Impact of High Photo-Voltaic Penetration on Distribution Systems

FINAL REPORT

Team Number 57

Client / Advisor
Dr. Ajjarapu

Team Members/Roles
Thomas Coleman – Team Leader
Andrew Chaney – Project Engineer
Daniel Riley – Assistant Project Engineer/Editor
Kenneth Prell – Assistant Project Engineer/Document Architect

Team Email
sdmay20-57@iastate.edu

Team Website
http://sdmay20-57.sd.ece.iastate.edu/

Revised: Date/Version
April 25, 2020 – Version 4

Executive Summary

Engineering Standards and Design Practices

The addition of distributed and spot PV generation to distribution systems causes changes in voltage profile and power flow. Voltages must be maintained between 0.95 and 1.05 per unit, currents in distribution lines must not exceed rated ampacity of the line, and KVA ratings of the transformers shall not be violated during operation of the system.

Summary of Requirements

Design a simulation of the IEEE 34 node and 123 node test feeders that will produce useful data about additional penetration of photo-voltaic cells.

Design the simulation where photo-voltaic cell placement will be feasible for useful distribution.

Observe all distribution system limitations when designing our simulation.

Minimize reverse powering the transmission system since there is not a set method to measure the reverse flow.

Applicable Courses from Iowa State University Curriculum

MATH 207 – Linear Algebra

EE 230 – Electronic Systems and Circuits

EE 303 – Energy Systems and Power Electronics

EE 452 – Electrical Machines and Power Electronic Devices

EE 455 – Introduction to Energy Distribution Systems

EE 456 – Power Systems Analysis I

EE 457 – Power Systems Analysis II

EE 458 – Economic Systems for Electric Power Planning

New Skills/Knowledge acquired that was not taught in courses

Not officially acquired from ISU curriculum during our tenure: familiarization with the program OpenDSS.

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

We would like to begin by thanking Alok Bharati, a graduate student of Dr. Ajjarapu. Throughout our project, Mr. Bharati has provided invaluable assistance. This came in the form of providing literature to begin the project, troubleshooting our early designs, and guiding the direction of our project.

1.2 Problem and Project Statement

Climate concerns as well as economic incentives in the form of government subsidies have made solar energy increasingly attractive for both utilities and private consumers. Whereas utilities can better regulate power distribution to avoid harming their transmission networks, private solar cells often cause issues on distribution networks by reducing power quality. This reduced power quality leads to higher energy usage as well as degradation of these networks. Utilities prefer to avoid this, and as such will perform analyses to improve the transition of solar power into their networks.

The goal of our project is to establish a simulation model which models a distribution network given nameplate data from the utility company managing power in a given area, and then identifies weaknesses within the system. The simulation will be generalized, with specific examples (IEEE distribution system test feeders) being tested to ensure model accuracy. The model itself aims only to simulate a distribution network, but those using it will then be able to draw conclusions on necessary changes. We have attempted to use realistic networks with the aim of providing general advice.

1.3 Operational Environment

Our project solely focuses on the simulation of a distribution system with penetration of photo-voltaic cells. As such, discussion of an operational environment is not applicable to our scope.

1.4 Requirements

- Processing power for OpenDSS simulations
- Transmission Network expertise and assistance
- Real world transmission network data to simulate
- Solar PV simulation implements power injection and voltage regulation

1.5 Intended Users and Uses

The intended user of this project is utility companies working on integration of large percentage photo-voltaic production. The utility will then use our findings to understand the effect of integration of distributed (consumer owned) solar, or community (utility owned) solar into their distribution system, as well as recommended changes to their operation. With this information, they can choose the most effective method of loss reduction and voltage regulation based on method of integration and expected penetration level.

1.6 Assumptions and Limitations

<u>Assumptions</u>

- Distributed load on a line is modeled as one half of the load attached to each end of the connecting distribution line.

- Team will coordinate with the project advisor to determine the scope of the project.

- Team will use provided software to model the distribution system.

<u>Limitations</u>

- System must operate within given distribution limitations in order to give legitimate data and accurate recommendations.

- No budget is required for this project as OpenDSS (a free open source library) will be the only program used.

- There will be no tests performed directly on the transmission network or solar cell components. The team will rely on information provided in IEEE Test Feeder cases.

- Modeled system must not be so large (1000+ nodes) as to strain our processing power.

1.7 Expected End Product and Deliverables

The product to be delivered to our clients is a detailed report of findings found through our simulation. This report shall include all data gained while simulating the distribution system and any recommendations utility operators on where to add photo-voltaic cells (residential vs. community) and how to control them (MW injection vs. voltage regulation) for optimal power flow. The delivery date for this deliverable shall be 26 April 2020.

2 Specifications and Analysis

2.1 Proposed Design

Final Design Components

- Familiarized ourselves with the knowledge required by looking at simplified subsystems exhibiting the same type of behavior that will be encountered when looking at our final simulation.

- Created a simple solver in MATLAB for the iterative process required to solve our distribution systems. This solver operated on the subsystem discussed above and is used only for research' sake.

- Transferred the knowledge gained from MATLAB to OpenDSS in order to solve more complex simulations given by real distribution systems. This required expanding beyond the subsystem to an entire system that contains many of these subsystems.

- Created MATLAB scripts for data input into OpenDSS for the large systems being analyzed.

- Implemented load shape characteristics to model full day variations in demand.

- Implemented addition of distributed solar generation corresponding to maximum system loading. Solar is implemented as having capacity equal to a percentage of the real loading at a given node, with inverter KVA rating equal to the real power rating of the solar.

- Analyzed 34-node system with solar inverters operating in 3 modes of operation: unity power factor, constant power factor of .85, and constant KVAR injection mode.

- Sizing and setting of community solar for the 34-node system.

- Implemented power-flow in MATLAB for the purpose of optimization.

- Determined optimal community solar size for the 34-node system that would maximize system loss reduction.

- Simulated 123-node system with loadshape data, extension of distributed solar analysis performed for 34-node system.

- Determined minimum % PV penetration needed for voltage control without use of voltage regulators in constant KVAR injection mode.

- Recommended mode of operation for distributed solar in systems similar to 123-node system, notably low voltage, relatively short distance distribution systems with unregulated bus voltages between 1.0 and 0.9 in per unit quantities.

- Recommended procedure for sizing and setting of spot solar loading in rural feeder systems similar to 34-node system.

2.2 Design Analysis

Though our project has changed forms from Alliant Energy's original intentions, we have still accomplished the essence of this project. We have modified the 34-node system to implement community solar in order to minimize system losses using an optimization process that can be adjusted and used on other systems. We modeled a larger 123-node system, which is closer to the size of a small real-world model. This gave us the opportunity to see results that will likely be encountered during our careers as electrical engineers.

Getting the power loss to decrease by 9.01% after adding solar to node 832 was a major strength. However, because of the control method we chose to utilize we found a weakness in that it had no impact on the nodes that were already out-of-specification with regards to their voltage profiles. Another weakness is that by trying to implement distributed solar to both the primary and secondary injection sites it would increase the power loss of the system. Overall, it can be concluded that constant power factor, while being a viable method of control, may not be the optimal type depending on the goal and other methods should be studied by future teams.

2.3 Development Process

We are implementing the Agile process. Using this process will help us effectively organize our workloads evenly and provide accountability for each team member according to their roles. Agile will also aid in preventing an overload of our team by allowing us the ability to roll over assignments in the event of non-completion.

3. Statement of Work

3.1 Previous Work and Literature

Transmission network modelling is not unique to our project and has been performed by many utility companies both testing their current networks and prior to completing upgrades to their systems. These are of course proprietary, and thus we are unable to use them in preparing our project. However, many resources exist which allow engineers to test their understanding. We first worked extensively with a feeder example given in the textbook [1]. This test feeder contained 4 nodes and introduced us to basic concepts of feeder networks. Next, we worked on the IEEE 34-node Test Feeder. IEEE releases test feeders for engineers to practice their skills with new software or algorithms. These come with solutions attached, so we could check our work. The final network we will be working on is the IEEE 123 node test feeder that is representative of a low-voltage distribution feeder. This provides us with a medium scale system that provides a relatively stable voltage profile for analysis of the effects of PV on voltage regulation without the use of traditional tap changing transformers.

3.2 Technology Considerations

OpenDSS is an open-source software designed to give power-flow solutions for distribution networks. It is a powerful tool that directly relates to this project since we are looking at the effects of penetration of photo-voltaic cells on the distribution system (power-flow solutions).

A strength of this software is its capability to solve complex power-flow solutions. Since our study consists of hundreds of buses, OpenDSS will be invaluable in obtaining the data and testing how we can best implement and control photo-voltaic penetration. Another strength is that the software is open source, meaning that there is no cost for its acquisition, and it is continuously being improved in real-time.

One weakness associated with this software is a technical barrier to entry. There is no official tutorial on general operation, or any simple examples given as an aid for understanding. This then requires additional time needed to understand the program. However, this has helped to ensure that we understand the fine details regarding the software's operation.

MATLAB is used as a controller for OpenDSS, as well as for changing the system model and data interpretation. Each solution set requires iterating over 1-100% (with 1% step size) solar PV penetration, and each iteration is associated with changing at least one file in the OpenDSS model. Importing the data into MATLAB requires reading data from approximately twice as many files as there are nodes in the system. For larger systems, this results in significant overhead in runtime. A solution set containing 100 power flow solutions can take several minutes to produce, and larger solution sets take correspondingly longer.

3.3 Task Decomposition

Textbook research

- **Preliminary research on structure of 4-node problem**
- **Review power concepts**

Our project began with the building of preliminary knowledge. We were assigned reading from a textbook [1] given by Dr. Ajjarapu. This graduate level textbook covers many overarching topics in transmission networks. Our first subtask here was familiarizing ourselves with the concepts in power relevant to our project. We all had some power experience, so this was more of a review than new material.

Handwritten 4-node network

- **Understand problem geometry**
- **Understand solution**
- **Complete first iteration of problem by hand ourselves**

The textbook contains many examples to assist in learning, and we focused on a particular problem, the modeling of a 4-node distribution feeder. To begin with, we aimed to understand the particular geometry of the problem with relation to the review we had done in the first main task. The example provided a solution, which we went through next, making sure we understood fully what was being done in each step, and why. Lastly, we solved the problem by hand ourselves, checking for accuracy with the textbook. This problem also had a secondary version where a voltage regulator was added to increase system stability. We repeated all the above steps again for this complicated system.

MATLAB 4-node network

- **Convert data into declarations**
- **Convert equations into MATLAB readable form**
- **Check for accuracy, change code where necessary**
- **Add voltage regulator**
- **Check for accuracy again**

Once we had completed our calculations by hand, we then moved onto solving it with MATLAB. A lot of the program code was dedicated to declarations, inputting data on the line, transformer, and load specifications. The next logical step was inputting the actual math used to solve our problem. This math was iterative, and a large part of the function was a loop. Once we had completed the basic feeder calculations, we added the voltage regulator.

OpenDSS 4-node network

- **Learn OpenDSS syntax and commands**
- **Convert book problem into OpenDSS format**
- **Check results for accuracy, fix where necessary**
- **Convert regulator parameters as per book into OpenDSS format**
- **Check simulation against expected answers**

Finally, we had finished preparatory work and could begin using the software we would be using for the rest of the project, OpenDSS. We started here by learning the basic syntax and command format of OpenDSS through open source documentation as well as a lecture brought to us by ISU Faculty, Dr. Wang. Once we had familiarized ourselves, we implemented the 4-node problem into OpenDSS. In simulating, we checked for accuracy against the book's solution. We at first had an issue, which was found to be us unnecessarily adjusting input variables – OpenDSS would handle this itself when simulating. As before, we then redid the problem with the voltage regulator added. We then checked this against expected solutions and concluded this task.

34-node network

- **Use MATLAB scripts to input data into OpenDSS**
- **Simulate the network**
- **Evaluate and correct solution**
- **Observe system response with a variable 24-hour load period**
- **Inject PV into the network**
- **Test the different methods and control modes of PV to determine an optimal solution**

Next, we will continue to build upon our OpenDSS knowledge by completing the simulation of a more advanced network – a 34-node distribution feeder. This example is provided to us by IEEE, for students to test themselves. We will again begin by converting the data into a form that is readable by OpenDSS. As this is a larger network, we chose to create MATLAB scripts that would read from a .csv file to automatically read the system data. Once data has been received by OpenDSS, we will simulate the system and observe the values throughout the network.

The most distant node had a low out-of-specification value; we reported this to our advisor. Our response to this low voltage condition will be to add capacitors to bring the node back within specifications. We then replaced the capacitors with the intended photo-voltaic cells and saw how we could control them to regulate voltage and power delivery. We then observed how the system reacts over a variable 24-hour load period.

123-Node Network

- **Adjust MATLAB scripts to input data into OpenDSS**
- **Simulate the network**
- **Evaluate the solutions (Unity, Constant .85 PF, Constant kVar)**
- **Observe system response of a variable 24-hour load period**

With the 34-node test case complete, we moved to simulating larger feeder networks. We began this by adjusting the same MATLAB scripts used for the 34-node network to work on this system. Once our data input scripts are working as intended, we moved to troubleshooting run parameters of the system. For example, a centerpiece of our simulation was a controller in MATLAB, which controlled the interface between MATLAB and OpenDSS. Following troubleshooting, we simulated the system over a 24-hour loadshape and analyzed our results.

The 123 Node test case has a flatter voltage profile than the 34-node test case, which makes it a better candidate for voltage regulation via distributed solar. Different levels of inverter KVAR injection were tested to determine the minimum voltage penetration to allow traditional voltage regulators to be removed. Lower percentages of KVAR injection were tested to find at what point voltage regulation could be accomplished while reserving only 44% of inverter capacity for KVAR injection, as well as the minimum KVAR injection percentage at 100% solar PV penetration to maintain voltage within 0.95-1.05 per unit quantities.

34 Node Optimization

- **Write power-flow in MATLAB**
- **Determine PV size at each node from results**
- **Find maximum loss reduction node (primary)and most out-of-specification voltage node (secondary)**
- **Place PV size of primary node as a limit and adjust from 100% to 0% between primary and secondary nodes**

The Optimization process for the 34-node system is a process outlined in articles [1] and [2], both of which can be accessed via the IEEE Xplore digital library. For our simulation we decided to work with a constant power factor of 0.85. The first step in the process was to use the base data and run the power-flow in MATLAB to determine the size of the PV that should be added to each node within the system. From that data, we calculated which node within the system would be optimal based on it having the maximum loss reduction. This node became the primary PV injection site. The next step was to calculate which node had the worst out-of-specification voltage. This became the secondary PV injection site. With both sites identified, we started adjusting the PV that would be added at the primary site. For the purposes of our project it quickly became apparent that adding any PV into the secondary injection site would cause additional losses in the system and the voltage profile at the secondary site wasn't going to be brought into specifications due to the method of our testing. Testing could be expanded to include different types of control such as constant KVAR injection for voltage profile improvement.

3.4 Possible Risks and Risk Management

Due to the effects of the COVID-19 pandemic, team and advisor meetings are held remotely where possible, and MATLAB and OpenDSS scripts have been updated to allow them to be run on personal computers rather than senior design lab computers.

3.5 Project Proposed Milestones and Evaluation Criteria

- Textbook reading to understand distribution systems - September 16, 2019
  - Do we understand the basics of transmission lines?
- Hand calculations for 4-node example – September 22, 2019
  - Do our results match given solutions?
- MATLAB 4-node example – September 29, 2019
  - Do the calculations match given solutions?
- OpenDSS 4-node example – November 10, 2019
  - Does the simulation match the given solution?
- 34-node example OpenDSS initial solution – November 24, 2019
  - Does the simulation match the IEEE solution?
- Synthetic network initial solution – November 24, 2019

    - Does the simulation behave realistically?
- Observe 34-node example over a 24-hour load period – November 24, 2019
    - Does variable load affect system operation? Compare to real life system.
- Add photo-voltaic penetration to 34-node – December 8, 2019
    - Does PV reduce load on substation? Analyze effects.
- Observe 123 node system over a 24-hour load period – February, 2020
    - Does variable load affect system operation? Compare to real life system.
- Model IEEE 123 Node Test Feeder (Data input, loadshape, analysis) – April 2020
    - How does the new system compare to the 34 node system? Does this system get a different recommendation for ideal operation?
- Optimize 34-node spot PV implementation – April, 2020
    - Does the recommended system use PV placement effectively to produce minimum system loss?

3.6 Project Tracking Procedures

Our team used bi-weekly reports to track our progress. These reports include:

- Overall weekly summary
- Past weeks accomplishments
- Issues
- Individual contributions
- Plans for next week
- Summary of meeting with Dr. Ajjarapu and Alok Bharati

We are also utilizing Microsoft Teams as a means for document sharing and progress tracking.

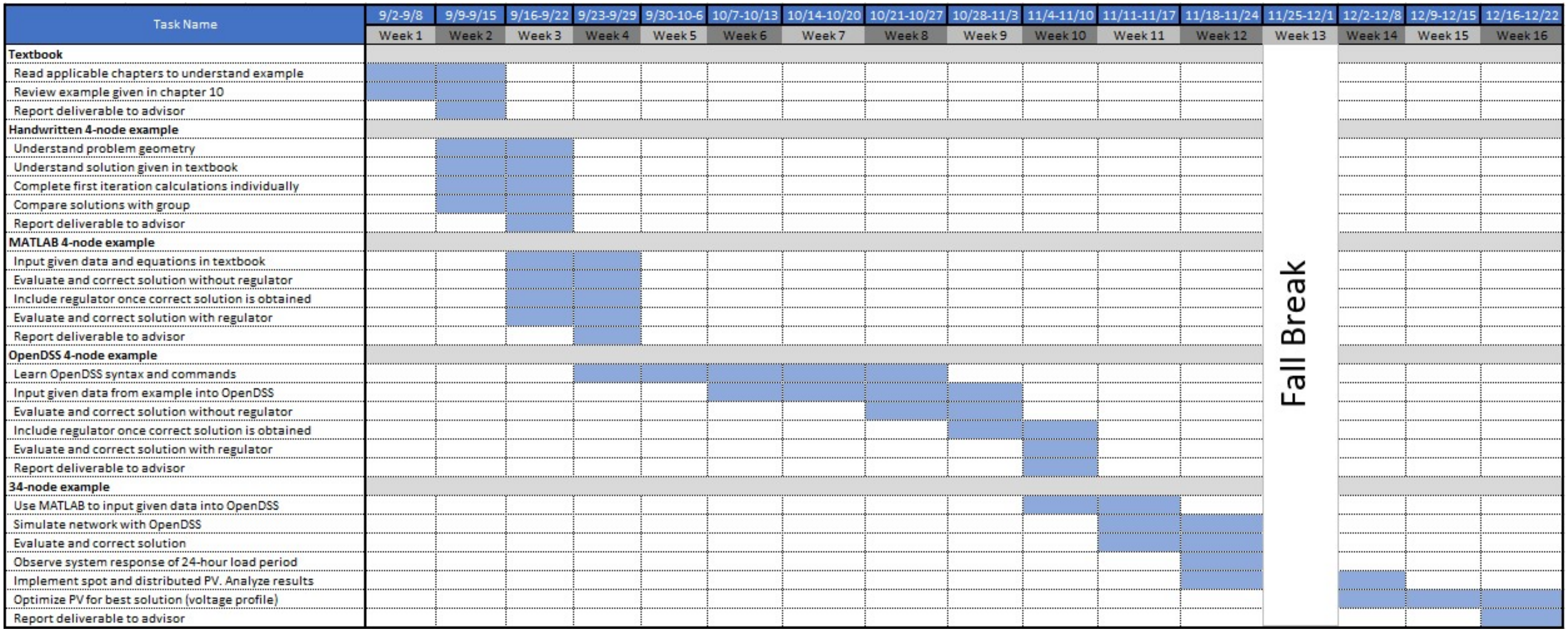
Github is used as a code repository.

3.7 Expected Results and Validation

The outcome of this project is to determine how to best implement photo-voltaic penetration. This implementation will be based upon how this penetration will be installed (residential vs. community owned farm) and how it will be controlled (regulation vs. power injection). Our goal is to provide a design solution such that the photo-voltaic penetration will provide the maximum benefits with respect to system losses and voltage deviation.

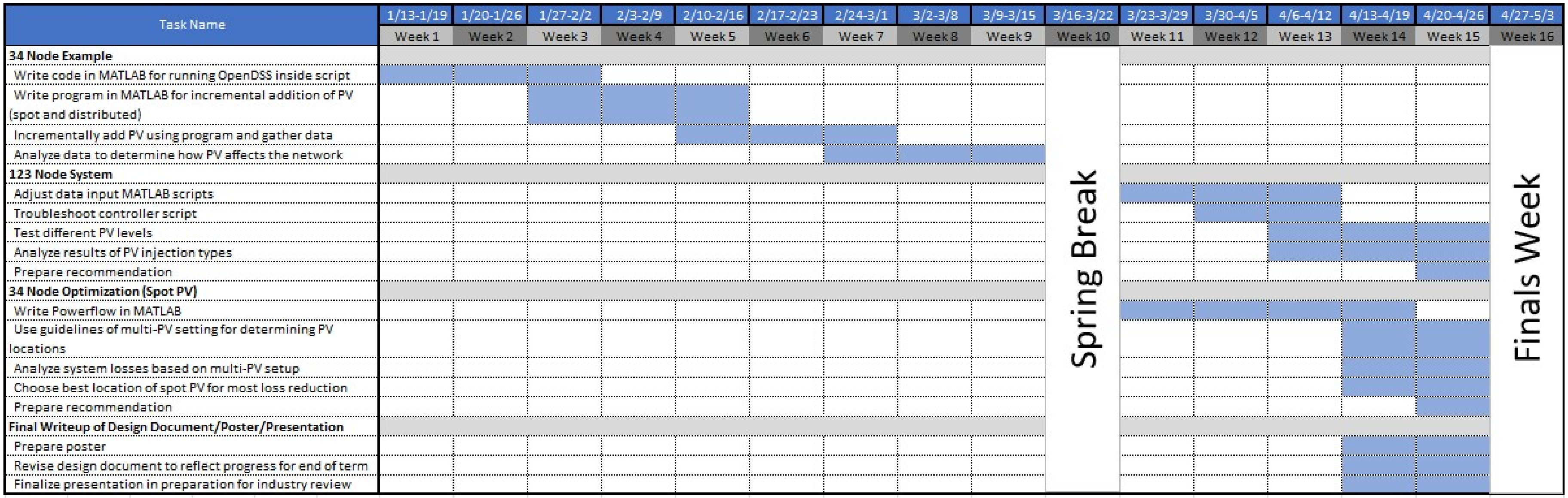
4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline



Task Name	9/2-9/8	9/9-9/15	9/16-9/22	9/23-9/29	9/30-10-6	10/7-10/13	10/14-10/20	10/21-10/27	10/28-11/3	11/4-11/10	11/11-11/17	11/18-11/24	11/25-12/1	12/2-12/8	12/9-12/15	12/16-12/22
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
Textbook													Fall Break			
Read applicable chapters to understand example	■	■														
Review example given in chapter 10	■	■														
Report deliverable to advisor		■														
Handwritten 4-node example																
Understand problem geometry		■	■													
Understand solution given in textbook		■	■													
Complete first iteration calculations individually		■	■													
Compare solutions with group		■	■													
Report deliverable to advisor			■													
MATLAB 4-node example																
Input given data and equations in textbook			■	■												
Evaluate and correct solution without regulator			■	■												
Include regulator once correct solution is obtained			■	■												
Evaluate and correct solution with regulator			■	■												
Report deliverable to advisor				■												
OpenDSS 4-node example																
Learn OpenDSS syntax and commands				■	■	■	■	■								
Input given data from example into OpenDSS						■	■	■	■							
Evaluate and correct solution without regulator								■	■							
Include regulator once correct solution is obtained									■	■						
Evaluate and correct solution with regulator										■						
Report deliverable to advisor										■						
34-node example																
Use MATLAB to input given data into OpenDSS										■	■					
Simulate network with OpenDSS											■	■				
Evaluate and correct solution											■	■				
Observe system response of 24-hour load period												■				
Implement spot and distributed PV. Analyze results												■		■		
Optimize PV for best solution (voltage profile)														■	■	■
Report deliverable to advisor																■

Figure 4-1 – Fall 2019 Gantt Chart



Task Name	1/13-1/19	1/20-1/26	1/27-2/2	2/3-2/9	2/10-2/16	2/17-2/23	2/24-3/1	3/2-3/8	3/9-3/15	3/16-3/22	3/23-3/29	3/30-4/5	4/6-4/12	4/13-4/19	4/20-4/26	4/27-5/3
	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16
34 Node Example										Spring Break						Finals Week
Write code in MATLAB for running OpenDSS inside script	■	■	■													
Write program in MATLAB for incremental addition of PV (spot and distributed)			■	■	■											
Incrementally add PV using program and gather data					■	■	■									
Analyze data to determine how PV affects the network							■	■	■							
123 Node System																
Adjust data input MATLAB scripts											■	■	■			
Troubleshoot controller script												■	■			
Test different PV levels													■	■	■	
Analyze results of PV injection types													■	■	■	
Prepare recommendation															■	
34 Node Optimization (Spot PV)																
Write Powerflow in MATLAB											■	■	■	■		
Use guidelines of multi-PV setting for determining PV locations														■	■	
Analyze system losses based on multi-PV setup														■	■	
Choose best location of spot PV for most loss reduction														■	■	
Prepare recommendation															■	
Final Writeup of Design Document/Poster/Presentation																
Prepare poster														■	■	
Revise design document to reflect progress for end of term														■	■	
Finalize presentation in preparation for industry review														■	■	

Figure 4-2 – Spring 2020 Gantt Chart

The above Gantt charts show our progress on this project over the two semesters we dedicated to it. The first semester was mostly focused on gaining knowledge on power systems operation and technology (OpenDSS and MATLAB). The second semester was then assigned to applying what we had learned. We completed two subprojects – mapping a larger system than before and optimizing our original system.

4.2 Personnel Effort Requirements

This table details approximate workloads for different parts of our projects. Most of our time was focused on simulation of our multiple networks.

Table 4-1 - Task Requirements

Task	Description	Time (hours)
Textbook	Read applicable chapters to understand examples Review example given in chapter 10 Report deliverable to advisor	20
Handwritten 4-node example	Understand problem geometry Understand solution given in textbook Complete first iteration calculations individually Compare solutions with group Report deliverable to advisor	30
MATLAB 4-node example	Input given data and equations in textbook Evaluate and correct solution without regulator Include regulator once correct solution is obtained Evaluate and correct solution with regulator Report deliverable to advisor	10
OpenDSS 4-node example	Learn OpenDSS syntax and commands Input given data from example into OpenDSS Evaluate and correct solution without regulator Include regulator once correct solution is obtained Evaluate and correct solution with regulator Report deliverable to advisor	30
34-node example	Use MATLAB to input given data into OpenDSS Simulate the network Evaluate and correct solution Observe system response of 24-hour load period	100

	Add PV into the network (residential and community) Optimize PV for best solution and control Report deliverable to advisor	
123 - Node	Adjust 34-Node data input MATLAB scripts Simulate the network Evaluate solution and troubleshoot Observe system response to 24-hour loadshape Report deliverable to advisor	120
34-Node Optimization	Use MATLAB to determine optimal PV sizing at each node Implement PV at each node to determine the lowest power loss of the system Determine the node with the most out of spec voltage profile Adjust the level of PV penetration at the determined nodes to find the optimal solution based on minimizing power loss	70

4.3 Other Resource Requirements

- OpenDSS open-source software
- Processing power for power-flow solutions and report readouts

4.4 Financial Requirements

Our project is completely simulation based and the software we are using is an open-source program that is free to download. There will be no other needed resources for our project and as such our senior design project will have zero financial requirements.

5. Testing and Implementation

5.1 Interface Specifications

Any modern PC is capable of running OpenDSS and MATLAB and as such interfacing is not a concern. Barring processing requirements, we are hardware agnostic.

5.2 Hardware and Software

For data input and to establish a COM server, we are using MATLAB. MATLAB is well designed to read csv files (how data is provided by IEEE), and scripts can easily be changed depending on exact data formats (34-node vs 123-node). We also use a controller script to create a COM server which controls OpenDSS files.

We are using OpenDSS in the testing phase to analyze values within the network. OpenDSS is a great tool for analyzing power flow solutions as it gives simple user-friendly text file outputs with detailed data of each node.

Lastly, we are using MATLAB for our optimization calculations. Originally, we considered using CPLEX, but the IEEE papers we read used MATLAB power-flow to optimize rather than objective function minimization. Based on time constraints, we chose to implement optimization in MATLAB rather than trying to learn multi-objective function optimization through CPLEX.

5.3 Functional Testing

Testing for distributed solar PV is performed on the 123-node test feeder, while community (spot) solar sizing and setting is done on the 34-node system, due to lower complexity. For the 123 Node system, the goal is to eliminate the need of traditional tap operated voltage regulators by use of KVAR injection from distributed generation (DG). PV percent penetration is raised from 1% to 100%, and the voltage profile of the system is sampled for each iteration. Initially, the PV inverters are operated in pure KVAR injection mode, to determine the minimum % penetration that makes the use of regulating transformers unnecessary. Then, KVAR injection is lowered to determine the maximum real power injection possible based on the level of PV penetration.

5.4 Non-Functional Testing

While we had originally planned for an economic analysis of our systems, this has been omitted due to time constraints. Ultimately our optimization designed to reduce power losses is functionally equivalent to economic optimization. While tap changes on transformers do have a material cost (due to wear), this cost is orders of magnitude smaller than the cost of the loss of MWh.

5.5 Process

The 4-node was tested against the values in [1] as a basis for our understanding of power-flow.

The 34-node was tested using a base-case model and PV implementation to compare various data points with each other to see how PV affected them.

The 123-node used a similar function as the 34-node for testing with more analysis on PV inverter control modes.

The 34-node optimization compared the base-case model losses to the optimized results via the use of size and setting of community solar-farms outlined in [2] and [3].

5.6 Results

4-Node: Successfully simulated the example given in the textbook by hand, in MATLAB, and in OpenDSS. We had difficulties in implementing transformer modeling with three-phases. Ultimately, we were able to figure this issue out by consulting with our advisors and researching within the OpenDSS forums. This example gave us the tools to understand the power-flow solutions and implement them in OpenDSS, which will be our primary program used for solving and optimizing Alliant Energy's network.

34-Node: We implemented the 34-node network as described by the IEEE test feeder case and ensured results matched with the solutions provided. Initially some issues were found relating to the implementation of autotransformers. We then implemented the load shape model and have begun developing algorithms to quickly analyze results. This testing in particular allowed us to work with PV injection, which will be paramount moving forward.

Figure 5-1 – 34-node bus 832 voltage profile with different % PV

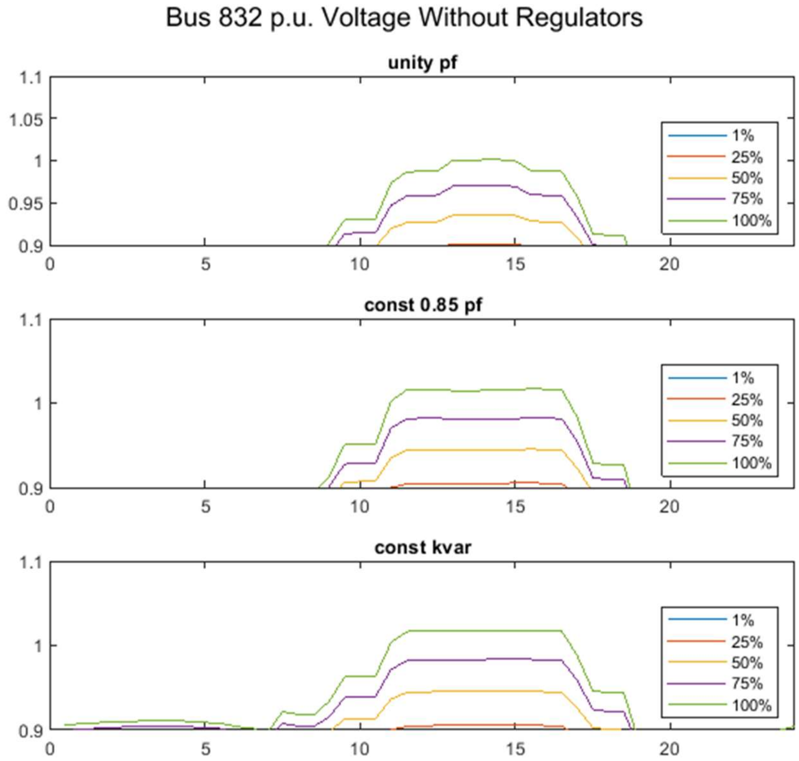
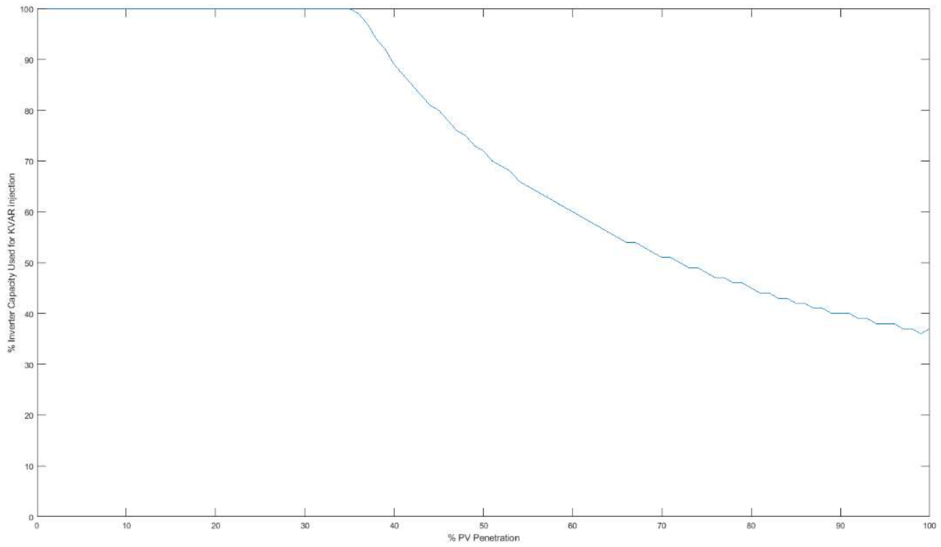


Figure 5-1 shows the most limiting node in terms of voltage for the 34-node system and how different types of PV implementation affected it. A unity power factor seemed to have the least impact for improving voltage on node 832. Constant .85 lagging power factor improved the voltage profile during the day and did not change the profile at all during the night. The constant KVAR mode improved the voltage profile the most, even raising the voltage levels during the night and early morning.

These results gave us an understanding of how PV can affect voltage profiles by using different inverter modes without the aid of voltage regulation components.

123-Node: As discussed above, we began by rewriting the data input MATLAB scripts as necessary. Once data input was confirmed to be working, we updated our controller script and troubleshot any remaining issues. The 123 Node system was then run from 1-100% solar PV penetration and 1-100% of inverter rating reserved for KVAR injection to determine what combinations of inverter operation and solar penetration would lead to sufficient voltage regulation for a given % PV penetration, as seen in figure 5-2.

Figure 5-2 – 123-node % PV injection used as complete voltage regulation



The curve in figure 5-2 represents the minimum % KVAR injection needed for a given % PV penetration in order to regulate voltage. Any operation above and to the right of the curve allows for voltage control without the use of voltage regulators. Agreed upon limits on inverter capacity reserved for KVAR injection by the Utility places bounds the area in which a specific utility is able to operate. For instance, if 44% of inverter capacity is reserved by the utility, complete voltage control is not achieved until PV penetration reaches 81% for this system. As a result, the recommendation for a hypothetical utility operating the 123-node system is to operate the solar inverters with a unity power factor until 81% of real loading is supplied by distributed PV, at which point inverters should be operated in constant KVAR injection mode of at least 44% of inverter capacity. An important note is that for unity pf, constant pf, and 44% KVAR injections modes of operation, losses always decrease as % PV penetration increases, so a Utility without the ability to easily modify inverter behavior would be able to set inverter to provide 44% KVAR injection at all levels of PV penetration, alongside traditional voltage regulators until PV penetration reaches 81%, as seen in figures 5-3 and 5-4.

Figure 5-3 – 123-node MWh losses

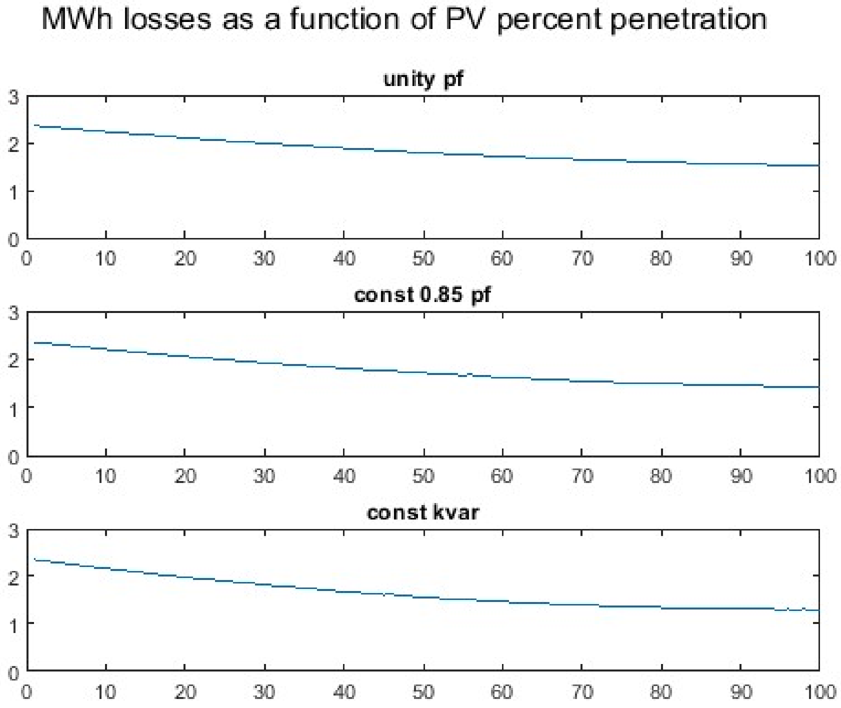
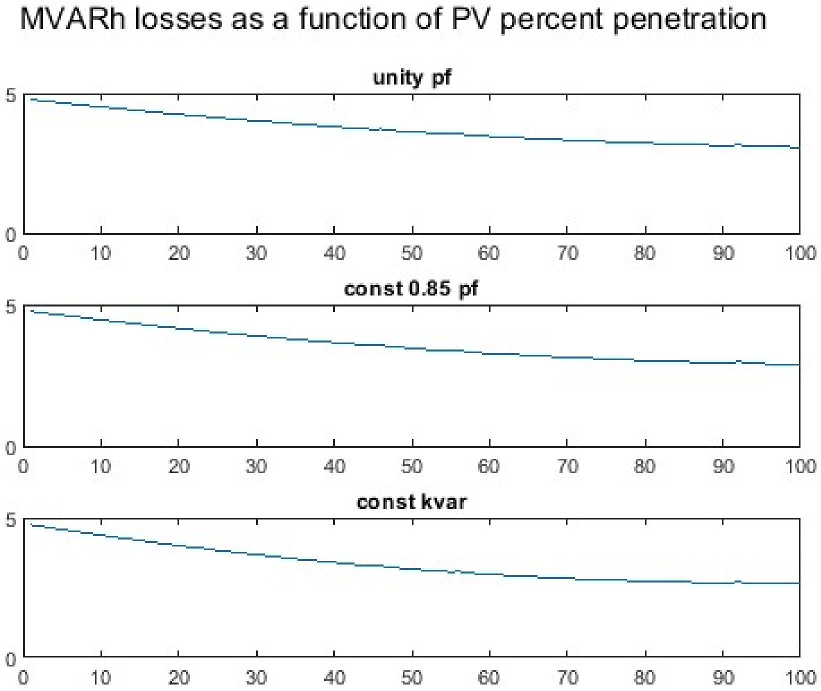


Figure 5-4 – 123-node MVARh losses



34-Node Optimization: After we implemented the power-flow into MATLAB and determined our primary and secondary nodes for community PV installation, we compared total system losses over a 24-hour period for each iteration.

Figure 5-5 – 34-node optimization results

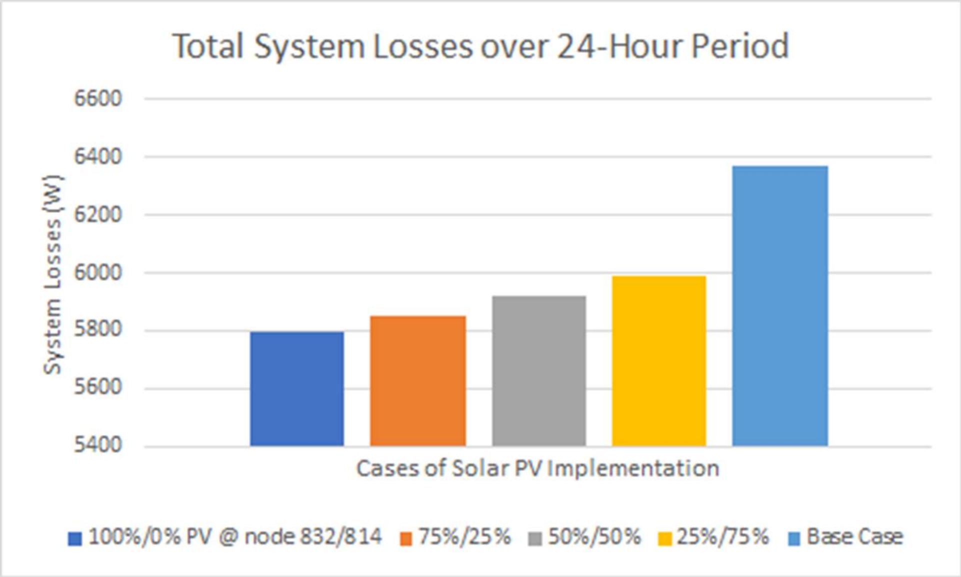


Figure 5.5 shows total system losses with different PV implementations on our primary and secondary nodes in each implementation. We started with the base case with no PV which produced a loss of 6.371 MW. We then added a 316 KW solar farm (size determined by MATLAB power-flow) at node 832 and reran the power-flow. This gave a system loss of 5.80 MW. As we changed the PV implementation (reducing at 832 and increasing at 814), we started to see total system losses rise. Based on these results, we determined the best solution for maximum system loss reduction was to only have one solar farm implemented at node 832, reducing the system loss by 9.01%.

6. Closing Material

6.1 Conclusion

We started by understanding the power-flow solution given in the textbook example [1] and implemented this solution in both MATLAB and OpenDSS. We then moved on to the 34-node network simulation using MATLAB and OpenDSS for modeling and analysis. Next, load shape was implemented to determine how the system responds over a 24-hour period in order to see what changes needed to be made to the network. With all of this, we began simulation of PV injection via community solar and individual household solar to determine which option was more viable.

In the second semester of our project, our work diverged into two subprojects. One was focused on simulating and analyzing another IEEE test feeder case, the 123-Node system. We were able to successfully adjust the system to run as desired and found that voltage control via only PV inverter control was possible. Based on this finding, the minimum %PV to ensure voltage control was determined for all levels of %KVAR injection. A specific level of KVAR injection (44% of inverter capacity reserved) for a hypothetical utility company was examined to determine the system losses as %PV penetration increased. Based on these 2 conclusions, the recommendation was made to operate in 44% KVAR injection for all PV penetration levels less than 81%, for the purpose of loss minimization, and operation along the developed curve above 81%, to ensure maximum power production from the PV while maintaining voltage within 0.95-1.05 p.u. quantities. The other subproject was 34-Node optimization, with the goal of reducing power losses. This subproject had the result of determining that the optimal type was community PV and the optimal place to inject PV was at node 832 with 316kw of PV being injected at that site. Furthermore, we also concluded that the type of control used, constant power factor, was not sufficient for this system since the voltage at node 814, the secondary injection site, did not improve as the PV injection was split between nodes 832 and 814 and power losses actually increased as less was added to node 832.

In our efforts in this project, we have completed power analysis of several distribution systems and reviewed the impact of solar energy penetration. While we were not able to analyze real life systems as was originally planned, our work will serve as a springboard for future groups working in this area. To this end, we have annotated our code and provided instruction to future users.

6.2 References

[1] W. H. Kersting, *Distribution System Modeling and Analysis*, 3rd ed. Boca Raton, FL: CRC Press, 2012 pp. 141-390

[2] K. Seepromting, R. Chatthaworn, P. Khunkitti, A. Kruesubthaworn, A. Siritaratiwat and C. Surawanitkun, "Optimal Grid-Connected with Multi-Solar PV Placement and Sizing for Power Loss Reduction and Voltage Profile Improvement," *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*, Bangkok, 2018, pp. 479-483.

[3] D. Q. Hung, N. Mithulananthan and K. Y. Lee, "Determining PV Penetration for Distribution Systems With Time-Varying Load Models," in *IEEE Transactions on Power Systems*, vol. 29, no. 6, pp. 3048-3057, Nov. 2014.

Appendix I: Operation Manual

MATLAB Control of OpenDSS

MATLAB control of the OpenDSS solver is accomplished through the use of a COM interface. While the standalone .exe version of OpenDSS can be run without installation, use of the COM server requires OpenDSS to be registered in Windows, which does require installation. If the system the program is being run on does not have administrator privileges, contact your system administrator about installation of OpenDSS.

The OpenDSS script that is compiled to run the daily solution set is not changed during a solution set. Changes to the OpenDSS model are accomplished by changing the contents of .txt files that are "redirected" by OpenDSS when the script is compiled. This ensures that while system model is changed for each iteration, the solution settings remain constant. If a specific OpenDSS command is needed however, the DSSStart.m script does provide the means by which a command can be inserted into the DSS.

A common assumption in the MATLAB script is that all buses are referred to by a 3 character string, with loads, lines, and PVsystem objects receiving names corresponding to the buses to which they are attached, e.g. a line object connecting bus 800 and 802 would be referenced as line.800802, a single phase load attached to phase c of bus 802 would be referenced as load.800cs, etc. Due to buses being handled internally by the MATLAB script as strings rather than numbers, the scripts should be extensible to larger systems by use of alphabetical characters to refer to more than 999 buses. This functionality is not guaranteed, and some troubleshooting may be required if scripts have not been fully updated to refer to bus names as strings internally. Due to the numbering of the buses in the 34-Node and 123-Node systems, busName and busNumber are generally interchangeable.

File paths used in the MATLAB and OpenDSS scripts assume a particular file system naming and organizational structure. Particularly, the OpenDSS model assumes that all loads fall into 2 categories, distributed or spot, and that all distributed loads and all spot loads are contained within the same .txt file that is redirected by the OpenDSS master script. Some OpenDSS models separate the scripts defining line, loads, and other circuit components into rural, suburban, and urban models. For use with our scripting assumptions, these would need to be consolidated into a single model, or the scripts used to update the model between iterations would need to be significantly overhauled.

Text files used by OpenDSS must be saved with ANSI encoding. This is not the default output from the MATLAB printf() function used in file generation. Conversion of an existing .txt file to ANSI encoding can be done by opening the file in Notepad++, changing the encoding, and resaving the file. To produce ANSI encoded .txt files directly from a MATLAB script, slCharacterEncoding('Windows-1252'); should be added prior to any printf() statements.

Code used for optimization of the 34-node system should not be considered extensible.

Appendix II: Initial goals

Our goals for this project were to familiarize ourselves with power-flow algorithms so that we could model and study medium to large distribution networks. Specifically, we were building our knowledge so we could model a network from Alliant Energy and implement PV in order to recommend the best installation and implementation method. However, our group was unable to maintain any steady line of communication with Alliant Energy and due to the COVID-19 pandemic, our goals were altered to model two different IEEE distribution systems: 34-node and 123-node.

We also wanted to have active control over PV inverter modes throughout our simulations. However, due to the constraints of running the simulations, fixed inverter settings were used for our simulations. Dynamic inverter settings would allow for a more optimal voltage control scheme, where KVAR injection would be minimized during periods of high solar power production, and greater KVAR injection as power began to decrease, specifically in the early evening when the combination of maximum power demand and decreasing solar production leads to large voltage droops.

Appendix III: Code

MATLAB Code

createMeters.m

```
function createMeters(lineDataFilename, numNodes)
%Creates a meter object at every bus in a system for use in finding
both
%the total losses in the system, as well as the losses at each node.
This
%file is extensible to all systems with less than 999 nodes, and
assumes
%that the OpenDSS line objects consist of 6 digit names, wehre the
first 3
%digits specify the sending bus, and the last 4 are the receiving bus.
For
%systems with node names that cannot be expressed in 3 digits,
significant
%rework to all script will be needed.
    slCharacterEncoding('Windows-1252');

    meterstrFormat = 'New Energymeter.%s line.%s 1 losses=YES\n';
    filepath = sprintf('%dNode/OpenDSStxtfiles/%s',numNodes,
lineDataFilename);

    lineCells = readcell(filepath);

    if numNodes ==34
        lineCodes = lineCells(:,2);
        for i = 1:length(lineCodes)
            str = lineCodes(i);
            lineNames(i,1) = string(str);
            lineStr = char(str);
            lineNum = lineStr(6:11);
            lineNums(i) = string(lineNum);
        end
    end
    if numNodes == 123
        lineCodes = lineCells(:,1);
        for i = 1:length(lineCodes)
            str = lineCodes(i);
            lineNames(i,1) = string(str);
            lineStr = char(str);
            lineNum = lineStr(10:15);
            lineNums(i) = string(lineNum);
        end
    end
    %% fopen
    outFilename = sprintf('C:\\may2020-
57/%dNode/openDSStxtfiles/busLossMeters.txt', numNodes);
    docoutput = fopen(outFilename, 'w');

    %% Iterate over all lines to create monitors
    for i = 1:length(lineCodes)
        fprintf(docoutput, meterstrFormat, lineNums(i), lineNums(i));
    end
    fclose all;

end
```

createMonitors.m

```
function createMonitors(lineDataFilename)
    slCharacterEncoding('Windows-1252');

    monitorstrFormat = 'New Monitor.%sMonitor mode=0 element=%s 2\n';
    filepath = sprintf('123Node/OpenDSStxtfiles/%s',lineDataFilename);

    lineCells = readcell(filepath);
    lineCodes = lineCells(:,2);

    docoutput1 = fopen('C:\may2020-
57/123Node/openDSStxtfiles/busVoltageMonitors.txt', 'w');
    docoutput2 = fopen('C:\may2020-
57/123Node/Results/monitorExports.txt', 'w');

    numMonitors = 0;
    for i = 1:length(lineCodes)
        str = lineCodes(i);
        lineNames(i,1) = string(str);
        busStr = char(str);
        if busStr(1) ~= 'r'
            numMonitors = numMonitors + 1;
            busNum = busStr(9:11);
            busNums(numMonitors) = string(busNum);
            fprintf(docoutput1, monitorstrFormat, busNums(numMonitors),
lineNames(i));
            fprintf(docoutput2, 'export monitor %sMonitor\n',
busNums(numMonitors));
        end
    end
    fclose all;
end
```

getBusNumbers.m

```
function busNumbers = getBusNumbers(lineDataFilename,numNodes)
    monitorstrFormat = 'New Monitor.%sMonitor mode=0 element=%s 2\n';
    filepath =
sprintf('%dNode/OpenDSStxtfiles/%s',numNodes,lineDataFilename);

    lineCells = readcell(filepath);
    if numNodes == 34
        lineCodes = lineCells(:,2);
        for i = 1:length(lineCodes)
            str = lineCodes(i);
            lineNames(i,1) = string(str);
            busStr = char(str);
            busNum = busStr(9:11);
            busNumbers(i) = string(busNum);
        end
    end
    if numNodes == 123
        lineCodes = lineCells(:,1);
         for i = 1:length(lineCodes)
            str = lineCodes(i);
            lineNames(i,1) = string(str);
            busStr = char(str);
            busNum = busStr(13:15);
            busNumbers(i) = string(busNum);
         end
    end
end
```

getMonitorData.m

```
function busVoltages = getMonitorData(busNum, openDSSCircuitName,
numNodes)
```

```
    filename =
sprintf('%dNode/Results/%s_Mon_%smonitor_1.csv',numNodes,
openDSSCircuitName, busNum);
    fopen(filename);
    monitorData = readtable(filename);
    hours = monitorData{:,1};
    sec = monitorData{:,2};
    v1 = monitorData{:,3};
    busVoltages.v2 = 0.*v1;
    busVoltages.v3 = 0.*v1;
    if monitorData.Properties.VariableNames{5} == 'V2'
        v2 = monitorData{:,5};
        busVoltages.v2 = v2;
        if monitorData.Properties.VariableNames{7} == 'V3'
            v3 = monitorData{:,7};
            busVoltages.v3 = v3;
        end
    end
   if monitorData.Properties.VariableNames{5} == 'V3'
        v3 = monitorData{:,5};
        busVoltages.v3 = v3;
    end

    busVoltages.v1 = v1;



    len = length(hours);
    for i=1:len
        busVoltages.time(i,1) = hours(i) + sec(i)/3600;
    end
end
```

distloadGen.m

```
%% Script description
%Converts a csv file from an IEEE test feeder describing distributed
loads
%into a text file readable by OpenDSS. The load is evenly split between
the
%2 buses along which the load is distributed. Note that this is not a
fully
%accurate assumption. If used in future, this script should be updated
to
%place 2/3 of the loading at 1/4 of the distance between the buses, and
the
%remaining one third at the destination bus in order to get fully
accurate
%losses. The assumption used in this script does produce the correct
%voltage drop.
```

```
clc
clear all
slCharacterEncoding('Windows-1252'); %needed for .txt tilfes to be read
by OpenDSS

%Filepath of the source and destination files
loaddata = readcell('C:\may2020-57/feeder123/feeder123/spot load
data.csv');
DocOutput = fopen('C:\may2020-
57/123Node/OpenDSStxtfiles/spotloadData.txt','w');

%The csv file has header type information in the first 4 lines that can
be
%freely discarded.
for n=5:size(loaddata,1)-1
    %Determines the load type and assigns the appropriate code for
openDSS
    if loaddata{n,4} == 'PQ'
        modeltype=1;
    elseif loaddata{n,4} == 'I'
        modeltype=5;
    elseif loaddata{n,4} == 'Z'
        modeltype=2;
    end
    %Parses the input to find the buses the loads are attached to and
the
    %kW and kVA rating of each
    bus1 = loaddata{n,1};
    bus2 = loaddata{n,2};
    kWA = loaddata{n,5}/2;
    kVARA = loaddata{n,6}/2;
    kWB = loaddata{n,7}/2;
    kVARB = loaddata{n,8}/2;
    kWC = loaddata{n,9}/2;
    kVARC = loaddata{n,10}/2;

    %Vminpu=0.85 ensures proper power flow solution if the regulators
in
    %the systems are turned off to get an idea of unregulated
performance.
    if loaddata(n, 3) == "Y"
        conntype = "Wye";
        kV = 24.9/sqrt(3);
        formatSpecA1 = 'New load.%d%da Phases=%d Bus1=%d.1 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
        formatSpecB1 = 'New load.%d%db Phases=%d Bus1=%d.2 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
```

```
    formatSpecC1 = 'New load.%d%dc Phases=%d Bus1=%d.3 conn=%s model=%d
kv=%f kw=%f kVar=%f vminpu=0.85 \n';
    else
        conntype = "Delta";
        kV = 24.9;
        if (kWA&&kWB || kWA&&kWC || kWB&&kWC)
            formatSpecA1 = 'New load.%d%da Phases=%d Bus1=%d.1.2
conn=%s model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecB1 = 'New load.%d%db Phases=%d Bus1=%d.2.3
conn=%s model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecC1 = 'New load.%d%dc Phases=%d Bus1=%d.3.1
conn=%s model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
        else
            formatSpecA1 = 'New load.%d%da Phases=%d Bus1=%d.1 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecB1 = 'New load.%d%db Phases=%d Bus1=%d.2 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecC1 = 'New load.%d%dc Phases=%d Bus1=%d.3 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
        end
    end

    %Producing load objects withg zero loading created floating point
math
    %errors in the OpenDSS solution, so these loads are discarded.
    if (kWA ~= 0)

fprintf(DocOutput,formatSpecA1,bus1,bus2,1,bus1,conntype,modeltype,kV,k
WA,kVARA);

fprintf(DocOutput,formatSpecA1,bus2,bus1,1,bus2,conntype,modeltype,kV,k
WA,kVARA);
    end
    if (kWB ~= 0)

fprintf(DocOutput,formatSpecB1,bus1,bus2,1,bus1,conntype,modeltype,kV,k
WB,kVARB);

fprintf(DocOutput,formatSpecB1,bus2,bus1,1,bus2,conntype,modeltype,kV,k
WB,kVARB);
    end
     if (kWC ~= 0)

fprintf(DocOutput,formatSpecC1,bus1,bus2,1,bus1,conntype,modeltype,kV,k
WC,kVARC);

fprintf(DocOutput,formatSpecC1,bus2,bus1,1,bus2,conntype,modeltype,kV,k
WC,kVARC);
     end
     fprintf(DocOutput, "\n");
```

```
end
```

distloadMod.m

```
%% Produces a .txt file in appropriate format to apply modifiers to
distributed load files
%This script reads a .dss or .txt file that is ofrmatted for OpenDSS
and
%adds a set of modifiers to allow control of loadshape, etc.
automatically



%% Code
clear all
clc

filename = 'C:\may2020-57/123node/OpenDSStxtFiles/spotloadData';
inputFilename = strcat(filename, '.txt');
input = fopen(inputFilename, 'r');

inputData = readtable(inputFilename);

loadshapestr = '';
xlsData = readcell('C:\may2020-57/ExcelFiles/loadshapeData.xlsx');%%
Read the load shape xlsx file

for n = 2:49 %convert the load shape data into a vector and format the
string used by opendss
    loadshape(n-1) = xlsData{n,9};
    strformat = ' %f';
    loadshapestr = strcat(loadshapestr, sprintf(strformat, loadshape(n-
1)));% Produces a string that is a row vector of all the vlaues for the
loadshape
end

outputFilename = strcat(filename, 'Modifiers.txt');
output = fopen(outputFilename, 'w');% Sets the modifier output to be
the inputfilename + modifiers
[length, trash] = size(inputData);%Find the length of the input data
loadNamesPrelimCell = inputData.Var1;%Get the names of the loads that
are being modified

for i=1:length
    str = loadNamesPrelimCell{i};
    loadNames(i,1) = string(str(5:14));
end
```

```
modifierformat = 'loadshape=[%s]'; %All loads will have the same
loadshape modifier
modifier = sprintf(modifierformat, loadshapestr);%Make the string
%% Changed to nothing when xls data not needed
modifier="";
%%
formatString = '%s.%sdaily=loadsh\n';%Format the output
for i=1:length
    fprintf(output,formatString, loadNames(i), modifier); %Output
should be load.busname.modifier=[row vector]
end
```

distPVgen.m

```
function distPVgen(percPen, distLoadFileName, Model, Variable,
numNodes)
%DISTPVGEN creates distributed PVSystem objects for each distibuted
load
%
%percPen is the percentage of peak real load at each node the PV can
%supply e.g. percPen = 100 is all real loading covered by the solar
system
%
%distLoadFileName is a string containing the file name of the
distributed
%loads to be covered e.g. 'exampleDistLoadFilename'
%
%Model: 1=unity PF, 2=constPF, 3=constQ, 4=variablePF
%
%Variable is the applicable variable for the inverter control mode
%  Unity Power Factor    : any integer value
%  Constant Power Factor : [0,1]
%  Constant Q            : Q as a factor of max inverter rating [0,1]
%                          For instance 0.25 is 25% of kVA for Q
injection
%  Variable Q          : Power factor determined by pervious solution
%
%Usage:
%  distPVgen(20,'distloadData34Node.dss',1,0) adds 20% PV penetration
to
%    the 34 node IEEE test feeder at unity power factor
%  distPVgen(20,'distloadData34Node.dss',3,0.75) uses 75% of inverter
%    rating for Q injection at each distibuted PVSystem
%{
percPen = 20;
distLoadFileName = 'distloadData34Node.txt';
Model = 1;
Variable = 1;
%}
```

```
    slCharacterEncoding('Windows-1252');
    pcPen = percPen/100;
    inputFile = fopen(distLoadFileName, 'r');
    inputData = readtable(distLoadFileName,'delimiter', '=');
    fclose('all');
    output = fopen(sprintf('C:\may2020-
57/%dnode/OpenDSStxtFiles/distPV34Node.txt', numNodes), 'w');% Sets the
output to be the PVsystem File
    [length, trash] = size(inputData);%Find the length of the input
data
    loadNamesPrelimCell = inputData.Var1;%Get the names of the buses
that PV is being attached to
    %Extract Name of each object
    for i=1:length
        str = loadNamesPrelimCell{i};
        loadNames(i,1) = string(str(5:16));
        individualLoad = char(loadNames(i));
        PVname = string(individualLoad(6:12));
        PVSystemNames(i,1) = PVname;
    end

    %Extract connection type of PV
    loadConnPrelimCell = inputData.Var4;
    for i=1:length
        str = loadConnPrelimCell{i};
        loadConn(i,1) = string(str(1:9));
        individualConn = char(loadConn(i));
        PVConn = string(individualConn(1:3));
        PVSystemConns(i,1) = PVConn;
    end
    %Extract bus PV is connected to
    loadBusPrelimCell = inputData.Var3;
    for i=1:length
        str = loadBusPrelimCell{i};
        loadBuss(i,1) = string(str(1:10));
        individualBus = char(loadBuss(i));
        if individualBus(7) == 'c'
            strend = 5;
        else
            strend = 7;
        end
        PVbus = string(individualBus(1:strend));
        PVSystemBuses(i,1) = PVbus;
    end
    %Extract kV Rating of the PV
    %Note, variable names used to extract information from the text
file
    %are re-used
    loadBusPrelimCell = inputData.Var6;
    for i=1:length
```

```
        str = loadBusPrelimCell{i};
        loadBus(i,1) = string(str(1:10));
        individualBus = char(loadBus(i));
        PVbus = string(individualBus(1:9));
        PVSystemkVstr(i,1) = PVbus;
        PVSystemkV(i,1) = str2double(PVSystemkVstr(i));
    end
    %Extract kW Rating of the PV
    %Note, variable names used to extract information from the text
file
    %are re-used
    loadBusPrelimCell = inputData.Var7;
    for i=1:length
        str = loadBusPrelimCell{i};
        loadBus(i,1) = string(str(1:10));
        individualBus = char(loadBus(i));
        PVbus = string(individualBus(1:9));
        PVSystemkWstr(i,1) = PVbus;
        PVSystemkW(i,1) = str2double(PVSystemkWstr(i));

    end
    for n=1:length
        %Parses the input to find the buses the loads are attached to
and the
        %kW and kVA rating of each
        bus1 = PVSystemBuses(n);
        kW = pcPen * PVSystemkW(n);
        kVA = kW;
        %Does preformatting based on Wye vs Delta Connection
        if PVSystemConns(n) == "Wye"
            conntype = "Wye";
            formatSpec = 'New PVSystem.%s Phases=1 Bus1=%s conn=%s
kv=%s kVA=%d Pmpp=%d\n~ effcurve=Myeff P-Tcurve=MyPvsT Daily=MyIrrad
TDaily=MyTemp';
        else
            conntype = "Delta";
            formatSpec = 'New PVSystem.%s Phases=1 Bus1=%s conn=%s
kv=%s kVA=%d Pmpp=%d\n~ effcurve=Myeff P-Tcurve=MyPvsT Daily=MyIrrad
TDaily=MyTemp';
        end

        switch Model
            case 1
                pf = 1;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
            case 2
                pf = Variable;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
            case 3
                kVAR = Variable * kVA;
                str = sprintf(' kvar=%f \n', kVAR);
                formatSpec1 = strcat(formatSpec, str);
            case 4
                pf = Variable;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
        end

        fprintf(output, formatSpec1,PVSystemNames(n), PVSystemBuses(n),
conntype, PVSystemkVstr(n), kVA, kW);
        fprintf(output, '\n');
    end
    fclose(output);
end
```

DSSStartup.m

```
function [Start,Obj,Text] = DSSStartup(mydir)
    % Function for starting up the DSS
    % make sure the proper directory is the active filepath
    cd(mydir);
    %
    %instantiate the DSS Object
    Obj = actxserver('OpenDSSengine.DSS');
    %
    %Start the DSS. Without this statement, DSS will not run (gives
    %max_circuit=0 error). Only needs to be executed the first time
    %w/in a
    %Matlab session
    % Define the text interface
    Start = Obj.Start(0);
    Text = Obj.Text;
End
```

inputdata.m

```
clc
clear all
close all


%%
slCharacterEncoding('Windows-1252'); %needed for .txt tilfes to be read
by OpenDSS

%Output filepath
DocOutput = fopen('C:\may2020-
57/123Node/OpenDSStxtfiles/LineConfigs.txt','w');
```

```
%Reads the line data from a csv file produced from the IEE test feeder
%.xls files
linedata = readmatrix('C:\may2020-57/feeder123/feeder123/line
data.csv');
%Specifies the number of phases associated with each line
configuration.
%This will need to be changed for each new system depending on the
%configuations present in that system.
threephase = [1,2,3,4,5,6,12];
twophase = [7,8];
onephase = [9,10,11];

%Iterates over the line data and produces OpenDSS objects corresponding
to
%them. Some of the line definitions will need to be changed in order to
%support the definitions of the regulators as 3 single phase
%autotransformers.
for n=1:size(linedata,1)
    formatSpec3 = 'New Line.%s%s Phases=%d Bus1=%s Bus2=%s
linecode=config%d Length=%d units=ft \n';
    formatSpec2 = 'New Line.%s%s Phases=%d Bus1=%s.%d.%d.0
Bus2=%s.%d.%d.0 linecode=config%d Length=%d units=ft \n';
    formatSpec1 = 'New Line.%s%s Phases=%d Bus1=%s.%d.0 Bus2=%s.%d.0
linecode=config%d Length=%d units=ft \n';
    config = linedata(n,4);
    %Determines the number of phases of the line based on the
configuration
    %type.
    if ismember(config, threephase)
        phases = 3;
    elseif ismember(config, twophase)
        phases = 2;
    else
        phases = 1;
    end
    %Turns the bus number into a string with zeros appended to the
front for buses that would not otherwise contain 3 digits. for systems
with more than 999 buses, all script will need significant overhaul,
since all functions relating to line configurations assume that line
objects have a 6 character numerical name, with the first 3 digits
representing the first bus, and the 4th through 6th digits representing
the second bus.
    bus1 = linedata(n,1);
    if bus1 <10
        bus1str = strcat("00", num2str(bus1));
    elseif bus1<100
        bus1str = strcat("0", num2str(bus1));
    else
        bus1str = num2str(bus1);
    end
    bus2 = linedata(n,2);
    if bus2 <10
        bus2str = strcat("00", num2str(bus2));
    elseif bus2<100
        bus2str = strcat("0", num2str(bus2));
    else
        bus2str = num2str(bus2);
    end
    len = linedata(n,3);
    %Prints the data, based on which phases are active
    if phases== 3

fprintf(DocOutput,formatSpec3,bus1str,bus2str,phases,bus1str,bus2str,co
nfig,len);
    end
    if phases == 2
        if config == 7
            subBus1 = 1;
            subBus2 = 3;
        else
            subBus1=1;
            subBus2=2;
        end
        fprintf(DocOutput,formatSpec2,bus1str, bus2str, phases,
bus1str, subBus1, subBus2, bus2str, subBus1,subBus2,config, len);
    end

    if phases == 1
        if config == 9
            subBus = 1;
        elseif config == 10
            subBus = 2;
        elseif config == 11
            subBus = 3;
        end

fprintf(DocOutput,formatSpec1,bus1str,bus2str,phases,bus1str,subBus,bus
2str,subBus,config,len);
    end

end
fclose(DocOutput);
```

loadcsvConverter.m

```
clc
clear all

%% Script Information
```

```
%Sanitizes the load .xlsx files for use in the distloadgen.m and
%spotloadgen.m files. The primarily consists of separating the
connection
%and mode into 2 separate columns, and spacing the rest of the data
%accordingly.
data = readcell('C:/may2020-57/feeder123/feeder123/spot loads
data.xls');
[sz,trash] = size(data);
csvdata = cell(sz,9);

csvdata{1,1} = data{1,1};
csvdata{3,1} = data{3,1};

for n=3:4
    for k=8:-1:3
        csvdata{n,k+1} = data{n,k};
    end

end

for n=5:sz-1
    for k=8:-1:3
        csvdata{n,k+1} = data{n,k};
    end
    datastr = data{n,2};
    csvdata{n,3} = datastr(3);
    if csvdata{n,3} == "P"
        csvdata{n,3} = "PQ";
    end
    csvdata{n,2} = data{n,2}(1);
    csvdata{n,1} = data{n,1};
end

writecell(csvdata, 'C:/may2020-57/feeder123/feeder123/spot load
data.csv');
```

lossCalc.m

```
function output = lossCalc(numNodes)
%Uses the output of the energymeter objects in OpenDSS to find the
total MW
%and MVAR losses in the system. THe number of nodes is needed to
specify
%the filepath. After the data is read, the lossmonitor.csv file is
deleted,
%preparing the system for another solution.
    filename = sprintf('C:/may2020-57/%dNode/Results/lossmonitor.csv',
numNodes);
    doc = fopen(filename);
    lossData = readtable(filename);
    output.MW = sum(lossData.ZoneLossesKWh)/1e3;
    output.MVAR = sum(lossData.ZoneLossesKvarh)/1e3;
    fclose(doc);
    delete(sprintf('C:/may2020-57/%dNode/Results/lossmonitor.csv',
numNodes))

end
```

OpenDSScontroller.m

```
clc
clear all
close all
opengl software
delete C:/may2020-57/34Node/Results/lossmonitor.csv
warning off MATLAB:table:ModifiedAndSavedVarnames
%% Starting OpenDSS server
% Start the openDSS COM server
%NOTE: OpenDSS must be installed to be registered to windows for use as
a
%COM server. Simply copying the .exe file to the drive will not work.
global DSSStartOk;
global DSSObj;
global DSSText;

[DSSStartOk, DSSObj, DSSText] = DSSStartup('C:\may2020-57/34Node');

%% Fetching the active circuit name

DSSCircuit = DSSObj.ActiveCircuit;


%% Getting Bus Numbers for use in monitor data collection
busNames = getBusNumbers('LineConfigs34Node.txt', 34);


%% Increasing PV penetration and changing mode of implementation
%This descrtibes the filepaths for the distributed and spot loads in
the
%system. THis allows integration of solar modeled as a percentage of
the
%customers at each node implementing solar individually.
distloadFilepath = 'C:\may2020-
57/34Node/OpenDSStxtfiles/distloadData34Node';
spotloadFilepath = 'C:\may2020-
57/34Node/OpenDSStxtfiles/spotloadData34Node';

%j represents the mode of operation of the solar inverters
%i is the percent integration
%n is used as an indexing variable for the coallation of data
```

```
for j = 1:3
    %The strings used in filenames and plot titles are defined based on
the
    %mode of operation of the silar inverters at the beginning of each
    %solution set
    switch j
        case 1
            plotTitle = 'unity pf';
            lossName = 'unity';
            var = 1;
        case 2
            plotTitle = 'const 0.85 pf';
            lossName = 'const_pf';
            var = 0.85;
        case 3
            plotTitle = 'const kvar';
            lossName = 'const_kvar';
            var = 0.5;
        otherwise
            plotTitle = 'var pf';
    end
    %Raising solar penetration from 1% to 100% in 1% increments
    for i=1:100
        %Generation of the
        distPVgen(i,distloadFilepath,j,var,34);
        spotPVgen(i,spotloadFilepath,j,var);
        DSSText.Command = ('compile 34NodeTestFeeder.dss');
        losstable = readtable('C:\may2020-
57/34Node/Results/lossmonitor.csv');
        for n = 1:length(busNames)
            busVoltages(i,n).name = busNames(n);
            busVoltages(i,n).voltage = getMonitorData(busNames(n),
'34NodeFeeder', 34);
        end
        for n = 1:height(losstable)
            nodeKWLosses(i+1,n) =
convertCharsToStrings(num2str(losstable.ZoneMaxKWLosses(n)));
            nodeKVARLosses(i+1,n) =
convertCharsToStrings(num2str(losstable.ZoneMaxKvarLosses(n)));
        end
        losses(i,j) = lossCalc(34);
        %OOS(i) = MonitorOOS(busVoltages(i),24900/sqrt(3));
        sprintf('%5.2f %% Complete',((j-1)+(i/100))*(100/3))
    end
    nodeKWLosses(1,:) = convertCharsToStrings(losstable.Meter)';
    nodeKVARLosses(1,:) = convertCharsToStrings(losstable.Meter)';
    kwWriteFilepath = sprintf('C:\\may2020-
57/34Node/Results/%s_kw_losses.csv', lossName);
    kvarWriteFilepath = sprintf('C:\\may2020-
57/34Node/Results/%s_kvar_losses.csv', lossName);
```

```
writematrix(nodeKWLosses, kwWriteFilepath);
writematrix(nodeKVARLosses, kvarWriteFilepath);

vnom = 1.4376e4;

%% Plotting Bus 824 Voltage
%% Plotting Bus 832 Voltage
figure(1);
v1 = busVoltages(1,25).voltage.v1 ./ vnom;
time = busVoltages(1,25).voltage.time;
v2 = busVoltages(25,25).voltage.v1 ./ vnom;
v3 = busVoltages(50,25).voltage.v1 ./ vnom;
v4 = busVoltages(75,25).voltage.v1 ./ vnom;
v5 = busVoltages(100,25).voltage.v1 ./ vnom;

subplot(3,1,j);
sgtitle('Bus 832 p.u. Voltage Without Regulators');
plot(time, v1, time, v2,time, v3, time, v4,time, v5);
xlim([0,24]);
ylim([0.9,1.1]);
title(plotTitle);
  legend('1%','25%','50%','75%','100%', 'Location', 'southeast');
%% Plotting bus 890 Voltage
figure(2);
vnom = 2.4e3;
v1 = busVoltages(1,32).voltage.v1 ./ vnom;
time = busVoltages(1,32).voltage.time;
v2 = busVoltages(25,32).voltage.v1 ./ vnom;
v3 = busVoltages(50,32).voltage.v1 ./ vnom;
v4 = busVoltages(75,32).voltage.v1 ./ vnom;
v5 = busVoltages(100,32).voltage.v1 ./ vnom;


subplot(3,1,j);
sgtitle('Bus 890 p.u. Voltage Without Regulators');
plot(time, v1, time, v2,time, v3, time, v4,time, v5);
xlim([0,24]);
ylim([0.9,1.1]);
title(plotTitle);
  legend('1%','25%','50%','75%','100%', 'Location', 'southeast');
%% Plotting Losses as a function of PV percent Penetration
figure(3);
for n = 1:100
    MWLosses(n) = losses(n,j).MW;
    MVARLosses(n) = losses(n,j).MVAR;
end
subplot(3,1,j);
sgtitle('MWh losses as a function of PV percent penetration');
plot(1:1:100, MWLosses);
xlim([0,100]);
```

```
    ylim([0,10]);
    title(plotTitle);


    figure(4);
    subplot(3,1,j);
    sgtitle('MVARh losses as a function of PV percent penetration');
    plot(1:1:100, MVARLosses);
    xlim([0,100]);
    ylim([-5,5]);
    title(plotTitle);
end
```

OpenDSScontroller123Node.m

```
clc
clear all
close all
opengl software
delete C:/may2020-57/123Node/Results/lossmonitor.csv %Each solution of
the DSS appends the energy meter data to the end of the specified file.
This ensures the first solution only has one set of data.
warning off MATLAB:table:ModifiedAndSavedVarnames %Disbales warning on
readtable operations
%% Starting OpenDSS server
% Start the DSS
global DSSStartOk;
global DSSObj;
global DSSText;

[DSSStartOk, DSSObj, DSSText] = DSSStartup('C:\may2020-57/123Node');

%% Changing the solution mode

DSSCircuit = DSSObj.ActiveCircuit;
%DSSText.Command = ('compile 34NodeTestFeeder.dss');

%% Getting Bus Numbers for use in monitor data collection
busNames = getBusNumbers('LineConfigs.txt', 123);


%% Increasing PV penetration

%Load filepaths. If distributed loads are included, those filepaths
should
%be specified here as well
spotloadFilepath = 'C:\may2020-
57/123Node/OpenDSStxtfiles/spotloadData';
```

```
for j = 1:3 %j is used as an indexing variable for the moder of
operation of the solar inverters
    switch j
        case 1
            plotTitle = 'unity pf';
            lossName = 'unity';
            var = 1;
        case 2
            plotTitle = 'const 0.85 pf';
            lossName = 'const_pf';
            var = 0.85;
        case 3
            plotTitle = 'const kvar';
            lossName = 'const_pf';
            var = 0.44;
        otherwise
            plotTitle = 'var pf'; %Note: Can be used for voltage
controlled operation, not used in this solution set
    end
    for i=1:100
        %Generates PV associated with spot loads
        spotPVgen123Node(i,spotloadFilepath,j,var, 0);

        DSSText.Command = ('compile 123NodeFeeder.dss'); %Runs the
power flow solution

        losstable = readtable('C:\may2020-
57/123Node/Results/lossmonitor.csv');%Reads the data from the loss
table

        for n = 1:length(busNames)
            %Finds bus voltages, as well as the minimum and max for
each solution
            busVoltages(i,n).name = busNames(n);
            busVoltages(i,n).voltage = getMonitorData(busNames(n),
'123NodeFeeder', 123);
            max1(i,n) = max(busVoltages(i,n).voltage.v1);
            max2(i,n) = max(busVoltages(i,n).voltage.v2);
            max3(i,n) = max(busVoltages(i,n).voltage.v3);
            min1(i,n) = min(busVoltages(i,n).voltage.v1);
            min2(i,n) = min(busVoltages(i,n).voltage.v2);
            min3(i,n) = min(busVoltages(i,n).voltage.v3);

        end

        for n = 1:height(losstable)
            %Finds the losses at each node in kw and kvar
            nodeKWLosses(i+1,n) =
convertCharsToStrings(num2str(losstable.ZoneMaxKWLosses(n)));
```

```
            nodeKVARLosses(i+1,n) =
convertCharsToStrings(num2str(losstable.ZoneMaxKvarLosses(n)));
        end

        % Finds the total losses in the system, and deletes
lossmonitor.csv in preparation for the next solution
        losses(i,j) = lossCalc(123);
        sprintf('%5.2f %% Complete',((j-1)+(i/100))*(100/3))
    end
    %Data conversion that prepares the data to be written to csv files
for
    %use in CPLEX for optimization.
    nodeKWLosses(1,:) = convertCharsToStrings(losstable.Meter)';
    nodeKVARLosses(1,:) = convertCharsToStrings(losstable.Meter)';
    kwWriteFilepath = sprintf('C:\\may2020-
57/123Node/Results/%s_kw_losses.csv', lossName);
    kvarWriteFilepath = sprintf('C:\\may2020-
57/123Node/Results/%s_kvar_losses.csv', lossName);
    writematrix(nodeKWLosses, kwWriteFilepath);
    writematrix(nodeKVARLosses, kvarWriteFilepath);

    %Nominal voltage for use in plotting. Does not affect the
underlying
    %data.
    vnom = 4.16e3/sqrt(3);

    %% Plotting Bus Voltage Profiles for Each Bus
    %Commented out to improve runtime. When included, this section
plots
    %the pu voltage of phase 1 for all buses in the system. a smaller
    %seciton of busNames can be selected as the indexing limit in order
to
    %select specific buses to monitor.
    %{
    for n = 1:length(busNames)
        figure(n);
        v1 = busVoltages(1,n).voltage.v1 ./ vnom;
        time = busVoltages(1,n).voltage.time;
        v2 = busVoltages(25,n).voltage.v1 ./ vnom;
        v3 = busVoltages(50,n).voltage.v1 ./ vnom;
        v4 = busVoltages(75,n).voltage.v1 ./ vnom;
        v5 = busVoltages(100,n).voltage.v1 ./ vnom;

        subplot(3,1,j);
        sgtitle(sprintf('Bus %s p.u. Voltage W/out Voltage Regulators',
busNames(n)));
        plot(time, v1, time, v2,time, v3, time, v4,time, v5);
        xlim([0,24]);
        ylim([0.9,1.1]);
        title(plotTitle);
        legend('1%','25%','50%','75%','100%', 'Location', 'southeast');
    end
    %}
    %% Plotting Losses as a function of PV percent Penetration
    k = n; % saves the number of buses for use in figure numbering,
freeing n for use as an indexing variable.
    figure(k+1);
    for n = 1:100
        MWLosses(n) = losses(n,j).MW;
        MVARLosses(n) = losses(n,j).MVAR;
    end
    subplot(3,1,j);
    sgtitle('MWh losses as a function of PV percent penetration');
    plot(1:1:100, MWLosses);
    xlim([0,100]);
    ylim([0,3]);
    title(plotTitle);


    figure(k+2);
    subplot(3,1,j);
    sgtitle('MVARh losses as a function of PV percent penetration');
    plot(1:1:100, MVARLosses);
    xlim([0,100]);
    ylim([0,5]);
    title(plotTitle);

    %Determines the max and min voltage in the system for each
solution.
    %Used to find the minimum PV penetration needed to get all voltages
    %within spec.
    maxV1(j,:) = max(max1');
    maxV2(j,:) = max(max2');
    maxV3(j,:) = max(max3');
    minV1(j,:) = min(min1');
    minV2(j,:) = min(min2');
    minV3(j,:) = min(min3');

end
minv1pu = minV1(3,1) ./ 2400;
k = 1;
while minv1pu <= 0.95
    k = k+1;
    minv1pu = minV1(3,k) ./ 2400
end
```

spotloadgen.m

```
%% Script description
%Converts a csv file from an IEEE test feeder describing spot loads
```

```
%into a text file readable by OpenDSS. The load is attached directly to
the
%bus specified in the .csv file. This script is extensible to other
IEEE
%test feeder by updating filepaths.

clear all
slCharacterEncoding('Windows-1252');

loaddata = readcell('C:\may2020-57/feeder123/feeder123/spot load
data.csv');
DocOutput = fopen('C:\may2020-
57/123Node/OpenDSStxtfiles/spotloadData.txt','w');

zero48kvbuses = [610];

for n=5:size(loaddata,1)-1
    if ismember(loaddata{n,1}, zero48kvbuses)
        baseKv = 0.48;
    else
        baseKv = 4.16;
    end
    if loaddata{n, 2} == 'Y'
        conntype = "Wye";
        kV = baseKv/sqrt(3);
    else
        conntype = "Delta";
        kV = baseKv;
    end
    if loaddata{n,3} == "PQ"
        modeltype=1;
    elseif loaddata{n,3} == "I"
        modeltype=5;
    elseif loaddata{n,3} == "Z"
        modeltype=2;
    end
    bus1 = loaddata{n,1};
    if bus1 <10
        bus1str = strcat("00", num2str(bus1));
    elseif bus1<100
        bus1str = strcat("0", num2str(bus1));
    else
        bus1str = num2str(bus1);
    end

    kWA = loaddata{n,4};
    kVARA = loaddata{n,5};
    kWB = loaddata{n,6};
    kVARB = loaddata{n,7};
    kWC = loaddata{n,8};
    kVARC = loaddata{n,9};

    if conntype == "Wye"
        if ~(kWA == kWB && kWA == kWC)
            formatSpecA1 = 'New load.%sas Phases=%d Bus1=%s.1 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecB1 = 'New load.%sbs Phases=%d Bus1=%s.2 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecC1 = 'New load.%scs Phases=%d Bus1=%s.3 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
        else
            formatSpec3 = 'New load.%scs Phases=%d Bus1=%s conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
        end

    elseif conntype == "Delta"
        if ~(kWA == kWB && kWA == kWC)
            formatSpecA1 = 'New load.%sas Phases=%d Bus1=%s.1.2 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecB1 = 'New load.%sbs Phases=%d Bus1=%s.2.3 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
            formatSpecC1 = 'New load.%scs Phases=%d Bus1=%s.3.1 conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
        else
            formatSpec3 = 'New load.%scs Phases=%d Bus1=%s conn=%s
model=%d kv=%f kw=%f kVar=%f vminpu=0.85 \n';
        end
    end

    if kWA == 0
        formatSpecA1 = '';
    end
    if kWB == 0
        formatSpecB1 = '';
    end
    if kWC == 0
        formatSpecC1 = '';
    end
    if ~(kWA == kWB && kWA == kWC)

fprintf(DocOutput,formatSpecA1,bus1str,1,bus1str,conntype,modeltype,kV,
kWA,kVARA);

fprintf(DocOutput,formatSpecB1,bus1str,1,bus1str,conntype,modeltype,kV,
kWB,kVARB);

fprintf(DocOutput,formatSpecC1,bus1str,1,bus1str,conntype,modeltype,kV,
kWC,kVARC);
    else
```

```
fprintf(DocOutput,formatSpec3,bus1str,3,bus1str,conntype,modeltype,kV,k
WA+kWB+kWC,kVARA+kVARB+kVARC);
    end
     fprintf(DocOutput, "\n");
end
fclose(DocOutput);
```

spotPVgen.m

```
function spotPVgen(percPen, spotLoadFileName, Model, Variable)
%DISTPVGEN creates PVSystem objects for each spot load
%
%percPen is the percentage of peak real load at each node the PV can
%supply e.g. percPen = 100 is all real loading covered by the solar
system
%
%spotLoadFileName is a string containing the file name of the
distributed
%loads to be covered e.g. 'exampleSpotLoadFilename'
%
%Model: 1=unity PF, 2=constPF, 3=constQ, 4=variablePF
%
%Variable is the applicable variable for the inverter control mode
%  Unity Power Factor    : any integer value
%  Constant Power Factor : [0,1]
%  Constant Q            : Q as a factor of max inverter rating [0,1]
%                          For instance 0.25 is 25% of kVA for Q
injection
%  Variable Q          : Power factor determined by pervious solution
%
%Usage:
%  spotPVgen(20,'spotloadData34Node.dss',1,0) adds 20% PV penetration
to
%    the 34 node IEEE test feeder at unity power factor
%  spotPVgen(20,'spotloadData34Node.dss',3,0.75) uses 75% of inverter
%    rating for Q injection at each distibuted PVSystem
%{
percPen = 20;
distLoadFileName = 'spotloadData34Node.txt';
Model = 1;
Variable = 1;
%}
%Note that this script is not extensible to other systems due to
%idosyncrasies in how readtable parses the load .txt file.
    slCharacterEncoding('Windows-1252');
    pcPen = percPen/100;
    inputFile = fopen(spotLoadFileName, 'r');
    inputData = readtable(spotLoadFileName,'delimiter', '=');
    fclose('all');
    output = fopen('C:\may2020-
57/34node/OpenDSStxtFiles/spotPV34Node.txt', 'w');% Sets the output to
be the PVsystem File
    [length, trash] = size(inputData);%Find the length of the input
data
    loadNamesPrelimCell = inputData.Var1;%Get the names of the buses
that PV is being attached to
    %Extract Name of each load object
    for i=1:length
        str = loadNamesPrelimCell{i};
        loadNames(i,1) = string(str(5:16));
        individualLoad = char(loadNames(i));
        PVname = string(individualLoad(6:10));
        PVSystemNames(i,1) = PVname;
    end

    %Extract phase number and connection type of PV
    loadConnPrelimCell = inputData.Var4;
    loadPhasePrelimCell = inputData.Var2;
    for i=1:length
        str = loadConnPrelimCell{i};
        loadConn(i,1) = string(str(1:9));
        individualConn = char(loadConn(i));
        PVConn = string(individualConn(1:3));
        PVSystemConns(i,1) = PVConn;
    end
    %Extract bus PV is connected to
    loadBusPrelimCell = inputData.Var3;
    for i=1:length
        str = loadPhasePrelimCell{i};
        str2 = loadBusPrelimCell{i};
        loadPhase = str(1);
            if loadPhase == '3'
                busCharNum = 3;
                numPhases(i) = 3;
            else
                busCharNum = 7;
                numPhases(i) = 1;
            end
        PVSystemBuses(i,1) = string(str2(1:busCharNum));
    end
    %Extract kV Rating of the PV
    %Note, variable names used to extract information from the text
file
    %are re-used
    loadBusPrelimCell = inputData.Var6;
    for i=1:length
        str = loadBusPrelimCell{i};
        loadBus(i,1) = string(str(1:10));
        individualBus = char(loadBus(i));
        PVbus = string(individualBus(1:9));
        PVSystemkVstr(i,1) = PVbus;
        PVSystemkV(i,1) = str2double(PVSystemkVstr(i));
    end
    %Extract kW Rating of the PV
    %Note, variable names used to extract information from the text
file
    %are re-used
    loadBusPrelimCell = inputData.Var7;
    for i=1:length
        str = loadBusPrelimCell{i};
        loadBus(i,1) = string(str(1:10));
        individualBus = char(loadBus(i));
        PVbus = string(individualBus(1:9));
        PVSystemkWstr(i,1) = PVbus;
        PVSystemkW(i,1) = str2double(PVSystemkWstr(i));

    end

    for n=1:length
        %Parses the input to find the buses the loads are attached to
and the
        %kW and kVA rating of each
        bus1 = PVSystemBuses(n);
        kW = pcPen * PVSystemkW(n);
        kVA = kW;
        %Does preformatting based on Wye vs Delta Connection
        if PVSystemConns(n) == "Wye"
            conntype = "Wye";
            formatSpec = 'New PVSystem.%s Phases=%d Bus1=%s conn=%s
kv=%s kVA=%d Pmpp=%d\n~ effcurve=Myeff P-Tcurve=MyPvsT Daily=MyIrrad
TDaily=MyTemp';
        else
            conntype = "Delta";
            formatSpec = 'New PVSystem.%s Phases=%d Bus1=%s conn=%s
kv=%s kVA=%d Pmpp=%d\n~ effcurve=Myeff P-Tcurve=MyPvsT Daily=MyIrrad
TDaily=MyTemp';
        end

        switch Model
            case 1
                pf = 1;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
            case 2
                pf = Variable;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
            case 3
                kVAR = Variable * kVA;
                str = sprintf(' kvar=%f \n', kVAR);
                formatSpec1 = strcat(formatSpec, str);
            case 4
                pf = Variable;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
        end

        fprintf(output, formatSpec1,PVSystemNames(n),numPhases(n),
PVSystemBuses(n), conntype, PVSystemkVstr(n), kVA, kW);
        fprintf(output, '\n');
    end
    fclose(output);
end
```

spotPVgen123Node.m

```
function spotPVgen123Node(percPen, spotLoadFileName, Model, Variable1,
Variable2)
%SPOTPVGEN123Node creates PVSystem objects for each spot load in the
123
%Node system
%
%percPen is the percentage of peak real load at each node the PV can
%supply e.g. percPen = 100 is all real loading covered by the solar
system
%
%spotLoadFileName is a string containing the file name of the
distributed
%loads to be covered e.g. 'exampleSpotLoadFilename'
%
%Model: 1=unity PF, 2=constPF, 3=constQ, 4=variablePF
%
%Variable is the applicable variable for the inverter control mode
%  Unity Power Factor    : any integer value
%  Constant Power Factor : [0,1]
%  Constant Q            : Q as a factor of max inverter rating [0,1]
%                          For instance 0.25 is 25% of kVA for Q
injection
%  Variable Q          : Power factor determined by pervious solution
%
%Usage:
%  spotPVgen(20,'spotloadData34Node.dss',1,0) adds 20% PV penetration
to
%    the 34 node IEEE test feeder at unity power factor
%  spotPVgen(20,'spotloadData34Node.dss',3,0.75) uses 75% of inverter
%    rating for Q injection at each distibuted PVSystem
%{
percPen = 20;
spotLoadFileName = 'spotloadData34Node.txt';
Model = 1;
Variable = 1;
%}
%Note that this script is not extensible to other systems due to
%idosyncrasies in how readtable parses the load .txt file.
    slCharacterEncoding('Windows-1252');
    pcPen = percPen/100;
    inputFile = fopen(spotLoadFileName, 'r');
    inputData = readtable(spotLoadFileName,'delimiter', '=');
    fclose('all');
    output = fopen('C:\may2020-57/123node/OpenDSStxtFiles/spotPV.txt',
'w');% Sets the output to be the PVsystem File
    [length, trash] = size(inputData);%Find the length of the input
data
    loadNamesPrelimCell = inputData.Var1;%Get the names of the buses
that PV is being attached to
    %Extract Name of each object
    for i=1:length
        str = loadNamesPrelimCell{i};
        loadNames(i,1) = string(str(5:14));
        individualLoad = char(loadNames(i));
        PVname = string(individualLoad(6:10));
        PVSystemNames(i,1) = PVname;
    end

    %Extract connection type of PV
    loadConnPrelimCell = inputData.Var4;
    for i=1:length
        str = loadConnPrelimCell{i};
        loadConn(i,1) = string(str(1:9));
        individualConn = char(loadConn(i));
        PVConn = string(individualConn(1:3));
        PVSystemConns(i,1) = PVConn;
    end
    %Extract bus PV is connected to
    loadBusPrelimCell = inputData.Var3;
    for i=1:length
        str = loadBusPrelimCell{i};
        loadBuss(i,1) = string(str(1:8));
        individualBus = char(loadBuss(i));
        if individualBus(7) == 'c'
            strend = 5;
            phases(i) = 1;
        elseif individualBus(7) == 'n'
            strend = 3;
            phases(i) = 3;
        else
            strend = 7;
            phases(i) = 1;
        end
        PVbus = string(individualBus(1:strend));
        PVSystemBuses(i,1) = PVbus;
    end
    %Extract kV Rating of the PV
    %Note, variable names used to extract information from the text
file
    %are re-used
    loadBusPrelimCell = inputData.Var6;
    for i=1:length
        str = loadBusPrelimCell{i};
        loadBus(i,1) = string(str(1:8));
        individualBus = char(loadBus(i));
        PVbus = string(individualBus(1:8));
        PVSystemkVstr(i,1) = PVbus;
        PVSystemkV(i,1) = str2double(PVSystemkVstr(i));
    end
    %Extract kW Rating of the PV
    %Note, variable names used to extract information from the text
file
    %are re-used
    loadBusPrelimCell = inputData.Var7;
    for i=1:length
        str = loadBusPrelimCell{i};
        loadBus(i,1) = string(str(1:10));
        individualBus = char(loadBus(i));
        PVbus = string(individualBus(1:9));
        PVSystemkWstr(i,1) = PVbus;
        PVSystemkW(i,1) = str2double(PVSystemkWstr(i));

    end
    for n=1:length
        %Parses the input to find the buses the loads are attached to
and the
        %kW and kVA rating of each
        bus1 = PVSystemBuses(n);
        kW = pcPen * PVSystemkW(n);
        kVA = kW;
        %Does preformatting based on Wye vs Delta Connection
        if PVSystemConns(n) == "Wye"
            conntype = "Wye";
            formatSpec = 'New PVSystem.%s Phases=%d Bus1=%s conn=%s
kv=%s kVA=%d Pmpp=%d\n~ effcurve=Myeff P-Tcurve=MyPvsT Daily=MyIrrad
TDaily=MyTemp';
        else
            conntype = "Delta";
            formatSpec = 'New PVSystem.%s Phases=%d Bus1=%s conn=%s
kv=%s kVA=%d Pmpp=%d\n~ effcurve=Myeff P-Tcurve=MyPvsT Daily=MyIrrad
TDaily=MyTemp';
        end
        switch Model
            case 1
                pf = 1;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
            case 2
                pf = Variable1;
                str = sprintf(' pf=%f \n',pf);
                formatSpec1 = strcat(formatSpec, str);
            case 3
                kVAR = Variable1 * kVA;
                str = sprintf(' kvar=%f \n', kVAR);
                formatSpec1 = strcat(formatSpec, str);
            case 4
                pf = Variable1;
                kVAR = Variable2;
                if pf == 0
                    str = sprintf(' kvar=%f, \n', kVAR);
                end
                if kVAR == 0
                    str = sprintf(' pf=%f, \n', pf);
                end
                formatSpec1 = strcat(formatSpec, str);
        end

        fprintf(output, formatSpec1,PVSystemNames(n),phases(n),
PVSystemBuses(n), conntype, PVSystemkVstr(n), kVA, kW);
        fprintf(output, '\n');
    end
    fclose(output);
end
```

MATLAB CODE FOR OPTIMIZATION

OpendDSSController.m

```
clc

clear all

close all

opengl software

delete C:/may2020-57/34Node/Results/lossmonitor.csv

warning off MATLAB:table:ModifiedAndSavedVarnames

%% Starting OpenDSS server

% Start the openDSS COM server

%NOTE: OpenDSS must be installed to be registered to windows for
use as a

%COM server. Simply copying the .exe file to the drive will not
work.

global DSSStartOk;

global DSSObj;

global DSSText;


[DSSStartOk, DSSObj, DSSText] = DSSStartup('C:\may2020-
57/34Node');


%% Fetching the active circuit name


DSSCircuit = DSSObj.ActiveCircuit;




%% Getting Bus Numbers for use in monitor data collection
```

```
busNames = getBusNumbers('LineConfigs34Node.txt', 34);



%% Increasing PV penetration and changing mode of implementation
%This descrtibes the filepaths for the distributed and spot loads
in the
%system. THis allows integration of solar modeled as a percentage
of the
%customers at each node implementing solar individually.
distloadFilepath = 'C:\may2020-
57/34Node/OpenDSStxtfiles/distloadData34Node';
spotloadFilepath = 'C:\may2020-
57/34Node/OpenDSStxtfiles/spotloadData34Node';


%j represents the mode of operation of the solar inverters
%i is the percent integration
%n is used as an indexing variable for the coallation of data
for j = 1:32
    %The strings used in filenames and plot titles are defined
based on the
    %mode of operation of the silar inverters at the beginning of
each
    %solution set
    %{
    switch j
        case 1
            plotTitle = 'unity pf';
            lossName = 'unity';
            var = 1;
        case 2
            plotTitle = 'const 0.85 pf';
            lossName = 'const_pf';
            var = 0.85;
        case 3
            plotTitle = 'const kvar';
            lossName = 'const_kvar';
            var = 0.5;
        otherwise
            plotTitle = 'var pf';
    end
    %}
    %Raising solar penetration from 1% to 100% in 1% increments
    doc34 = fopen('C:\may2020-
57/34Node/OpenDSStxtfiles/redirectPV.txt', 'w');
    str = sprintf('redirect optimizationPV%s.txt', busNames(j));
    fprintf(doc34, str);
    fclose('all');
    for i=1:1
        %Generation of the
        %distPVgen(i,distloadFilepath,1,var,34);
        %spotPVgen(i,spotloadFilepath,1,var);
        DSSText.Command = ('compile 34NodeTestFeeder.dss');
        losstable = readtable('C:\may2020-
57/34Node/Results/lossmonitor.csv');
        for n = 1:length(busNames)
            busVoltages(i,n).name = busNames(n);
            busVoltages(i,n).voltage =
getMonitorData(busNames(n), '34NodeFeeder', 34);

        end

        for n = 1:height(losstable)

            nodeKWLosses(i+1,n) =
convertCharsToStrings(num2str(losstable.ZoneMaxKWLosses(n)));

            nodeKVARLosses(i+1,n) =
convertCharsToStrings(num2str(losstable.ZoneMaxKvarLosses(n)));

        end

        losses(i,j) = lossCalc(34);

        sprintf('%5.2f %% Complete',((j-1)+(i/100))*(100/3))

    End
```

Optimization.m

```
%% Variable equations
ak = tan(acos(.85));
sigma1 = 1;
sigma2 = 1;
sigma3 = 1;

mile = 5280;
R_300 = 1.30814;
R_301 = 1.89095;
R_302 = 2.73327;
R_303 = 2.73327;
R_304 = 1.89067;
X_300 = 1.34918;
X_301 = 1.42374;
X_302 = 1.48505;
X_303 = 1.48505;
X_304 = 1.4241;

ndlen = [2580; 1730; 32230; 5804; 37500; 29730; 10; 1710;...
              10210; 48150; 13740; 3030; 840; 20440; 520;
4900;...
              2020; 280; 860; 280; 1350; 3640; 530; 310; 10;...
              23330; 36830; 1620; 5830; 2680; 4860; 10560];
ndlen = ndlen./5280;
```

```
R           = [R_300*ndlen(1),R_300*ndlen(2),R_300*ndlen(3),...
                 R_303*ndlen(4),R_300*ndlen(5),R_300*ndlen(6),...
                 R_301*ndlen(7),R_302*ndlen(8),R_301*ndlen(9),...
                 R_302*ndlen(10),R_302*ndlen(11),R_303*ndlen(12),...
                 R_301*ndlen(13),R_301*ndlen(14),R_301*ndlen(15),...
                 R_301*ndlen(16),R_301*ndlen(17),R_301*ndlen(18),...
                 R_301*ndlen(19),R_301*ndlen(20),R_301*ndlen(21),...
                 R_301*ndlen(22),R_301*ndlen(23),R_301*ndlen(24),...
                 R_301*ndlen(25),R_303*ndlen(26),R_301*ndlen(27),...
                 R_302*ndlen(28),R_301*ndlen(29),R_301*ndlen(30),...
                 R_304*ndlen(31),R_300*ndlen(32)];
X           = [X_300*ndlen(1),X_300*ndlen(2),X_300*ndlen(3),...
                 X_303*ndlen(4),X_300*ndlen(5),X_300*ndlen(6),...
                 X_301*ndlen(7),X_302*ndlen(8),X_301*ndlen(9),...
                 X_302*ndlen(10),X_302*ndlen(11),X_303*ndlen(12),...
                 X_301*ndlen(13),X_301*ndlen(14),X_301*ndlen(15),...
                 X_301*ndlen(16),X_301*ndlen(17),X_301*ndlen(18),...
                 X_301*ndlen(19),X_301*ndlen(20),X_301*ndlen(21),...
                 X_301*ndlen(22),X_301*ndlen(23),X_301*ndlen(24),...
                 X_301*ndlen(25),X_303*ndlen(26),X_301*ndlen(27),...
                 X_302*ndlen(28),X_301*ndlen(29),X_301*ndlen(30),...
                 X_304*ndlen(31),X_300*ndlen(32)];
for i = 1:32
    P(i) =
(busVoltages(1,i).voltage.v1(1)*cosd(busVoltages(1,i).voltage.van
gle(1)))...

*((busVoltages(1,i).voltage.i1(1))*cosd(busVoltages(1,i).voltage.
iangle(1)));
    Q(i) =
((busVoltages(1,i).voltage.v1(1)*sind(busVoltages(1,i).voltage.va
ngle(1))))...

*((busVoltages(1,i).voltage.i1(1))*sind(busVoltages(1,i).voltage.
iangle(1)));
    V(i) = busVoltages(1,i).voltage.v1(1);
    Pl(i) = (P(i)^2+Q(i)^2)*R(i)/(abs(V(i)))^2;
    Ql(i) = (P(i)^2+Q(i)^2)*X(i)/(abs(V(i)))^2;
end
    V(33) = V(32);
for i = 1:32
    Ak(i) = (R(i)*P(i))/V(i);
    Bk(i) = (R(i)*Q(i))/V(i);
    Ck(i) = (R(i))/V(i);
    Dk(i) = (X(i)*P(i))/V(i);
    Ek(i) = (X(i)*Q(i))/V(i);
    Fk(i) = (X(i))/V(i);
    Gk(i) = (R(i)^2)/V(i+1);
    Hk(i) = (R(i)^2*P(i))/V(i+1);
    Ik(i) = X(i)^2/V(i+1);
    Jk(i) = (X(i)^2)/V(i+1);
    Kk(i) = (X(i)^2*Q(i))/V(i+1);
    Lk(i) = (R(i)*X(i)*Q(i))/V(i+1);
    Mk(i) = (R(i)*X(i))/V(i+1);
    VD_2(i) = (((R(i)*P(i)+X(i)*Q(i))^2)/V(i+1));

    Ppvk(i) = (((sigma1/Pl(i))*(Ak(i)+ak
*Bk(i)))+((sigma2/Ql(i))*(Dk(i)+ak
*Ek(i)))+((sigma3/VD_2(i))*(Hk(i)+ak
*Jk(i)+ak*Kk(i)+Lk(i))))/...

(((sigma1/Pl(i))*(Ck(i)+ak^2*Ck(i)))+((sigma2/Ql(i))*(Fk(i)+ak^2*
Fk(i)))+((sigma3/VD_2(i))*(Gk(i)+ak^2*Ik(i)+2*ak*Mk(i))));
%% calculate VDpv

    VDpart1(i) = (R(i)^2*(Ppvk(i)^2-
2*P(i)*Ppvk(i)))/abs(V(i+1))^2;
    VDpart2(i) = (X(i)^2*(ak^2*Ppvk(i)^2-
2*Q(i)*ak*Ppvk(i)))/abs(V(i+1))^2;
    VDpart3(i) = (R(i)*X(i)*(P(i)*ak*Ppvk(i)^2+Q(i)*Ppvk(i)-
ak*Ppvk(i)^2))/abs(V(i+1))^2;

    VD_2pv(i) = VDpart1(i) + VDpart2(i) - 2*VDpart3(i) + VD_2(i);
    IVD(i) = VD_2pv(i)/VD_2(i);

%% calculate Plpv

    Plpart1(i) = ((Ppvk(i)^2-2*P(i)*Ppvk(i))*R(i))/(abs(V(i))^2);
    Plpart2(i) = (ak^2*Ppvk(i)^2-
2*Q(i)*ak*Ppvk(i)*R(i))/(abs(V(i))^2);

    Plpv(i) = Plpart1(i) + Plpart2(i) + Pl(i);
    ILP(i) = Plpv(i)/Pl(i);

%% calculate Qlpv
    Qlpart1(i) = ((Ppvk(i)^2-2*P(i)*Ppvk(i))*X(i))/(abs(V(i))^2);
    Qlpart2(i) = ((ak^2*Ppvk(i)^2-
2*Q(i)*ak*Ppvk(i))*X(i))/(abs(V(i))^2);

    Qlpv(i) = Qlpart1(i) + Qlpart2(i) + Ql(i);
    ILQ(i) = Qlpv(i)/Ql(i);
    IMO(i) = sigma1*ILP(i)+sigma2*ILQ(i)+sigma3*IVD(i);
End
```

GetMonitorData.m

```
function busVoltages = getMonitorData(busNum, openDSSCircuitName,
numNodes)

    filename =
sprintf('%dNode/Results/%s_Mon_%smonitor_1.csv',numNodes,
openDSSCircuitName, busNum);

    fopen(filename);

    monitorData = readtable(filename);

    hours = monitorData{:,1};

    sec = monitorData{:,2};

    v1 = monitorData{:,3};

    if monitorData.Properties.VariableNames{5} == 'I1'

        i1 = monitorData{:,5};

        vangle = monitorData{:,4};

        iangle = monitorData{:,6}-180;

    else

        i1 = monitorData{:,9};

        vangle = monitorData{:,4};

        iangle = monitorData{:,10}-180;

    end

    busVoltages.v2 = 0.*v1;

    busVoltages.i2 = 0.*i1;

    busVoltages.v3 = 0.*v1;

    busVoltages.i3 = 0.*i1;

    if monitorData.Properties.VariableNames{5} == 'V2'

        v2 = monitorData{:,5};
        i2 = monitorData{:,11};

        busVoltages.v2 = v2;

        busVoltages.i2 = i2;

    end

    if monitorData.Properties.VariableNames{5} == 'V2'

        v3 = monitorData{:,7};

        i3 = monitorData{:,13};

        busVoltages.v3 = v3;

        busVoltages.i3 = i3;

    end

    busVoltages.v1 = v1;

    busVoltages.i1 = i1;

    busVoltages.vangle = vangle;

    busVoltages.iangle = iangle;


    len = length(hours);

    for i=1:len

        busVoltages.time(i,1) = hours(i) + sec(i)/3600;

    end

End
```

CreatePV.m

```
function createPV(Ppvk, buses)

    slCharacterEncoding('Windows-1252');

    strformat = 'New PVSystem.%s Phases=%d Bus1=%s conn=%s kv=%s
kVA=%d Pmpp=%d\n~ effcurve=Myeff P-Tcurve=MyPvsT Daily=MyIrrad
TDaily=MyTemp pf=0.85\n';

    for i=1:length(buses)
```

```
        filepath = sprintf('C:\\may2020-
57/34Node/OpenDSStxtfiles/optimizationPV%s.txt', buses(i));
        docoutput = fopen(filepath, 'w');
        if ismember(buses(i), ['888', '890'])
            kv = 4.16;
        else
            kv = 24.9;
        end
        if ~ismember(buses(i), ['810', '818', '820', '822',
'826', '856', '864', '838'])

fprintf(docoutput,strformat,buses(i),3,buses(i),'delta',kv,
Ppvk(i)/1e3, Ppvk(i)/1e3);
        else
            strcat('!', strformat); fprintf(docoutput, '');
        end
        fclose('all');
    end
End
```

Losscalc.m

```
function output = lossCalc(numNodes)
    filename = sprintf('C:/may2020-
57/%dNode/Results/lossmonitor.csv', numNodes);
    doc = fopen(filename);
    lossData = readtable(filename);
    output.MW = sum(lossData.ZoneLossesKWh)/1e3;
    output.MVAR = sum(lossData.ZoneLossesKvarh)/1e3;
    fclose(doc);
    delete(sprintf('C:/may2020-
57/%dNode/Results/lossmonitor.csv', numNodes))


end
```