storing the median of the feature, and then the scaling range is calculated using 1st and 3rd quartile ranges as bounds by default. The user is able to adjust these ranges to better meet the needs of the dataset. The final value of the scaled data point is calculated using the same equation as the Minimum Maximum Scaler, just with the chosen statistic, rather than the actual value. Figure 11, shows a comparison of data that is scaled using the Robust Scaling method and data that is unscaled. Figure 12 shows the output of the scaling code

The scikit-learn function for robust scaling is `.RobustScaler()`. The user can adjust the quartile range for the scaler and control if the data is centered prior to scaling. By default and for this project, the `.RobustScaler()` method will use the unit variance. The following equation is used for the `.RobustScaler()` method:

$$Scaled\ x_i = \frac{x_i - 1st\ quatrtile\ range(x)}{3rd\ quartile\ range(x) - 1st\ quartile\ range(x)}$$

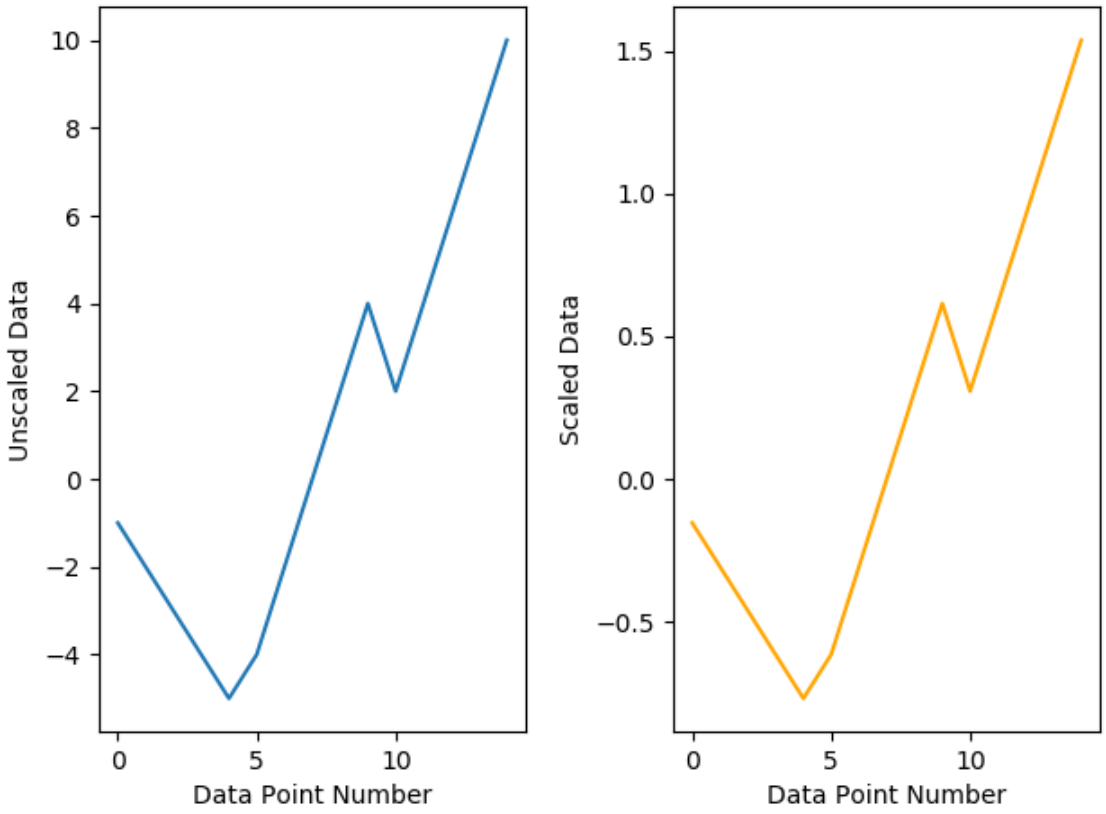


Figure 11: Comparison of Robust Scaled Data and Unscaled Data

```
Unscaled Data          Scaled Data
[[-1]                   [[-0.15384615]
 [-2]                   [-0.30769231]
 [-3]                   [-0.46153846]
 [-4]                   [-0.61538462]
 [-5]                   [-0.76923077]
 [-4]                   [-0.61538462]
 [-2]                   [-0.30769231]
 [ 0]                   [ 0.        ]
 [ 2]                   [ 0.30769231]
 [ 4]                   [ 0.61538462]
 [ 2]                   [ 0.30769231]
 [ 4]                   [ 0.61538462]
 [ 6]                   [ 0.92307692]
 [ 8]                   [ 1.23076923]
 [10]]                  [ 1.53846154]]
```

Figure 12: Example of Robust Scaled Data

Maximum Absolute Value Scaler

The Maximum Absolute Value Scaler method will scale and translate the values in a dataset so the max value in that set is 1. Scaling in this way helps manage bias that results if the features are of different measurements, such as temperature and pressure. One issue that needs to be considered is that this scaling method is sensitive to outliers and failure to properly deal with this may result in skewed machine learning models. The scikit-learn method, `.MaxAbsScaler()` will ignore missing values and they are not changed during this process. Figure 13 and 14 show a comparison of data that has not been scaled and data that has been scaled using the Maximum Absolute Value Scaler method [25].

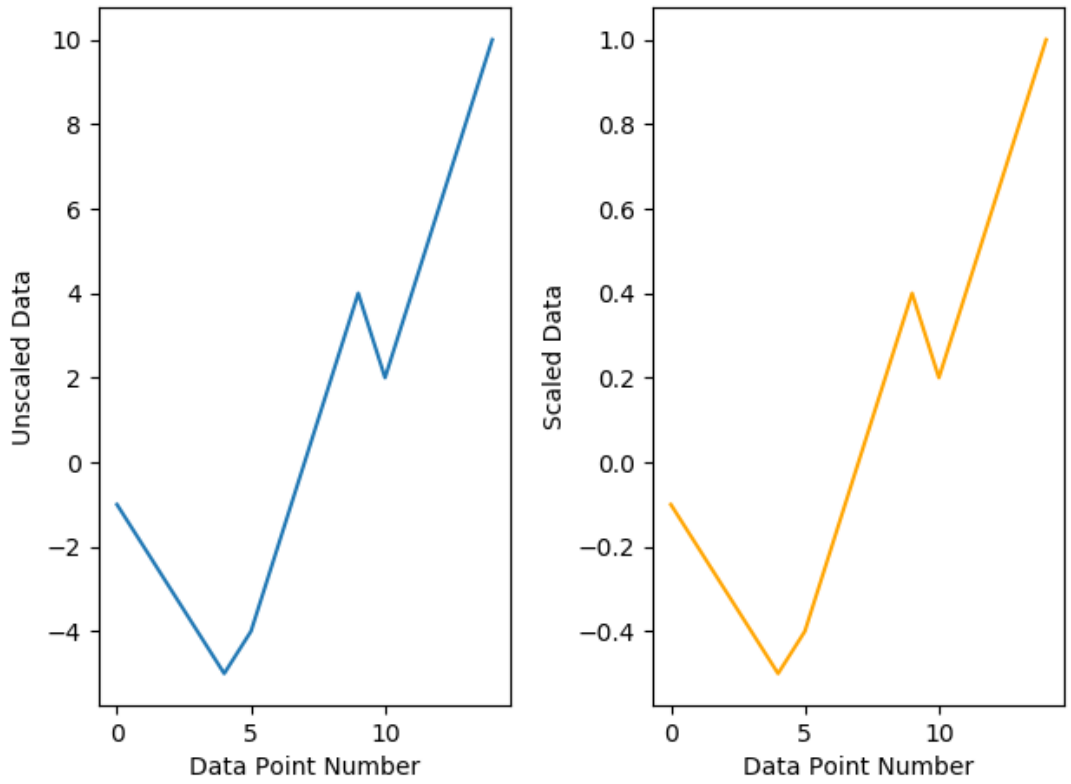


Figure 13: Comparison of Maximum Absolute Value Scaled Data & Unscaled Data

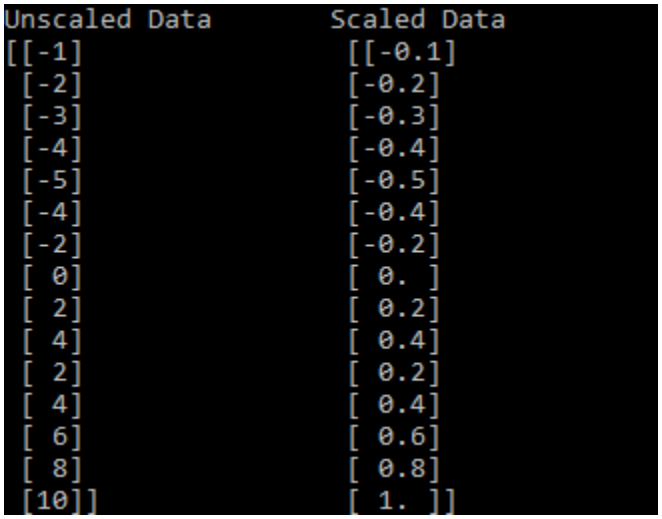


Figure 14: Example of Data Scaled by Maximum Absolute Value Method

Minimum Maximum Scaler

. The Minimum Maximum Scaling method allows the user to scale the data to a particular range. This range can be specified by the user to better fit the needs of the analysis. This project used the default range of 0 to 1. Each feature is scaled individually using the following equations:

$$X(Standard) = \frac{X - X(Minimum)}{X(Maximum) - X(Minimum)}$$

$$X(Scaled) = X(Standard) * (Maximum\ Bound - Minimum\ Bound) + Minimum\ Bound$$

Where:

X is the specific point in the dataset being scaled

Maximum Bound is the highest scaled value

Minimum Bound is the lowest scaled value

This method, like Maximum Absolute Scaler method, is useful to scale features that of different measurements to reduce feature bias in the model. This method is also sensitive to the presence of outliers which could lead to skewed machine learning models if not dealt with properly. Also, this method is useful for the multinomial naïve Bayes algorithm, as it scales negative values to values between 0 and 1, which are compatible with the algorithm. The scikit-learn method for the Minimum Maximum Scaler is `.MinMaxScaler()`. This method will ignore and leave untreated missing values from a dataset. Figure 15 shows a comparison of data that has been scaled using the `.MinMaxScaler()` method and unscaled data. Figure 16 shows the code output using the `.MinMaxScaler()` method.

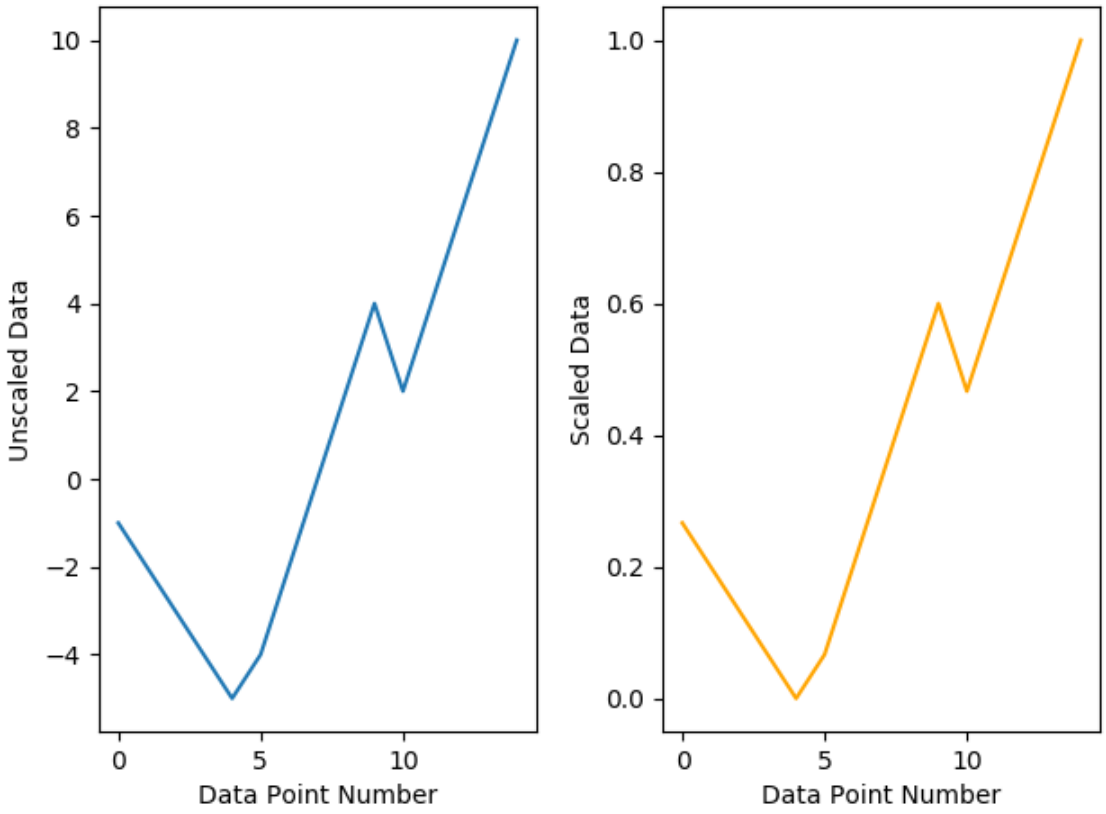


Figure 15: Comparison of Unscaled Data and Min Max Scaled Data

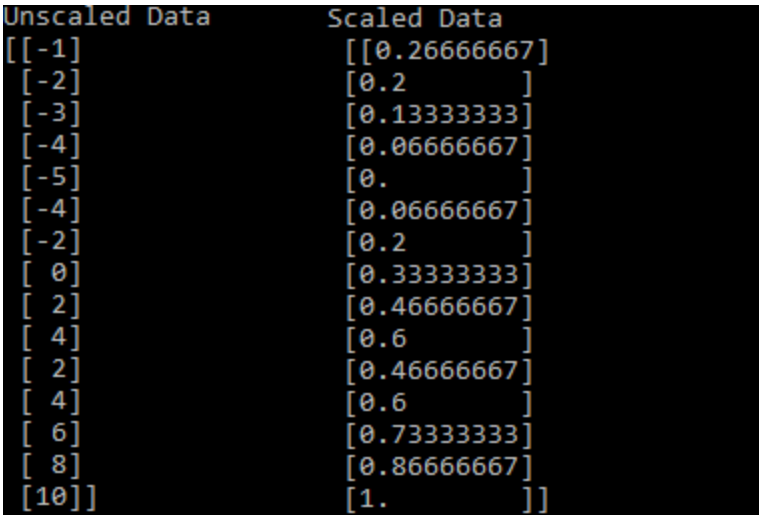


Figure 16: Example of Min Max Scaled Data

Normalization

The normalization method samples each feature independently to the unit norm. The scikit-learn method for normalization is `.Normalizer()`. The user is able to specify what type of regularization is used for the normalization, either L1 or L2. L1 is the sum of the regularization weights and L2 is the sum of the squares of the regularization weights. L1 typically is more robust and less efficient. L2 while less robust, will always be more efficient than L1. L1 and L2 are often referred to as least absolute deviations and least squares error respectively. The default setting for this method is L2. Normalization is done using the following equation:

$$L2\ Normalized\ Value\ of\ X = \frac{X}{\sqrt{\sum_{i=1}^{n} X_i^2}}$$

Normalizing data can help make the dataset less sensitive to the magnitude of the features and prevent bias. Also, data normalization is needed if a method such as Gaussian naïve Bayes is used, as these methods depend on data that is distributed

normally. Finally, normalization can help with the convergence of the machine learning algorithm. Figure 17 shows a comparison of normalized data and unscaled data. Figure 18 shows the output of the normalization method.

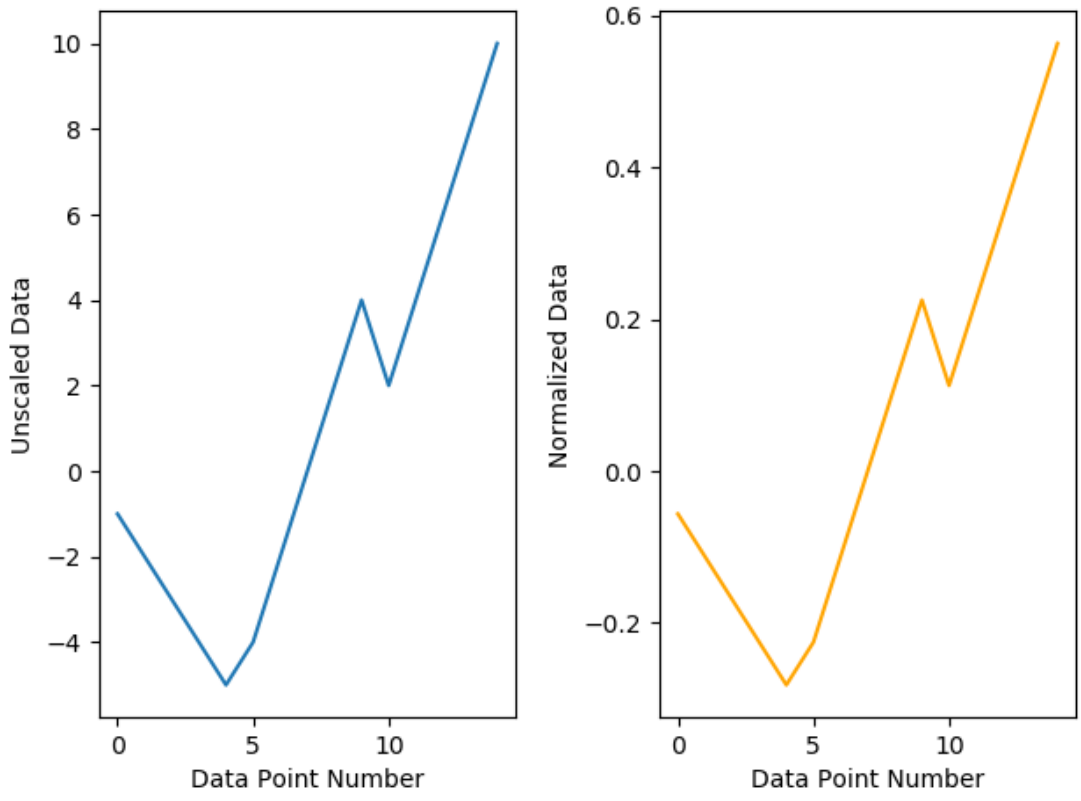


Figure 17: Comparison of Unscaled Data and Normalized Data

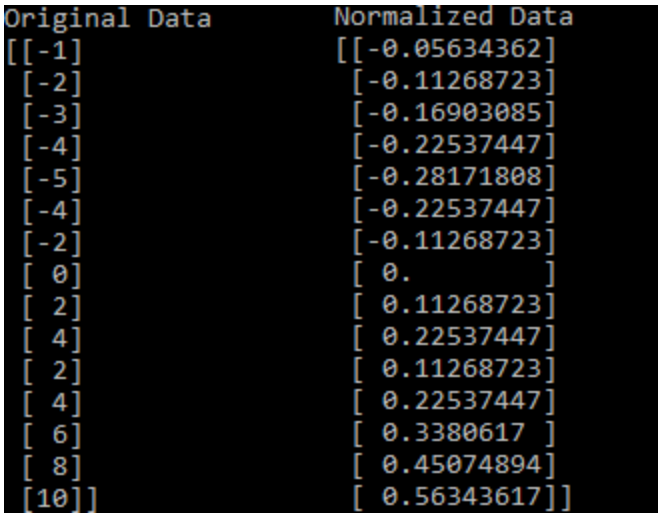


Figure 18: Example Normalized Preprocessed Data

Radial Basis Function Sampling

Radial Basis Function (RBF) sampling is a form of Random Fourier Features, more commonly referred to as Random Kitchen Sinks. This technique is intended to replace weight minimization with randomization in order to improve the classification. The RBF sampler maps a kernel using a Monte Carlo approximation. This is used in kernel-based machine learning algorithms, such as k-nearest neighbors and k-means, but is more widely used in neural networking and other support vector learning applications. RBF sampling is computationally less expensive than other kernel mapping techniques, such as the Nystroem method, but can be less accurate. Due to this, RBF Sampling is better used in cases where there are clearer differences between classes [26].

In scikit-learn, the RBF Sampling function is `.RBFSampler()`. The user can specify the parameters of the kernel, gamma, as well as the dimensionality of the components. As with other Monte Carlo approximations, the more components used the better the accuracy, but more computation time is needed. A random number seed may be

specified in order to replicate the results. Once the RBFSampler parameters are set, the data must be fit to perform the Monte Carlo analysis and then transformed to model the kernel map.

Feature Agglomeration

Feature Agglomeration is a form a Hierarchical clustering, sometimes referred to as agglomerative clustering. This process takes in the data and groups similar data into a predefined number of groups or clusters. This technique is used as the basis for some unsupervised learning algorithms, but it has applications in supervised methods as well. In supervised learning this method can be used to aid in feature reduction for complex datasets to help deal with models that are overfit [27].

The scikit-learn method for Feature Agglomeration, `.FeatureAllgomeration()`, merges features together in order to reduce the number features used in a machine learning algorithm. The user has the ability to enter several parameters to better tune the reductions to the needs of the analysis. The first of these is the number of clusters that the methods will create. The default for this method is 2, as this is typically used to determine if it is appropriate to merge 2 features together to reduce overfit. Another key parameter for this method is the ability to change the linkage criteria of the method. This is called affinity. The linkage criteria available are: ward, single, complete and average. The ward criterion looks to minimize the variance between the features being merged. The average criterion uses the average distance of the features for linkage. The single and complete criteria use the minimum and maximum distances respectively between the features for linkage. By default and for this project, the ward

criterion was used. Finally, the user can define the affinity, the type of distance, applied to the criterion of the method. This is the metric that is used for linkage calculations. The options for affinity include: euclidean, l1,l2, manhattan, cosine or precomputed. Since this project uses the ward criteria the euclidean metric must be used [28].

Principal Component Analysis

Principal Component Analysis (PCA) is a statistics-based technique that is designed to reduce the number of dimensions a dataset contains. PCA utilizes orthogonal transformations to combine correlated variables, using basic statistics and linear algebra techniques to identify patterns within data rather than using visualization. Once patterns are found within a dataset, it is then possible to reduce the number of features in that dataset based on the PCA results. This helps reduce model overfit and produces better machine learning models. PCA is also used in data compression applications as the original data can be recovered later if needed.

PCA works by first removing the mean of each of the datasets features by subtracting the mean value of each feature by the individual points that correspond to that feature. A covariance matrix is then created; the size of this square matrix is equal to the number of features present in the dataset. The eigenvectors and eigenvalues of the covariance matrix are then calculated, the length of this eigenvectors is equal to 1. The eigenvalues are then sorted from highest to lowest, which shows the significance of each component. Eigenvalues with small significance can be removed and while some information is lost, that information is small and has less impact on the model while reducing model over fit. Using the eigenvalues that were kept, a new feature matrix is

created. This matrix is then transposed and multiplied by the transpose of the mean adjusted data. This will give the new dataset with only the higher significant features left [29]. The user typically specifies the amount of variance that is acceptable to lose in the reduction. This technique is also useful for reducing statistical noise.

The scikit-learn function for PCA is `.PCA().` The user is able to specify the amount of explained variance acceptable to lose for the creation of the covariance matrix, as well as specify and empirical mean, if needed. The function will detect the number of features in the dataset. It should be noted that PCA requires enough memory to fit all of the data present in the dataset. This can be a problem for very large datasets. In these cases PCA can be performed in increments using the `IncrementalPCA` function [30].

Feature Selectors

As mentioned before, model overfit due to an abundance of features is a major consideration when creating a machine learning model. An overfit model may be able to predict accurately given the general case, but once small variations begin to occur, the model performance may suffer. In addition to using PCA, three different feature selection methods were implemented into the TPOT dictionary for this project to improve the model: Family Wise Error Rate (FWER), Select Percentile and Variance Threshold Selection. This attempts to remove features that are non-informative from the model.

Family Wise Error Feature Rate Selection

The FWER method is a univariate statistical approach used in hypothesis testing. FWER is the probability of at least 1 false positive, Type 1 error, in a group of hypothesis

test. It is calculated by taking the probability value (p-value) for a set of tests and rejecting hypothesis that fail a specified test. A common test for this rejection is the Bonferroni test [31]. This test rejects p-values based on the following expression:

$$Reject\ if\ p_i \geq \frac{\alpha}{h}$$

Where:

α is the specified criteria for the hypothesis test

h is the number of hypothesis tested.

The scikit-learn method for FWER is `.SelectFwe`. This method uses a statistical approach in order to calculate the p-value such as Chi-Square or F-value. The F-value is used by default. Also, an alpha may be specified by the use. The default alpha is 0.05. Figure 19 and 20 show an FWER example from the scikit-learn documentation [32].

This example loads the wine data from the scikit-learn repository. This set has 178 samples and 13 features. The FWER method used Chi-Squared to find p-values and a 0.01 alpha was selected to for the evaluation criteria. 5 features from the dataset failed the FWER test and were removed from the dataset.

```
from sklearn.datasets import load_wine
from sklearn.feature_selection import SelectFwe
from sklearn.feature_selection import chi2
wine = load_wine()
X, y = wine.data, wine.target
print('The Wine Dataset original size is:')
print(X.shape)
X_new = SelectFwe(chi2, alpha=0.01).fit_transform(X, y)
print('After applying the FWE method the wine dataset is now size:')
print(X_new.shape)
```

Figure 19 Example Code of Family Wise Error Rate Feature Selection

```
The Wine Dataset original size is:
(178, 13)
After applying the FWE method the wine dataset is now size:
(178, 8)
```

Figure 20 Output of Family Wise Error Rate Feature Selection Example

Select Percentile

The select percentile method, similar to FWER, is another univariate selection method that uses a statistical test to determine which features should be removed from the dataset. The key difference between the select percentile method and FWER is that instead of specifying an alpha for the p-value rejection criteria, the user inputs a percentile value and p-values are either rejected or accepted based on the scores when compared to that percentile.

The scikit-learn function for the select percentile technique is `.SelectPercentile`. Similar to the FWER function, the user selects the method for determining the p-values of the desired feature and the percentile for the p-value evaluation. This project uses the default F-value method for determining the p-values and a 10 percentile for the evaluation criteria.

Variance Threshold Selection

The variance threshold technique examines the features and does not factor the model outcomes, unlike univariate methods such as FWER. As such, variance threshold is an ideal feature selection method for unsupervised learning. Variance threshold selection calculates the variance of the individual features and removes those that do not meet the user specified requirements.

The scikit-learn function for variance threshold is `.VarianceThreshold()`. This function requires the user inputs a variance for the evaluation criteria. The default value of this is 0, which removes features that have the same value in all samples. This was the method used for this project. Figure 21 from the scikit-learn documentation shows a code example of the technique being applied.

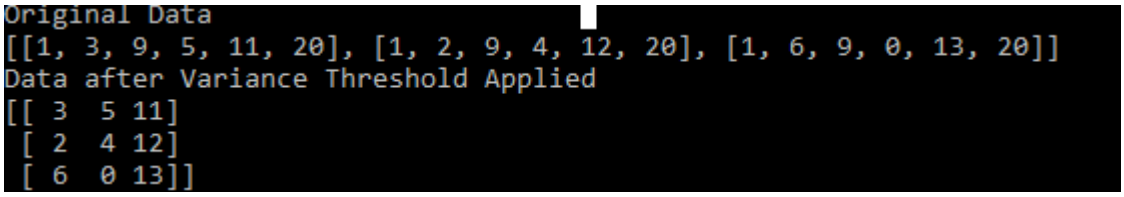


Figure 21: Example of Variance Threshold Feature Selection

This example creates a Python list with the 1st and 4th feature, 0 and 3, repeated in each row. The default variance of 0 is used for the feature selection. Once the method is applied, the repeated features have been removed from the dataset, leaving only the non-zero variance data in the set.

Machine Learning Classification Models

Naïve Bayes Classification

Naïve Bayes classification is a supervised machine learning algorithm that relies on probabilities. This method is based on Bayes' theorem and assumes statistical independence between two data points. Naïve Bayes classification is used in a variety of industries, i.e. the medical industry and spam email detection. Due to the probabilistic nature of naïve Bayes classification, there are several different models that can be created based on distributions. This project uses three different naïve Bayes models: Gaussian, Bernoulli and multinomial. Detailed explanations on each will be given below [33].

Bayes' Theorem

Bayes' theorem when applied for naïve Bayes classification is given by the following equation:

$$P(Y|X_1, \ldots., X_n) = \frac{P(Y) * P(\, X_1, \ldots., X_n|\, Y)}{P(X_1, \ldots., X_n)}$$

Where:

P(Y) represents the probability of event Y occurring

$P(X_1, \ldots., X_n)$ represents the probability of the X events occurring

$P(\, X_1, \ldots., X_n|\, Y)$ represents the probability of the X events occurring given Y

$P(Y|X_1, \ldots., X_n)$ represents the probability of event Y occurring given the X events

It is important to note that naïve Bayes always assumes statistical independence; as such the relationship between the event Y and the X events can be simplified to the following equation:

$$P(Y|X_1, \ldots., X_n) = \frac{(P(Y)\prod_{i=1}^{n} P(X_i|Y)}{P(X_1, \ldots., X_n|Y)}$$

Under these assumptions, the Maximum A Posteriori (MAP) technique can be used to determine P(Y) and $P(X_i|Y)$. In this case the MAP technique estimates P(Y) based on the mode of $P(X_i|Y)$. The key difference between the several naïve Bayes techniques is how $P(X_i|Y)$ is calculated.

Naïve Bayes Advantages & Disadvantages

Naïve Bayes classification models have several advantages over other supervised classification techniques. The first of these is that the time required to calculate the model is less than that of other techniques, such as k-nearest neighbors. Also, naïve Bayes techniques can be performed using less test data than other techniques due to the conditional independence assumption of the technique. Finally, again due to the conditional independence assumption, the technique is less likely to suffer from overfitting due to a high number of features in the dataset. This is because feature distributions are decoupled, allowing each feature distribution to be estimated as a single distribution. There are drawbacks to this technique though. Statistical independence is not common in the real world. As such, this technique is poor at creating estimates and is not a useful tool in regression analysis. Still, real-world applications have proven it to be

an effective classification tool as the dependencies between features tend to cancel out in the classification process [34].

Gaussian Naïve Bayes

The first naïve Bayes method used for this project was the Gaussian classification technique. As the name suggests, this method assumes that the probabilities of features are Gaussian, or a standard normal distribution. Gaussian distributions are symmetrical and only have a single peak. The Gaussian naïve Bayes technique uses the following equation to calculate the probability of the set events X_i given event Y:

$$P(X_i|Y) = \left(\frac{1}{\sqrt{2\sigma_Y^2}}\right) e^{-\frac{X_i - \mu_Y}{2\sigma_y^2}}$$

Where:

μ_y is the mean of event Y

σ_y is the standard deviation of event Y

Bernoulli Naïve Bayes

The Bernoulli naïve Bayes method assumes the data follows a multivariate Bernoulli distribution. A Bernoulli distribution assumes that the values are boolean, either 0 or 1. Due to this, the values entered into the model must be converted into this format. The probability of the set of events X_i given event Y, is found using the following equation:

$$P(X_i|Y) = P(i|y)x_i + (1 - P(i|y)(1 - x_i)$$

It should be noted that this method penalizes the score of the model if the feature i does not occur for a given Y.

Multinomial Naïve Bayes

The final naïve Bayes method used for this project is the multinomial classification technique. Multinomial distribution is a generalized form of the binomial distribution. Unlike a binomial distribution, a multinomial distribution can have values other than 0 or 1. This is typically used to determine the probability of a series of mutually exclusive events occurring at a given time. A key difference between the multinomial and Bernoulli methods is that the multinomial technique does not penalize the score of the model if a feature does not occur within a given data point. Probability of a set of events X_i given event Y is calculated using the following equation:

$$P(X_i|Y) = \theta_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

Where:

N_{yi} is the number of times a features appears in the training set

N_y is the number of times a features occurs in both the testing and training sets

α is a smoothing factor to prevent the nonoccurrence of a feature from penalizing the model

Naïve Bayes Classification Example

Unlike the k-nearest neighbor, naïve Bayes classification is difficult to visualize as it makes use of probabilities, rather than distance. As such, this example is provided in

order to demonstrate how a naïve Bayes classification works in real-world applications. Perhaps, the most common application of naïve Bayes classification is for spam/junk email detection. In this type of classification there are only two possible outcomes: the email is either spam or it not spam. As naïve Bayes is a supervised method, the first step is to have a training set of email data that is already classified as either spam or not-spam.

The key with naïve Bayes classification is that the outcome probabilities must be calculated using the dataset features. The features of this type of dataset would be the words inside the email. Using the training set data, the probability that if an email contains a certain word that the email is spam is calculated. Since there are only two outcomes, the probability that given a certain word that the email is not spam is simply 1 minus the probability that given the word an email is spam. Typically, common words such as 'it' will be assigned neutral probabilities, while other keywords are assigned higher probabilities. Next, the probability of a general email being spam or not being spam is determined. This can be done using either the training dataset, to better tailor the model to a specific email account, or the probability can be assigned using outside data and assumptions [35].

Once all of the probabilities have been determined, the probabilities of an email being spam given a certain word, can be calculated using Bayes' theorem. This process is repeated for every word in the email. Once probabilities have been done for every word in the email, the probabilities are then combined so the probability of the email being spam given the set of words can be determined. This is done using the following equation:

$$P(S|\boldsymbol{W}) = \frac{P_1 * P_2 * \dots P_N}{P_1 * P_2 * \dots P_N + (1 - P_1)(1 - P_2)\dots(1 - P_N)}$$

Where:

$P(S|\boldsymbol{W})$ is the probability of the email being spam given the set or words

P_N are the probabilities of the email being spam given an individual word

Using this total probability, the system determines whether to classify the email as spam or not-spam, based on user preferences.

K-Nearest Neighbors

K-nearest neighbors is a supervised machine learning algorithm used in both, classification and regression models. This method is non-parametric, meaning that the algorithm does not rely on a set number of parameters and can be flexible depending on the situation. K-nearest neighbors works by using a user-defined constant integer known as k. K is the number of nearest neighbors that the algorithm looks for in the classification. A majority vote of the nearest neighbors is used to determine which class a data point belongs to. This means that the testing data point will be classified according to which training data it is closest to, in the feature space. The size of k must be balanced when using this method. If k is too small, the model may be overfit. If k is too large, the possibility of an under-fit model that leaves out important details increases [36].

The k-nearest neighbors algorithm also requires a distance function in order to calculate the distance between a given testing data point and the different training data points. This algorithm typically uses Euclidean distance, also known as the straight line

distance between two points in Euclidean space. This is very similar to the distance formula used in basic algebra and geometry. The Euclidean distance is found using the following equation:

$$D(x_{i,}x_{j,}) = \sqrt{\sum_{m=1}^{D}(x_{im} - x_{jm})^2} = \sqrt{||x_i||^2 + ||x_j||^2 - 2x_i^T x_j}$$

$D(x_{i,}x_{j,})$ is the distance between the 2 points

$||x_i||^2$ and $||x_j||^2$ are norm of the respective points

$2x_i^T x_j$ is the dot project between the two points.

The k-nearest neighbors algorithm works best when the data points are scaled to balance the magnitude of different features and are normalized. K-nearest neighbors is simple in nature and easy to visualize, especially in datasets with fewer features. Also, with more data and a large k, the algorithm can produce very accurate results. Unfortunately, there are disadvantages associated with using k-nearest neighbors. The first is that it can be computationally expensive, especially with larger datasets. Also, it can be sensitive to statistical noise from features. Finally, the model can suffer if too many features are used [37].

Logistic Regression

Logistic regression is a supervised machine learning algorithm designed to deal with complex scenarios. While its name suggests this method is a regression method, it is actually a classification method. Similar to the naïve Bayes method, logistic regression

uses probabilities to predict to what class a set of features belongs to, i.e. the probability of Y given X, or a set X_i. Outputs can be boolean, multinomial or for cases where more than one class exists, One vs. Rest. An important difference between the two methods is that naïve Bayes assumes the features are statistically independent, while logistic regression does not. This results in the naïve Bayes model having more bias but less variance when compared to a logistic regression model. The decision of which model is best depends on the data used in creating the two models. Typically, logistic regression is preferred when the data has a large number of features, while naïve Bayes works better with less complex data. Logistic regression takes the different outcome probabilities of a given data point and models them using a logistic function [38].

This project involves 5 different categories for classification, as such; the One vs. Rest method of logistic regression will be used to create the classification model. One vs. Rest logistic regression uses the following equation:

$$P(Y_k|X) = \frac{1}{1 + \sum_{j=1}^{k=1} \exp(w_{jO} + \sum_{i=1}^{n} w_{ji} X_i)}$$

Where:

$P(Y_k|X)$ is the probability of Y belonging to class k given X

w_{jO} and w_{ji} are the weights associated with class j

Scikit-learn has the ability to otimize this equation by modifying the weights with user specified input [39]. This can be done using three methods: L1, L2 or Elastic Net

regularization. L1 solves issues with weight optimization given by the following equation:

$$L1 = \ min_{wc} |w| + C \sum_{i=1}^{n} \log(\exp\left(-y_i(X_i^T w + c)\right) + 1$$

L2 minimizes the cost function of the weights using the following equation:

$$L2 = \ min_{wc} \frac{1}{2} \ w^T w + C \sum_{i=1}^{n} \log(\exp\left(-y_i(X_i^T w + c)\right) + 1$$

and Elastic Net regularization is used when there are issues with both cost and optimization using the following equation:

$$min_{wc} \frac{1-p}{2} \ w^T w + p \ |w| + C \sum_{i=1}^{n} \log(\exp\left(-y_i(X_i^T w + c)\right) + 1$$

Decision Tree Classification

Decision tree analysis is a non-parametric, supervised learning method that can be used in both classification and predictive regression. Decision tree analysis, unlike other methods, such as k-nearest neighbors and naïve Bayes, is designed to deal with statistical noise that can deter a model's performance. Common applications for decision tree analysis include: credit and loan assessment, medical diagnostics and performance evaluation/prediction.

The top of a decision tree is the outcome of the analysis. Decision tree analysis begins by taking instances from the data provided. These are typically the different classes of the data provided to the model. Next, a data point is tested, typically using a

true or false evaluation, though it is possible to use non-boolean responses for testing, if appropriate. Depending on the outcome, further testing may occur or the model may have enough information to make a determination and classify a data point. If more testing is needed, the model will continue to evaluate the data point provided until a determination can be made or there are no more evaluation metrics to test. A good analogy for a decision tree classification would be a personality test, such as the Myers-Briggs indicator. In this survey, the respondent is asked several questions and at the end of the test, the user is put in a class based on the responses to the questions. In a decision tree classification, the features provided in a data point are the responses to the questions at each root of the tree [40].

Decision trees have several advantages over other machine learning methods. Unlike logistic regression or naïve Bayes methods, decision trees are easy to visualize and conceptually simple. Decision trees can work with both, qualitative and quantitative data, and data does not need to be of the same type. Also, decision trees are better able to deal with missing values and a classification can still be done if some features are missing. This reduces the amount of data preparation that needs to be done in order to perform the analysis. Finally, decision trees are able to handle multiple output models, which allow the method to apply more complex problems. Still, there are some disadvantages associated with decision trees. The order of the evaluations in the tree is extremely important. Some orders may filter out critical data and lead to inaccurate results and misclassifications. This can be managed by creating several different trees to determine which order best fits the model. This can be computationally expensive and time consuming. A better method of dealing with this would be to calculate an

evaluation's entropy. This will be explained below. Another issue with decision trees is that models can become overly complex and overfit, if too many features are present. Data preprocessing and selection methods, such as PCA can be used to help manage this issue.

The order in which each evaluation is performed is a key to the effectiveness of this method. In order to determine which evaluations provide the most insight or information gain, it is necessary to calculate the entropy of the evaluation. In this context, entropy is a measure of the purity/impurity of the data samples. This can be calculated using the following equation:

$$S = -\sum p \log_2 p$$

Where:

S is the Entropy of the collection of data

P is the mass probability function of the evaluation

If the responses to the evaluation are boolean the equation can be expressed as the combination of the negative responses and the positive responses with the following equation:

$$S = -p \log_2 p - p \log_2 p$$

Evaluations with higher entropy are considered more insightful and are prioritized earlier in the tree, while those with lower entropy values are placed lower in the tree. This allows

the model to determine the sequence that yields the most gain [40]. This approach is known as a ‘greedy’ algorithm, where the algorithm searches for the optimal result rather than the best.

Decision trees and other machine learning models can be improved using Ensemble methods. These methods improve models, either through averaging or boosting. One of the averaging techniques for decision trees is known as random forest. This method constructs several different decision trees rather than just a single tree. The results from each of the different trees are averaged and a final result is determined. In classification, the result is determined by which result received the most votes from the individual tress in the forest. Regression uses the average of output to determine the final result. The use of random forest can reduce overfitting and improve accuracy. Another ensemble method that can be used to improve a decision tree is gradient boosting. Typically used in regression, gradient boosting uses weak learners and the errors from those learners to compute a residual. The model is then trained on the residual and the model then tries to predict those residuals. It should be noted that boosting is a greedy algorithm and can produced overfit results [42].

Model Validation

Once a machine learning model has been trained and tested, it is necessary to validate the model to determine how well the model performed. Accuracy is the most common validation measurement used to assess a model’s performance. The accuracy of the model is simply the number of correct classifications divided by the total number of

test classifications performed [43]. Scikit-learn uses the `.accuracy_score()` method to measure a model's accuracy using the following expression:

$$accuracy(y, y_{bar}) = \frac{1}{n_{samples}} * \left(\sum_{i=1}^{n_{samples-1}} 1(y_{bar} = y_i) \right)$$

In addition to accuracy, other validation measurements are necessary. Relying only on accuracy measurements would only tell the user that a misclassification had occurred. Using other validation measurements allows for further exploration of a model's mistakes. Measuring a model's precision allows the user to see the ratio of true positives to predicted positives. This shows how many false positive, or type 1 errors, occurred in the model's testing. Scikit-learn uses the `.precision_score()` method to calculate a model's precision using the following equation:

$$Precison = \frac{\#\ of\ True\ Positives}{\#\ of\ False\ Positive + \#\ of\ True\ Positives}$$

The next validation measurement performed for this project is recall. Recall is a measurement of a model's ability to classify positive samples. This is referred to as sensitivity. Recall allows the user to evaluate a model's false negative error, or Type II error. Scikit-learn uses the `.recall_score` method to calculate a model's recall using the following equation:

$$Recall = \frac{\#\ of\ True\ Positives}{\#\ of\ False\ Negatives\ + \#\ of\ True\ Positives}$$

In classifying reactor transients, false positive errors potentially result in action that is costly and time consuming. False negatives can result in potential reactor damage and events hazardous to the public. Both of these must be considered when evaluating models. As such, it is necessary to have a validation measurement that balances type 1 and type II error. This measurement is known as the F1 score. The F1 score measures a weighted average between precision and recall. Scikit-learn uses the `.f1_score()` function to calculate a model's F1 score using the following equation:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

When dealing with percentages it is important to consider the context of the situation. In this case, since over 15,000 samples are being looked at, a 1% change in accuracy or precision would affect 150 samples. As such, it is necessary to use a method that tells the exact number of Type 1 and Type II errors that occurred and between which transients these occurred. To do this a confusion matrix can be created. A confusion matrix shows the true positives for each classification down the diagonal of the matrix. The false positives are shown in the columns and the false negatives are in the rows. This allows not only for the exact number and type of errors to be shown, but also where they occurred. This can provide insight into where and why a classification model is having issues. Confusion matrices will be generated for all six machine learning models used in this project.

GPWR Reactor Simulator

Data used for this experiment was collected using the Generic Pressurized Water Reactor (GPWR) simulator at the Center for Advance Energy Studies (CAES). The simulator was purchased by University of Idaho, Idaho State University, CAES entities from the Western Service Corporation (WSC). This simulator emulates the behavior of a "generic" pressurized water reactor (PWR). The thermal output is rated at 4000 MWt/1400 MWe. The simulator does not directly incorporate the design of any specific PWR. The reactor systems include 1 high-pressure turbine and 3 low-pressure turbines, and a configuration that includes 2 loops, 4 coolant pumps and 2 steam generators.[44] The simulator software runs on a standard Windows 10 PC, Figure 22 shows the simulator at the visualization lab at CAES.

Figure 22: GPWR Simulator Setup at CAES

The simulator comes preloaded with 14 different initial conditions for the reactor; factors that vary include plant power, core life, etc. Table 3 shows the list of the preprogrammed initial conditions. The simulator has several reactor control panels that allow the user to change the reactor system, including the operation of pumps, the opening/closing of valves, the flow of coolant, reactor power, etc. Figure 23 shows an example of the simulator interface that is used to control the reactor. The simulator is also equipped with an alarm system to inform the user of abnormal conditions. Similar to an actual reactor, the simulator can be scrammed by the operator and automatically scrams under certain conditions, including both a reactor trip and a turbine trip. The simulator is also programmed to emulate the containment structure and engineering safety features and the behavior of these components under accident conditions, such as a small break loss of coolant accident (LOCA). The simulator has also been programmed with the ability to emulate malfunctions of components that could potentially occur within the power plant. The reactor will behave accordingly in a malfunction. Some of these malfunctions include: heat exchanger degradation, motor shearing/seizures, valve failure to open/close, etc. These events can be triggered by the user or programmed as part of an accident scenario.

TABLE 3: GPWR Preprogramed Initial Conditions				
BOL, 100% Power	MOL, 100% Power	EOL, 100% Power	BOL, 50% Power	MOL, 50% Power
EOL, 50% Power	BOL, 1% Power	MOL, 1% Power	EOL, 1% Power	BOL, Subcritical
MOL, Subcritical	EOL, Subcritical	BOL, Xe Equilibrium	MOL, 5% Power	

The GPWR displays critical parameters to the user. Many of these are those that an operator would see while running an actual reactor. There are 18 different parameters that are always displayed to the user, including reactor power (in both MW and a percentage), steam generator pressure and flow and turbine/reactor status. Figure 24 shows how this output is displayed to the user. The user is also able to see how long the simulator has been running, as well as pause and restart simulations. The simulation can be run in real-time, slowed down to 0.1 times normal speed or speed up to 10 times normal speed. This can be changed at any time during the simulation. In addition, the user has the ability to backtrack to a previous time in the simulation. The software will automatically save conditions every few minutes in order for the user to easily return to a previous state.

The GPWR simulator also allows for the user to switch between different interfaces in order to observe, manipulate and record the behavior of components that are not shown on the reactor interface page. Figure 25 shows the overview home page for the GPWR.

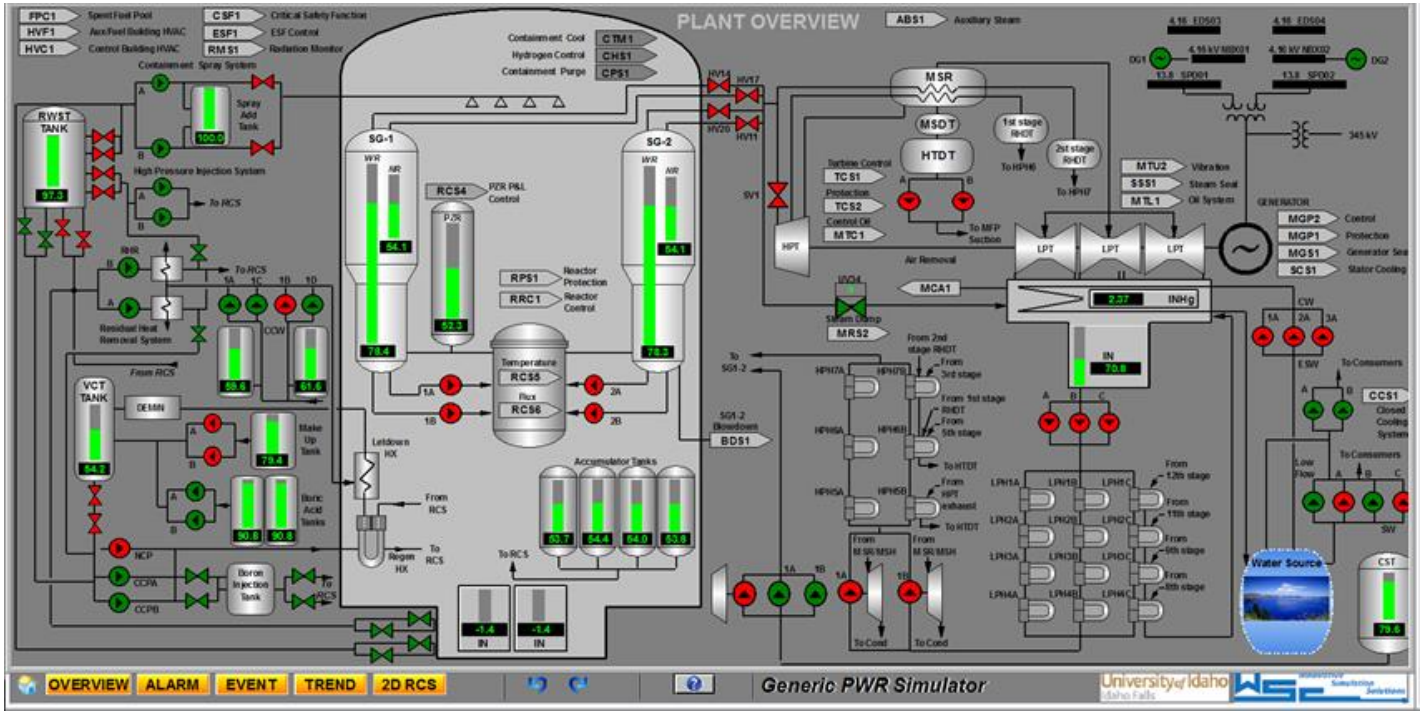


Figure 23: GPWR Simulator Interface

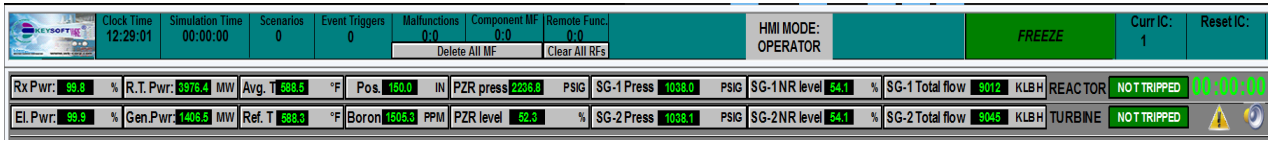


Figure 24 Information Panel Displayed in GPWR

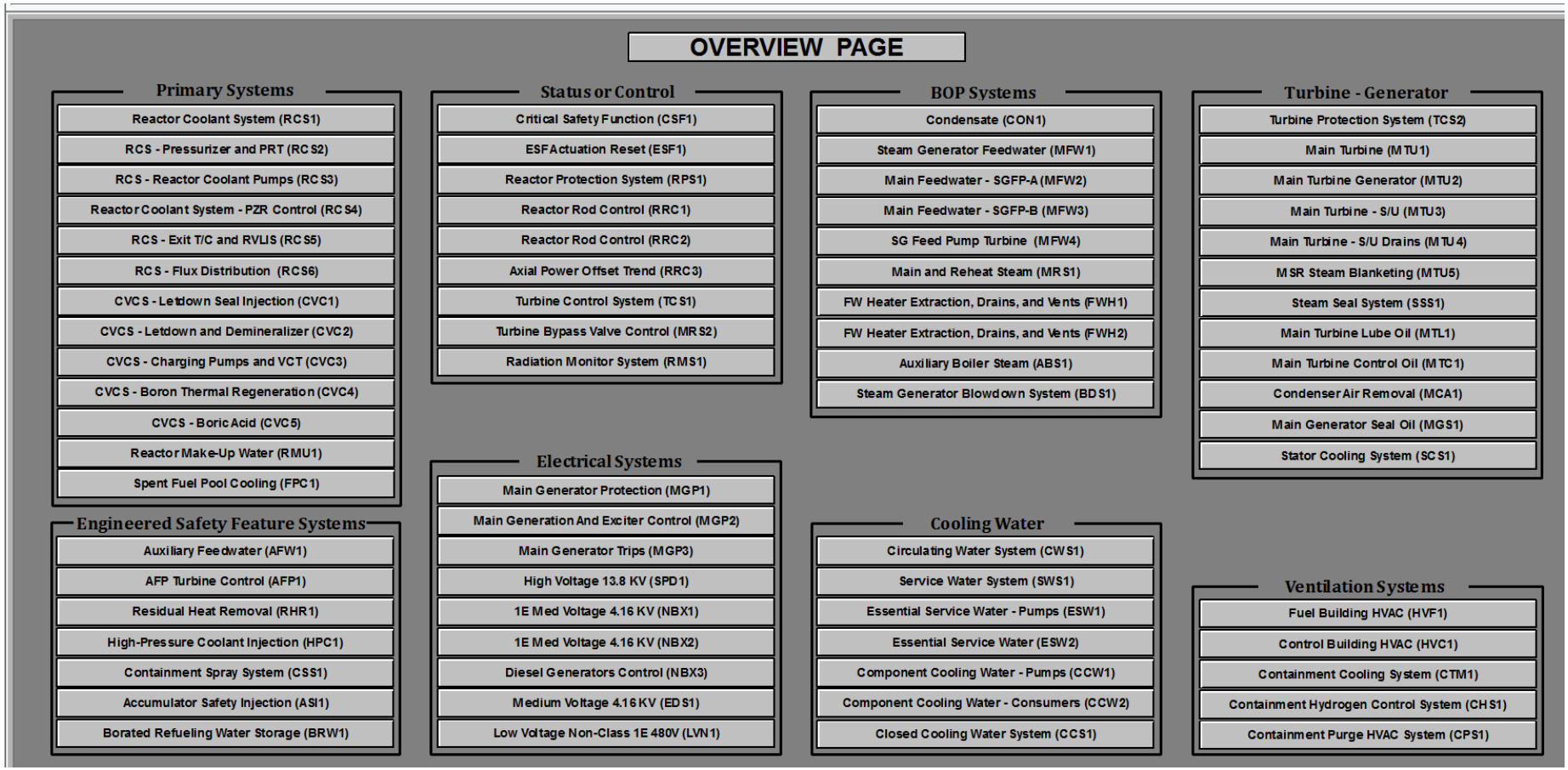


Figure 25: GPWR Navigation Home Table

Data Gathering

Feature Selection

In order to construct machine learning models it was necessary to decide on the features that would be measured using the GPWR reactor simulator. The simulator is able to measure and collect data from several reactor components, such as reactor power output and steam generator pressure. It was decided that for this model the data gathered would consist of data that a reactor operator would have access to and be readily available. Thirty three features were chosen and programmed into the simulator data collector, including: reactor power output, steam generator temperature, flow and pressure, as well as reactor temperature. The complete list of features gathered is shown in Table 4. All of the features collected from the reactor simulator were quantitative in nature.

TABLE 4: Features Collected from GPWR Simulator					
Normalized Flux	RCS LVL Loop 1 WR	RCS LVL Loop 1 NR	Hot Leg 1/ 2 Temperature	Cold Leg 1/ 2A Temperature	Cold Leg 1/2B Temperature
RC Loop-1/2A Norm Flow	RC Loop -1/2B Norm Flow	Pressurizer Surge Line Temperature	PORV Discharge Pressurizer Temperature	Containment Pressure	Containment Temperature
MS Flow from SG-1 Line-1/2B	SG-1/2 Pressure	Average Temperature	SG-1/2 NR Level	FW Flow to SG-1/2	Pressurizer Pressure
Pressurizer Steam Temperature	Norm Pressurizer Level	Pressurizer Water Temperature	Generator Power	MS Flow from SG-1 Line-1/2A	

Initial Conditions

In order to see how the model would be impacted by changes in the reactor system over time, it was decided that several runs would be conducted changing the initial conditions of the simulated system for each run. The first change to the system was the power output of the reactor. Three different conditions were used: full power, where the reactor is operating as to generate electricity; half power, where the reactor is being shut down and output is at approximately 50% of capacity; and low power where the reactor is critical and being prepared for startup, but power generation is between 0 and 1% capacity. The second initial condition changed for the reactor system involves the stage of the reactor's lifetime. Three different conditions were available for use: beginning of life (BOL) where the reactor is brand new; middle of life (MOL), where the reactor is close to 30 years old; and end of life (EOL), where the reactor is close to decommissioning approximately 60 years into its operating life. Using these two features it was possible to collect data on nine different initial condition combinations while the reactor is functioning as intended. Each run was conducted for 1200 seconds and data was collected for each of the 33 measurable features every second during the run. Seconds are the smallest increment of time that can be used for data collection.

Transient Events

In addition to collecting data when the reactor is under normal operating conditions, four transient events were simulated using the nine different initial condition configurations. The first transient event selected was a simultaneous trip of all feed water pumps. In this transient, the primary and auxiliary feed water pumps malfunction and

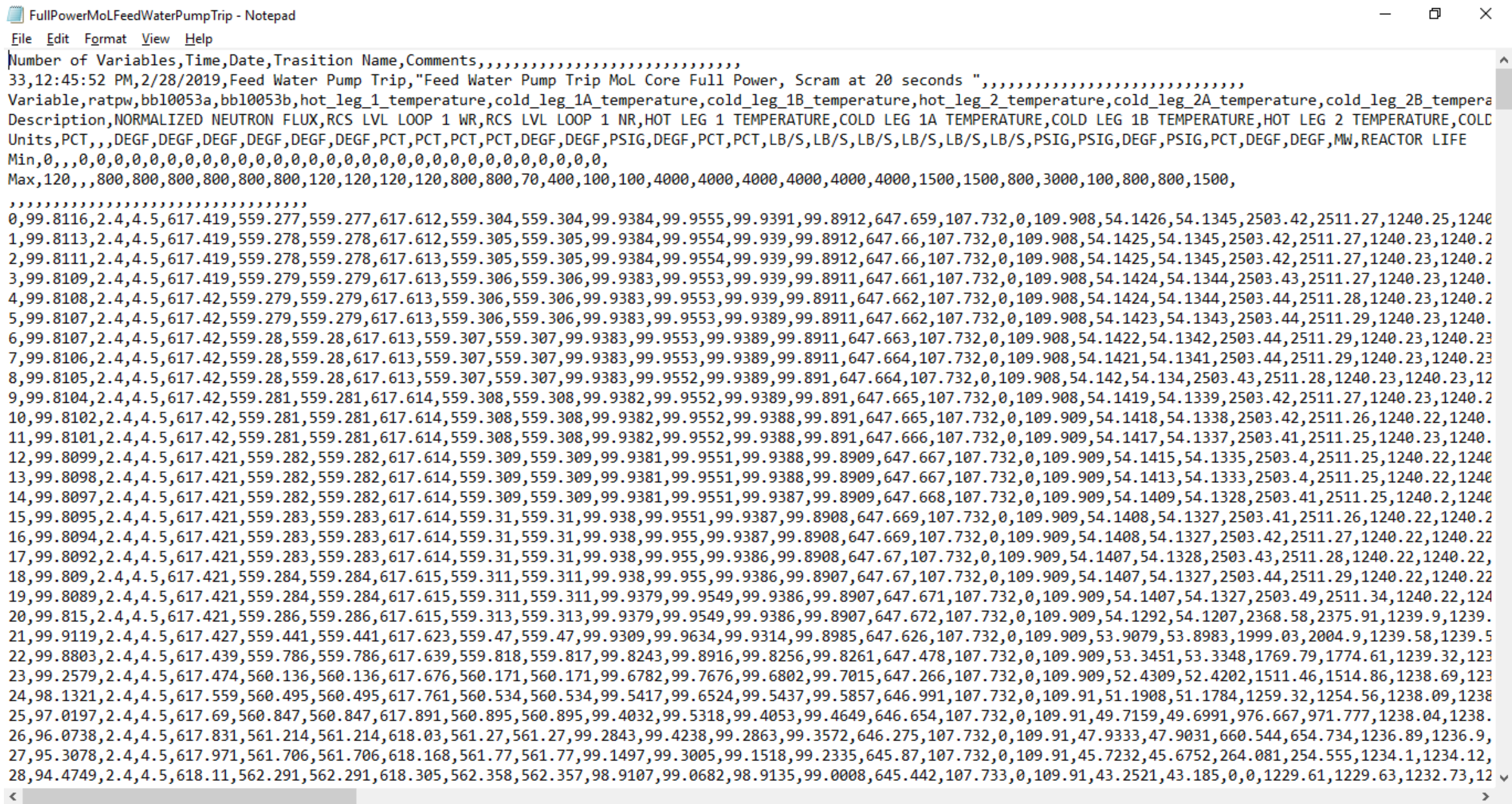
cease operations. The breakers connected to these two pumps also trip. The transient was programmed to occur 20 seconds after the simulation began. A run under each of the nine different initial condition configurations was performed and data was collected for 600 seconds after the transient occurred. Under this transient, the runs that were performed at full power and half power scrammed the second the transient occurred. During the run that occurred at low power, no scram occurred during the simulation [45].

The next transient event that was used to collect data was a simultaneous closure of main steam isolation valves (MSIV). In this transient, a command signal is sent to all MSIVs after 20 seconds switching the valves from the open position to the closed position. Data was collected for 600 seconds after the command signal was sent. Each of the 9 different initial condition configurations was used to collect data on this transient. In this event, runs performed under full and half power experienced a scram 40 second after the simulation began, 20 seconds after the command signal was sent. Runs performed at low power did not scram during the simulation.

The third transient event used in this experiment was a maximum reactor coolant rupture combined with a complete loss of offsite power (LOOP). During this transient, a double ended guillotine break occurs within line 1A of the reactor coolant system (RCS). This is combined with a complete loss of electrical power to the plant. The transient occurred 20 seconds after the simulation began and data was collected for 600 seconds after the transient occurred. Nine separate runs were performed using the initial condition configurations. During this simulation, the reactor experienced a scram at all power levels used.

The final transient event used to collect data was a rapid power change. In this transient, the reactor begins 1400 MWe full power and drops to 1050 MWe, approximately 75% of the plant's maximum power, before returning to 1400 MWe. Data was collected until the reactor reached full power, approximately 1000 seconds. Due to the nature of this transient, only the reactor core life initial conditions were changed and three runs were performed.

After the completion of a run, the data was saved from the reactor simulator to a comma separated values (CSV) file. In total, 39 different CSV files were generated and saved. All data gathered directly from the simulator was quantitative. After each run, the reactor core lifetime feature was added to each instant from the dataset from the run using either 'BOL', 'MOL' or 'EOL' or. Also, the transient that occurred was added to each instant in the dataset. It should be noted, that the instances up to the 20 second mark were labeled as normal operations, as the transient had not yet occurred. These additions were done using Microsoft Excel. The datasets remained in a CSV format. Figure 26 shows a screenshot of one of the CSV files collected from the simulator.



```
FullPowerMoLFeedWaterPumpTrip - Notepad
File Edit Format View Help
Number of Variables,Time,Date,Trasition Name,Comments,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
33,12:45:52 PM,2/28/2019,Feed Water Pump Trip,"Feed Water Pump Trip MoL Core Full Power, Scram at 20 seconds ",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
Variable,ratpw,bbl0053a,bbl0053b,hot_leg_1_temperature,cold_leg_1A_temperature,cold_leg_1B_temperature,hot_leg_2_temperature,cold_leg_2A_temperature,cold_leg_2B_tempera
Description,NORMALIZED NEUTRON FLUX,RCS LVL LOOP 1 WR,RCS LVL LOOP 1 NR,HOT LEG 1 TEMPERATURE,COLD LEG 1A TEMPERATURE,COLD LEG 1B TEMPERATURE,HOT LEG 2 TEMPERATURE,COLD
Units,PCT,,,DEGF,DEGF,DEGF,DEGF,DEGF,DEGF,PCT,PCT,PCT,PCT,DEGF,DEGF,PSIG,DEGF,PCT,PCT,LB/S,LB/S,LB/S,LB/S,LB/S,LB/S,PSIG,PSIG,DEGF,PSIG,PCT,DEGF,DEGF,MW,REACTOR LIFE
Min,0,,,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
Max,120,,,800,800,800,800,800,800,120,120,120,120,800,800,70,400,100,100,4000,4000,4000,4000,4000,4000,1500,1500,800,3000,100,800,800,1500,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
0,99.8116,2.4,4.5,617.419,559.277,559.277,617.612,559.304,559.304,99.9384,99.9555,99.9391,99.8912,647.659,107.732,0,109.908,54.1426,54.1345,2503.42,2511.27,1240.25,1240
1,99.8113,2.4,4.5,617.419,559.278,559.278,617.612,559.305,559.305,99.9384,99.9554,99.939,99.8912,647.66,107.732,0,109.908,54.1425,54.1345,2503.42,2511.27,1240.23,1240.2
2,99.8111,2.4,4.5,617.419,559.278,559.278,617.613,559.305,559.305,99.9384,99.9554,99.939,99.8912,647.66,107.732,0,109.908,54.1425,54.1345,2503.42,2511.27,1240.23,1240.2
3,99.8109,2.4,4.5,617.419,559.279,559.279,617.613,559.306,559.306,99.9383,99.9553,99.939,99.8911,647.661,107.732,0,109.908,54.1424,54.1344,2503.43,2511.27,1240.23,1240.
4,99.8108,2.4,4.5,617.42,559.279,559.279,617.613,559.306,559.306,99.9383,99.9553,99.939,99.8911,647.662,107.732,0,109.908,54.1424,54.1344,2503.44,2511.28,1240.23,1240.2
5,99.8107,2.4,4.5,617.42,559.279,559.279,617.613,559.306,559.306,99.9383,99.9553,99.9389,99.8911,647.662,107.732,0,109.908,54.1423,54.1343,2503.44,2511.29,1240.23,1240.
6,99.8107,2.4,4.5,617.42,559.28,559.28,617.613,559.307,559.307,99.9383,99.9553,99.9389,99.8911,647.663,107.732,0,109.908,54.1422,54.1342,2503.44,2511.29,1240.23,1240.23
7,99.8106,2.4,4.5,617.42,559.28,559.28,617.613,559.307,559.307,99.9383,99.9553,99.9389,99.8911,647.664,107.732,0,109.908,54.1421,54.1341,2503.44,2511.29,1240.23,1240.23
8,99.8105,2.4,4.5,617.42,559.28,559.28,617.613,559.307,559.307,99.9383,99.9552,99.9389,99.891,647.664,107.732,0,109.908,54.142,54.134,2503.43,2511.28,1240.23,1240.23,12
9,99.8104,2.4,4.5,617.42,559.281,559.281,617.614,559.308,559.308,99.9382,99.9552,99.9389,99.891,647.665,107.732,0,109.908,54.1419,54.1339,2503.42,2511.27,1240.23,1240.2
10,99.8102,2.4,4.5,617.42,559.281,559.281,617.614,559.308,559.308,99.9382,99.9552,99.9388,99.891,647.665,107.732,0,109.909,54.1418,54.1338,2503.42,2511.26,1240.22,1240.
11,99.8101,2.4,4.5,617.42,559.281,559.281,617.614,559.308,559.308,99.9382,99.9552,99.9388,99.891,647.666,107.732,0,109.909,54.1417,54.1337,2503.41,2511.25,1240.23,1240.
12,99.8099,2.4,4.5,617.421,559.282,559.282,617.614,559.309,559.309,99.9381,99.9551,99.9388,99.8909,647.667,107.732,0,109.909,54.1415,54.1335,2503.4,2511.25,1240.22,1240
13,99.8098,2.4,4.5,617.421,559.282,559.282,617.614,559.309,559.309,99.9381,99.9551,99.9388,99.8909,647.667,107.732,0,109.909,54.1413,54.1333,2503.4,2511.25,1240.22,1240
14,99.8097,2.4,4.5,617.421,559.282,559.282,617.614,559.309,559.309,99.9381,99.9551,99.9387,99.8909,647.668,107.732,0,109.909,54.1409,54.1328,2503.41,2511.25,1240.2,1240
15,99.8095,2.4,4.5,617.421,559.283,559.283,617.614,559.31,559.31,99.938,99.9551,99.9387,99.8908,647.669,107.732,0,109.909,54.1408,54.1327,2503.41,2511.26,1240.22,1240.2
16,99.8094,2.4,4.5,617.421,559.283,559.283,617.614,559.31,559.31,99.938,99.955,99.9387,99.8908,647.669,107.732,0,109.909,54.1408,54.1327,2503.42,2511.27,1240.22,1240.22
17,99.8092,2.4,4.5,617.421,559.283,559.283,617.614,559.31,559.31,99.938,99.955,99.9386,99.8908,647.67,107.732,0,109.909,54.1407,54.1328,2503.43,2511.28,1240.22,1240.22,
18,99.809,2.4,4.5,617.421,559.284,559.284,617.615,559.311,559.311,99.938,99.955,99.9386,99.8907,647.67,107.732,0,109.909,54.1407,54.1327,2503.44,2511.29,1240.22,1240.22
19,99.8089,2.4,4.5,617.421,559.284,559.284,617.615,559.311,559.311,99.9379,99.9549,99.9386,99.8907,647.671,107.732,0,109.909,54.1407,54.1327,2503.49,2511.34,1240.22,124
20,99.815,2.4,4.5,617.421,559.286,559.286,617.615,559.313,559.313,99.9379,99.9549,99.9386,99.8907,647.672,107.732,0,109.909,54.1292,54.1207,2368.58,2375.91,1239.9,1239.
21,99.9119,2.4,4.5,617.427,559.441,559.441,617.623,559.47,559.47,99.9309,99.9634,99.9314,99.8985,647.626,107.732,0,109.909,53.9079,53.8983,1999.03,2004.9,1239.58,1239.5
22,99.8803,2.4,4.5,617.439,559.786,559.786,617.639,559.818,559.817,99.8243,99.8916,99.8256,99.8261,647.478,107.732,0,109.909,53.3451,53.3348,1769.79,1774.61,1239.32,123
23,99.2579,2.4,4.5,617.474,560.136,560.136,617.676,560.171,560.171,99.6782,99.7676,99.6802,99.7015,647.266,107.732,0,109.909,52.4309,52.4202,1511.46,1514.86,1238.69,123
24,98.1321,2.4,4.5,617.559,560.495,560.495,617.761,560.534,560.534,99.5417,99.6524,99.5437,99.5857,646.991,107.732,0,109.91,51.1908,51.1784,1259.32,1254.56,1238.09,1238
25,97.0197,2.4,4.5,617.69,560.847,560.847,617.891,560.895,560.895,99.4032,99.5318,99.4053,99.4649,646.654,107.732,0,109.91,49.7159,49.6991,976.667,971.777,1238.04,1238.
26,96.0738,2.4,4.5,617.831,561.214,561.214,618.03,561.27,561.27,99.2843,99.4238,99.2863,99.3572,646.275,107.732,0,109.91,47.9333,47.9031,660.544,654.734,1236.89,1236.9,
27,95.3078,2.4,4.5,617.971,561.706,561.706,618.168,561.77,561.77,99.1497,99.3005,99.1518,99.2335,645.87,107.732,0,109.91,45.7232,45.6752,264.081,254.555,1234.1,1234.12,
28,94.4749,2.4,4.5,618.11,562.291,562.291,618.305,562.358,562.357,98.9107,99.0682,98.9135,99.0008,645.442,107.733,0,109.91,43.2521,43.185,0,0,1229.61,1229.63,1232.73,12
```

Figure 26: Screenshot of a CVS file from the GPWR Simulator.

Data Preparation

Data Compiling

In order for the data collected from the reactor simulator to be used to create a machine learning model, some data modifications needed to be performed. First, the data was in 39 separate datasets, to avoid issues with the constant moving, modifying, loading etc. of the data these sets were combined into one complete dataset. This set consisted of 30,710 data points, each consisting of the 33 measured features and the features added for reactor core life and transient event. Also, to minimize confusion and ensure only the data was imported into the machine learning model, the feature labels and heading information was not included in the final dataset. These preparations were done using Microsoft Excel and the final dataset was saved as a CSV file. The 39 individual datasets will be maintained in the event any unexpected issues occur with the complete dataset.

Modifications using Python

Once the data was compiled into a single dataset Python was used to modify the data. All code written for this project was done using Python version 3.7.2, the most up-to-date version available at the time. The scripts were written using Atom text editor and all code compiling was done using the Anaconda Python distribution. The complete CSV file was imported using the Pandas package. This converted the data from a CSV file into a Pandas `DataFrame`. No header was used in the importing of the data. The contents of the `DataFrame` was then verified using the `.head()`, `.shape()` and `.describe()` commands. The `.head()` command allows the user to view the contents of the first 5 rows of a dataset. This was done to ensure that all the features

appear correctly in the `DataFrame`. The `.describe()` command provides the descriptive statistics of the data stored in the `DataFrame`. This includes the mean, standard deviation, data point count, as well as the minimum and maximum values of each feature. The summary statistics for the first and last 3 columns of the dataset are shown in Figure 27. The `.shape()` command provides the This allowed for verification that all the data points had imported into the `DataFrame`.

The addition of the reactor core life introduced a qualitative feature into the dataset. Machine learning algorithms are only able to use quantitative data to produce a model. In order to properly account for the reactor lifetime, it was necessary to convert the qualitative data into quantitative data; this was done using dummy variables. Dummy variables are typically used to represent qualitative data in a 0, 1 scale. In this case, since there are three different types of qualitative data (BOL, EOL and MOL) three dummy variables and two extra factors were needed. It was possible to convert this data using the Pandas function `.get_dummies`. This function was used to create a dummy variable `DataFrame` using the reactor core life column of the dataset. The dummy variable `DataFrame` consists of 2 columns. BOL data points were converted to 0,0, EOL data points were converted to 1,0 and MOL were converted to 0,1. The dummy variable `DataFrame` was then added to the end of the dataset using Pandas' `.concat` function which is used to merge two or more `DataFrames`. Finally, the original reactor core lifetime column was dropped from the `DataFrame`. To ensure that the process had been done correctly, the new `DataFrame` was explored.

	0	1	2	...	19	20	21
count	30710.000000	30710.000000	30710.000000	...	30710.000000	30710.000000	30710.000000
mean	541.561225	521.515862	518.776119	...	2131.796682	590.801487	384.920847
std	105.355255	98.713384	107.142703	...	202.855889	136.808571	555.122586
min	183.631000	201.266000	98.079500	...	1700.000000	213.138000	-19.339300
25%	564.761000	559.310000	559.310000	...	2164.875000	648.124250	0.000000
50%	573.993000	562.153000	562.153000	...	2236.280000	652.786000	0.000025
75%	593.725750	564.770500	564.770500	...	2239.667500	653.004000	655.580000
max	622.519000	577.808000	577.808000	...	2314.580000	657.763000	1549.110000

Figure 27: Descriptive Statistics for Dataset

The next step in preparing the data was to prepare the target data of the dataset. As mentioned earlier, each data point was given a label of the transient event that occurred when the data was collected. This column was also a qualitative feature. Unlike the reactor core lifetime, there was no need to use dummy variables when modifying this dataset. Instead, each transient was designated a number: the feed water pump trip was assigned 1, the LOCA-LOOP 2, the steam generator valve closure was assigned 3 and the rapid power change was assigned a 4. Normal operations were assigned 0. Using Pandas' `.map` function it was possible to change all the qualitative data to the assigned numerical value. The dataset was once again explored to ensure that the process had been implemented correctly.

Data Splitting

The final step in preparing the reactor simulator dataset was to split the dataset into a training set and a testing set. In supervised machine learning, data should be split in order to validate the results. Validation allows for a measurement on the quality of the model's results. In the case of this project, validation is critical. Regulatory agencies, such as the Nuclear Regulatory Commission, have strict requirements in proving that any system or component within a reactor will behave as it is intended, especially if it will be relied upon in abnormal events. An important aspect of validation is that the data used in

the testing must be completely independent of the data used in creating the model. Failure to ensure this could result in biased models that do not learn the actual case of the testing data.

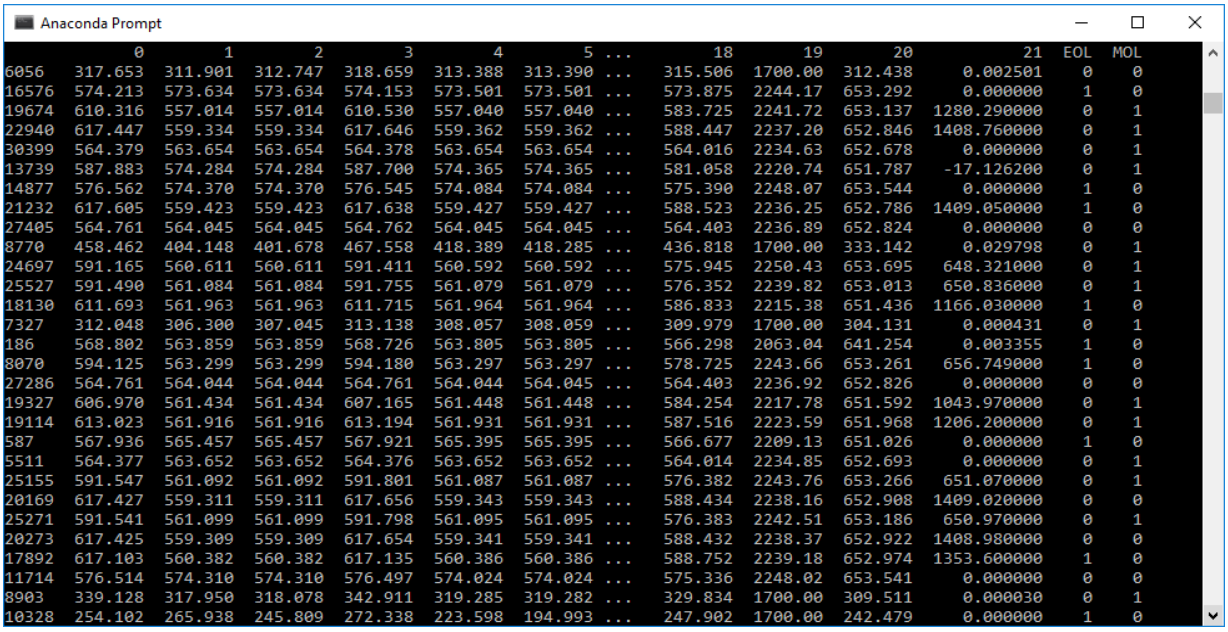
It is important to balance how much of the data is split between the two sets. If too little data is put into the training dataset the algorithm will not be able to learn the differences between the data points, this will result in less accurate models, which will be less effective in performing the task intended for the model. It is also necessary to have enough testing data. If the algorithm lacks sufficient testing data it will be difficult to verify that the model created by the supervised learning algorithm is reliable. Finally, as is the case in most statistical procedures, it is important that the data splitting be random to avoid any biases and to provide a good sample for both, the testing and training sets.

The data splitting for this project was done using the scikit-learn package. This is done using the `test_train_split` function. This function uses Bernoulli sampling in order to create testing and training sets that are pseudo-random. The pseudo-random nature of the splitting allows for the process to be repeated over and over again with no changes to the outcome while maintaining the randomness of the selection. The function requires that features and target data be provided as well, as the desired split between testing and training data. Also, the user may specify the seed of the random number generator, if desired. The output will be four different NumPy arrays, two arrays for the feature and target training data and two arrays will be for the feature and target testing data. These were labeled as `X_test` and `X_train` for the feature data and `Y_test` and `Y_train` for the target data. For this model, the target data will be the numerically-labeled transient types and the feature data will be the 33 features collected from the

reactor simulator. Half of the data collected will be used for training and the other half will be used for testing. The default random number seed for this function will be used for all data splitting on this project. The Python code used is shown in Figure 28. Figure 29 shows the output for the `X_train` array.

```
#Splitting the Dataset into Training and Testing Set 50 50 Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.5, test_size=0.5)
```

Figure 28: Test Train Split code for project.



	0	1	2	3	4	5 ...	18	19	20	21	EOL	MOL
6056	317.653	311.901	312.747	318.659	313.388	313.390 ...	315.506	1700.00	312.438	0.002501	0	0
16576	574.213	573.634	573.634	574.153	573.501	573.501 ...	573.875	2244.17	653.292	0.000000	1	0
19674	610.316	557.014	557.014	610.530	557.040	557.040 ...	583.725	2241.72	653.137	1280.290000	0	1
22940	617.447	559.334	559.334	617.646	559.362	559.362 ...	588.447	2237.20	652.846	1408.760000	0	1
30399	564.379	563.654	563.654	564.378	563.654	563.654 ...	564.016	2234.63	652.678	0.000000	0	1
13739	587.883	574.284	574.284	587.700	574.365	574.365 ...	581.058	2220.74	651.787	-17.126200	0	1
14877	576.562	574.370	574.370	576.545	574.084	574.084 ...	575.390	2248.07	653.544	0.000000	1	0
21232	617.605	559.423	559.423	617.638	559.427	559.427 ...	588.523	2236.25	652.786	1409.050000	1	0
27405	564.761	564.045	564.045	564.762	564.045	564.045 ...	564.403	2236.89	652.824	0.000000	0	0
8770	458.462	404.148	401.678	467.558	418.389	418.285 ...	436.818	1700.00	333.142	0.029798	0	1
24697	591.165	560.611	560.611	591.411	560.592	560.592 ...	575.945	2250.43	653.695	648.321000	0	1
25527	591.490	561.084	561.084	591.755	561.079	561.079 ...	576.352	2239.82	653.013	650.836000	0	1
18130	611.693	561.963	561.963	611.715	561.964	561.964 ...	586.833	2215.38	651.436	1166.030000	1	0
7327	312.048	306.300	307.045	313.138	308.057	308.059 ...	309.979	1700.00	304.131	0.000431	0	1
186	568.802	563.859	563.859	568.726	563.805	563.805 ...	566.298	2063.04	641.254	0.003355	1	0
8070	594.125	563.299	563.299	594.180	563.297	563.297 ...	578.725	2243.66	653.261	656.749000	1	0
27286	564.761	564.044	564.044	564.761	564.044	564.045 ...	564.403	2236.92	652.826	0.000000	0	0
19327	606.970	561.434	561.434	607.165	561.448	561.448 ...	584.254	2217.78	651.592	1043.970000	0	1
19114	613.023	561.916	561.916	613.194	561.931	561.931 ...	587.516	2223.59	651.968	1206.200000	0	1
587	567.936	565.457	565.457	567.921	565.395	565.395 ...	566.677	2209.13	651.026	0.000000	1	0
5511	564.377	563.652	563.652	564.376	563.652	563.652 ...	564.014	2234.85	652.693	0.000000	0	1
25155	591.547	561.092	561.092	591.801	561.087	561.087 ...	576.382	2243.76	653.266	651.070000	0	1
20169	617.427	559.311	559.311	617.656	559.343	559.343 ...	588.434	2238.16	652.908	1409.020000	0	0
25271	591.541	561.099	561.099	591.798	561.095	561.095 ...	576.383	2242.51	653.186	650.970000	0	1
20273	617.425	559.309	559.309	617.654	559.341	559.341 ...	588.432	2238.37	652.922	1408.980000	0	0
17892	617.103	560.382	560.382	617.135	560.386	560.386 ...	588.752	2239.18	652.974	1353.600000	1	0
11714	576.514	574.310	574.310	576.497	574.024	574.024 ...	575.336	2248.02	653.541	0.000000	0	0
8903	339.128	317.950	318.078	342.911	319.285	319.282 ...	329.834	1700.00	309.511	0.000030	0	1
10328	254.102	265.938	245.809	272.338	223.598	194.993 ...	247.902	1700.00	242.479	0.000000	1	0

Figure 29: Sample from X Train Dataset

Results & Analysis

K-Nearest Neighbors Results

The entire process of building and evaluating the k-nearest neighbors model in TPOT took approximately 1 hour and 30 minutes. The accuracy of this model was 98.35%, the precision was 98.02%, recall was calculated to be 98.01%, and the F1 score was also calculated to be 98.01%. Table 5 shows the individual accuracies for each transient from this model.

TABLE 5: K-Nearest Neighbors Model Individual Transient Accuracies

Normal Operations	Feed Water Pump Trip	LBLOCA + LOOP	Valve Closure	Rapid Power Change
100%	96.6%	97.57%	98.01%	97.86%

The k-nearest neighbors method was able to correctly identify 15,103 instances of the 15,355 samples tested during the validation process. Of the 252 misclassified instances, the largest amounts of misclassifications were from the feed water pump trip transient. 172 of the 252 misclassifications, 60% of total errors, were from this transient. Of those 172 errors, 95 were false positives and 77 were false negatives. The model's biggest issue was distinguishing the feed water pump trip transient from the valve closure transient: a total of 76 instances, 30% of the total misclassifications were between these two transients. The k-nearest neighbors model was able to perfectly distinguish normal operation instances from transient instances as there were no type I or type II errors for the normal operation transient. Figure 30 shows the confusion matrix for the k-nearest neighbor model. The code was designed to export the instances where misclassifications

occurred. Initial analysis of these instances showed no true pattern or bias of when in the transient the misclassifications occurred. Figure 31 shows a graph of these misclassifications.

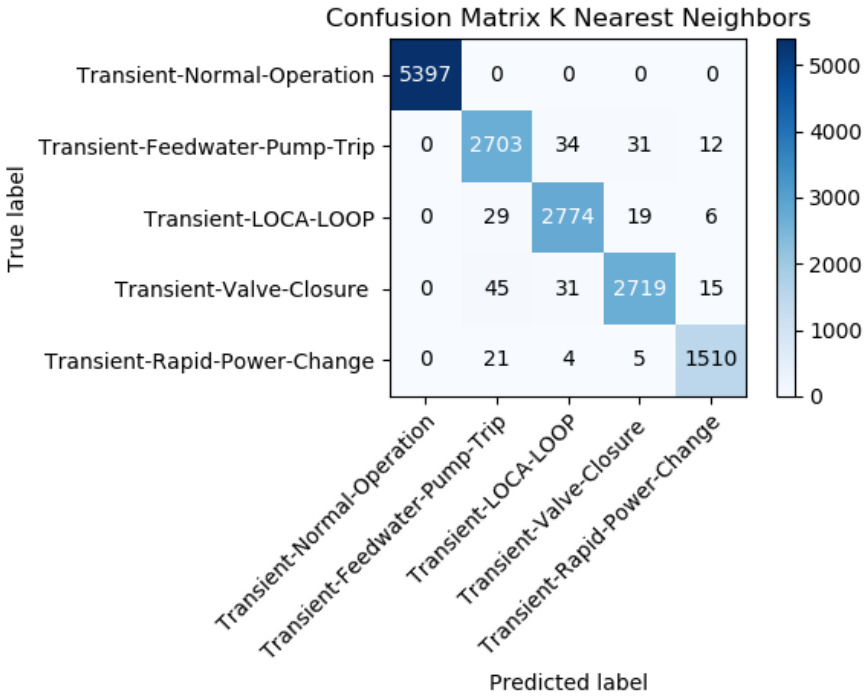


Figure 30: Confusion Matrix for K-Nearest Neighbors Models

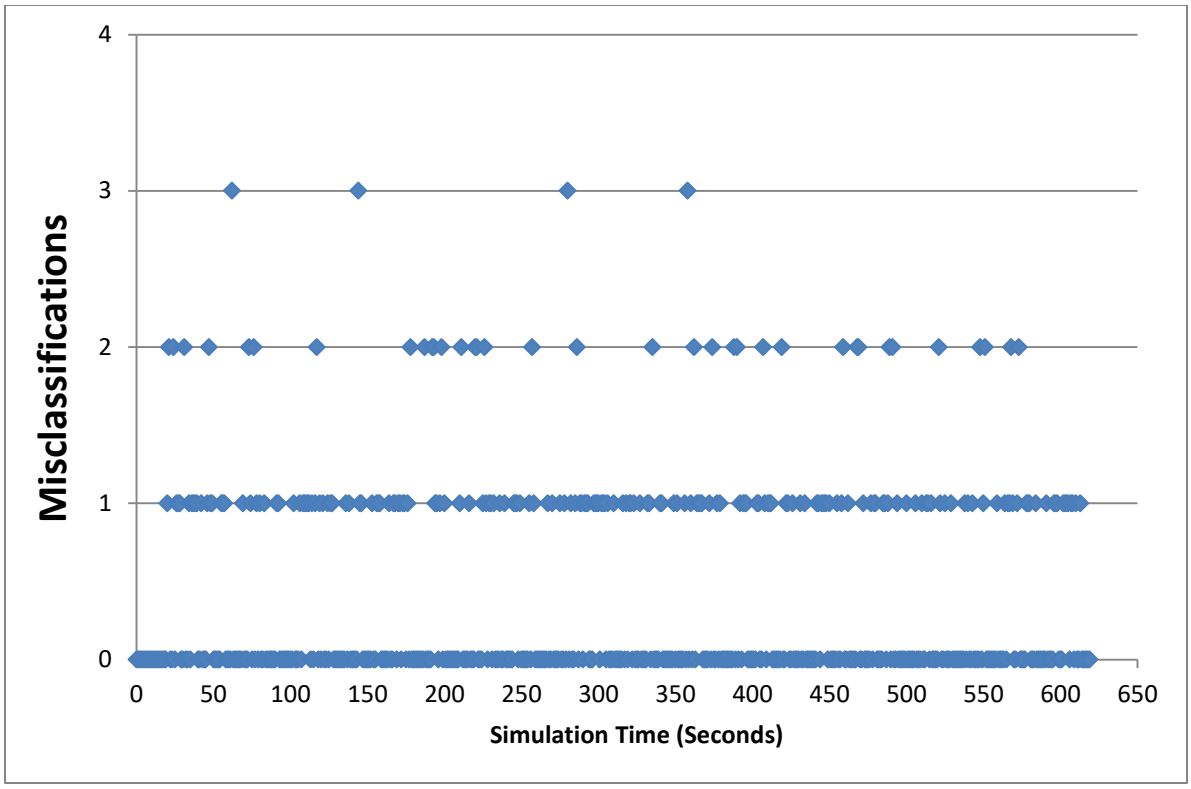


Figure 31: Graph of Misclassifications for K-Nearest Neighbors Models

Bernoulli Naïve Bayes Results

In TPOT, the Bernoulli naïve Bayes model took approximately 1 hour to build and validate. The accuracy of this model was 97.45%, the precision was calculated to be 97.18%, the recall of the model was 96.73 %, and the F1 score was 96.87%. Table 6 shows the accuracies of the individual transients from this model.

TABLE 6: Bernoulli Naïve Bayes Model Individual Transient Accuracies

Normal Operations	Feed Water Pump Trip	LBLOCA + LOOP	Valve Closure	Rapid Power Change
100%	90.05%	100%	96.1%	93.72%

The Bernoulli naïve Bayes model correctly identified 14,964 instances of the 15,355 tested. Of the 391 incorrect classifications 258 of them, 66% of total misclassifications, were from the feed water pump trip transient. Of these, 191 were false positives for the valve closure transient. The model also 33 false positive classifications for the rapid power change transient. Under this configuration, the model was able to correctly distinguish between a transient and non-transient event with no Type I or Type II errors for the normal operations event. The Bernoulli naïve Bayes model had no Type I errors for the LOCA-LOOP transient, nor were there any Type II errors for the rapid power change transient. The confusion matrix for the Bernoulli naïve Bayes model is shown in Figure 32. Initial analysis of the misclassified instances did show some possible grouping of the misclassifications around the half way point of the simulation, between 200 & 400 seconds. This is shown in Figure 33. Also of note, most of the misclassifications occurred at 10 second points. The reason behind this is unclear and this trend did not occur in the other models.

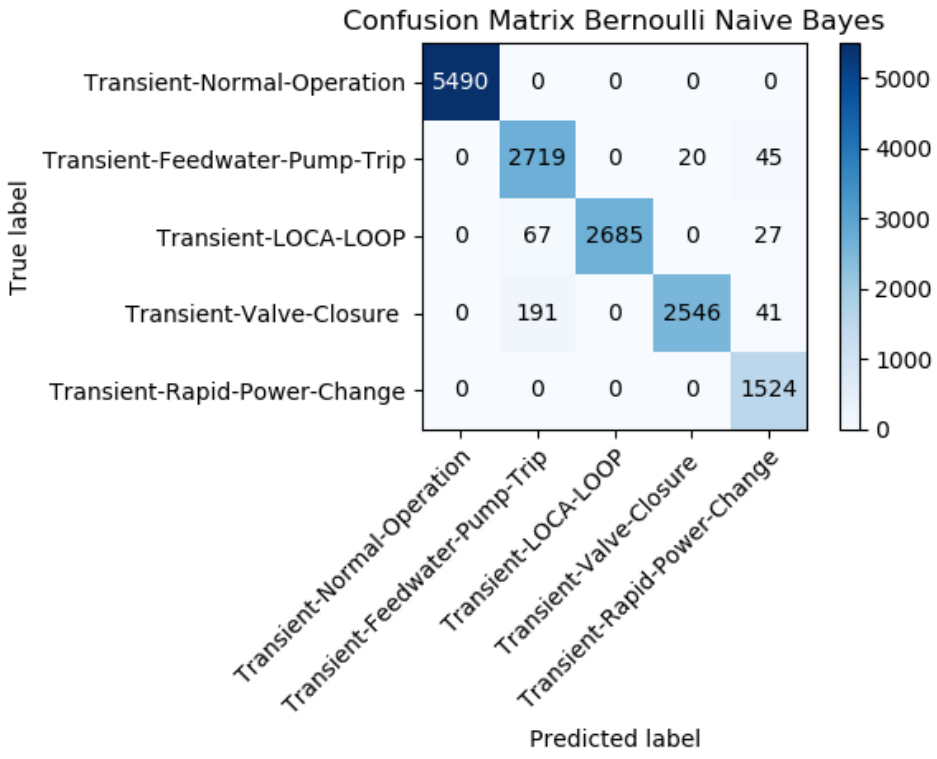


Figure 32: Confusion Matrix for the Bernoulli Naïve Bayes Model

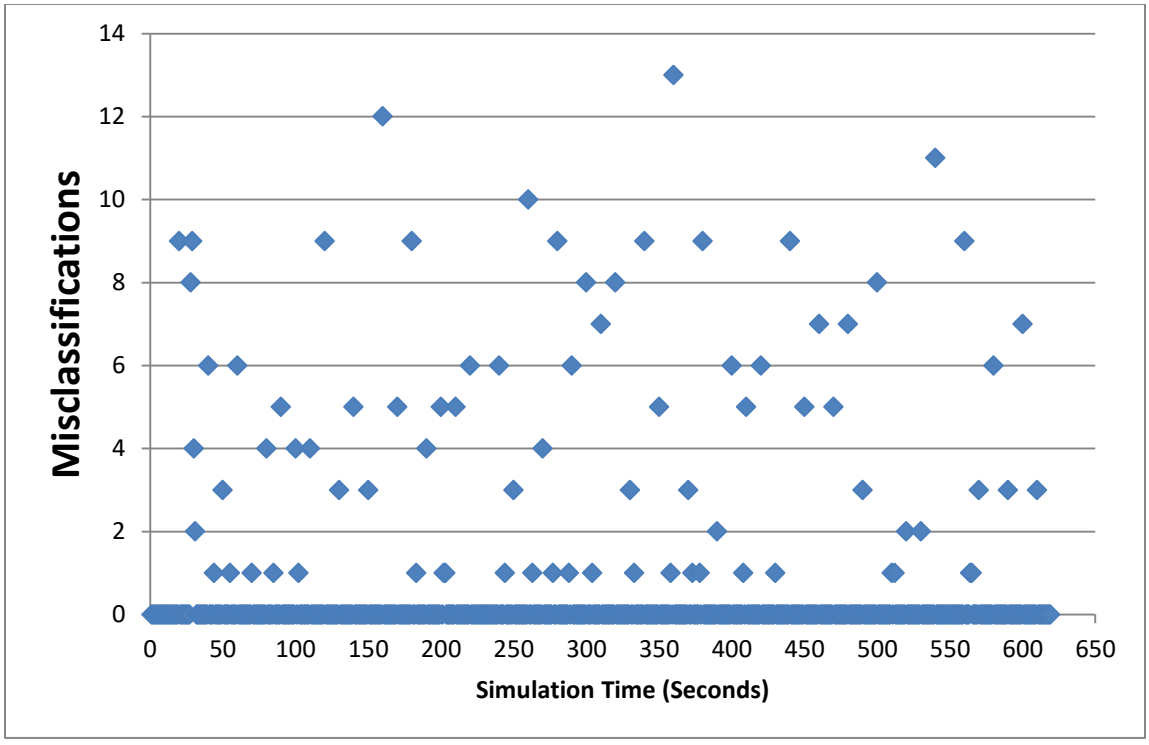


Figure 33: Graph of Misclassifications for Bernoulli Naïve Bayes Model

Gaussian Naïve Bayes Results

Similar to the Bernoulli naïve Bayes model, the Gaussian naïve Bayes model took approximately 1 hour to build and test. The accuracy of this model was found to be 97.45%. The precision was scored at 97.2%, the recall was calculated at 96.83%, and the F1 score was 96.96%. Table 7 shows the model's accuracy for the individual transients.

TABLE 7: Gaussian Naïve Bayes Model Individual Transient Accuracies				
Normal Operations	Feed Water Pump Trip	LBLOCA + LOOP	Valve Closure	Rapid Power Change
100%	88.77%	100%	96.59%	94.0%

This model was able to correctly identify 14,833 of the 15,355 samples tested. The Gaussian naïve Bayes model performed perfectly in identifying the non-transients and transients as there were no false positives or negatives for the normal operation event. There were also no false positives for the LOCA LOOP transient and no false negative for the rapid power change transient. The model struggled the most with correctly classifying the feed water pump trip transient. Of the 522 misclassified transients 337 of them, nearly 65% of all the model's total errors were from this transient. Of those, 302 were misclassifications between the feed water pump transient and the valve closure transient. The confusion matrix for the Gaussian naïve Bayes model is shown in Figure 34. Initial analysis of the misclassified instances shows that there was no grouping and that misclassifications were spread throughout the simulation. Figure 35 shows a graph of the misclassifications.

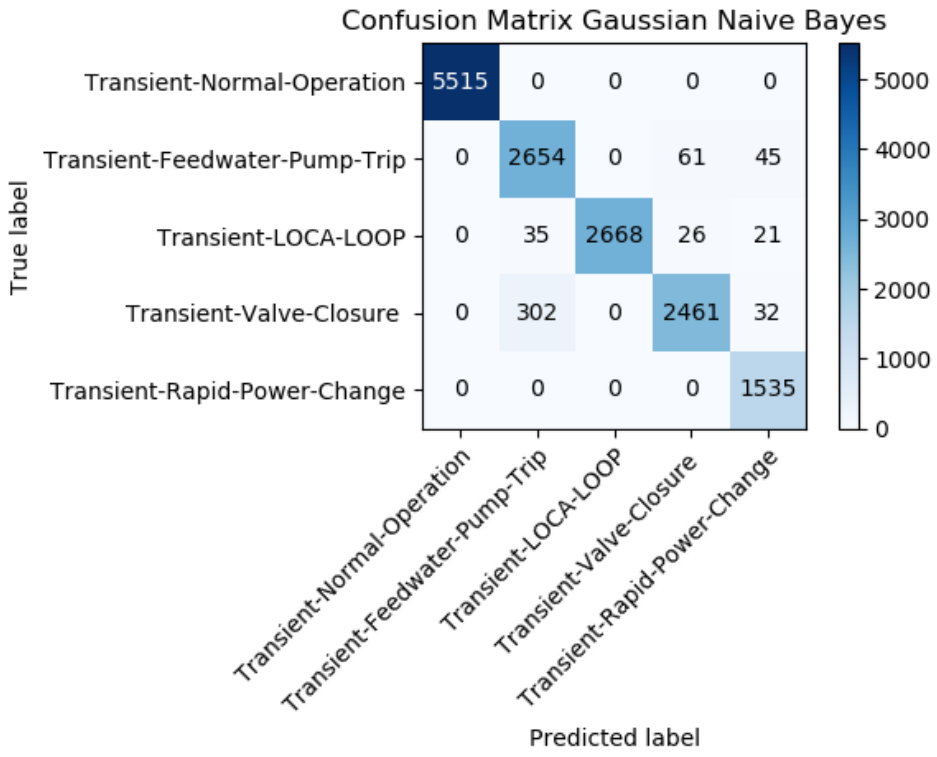


Figure 34: Confusion Matrix for the Gaussian Naïve Bayes Model

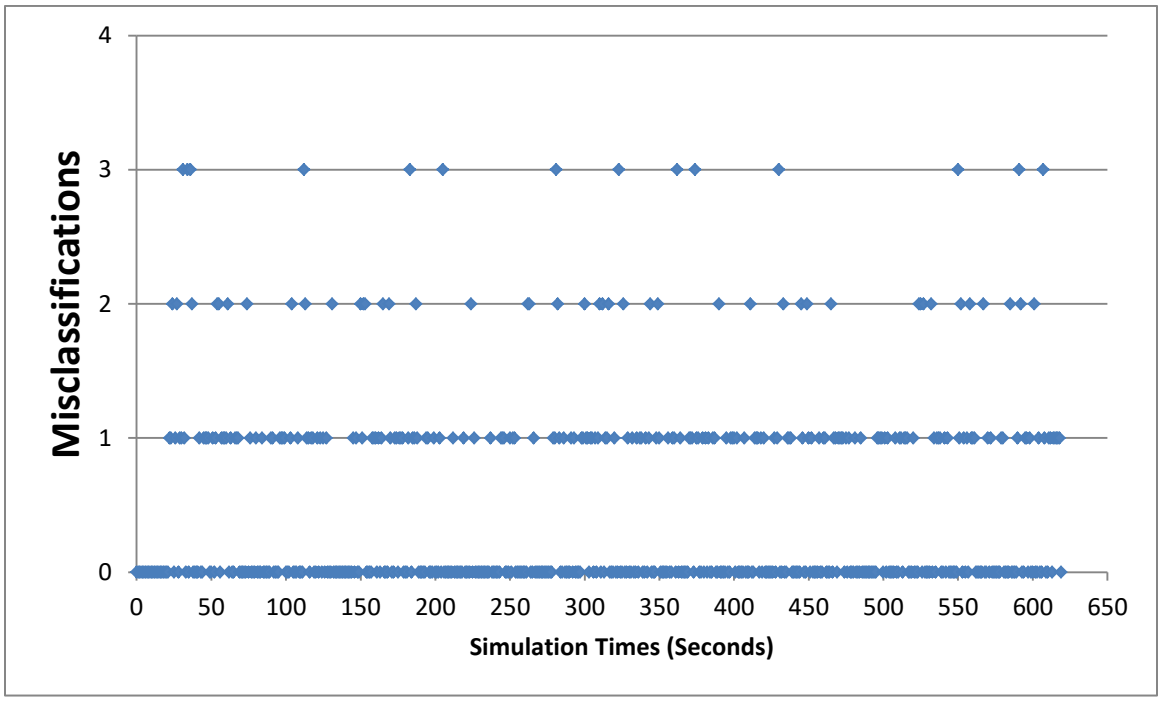


Figure 35: Graph of Misclassifications for Gaussian Naïve Bayes Model

Multinomial Naïve Bayes Results

The multinomial naïve Bayes model took approximately 1 hour to be built and tested using TPOT. The accuracy of the multinomial naïve Bayes model was 96.71%. The precision of this model was calculated to be 96.38%, the recall was 95.41%, and the F1 score was calculated to be 96.10%. Table 8 shows the accuracies of the individual transients from this model.

TABLE 8: Multinomial Naïve Bayes Model Individual Transient Accuracies

Normal Operations	Feed Water Pump Trip	LBLOCA + LOOP	Valve Closure	Rapid Power Change
100%	90.05%	100%	96.1%	93.72%

The multinomial naïve Bayes model was able to correctly classify 14,833 of the reactor transient instances tested. Similar to the other naïve Bayes models, the multinomial method was able to perfectly distinguish between transient events and non-transient events, as there were no Type I or Type II errors for normal operations. The rapid power transient also had no false negative results and the LOCA LOOP transient had no false positives. Also, the model struggled most with the feed water pump trip transient with 296 misclassifications occurring with this transient. The confusion matrix for this model is shown in Figure 36. The misclassified instances show that most of the misclassifications are spread throughout the simulation with no pattern or large groups. This is shown in Figure 37.

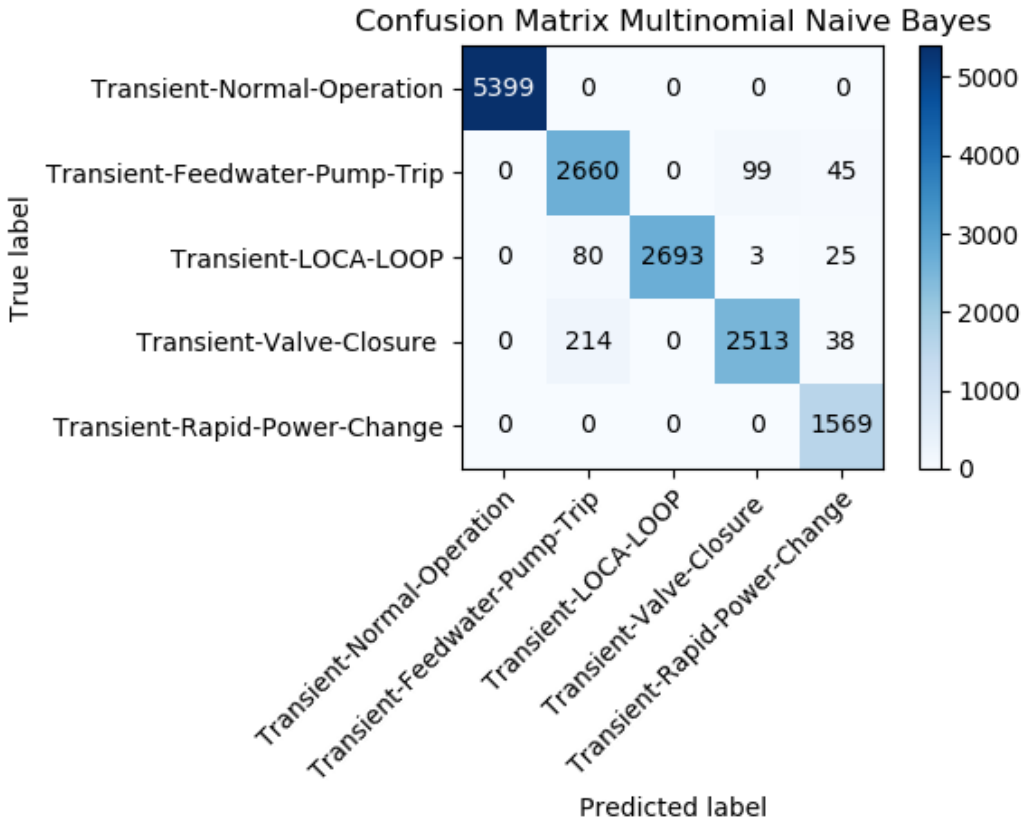


Figure 36: Classification Matrix for Multinomial Naïve Bayes Model

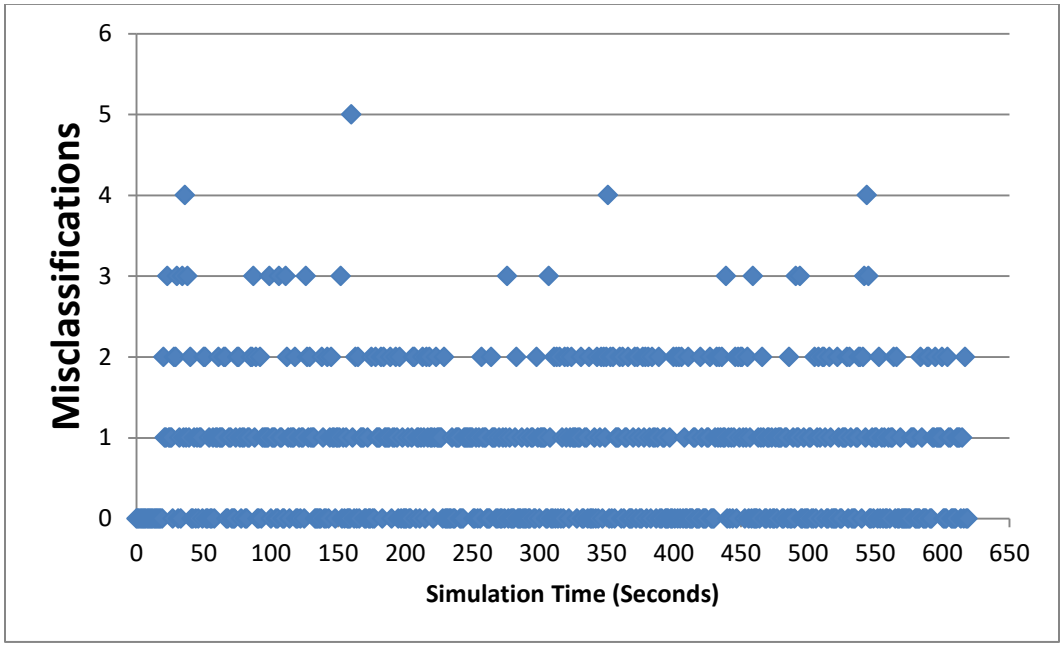


Figure 37: Graph of Misclassifications for Multinomial Naïve Bayes Model

Logistic Regression Results

The logistic regression model took approximately 48 hours to run. This was the most computationally expensive of all the models evaluated. The accuracy of the logistic regression model was found to be 98.55%. The precision was calculated to be 98.41%, recall was 98.04% and the F1 score was found to be 98.21%. Table 9 shows the individual accuracies for the reactor transients for this.

This model correctly identified 15,133 transient instances of the 15,355 samples tested. The logistic regression model perfectly classified transient and non-transient events; there were no false positives or negatives from the normal operation event. The model also had no false positives for the LOCA LOOP and there were no false negatives for the rapid power change. The model scored well on all the transients with accuracies above 95% across all 5 events. The model had the highest number of misclassifications with the feed water pump, though the rapid power change had a lower accuracy. The largest number of errors, 52, came from false positives of the feed water pump transient from the valve closure transient. Figure 38 shows the confusion matrix for this model. No easily identified groups were found when looking at the misclassified instances, similar to some of the other models the misclassifications appear spread out. Figure 39 shows a graph of the misclassifications.

TABLE 9: Logistic Regression Model Individual Transient Accuracies

Normal Operations	Feed Water Pump Trip	LBLOCA + LOOP	Valve Closure	Rapid Power Change
100%	96.87%	100%	97.62%	95.71%

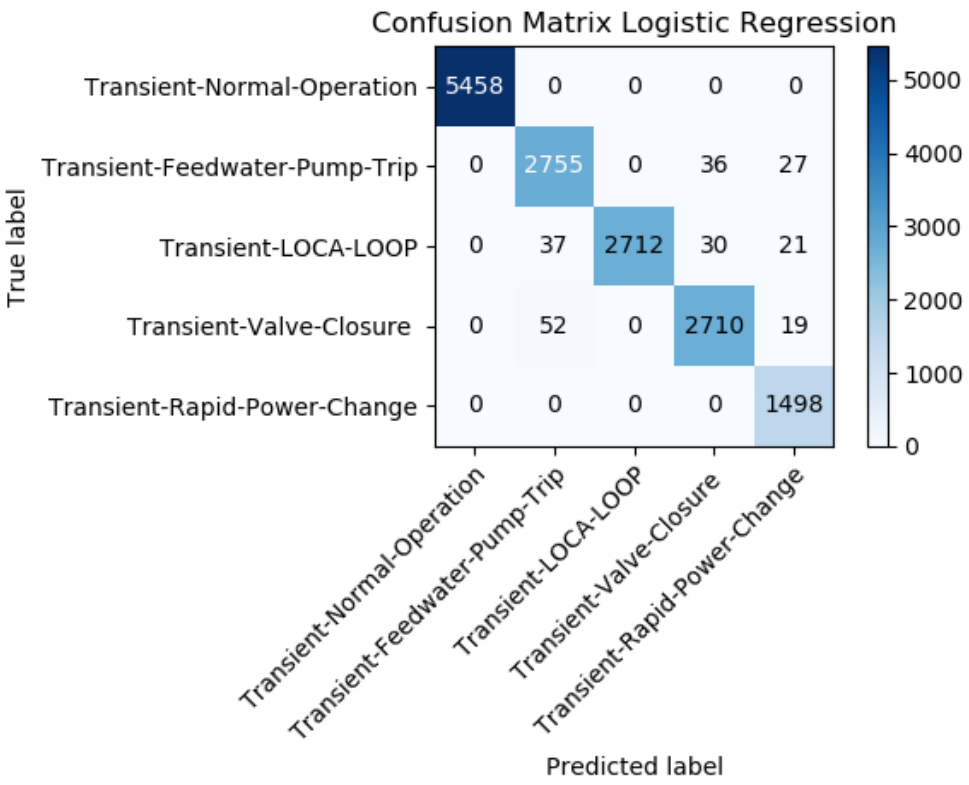


Figure 38: Confusion Matrix for the Logistic Regression Model

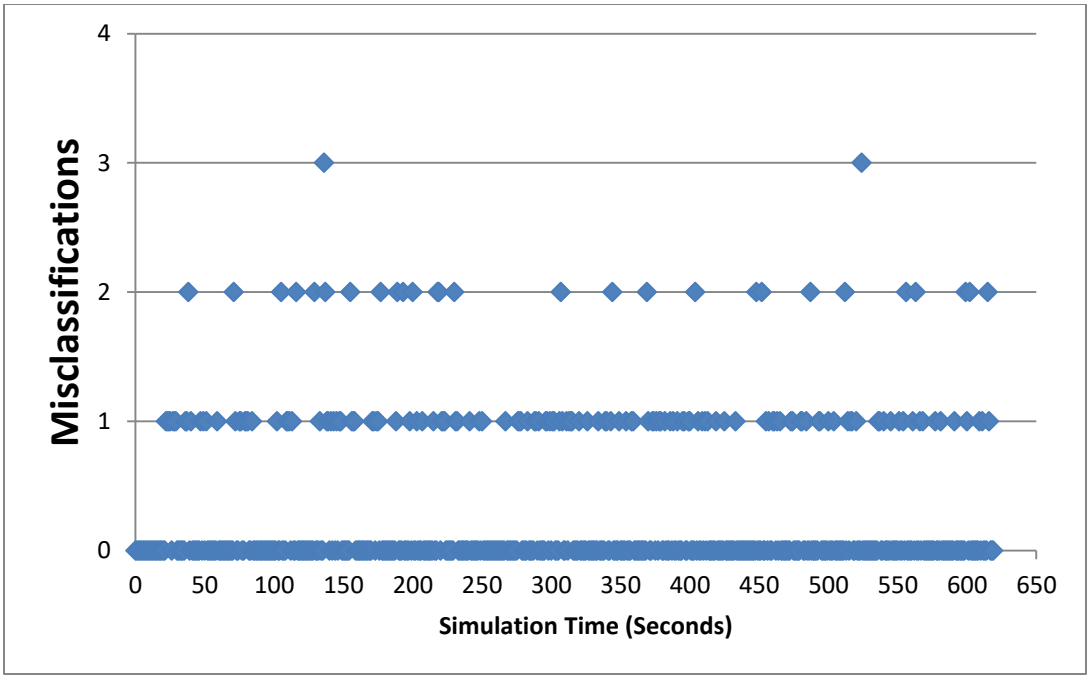


Figure 39: Graph of Misclassifications for Logistic Regression Model

Decision Tree Results

The decision tree model took approximately 3 hours to build and validate. The accuracy of this model was 98.6%, precision was calculated at 98.46%, recall was found to be 98.1% and the F1 score was 98.27%. Table 10 shows the individual accuracies of the transient events for the decision tree model.

The decision tree model was able to classify correctly 15,140 of the 15,355 transient instances tested. The model was able to perfectly classify all of the normal operation instances and there were no false positive or false negative errors from that event. The model was able to classify the LOCA LOOP transient with no false positives and the rapid power change transient had no false negatives. As with the other models, the decision tree model's biggest issues were from the feed water pump transient: 35% of the errors and 76 instances were from this transient. Forty one of those were false positives with the valve closure transient. The confusion matrix for this model is shown in Figure 40. Looking at misclassified instances, no obvious grouping appeared. With the exception of the Bernoulli naïve Bayes model, it appears that the misclassifications experienced were typically spread out rather than grouped together. Figure 41 shows the graph for the decision tree model's misclassifications.

TABLE 10: Decision Tree Model Individual Transient Accuracies

Normal Operations	Feed Water Pump Trip	LBLOCA + LOOP	Valve Closure	Rapid Power Change
100%	97.33%	100%	97.57%	95.62%

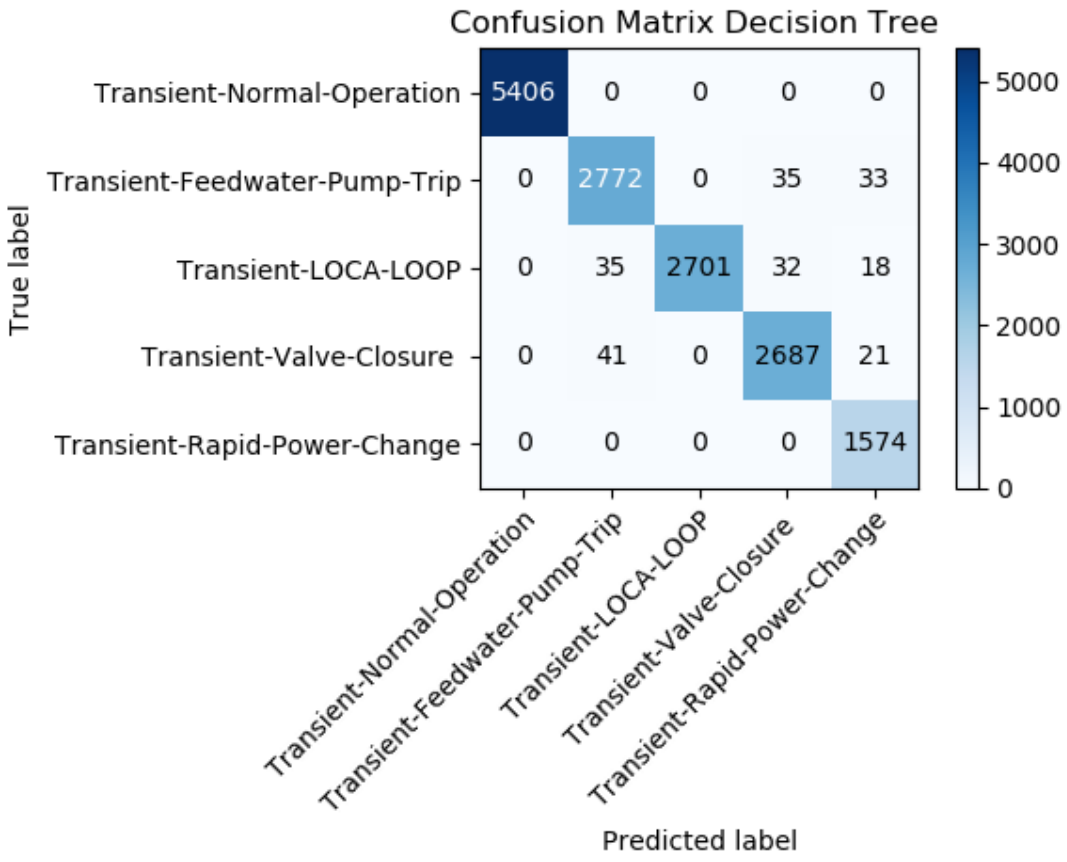


Figure 40: Confusion Matrix for Decision Tree Model

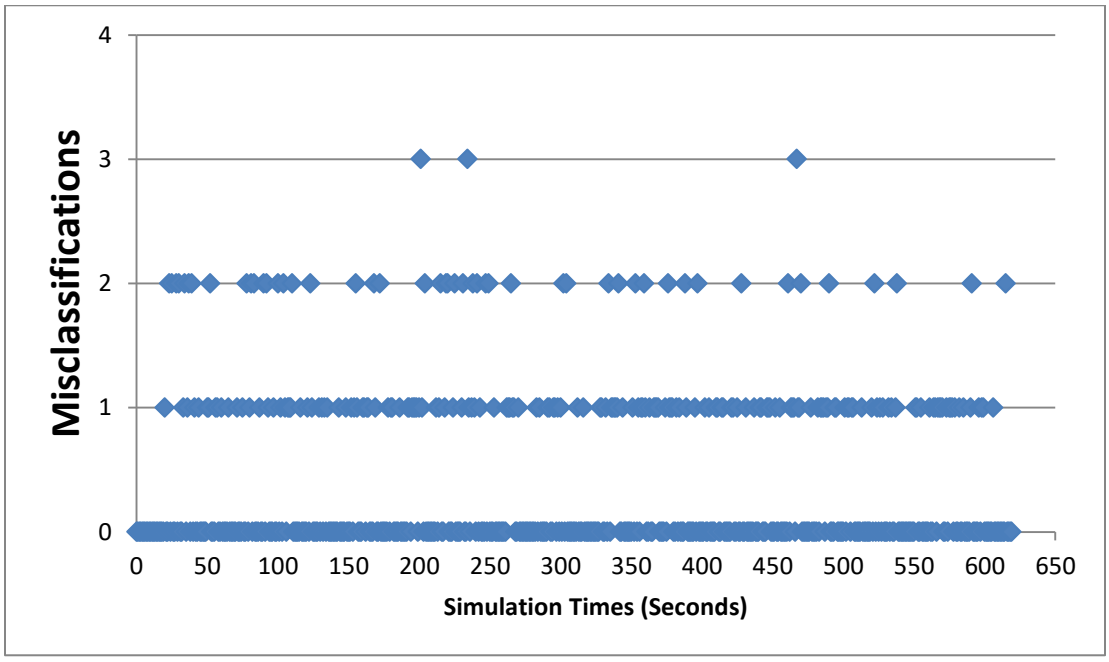


Figure 41: Graph of Misclassifications for Decision Tree Model

Conclusions

The results from the machine learning models show very positive results. All of the models had validation scores in the mid-90's. Table 11 below summarizes the result from all 6 of the machine learning models. Under the configurations selected for the TPOT dictionary, all of the models were able to perfectly tell the difference between normal operations and transient events. It should be noted that the dataset did contain more of this type of data, but this should not be an issue, as with real nuclear reactors the amount of data, as well as the quality, will almost certainly be higher for a real reactor under normal operations. With the exception of the k-nearest neighbors model, the models were able to able to classify the LOCA LOOP transient with perfect accuracy, though there were false negative classifications across all these models in the study.

All of the models had the most difficulty distinguishing between the feed water pump trip and the valve closure transients, as this transient had the lowest individual accuracy of the five events across all the models. Also, a large percentage of the total errors from these models came from false positives between these two transients. The models appear to have a tendency towards having more Type II error over Type 1, as the precision of all the models is higher than the recall. Since Type II error can result in a more dangerous scenario with a nuclear reactor, it is important that the recall always be considered when making determinations.

In terms of performance, the decision tree, k-nearest neighbors and logistic regression models were better than the naïve Bayes models, having validation measurements all above 98%. The decision tree model performed the best. Interestingly,

the decision tree model slightly outperformed the logistic regression model. This is important as the logistic regression model is more computationally expensive, requiring 48 hours to compute on its initial run, while the decision tree model only took 3 hours. In this case, it is likely the data was not overly complex, so the decision tree model was able to better fit the data without needing to perform the large amount of feature analysis required with the logistic regression model. This is encouraging, as time is a major consideration when selecting a model that will be used in real-time. Using more sophisticated equipment, it is possible that a decision tree or k-nearest neighbors model could provide valuable information to reactor operators in a matter of minutes, if a model needs to be trained quickly. A TPOT model was constructed early on in the experiment without defining a specific machine learning model to be built. The optimal model from this run was a decision tree, with an accuracy of 98.57%. This verifies the results from the individual tests, where the decision tree was the most accurate model. A possible reason for the small decrease in accuracy is the change in random state that occurred in the evaluation.

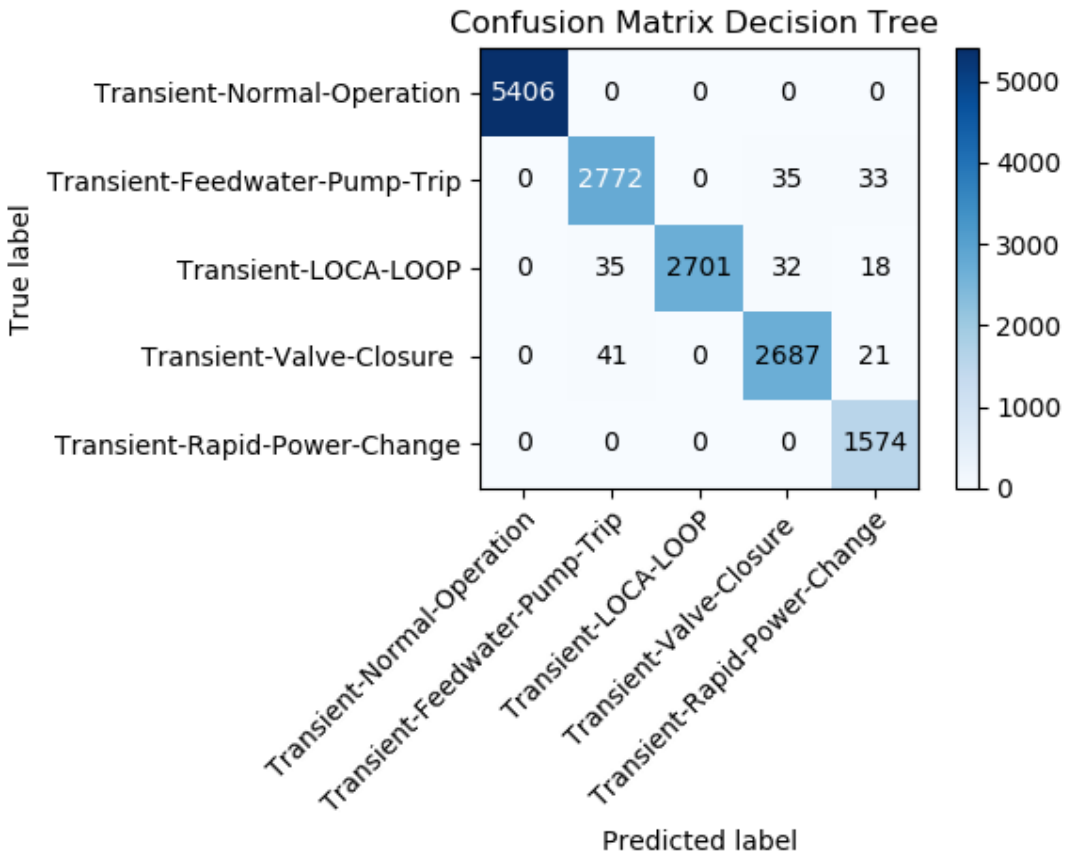
The naïve Bayes models, while having high validation measurements, did not perform as well as the other three models. The multinomial model had the lowest accuracy of the six models. A likely cause of this is that the data better fit the Gaussian and Bernoulli distributions than the multinomial. The accuracies with the feed water pump transient were the lowest with the Gaussian model only scoring 88% accuracy for that transient. The most likely cause of this is that the probabilities calculated by the naïve Bayes models favored the feed water transient over the valve closure, resulting in the false positive classifications. The overall accuracies of the Gaussian and Bernoulli

models were identical. This is likely coincidental, as both models have different accuracies for three of the individual transients.

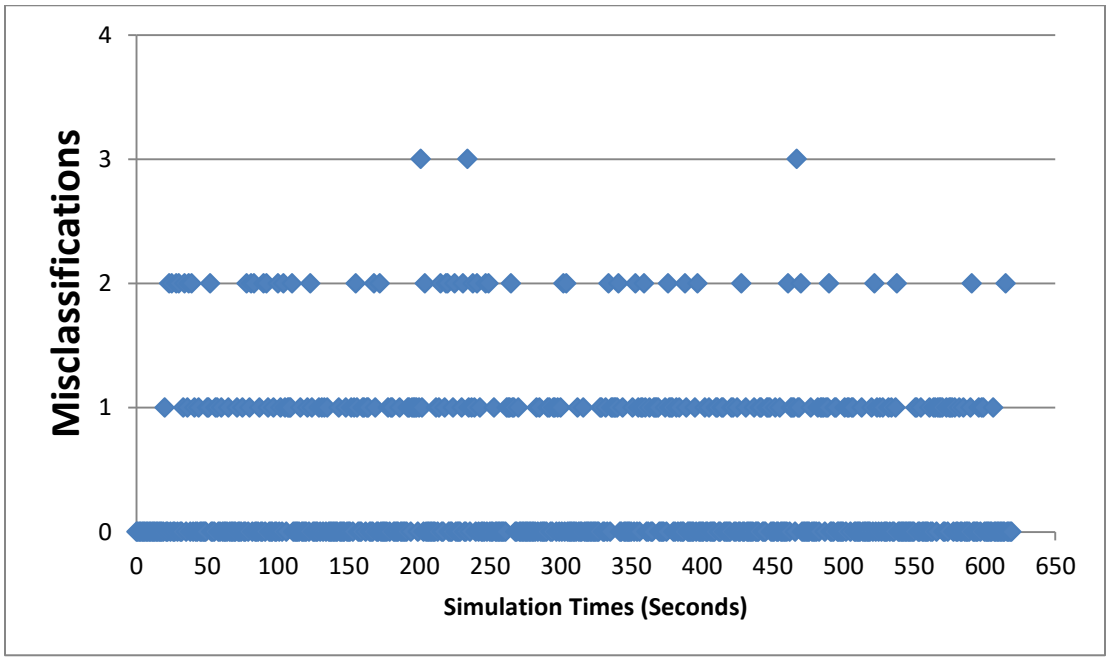
In order to see each model's sensitivity to changes in random numbers, two additional runs were performed for each model using different random states. No model experienced a change greater than half of a percent for an overall validation measurement. The model with the largest change was the multinomial naïve Bayes model. During the third simulation, the model's accuracy increased to 97.11% from 96.71% in the initial run. The most interesting change was that all the logistic regression's results increased in the additional runs and the decision tree model's results decreased. Further exploration of this could help better determine which model is better for this classification. The k-nearest neighbors model experienced the least changes with the additional runs.

In the additional runs, there were changes to the individual accuracies as expected. There was a single false positive of a normal operation point in the third Bernoulli naïve Bayes model. This was the only normal operation misclassification from all 18 runs done for this project. Also, the multinomial naïve Bayes model struggled with the rapid power transient in the third run, with an individual accuracy of 86.11%, the lowest individual accuracy for any transient in this project. The decision tree and logistic regression models experienced issues with the LOCA-LOOP transient. In the initial run, both models had accuracies of 100%. In the additional runs, both models experienced misclassifications. The decision tree model scored 97.55% and 97.71% and the logistic regression model scored 99.81% and 99.77% for this transient. The complete results from this analysis can be found in Appendix 2.

Due to the high validation measurements, it does appear from this analysis that there is promise in the area of applying machine learning to reactor safety. Applying trained machine learning models to reactor safety could lead to faster transient diagnoses, accident mitigation, and help keep the general public better informed of issues at a nuclear power plant. Areas that could be further explored include introducing more transients and more complex normal operation data to the dataset to see how the models perform and which models perform better with the more complex data. Another interesting area would be applying data from other similar reactor simulators to see if the models change much with a different reactor. Also, further exploration into the errors within these models to look for more patterns and factors behind the existing errors, would be another possible area of research. Finally, another worthwhile area of research would be, exploring other machine learning models, such as unsupervised learning, to see if the models can blindly group transient data.



Confusion Matrix for Decision Tree Model



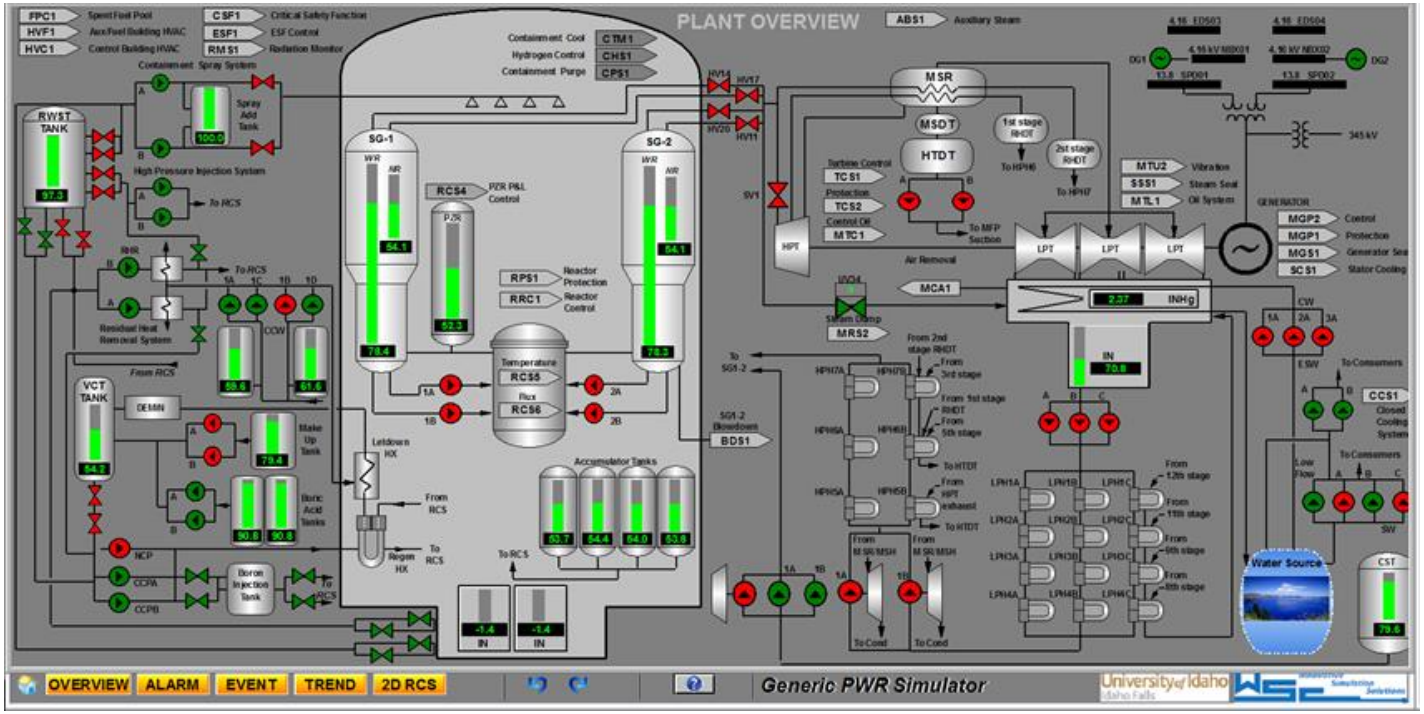
Misclassification Graph for Decision Tree Model

Table 11: Summary of Machine Learning Model Results										
Overall Validation Measurements					Individual Transient Accuracies					Time
	Accuracy	Precision	Recall	F1 Score	Normal Operation Transient	Feed Water Pump Trip	LOCA + LOOP	Valve Closure	Rapid Power Change	Time Required (Approx.)
K-Nearest Neighbors	98.35%	98.02%,	98.01%	98.01%	100%	96.6%	97.57%	98.01%	97.86%	1.5 Hours
Bernoulli Naïve Bayes	97.45%,	97.18%,	96.73 %	96.87%.	100%	90.05%	100%	96.1%	93.72%	1 Hour
Gaussian Naïve Bayes	97.45%	97.2%	96.83%	96.96%	100%	88.77%	100%	96.59%	94.0%	1 Hour
Multinomial Naïve Bayes	96.71%.	96.38%,	95.41%	96.10%.	100%	90.05%	100%	96.1%	93.72%	1 Hour
Logistic Regression	98.55%.	98.41%,	98.04%	98.21%	100%	96.87%	100%	97.62%	95.71%	48 Hours
Decision Tree Analysis	98.6%	98.46%,	98.1%	98.27%.	100%	97.33%	100%	97.57%	95.62%	3 Hours

Summary

The purpose of this paper is to explore the ability of machine learning algorithms to identify transient events occurring with a nuclear reactor. In order to perform this experiment, it was necessary to gather nuclear reactor data to create machine learning models. Due to the extremely high cost of nuclear reactors, as well as the possible serious health and environmental consequences of a nuclear power plant accident, it is impossible to create situations where an actual nuclear reactor experiences serious events or accident conditions that could be used to gather data. Instead, a reactor simulator was used to gather the synthetic data needed to apply the machine learning algorithms. Once the data was properly formatted, the data was applied using machine learning algorithms within two python machine learning packages, TPOT and scikit-learn. This produced several different machine learning models in the form of supervised learning. Finally, each machine learning model created was validated using measurements, such as accuracy and precision.

Data used for this experiment was collected using a Generic Pressurized Water Reactor (GPWR) simulator at the Center for Advanced Energy Studies (CAES). The simulator was purchased by University of Idaho, Idaho State University and other CAES institutions from the Western Service Corporation (WSC). This simulator emulates the behavior of a "generic" pressurized water reactor (PWR). The output is rated at 4000 MW thermal/1400 MW electric. The simulator does not directly incorporate the design of any specific PWR. The reactor systems emulated include a high-pressure turbine and three low-pressure turbines and a configuration that includes: two loops, four coolant pumps and two steam generators. The interface for the simulator is shown below.



GPWR Simulator Interface

The GPWR simulator comes preloaded with 14 different initial conditions for the reactor; factors that vary include reactor power, core life, etc. These are shown in the table below. The simulator has several reactor control panels that allow the user to change the reactor system. These include: the operation of pumps, the opening/closing of valves, the flow of coolant, reactor power, etc. The simulator is equipped with an alarm system to inform the user of abnormal conditions occurring within the system. Similar to an actual reactor, the simulator can be scrammed by the operator and automatically under certain conditions, including both a reactor trip and a turbine trip. The simulator is also programmed to emulate the containment structure, engineering safety features and the behavior of these components under accident conditions. The software was designed with the ability to mimic malfunctions of components that could potentially occur within a power plant. The reactor will behave accordingly to a malfunction. Some of these malfunctions include: heat exchanger degradation, motor shearing/seizures, valve failure to open/close, etc. These events can be triggered by the user or programmed as part of an accident scenario.

GPWR Preprogramed Initial Conditions				
BOL, 100% Power	MOL, 100% Power	EOL, 100% Power	BOL, 50% Power	MOL, 50% Power
EOL, 50% Power	BOL, 1% Power	MOL, 1% Power	EOL, 1% Power	BOL, Subcritical
MOL, Subcritical	EOL, Subcritical	BOL, Xe Equilibrium	MOL, 5% Power	

In order to construct machine learning models, it was necessary to decide on the features that would be measured using the GPWR reactor simulator. The simulator is able to measure and collect data from several reactor components, such as reactor power output and steam generator pressure. It was decided that for this model, the data gathered would consist of data that a reactor

operator would have access to and that would be readily available. Thirty three features were chosen and programmed into the simulator data collector. These features include: reactor power output, steam generator temperature and pressure etc. The table below shows these features. All of the features collected from the reactor simulator were quantitative in nature.

Features Collected from GPWR Simulator					
Normalized Flux	RCS LVL Loop 1 WR	RCS LVL Loop 1 NR	Hot Leg 1/ 2 Temperature	Cold Leg 1/ 2A Temperature	Cold Leg 1/2B Temperature
RC Loop-1/2A Norm Flow	RC Loop - 1/2B Norm Flow	Pressurizer Surge Line Temperature	PORV Discharge Pressurizer Temperature	Containment Pressure	Containment Temperature
MS Flow from SG-1 Line-1/2B	SG-1/2 Pressure	Average Temperature	SG-1/2 NR Level	FW Flow to SG-1/2	Pressurizer Pressure
Pressurizer Steam Temperature	Norm Pressurizer Level	Pressurizer Water Temperature	Generator Power	MS Flow from SG-1 Line-1/2A	

In order to see how the model would be impacted by changes in the reactor system overtime, it was decided to conduct several runs changing the initial conditions of the simulated system. The first change to the system was the reactor power output. Three different conditions were used: full power, where the reactor is operating as to generate electricity; half power, where the reactor is being shut down and output is at approximately 50% of capacity, and low power where the reactor is critical and being prepared for startup but power generation is between 0 and 1% capacity. The second initial condition changed for the reactor system involved the stage of the reactor's lifetime. Three different conditions were available for use: beginning of life (BOL), where the reactor is brand new; middle of life (MOL), where the reactor is close to 30 years old; and end of life (EOL), where the reactor is close to decommissioning, 60 years into its operating life. Using these two features, it was possible to collect data on nine different initial condition

combinations, while the reactor is functioning as intended. Each run was conducted for 1200 seconds and data was collected for each of the thirty three measurable features every second during the run.

In addition to collecting data when the reactor is under normal operating conditions, four transient events were selected to perform runs using the nine different initial condition configurations. The transient events selected were: a simultaneous trip of all feed water pumps, a simultaneous closure of all main steam isolation valves (MSIV), a maximum reactor coolant rupture combined with a complete loss of offsite power (LOOP), and a rapid power change, where power drops from 100% to 75% and then increases back up, 100% in a 10 minute period. Runs for the first three transients were done using the nine different configurations, with the transient occurring 20 seconds after the simulation began. Data was collected for 600 seconds after the transient occurred. For the rapid power change, due to the nature of the transient only three runs were done using different core lifetimes. These runs were 1100 seconds long.

After the completion of a run, the data was saved from the reactor simulator to a comma separated values (CSV) file. In total, thirty nine different CSV files were generated and saved. After each run, the reactor core lifetime feature was added to each instant from the dataset for the run using either 'BOL', 'EOL' or 'MOL'. Also, the transient that had occurred was also added to each instant in the dataset. The initial 20 points in the each transient dataset were labeled as ‘Normal Operation’.

In order to create a machine learning model using the data collected from the reactor simulator, modifications were needed. The thirty nine CVS files were combined in a single file. This set consists of 30,710 data points, each consisting of the thirty three measured features, plus

the features added for reactor core life and transient event. Time stamps were not included in the final CSV file. The complete CSV file was imported using Pandas, into a `DataFrame` for easier modifications, prior to use in a machine learning model.

The addition of the reactor core lifetime introduced a qualitative feature into the datasets which machine learning algorithms cannot use. In order to properly account for the reactor lifetime, it was necessary to convert the qualitative data into quantitative data using dummy variables. In this case, since there are three different types of qualitative data (BOL, EOL and MOL), two extra dummy variables were needed. The reactor simulator dataset was split into a training set and a testing set, with half the data in each. In supervised machine learning algorithms, data should be split in order to validate the results.

The models for this project were created using the Tree-based Pipeline Optimization Tool (TPOT) package in Python. TPOT was developed by the Computational Genetics Lab at the University of Pennsylvania. The package is free and open-source. Data used in machine learning models must be preprocessed in order to provide reliable results. Also, there are several different models that can be created. Therefore choosing the best model can be difficult. The purpose of TPOT is to simplify and automate parts of the process, while providing better results due to improved data preprocessing and the use of multiple different methods.

The TPOT package makes use of data preprocessing and model construction functions within the scikit-learn package. TPOT cannot account for qualitative data or missing values and as such, the user must perform data exploration prior to using the package. Preprocessing with TPOT uses nine different preprocessing methods: Binarization, Feature Agglomeration, Maximal Absolute Scaling, Minimum Maximum Scaling, Normalization,

Principal Component Analysis, Robust Scaling, Standard Scaling and RBF kernel sampling. In addition, TPOT uses three selection methods: Select Family Wise Error, Select Percentile and Variance Threshold Selection. The process of data preprocessing and selection is known as a pipeline.

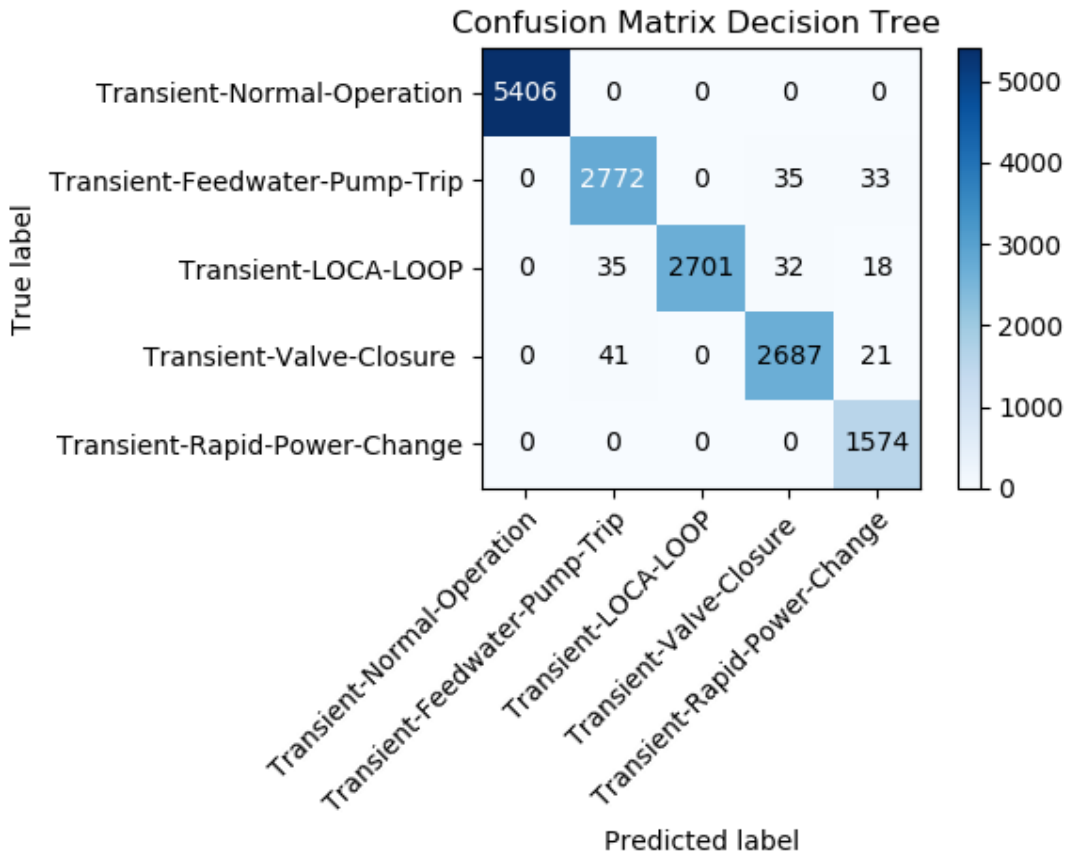
Once the pipeline has been constructed, TPOT will begin creating and validating supervised learning models. TPOT can perform either a regression analysis or classification. For classification, TPOT makes use of six different methods: Gaussian naïve Bayes, Bernoulli naïve Bayes, multinomial naïve Bayes, k-nearest neighbors, decision tree classification and logistic regression. It should be noted that the user can specify which model, selection and preprocessing methods are implemented, using a TPOT dictionary.

The TPOT Classifier also allows the user to define the parameters of the model creation. One important parameter is the number of generations that will be used in the model creation. This is the number of iterations that will be used in the optimization process. Typically, the more generations run, the better the results will be. However, the process will take longer. Another important parameter that can be specified is the population size used. This number is the number of pipelines in each generation. Again, a larger population produces better results, but increases the time needed to complete the model.

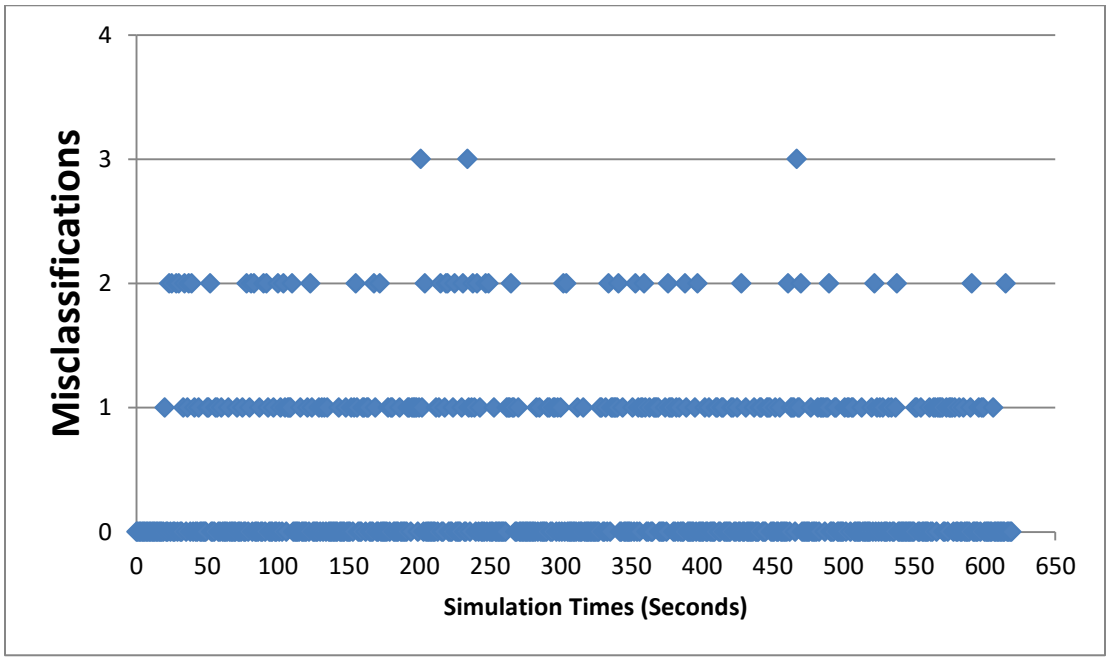
Once the parameters of the classifier have been set by the user, the program will then take the test and train datasets and determine which model is optimal. This process is time consuming and can take days to run, depending on the parameters and dictionaries used. TPOT will output the type of model that is optimal, as well as a single validation measurement.

Models were created using all six of the TPOT Classifiers. The parameters for these models include: 100 generations, a population size of 100. A dictionary was defined for both models, including all nine preprocessing methods and the three feature selection methods. To further evaluate the effectiveness of the model, the following measurements were used: accuracy, precision, recall and F1 score using scikit-learn.

The results from the classification are good. The models were able to determine transient and non-transient behavior perfectly, as there were no false positives or negatives with the normal operations data. While there were errors, the model did well in distinguishing the different transients. All models scored in the mid-90s on all overall validation measurements and over 88% on all individual transient accuracies. The table below summarizes the results from this project. The confusion matrix and misclassification graph for the decision tree model are also given below. These results indicate that these techniques could be useful tools for assisting reactor operators in order to diagnose transients at power plants. This could mitigate or prevent reactor damage and help with overall perception of nuclear power.



Confusion Matrix for Decision Tree Model



Graph of Misclassifications for Decision Tree Model

Summary of Machine Learning Model Results										
Overall Validation Measurements					Individual Transient Accuracies					Time
	Accuracy	Precision	Recall	F1 Score	Normal Operation Transient	Feed Water Pump Trip	LOCA + LOOP	Valve Closure	Rapid Power Change	Time Required (Approx.)
K-Nearest Neighbors	98.35%	98.02%,	98.01%	98.01%	100%	96.6%	97.57%	98.01%	97.86%	1.5 Hours
Bernoulli Naïve Bayes	97.45%,	97.18%,	96.73 %	96.87%.	100%	90.05%	100%	96.1%	93.72%	1 Hour
Gaussian Naïve Bayes	97.45%	97.2%	96.83%	96.96%	100%	88.77%	100%	96.59%	94.0%	1 Hour
Multinomial Naïve Bayes	96.71%.	96.38%,	95.41%	96.10%.	100%	90.05%	100%	96.1%	93.72%	1 Hour
Logistic Regression	98.55%.	98.41%,	98.04%	98.21%	100%	96.87%	100%	97.62%	95.71%	48 Hours
Decision Tree Analysis	98.6%	98.46%,	98.1%	98.27%.	100%	97.33%	100%	97.57%	95.62%	3 Hours

References

1. B. Marr “The Amazing Ways Coca Cola Uses Artificial Intelligence And Big Data To Drive Success”, Forbes, September 18, 2017. https://www.forbes.com/sites/bernardmarr/2017/09/18/the-amazing-ways-coca-cola-uses-artificial-intelligence-ai-and-big-data-to-drive-success/#7fbeecfe78d2

2. ”System uses ‘deep learning’ to detect cracks in nuclear reactors”, Purdue University, November 6, 2017. https://www.purdue.edu/newsroom/releases/2017/Q4/system-uses-deep-learning-to-detect-cracks-in-nuclear-reactors.html

3. D. GALEON, “How Artificial Intelligence Is Making Nuclear Reactors Safer”, Futurism, November 23, 2017. https://futurism.com/researchers-training-ai-make-nuclear-reactors-safer

4. R. GRONBERG, “AI could one day control nuclear reactors; NC State researchers could make it happen”, News Observer, June 6, 2018. https://www.newsobserver.com/news/local/education/article212560909.html

5. “Modeling-Enhanced Innovations Trailblazing Nuclear Energy Reinvigoration (MEITNER)”, ARPA-E/Department of Energy, June 4, 2018.

6. S. MILLION-WEAVER “Eagle-eyed machine learning algorithm outdoes human experts”, University of Wisconsin Madison, July19th, 2018. https://news.wisc.edu/eagle-eyed-machine-learning-algorithm-outdoes-human-experts/

7. ”Reactor analytics drives nuclear industry towards machine learning”, Nuclear Energy Insider, November 15, 2017. https://analysis.nuclearenergyinsider.com/reactor-analytics-drives-nuclear-industry-towards-machine-learning

8. H. VELLA, “What Will the Power Plant of the Future Look Like?”, General Electric April 26, 2018. https://www.ge.com/power/transform/article.transform.articles.2018.apr.the-power-plant-of-the-future#

9. N. NILSSON, “Introduction to Machine Learning”, Stanford University November 3, 1998.

10. E. Learned-Miller, “Introduction to Supervised Learning”, University of Massachusetts, February 17, 2014.

11. A. Ramanujam, “Python at Netflix”. Netflix, April 28, 2019. https://medium.com/netflix-techblog/python-at-netflix-bba45dae649e

12. T. Oliphant, “Guide to NumPy”. December 7 2006.

13. “Array Objects”, SciPy Community. June 4 2019. https://www.numpy.org/devdocs/reference/arrays.html

14. W. McKinney, “pandas: powerful Python data analysis toolkit Release 0.24.2.” March 13 2019.

15. W. McKinney, “pandas: a Foundational Python Library for Data Analysis and Statistics”.

16. “About Us”, Scikit-learn Developers. Scikit-learn.org. https://scikit-learn.org/stable/about.html#people

17 “Who is using scikit-learn?”, Scikit-learn Developers. Scikit-learn.org https://scikit-learn.org/stable/testimonials/testimonials.html

18. Pedregosa et al, scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830, 2011.

19. “Scikit-Learn User Guide Release 0.21.2”. Scikit-Learn, May 24, 2019.

20. “Cloud AutoML”. Alphabet. https://cloud.google.com/automl/docs/

21. “Home-TPOT”, Epistassislabs. https://epistasislab.github.io/tpot/.

22. R. Olson, N. Bartley, R. Urbanowicz, & Moore (2016). “Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science”. GECCO 2016, pages 485-492.

23. E. Smith, L. Likforman-Sulem, J. Darbon, “Effect of Pre-Processing on Binarization”. Boise State University, January 1, 2010.

24. “5.3. Preprocessing data”, Scikit-learn Developers. https://scikit-learn.org/stable/modules/preprocessing.html#.

25. “Compare the effect of different scalers on data with outliers”, Scikit-learn Developers. https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py

26. A. Rahimi, B. Recht, “Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning.” UC Berkeley.

27. “sklearn.cluster.FeatureAgglomeration”, Scikit-learn Developers. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.FeatureAgglomeration.html

28. “2.3. Clustering”, Scikit-learn Developers. https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering

29. L. Smith, “A Tutorial on Principal Component Analysis”. University of Montreal, February 26th 2002.

30. “2.5. Decomposing signals in components”, Scikit-learn Developers. https://scikit-learn.org/stable/modules/decomposition.html#

31. P. Breheny, ”Family Wise Error Rates”. University of Iowa, January 25th, 2016.

32. “1.13. Feature Selection”, Scikit-learn Developers. https://scikit-learn.org/ stable/modules /feature_selection.html#

33. “1.9. Naive Bayes”, Scikit-learn Developers. https://scikit-learn.org/stable/modules/naive_bayes.html

34. H. Zhang, “The Optimality of Naïve Bayes”. University of New Brunswick

35. J. Eberhardt, “Bayesian Spam Detection”. University of Minnesota Morris.

36.P. Rai. “Supervised Learning: K-Nearest Neighbors and Decision Trees”. University of Utah, August 25, 2011.

37. “1.6. Nearest Neighbors”, Scikit-learn Developers. https://scikitlearn.org/stable/ modules/neighbors.html#

38. T. Mitchel, “GENERATIVE AND DISCRIMINATIVE CLASSIFIERS: NAIVE BAYES AND LOGISTIC REGRESSION” , Carnegie Mellon/McGraw Hill, September 23 2017.

39. “1.1. Generalized Linear Models”. Scikit-learn Developers, https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

40. R. Mitchel, “Decision Tree Learning”. Princeton University.

41. “1.10. Decision Trees”, Scikit-learn Developers. https://scikit-learn.org/stable/ modules/tree.html

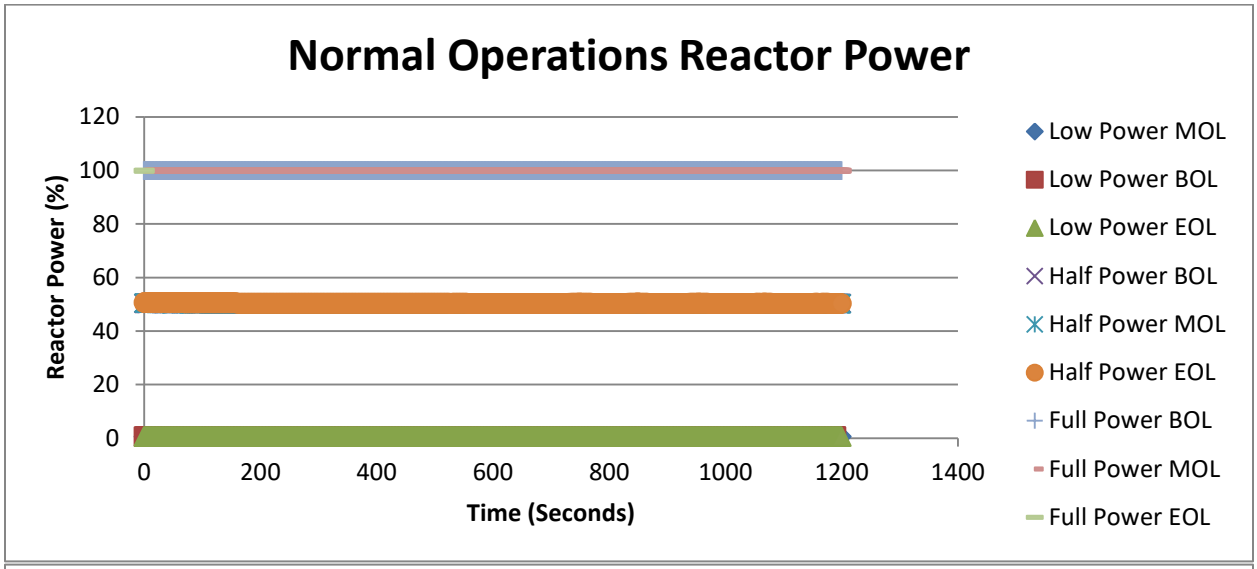
42. “1.11. Ensemble methods”, Scikit-learn Developers. https://scikit-learn.org/stable/modules /ensemble.html#forest

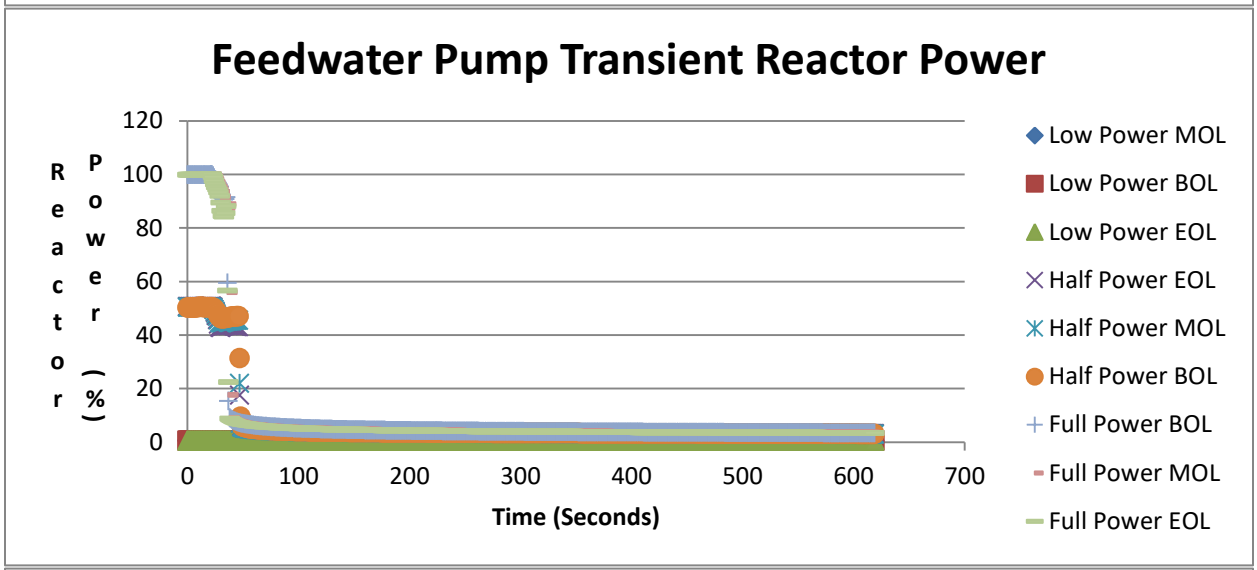
43. “3.3. Model evaluation: quantifying the quality of predictions”, Scikit-learn Developers, https://scikit-learn.org/stable/modules/model_evaluation.html#

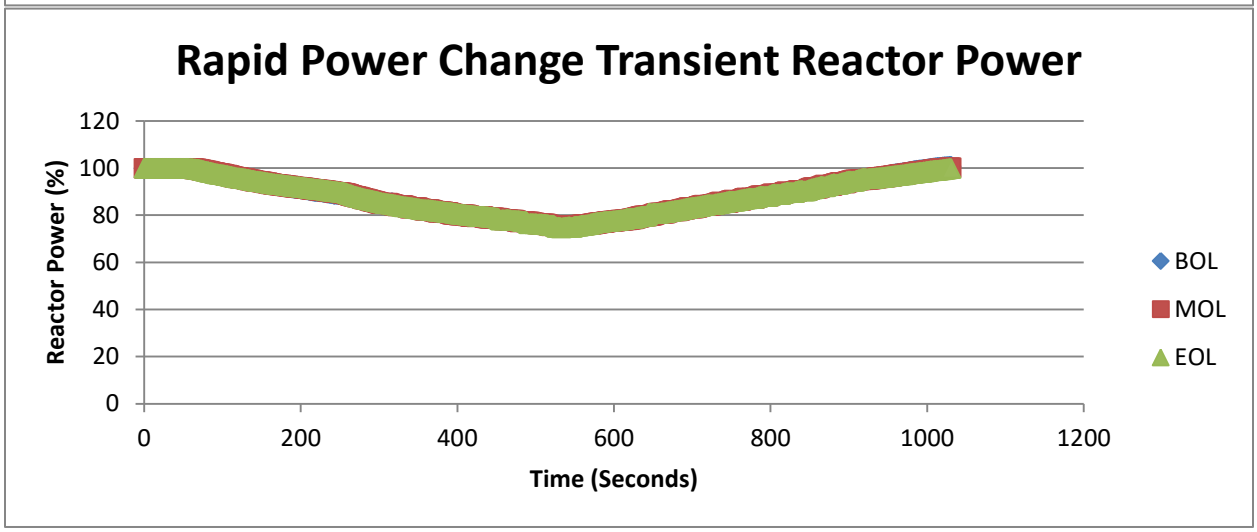
44. “GENERIC PWR SIMULATOR Training Guide”, WCS, May 2017

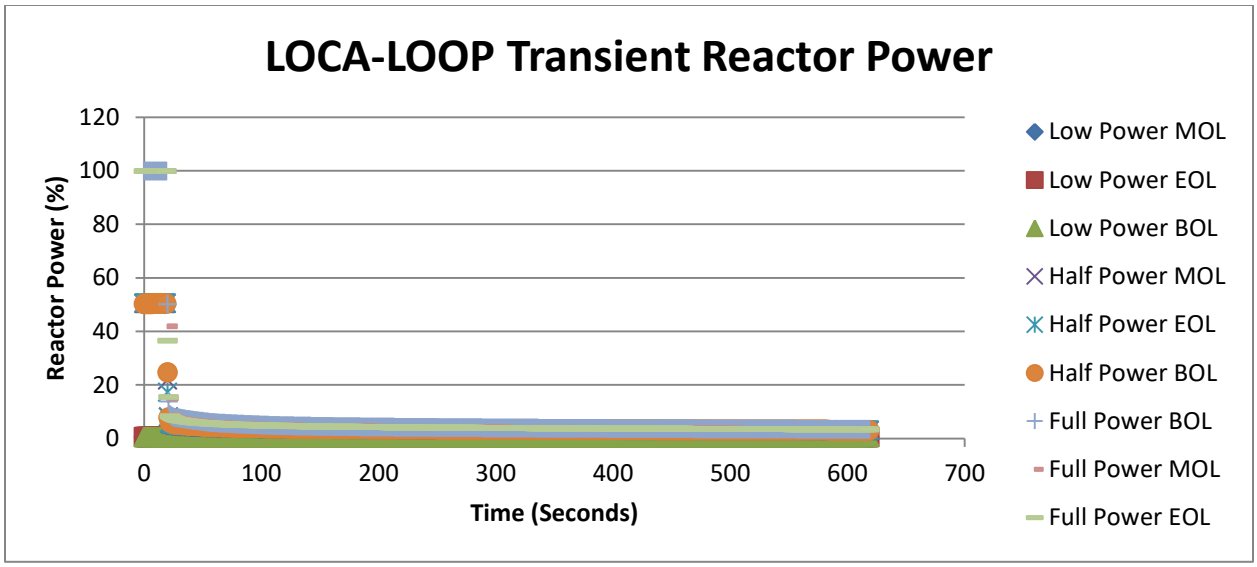
45. “GENERIC PWR SIMULATOR Major Transients Report”, WCS, June 2014

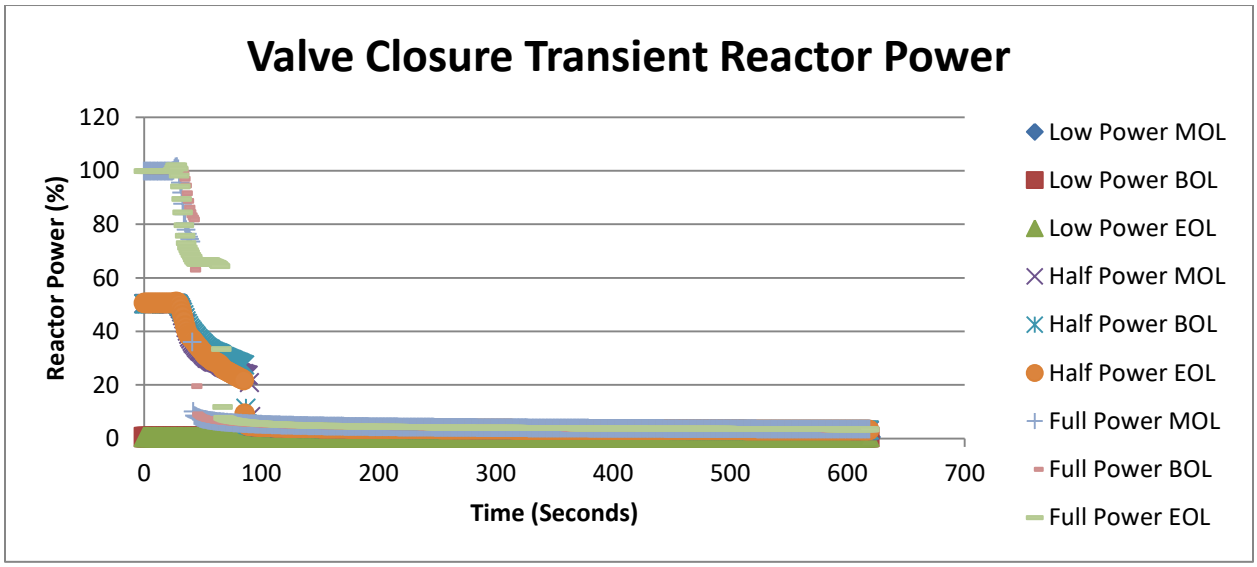
Appendix 1.1 Reactor Power Behavior



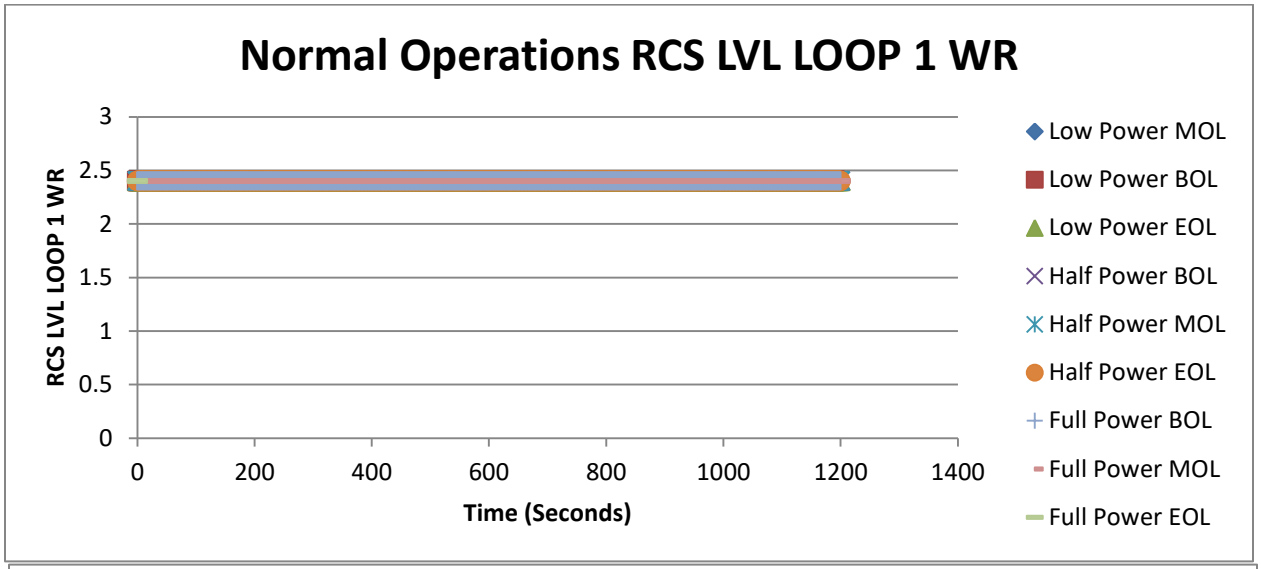


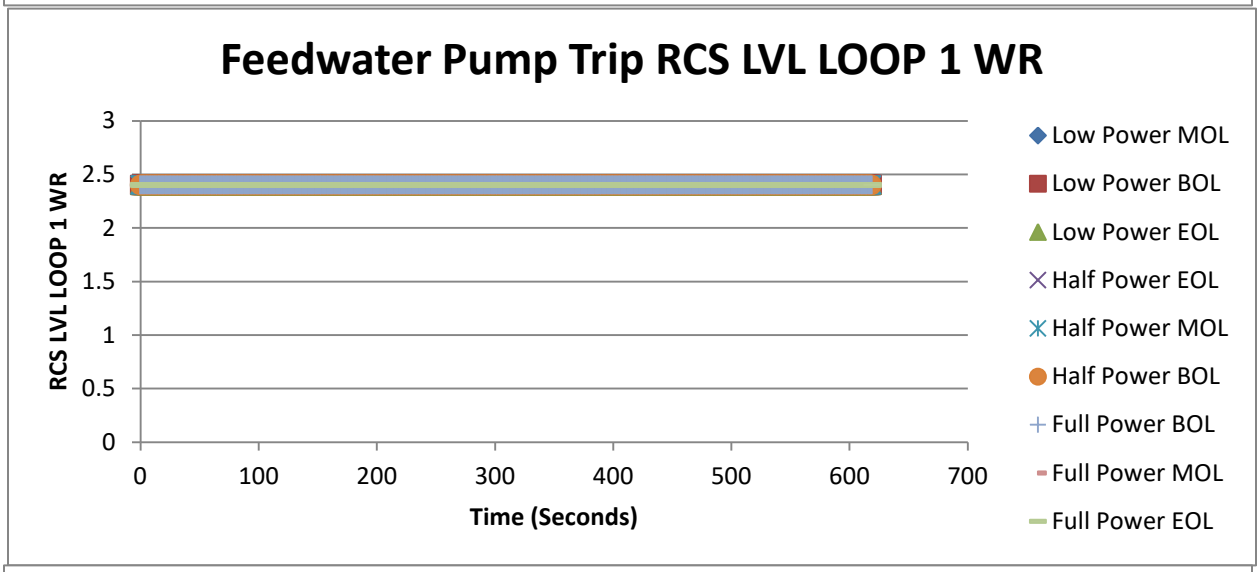


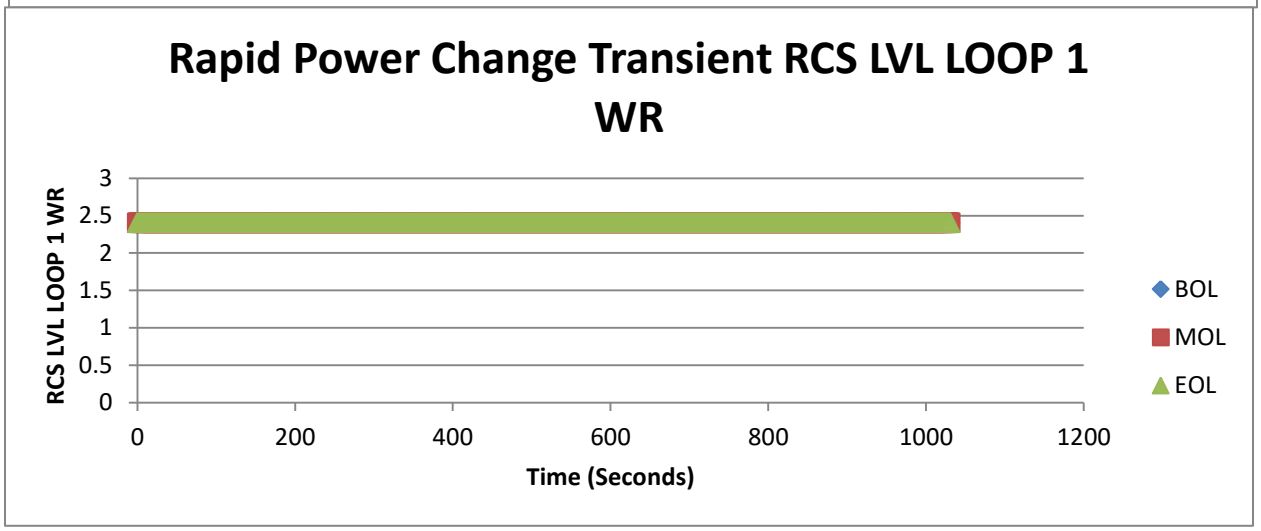

LOCA-LOOP Transient Reactor Power
Reactor Power (%)
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power EOL
Low Power BOL
Half Power MOL
Half Power EOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

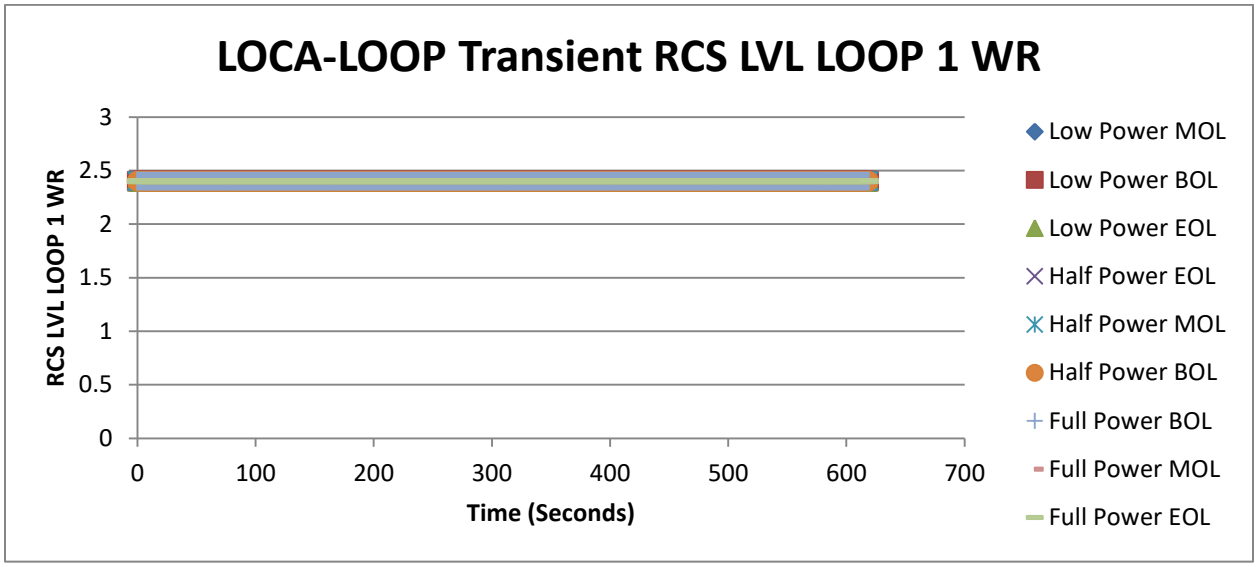

Valve Closure Transient Reactor Power
Reactor Power (%)
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

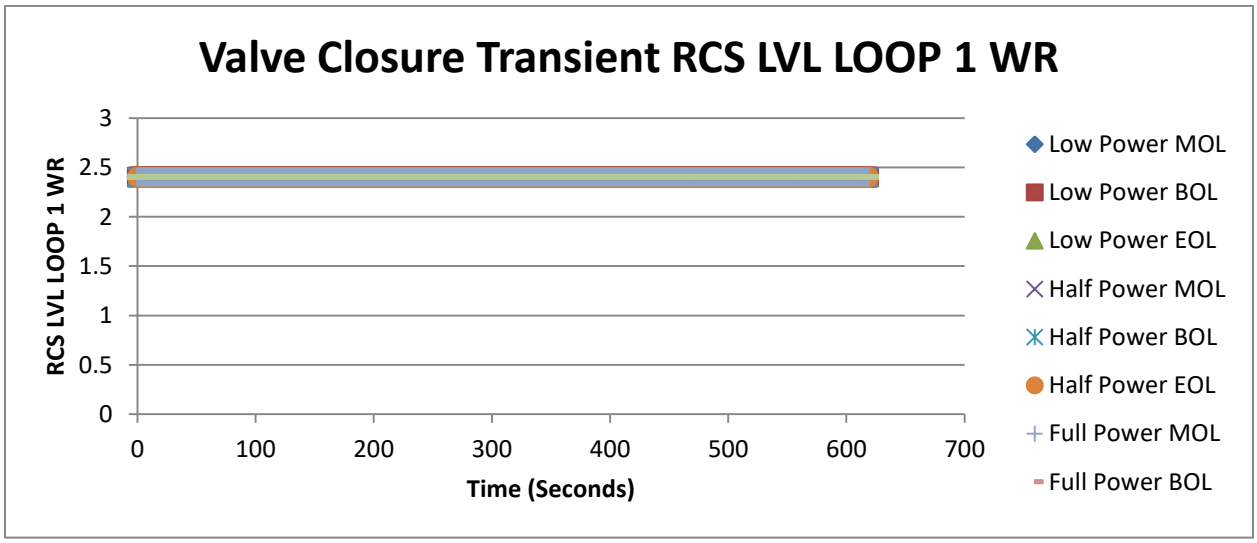
Appendix 1.2 RCS LVL LOOP 1 WR



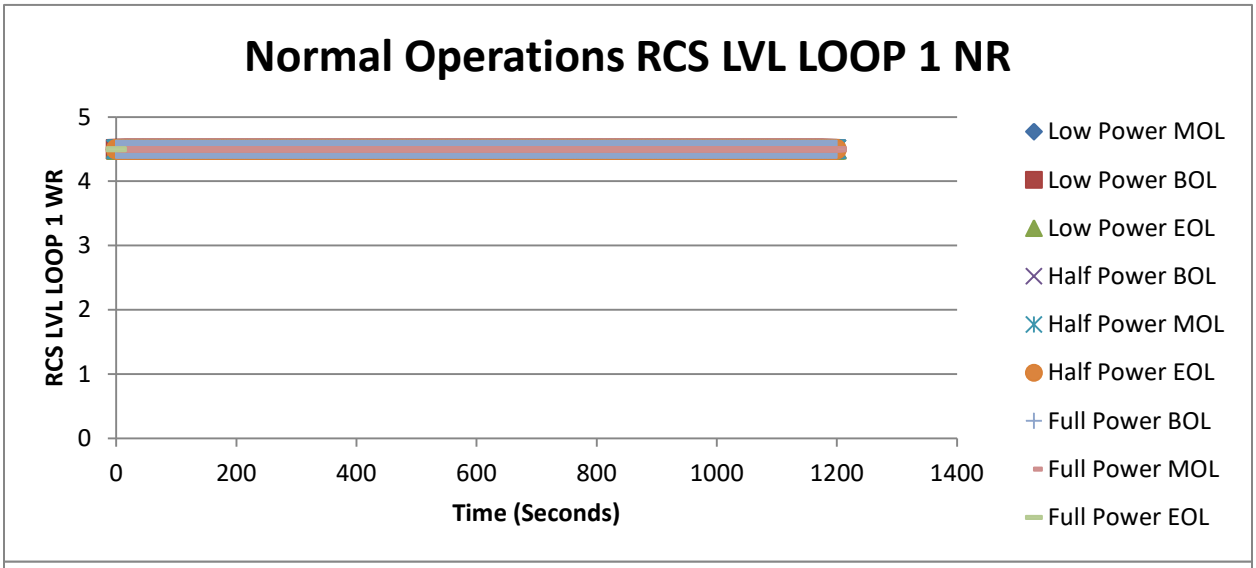


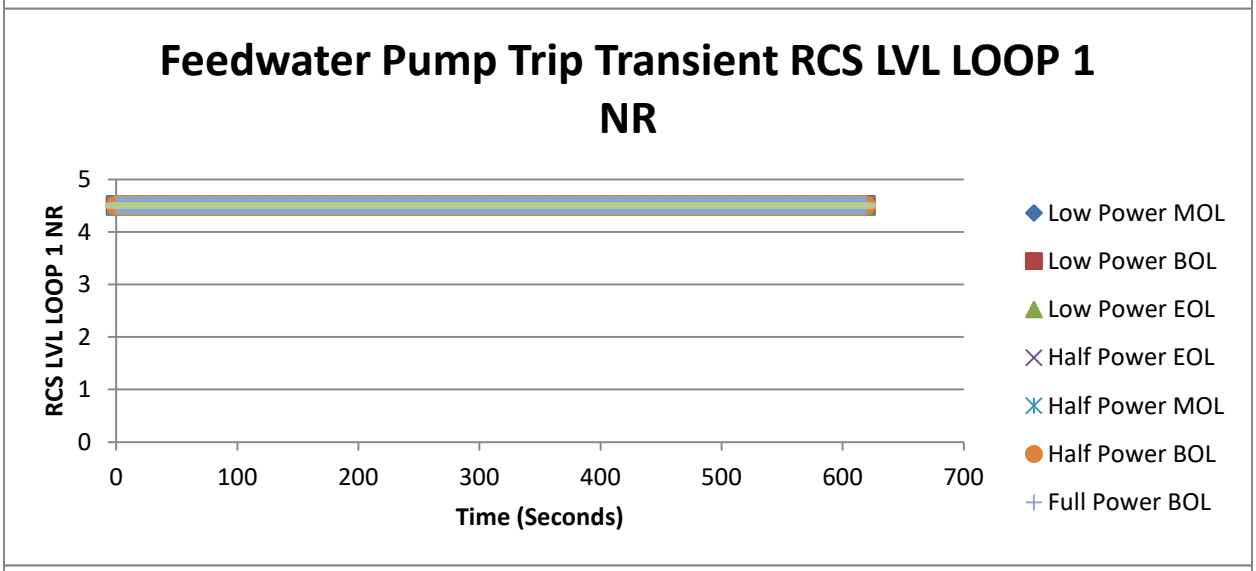


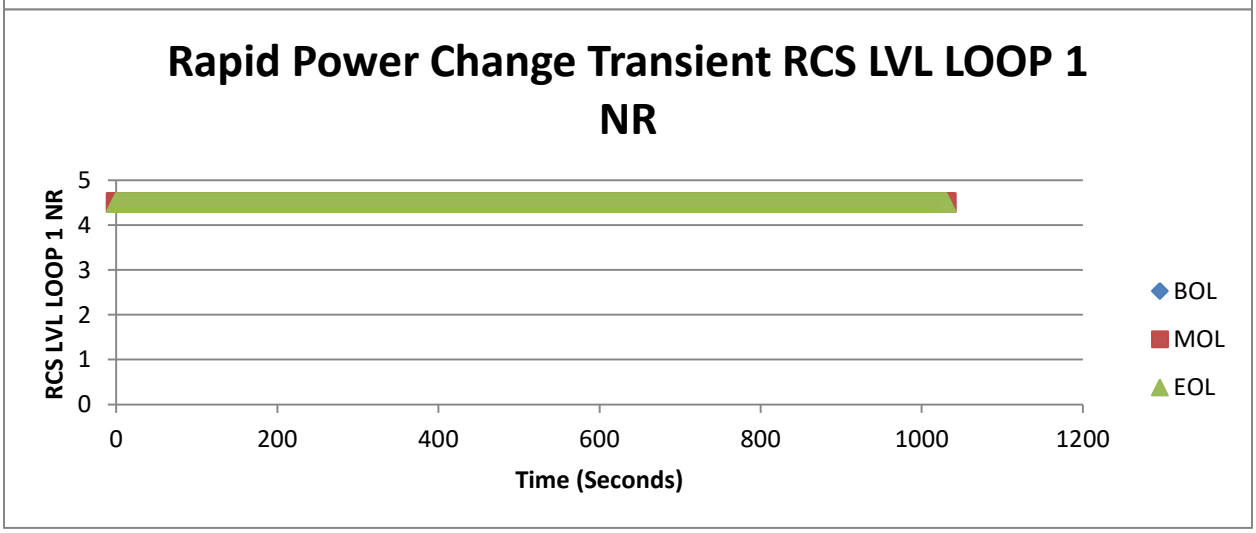

LOCA-LOOP Transient RCS LVL LOOP 1 WR
RCS LVL LOOP 1 WR
3
2.5
2
1.5
1
0.5
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

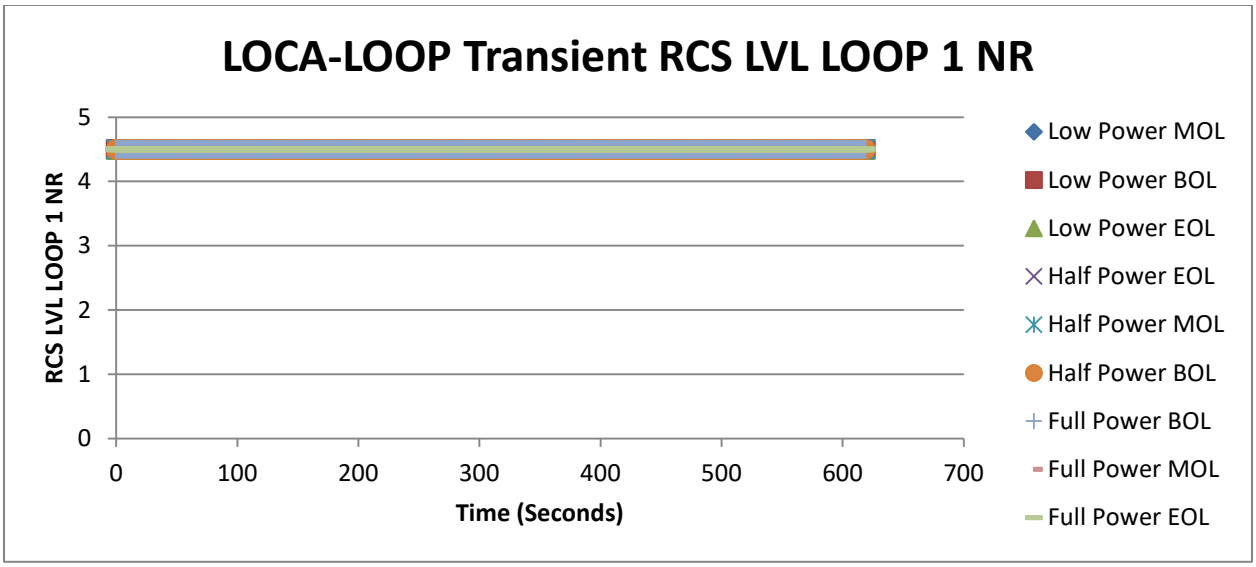

Valve Closure Transient RCS LVL LOOP 1 WR
RCS LVL LOOP 1 WR
3
2.5
2
1.5
1
0.5
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL

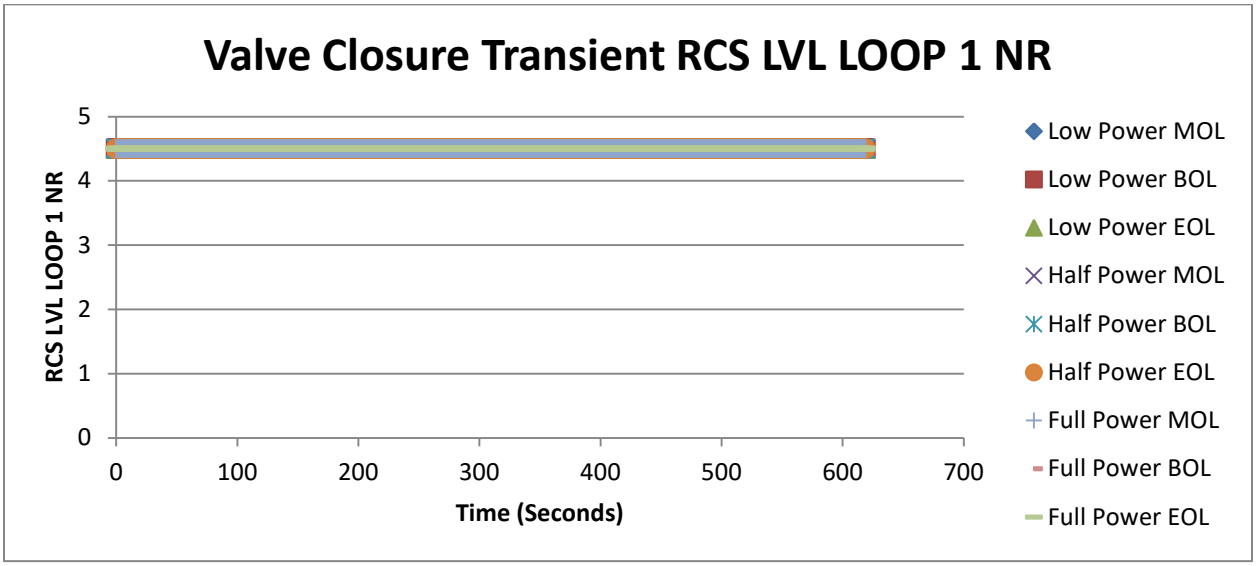
Appendix 1.3 RCS LVL LOOP 1 NR



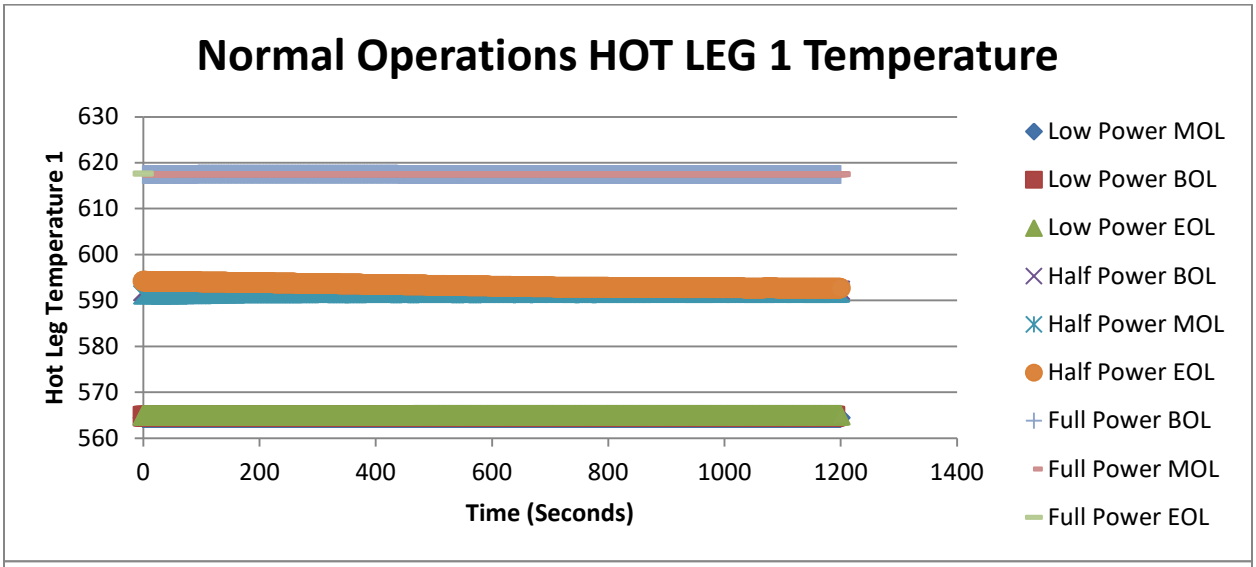


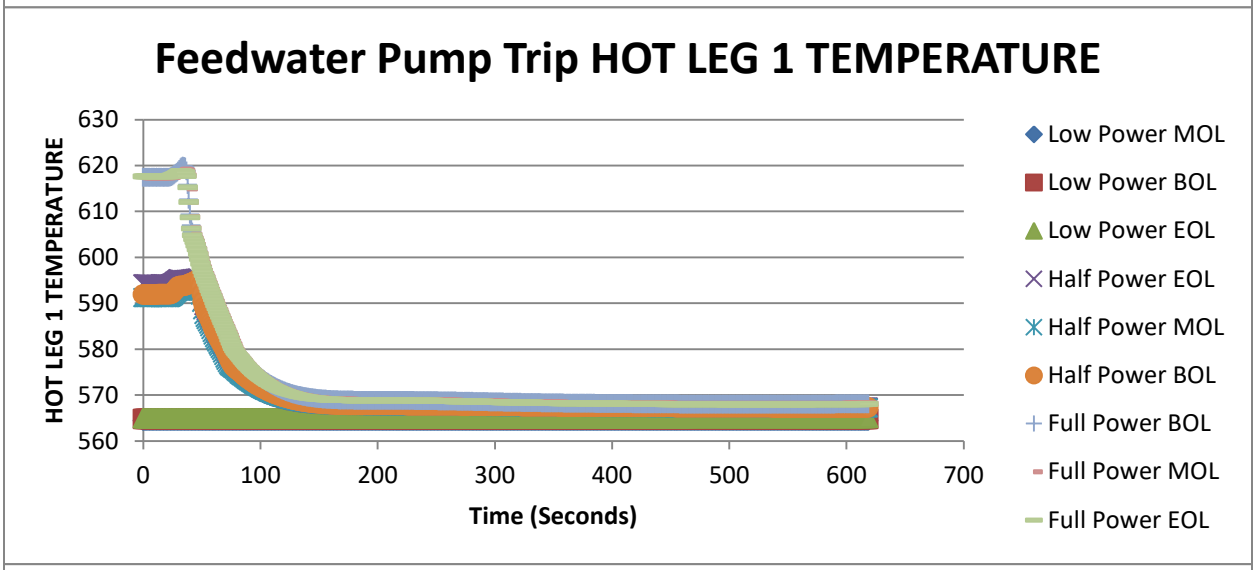


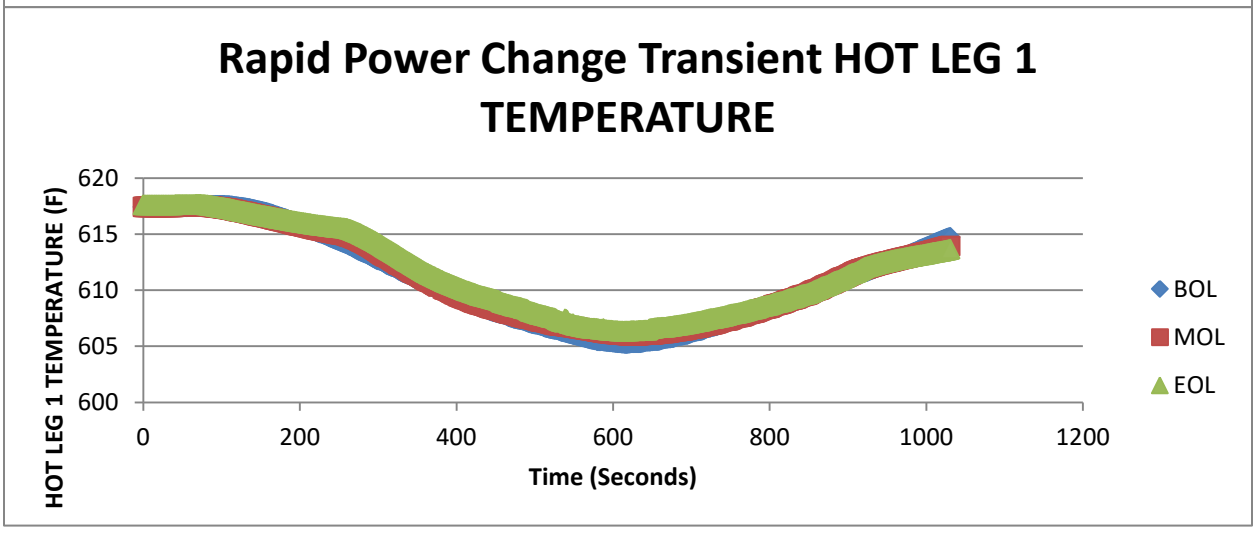

LOCA-LOOP Transient RCS LVL LOOP 1 NR
RCS LVL LOOP 1 NR
0
1
2
3
4
5
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

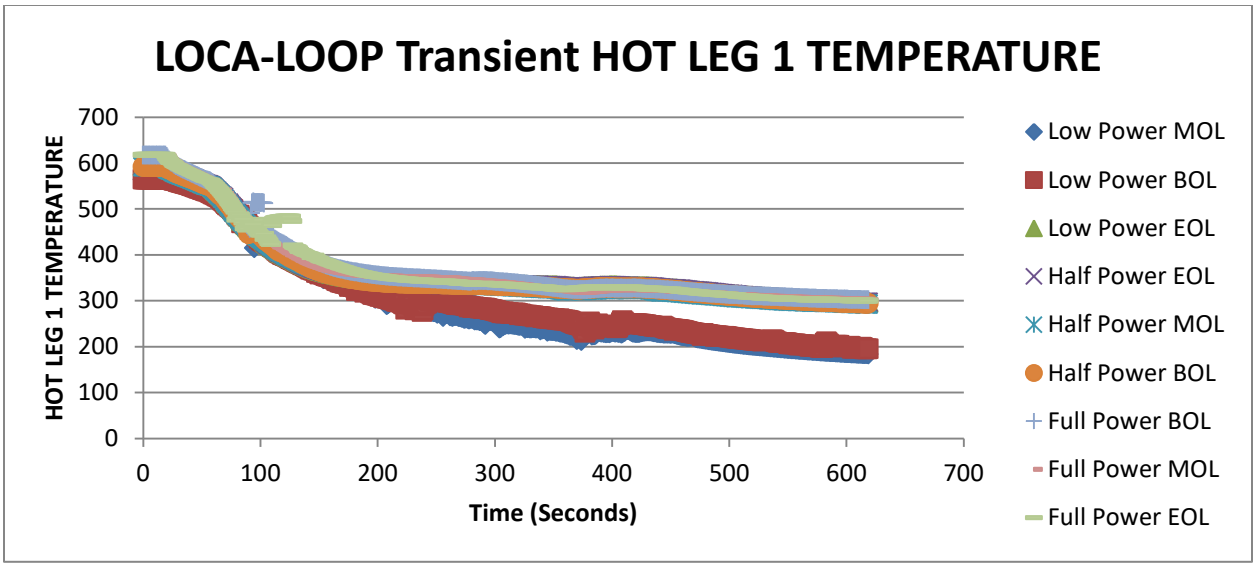

Valve Closure Transient RCS LVL LOOP 1 NR
RCS LVL LOOP 1 NR
0
1
2
3
4
5
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

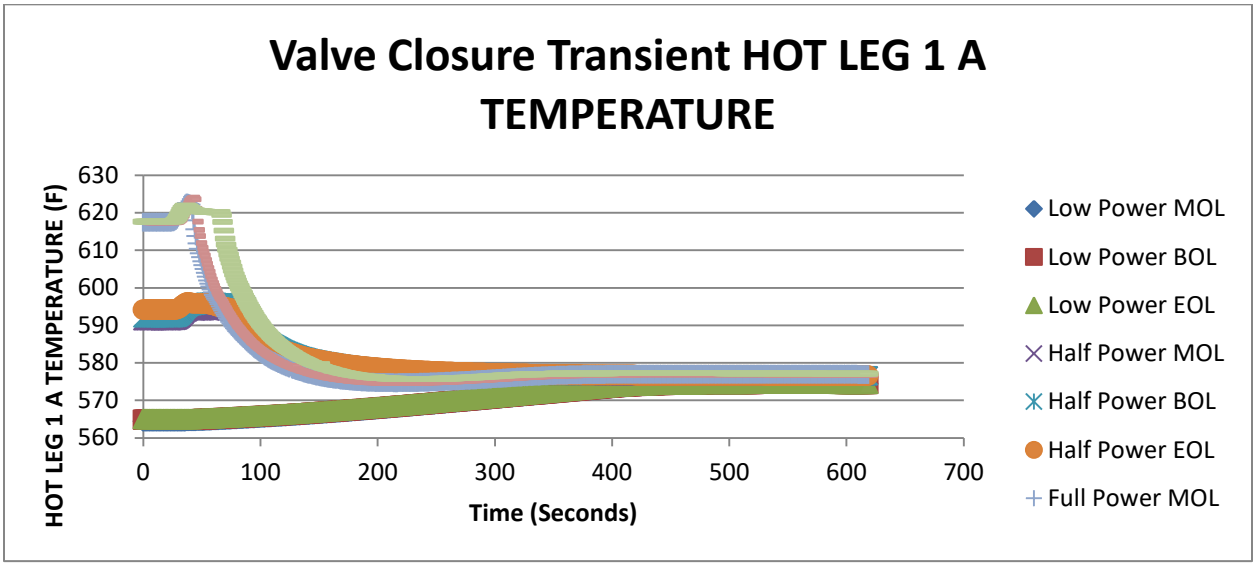
Appendix 1.4 HOT LEG 1 Temperature



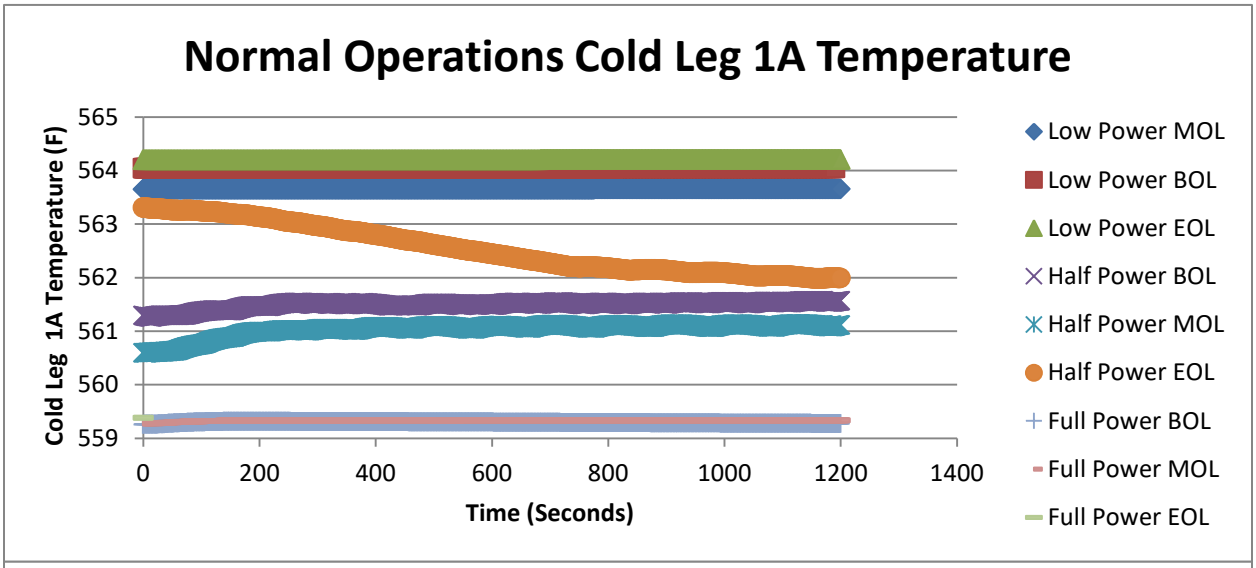


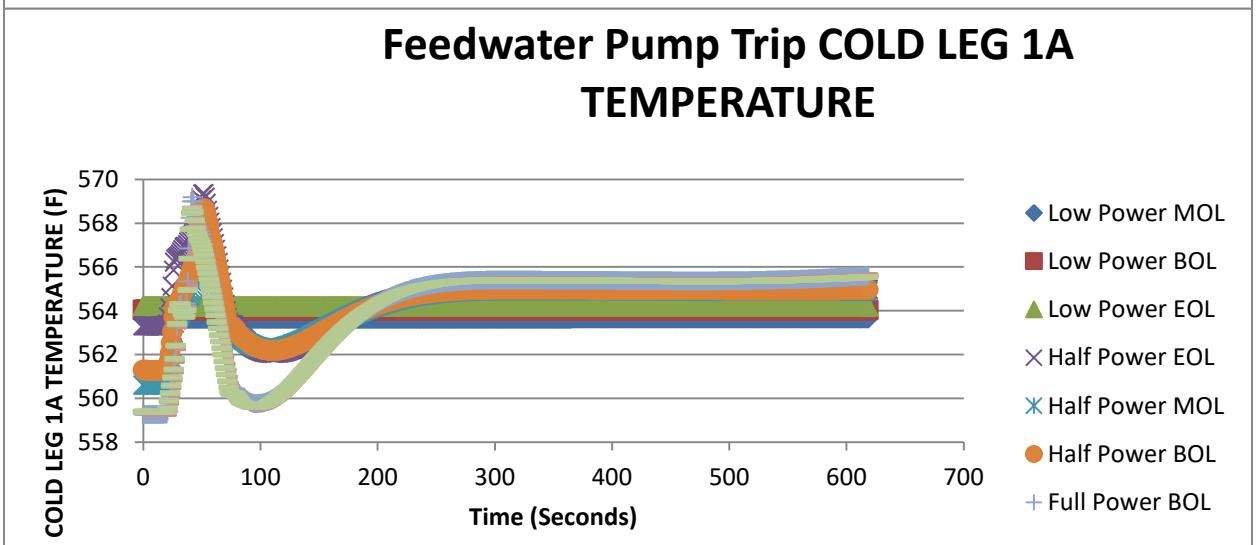


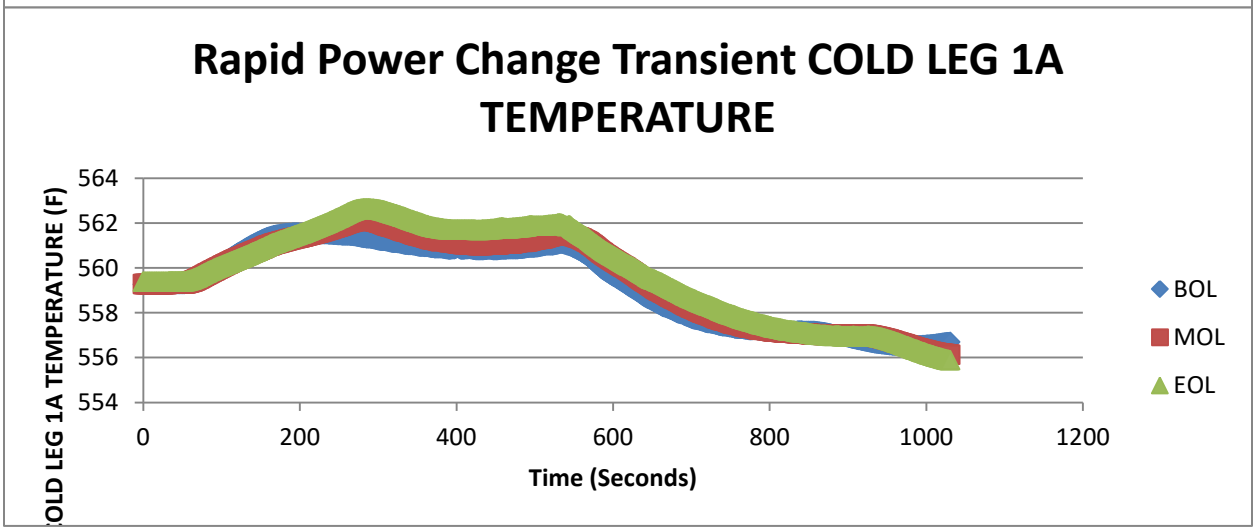

LOCA-LOOP Transient HOT LEG 1 TEMPERATURE
HOT LEG 1 TEMPERATURE
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

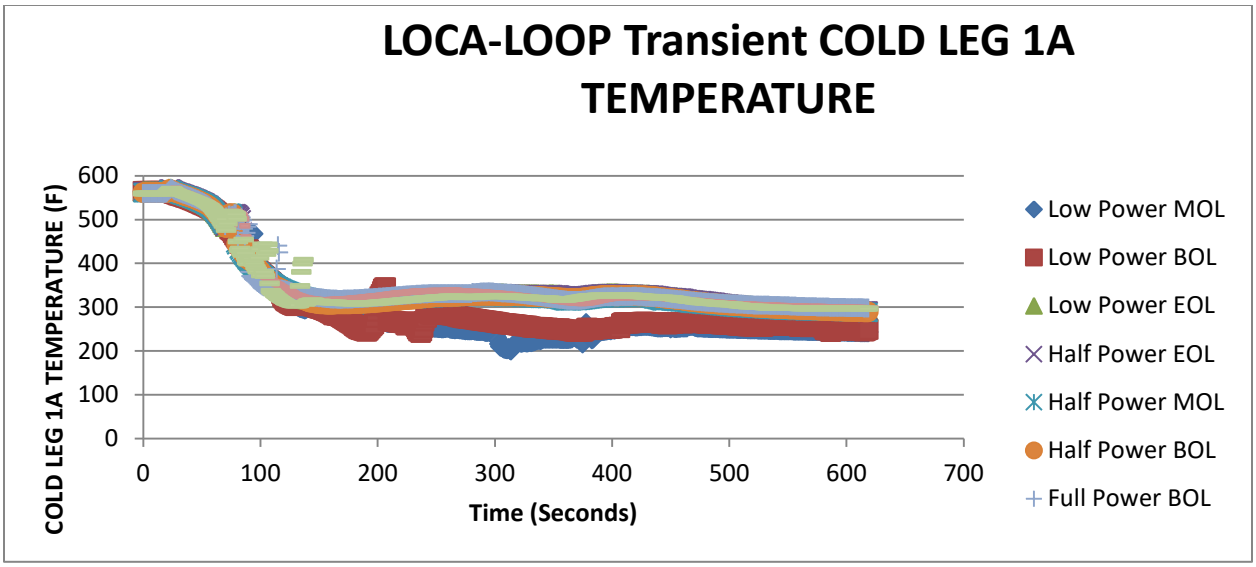

Valve Closure Transient HOT LEG 1 A TEMPERATURE
HOT LEG 1 A TEMPERATURE (F)
630
620
610
600
590
580
570
560
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL

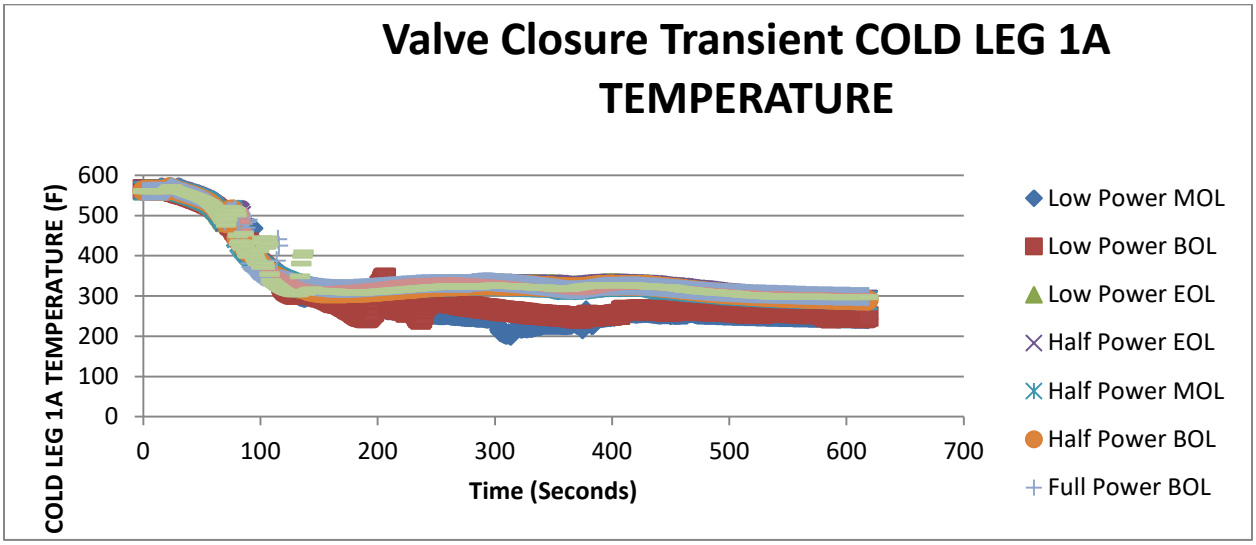
Appendix 1.5 HOT LEG 1A Temperature



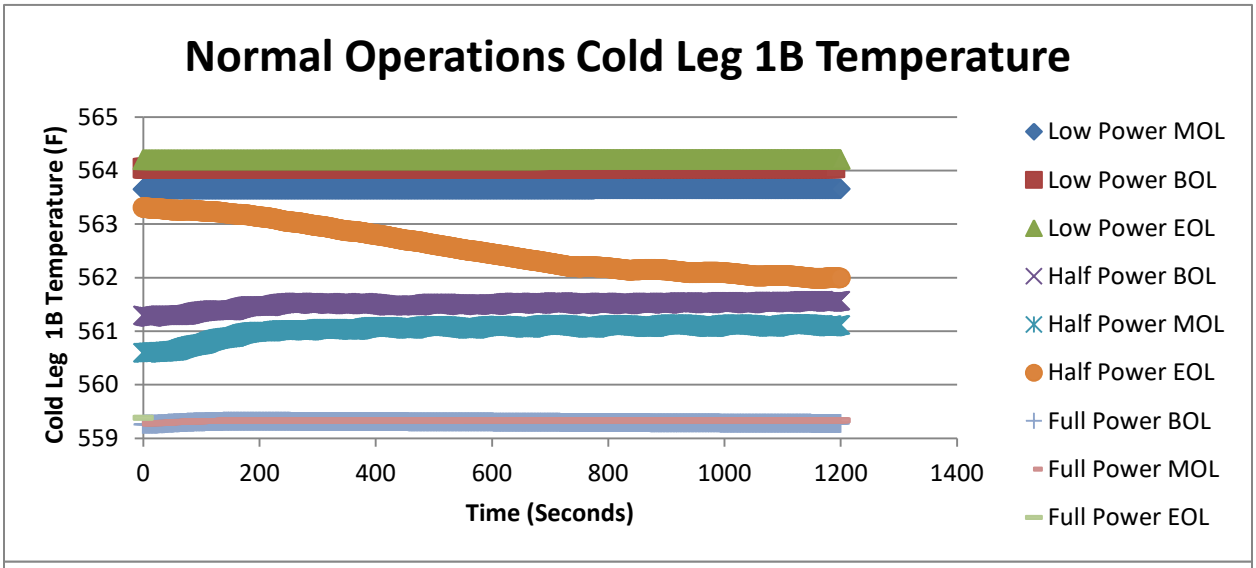


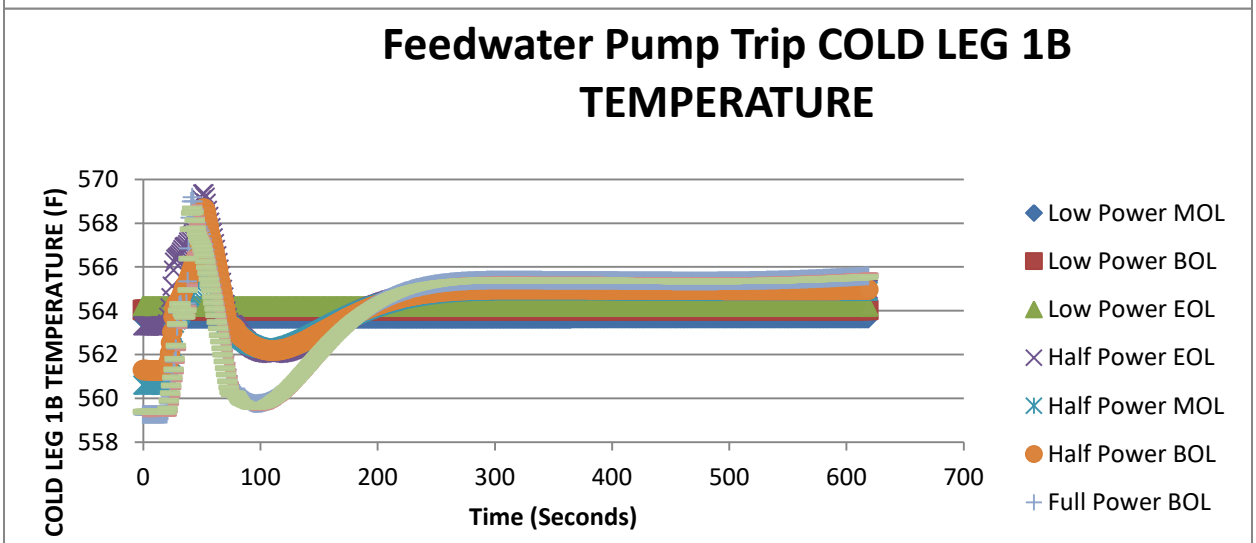


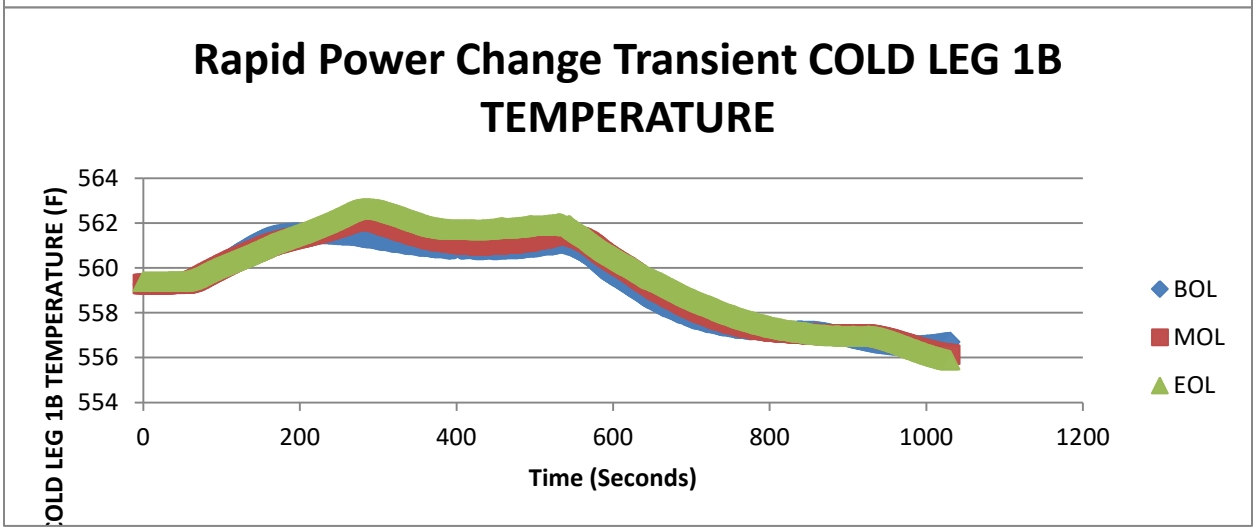

LOCA-LOOP Transient COLD LEG 1A TEMPERATURE
COLD LEG 1A TEMPERATURE (F)
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL

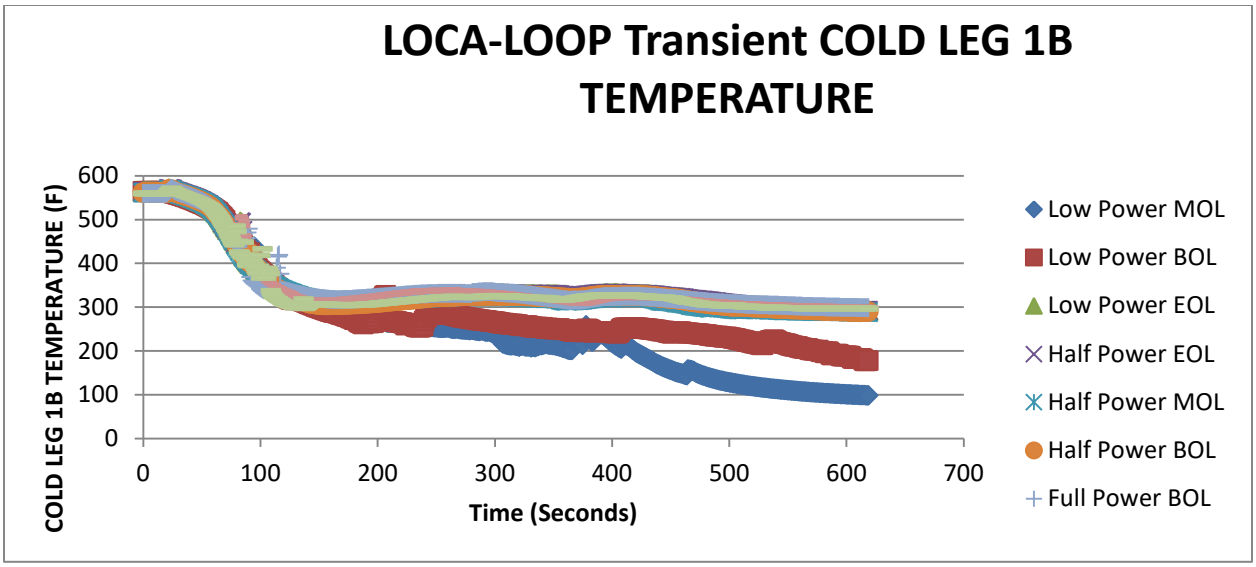

Valve Closure Transient COLD LEG 1A TEMPERATURE
COLD LEG 1A TEMPERATURE (F)
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL

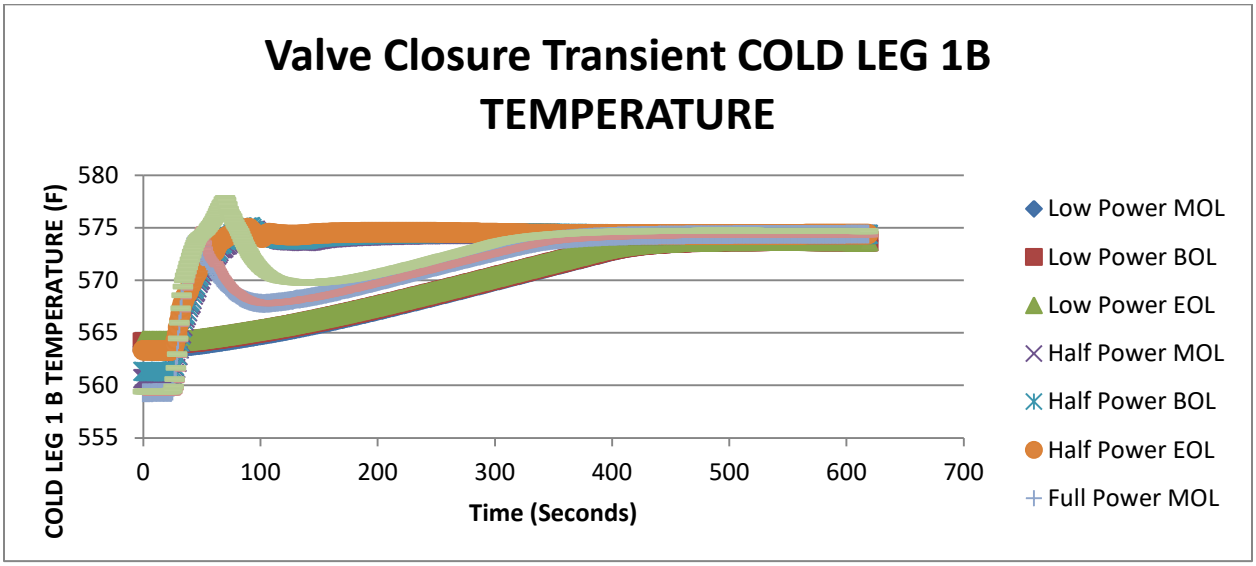
Appendix 1.6 Cold Leg 1B Temperature



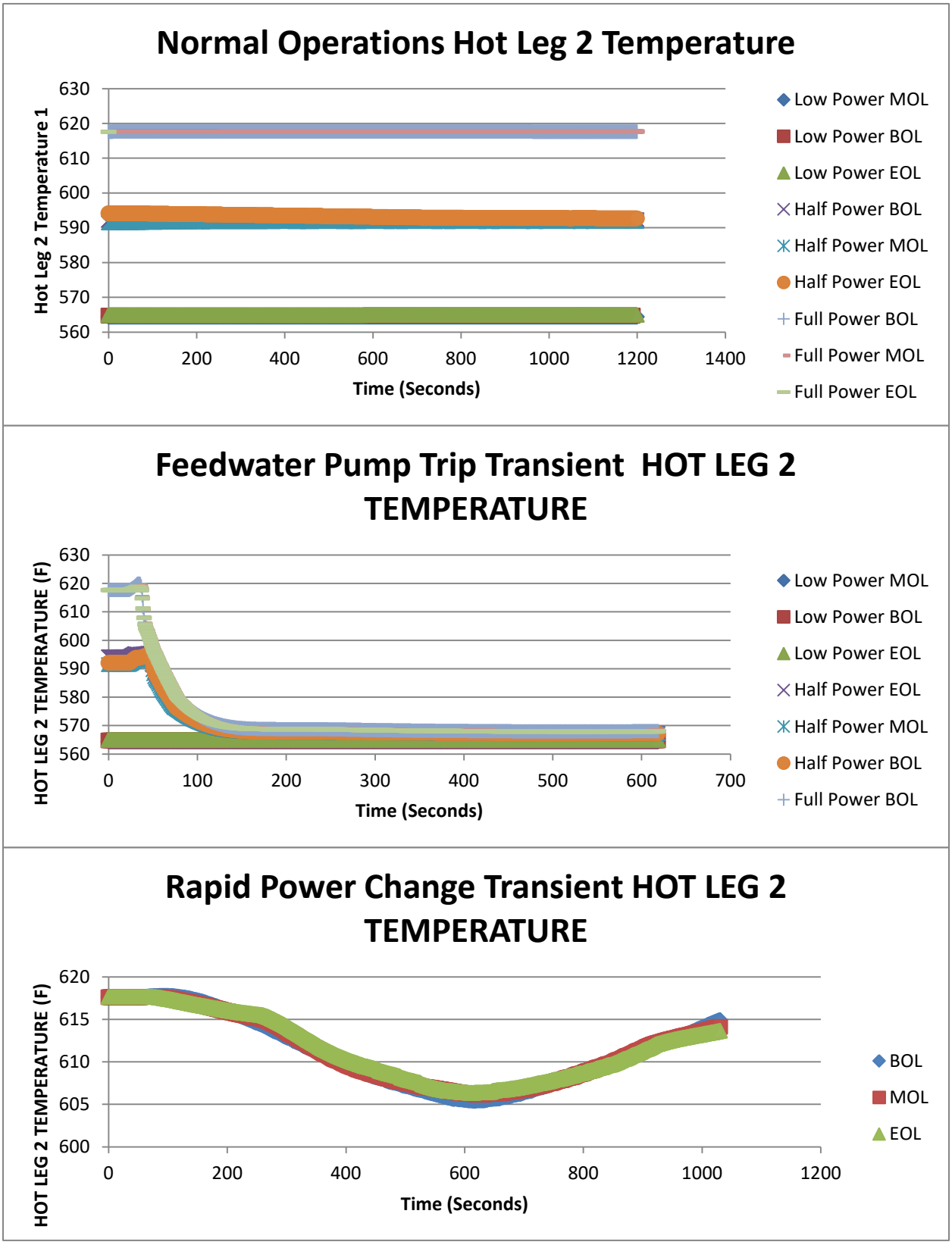


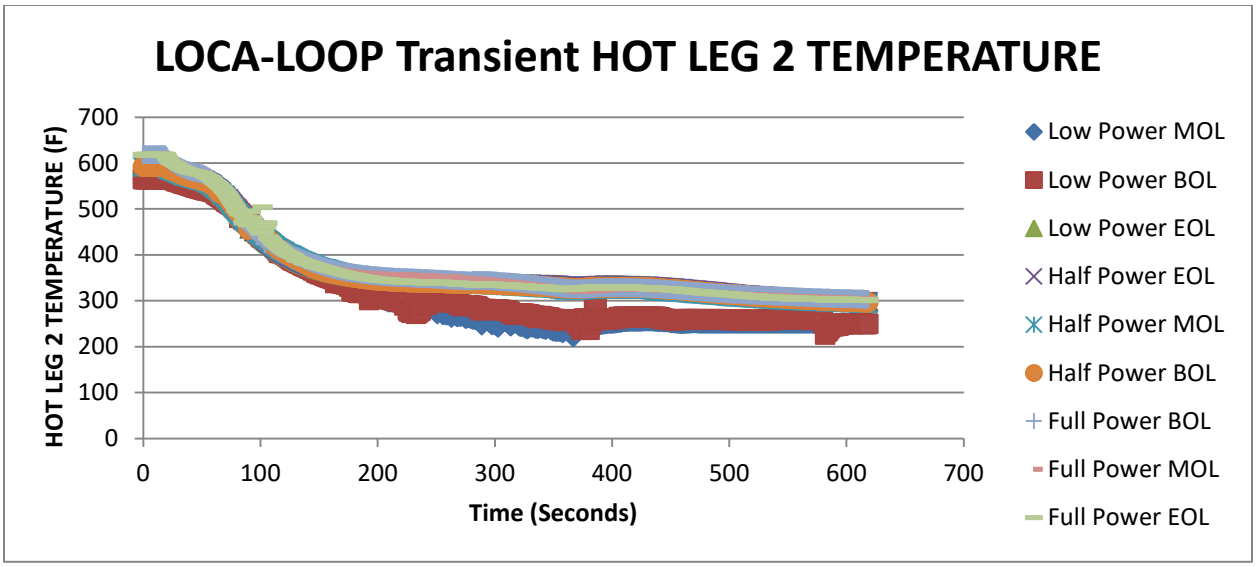


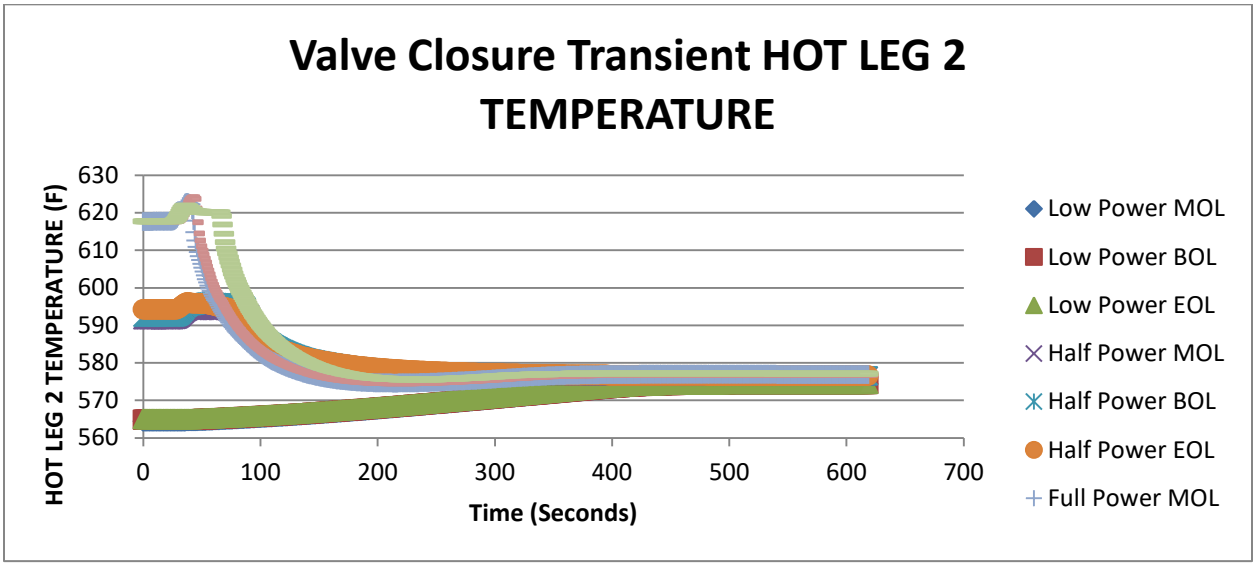

LOCA-LOOP Transient COLD LEG 1B TEMPERATURE
COLD LEG 1B TEMPERATURE (F)
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL


Valve Closure Transient COLD LEG 1B TEMPERATURE
COLD LEG 1 B TEMPERATURE (F)
580
575
570
565
560
555
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL

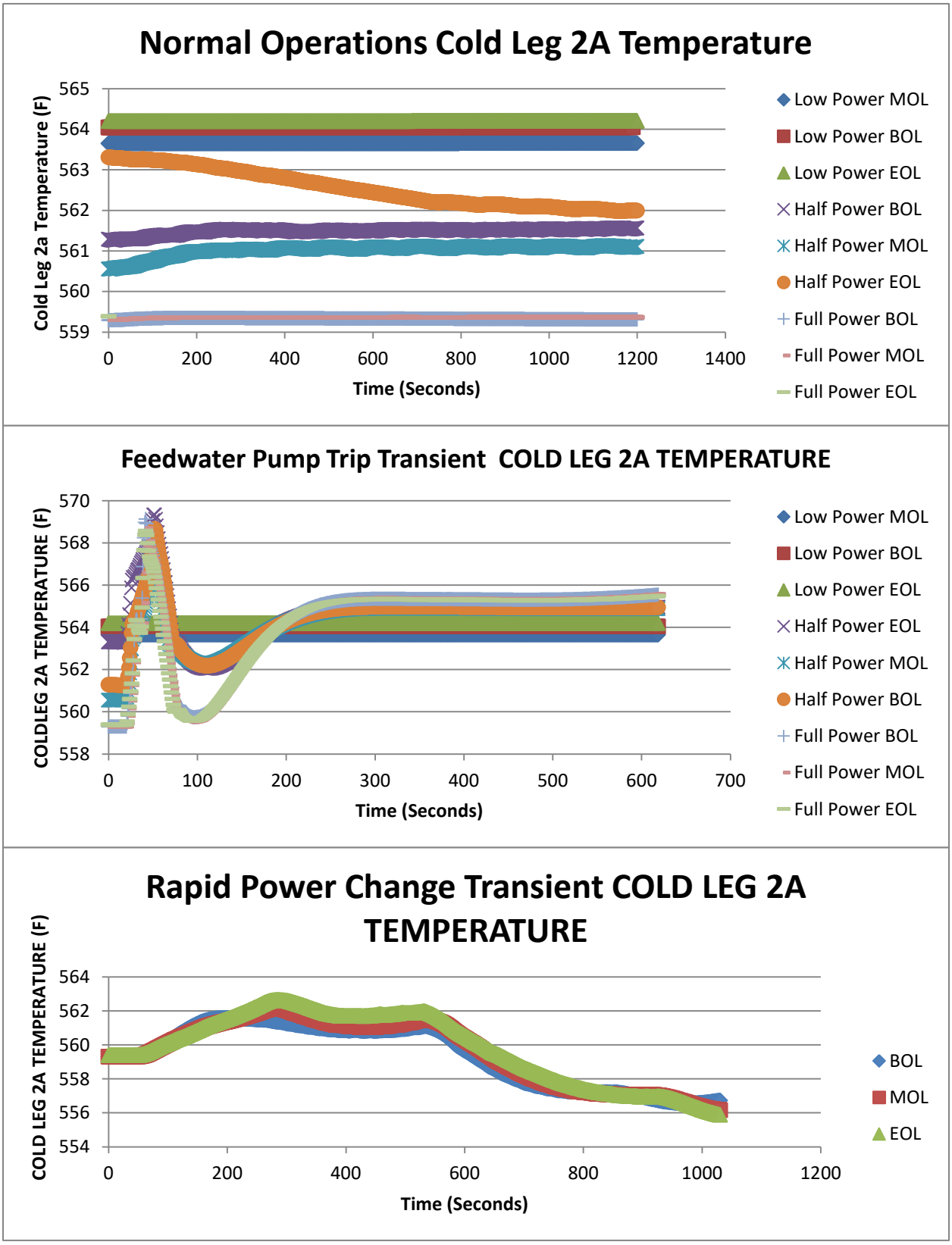
Appendix 1.7 Hot Leg 2 Temperature

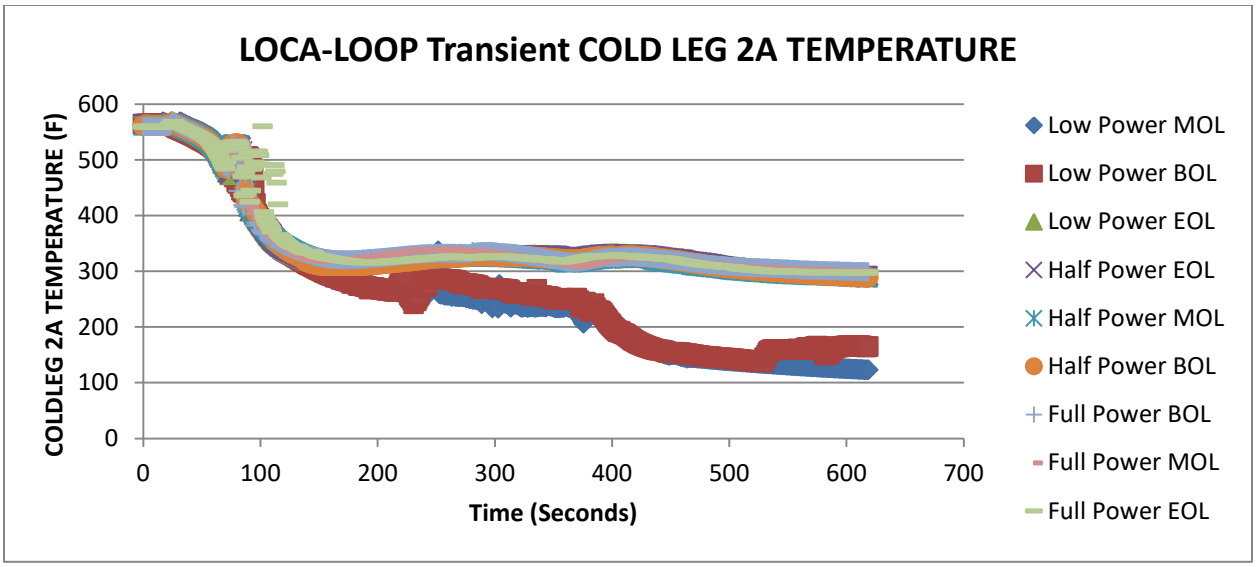


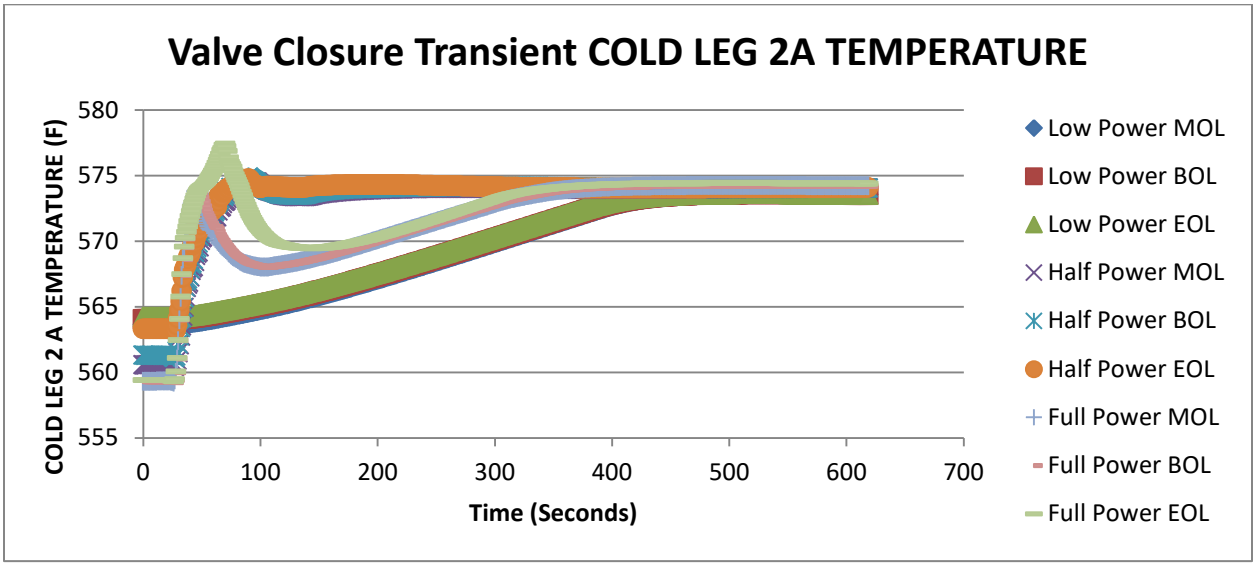

LOCA-LOOP Transient HOT LEG 2 TEMPERATURE
HOT LEG 2 TEMPERATURE (F)
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL


Valve Closure Transient HOT LEG 2 TEMPERATURE
HOT LEG 2 TEMPERATURE (F)
630
620
610
600
590
580
570
560
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL

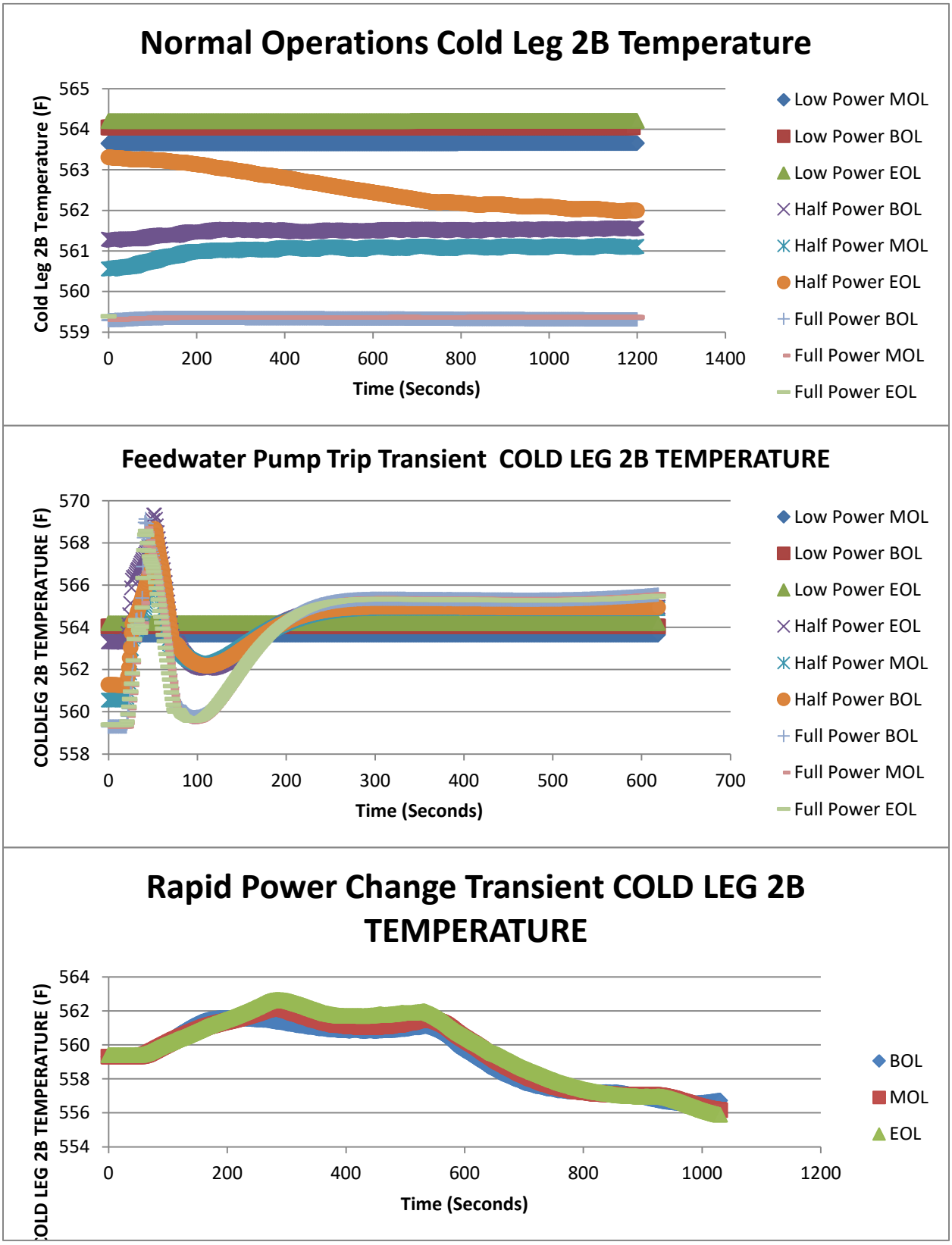
Appendix 1.8 Cold Leg 2A Temperature

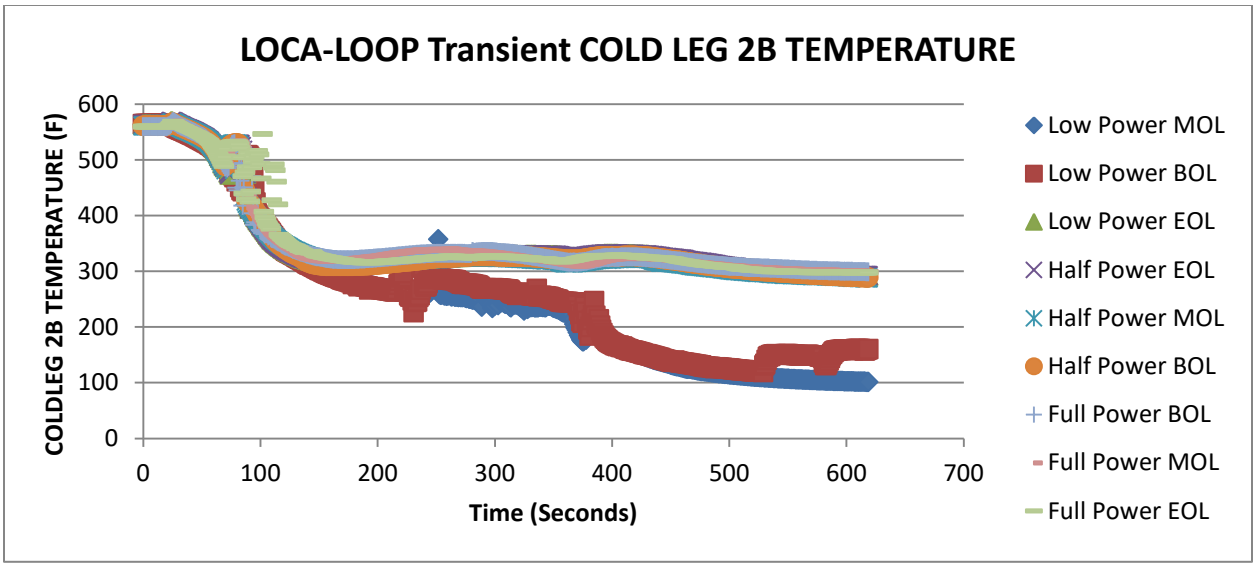


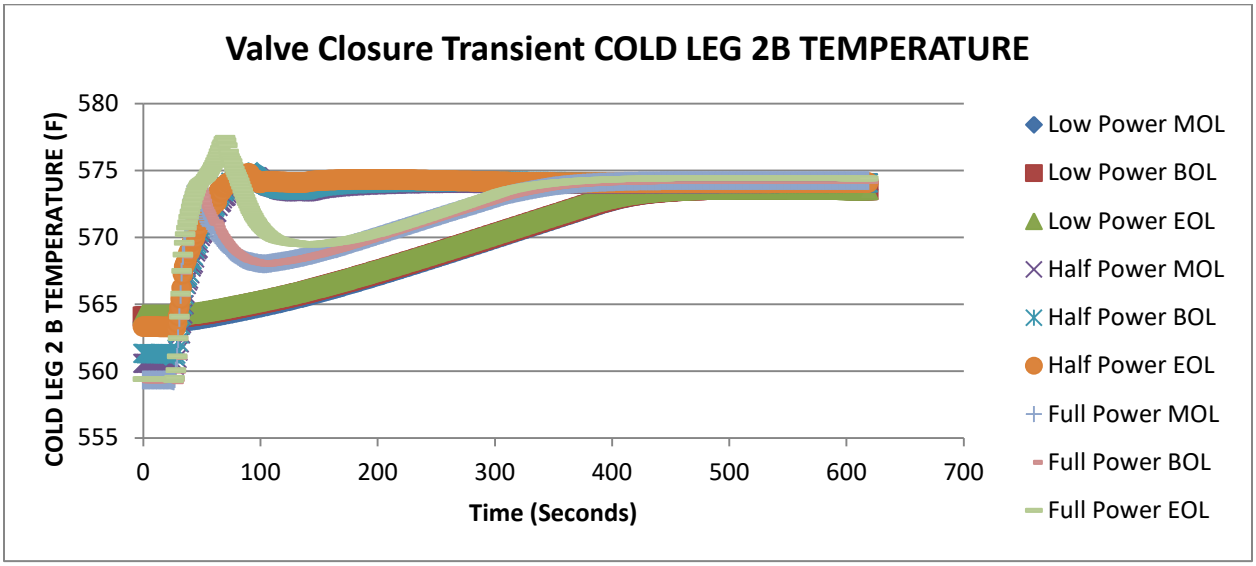

LOCA-LOOP Transient COLD LEG 2A TEMPERATURE
COLDLEG 2A TEMPERATURE (F)
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL


Valve Closure Transient COLD LEG 2A TEMPERATURE
COLD LEG 2 A TEMPERATURE (F)
580
575
570
565
560
555
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

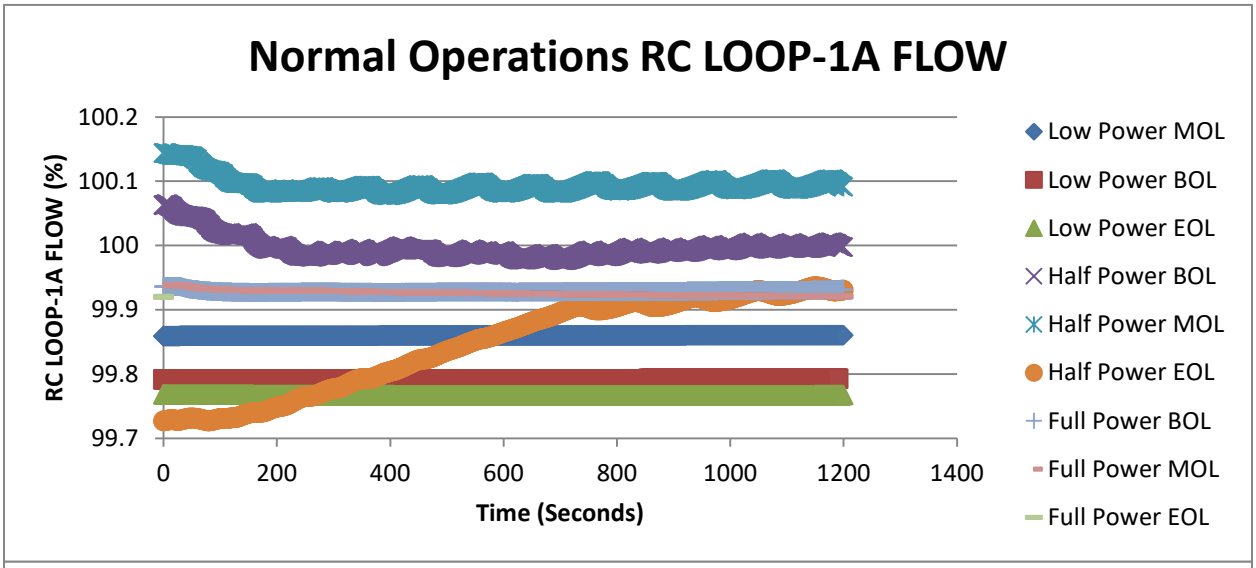
Appendix 1.9 Cold Leg 2B Temperature

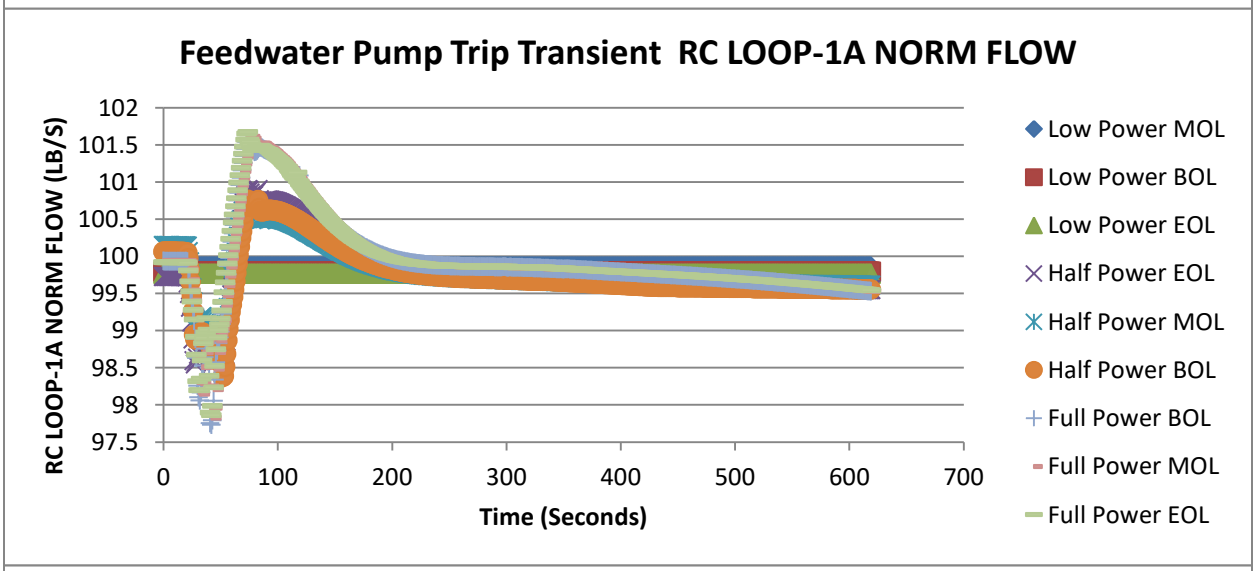


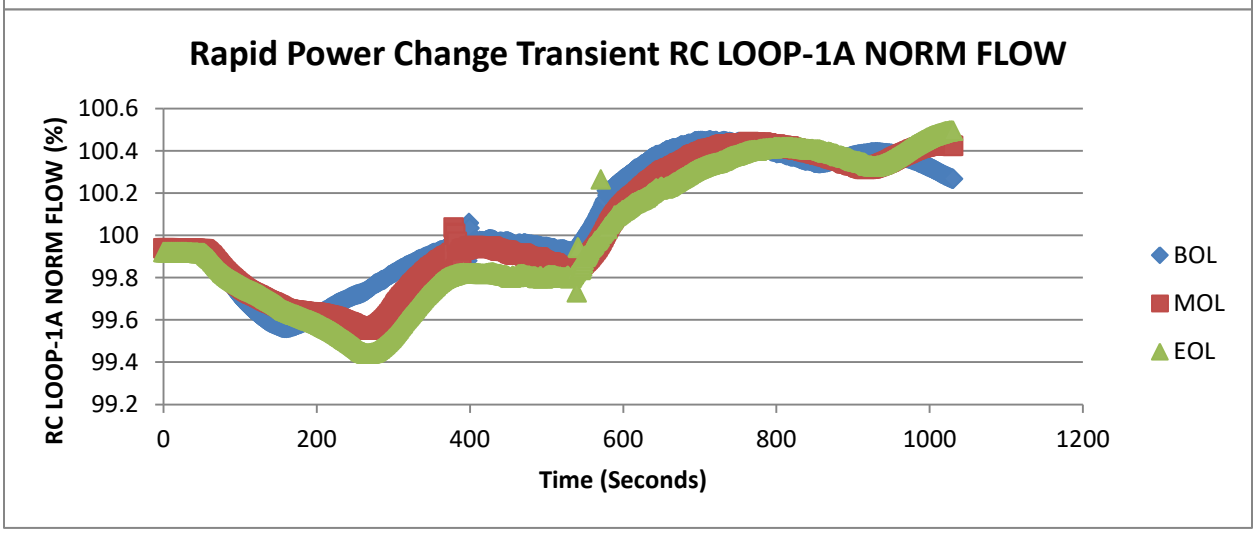

LOCA-LOOP Transient COLD LEG 2B TEMPERATURE
COLDLEG 2B TEMPERATURE (F)
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

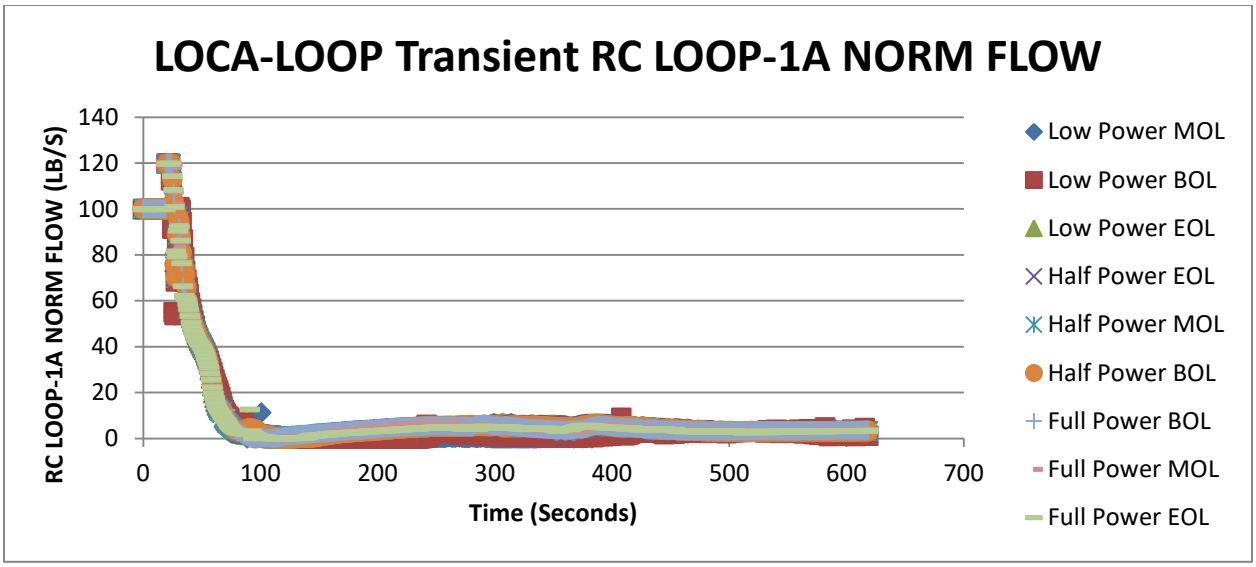

Valve Closure Transient COLD LEG 2B TEMPERATURE
COLD LEG 2 B TEMPERATURE (F)
580
575
570
565
560
555
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

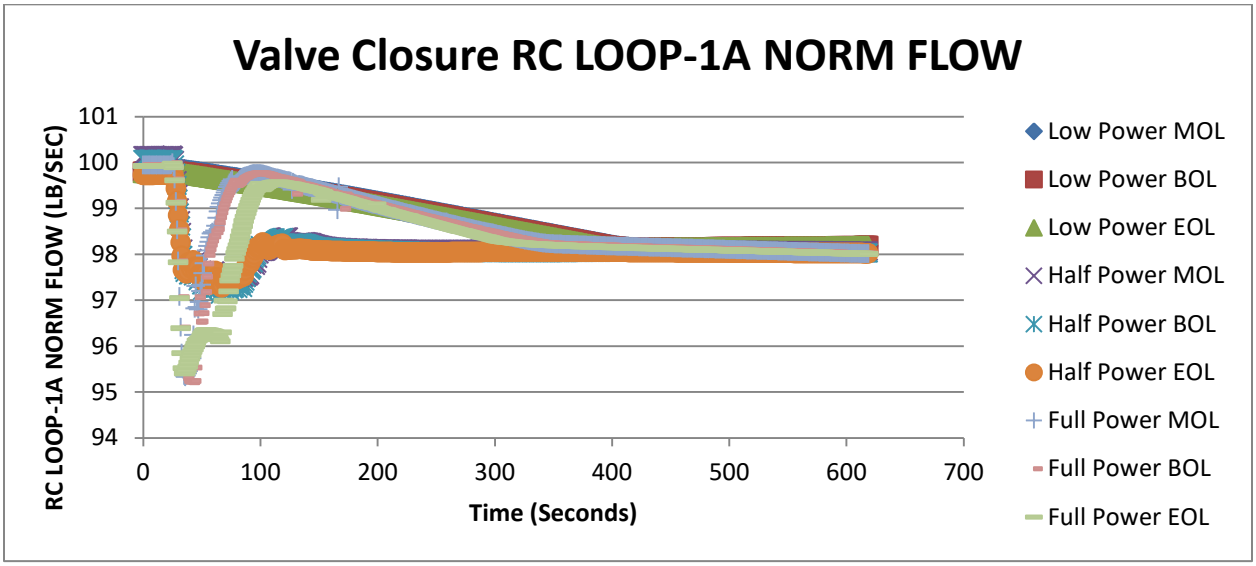
Appendix 1.10 RC LOOP-1A FLOW



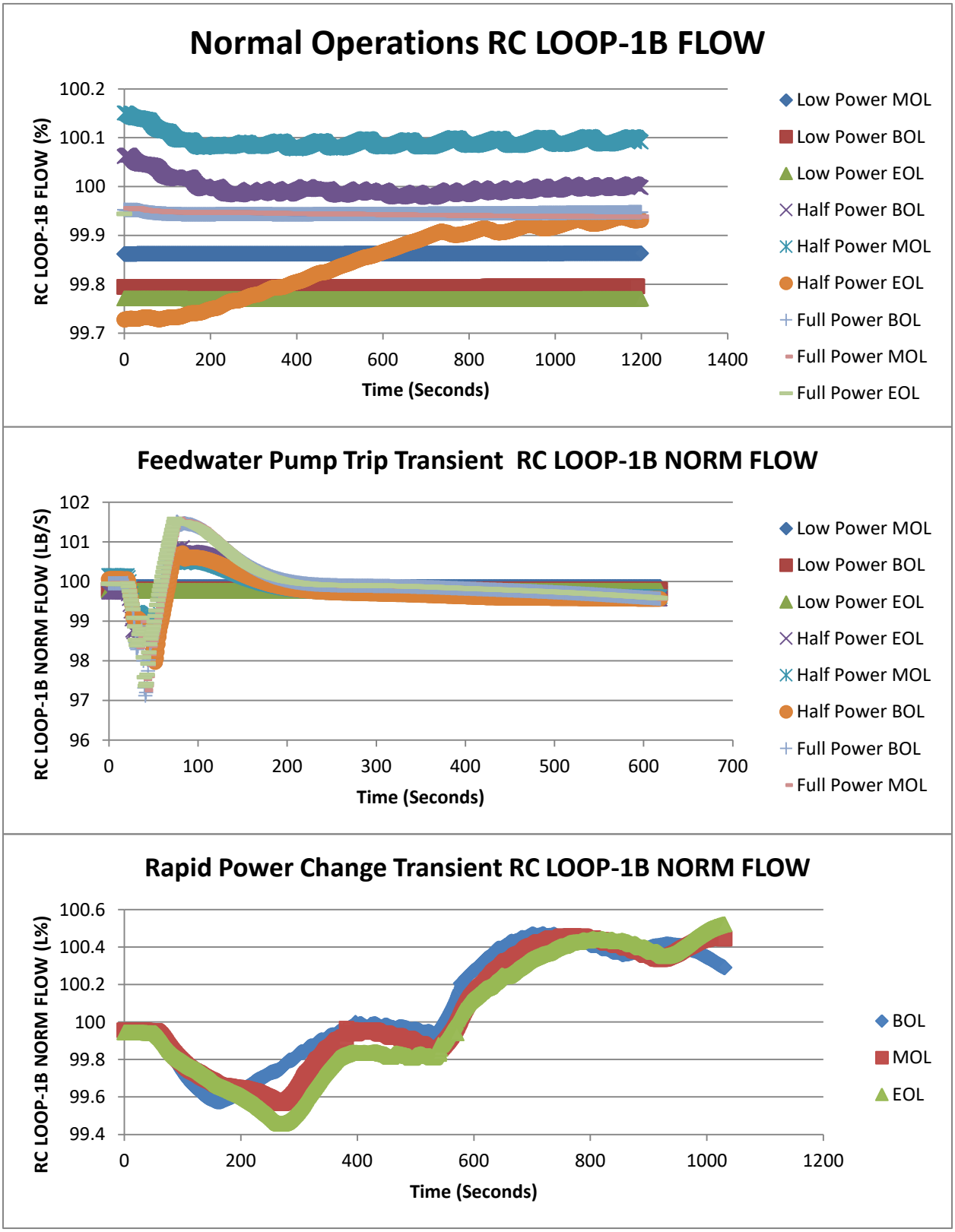


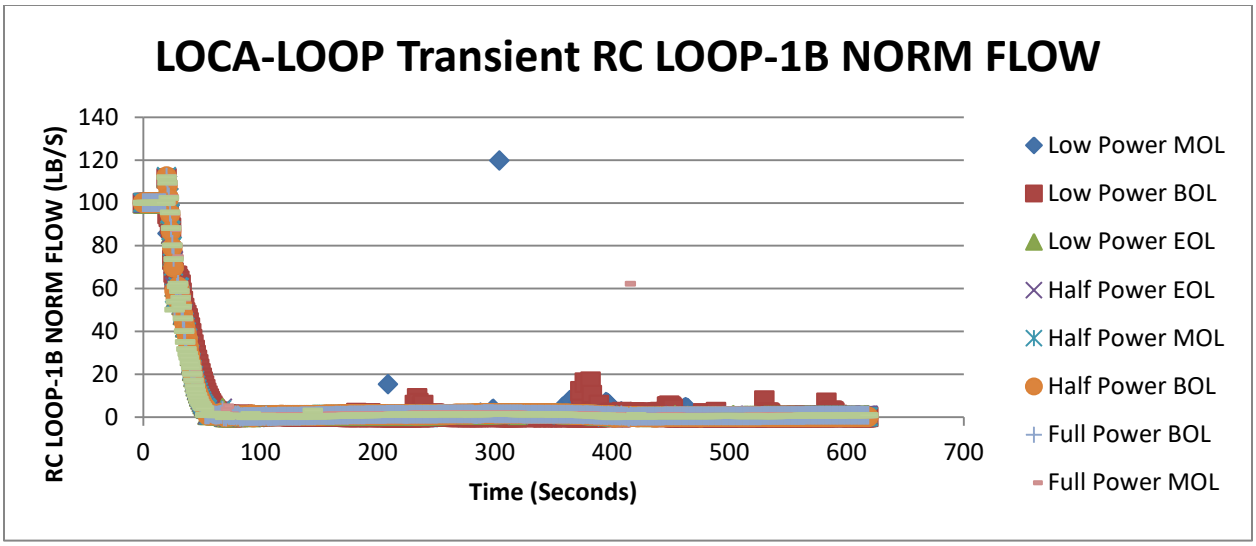


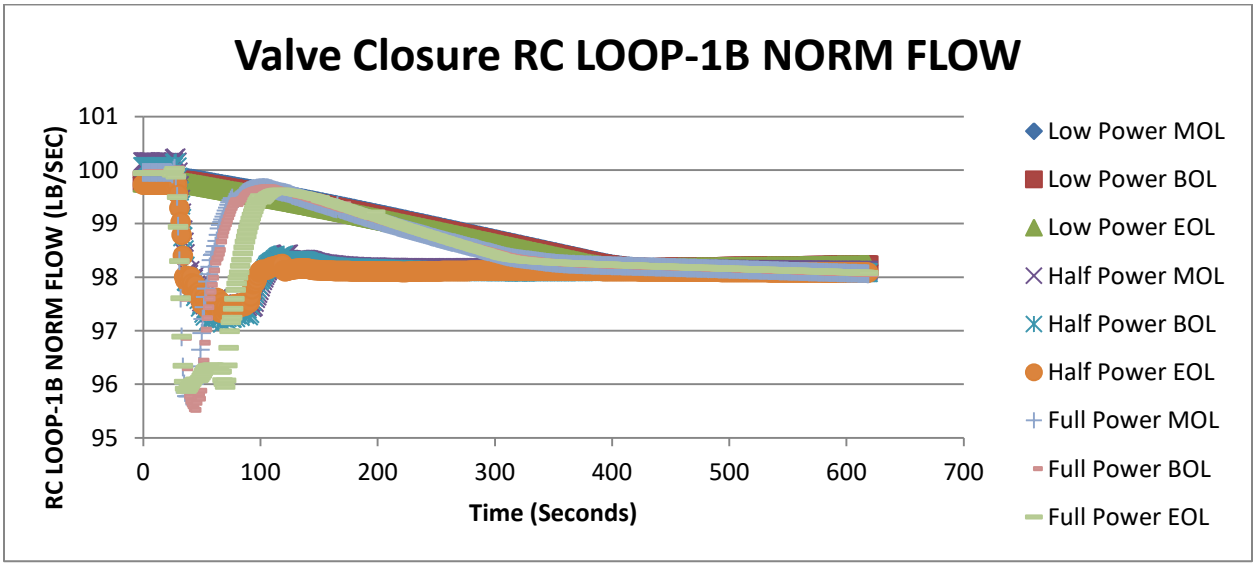

LOCA-LOOP Transient RC LOOP-1A NORM FLOW
RC LOOP-1A NORM FLOW (LB/S)
140
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL


Valve Closure RC LOOP-1A NORM FLOW
RC LOOP-1A NORM FLOW (LB/SEC)
101
100
99
98
97
96
95
94
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

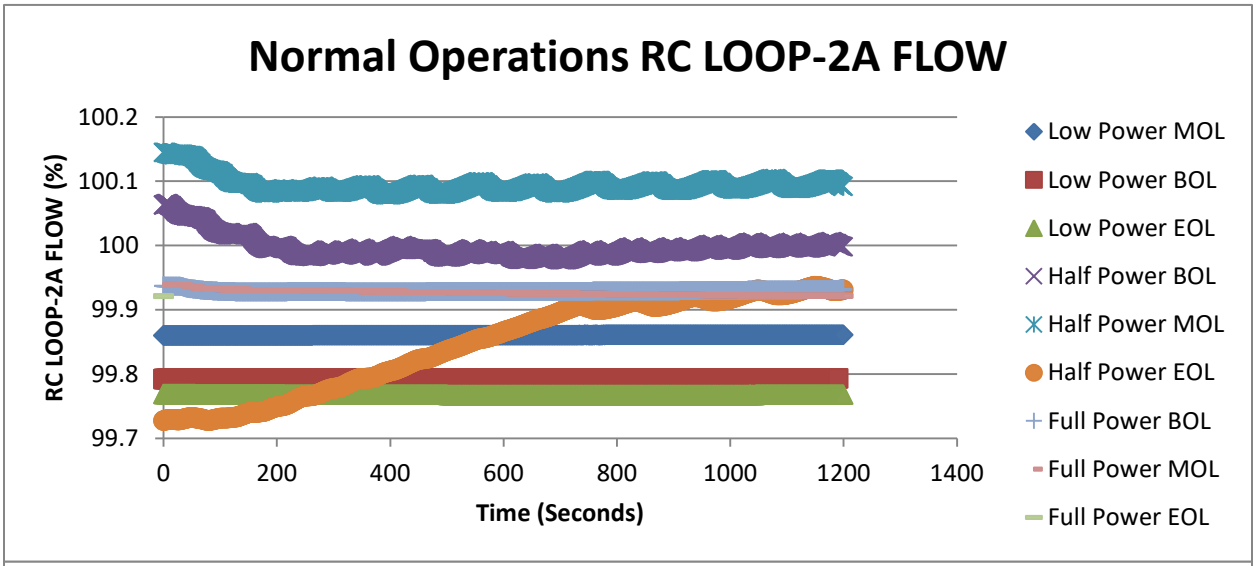
Appendix 1.11 RC LOOP-1B FLOW

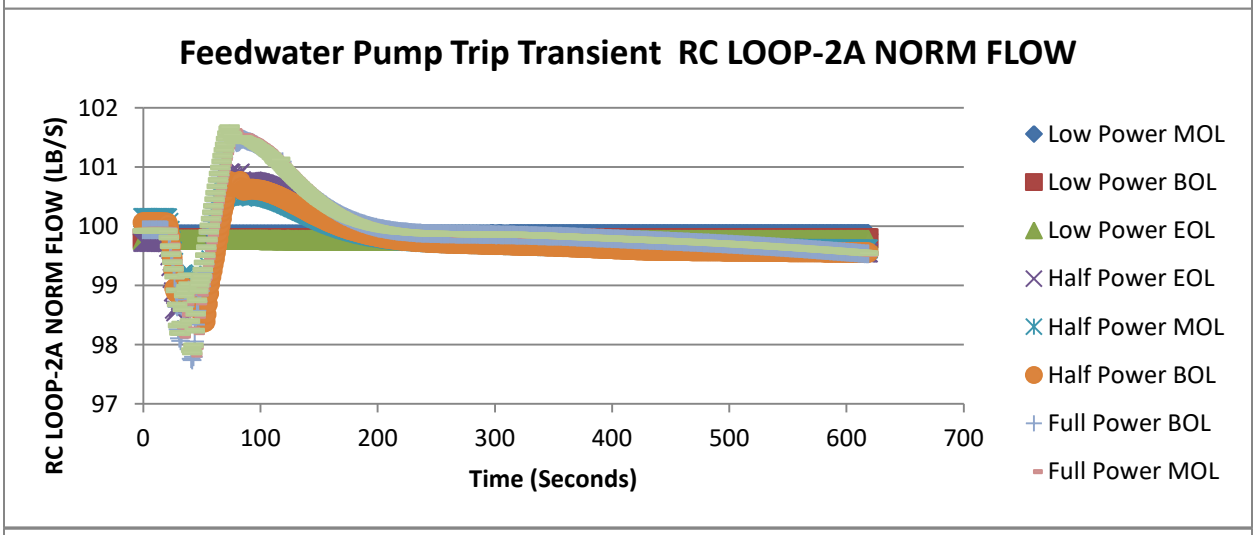


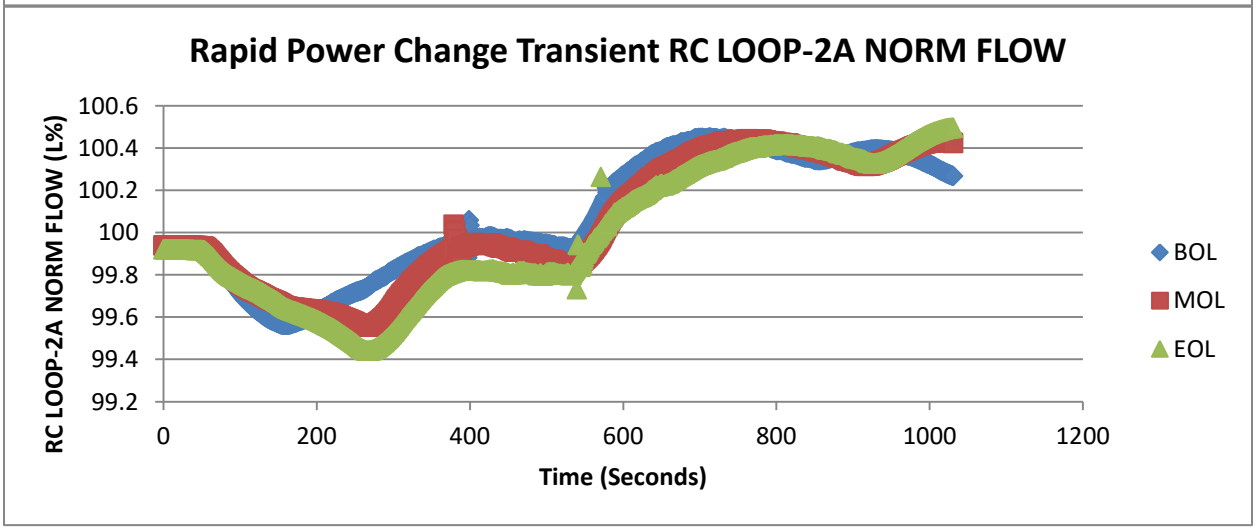

LOCA-LOOP Transient RC LOOP-1B NORM FLOW
RC LOOP-1B NORM FLOW (LB/S)
140
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL

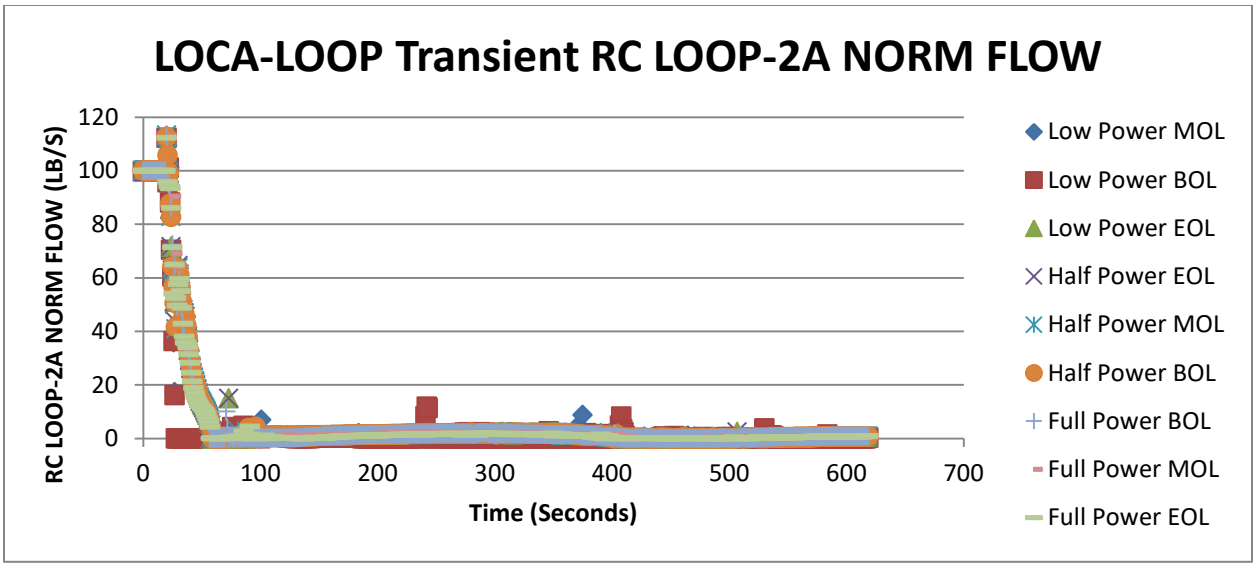

Valve Closure RC LOOP-1B NORM FLOW
RC LOOP-1B NORM FLOW (LB/SEC)
101
100
99
98
97
96
95
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

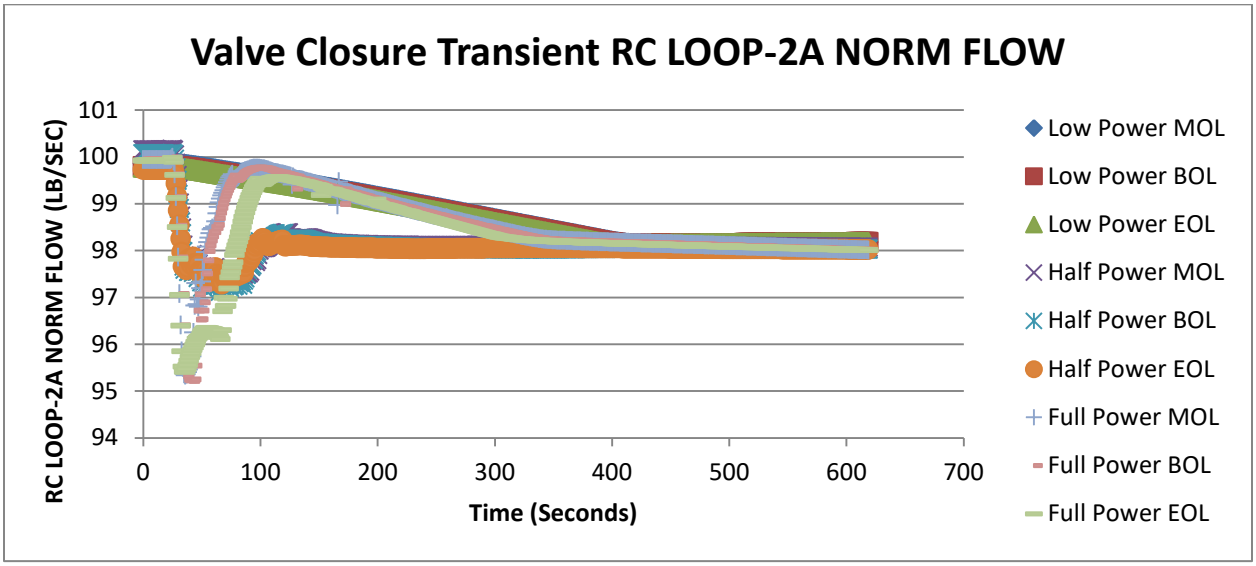
Appendix 1.12 RC LOOP-2A FLOW



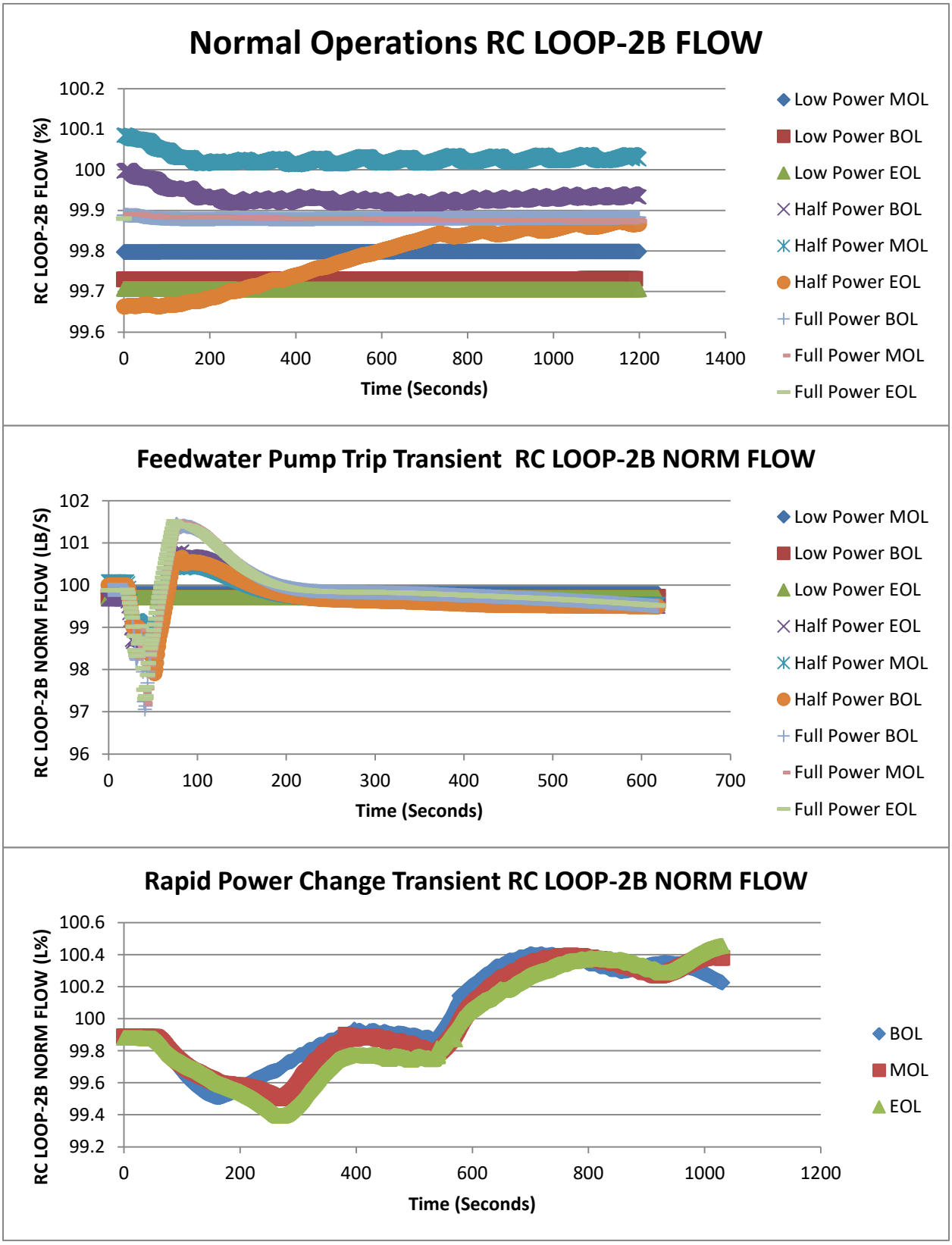


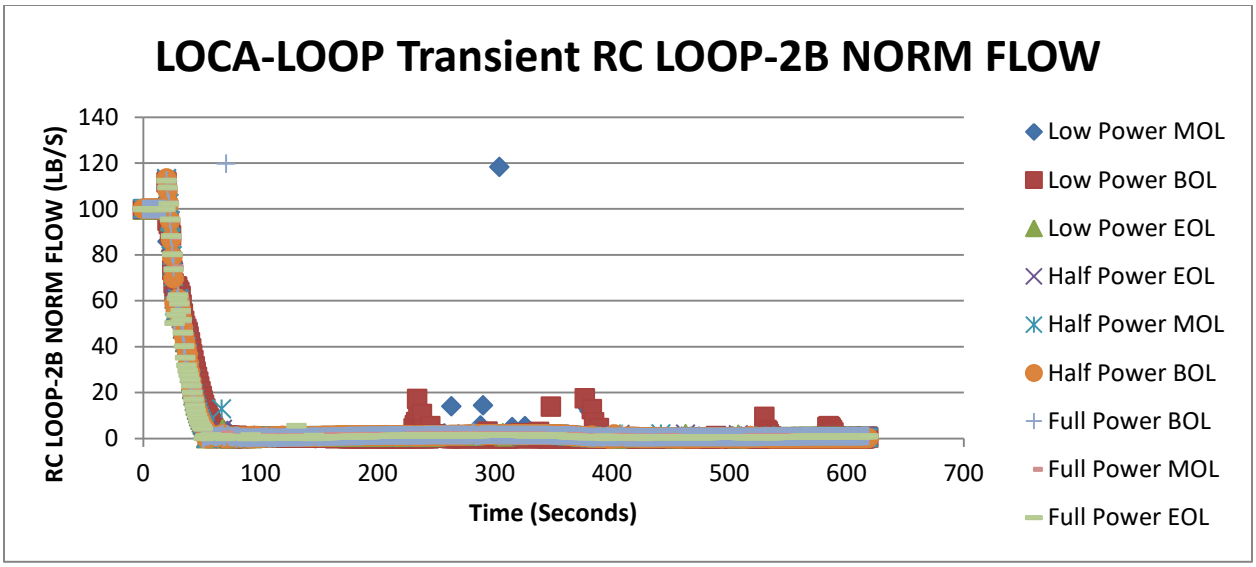


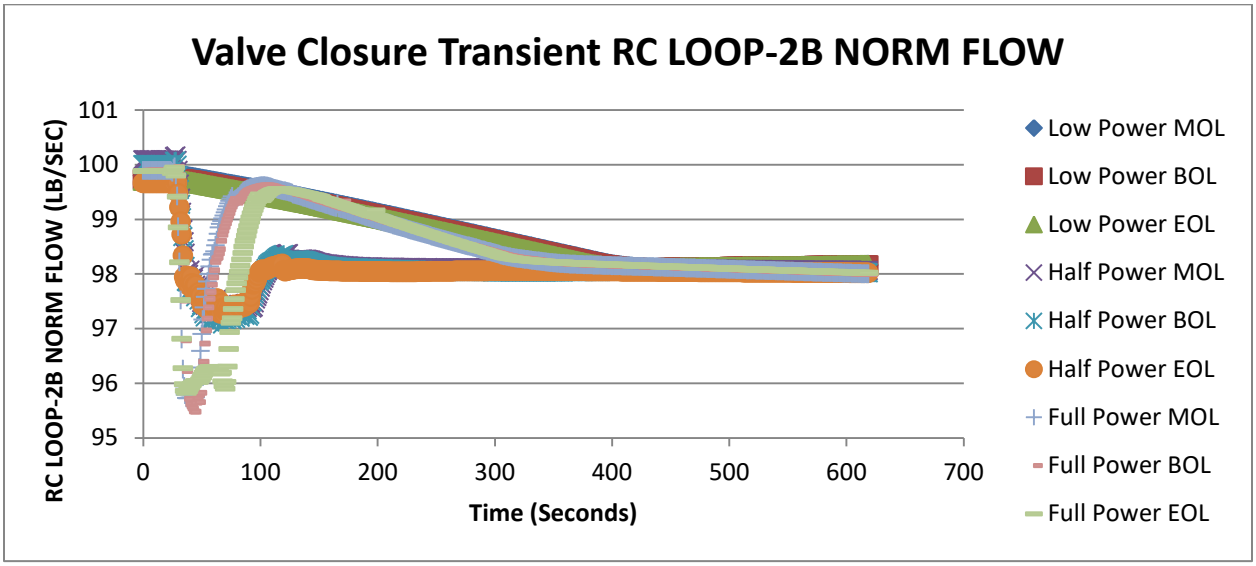

LOCA-LOOP Transient RC LOOP-2A NORM FLOW
RC LOOP-2A NORM FLOW (LB/S)
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL


Valve Closure Transient RC LOOP-2A NORM FLOW
RC LOOP-2A NORM FLOW (LB/SEC)
101
100
99
98
97
96
95
94
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

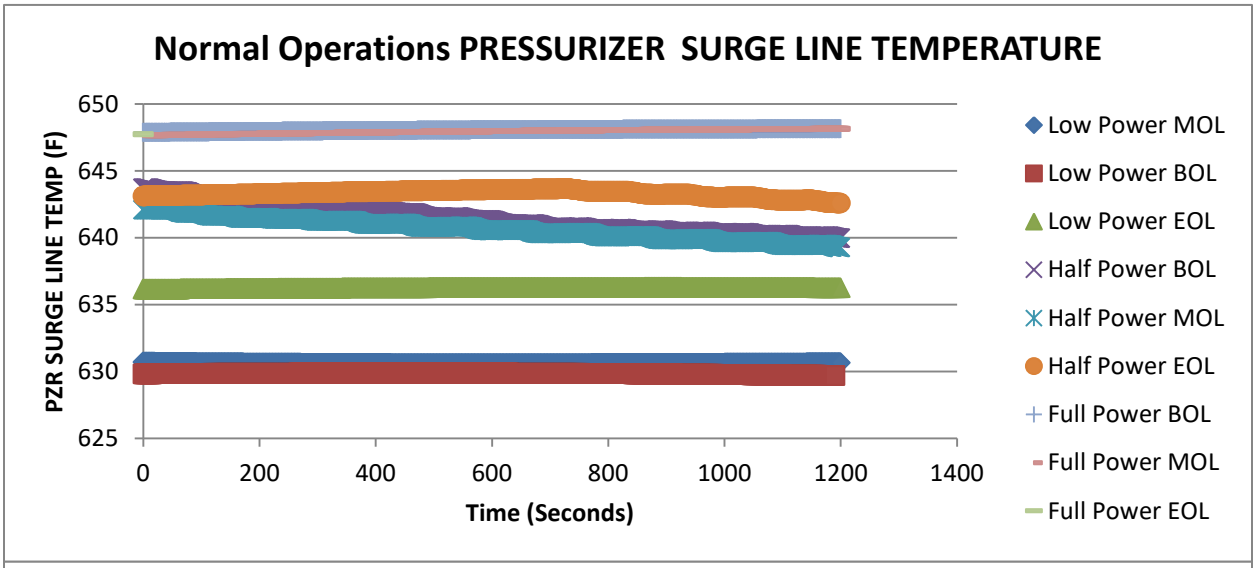
Appendix 1.13 RC LOOP-2B FLOW

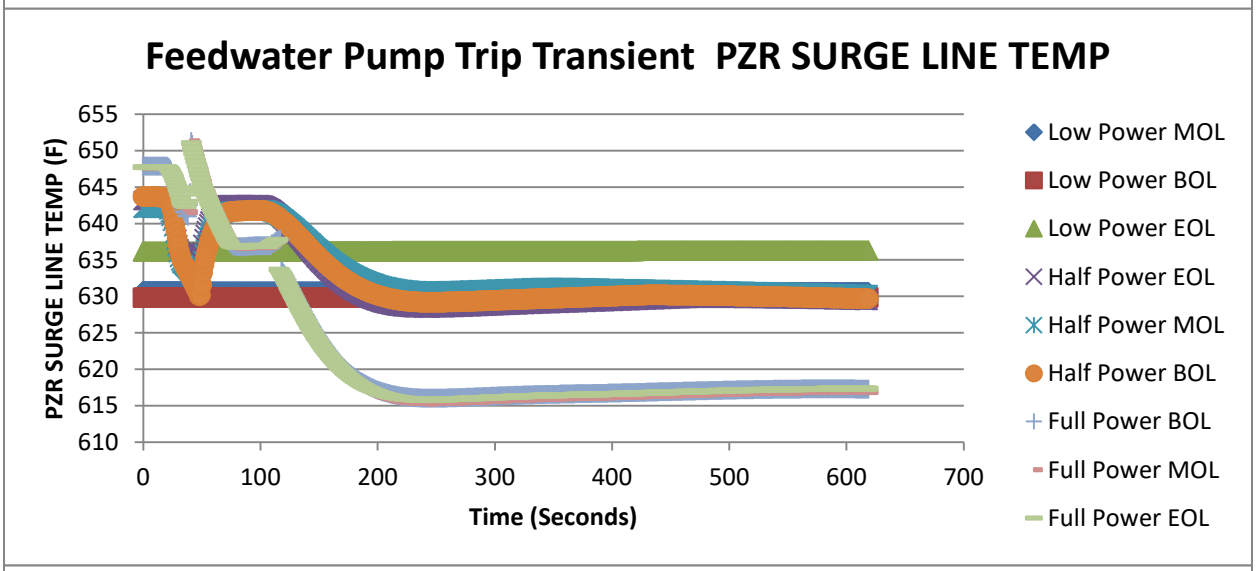


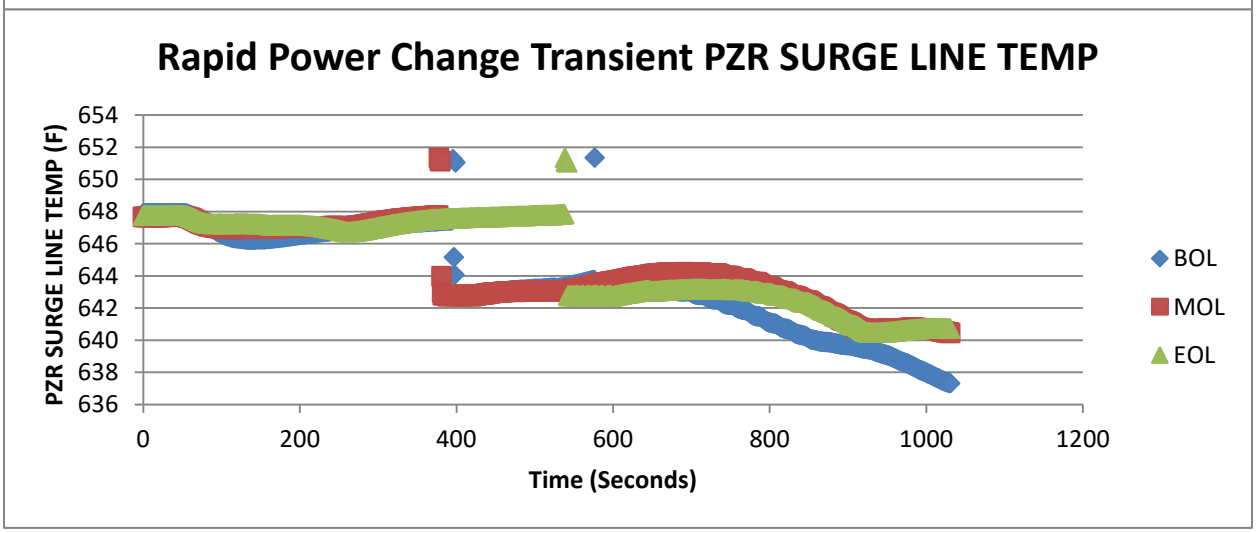

LOCA-LOOP Transient RC LOOP-2B NORM FLOW
RC LOOP-2B NORM FLOW (LB/S)
140
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

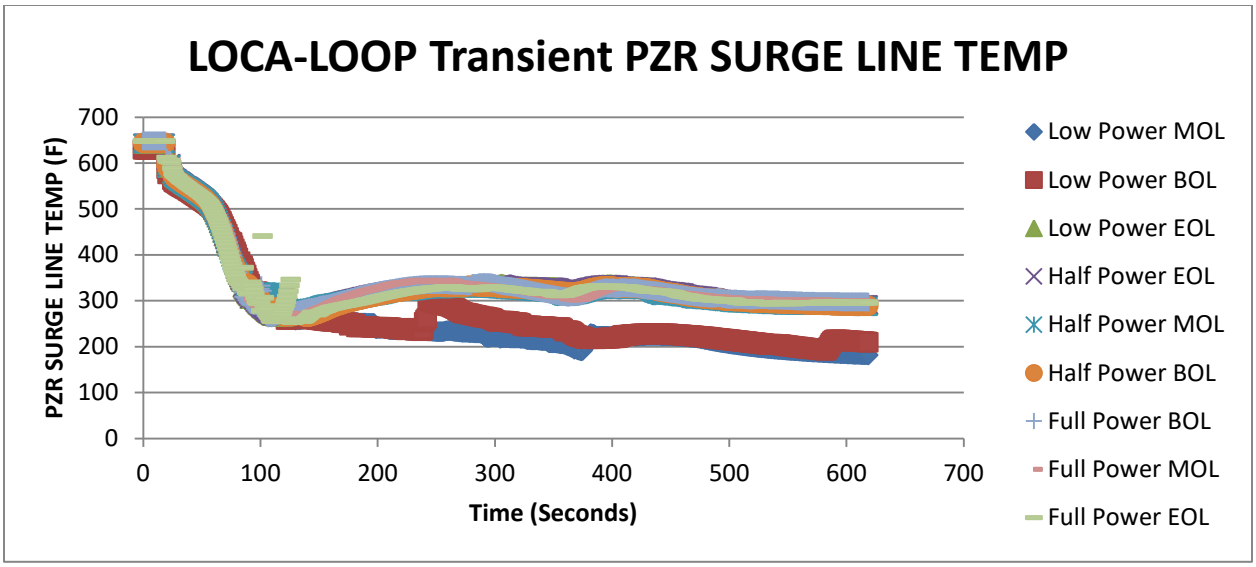

Valve Closure Transient RC LOOP-2B NORM FLOW
RC LOOP-2B NORM FLOW (LB/SEC)
101
100
99
98
97
96
95
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

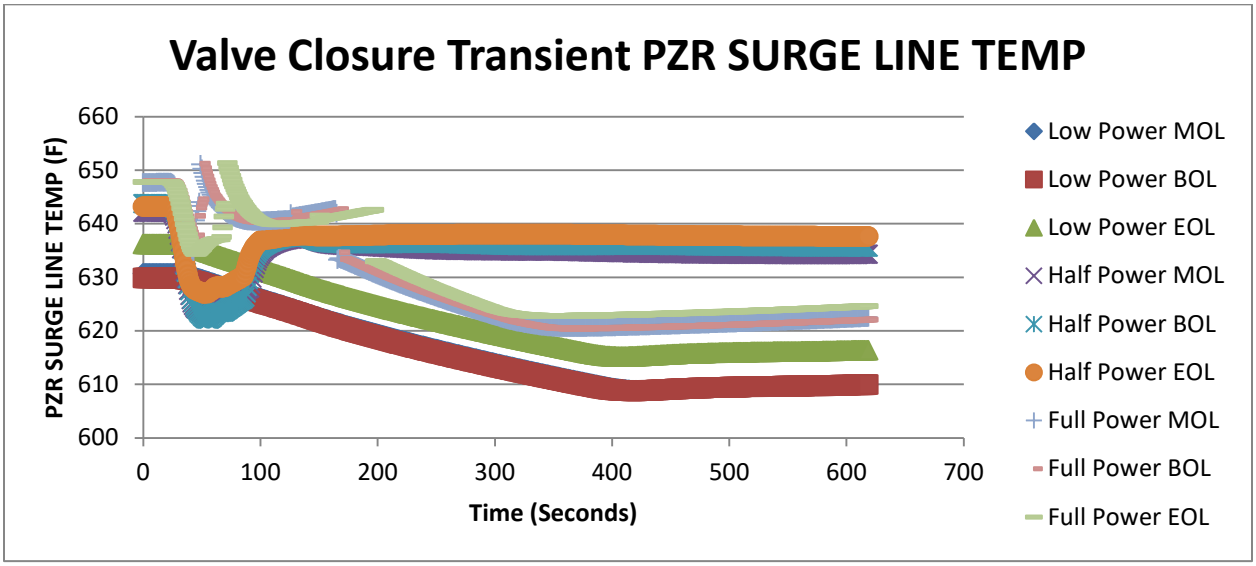
Appendix 1.14 PRESSURIZER SURGE LINE TEMPERATURE



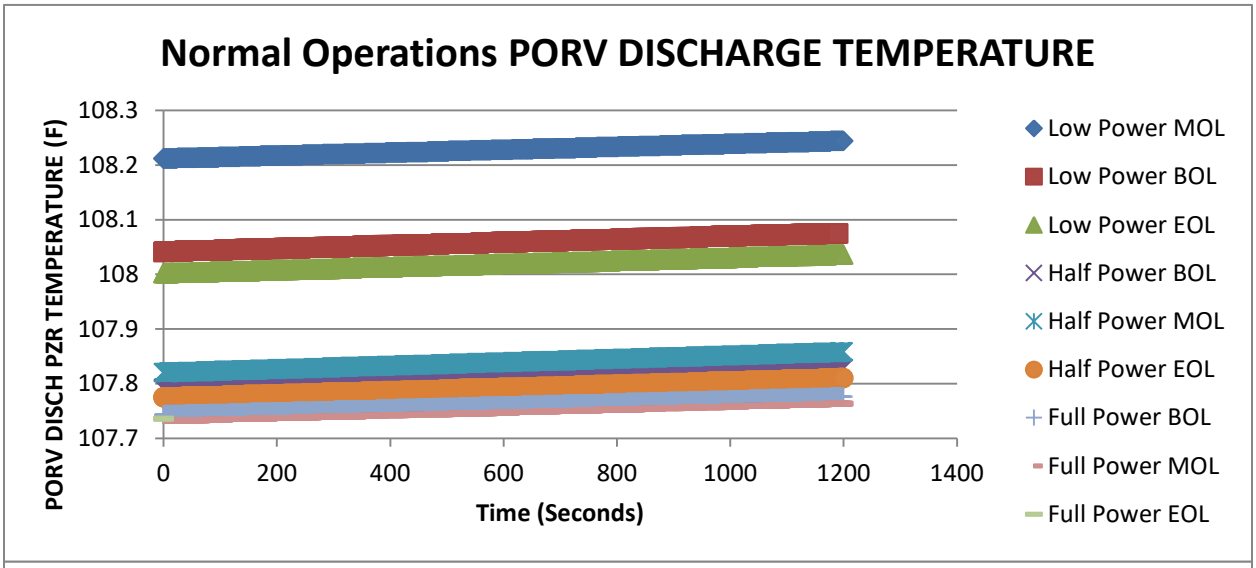


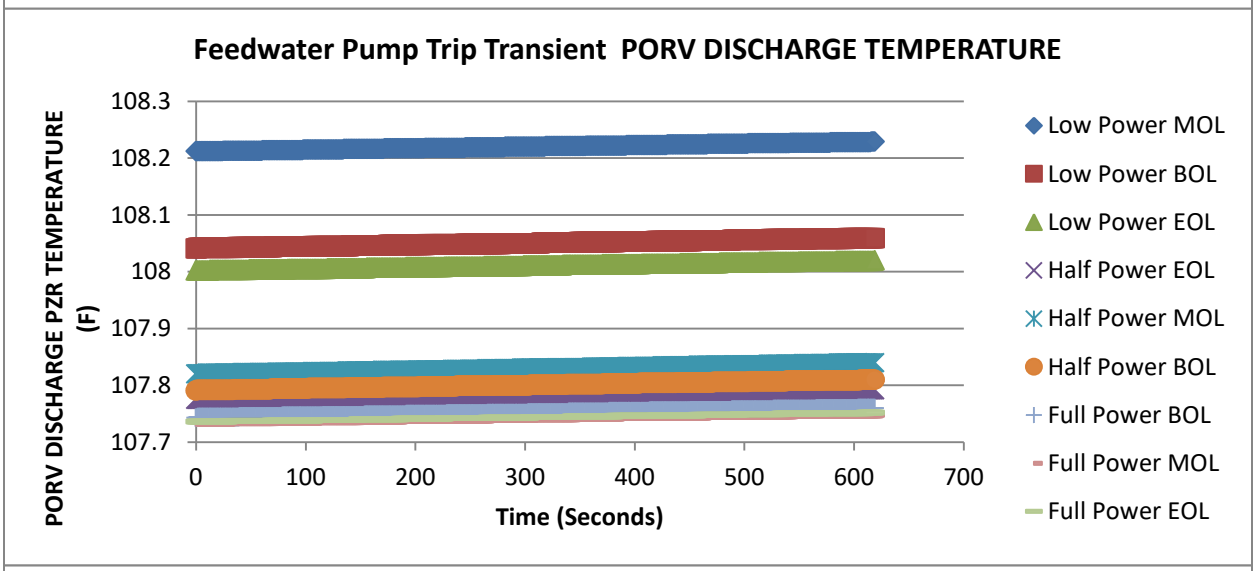


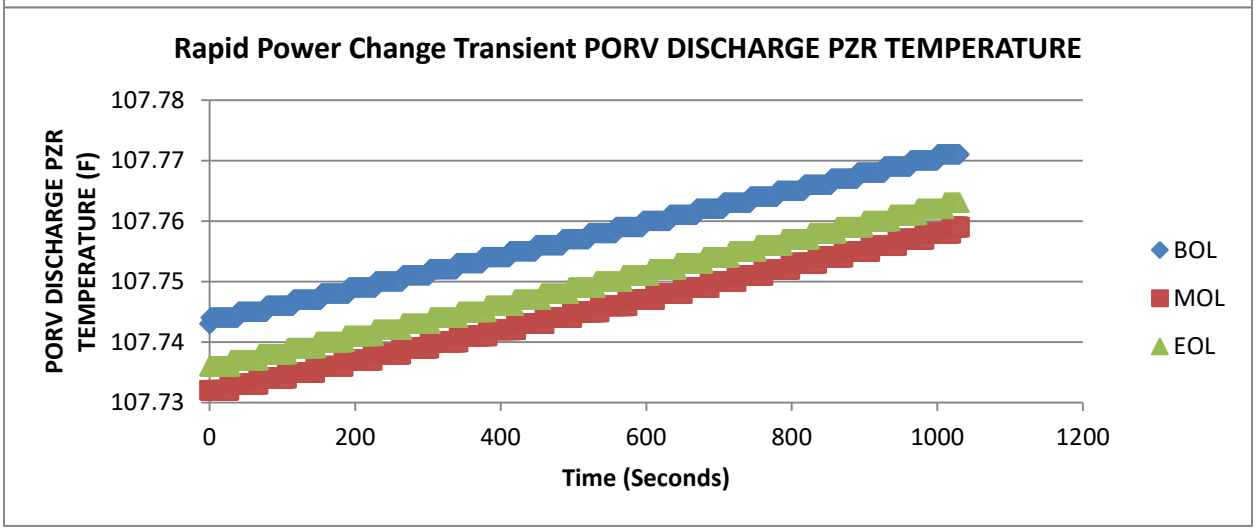

LOCA-LOOP Transient PZR SURGE LINE TEMP
PZR SURGE LINE TEMP (F)
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

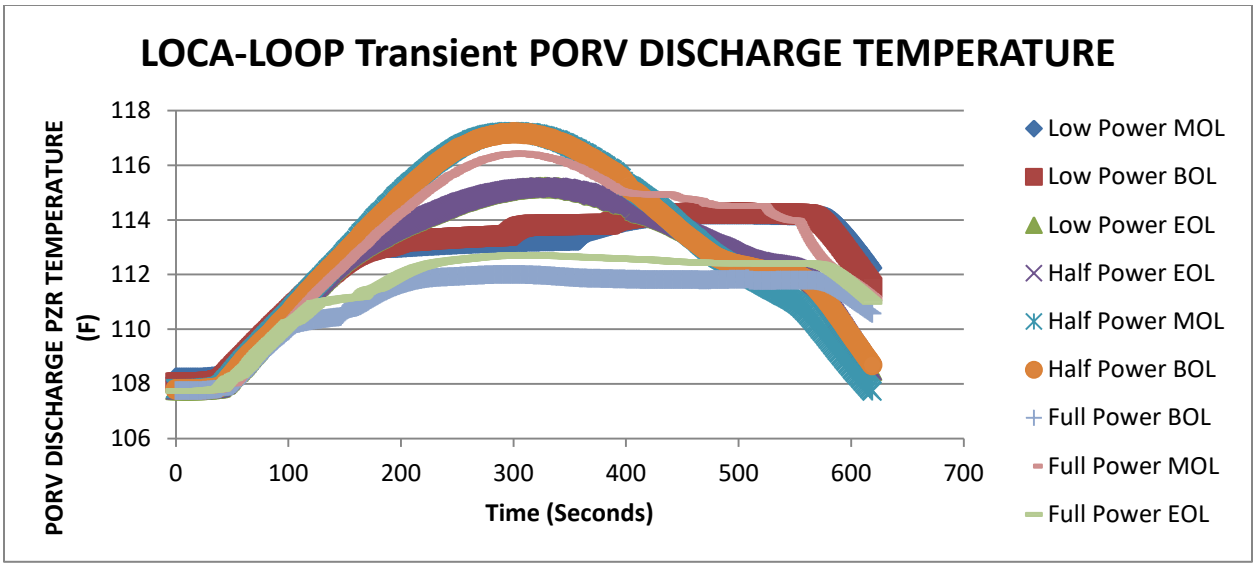

Valve Closure Transient PZR SURGE LINE TEMP
PZR SURGE LINE TEMP (F)
660
650
640
630
620
610
600
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

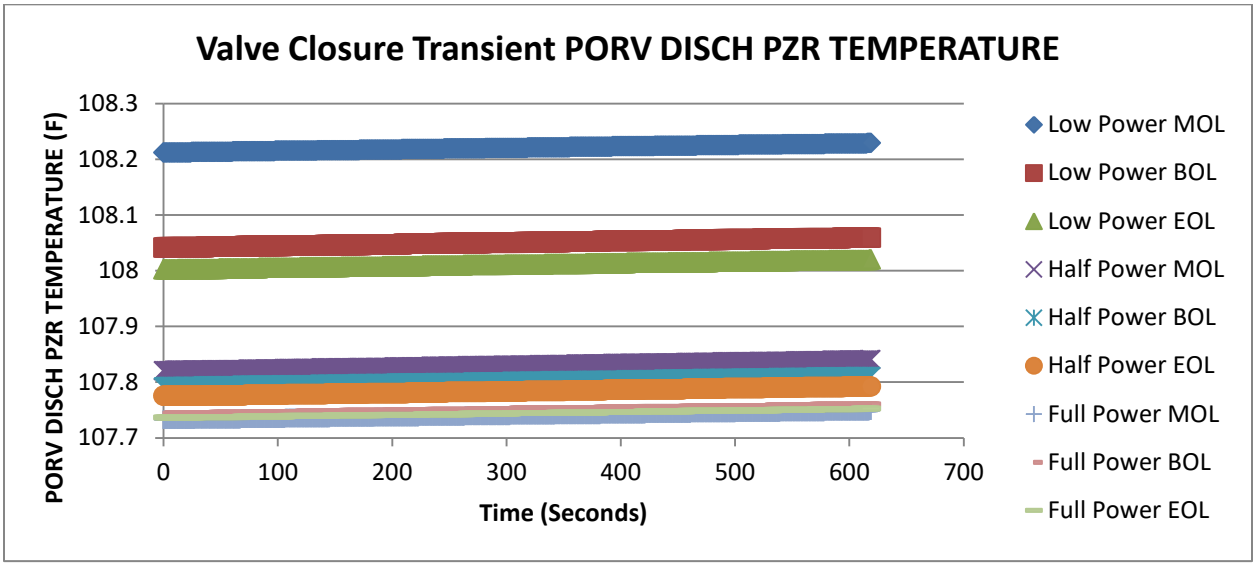
Appendix 1.15 PORV DISCHARGE TEMPERATURE



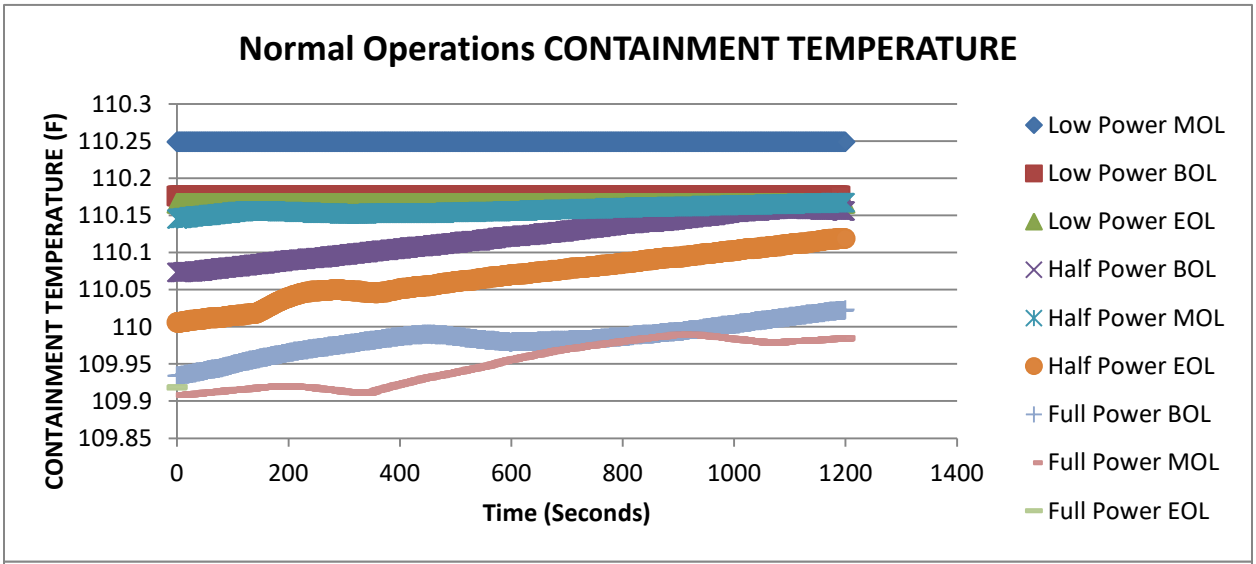


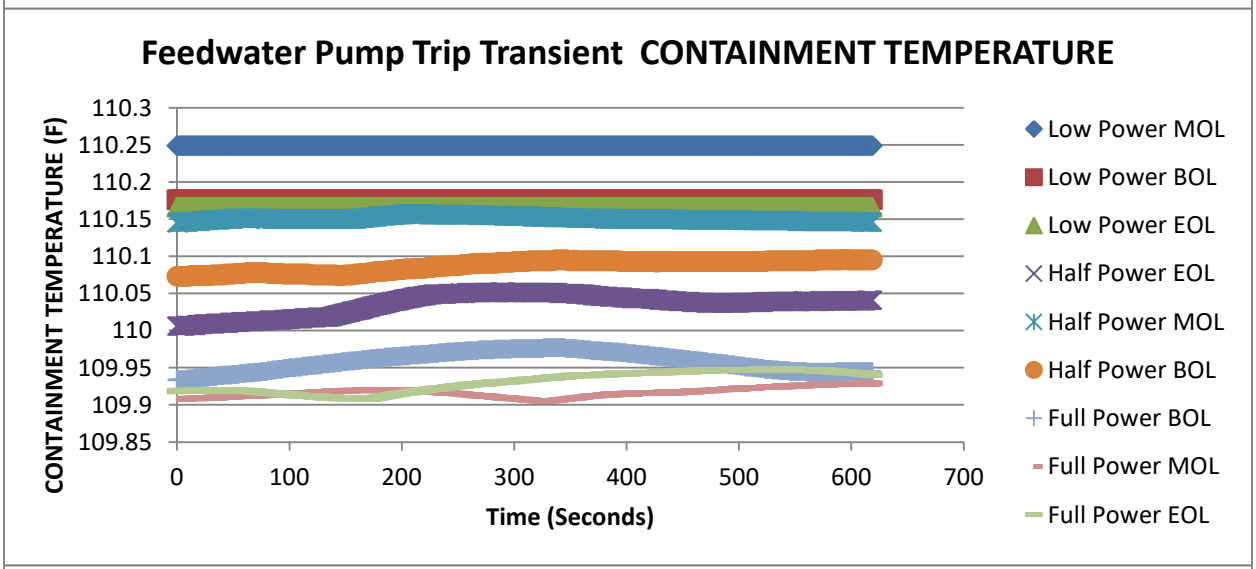


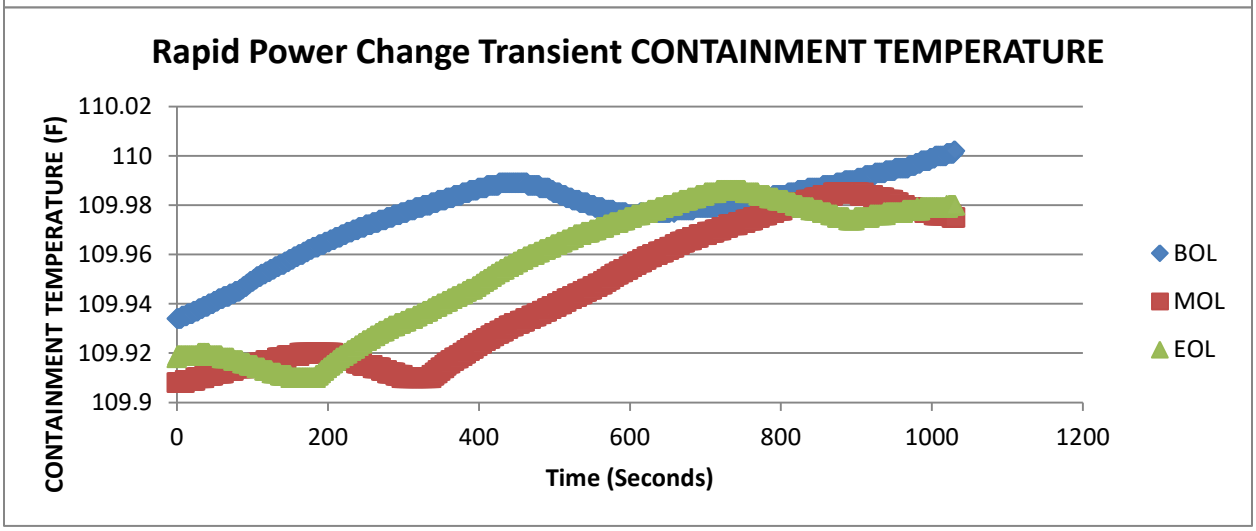

LOCA-LOOP Transient PORV DISCHARGE TEMPERATURE
PORV DISCHARGE PZR TEMPERATURE (F)
118
116
114
112
110
108
106
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

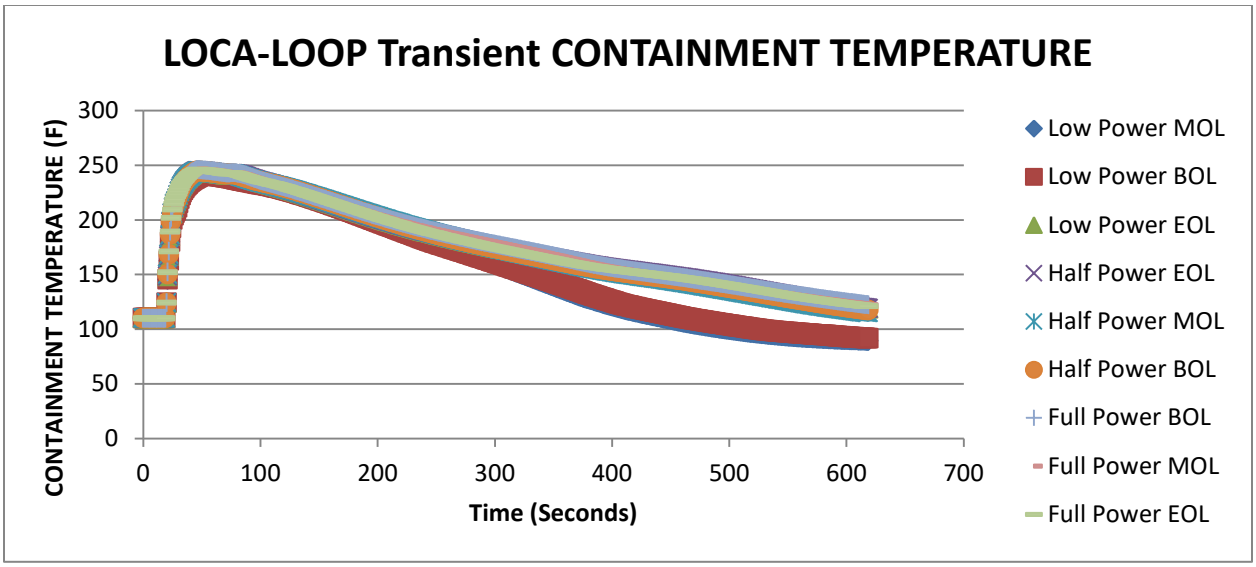

Valve Closure Transient PORV DISCH PZR TEMPERATURE
PORV DISCH PZR TEMPERATURE (F)
108.3
108.2
108.1
108
107.9
107.8
107.7
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

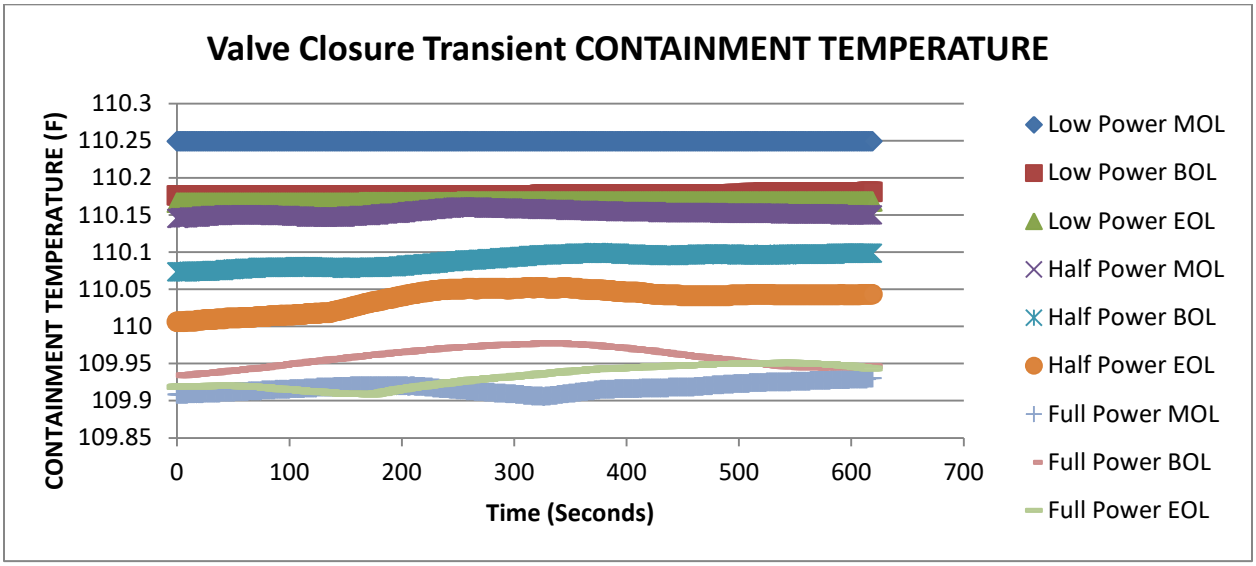
Appendix 1.16 CONTAINMENT TEMPERATURE



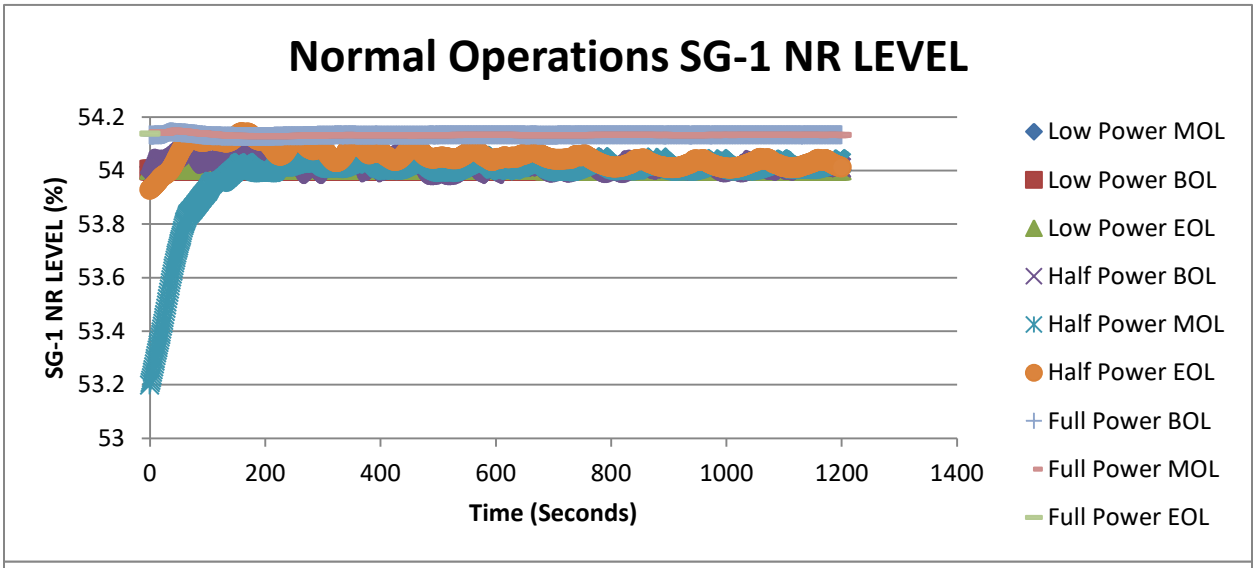


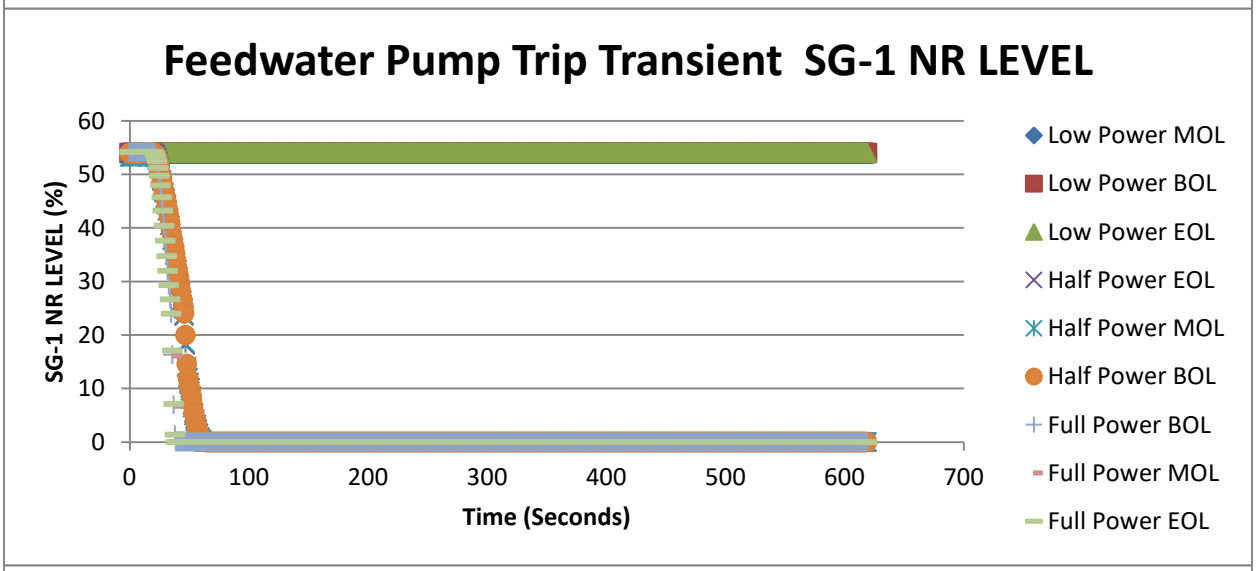


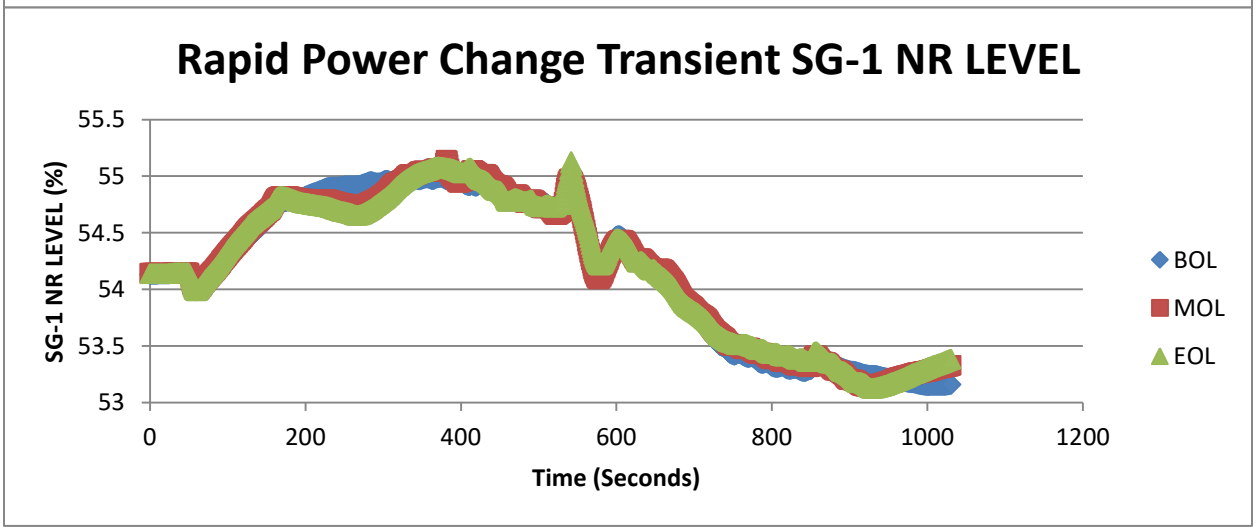

LOCA-LOOP Transient CONTAINMENT TEMPERATURE
CONTAINMENT TEMPERATURE (F)
300
250
200
150
100
50
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

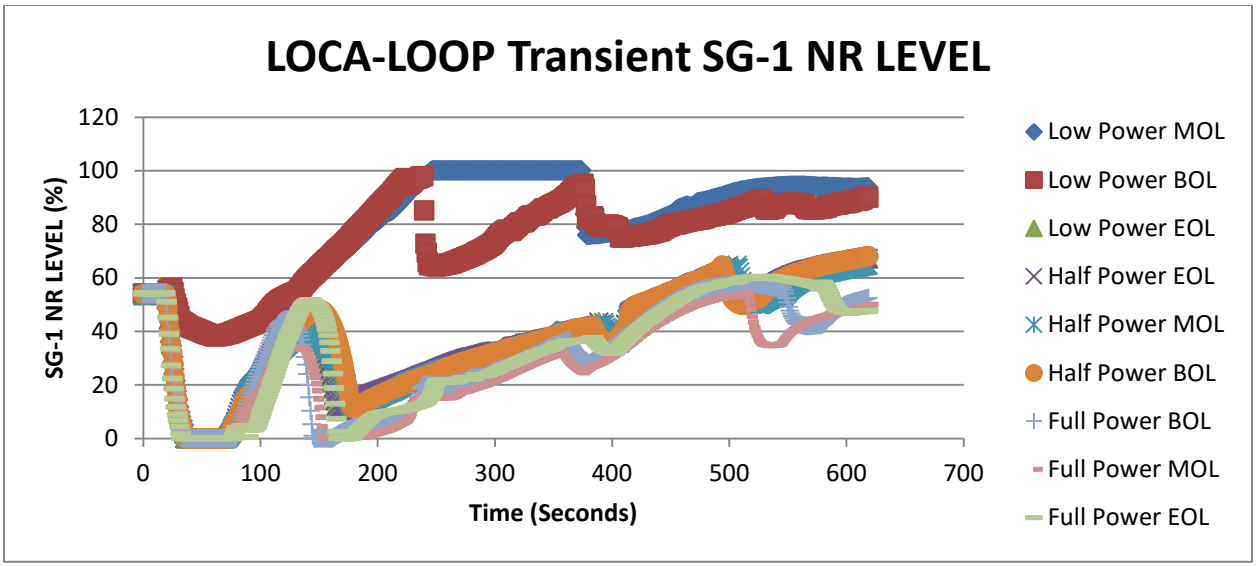

Valve Closure Transient CONTAINMENT TEMPERATURE
CONTAINMENT TEMPERATURE (F)
110.3
110.25
110.2
110.15
110.1
110.05
110
109.95
109.9
109.85
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

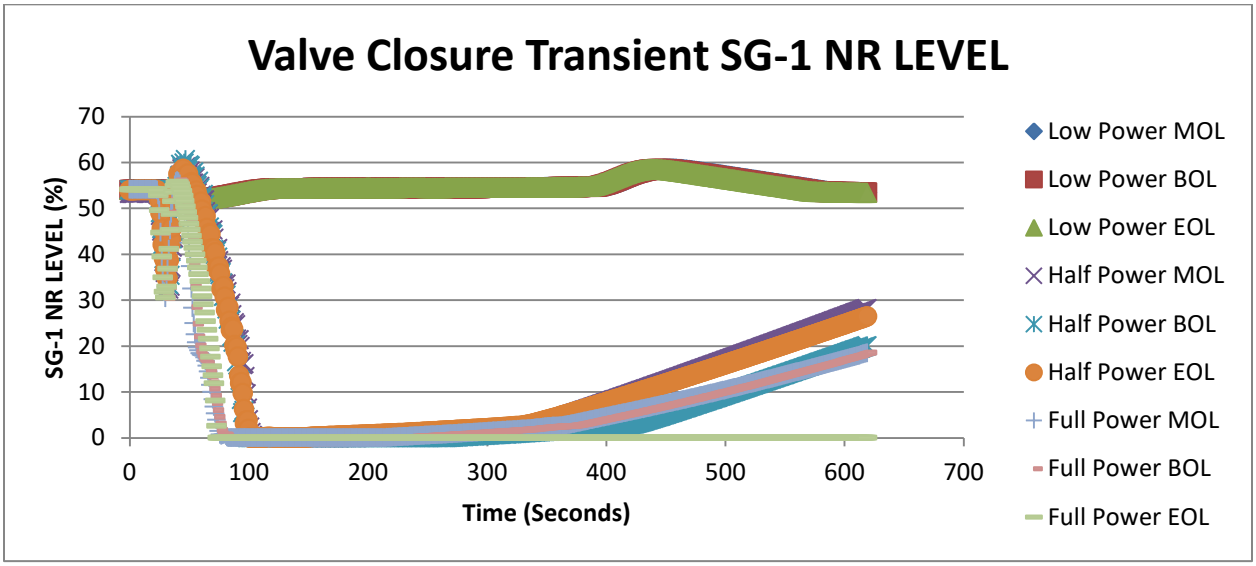
Appendix 1.17 SG-1 NR LEVEL



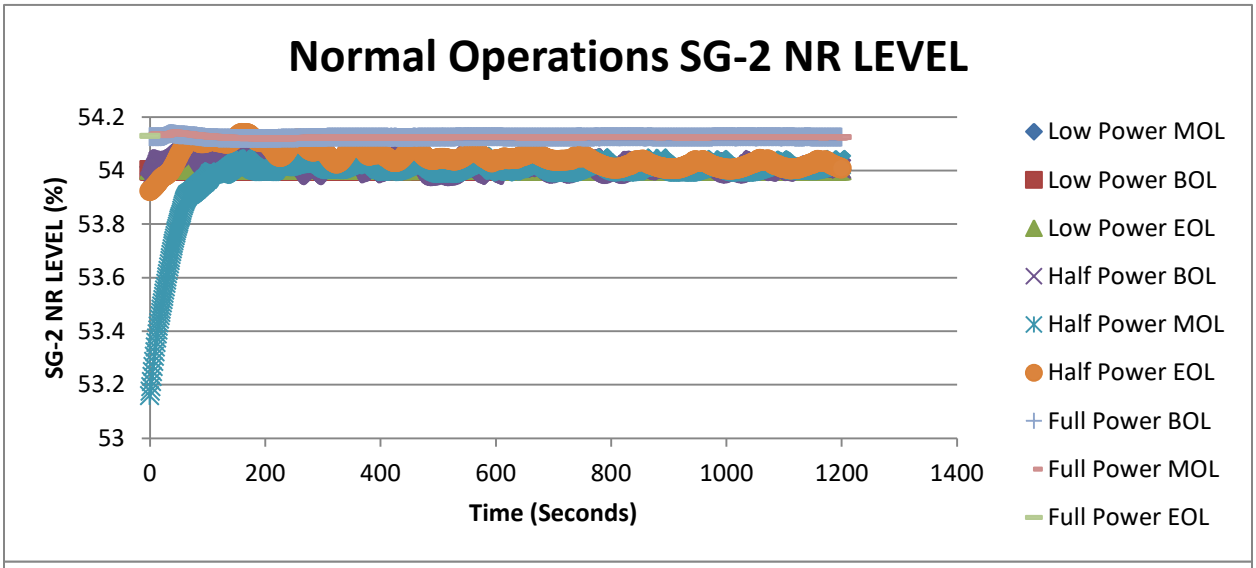


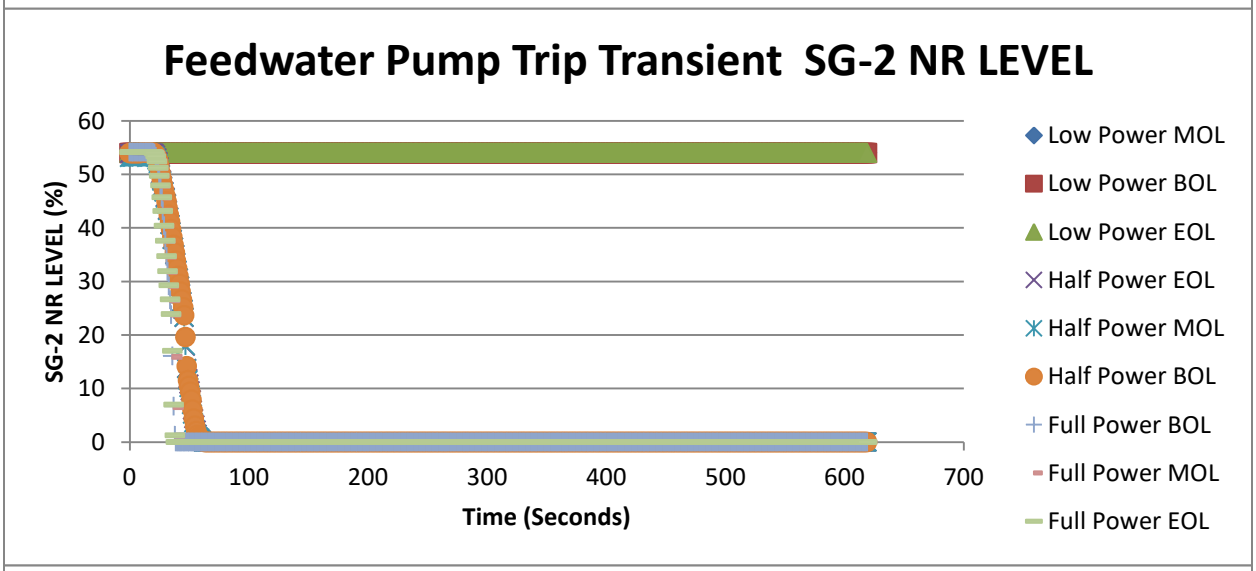


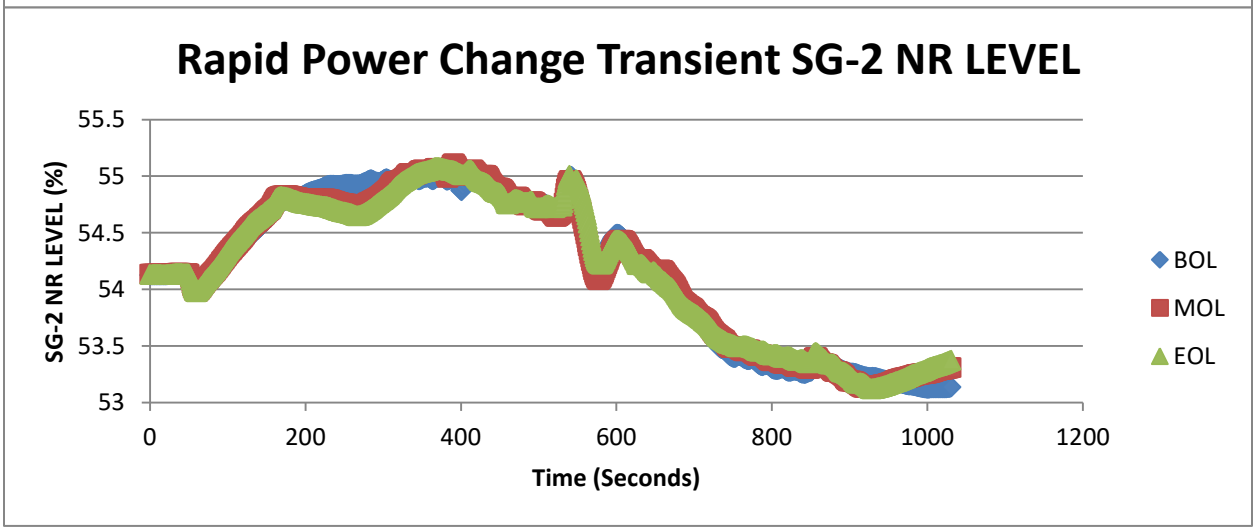

LOCA-LOOP Transient SG-1 NR LEVEL
SG-1 NR LEVEL (%)
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

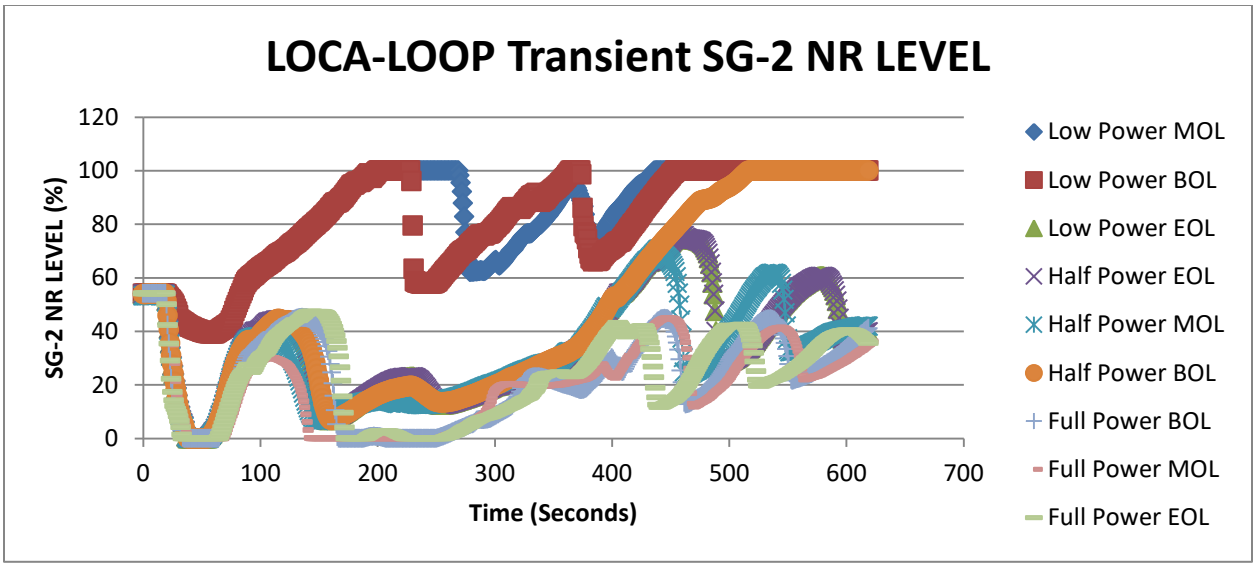

Valve Closure Transient SG-1 NR LEVEL
SG-1 NR LEVEL (%)
70
60
50
40
30
20
10
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

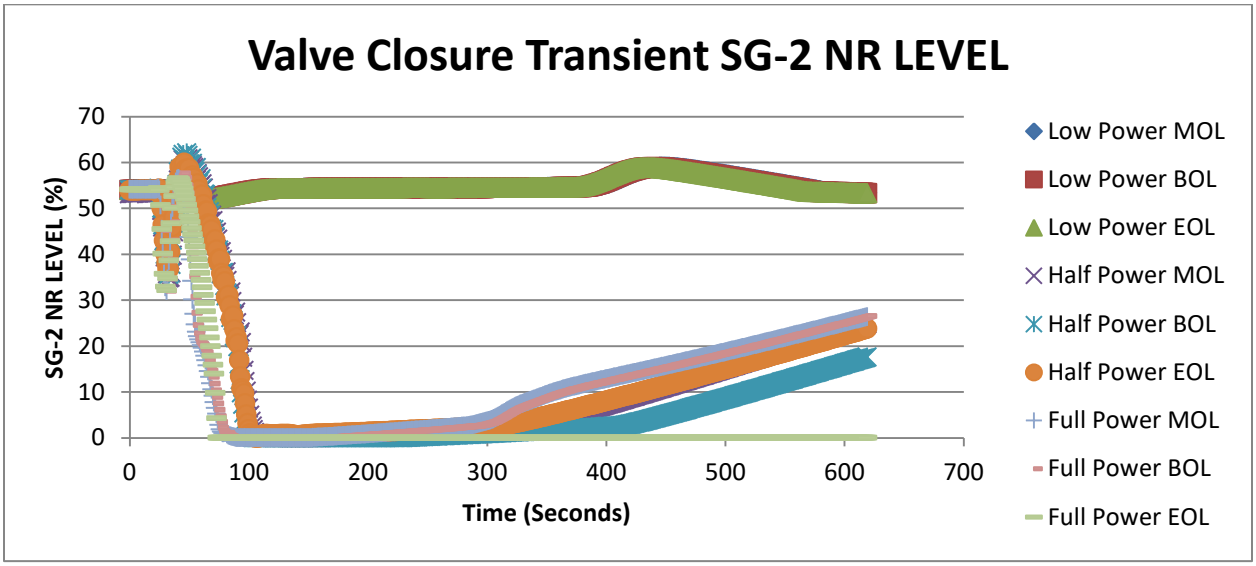
Appendix 1.18 SG-1 NR LEVEL



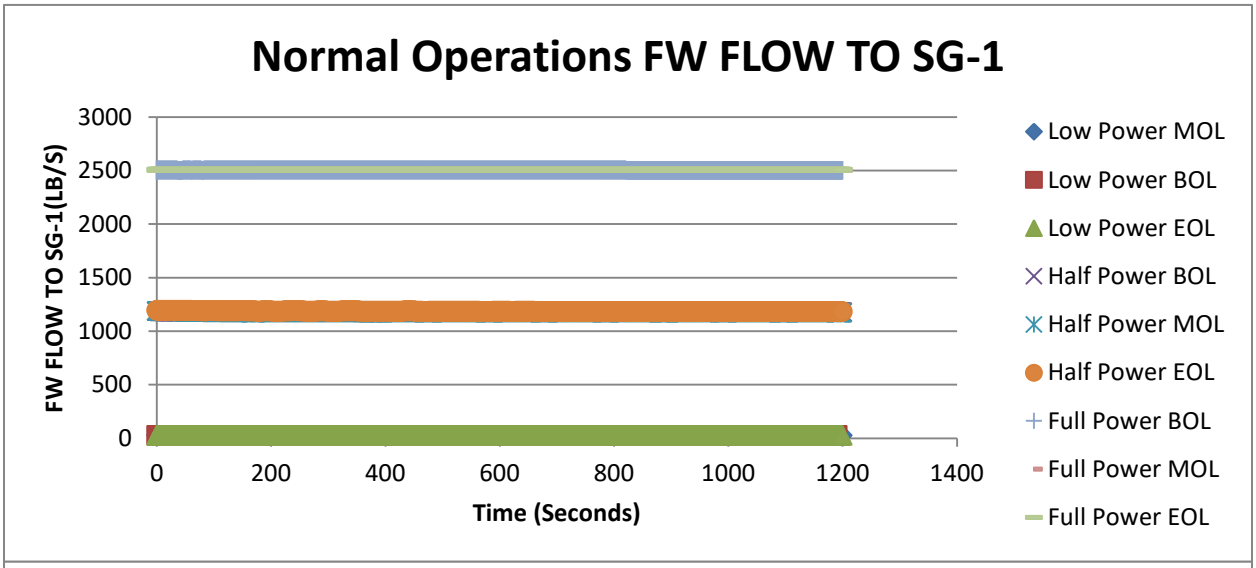


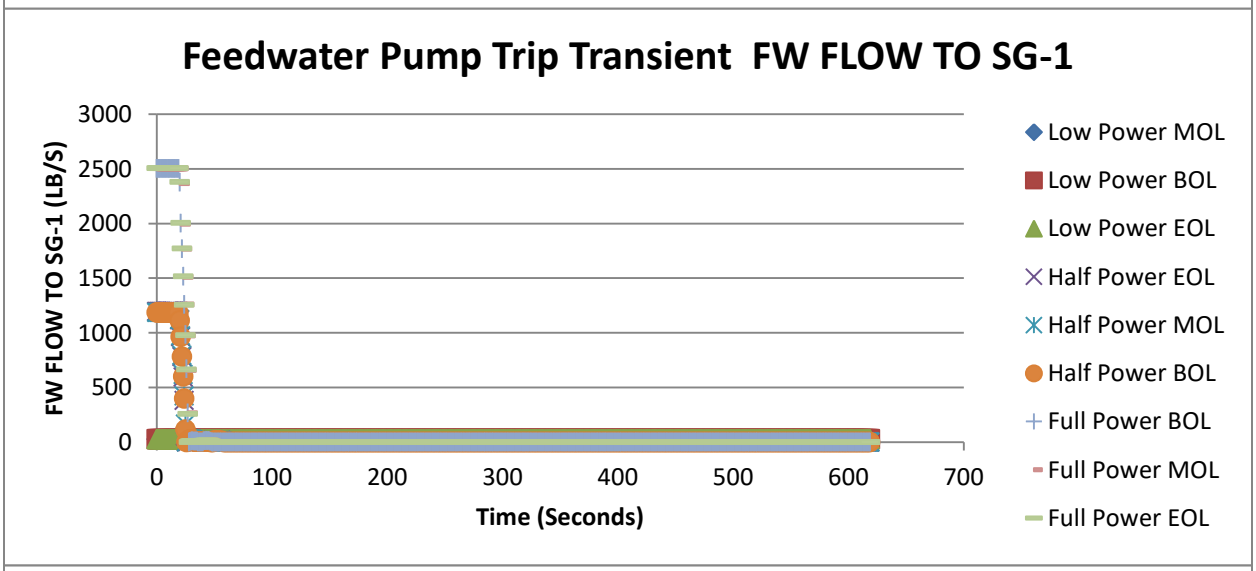


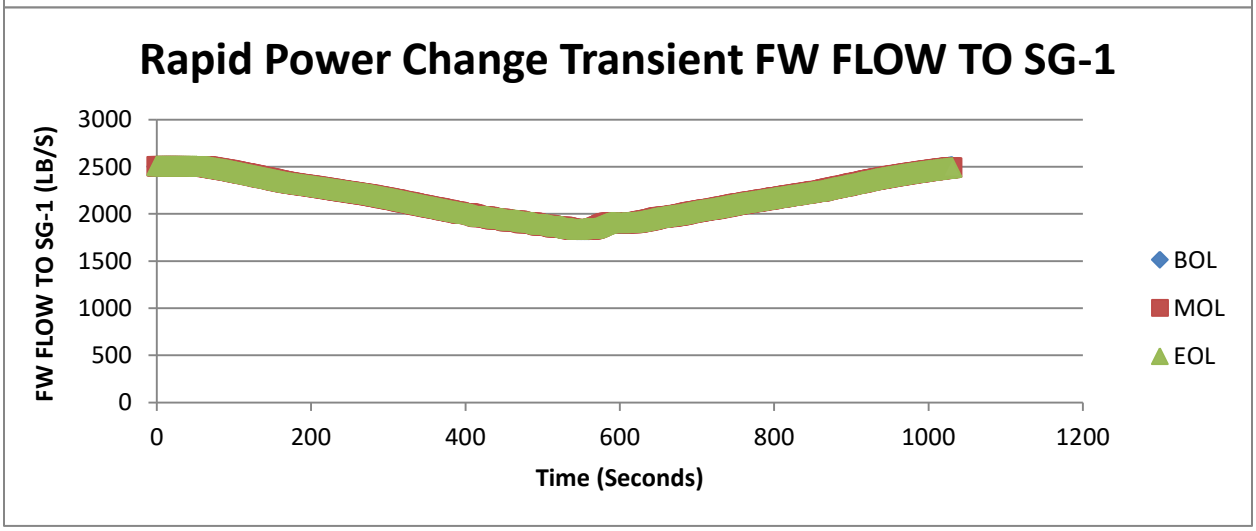

LOCA-LOOP Transient SG-2 NR LEVEL
SG-2 NR LEVEL (%)
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

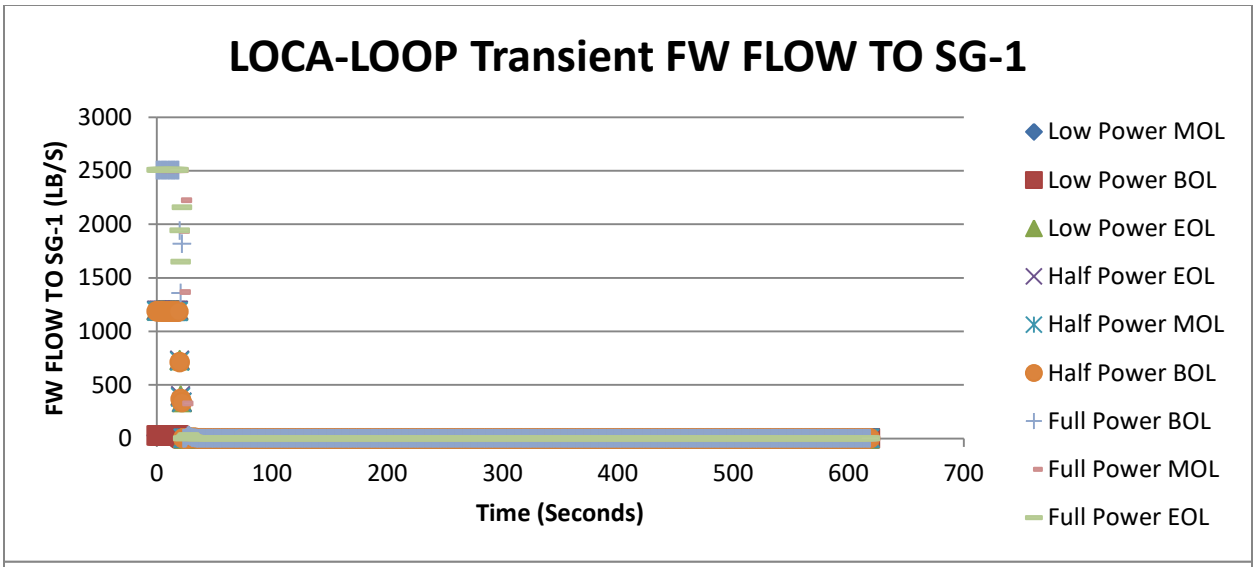

Valve Closure Transient SG-2 NR LEVEL
SG-2 NR LEVEL (%)
70
60
50
40
30
20
10
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

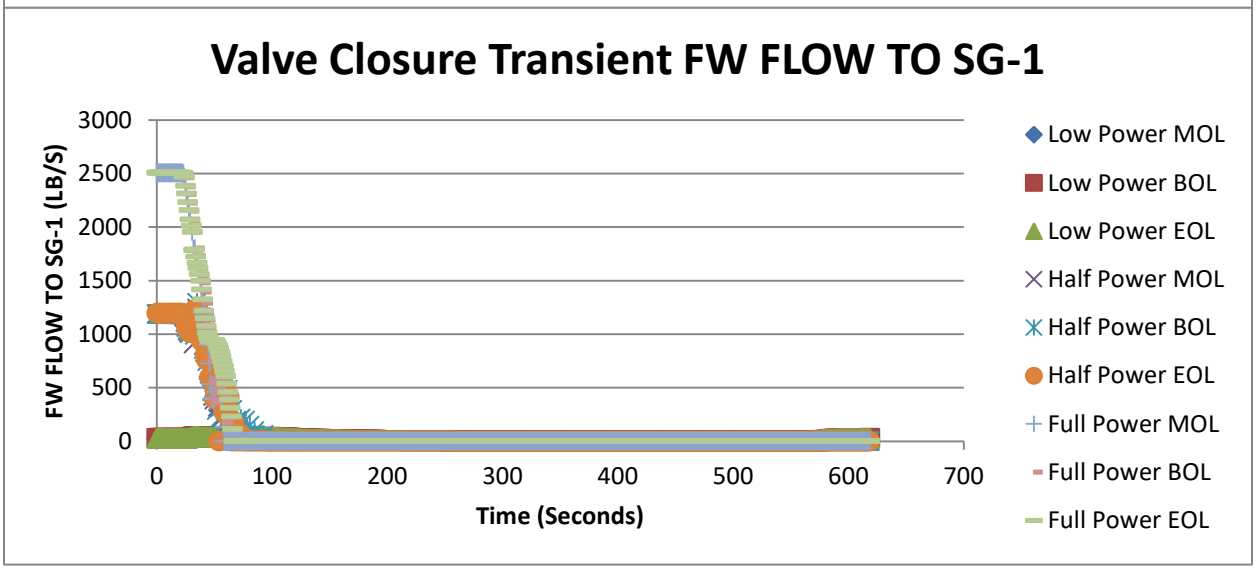
Appendix 1.19 FW FLOW TO SG-1



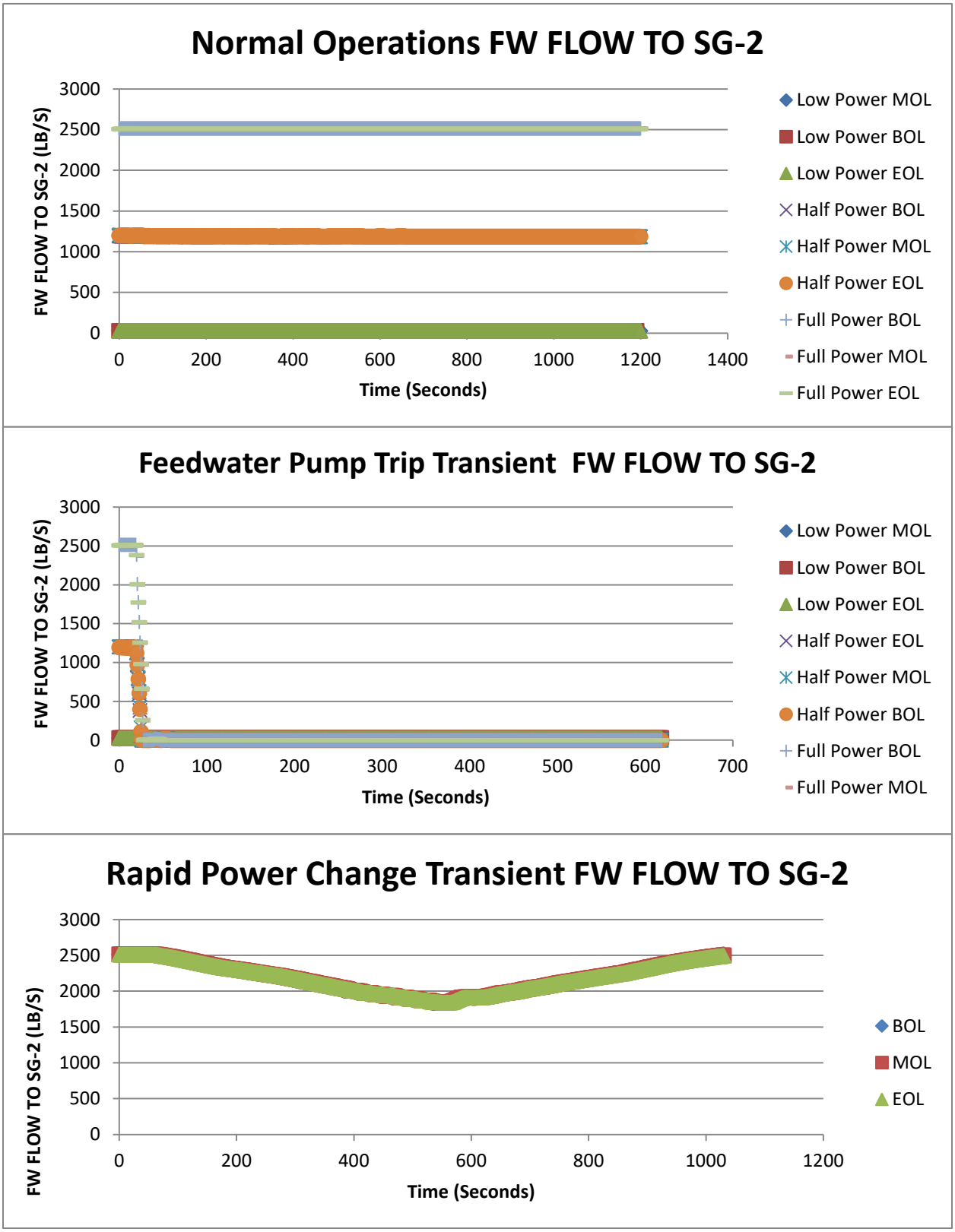


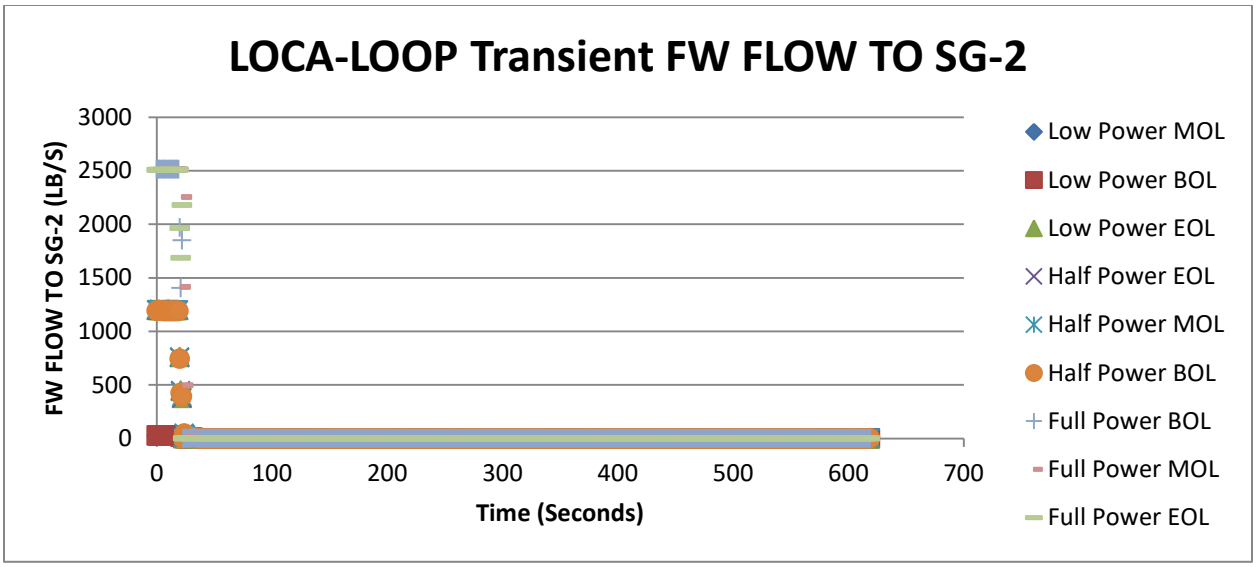


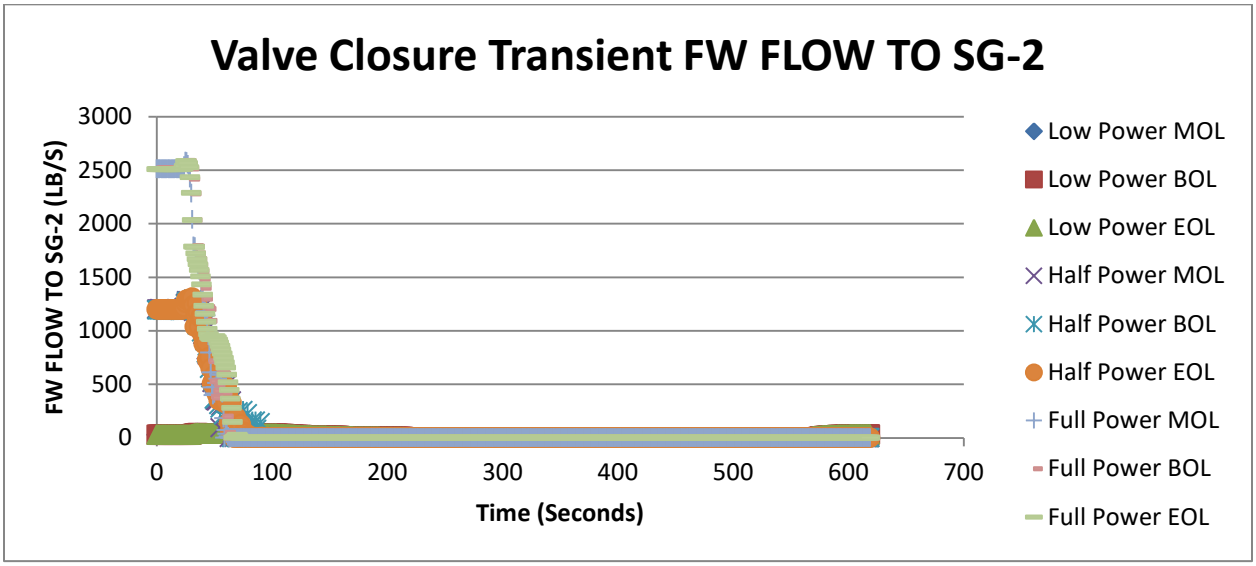

LOCA-LOOP Transient FW FLOW TO SG-1
FW FLOW TO SG-1 (LB/S)
3000
2500
2000
1500
1000
500
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL


Valve Closure Transient FW FLOW TO SG-1
FW FLOW TO SG-1 (LB/S)
3000
2500
2000
1500
1000
500
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

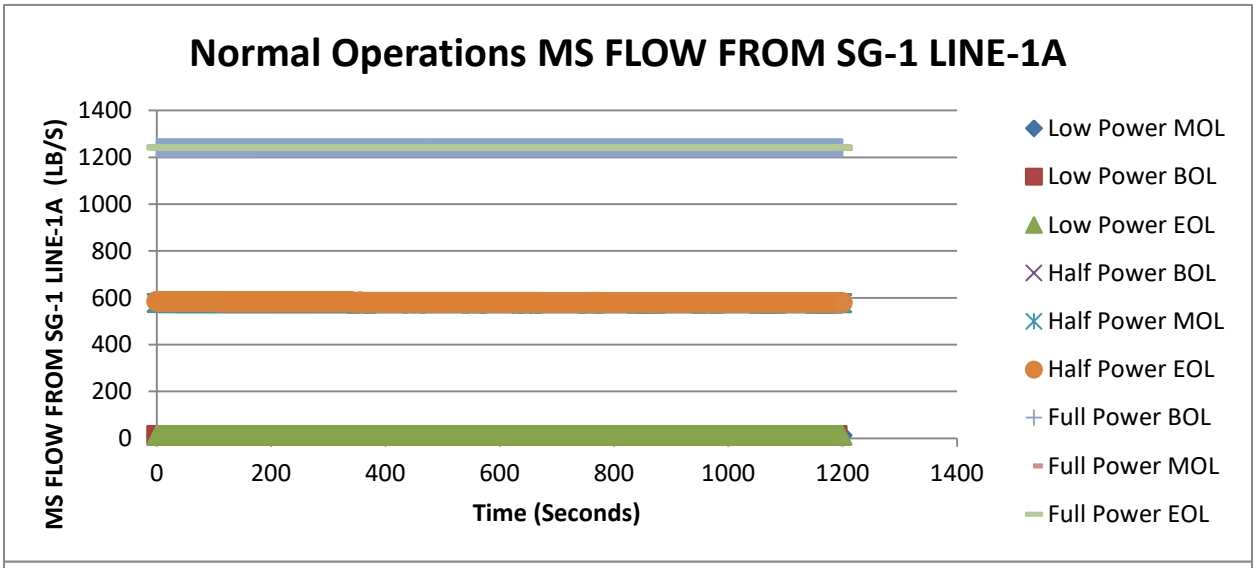
Appendix 1.20 FW FLOW TO SG-2

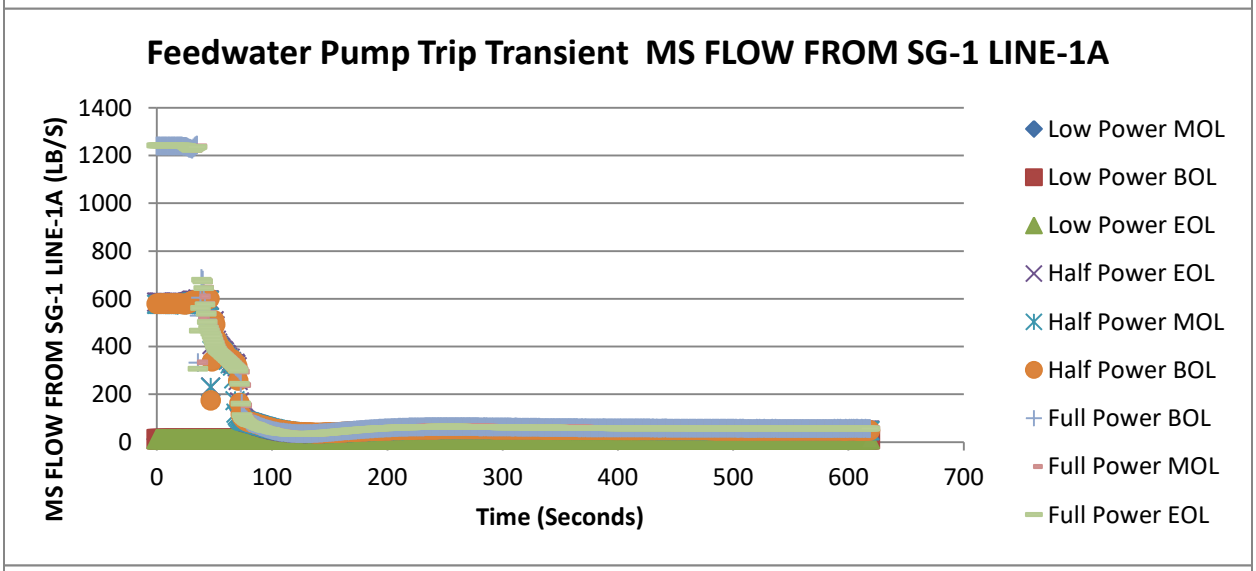


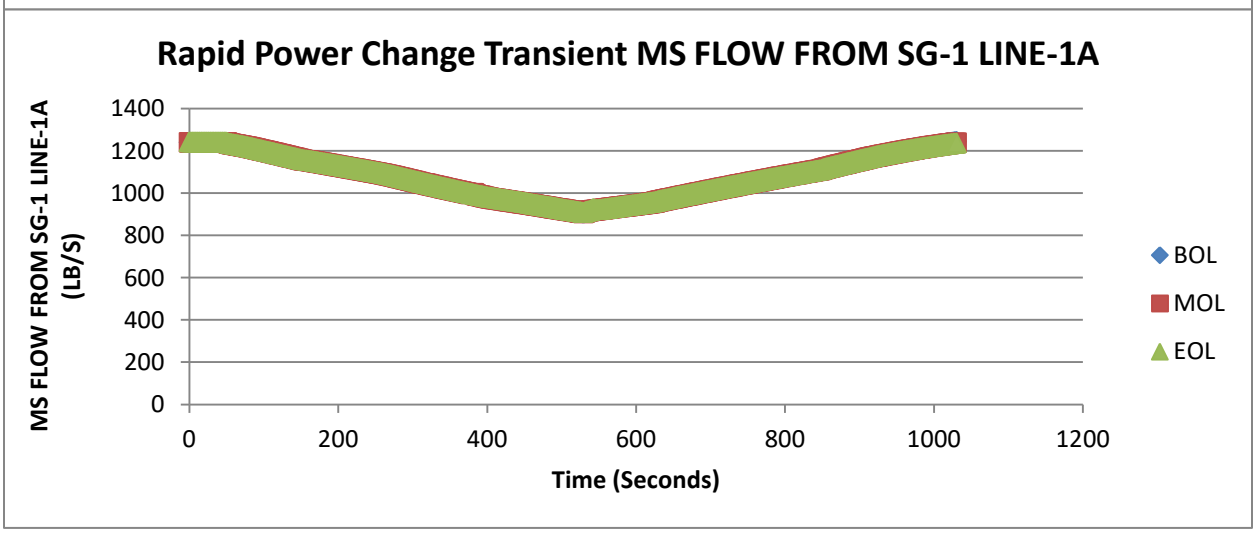

LOCA-LOOP Transient FW FLOW TO SG-2
FW FLOW TO SG-2 (LB/S)
3000
2500
2000
1500
1000
500
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

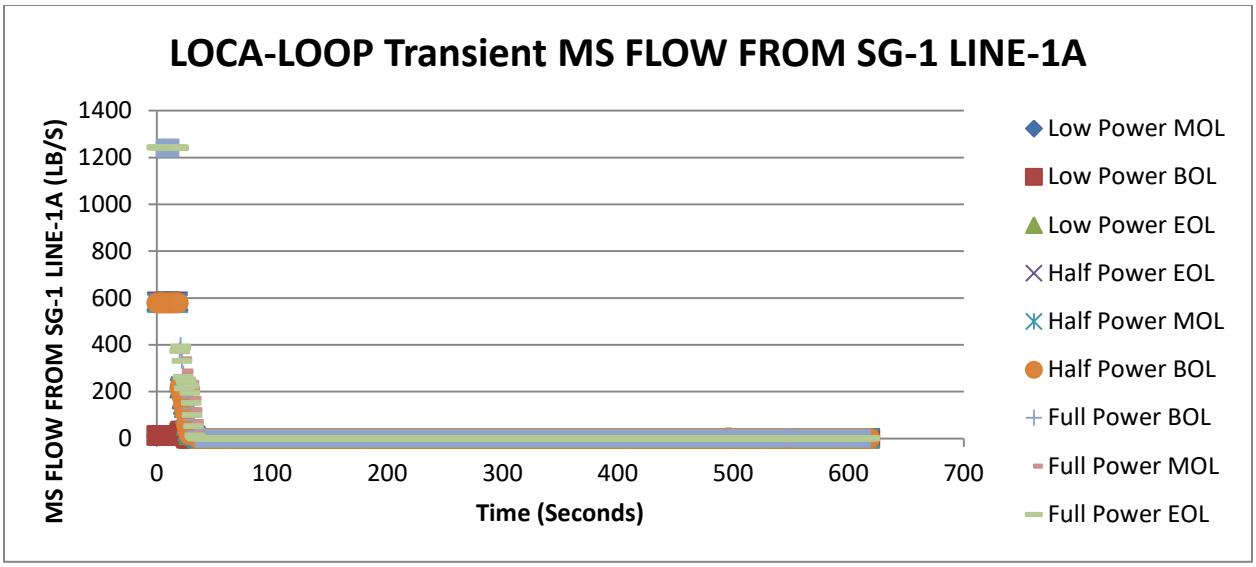

Valve Closure Transient FW FLOW TO SG-2
FW FLOW TO SG-2 (LB/S)
3000
2500
2000
1500
1000
500
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

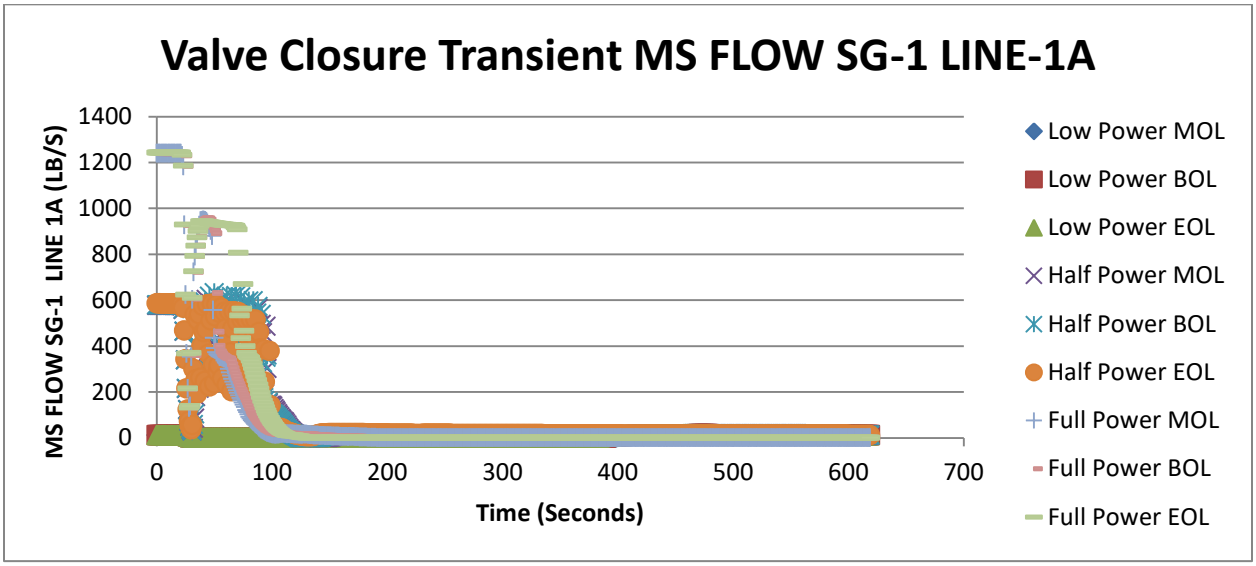
Appendix 1.21 MS FLOW FROM SG-1 LINE-1A



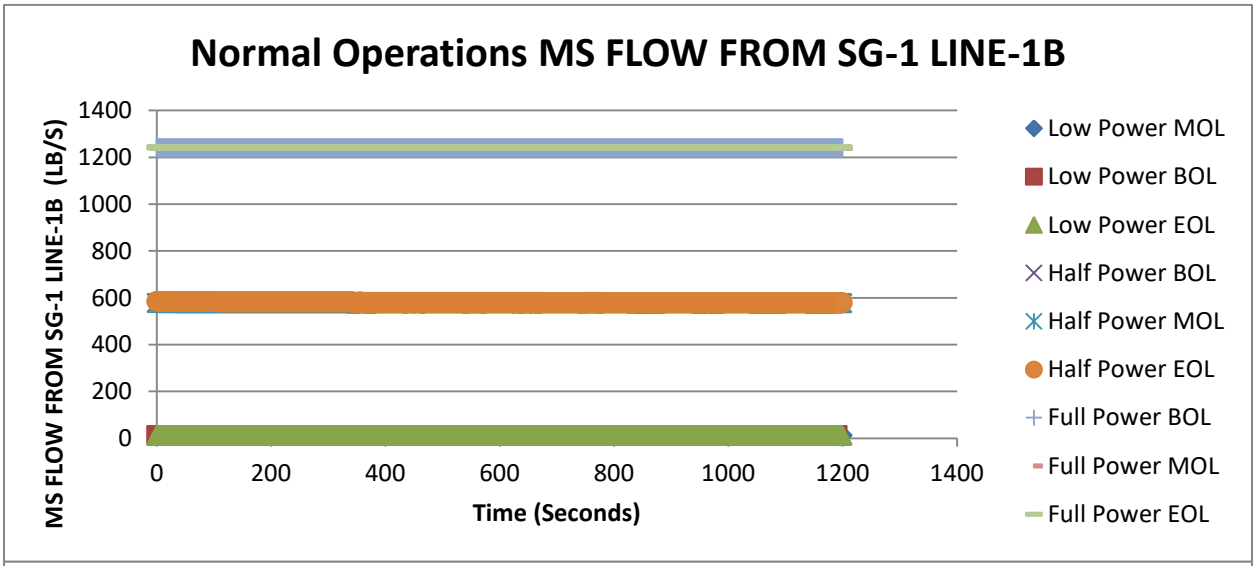


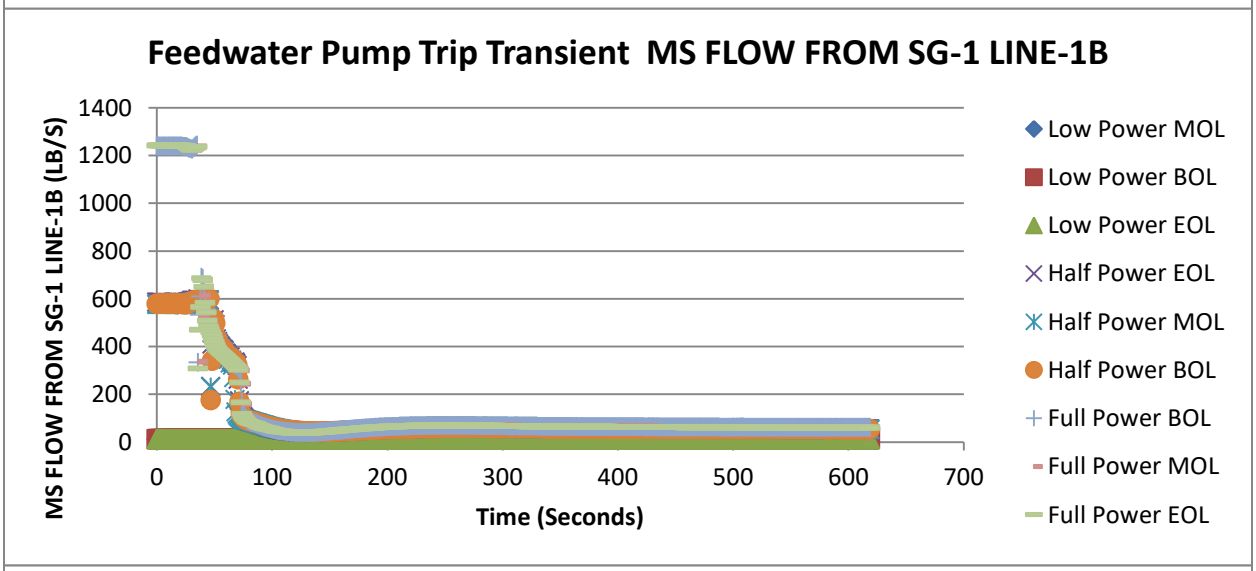


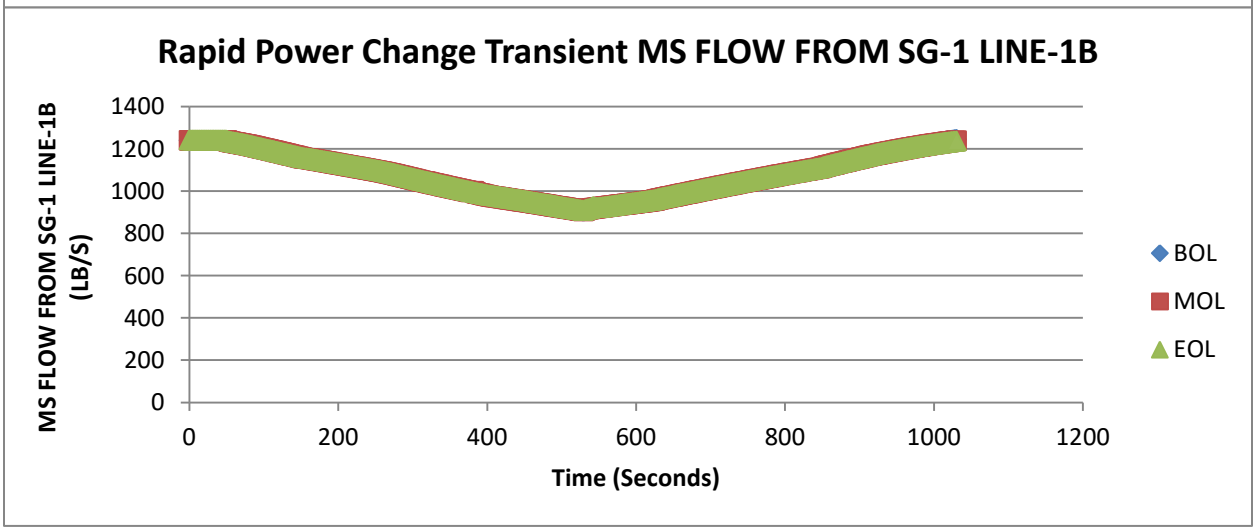

LOCA-LOOP Transient MS FLOW FROM SG-1 LINE-1A
MS FLOW FROM SG-1 LINE-1A (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

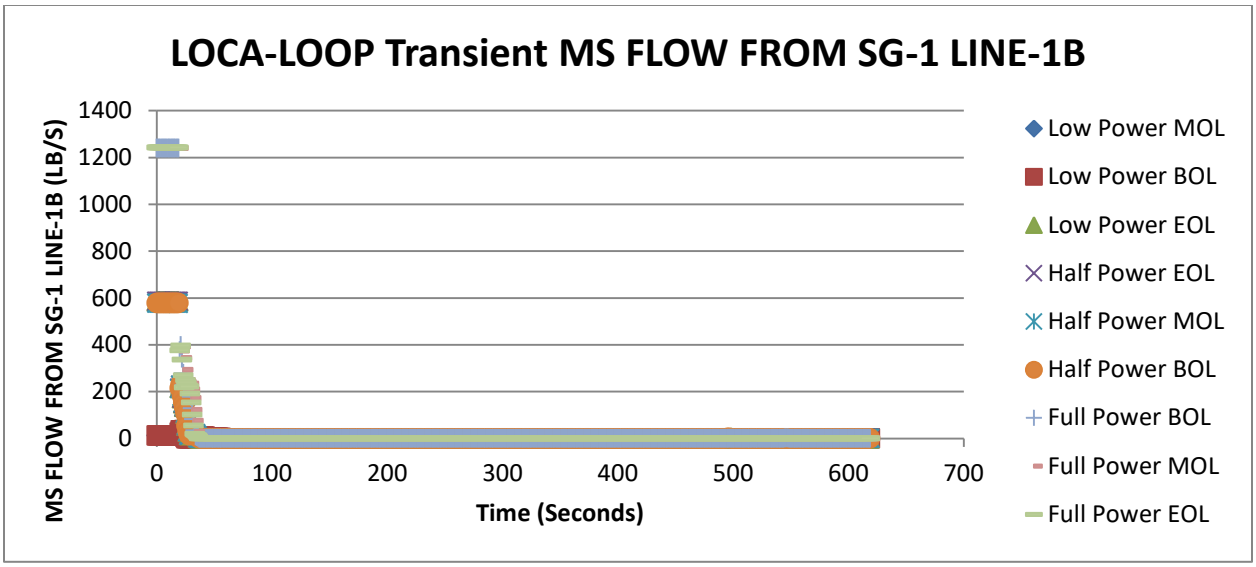

Valve Closure Transient MS FLOW SG-1 LINE-1A
MS FLOW SG-1 LINE 1A (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

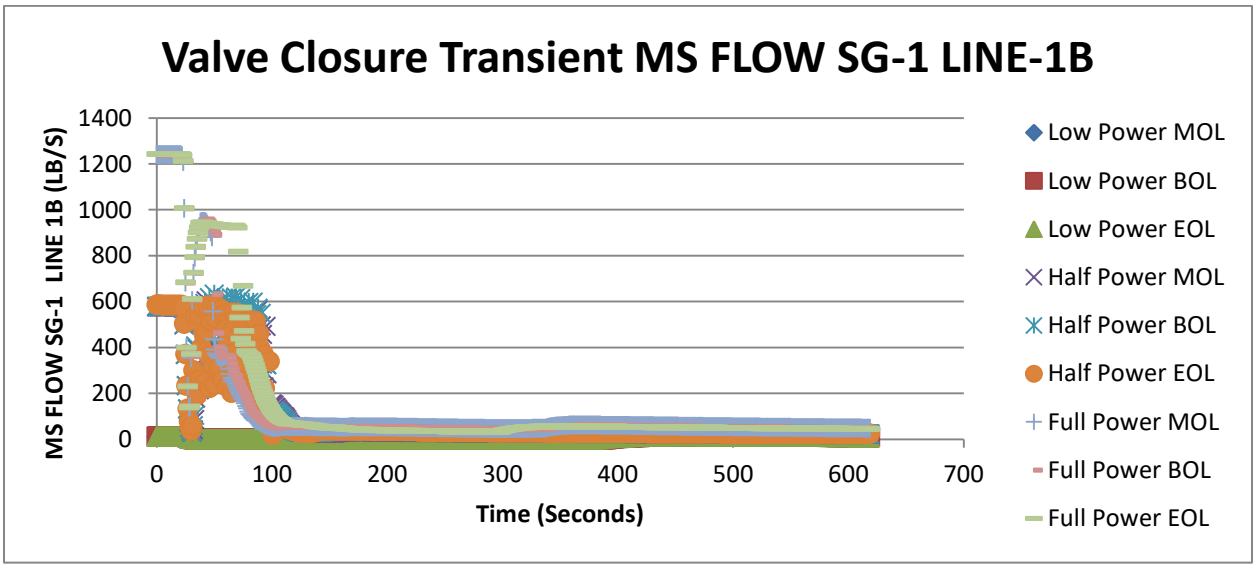
Appendix 1.22 MS FLOW FROM SG-1 LINE-1B



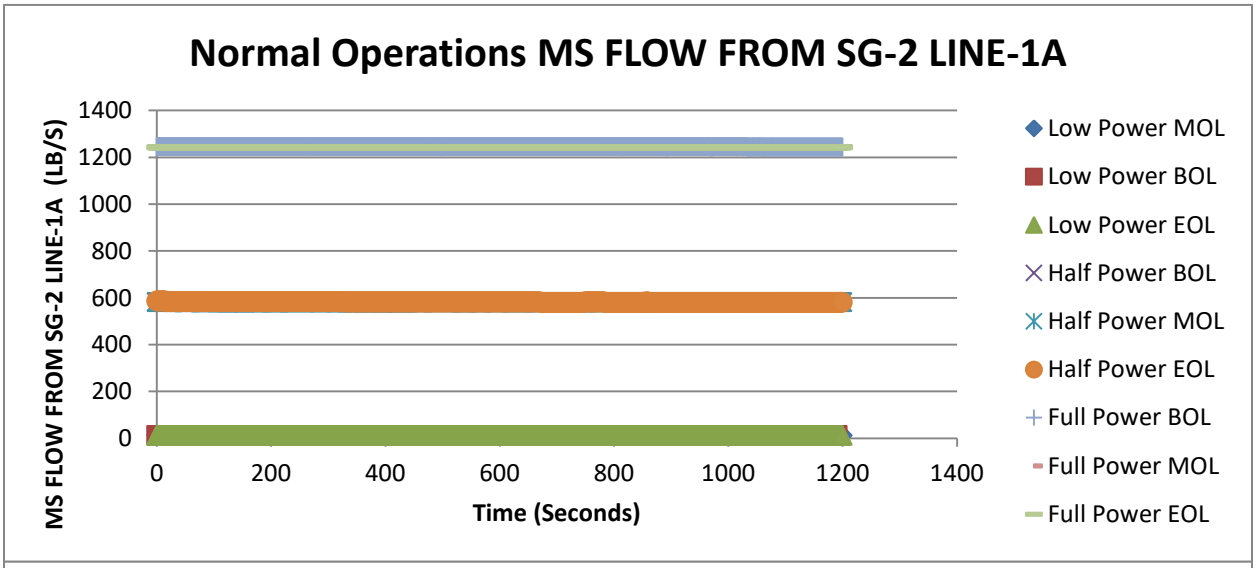


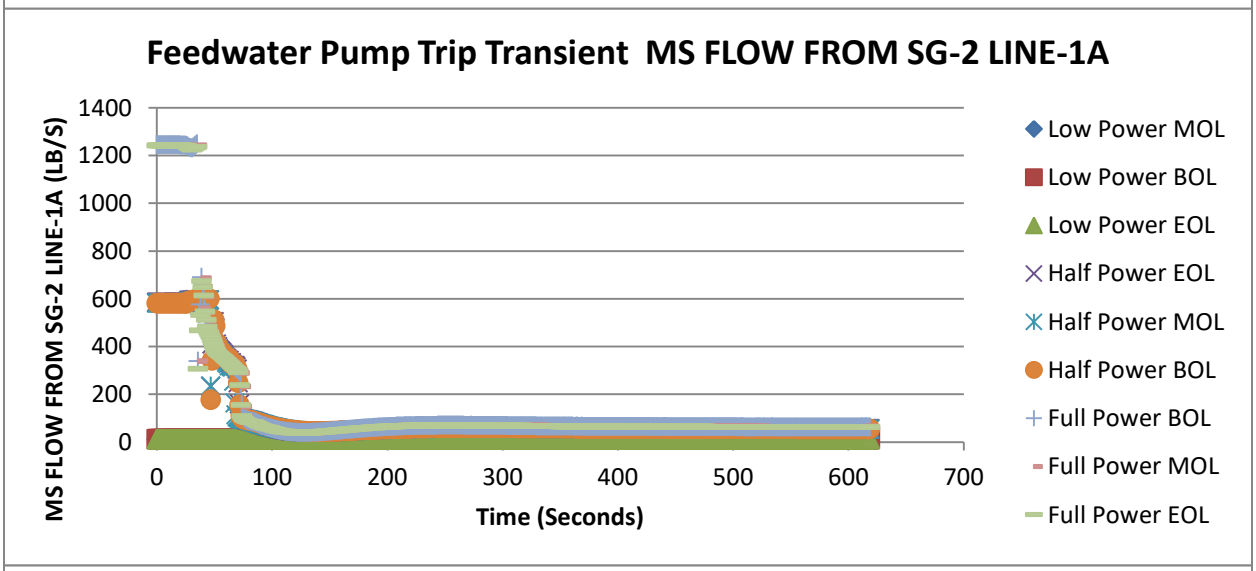


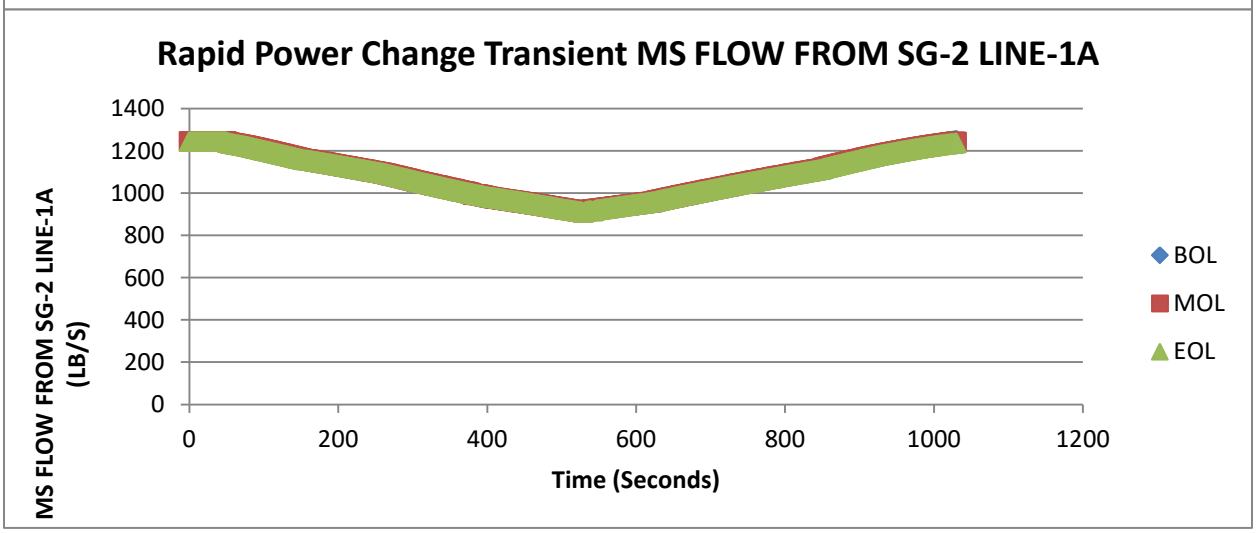

LOCA-LOOP Transient MS FLOW FROM SG-1 LINE-1B
MS FLOW FROM SG-1 LINE-1B (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

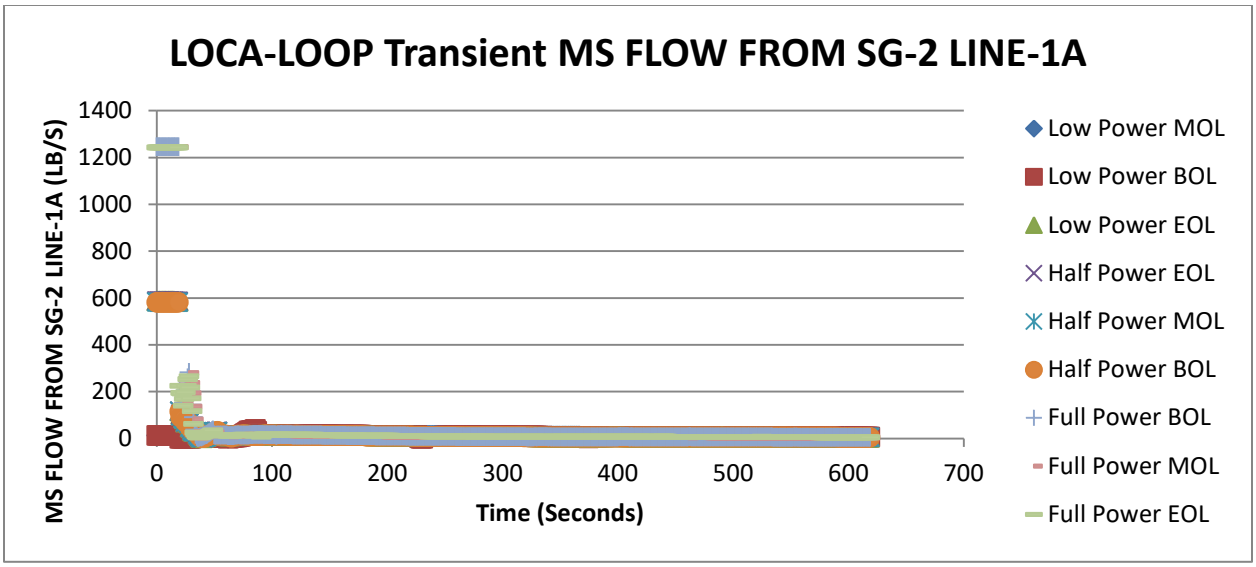

Valve Closure Transient MS FLOW SG-1 LINE-1B
MS FLOW SG-1 LINE 1B (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

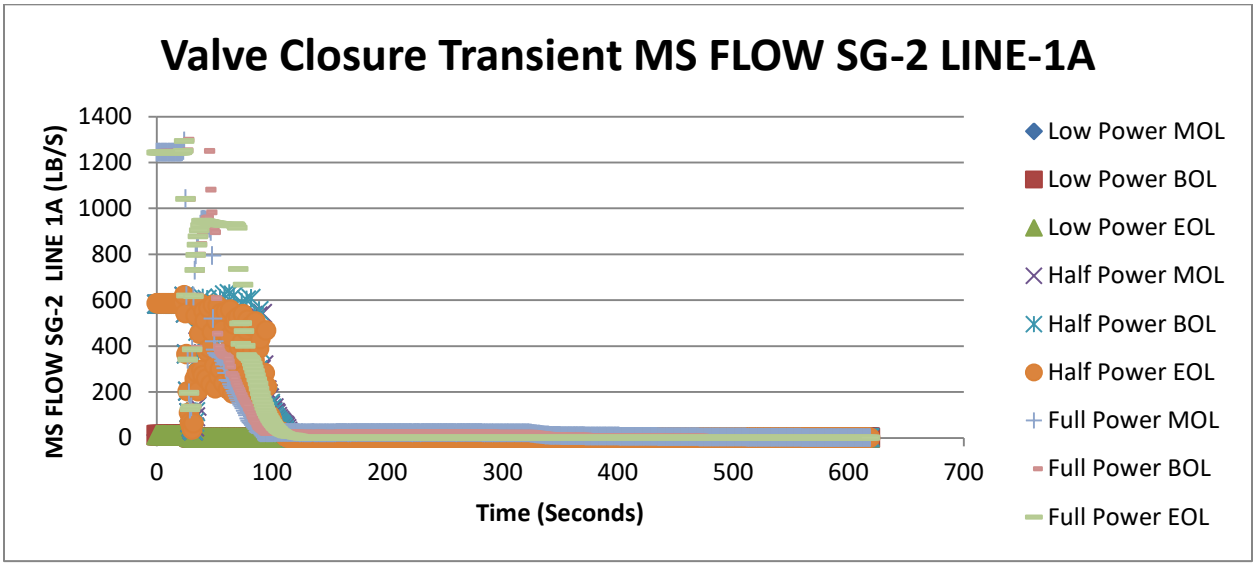
Appendix 1.23 MS FLOW FROM SG-2 LINE-1A



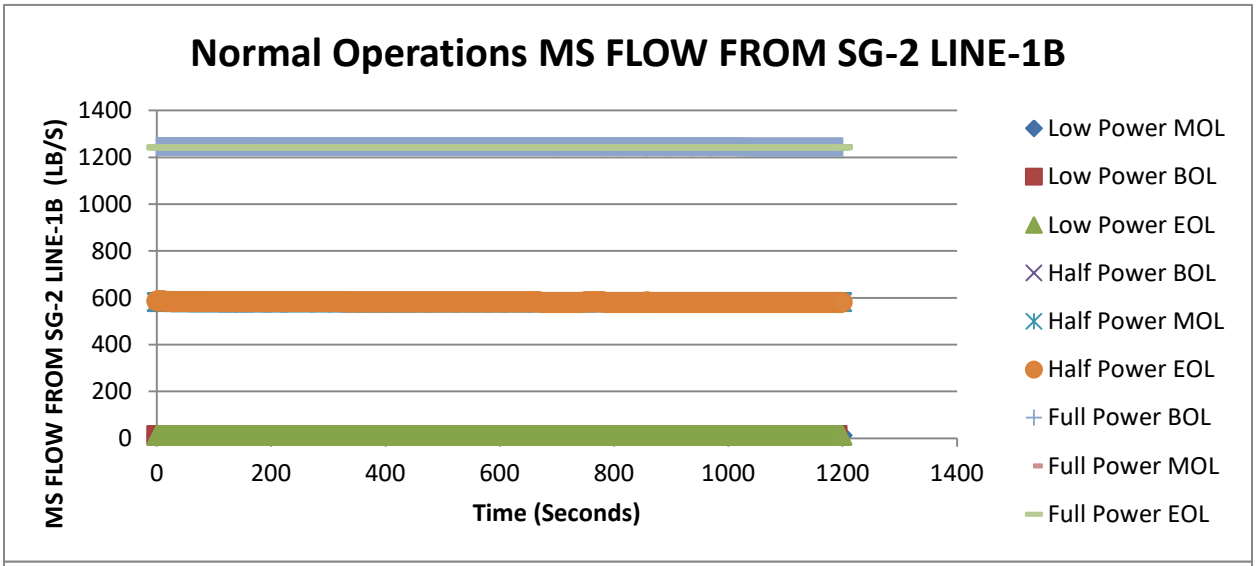


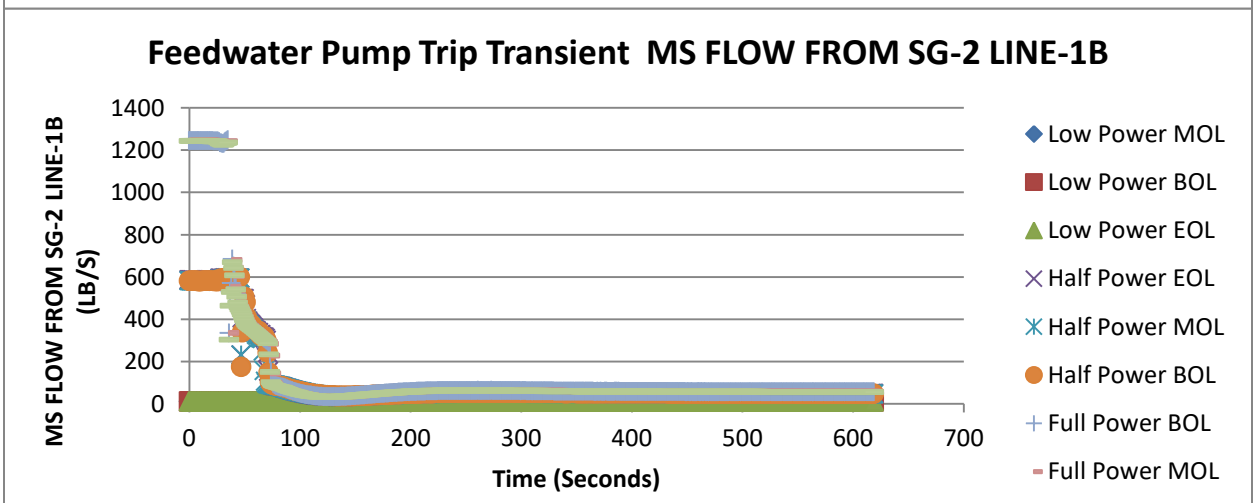


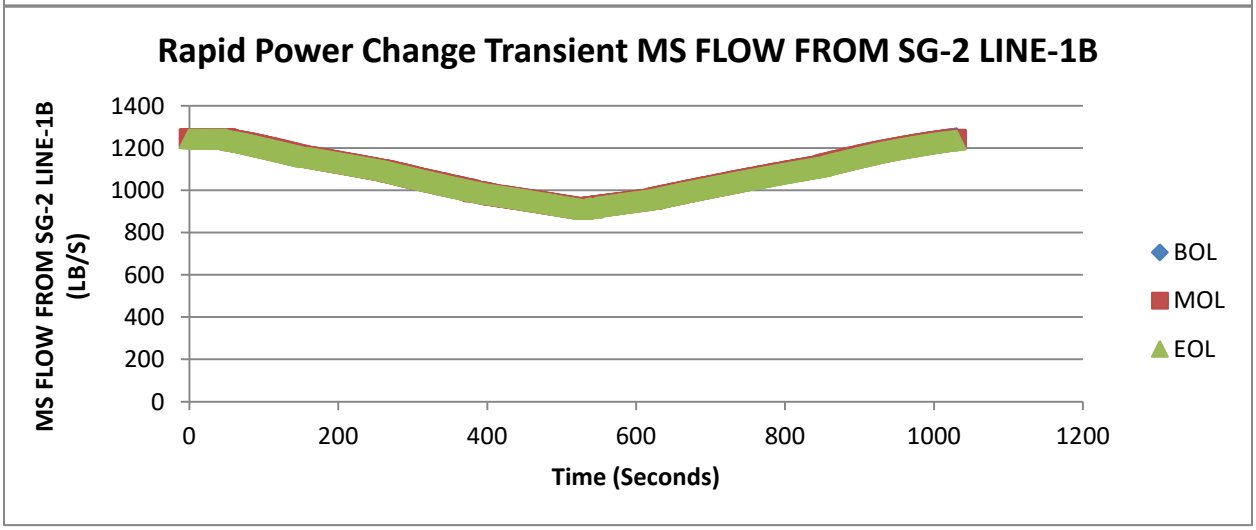

LOCA-LOOP Transient MS FLOW FROM SG-2 LINE-1A
MS FLOW FROM SG-2 LINE-1A (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

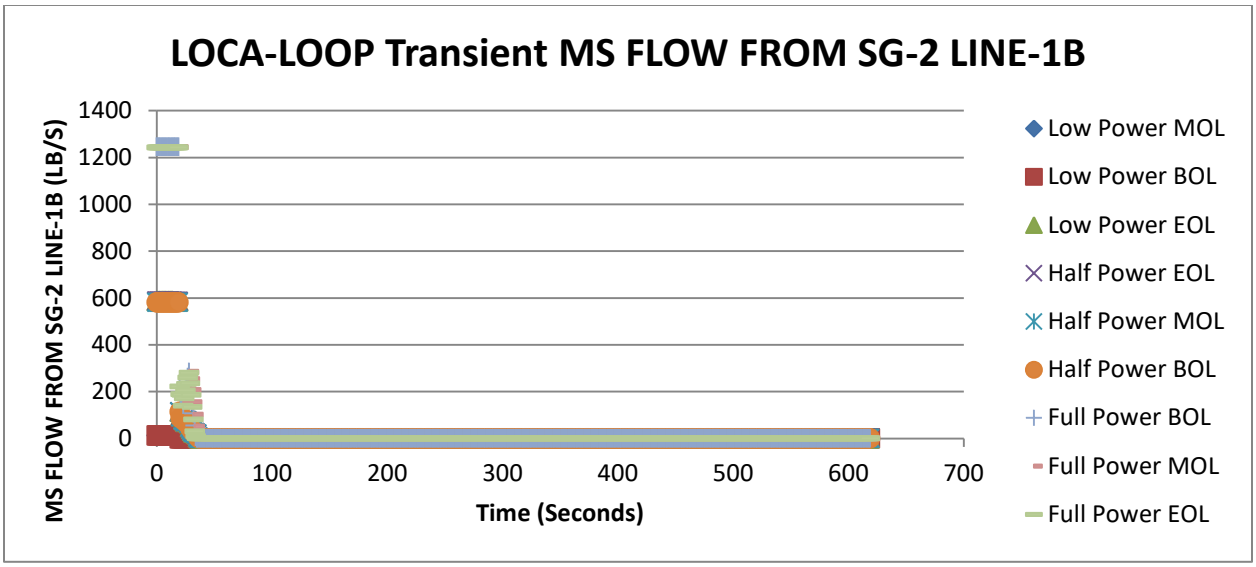

Valve Closure Transient MS FLOW SG-2 LINE-1A
MS FLOW SG-2 LINE 1A (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

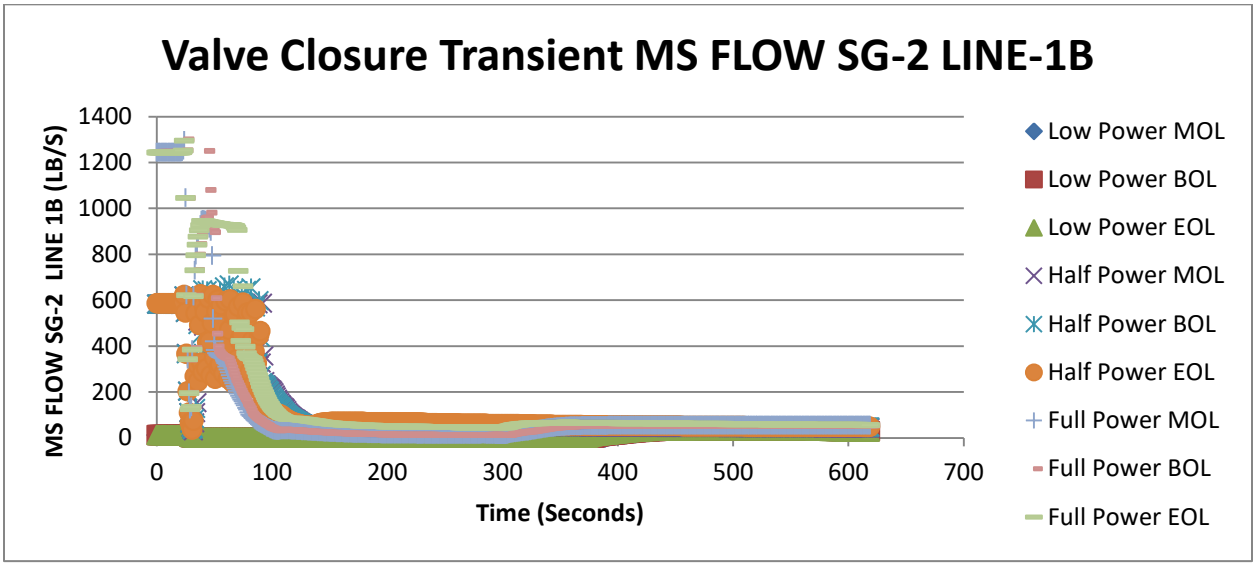
Appendix 1.24 MS FLOW FROM SG-2 LINE-1B



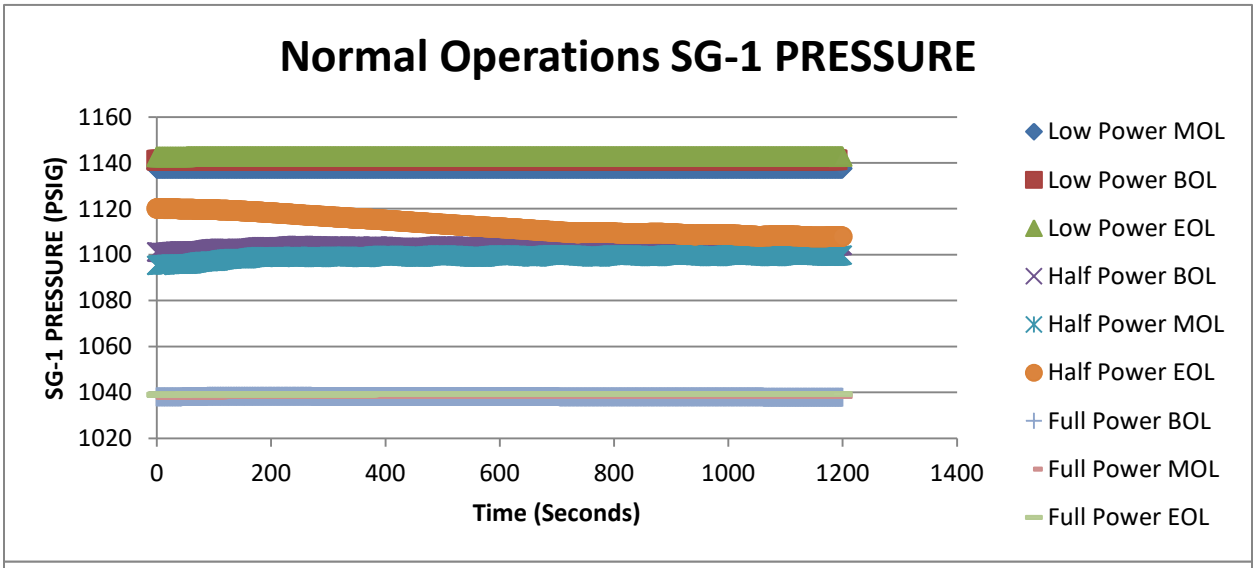


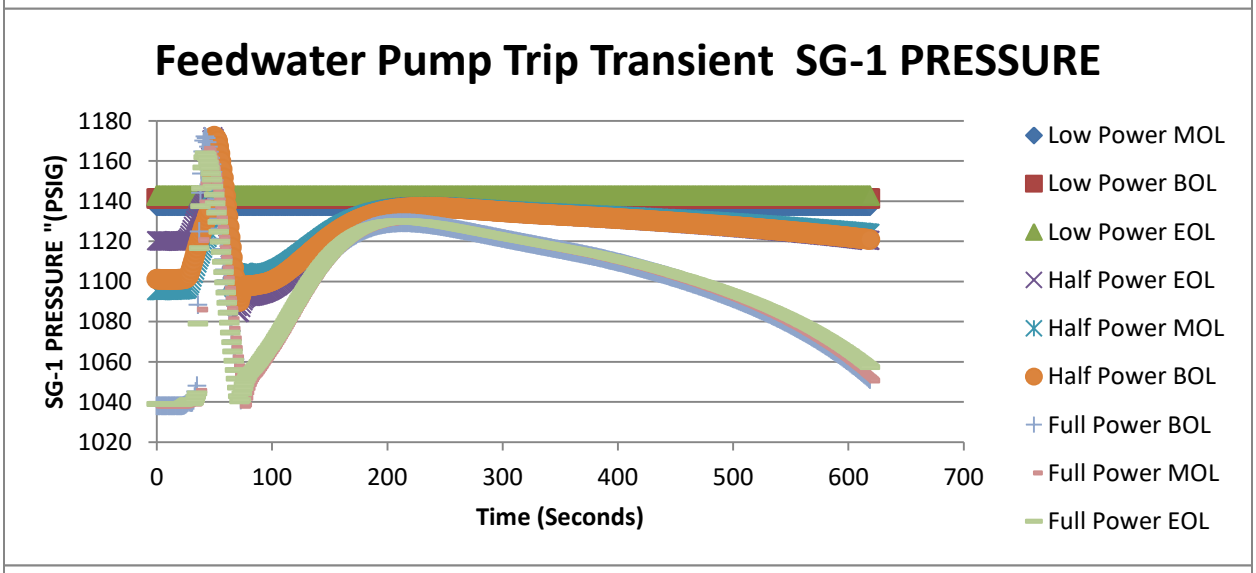


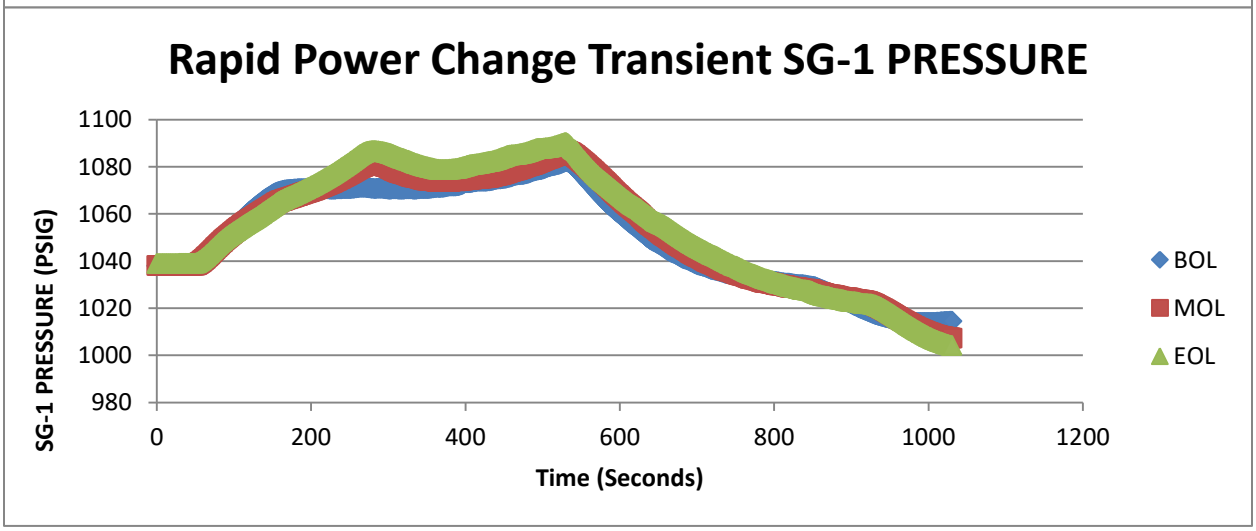

LOCA-LOOP Transient MS FLOW FROM SG-2 LINE-1B
MS FLOW FROM SG-2 LINE-1B (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

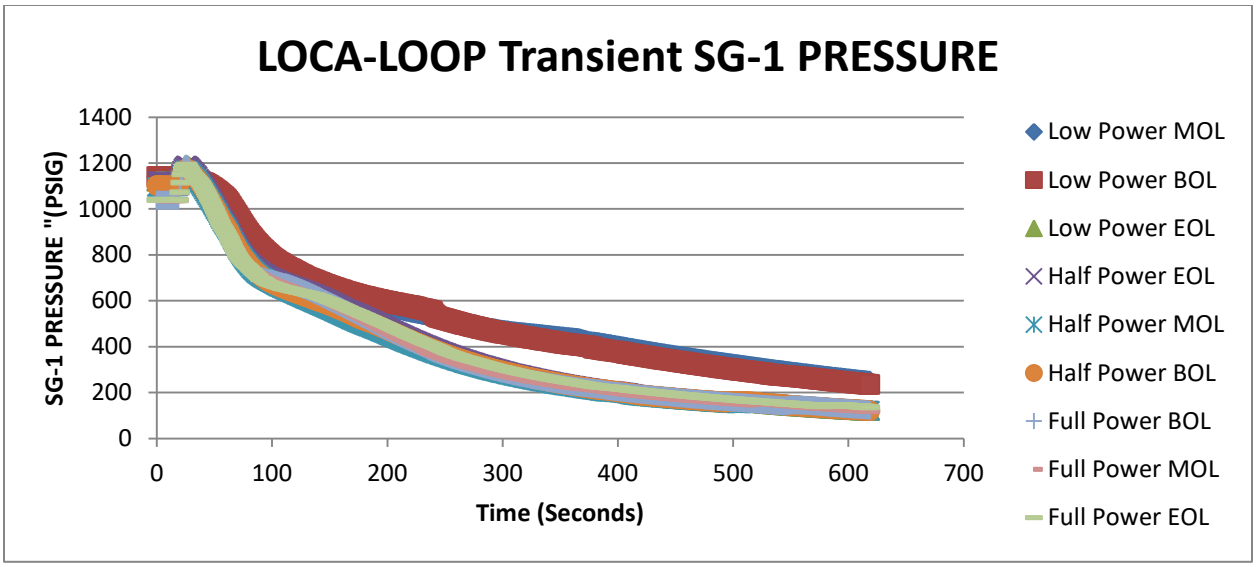

Valve Closure Transient MS FLOW SG-2 LINE-1B
MS FLOW SG-2 LINE 1B (LB/S)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

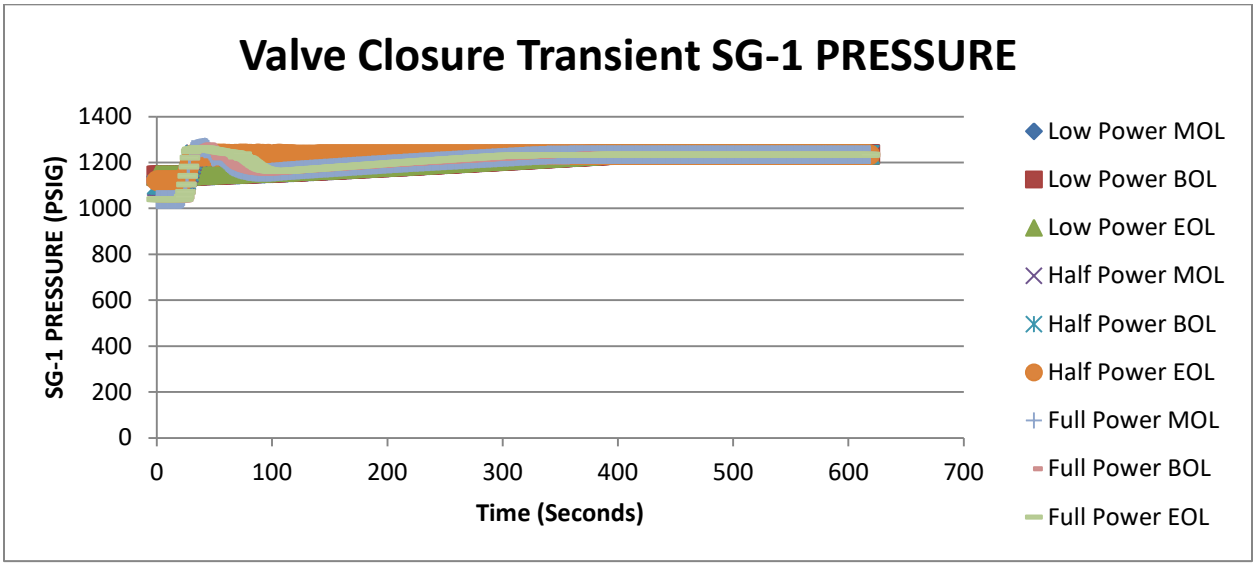
Appendix 1.25 SG-1 PRESSURE



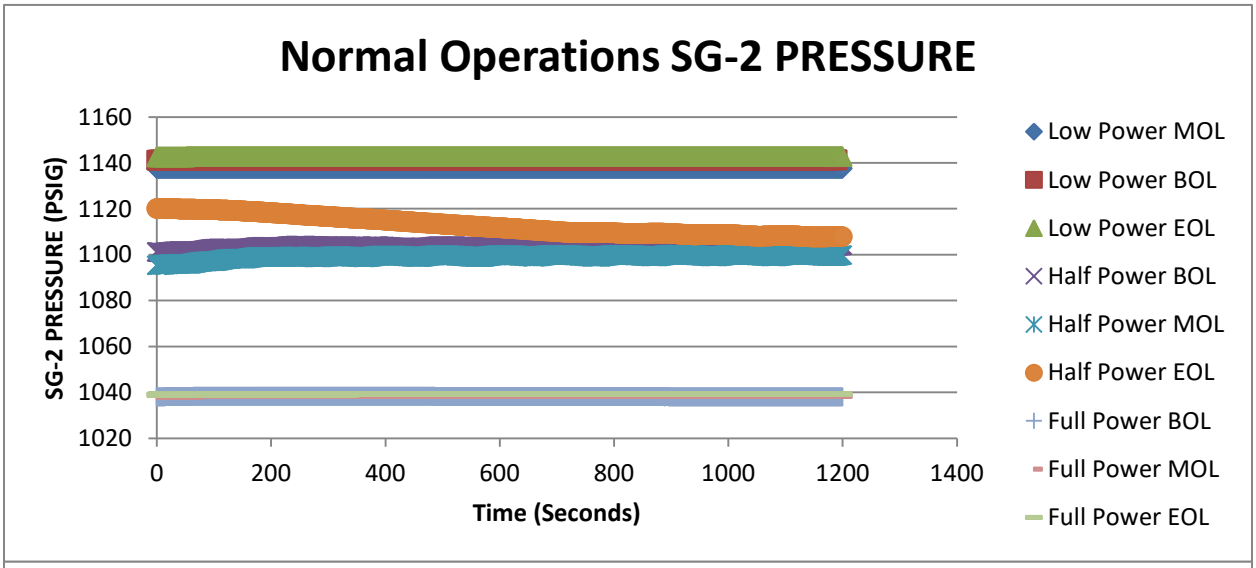


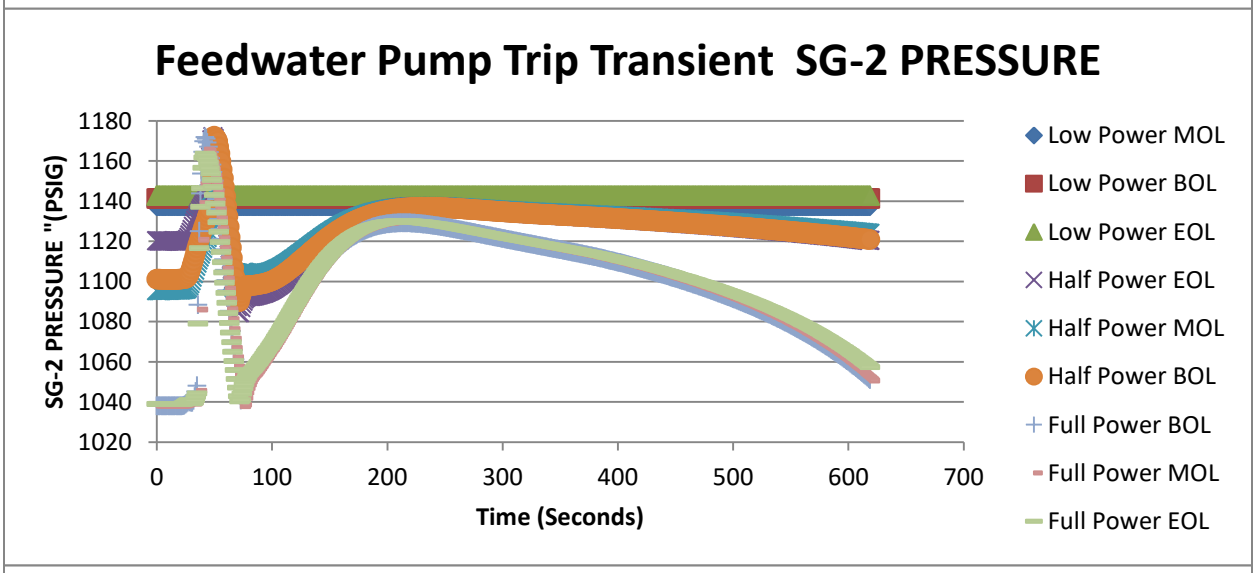


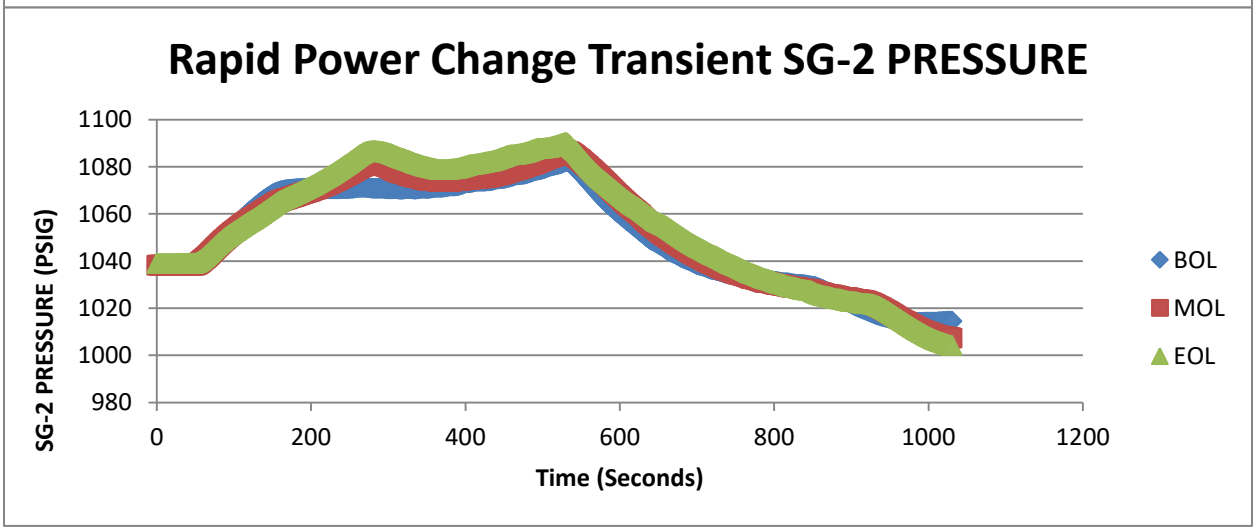

LOCA-LOOP Transient SG-1 PRESSURE
SG-1 PRESSURE "(PSIG)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

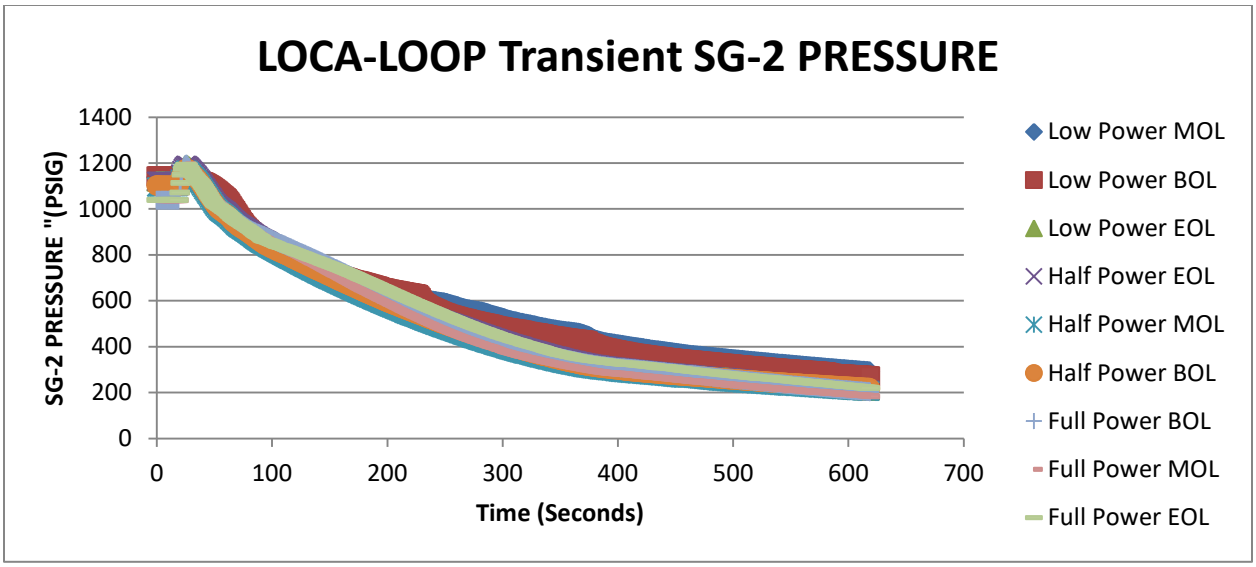

Valve Closure Transient SG-1 PRESSURE
SG-1 PRESSURE (PSIG)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

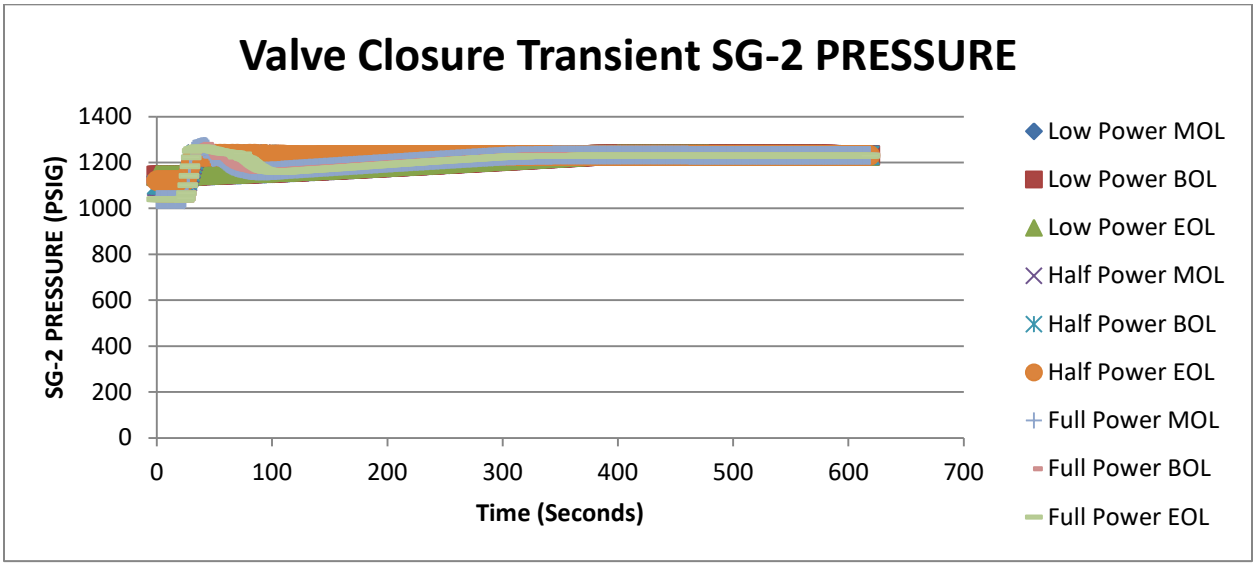
Appendix 1.26 SG-2 PRESSURE



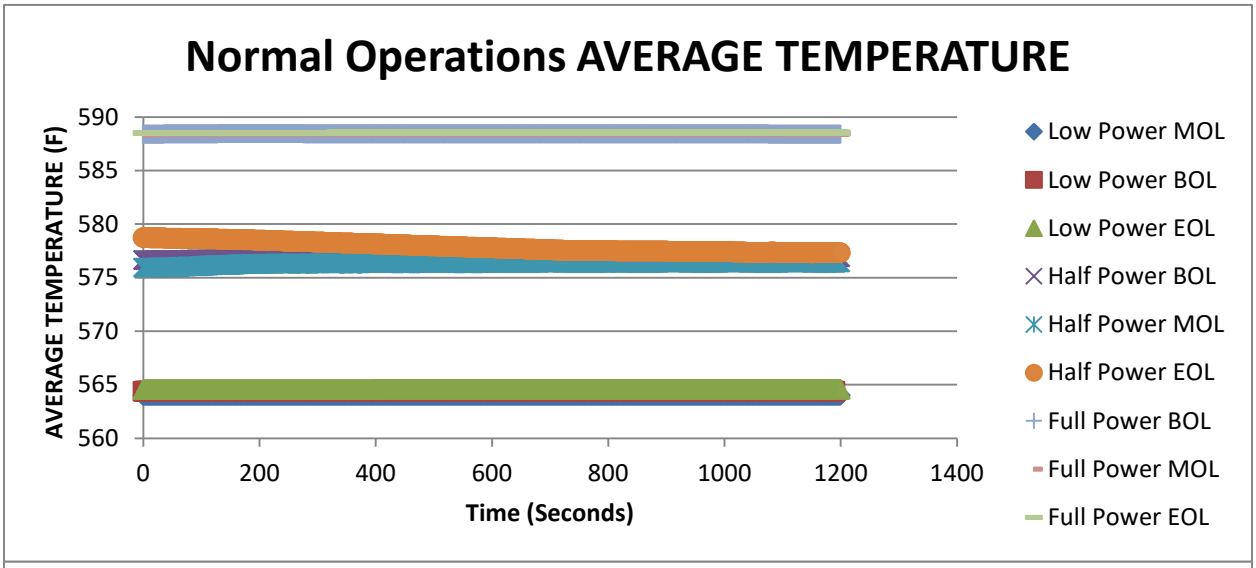


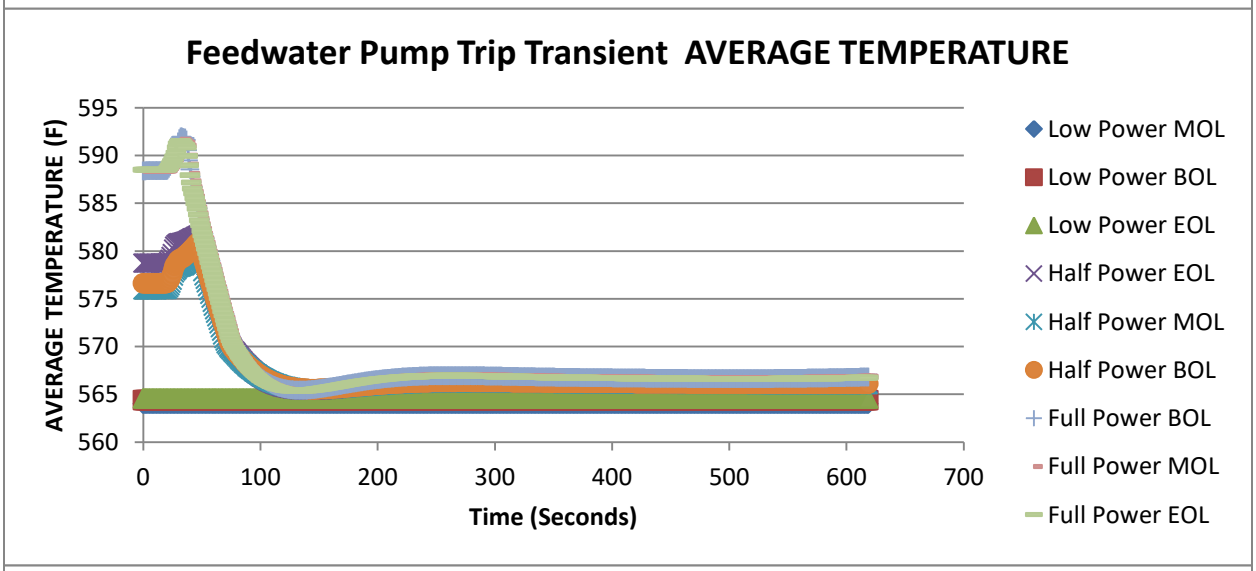


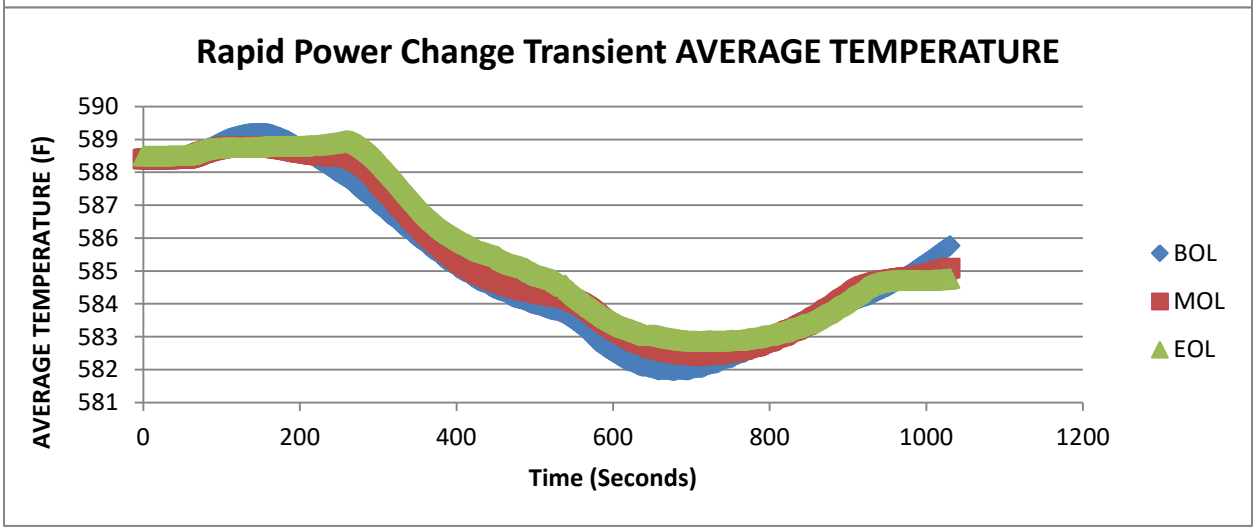

LOCA-LOOP Transient SG-2 PRESSURE
SG-2 PRESSURE "(PSIG)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

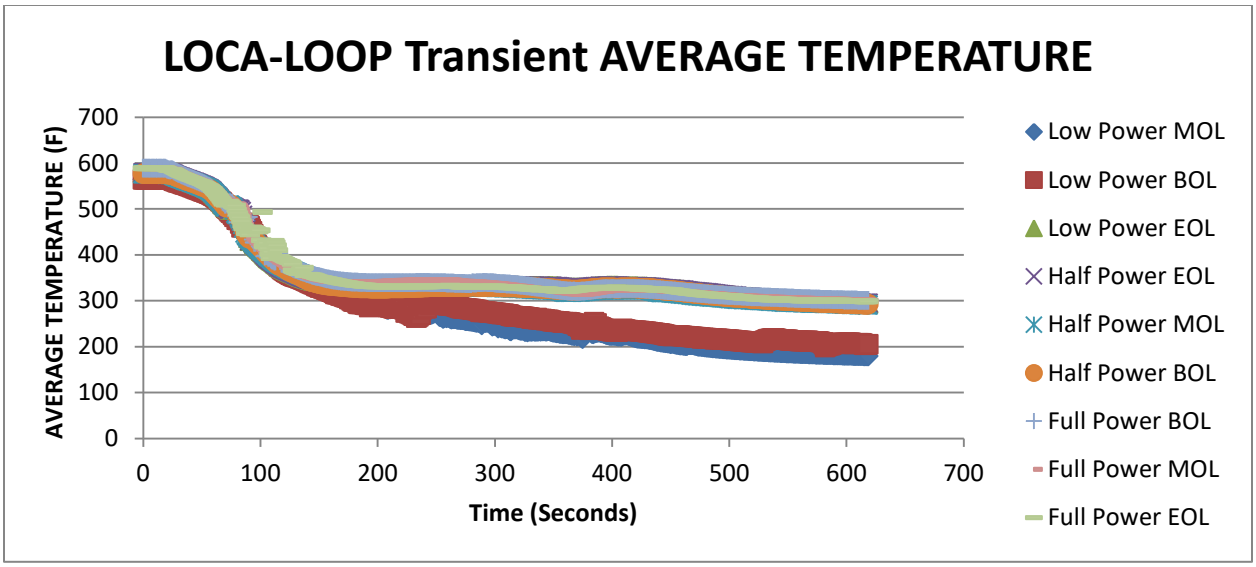

Valve Closure Transient SG-2 PRESSURE
SG-2 PRESSURE (PSIG)
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

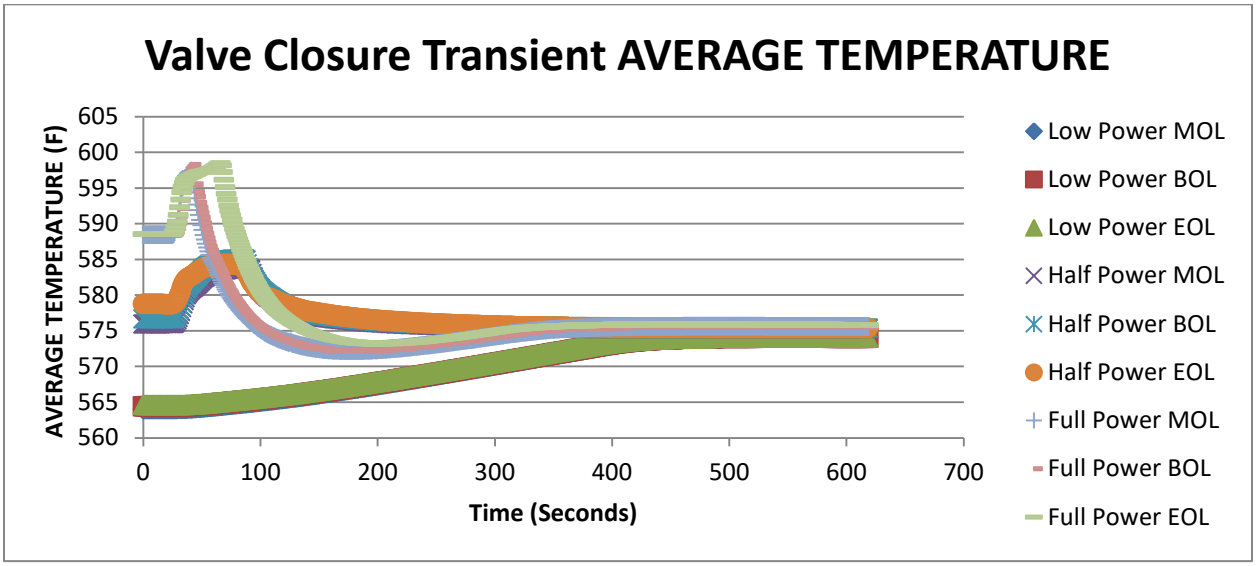
Appendix 1.27 AVERAGE TEMPERATURE



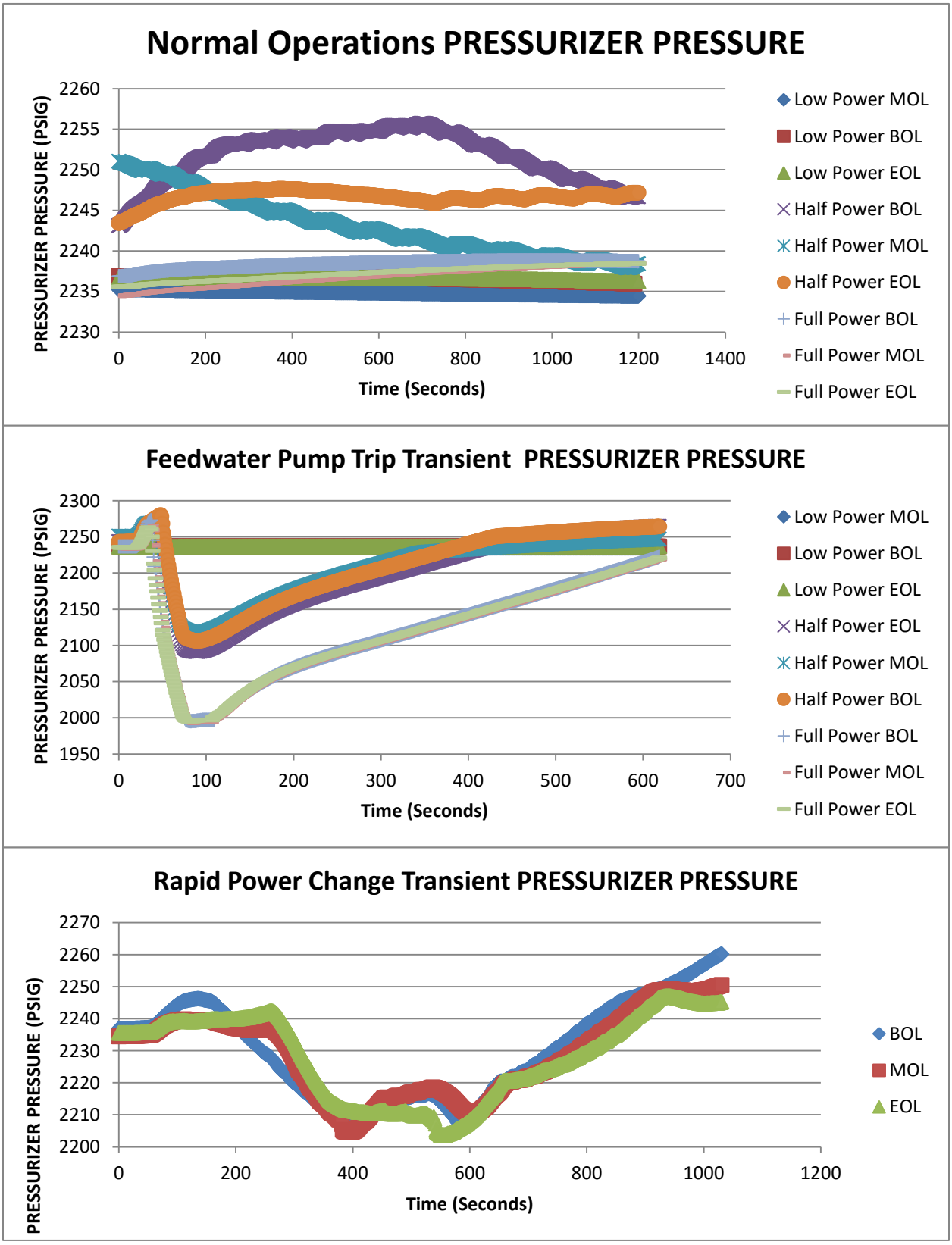


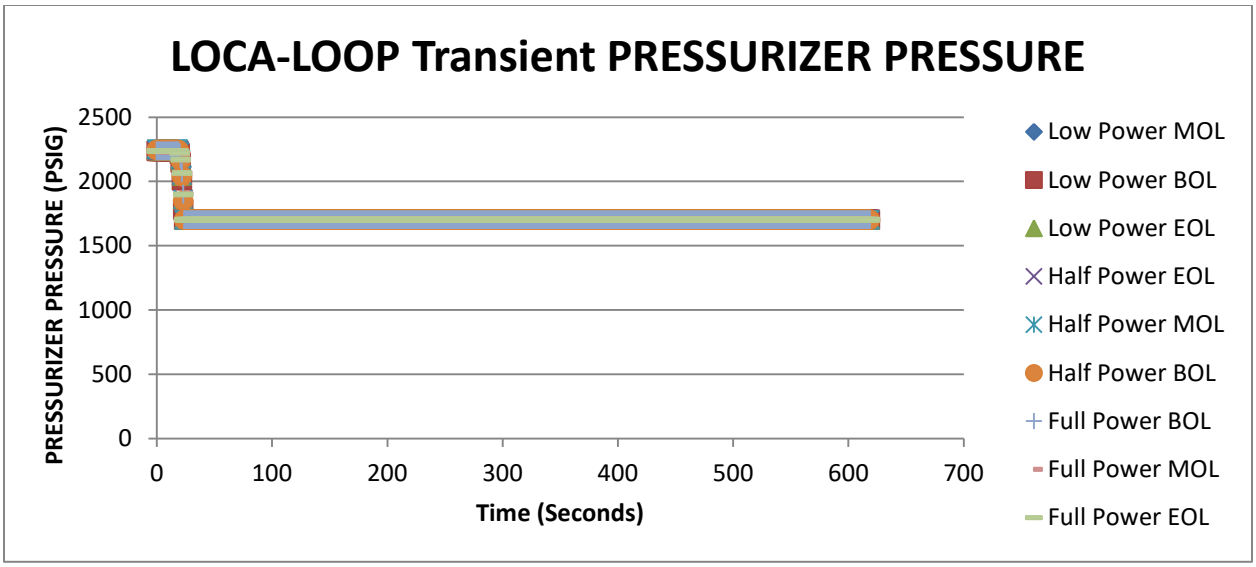


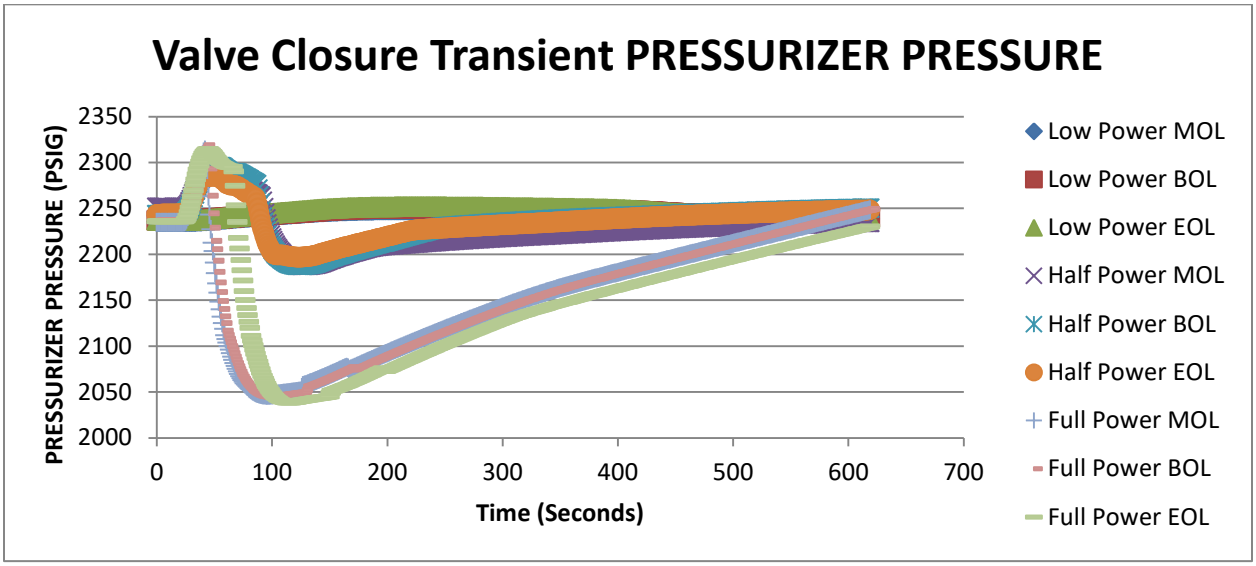

LOCA-LOOP Transient AVERAGE TEMPERATURE
AVERAGE TEMPERATURE (F)
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL


Valve Closure Transient AVERAGE TEMPERATURE
AVERAGE TEMPERATURE (F)
605
600
595
590
585
580
575
570
565
560
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

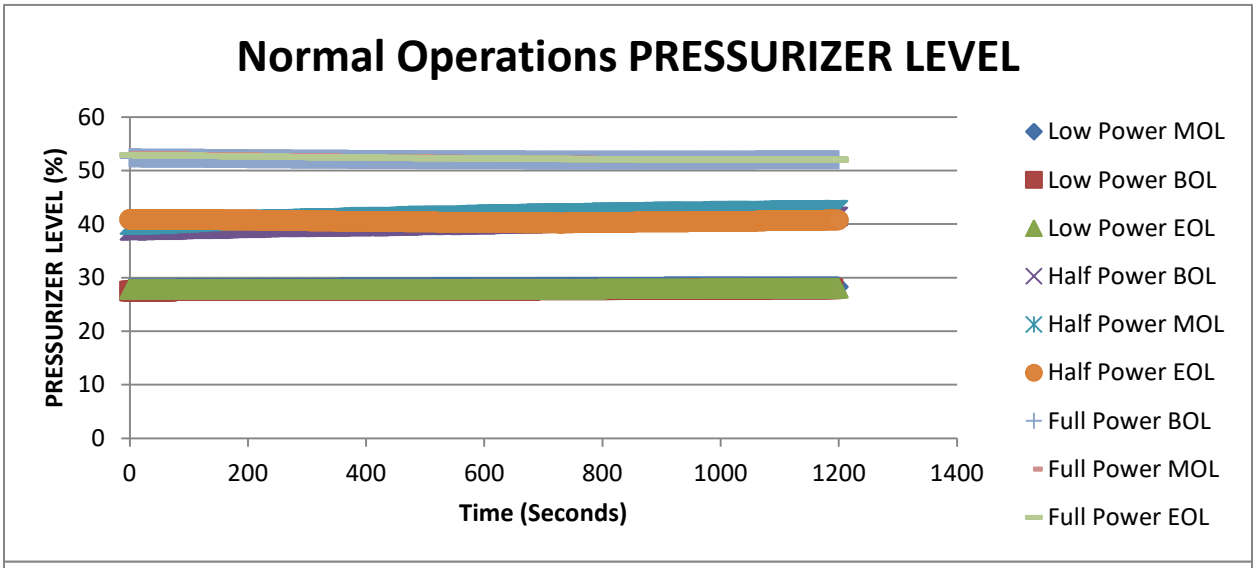
Appendix 1.28 PRESSURIZER PRESSURE

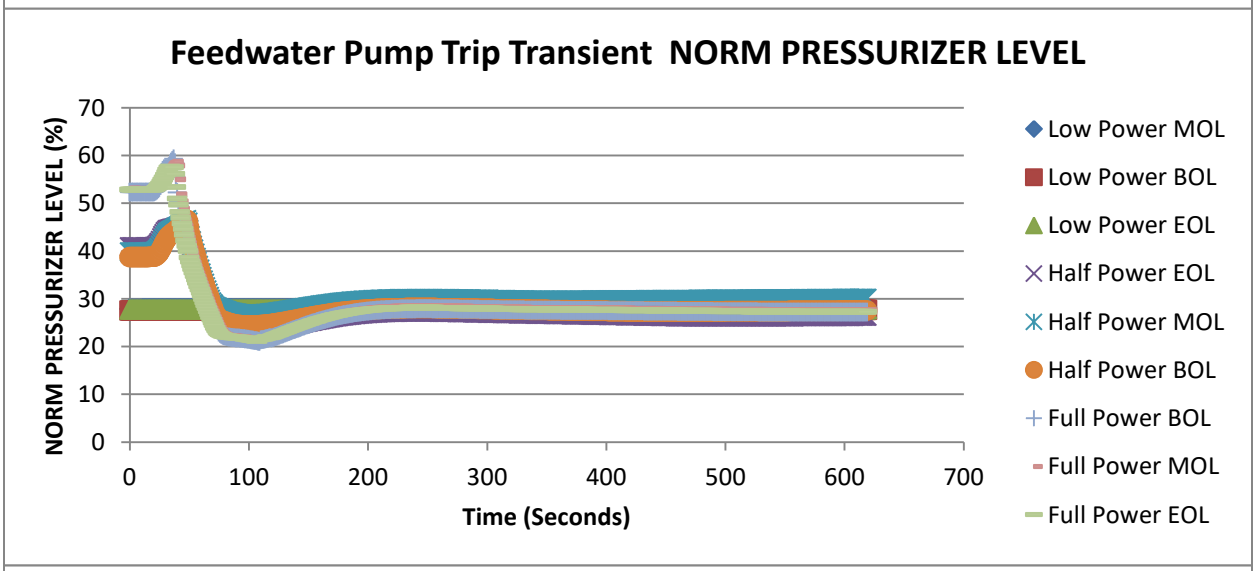


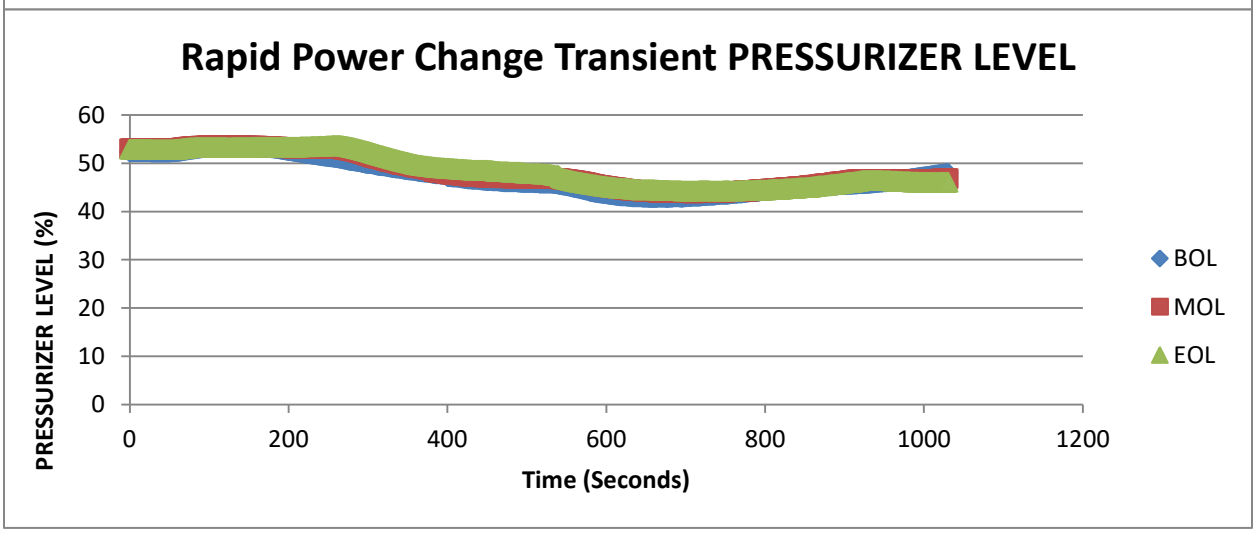

LOCA-LOOP Transient PRESSURIZER PRESSURE
PRESSURIZER PRESSURE (PSIG)
2500
2000
1500
1000
500
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

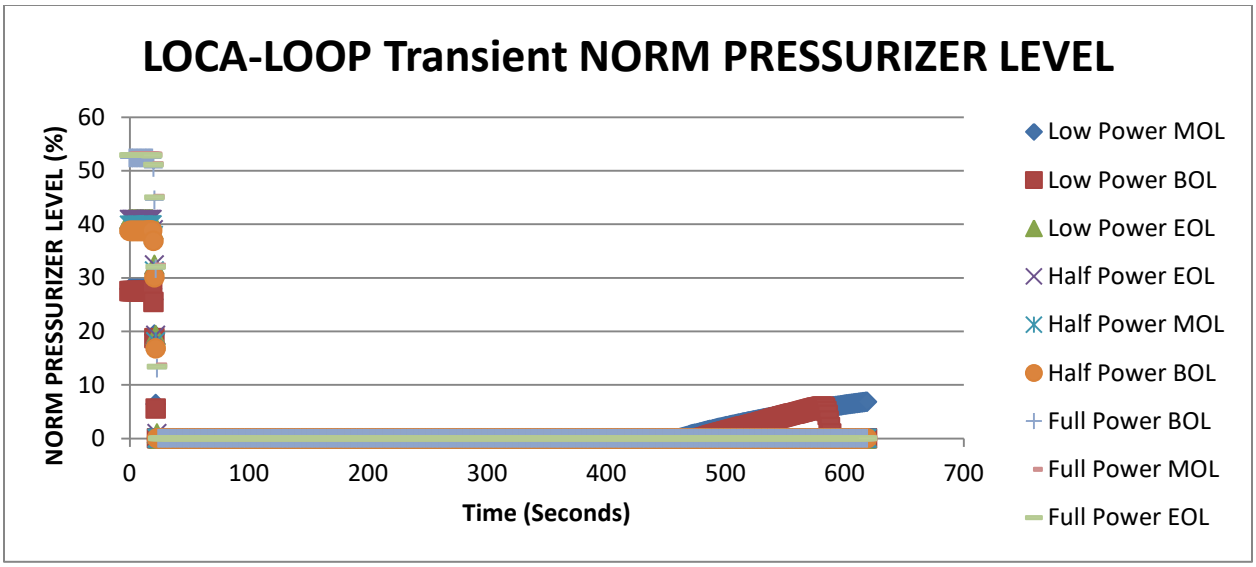

Valve Closure Transient PRESSURIZER PRESSURE
PRESSURIZER PRESSURE (PSIG)
2350
2300
2250
2200
2150
2100
2050
2000
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

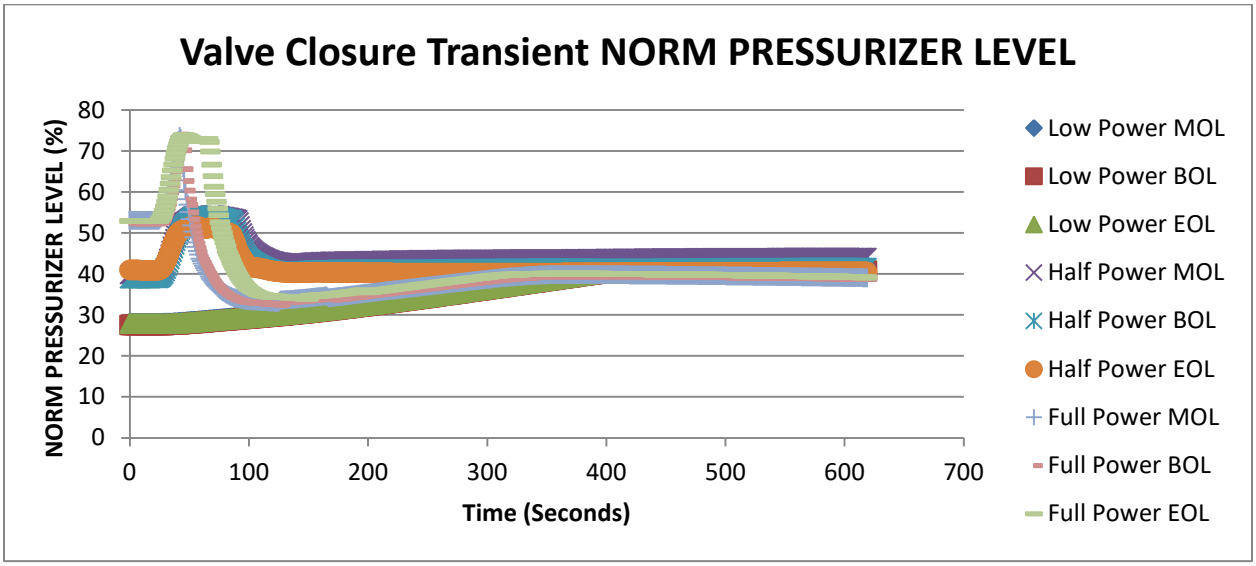
Appendix 1.29 PRESSURIZER LEVEL



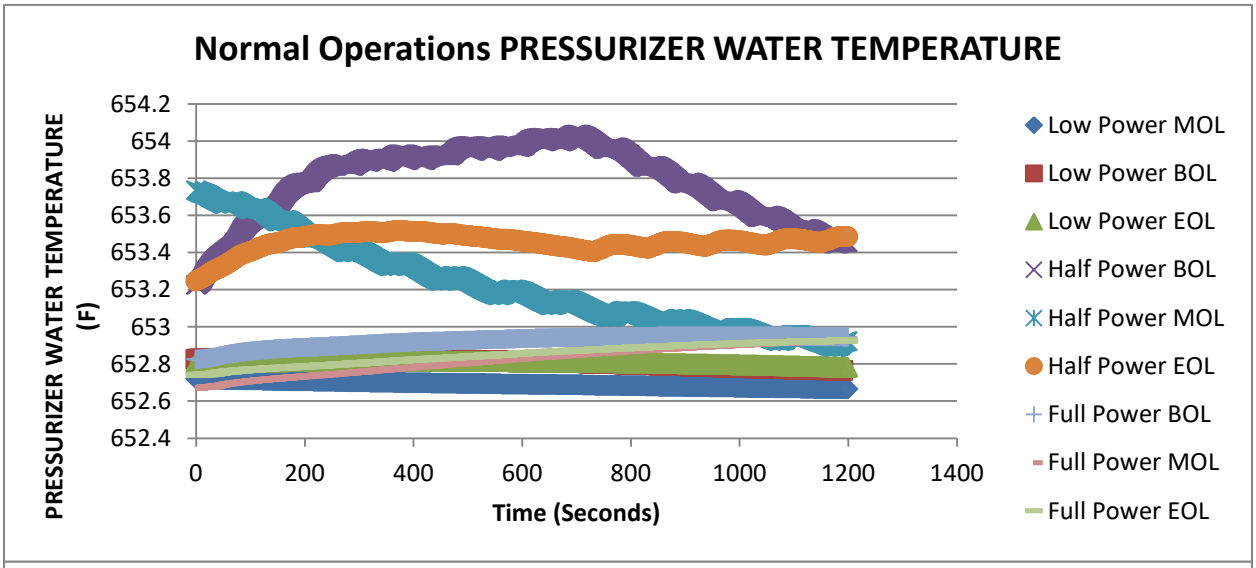


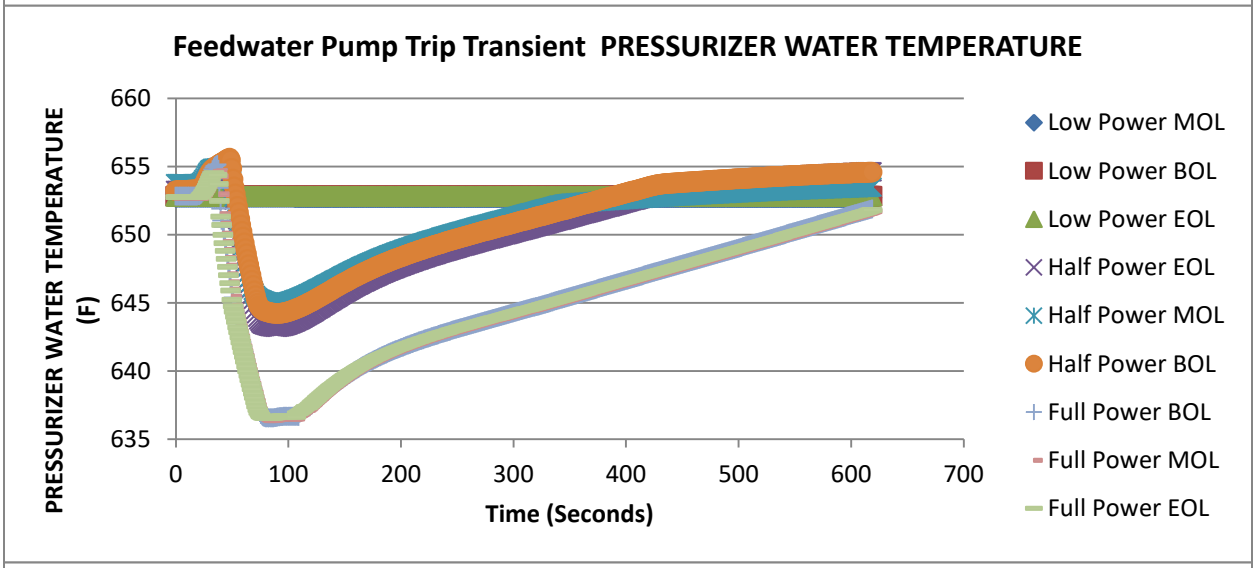


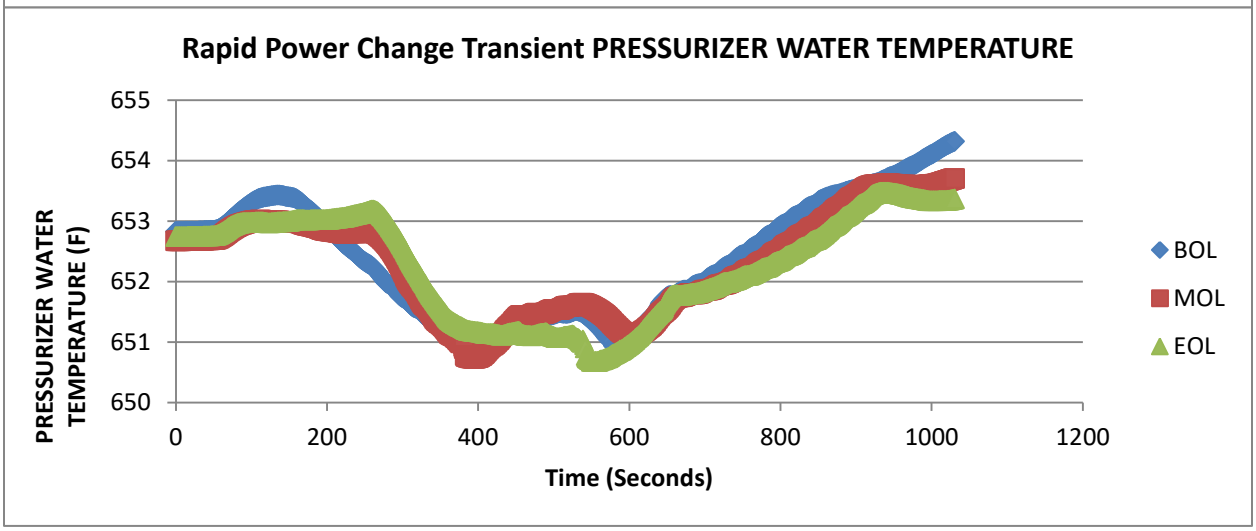

LOCA-LOOP Transient NORM PRESSURIZER LEVEL
NORM PRESSURIZER LEVEL (%)
60
50
40
30
20
10
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL

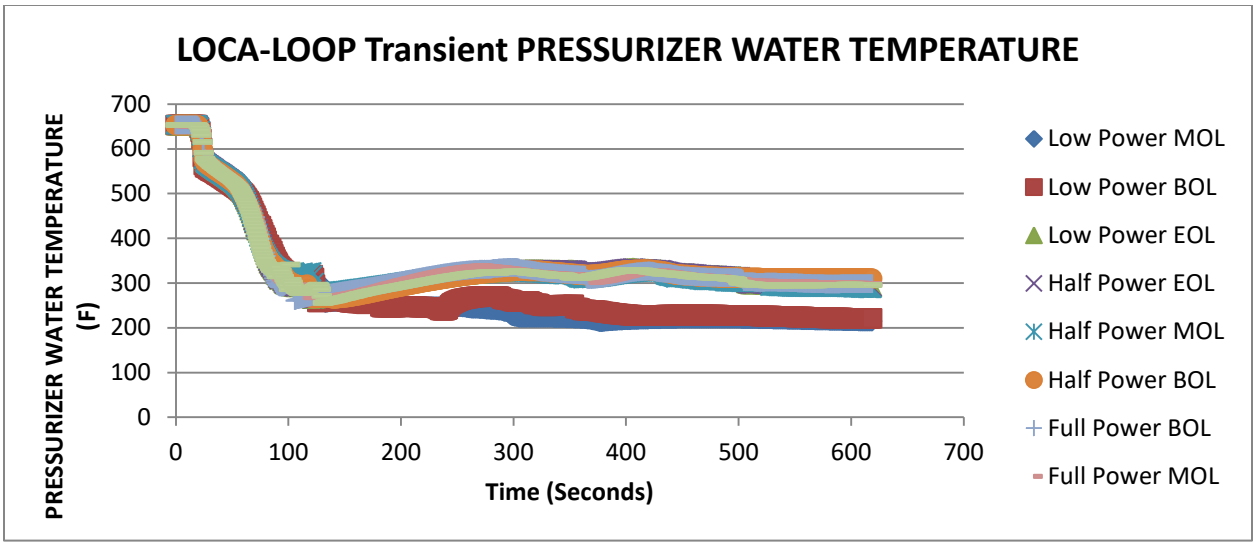

Valve Closure Transient NORM PRESSURIZER LEVEL
NORM PRESSURIZER LEVEL (%)
80
70
60
50
40
30
20
10
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

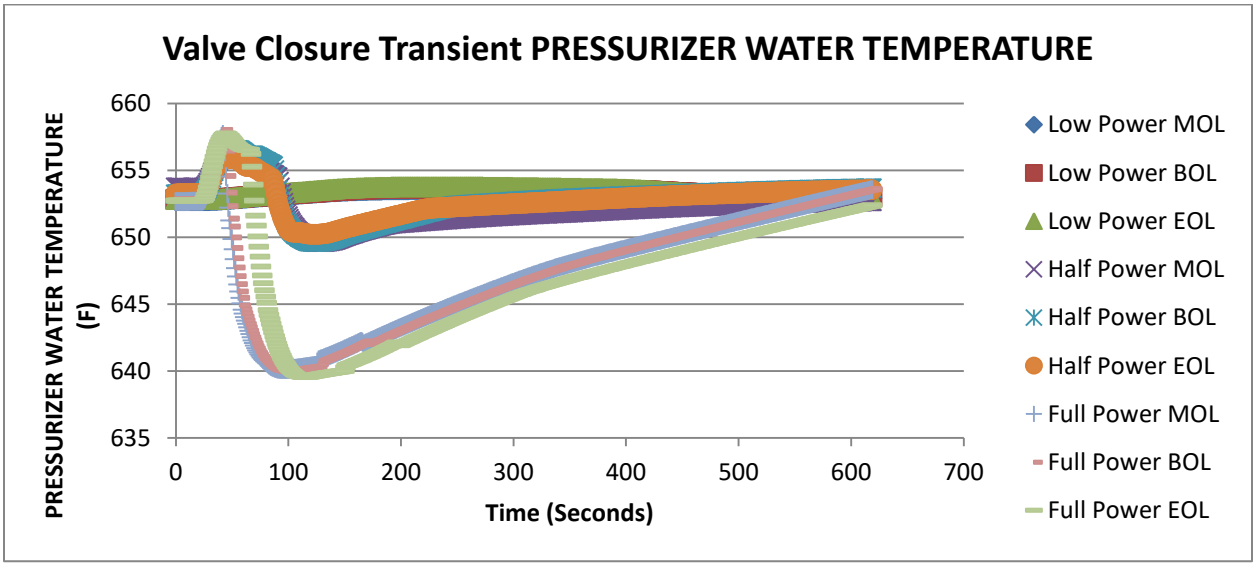
Appendix 1.30 PRESSURIZER WATER TEMPERATURE



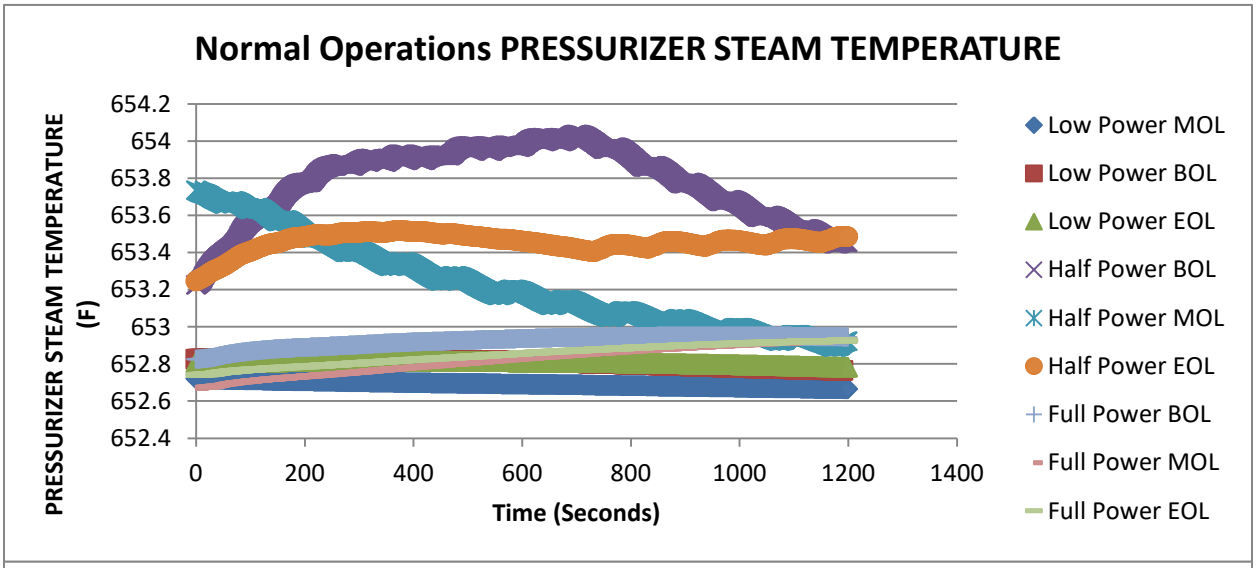


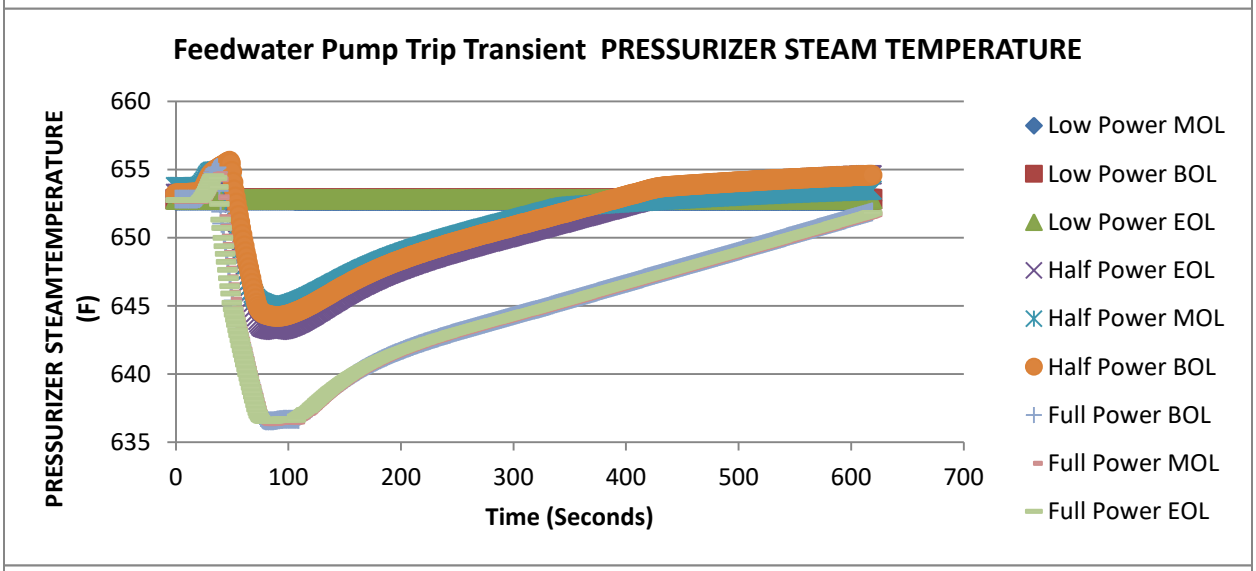


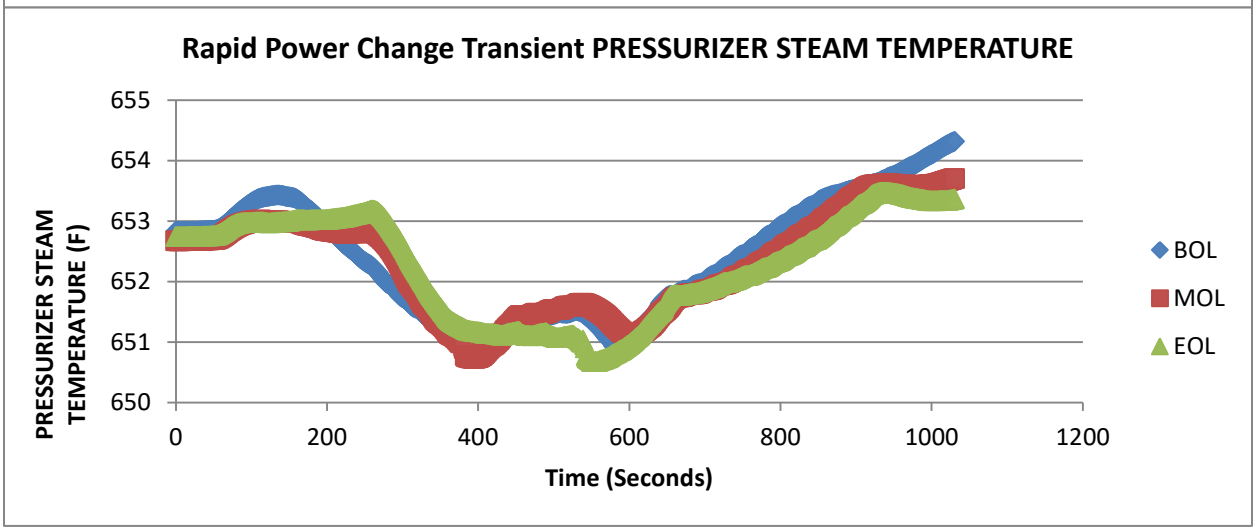

LOCA-LOOP Transient PRESSURIZER WATER TEMPERATURE
PRESSURIZER WATER TEMPERATURE (F)
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL

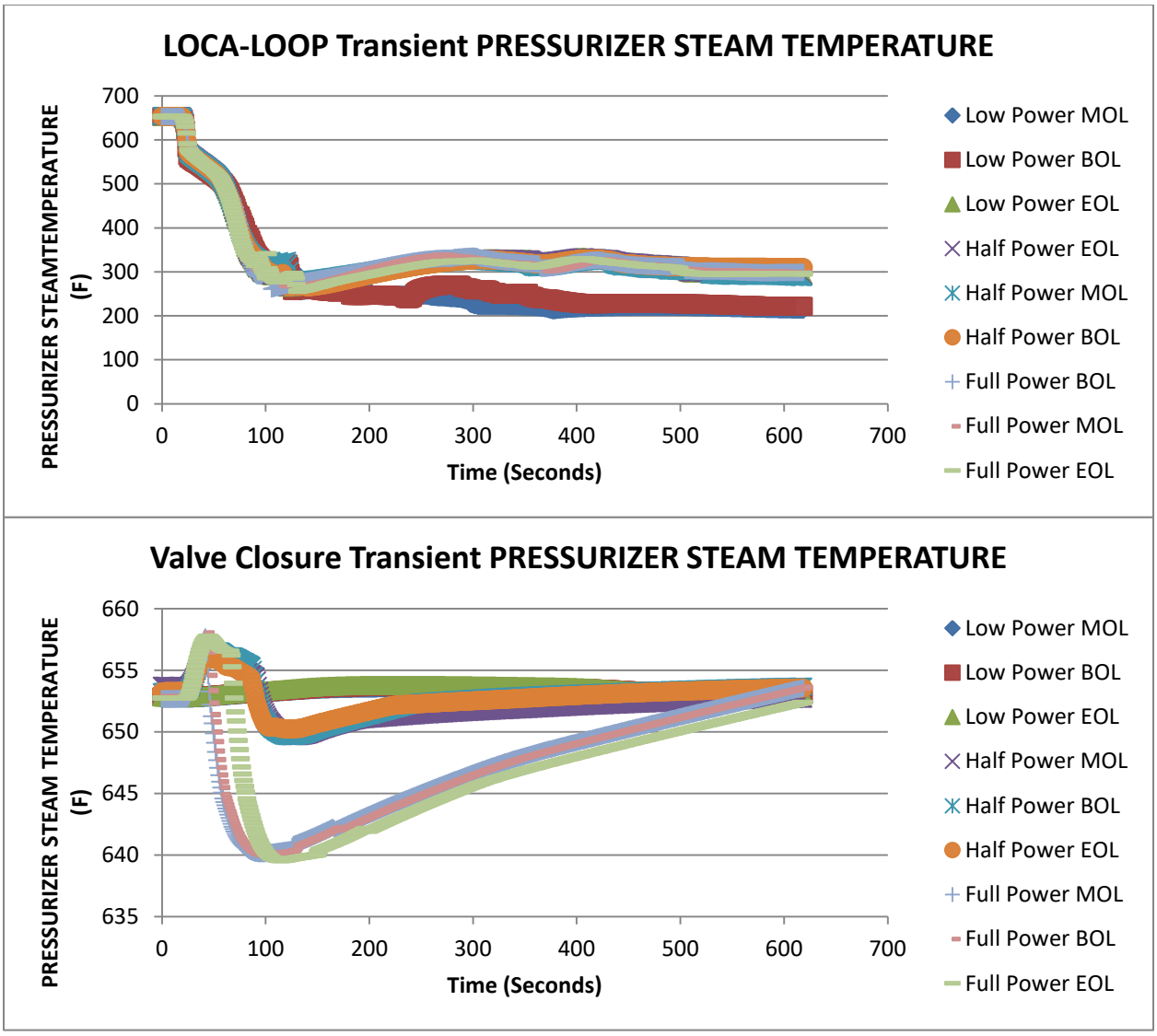

Valve Closure Transient PRESSURIZER WATER TEMPERATURE
PRESSURIZER WATER TEMPERATURE (F)
660
655
650
645
640
635
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

Appendix 1.31 PRESSURIZER STEAM TEMPERATURE

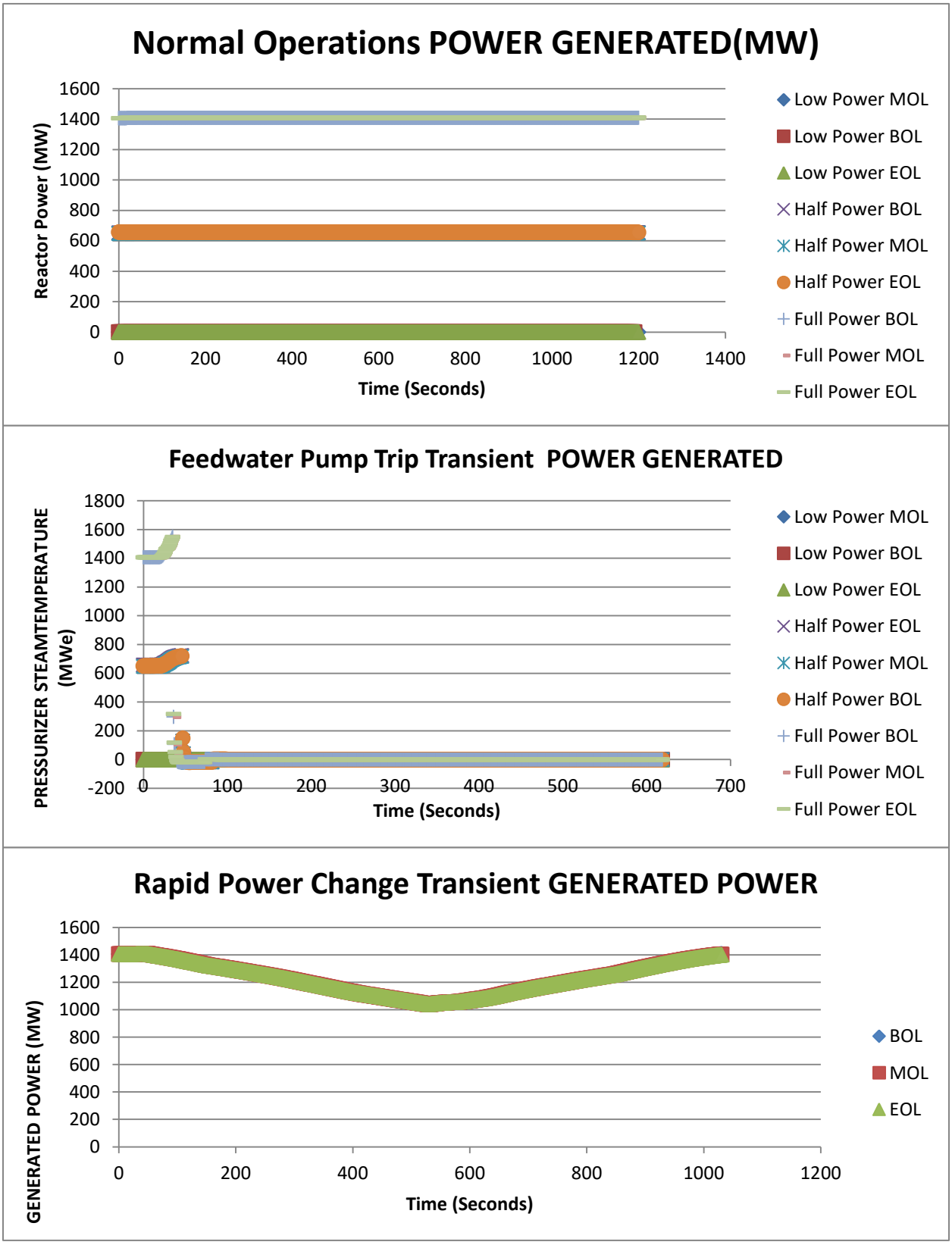


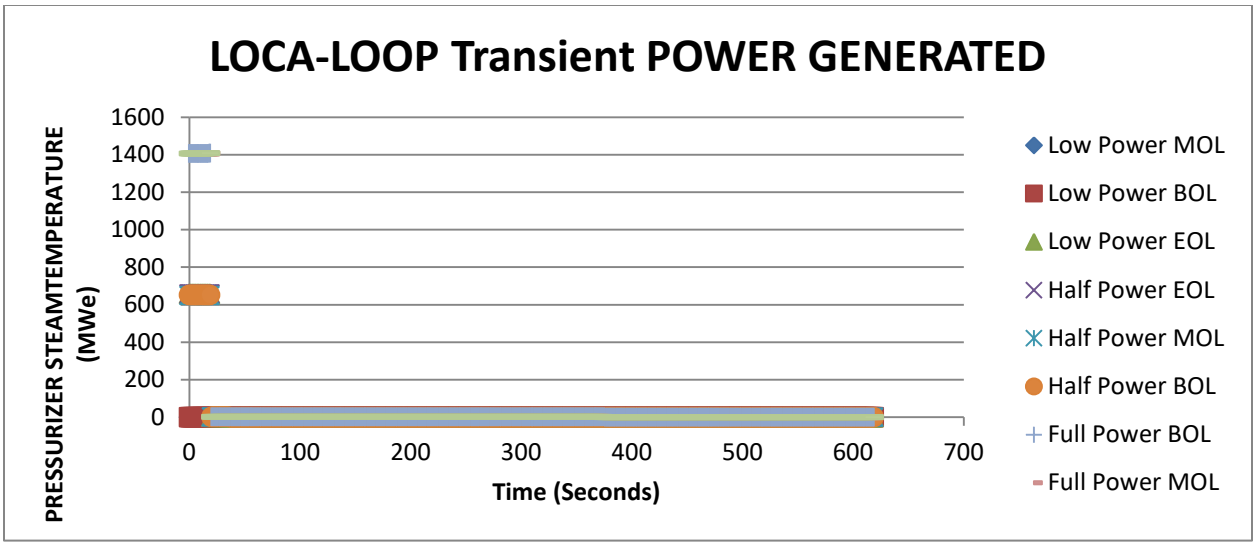


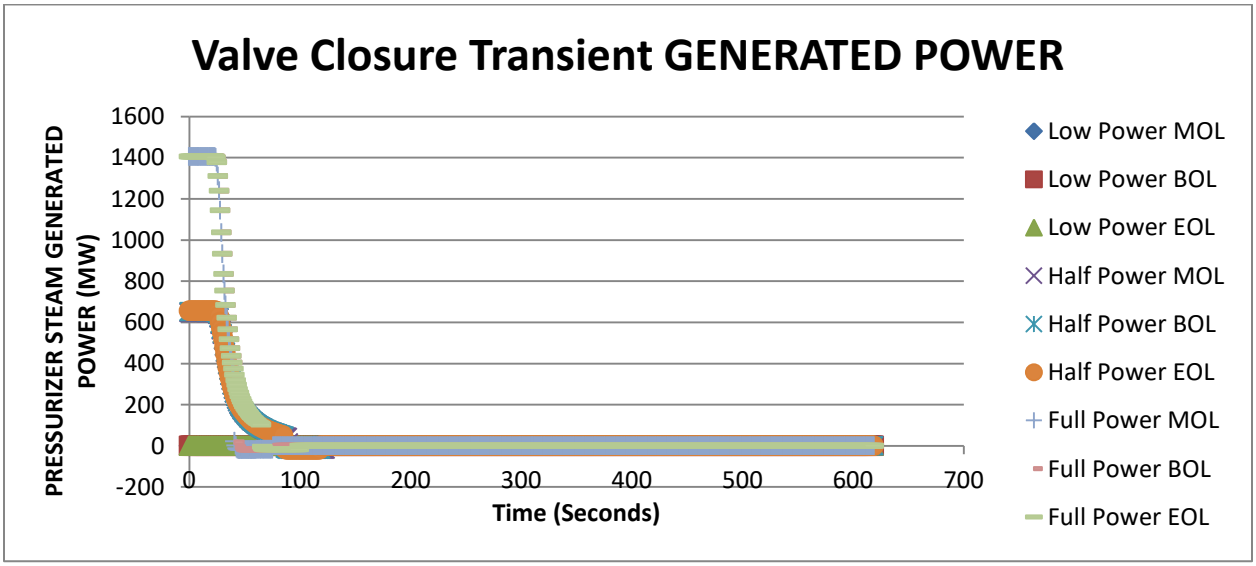



LOCA-LOOP Transient PRESSURIZER STEAM TEMPERATURE
PRESSURIZER STEAMTEMPERATURE (F)
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL
Full Power EOL
Valve Closure Transient PRESSURIZER STEAM TEMPERATURE
PRESSURIZER STEAM TEMPERATURE (F)
660
655
650
645
640
635
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

Appendix 1.32 Reactor Power (MW)




LOCA-LOOP Transient POWER GENERATED
PRESSURIZER STEAMTEMPERATURE (MWe)
1600
1400
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power EOL
Half Power MOL
Half Power BOL
Full Power BOL
Full Power MOL


Valve Closure Transient GENERATED POWER
PRESSURIZER STEAM GENERATED POWER (MW)
1600
1400
1200
1000
800
600
400
200
0
-200
0
100
200
300
400
500
600
700
Time (Seconds)
Low Power MOL
Low Power BOL
Low Power EOL
Half Power MOL
Half Power BOL
Half Power EOL
Full Power MOL
Full Power BOL
Full Power EOL

Appendix 2.1 Results from Second Run of Machine Learning Models

Overall Validation Measurements					Individual Transient Accuracies				
	Accuracy	Precision	Recall	F1 Score	Normal Operation Transient	Feed Water Pump Trip	LOCA + LOOP	Valve Closure	Rapid Power Change
K-Nearest Neighbors	98.34%	97.94%,	98.09%	98.00%	100%	94.78%	98.11%	98.43%	98.80%
Bernoulli Naïve Bayes	98.06%,	97.87%,	97.32%	97.53%.	100%	93.95%	100%	99.88%	93.72%
Gaussian Naïve Bayes	98.21%	98.03%	97.65%	97.82%	100%	96.71%	99.74%	96.71	95.21%
Multinomial Naïve Bayes	96.64%.	96.29%,	95.60%	95.74%.	100%	89.11%	100%	100%	86.11%
Logistic Regression	98.67%	98.54%	98.15%	98.33%	100%	97.58%	99.77%	97.90%	95.21%
Decision Tree Analysis	98.49%	98.19%	98.28%	98.24%	100%	97.21%	97.55%	97.15%	99.27%

Appendix 2.2 Results from Third Run of Machine Learning Models

Overall Validation Measurements					Individual Transient Accuracies				
	Accuracy	Precision	Recall	F1 Score	Normal Operation Transient	Feed Water Pump Trip	LOCA + LOOP	Valve Closure	Rapid Power Change
K-Nearest Neighbors	98.51%	98.21%,	98.35%	98.27%	100%	97.98%	97.93%	97.94%	96.29%
Bernoulli Naïve Bayes	97.74%,	97.51%,	96.75%	97.03%.	99.98%	94.17%	100%	99.92%	88.08%
Gaussian Naïve Bayes	97.41%	97.15%	96.75%	96.93%	100%	93.33%	100%	94.70%	94.75%
Multinomial Naïve Bayes	97.11%.	96.80%,	96.62%	96.62%.	100%	87.08%	100%	99.51%	95.29%
Logistic Regression	98.60%	98.41%	98.09%	98.24%	100%	96.80%	99.81%	98.07%	95.44%
Decision Tree Analysis	98.34%	97.94%	98.15%	98.04%	100%	94.90%	97.78%	98.22%	99.53%