Large-Scale DER Aggregations of Electric Water Heaters and Battery Inverter Systems

Kevin Marnell
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Recommended Citation
https://doi.org/10.15760/etd.6973

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.
Large-Scale DER Aggregations of Electric Water Heaters and Battery Inverter Systems

by

Kevin Marnell

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Robert Bass, Chair
John M. Acken
Douglas V. Hall

Portland State University
2019
Abstract

Distributed energy resources like residential electric water heaters and residential battery-inverter systems offer a small amount of change to the grid individually. When aggregated however, these assets can cause major effects to the electric grid. Aggregating these resources allows them to take on generator-like functions with the ability to increment power and decrement power.

The Western Energy Imbalance Market is an energy market offering 15 minute and 5 minute markets for energy transactions between balancing areas. Generation assets make increment and decrement bids. Traditionally the only entrants to this market have been large scale generators and large scale assets legally designated as generators. Aggregated distributed resources could offer the same increments and decrements from managing residential assets like electric water heaters and batteries.

DERAS, a Distributed Energy Resource Aggregation System developed by the Portland State Power Lab group, is an aggregator of residential resources that could offer increment and decrement bids to an energy market, like an Energy Imbalance Market. This research models and simulates aggregations of distributed energy resources. This work analyzes the effects of 10,000 electric water heaters and 10,000 battery inverter systems. A simulation program was built to simulate regular use of these assets, and then add the additional effects of a decrement bid into the Western Energy Imbalance Market. The effects of the bids on
energy levels inside the water heaters and batteries are examined. The power imported from
the grid is also analyzed as an effect of the aggregator attempting to cover a generation
decrement bid.
Acknowledgements

Thank you to Pr. Robert Bass and the rest of the power lab researchers, Manny Obi, Tylor Slay, and Leighton Clarke.
# Contents

Abstract i  
Acknowledgements iii  
List of Tables vi  
List of Figures vii  

## 1 Introduction  
1.1 Problem Statement 1  
1.2 Objectives of Work 1  
1.3 Distributed Energy Resources (DER) 2  
1.4 Virtual Power Plant (VPP) 3  
1.5 DERAS 4  
1.6 Operator Class 5  
1.7 Balancing Area Authorities 6  
1.8 Energy and Imbalances Markets 7  
1.9 Literature Review 8  

## 2 Design Methodology  
2.1 Modeled Aggregated Units 12  
2.2 Electric Water Heaters 13  
2.3 EWH simulation in MATLAB 15  
2.3.1 Loading in the data 16  
2.3.2 Creating a look-up matrix for draw events 17  
2.3.3 Simulating EWH Behavior 17  
2.3.4 Simulating Bids 21  
2.4 Battery Inverter Systems 23  
2.5 Battery Simulation 24  
2.5.1 Loading in the data 25  
2.5.2 Simulating BIS behavior 26  
2.5.3 Simulating a Battery Bid 29  

## 3 Results & Analysis  
3.1 Results of simulating EWH assets without bids 32
3.2 Results of the EWH on Assets with bids ........................................ 34
3.3 Results of Battery-Inverter Systems without bids .......................... 56
3.4 Results of Battery-Inverter Systems with EIM Bids ..................... 60

4 Discussion ......................................................................................... 76
  4.1 EWH Aggregations ................................................................. 76
  4.2 BIS Aggregations ................................................................. 78

5 Conclusion ....................................................................................... 82

Bibliography ........................................................................................ 84

Appendix A: MATLAB Scripts ........................................................... 89
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Census Water Heater Makeup</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>EWH Calculations at each state</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Statuses of EWH and BESS assets.</td>
<td>26</td>
</tr>
<tr>
<td>2.4</td>
<td>Statuses of EWH and BESS assets.</td>
<td>29</td>
</tr>
<tr>
<td>2.5</td>
<td>BIS calculations at each state</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Maximum power of EWH aggregation</td>
<td>54</td>
</tr>
</tbody>
</table>
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>DERAS.</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>State diagram for electric water heater model.</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Aggregator dispatch algorithm.</td>
<td>22</td>
</tr>
<tr>
<td>2.3</td>
<td>State diagram for battery inverter system model.</td>
<td>28</td>
</tr>
<tr>
<td>2.4</td>
<td>BIS states for EIM dispatch.</td>
<td>31</td>
</tr>
<tr>
<td>3.1</td>
<td>Simulated EWH energy take with no bids.</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Simulated EWH draws and heating.</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Simulated EWH energy take from 5 MW bid.</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>Simulated EWH heating from a 5 MW bid.</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>Simulated EWH energy take from a 6 MW bid.</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Simulated EWH heating from a 6 MW bid.</td>
<td>38</td>
</tr>
<tr>
<td>3.7</td>
<td>Power absorption from 5 and 6 MW, 1 hour bids.</td>
<td>39</td>
</tr>
<tr>
<td>3.8</td>
<td>Power absorption for 2 hour bids.</td>
<td>40</td>
</tr>
<tr>
<td>3.9</td>
<td>Simulated EWH energy take from a 1.5 MW, 18 hour bid.</td>
<td>41</td>
</tr>
<tr>
<td>3.10</td>
<td>Next day energy take from a 1.5 MW, 18 hour bid.</td>
<td>42</td>
</tr>
<tr>
<td>3.11</td>
<td>Simulated EWH heating from a 1.5 MW, 18 hour bid.</td>
<td>43</td>
</tr>
<tr>
<td>3.12</td>
<td>Next day EWH heating from a 1.5 MW, 18 hour bid.</td>
<td>44</td>
</tr>
<tr>
<td>3.13</td>
<td>Simulated EWH energy take from a 1.6 MW, 18 hour bid.</td>
<td>45</td>
</tr>
<tr>
<td>3.14</td>
<td>Simulated EWH heating from a 1.6 MW, 18 hour bid.</td>
<td>46</td>
</tr>
<tr>
<td>3.15</td>
<td>Simulated EWH energy take from a 2.0 MW, 18 hour bid.</td>
<td>47</td>
</tr>
<tr>
<td>3.16</td>
<td>Simulated EWH heating from a 2.0 MW, 18 hour bid.</td>
<td>48</td>
</tr>
<tr>
<td>3.17</td>
<td>Power absorption for 18 hour bids.</td>
<td>49</td>
</tr>
<tr>
<td>3.18</td>
<td>Simulated EWH energy take from a 43 MW, 5 minute bid.</td>
<td>50</td>
</tr>
<tr>
<td>3.19</td>
<td>Simulated EWH heating from a 43 MW, 5 minute bid.</td>
<td>51</td>
</tr>
<tr>
<td>3.20</td>
<td>Maximum EWH power availability across 24 hours.</td>
<td>52</td>
</tr>
<tr>
<td>3.21</td>
<td>Maximum EWH power availability across 2 hours.</td>
<td>53</td>
</tr>
<tr>
<td>3.22</td>
<td>Maximum biddable energy up to 2 hours.</td>
<td>54</td>
</tr>
<tr>
<td>3.23</td>
<td>100 EWH aggregate energy take bidding 50 kW.</td>
<td>55</td>
</tr>
<tr>
<td>3.24</td>
<td>100 EWH aggregate power absorption over 1 hour.</td>
<td>56</td>
</tr>
<tr>
<td>3.25</td>
<td>State of charge for BIS aggregation with no bids.</td>
<td>57</td>
</tr>
<tr>
<td>3.26</td>
<td>Logic states for BIS aggregation with no bids.</td>
<td>58</td>
</tr>
<tr>
<td>3.27</td>
<td>Actions of BIS aggregation with no bids.</td>
<td>59</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.28</td>
<td>PJM RegA signal</td>
<td>60</td>
</tr>
<tr>
<td>3.29</td>
<td>BIS aggregate state of charge for 20 MW bid.</td>
<td>61</td>
</tr>
<tr>
<td>3.30</td>
<td>BIS aggregate logic states for 20 MW bid.</td>
<td>62</td>
</tr>
<tr>
<td>3.31</td>
<td>BIS aggregate actions for 20 MW bid.</td>
<td>63</td>
</tr>
<tr>
<td>3.32</td>
<td>BIS aggregate state of charge for 50 MW bid.</td>
<td>64</td>
</tr>
<tr>
<td>3.33</td>
<td>BIS aggregate logic states for 50 MW bid.</td>
<td>65</td>
</tr>
<tr>
<td>3.34</td>
<td>BIS aggregate actions for 50 MW bid.</td>
<td>66</td>
</tr>
<tr>
<td>3.35</td>
<td>Power absorbed by BIS aggregation over 2 hours.</td>
<td>67</td>
</tr>
<tr>
<td>3.36</td>
<td>BIS aggregate state of charge for 15 MW 18 hour bid.</td>
<td>68</td>
</tr>
<tr>
<td>3.37</td>
<td>BIS aggregate logic states for 15 MW 18 hour bid.</td>
<td>69</td>
</tr>
<tr>
<td>3.38</td>
<td>BIS aggregate actions for 15 MW 18 hour bid.</td>
<td>70</td>
</tr>
<tr>
<td>3.39</td>
<td>BIS aggregate state of charge for 20 MW 18 hour bid.</td>
<td>71</td>
</tr>
<tr>
<td>3.40</td>
<td>BIS aggregate logic states for 20 MW 18 hour bid.</td>
<td>72</td>
</tr>
<tr>
<td>3.41</td>
<td>BIS aggregate actions for 20 MW 18 hour bid.</td>
<td>73</td>
</tr>
<tr>
<td>3.42</td>
<td>BIS power absorption for 18 hr bids.</td>
<td>74</td>
</tr>
<tr>
<td>3.43</td>
<td>Maximum power output for BIS aggregation over 3 hours.</td>
<td>75</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Problem Statement

Distributed Energy Resources Management (DERM) systems have been recognized as a means to provide energy or manage loads to better operate the power grid. By injecting power to the grid or reducing load, DERM can accomplish a number of useful grid functions. When generation is greater than load, DERM can turn on additional loads to balance the power grid. When load is greater than generation, DERM can turn off loads to balance the power grid. DERM can also provide regulation services like frequency control and reserve services like spinning and non-spinning reserves.

Portland State University’s Power Lab has built a computer program to aggregate DER assets. This aggregation program, known as the Distributed Energy Resources Aggregation System (DERAS), was examined for utilization in an energy imbalance market (EIM). Simply put, DERAS is a DERM that manages DER. The current prototype of DERAS has been used to control 100 DER. This work examines the aggregation of 10,000 DER.

1.2 Objectives of Work

I aimed to test the capabilities of DERAS through simulation. As part of this work, I spent time working with DERAS, turning on its assets both real and emulated, using an operator
class that I developed in C++. My goal was to take that knowledge of DERAS, and simulate it at a much larger scale capable of delivering utility scale power.

My objective was to test an aggregated system of 10,000 assets because at such a scale these assets can be useful to a public utility. DERAS in its current state can only handle 100 distinct DER assets. I aimed to simulate these 10,000 assets to examine resource at this scale.

I aimed to simulate water heaters and battery-inverter systems to see what kind of energy and power resources they could offer in the western EIM market. I simulated the large asset aggregations and then simulated the effects of making and clearing an EIM decrement bid on these system.

1.3 Distributed Energy Resources (DER)

DER are non-utility scale energy assets[17]. They include residential-scale distributed generation, distributed energy storage, and demand-responsive loads. These assets are connected to a distribution network, rather than to a transmission system. Alone, these assets would have a negligible effect on the power grid, but when aggregated these assets could produce major effects to the electrical grid.

Distributed generation includes small-scale generators like a fuel cell, a solar panel array, or a combustion engine. Distributed generators add additional electricity to the power grid.

Demand-response loads are electrical loads agreeing to reduce or increase their electrical usage upon request. Demand-responsive loads include loads from large power customers
like industrial plants or commercial buildings or can also come from a residential customer, like one agreeing to turn off a water heater. Demand-responsive loads can reduce power during an emergency grid event. They can also increase power consumption when an excess of electricity is available on the grid.

Distributed energy storage includes electric vehicles and home battery energy storage units, like a Tesla Powerwall. A storage asset can act as either a generator or load, depending on whether it is charging or discharging.

1.4 Virtual Power Plant (VPP)

A VPP is an aggregation of DER assets that together create effects similar to that of a physical power plant connected to a transmission system work. VPPs can produce power like a traditional generator by combining distributed generation assets creating meaningful amounts of megawatts to effect the electrical grid [25]. VPPs can also reduce electrical loads to reduce the amount of power required to feed the electrical grid [22]. This reduced load is often called a “negawatt” and is legally equivalent to a megawatt for the purposes of electricity market settlements[11]. VPPs can be any mixture of these.

When a VPP injects power to the grid or reduces load, the VPP has effectively produced an increment in generation. The VPP action of increasing load has the effect of a generation decrement. This decrement action from the VPP can be an alternative to curtailing generation.
DERAS is a compilation of work by Robert Bass, Manasseh Obi, Tylor Slay, Annie Clarke, Leighton Clarke, and myself. DERAS is designed to aggregate a number of small distributed energy resources into a larger virtual power plant. In its current incarnation, DERAS has the capability to aggregate 100 individual DER units. DERAS aggregates DER assets through its VPP class (a C++ class).

Figure 1.1 shows DERAS as an aggregator. DERAS manages DER assets through the AllJoyn IoT framework. Assets like water heaters or batteries are monitored by agents.
These agents talk to DERAS through an agent hub. Digital twins (emulations of the physical assets) are created for each asset, and these predict the asset state between updates. The updater transmits the new physical states of the DER assets to the digital twins every hour or whenever a major change to the asset’s physical state occurs.

The resource estimator looks at the digital twin models of the assets to predict the available resources that the aggregated assets can provide to a utility or other third party user. This paper examines resource estimation. DERAS currently cannot aggregate more than 100 assets, but this work examines how 10,000 assets could be aggregated into an energy resource. This work does not use DERAS, but rather simulates DERAS at scales relevant to electric utilities.

1.6 Operator Class

I created an operator class in C++ to thread into the AllJoyn based aggregation network. This operator class contacts DER assets and gives grid commands to the assets. The operator class takes two parameters, an instantiated VPP class and a file pointer for a schedule. The instantiated DER asset can be any DER asset object in C++, like an electric water heater, an inverter-battery system, or an HVAC system. The schedule tells the assets when to take grid actions like importing and exporting power and when to idle. The operator class determines what action to take from the schedule, and then sends the command to the DER asset. The operator class is called in a thread, allowing other computer system processes to run simultaneously.
1.7 Balancing Area Authorities

The electrical grid functions on the concept that electricity produced somewhere is in balance with electricity consumed elsewhere. If a town requires 300 MW of electricity, a group of generators somewhere will be producing that 300 MW of electricity. Loads change frequently, and generation must balance against these new loads. Balancing areas (BAs) are sections of the power grid that are monitored for an equal amount of generation and load. A Balancing Area Authority (BAA) is an entity that manages the balancing of loads and generation in the balancing area.

BAs vary in size. At the large end, the California Independent System Operator (CAISO) manages the majority of land area in California, and the Bonneville Power Administration manages about half of the Pacific Northwest [5]. Smaller companies like Gridforce Energy Management, LLC in Oregon or Griffith Energy LLC in Arizona manage very small balancing areas for small utility districts.

Each of these Balancing Area Authorities (BAA) is responsible for ensuring that enough power is produced to cover the loads in their areas. Power can be produced within their balancing area, or they can import power from other balancing areas.

Unbalanced power has consequences to the electrical grid as a whole. When too little power is produced, frequency will sag. Frequency is affected by imbalances in load and generation. When generation is greater than load, the extra energy in the system is translated into an increase in system kinetic energy, which means the generators, primarily spinning machines, spin a little a faster. When load is greater than generation, the opposite occurs.
The lack of power from the generators results in a decrease in kinetic energy, which means the generators will spin a little slower. These large spinning generators create a large amount of inertia on the power grid, which means changes in frequency are damped and smoothed. When non-spinning assets like solar panels and batteries are producing a large portion of the generation, the system has much less inertia from spinning machines, so changes in load or generation lead to sudden changes in frequency.

The BAA is responsible for maintaining frequency in their BA. When large amounts of renewable generation is on the grid, the BAA keeps reserve generation units to provide frequency response in the event of a sudden loss of generation or sudden spike in load.

1.8 Energy and Imbalances Markets

Energy Imbalances Markets (EIM) balance the short term energy needs for Balancing Area Authorities by allowing short term market transfers between balancing areas. The Western EIM is the only instance of an EIM market in the United States. Its balancing area authorities currently comprise CAISO, PacificCorp, Portland General Electric, Puget Sound Energy, Powerex, Idaho Power, and Arizona Public Service [6]. The Western EIM uses both 15 minute markets (FMM) and five minute markets, also called real time markets (RTM). These markets are managed by CAISO.

CAISO requires each BAA to have enough energy within its BA to participate in the EIM [3]. It does this through hourly resource sufficiency tests [27]. If a BA does not have sufficient generation to cover its forecasted demand, then it cannot participate in the EIM.
This ensures that no BA in the EIM is being propped up by another BA.

Balancing areas submit bids to the EIM, and the bids are optimized for economic dispatch. The base schedules are determined based on the forecasted load demand, but when this load changes, the FMMs and RTMs cover the differences [4]. This may lead to generation being increased or decreased. CAISO manages the markets, first optimizing the FMMs and then optimizing the RTMs. Dispatch orders are then sent to the participating generators.

1.9 Literature Review

Many papers discuss DER asset aggregation and its relationship to the power grid. Many papers discussed aggregation of batteries or EWHs.

Laurent and Malhame built a computer aggregation model to simulate aggregations of water heaters [19]. They tested the influence of certain factors on power ratings including insulation, element power rating, and variance in water demand. Their work was targeted more for modeling the behaviors of water heater aggregations for forecasting loads, not controlling them. Later Laurent et al. created an optimization method for load management with control of electric water heaters [18]. They used a column generation method to optimize the water heaters for peak load reduction.

Fitzgerald et al. modeled 100,000 EWH assets to improve efficiency of wind generation [9]. They designed a control algorithm to improve the likelihood of water heaters being able to take on extra load during an expected peak in wind generation. Their simulation showed
that applying their algorithm could result in electricity reduction of 25% and cost reductions of 38% (from scheduling power consumption at better times).

Kepplinger et al. tested demand side management control optimization schemes for aggregated EWHs [15]. They tested three optimization methods: a price driven optimization, an energy driven optimization, and an optimization method based on time of day. They determined that a price optimized control scheme was most cost efficient with fewer service errors.

Kapsalis and Loukas created an optimal scheduling algorithm for EWHs[13], where they used an objective function that weighed interests in keeping electrical costs low while keeping customer discomfort minimal. Their algorithm minimized both costs and discomfort based on their relative weightings. Their algorithm worked on two levels, being able to adjust customer setpoints and also having access to a direct on/off switch for the electrical coil. In a different paper, Kapsalis et al., presented a heuristic scheduling algorithm to balance decisions between cost of power and comfort of consumer hot water consumption [14]. They simulated their algorithm in a real-time market and found it performed as well as a standard optimization model with considerably less computational overhead.

Roux et al. developed a peak demand manager algorithm that focused on water heater usage. Essentially, they had EWHs compete to use the electrical grid during peak hours, and only water heaters with the most need turned on during peak hours. They tested their control method against a traditional control method, and found their algorithm could shift controlled amounts of electricity from peak times, with minimal events of user discomfort.
Li et al. created an aggregator service for large multi-tenant buildings [20]. They built an aggregation of EWHs, solar panels, battery energy storage, and electric vehicles. Then they built a communication structure using the ZigBee protocol. They examined control algorithms for the DER assets that would allow them to minimize power costs through load shifting and energy trading of the DER units. The overall effects were lower tenant bills and shorter return on investment periods for the building.

Faika et al. proposed an Internet of Things architecture for aggregating multiple batteries over a wide area network [8]. They tested their proposal on a 5 battery test bed, and found the latencies in the network were small enough to allow for massive scaling over a cloud network.

Khalid and Savkin analyzed aggregations of batteries to smooth wind power production [16]. Essentially, they used it to compensate for the small dips and rises in wind power. They found a semi-distributed system of batteries both near the wind turbines and away from the wind turbines was most effective at smoothing wind power.

Additionally, many papers discussed battery aggregations using electric vehicles (EV). Wu et al. determined the operations scheduling of an aggregated set of EVs from cost minimization perspectives. Ortega et al. discussed the changes necessary to allow for aggregated EV dispatch in a day-ahead electricity market. Gonzalez and Andersson, Vagropoulos and Bakirtzis, and Sarker et al. investigated the optimal bidding strategies of EV aggregations [24, 28, 29].

Asimakopoulou and Hatzigiorgiou modeled the economics of DER aggregation through
a bilevel programming problem using the DER aggregator’s bid decisions as the upper level problem and the market clearing decisions as the lower level problem [1]. They confirmed that the addition of DER assets had a stabilizing effect on locational marginal prices (LMPs) of bulk electricity. They found that when LMPs were high, DER aggregators performed best when exporting energy to the power grid, and when LMPs were low, DER aggregators performed best when importing energy from the power grid.

Calvillo et al. built an optimization model for a variety of DER assets including photovoltaic systems, air-sourced heat pump systems, batteries, and demand response assets [2]. They optimized a system using price-maker economics, assuming that the DER aggregation was large enough to influence market clearing prices. They analyzed aggregations of residential houses running from 40 thousand to 8 million houses. Calvillo et al. found that aggregating these assets could be profitable for both aggregators and prosumers. They tested their system optimization with both price-taker and price-maker strategies, and concluded that using price-maker economics allows for significantly better optimization of the system.
2 Design Methodology

2.1 Modeled Aggregated Units

In order to test an aggregated number of DER units for bidding into the EIM, I constructed a model of ten thousand residential aggregated DER assets. For simplicity, I considered two types, electric water heaters and residential batteries. The aggregated assets can draw energy from the grid, inject energy into the grid, or defer drawing energy from the grid.

At any point in time, these assets have an individual power, an individual energy take, and an individual ramp rate. The model simulates expected patterns of usage for these assets and then looks at the assets in aggregate to estimate the ability of the aggregate to offer generator functions, specifically decrement and increment.

A decrement, in generator terms, means reducing the generator power output to the grid. A generator decrements because an excess of power is on the electrical grid. By drawing more power from the grid, aggregated assets effectively decrement by canceling out excess generation with additional load. The power grid retains the same amount of generation on the grid, but the load increases to balance out the excessive generation. Excessive generation frequently occurs when wind or solar produce more power than forecasted.

An increment, in generator terms, means increasing the generator’s power output to the grid. A generator increment occurs when the current level of generation is unable to
serve the required electrical load. The aggregated assets could increment in two ways. By
injecting power to the grid, the assets would be acting almost exactly like generators. The
aggregated assets also have the ability to defer drawing power until a later time. By choosing
to defer their loads, the immediate impact would be a reduced load, which would mimic an
increase in generation.

2.2 Electric Water Heaters

To simulate a large number of random water heaters, I used the DHW Event Schedule
Generator created by Hendron et al. [10]. This program creates randomized water usage
profiles for one, two, three, four and five bedroom households. It takes into account average
household data and then uses a randomized clustering algorithm to approximate random,
regular use patterns [23]. It then checks the randomized patterns against statistical metrics
to verify that the randomly-produced schedules make sense. When parts of the schedule do
not pass the statistical checks, the system is reiterated until a schedule is produced that will
pass all checks.

The tool is an MS Excel workbook, which utilizes Excel’s macro capability. Macros are
programmable routines written in Visual Basic. The tool itself produces one annual schedule
every time a button is pressed. A typical annual schedule has around 15,000 individual water
draws. I reprogrammed the macro to produce many water usage schedules by inserting
a for-loop. I saved three relevant pieces of information: time of event, duration, and hot
water volume, to three additional spread sheets. I then ran this macro to create up to 2000
schedules at a single button press. Using a variety of research computers available in the power lab, I produced over 10,000 random water draw schedules.

I created a set of 10,000 water heater schedules based on a guess as to the relative abundance. I used 2017 census data, showing the number of bedrooms in homes as a percentage across the United States [21]. Based on these numbers, I estimated the make up of 10,000 electric water heater units. I assumed that 0-bedroom homes would share similar water characteristics to that of a 1-bedroom home. I also assumed that four bedroom homes would be twice as common as five bedroom homes. I ignored 6+ bedroom homes, as they were not available through the DHW Event Schedule Generator[10]. I summarized my model of 10,000 electric water heater units in Table 2.1. Each of these water heaters were modeled from individual schedules created by the DHW Event Schedule Generator[10]. I also assumed a size of water heater based on the size of the household. All models were AO Smith brand water heaters.

Table 2.1: Percent Make Up of Water Heater Units

<table>
<thead>
<tr>
<th># of Bedrooms</th>
<th>Census Makeup[21]</th>
<th>Model Makeup</th>
<th># of Modeled Units</th>
<th>Gallons</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.4%</td>
<td>0%</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>10.9%</td>
<td>13.2%</td>
<td>1320</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>26.3%</td>
<td>26.3%</td>
<td>2630</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>39.6%</td>
<td>39.6%</td>
<td>3960</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>16.5%</td>
<td>16.5%</td>
<td>1650</td>
<td>80</td>
</tr>
<tr>
<td>5</td>
<td>4.4**%</td>
<td>4.4%</td>
<td>440</td>
<td>80</td>
</tr>
</tbody>
</table>

** The Census makeup showed 5+ households.
2.3 EWH simulation in MATLAB

Electric water heaters with tanks heat their water to a setpoint, typically around 120º. The water inside the tank is insulated, but small amounts of heat slowly dissipate from the water inside the tank. When hot water is removed from the tank, it is replaced by cold water. When the tank temperature drops below a deadband temperature, heater elements turn on to increase the water in the tank back to the set point. This represents normal operation for an uncontrolled water heater.

When DERAS controls water heaters, it operates them as a controlled load. DERAS aims to maximize the amount of energy that the water heaters can absorb at any time. To do this, DERAS keeps the water heaters near the bottom of their temperature range. To do this, DERAS constantly sends out “shed” commands to the water heaters. In “shed” mode, the water heaters will not come on unless they drop below a low temperature threshold. For instance, a water heater might have a setpoint of 120ºF, a deadband of 3ºF, and a lower shed threshold of an additional 3ºF. In normal uncontrolled operation, the water heater keeps its temperature between the setpoint and deadband, or between 117ºF and 120ºF. However, when the shed command is being called, the water heaters will continue to lose temperature until that lower threshold of 114ºF. Once they cross this threshold, the heaters ignore grid commands and heat until they reach the deadband temperature of 117ºF. Then the water heaters allow DERAS to control them once again. And again, DERAS tells them to shed. By running the water heaters between 114ºF and 117ºF, instead of between 117ºF and 120ºF, the water heaters will always be able to absorb energy to raise their temperatures by at least
3°F and many will be able to raise their temperatures by 4°, 5° or 6°F.

I used MATLAB to sort through the data, simulate normal operation, and simulate a bid. I chose to track the water heater states by two variables, the energy take of the EWH and the status of the EWH. The energy take is a measurement of how much energy the water heater would need to draw to reach its setpoint temperature.

\[ Q = m \cdot c \cdot (T_{\text{setpoint}} - T_{\text{tank}}) \]

Where \( m \) is the mass of the water in the tank and \( c \) is the specific heat of that water, 4180 kJ/(kg C°). With this measurement, a water heater at its setpoint has zero energy take. The lower the temperature of the tank, the higher the energy take.

### 2.3.1 Loading in the data

Using the modified NREL program, I could produce water usage data. I needed to turn this water usage data into electrical energy use data through simulation. The first step was loading all of these data into MATLAB.

I took the hot water use data from the spreadsheets, and converted them to comma separated value (csv) files. I loaded these files into MATLAB to produce simulations of the water heaters. Through an initialization procedure, I loaded in each of the csv files. I assigned each set of water heater data an identification number. I parsed through each time and date entry for draw events, and converted them into 5 minute time intervals. I used 5 minute time intervals, because this is how the Western EIM RTM operates, with a new
market clearing every 5 minutes. Each day has 288 individual 5 minute intervals, which means a 365 day year contains 105,120 intervals.

I organized the data into a 9-column array with rows for each draw event. The nine column array had columns for year, month, day, hr, minute, interval number, duration of draw in seconds, gallons per minute of draw, and the identification numbers that I assigned each unit.

2.3.2 Creating a look-up matrix for draw events

After loading the data, assigning identification numbers to the individual water heaters, and sorting the draw events into 5 minute intervals, I created a matrix for looking up specific draws from water heaters. The rows running downward represented individual EWHs with unique identification numbers. The columns represented 105,120 time intervals over the course of the year. I sorted the individual draw events by their unit ID and the time interval in which they occurred. In a draw event, hot water is drawn from the water heater and replaced by an equal amount of cold water coming from the tap. I calculated the amount of water drawn from the water heater during each 5 minute interval, and saved that amount to the lookup table.

2.3.3 Simulating EWH Behavior

I next built an array to store the energy takes of the individual EWHs at each time interval. I used the columns of this array to store identifying information about the water heaters, like ID number, number of bedrooms, capacity of tank in liters, energy take at the dead
band, energy take at the low temperature shed threshold, ambient heat losses, and power of the heating element. To start the simulation, I gave each water heater a randomized energy take between their deadband energy take and their low temperature shed threshold. I also randomized the state of the EWH, whether it was under DERAS control and shedding or whether it was under local control and heating back up to the deadband energy take.

Figure 2.1: State diagram for electric water heater model.
I used a variable to represent the states of the EWHs. When the EWH was responding to shed commands from DERAS, I gave it a state value of 1. When the EWH was heating back up to the deadband energy take because it had crossed its low temperature shed threshold, I gave it a state of 2. Additionally I used a state of 3 to represent when DERAS takes over control of the EWH, commanding it to heat to its setpoint. These states and transitions are shown in Figure 2.4.

Using nested loops, I calculated the energy take and state of the EWHs at the next time interval to compile a full year’s worth of data. I assumed that ambient losses would be the same at all times, as would the heating. Draws would be found in the lookup array, and then converted into energy take.

\[
Q_{k,\text{draw}} = m_{\text{draw}} c (T_{k,\text{tank}} - T_{\text{amb}})
\]

\[
T_{k,\text{tank}} = T_{\text{setpoint}} - \frac{Q_{k-1}}{m_{\text{tank}} c}
\]

To start the simulation, I gave each water heater a randomized energy take between their deadband and lower limit shed threshold. When left idle, the water heaters slowly lose heat to the ambient conditions. Once the water heaters drop below their lower shed threshold, the heating elements turn on and heat the water back to their deadband limits. At that point DERAS reasserts control, and the water heaters lose temperature until they reach the lower shed threshold again. This is what happens in the absence of draw events. In the presence of draw events, the units operate the same, except that at any moment, a hot water draw could lower the temperature of the tank below the lower shed threshold, at which point the unit stops shedding, turns its heating elements on, and heats until it reached the deadband.
<table>
<thead>
<tr>
<th>State 1, DERAS Controlled Shed</th>
<th>$Q_k = Q_{k-1} + Q_{amb} + Q_{k,draw}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if $Q_k &lt; Q_{LowLimit}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 1</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>Account for Heating</td>
</tr>
<tr>
<td></td>
<td>Set State = 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State 2, Local Controlled Heating</th>
<th>$Q_k = Q_{k-1} + Q_{amb} + Q_{k,draw} - Q_{heat}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $Q_k &gt; Q_{db}$</td>
<td>Set State = 2</td>
</tr>
<tr>
<td>else</td>
<td>Account for Shedding</td>
</tr>
<tr>
<td>Set State = 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State 3, DERAS Controlled Heating</th>
<th>$Q_k = Q_{k-1} + Q_{amb} + Q_{k,draw} - Q_{heat}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $Q_k &gt; Q_{setpoint}$</td>
<td>Set State = 3</td>
</tr>
<tr>
<td>else</td>
<td>Account for Shedding</td>
</tr>
<tr>
<td>Set State = 1</td>
<td></td>
</tr>
</tbody>
</table>

temperature allowing DERAS to reassert control. I also randomized the current state of the water heaters. Most of the water heaters would be in DERAS’s control, but some would have hit the lower shed threshold and would be heating up to their deadband temperature before DERAS took back control.

I simulated all the water heaters at each 5 minute interval. I stored two pieces of data at each interval: the energy take of the unit, and the state of the unit (DERAS controlled shed or unit controlled heating). Each new interval depended on the energy take and state of unit in the previous interval, as well as some constants specific to each EWH. I simulated the
entire data set over the course of the year thereby establishing a bases case of uncontrolled assets.

### 2.3.4 Simulating Bids

I created a function in MATLAB to determine the effects of an EIM bid. This kind of bid would be submitted to an energy market like the Western EIM. If the market decided to clear the bid, the aggregated units, controlled by DERAS, would be dispatched to cover the bid energy. A typical EIM bid would include power and ramp rate over a given period time. Because EWH have a near instantaneous ramp rate, I assumed that bid would include two elements, amount of power and duration. The amount of power would be in megawatts (MW) and the duration would be in 5 minute intervals.

To create this bid, I imagined an energy merchant would have a forecast of expected loads. EWHs in DERAS would follow a normal trend when DERAS was not bidding their decrementing capacity into an EIM market. They would act like uncontrolled water heaters, recharging when they dropped in temperature below their lower limit threshold. The energy merchant’s forecast would expect a certain amount of regular water heater use. The energy merchant would be cheated if DERAS sold that expected load back as a decrement bid. Therefore, I only wanted to sell power to the merchant that it would not expect to receive anyway.

To do this simulation, I reordered all of the water heaters from highest energy take to lowest energy take. Then I figured out how much a bid would require in energy over each 5 minute interval. I allocated the necessary number of units to meet the necessary energy, and
changed their status to 3, a DERAS controlled import. If a unit was in state 1, it could be shifted to state 3, but if a unit was in state 2, the aggregator would have no control over it, so it could not be allocated for the bid. Figure 2.2 shows this visually, the aggregator skips EWHs in state 2, and allocates only those in state 1.

Figure 2.2: The aggregator dispatches the units with the highest energy takes, skipping those in state 2.

Then I determined whether or not a water heater would naturally turn off in a future interval due to losses. If so, then I added that amount of energy to the required energy in the
later interval. This would ensure that my decrement bids did not artificially affect the energy
merchant’s forecast. I then resimulated the EWH actions at each 5 minute interval for the
remainder of the day and the following day, using the calculations described in Table 2.2.

2.4 Battery Inverter Systems

Battery inverter systems (BIS) have two capacities, a power capacity and an energy capacity.
A BIS has the ability to discharge power, which is its power capacity. This power capacity
is the maximum continuous power that the BIS can discharge. The BIS also has an energy
capacity. The energy capacity relates to how much energy the battery can store. Once a
battery discharges all of its energy, it can no longer discharge. The power capacity is given
in watts or volt-amps. The energy capacity is given in W hr.

To simulate a large number of BIS, I assumed a specific inverter type, the Outback
8048 inverter. This inverter provides a maximum continuous power of 8000 Volt-Amps. I
characterized the BIS by the number of hours of rated discharge. I assumed the batteries to
be randomly rated nominally at two, four, six, eight, or ten hours of full-power discharge
capacity. I then assumed that the batteries could have lost up to 20% of their nominal energy
capacity due to aging. I used another random number generator to assign the nominal energy
capacities with roughly 20% of the batteries being assigned to each of the nominal capacities.
I then used another random number generator to derate the batteries by up to 20% of their
nominal energy capacities. I calculated these battery features into an array for each of the
10,000 batteries.
2.5 Battery Simulation

Batteries do not have a normal usage schedule like electric water heaters. Instead, they could serve multiple purposes, and the EIM could be one of them. I assumed a second service would be frequency regulation. This would allow me to sufficiently randomize the behavior of the batteries before putting them into use.

I modeled the batteries to follow a frequency regulation profile when they were not being called on by the EIM. I use PJM’s RegA profile to simulate typical battery behavior, because it is published and easily accessed [26]. PJM has two frequency regulation bids, RegA and RegD. The RegD is more dynamic than RegA and also pays better. Many battery/inverter systems can meet the requirements for RegD regulation. However, RegD, typically has a balanced amount of up and down regulation occurring across an hour. This would tend not to randomize my battery profiles. Instead, I used the RegA regulation, which is less dynamic and therefore easier for an asset to follow. It regulates both up and down regulation, but it has a particular tendency to go down. I examined the 2018 data, and found over the course of 365 days, the the RegA signal was net negative 364 days and net-positive on only 1 day [12].

Following the PJM’s RegA signal, the batteries will from time to reach their limits of capacity, either upwards or downwards. I assumed that the batteries would have five states that would reflect their behavior toward the RegA signal or the EIM signal. These states are summarized in Table 2.3. I used a typical summer load profile, and assume that anytime 85% of peak load is happening, that power was expensive. In these cases a battery
prioritizes idling over charging. When power is inexpensive, the battery prioritizes idling over discharging.

Essentially, this BIS model simulates three grid services. By default, the BIS follow the PJM Regulation A signal. When the state of charge in the batteries becomes too high or too low, the batteries participate in economic arbitrage. Finally, the BIS can bid into the Western EIM, at which point EIM bids take precedence over other actions.

2.5.1 Loading in the data

I used MATLAB to generate the BIS usage data. I created a vector of 10,000 empty data spaces. I used MATLAB’s `rand()` function to generate a random number between zero and one. I used this number to assign nominal battery size to an individual data slot. I evenly weighted chances of having nominal energy capacities between 2 hours, 4 hours, 6 hours, 8 hours, and 10 hours. I then generated another random number, and used it to assign anywhere from 0% to 20% of aging loss to the nominal capacity. I cycled through the 10,000 data spaces to create 10,000 BIS energy capacities. Each of these BIS would have the same power capacity owing to their use of the same inverter.

I found the PJM RegA data from the PJM data website [12]. The data came in a zip file for each month. The data showed the RegA signal as a percentage of an asset’s bid power. The files were sorted in columns by days and in rows by 2 second intervals. I copied and pasted all the data from the individual monthly spreadsheets into a year long .csv file. I used 365 columns for each day of the year and 43200 rows for each 2 second interval in the day. I loaded these data into MATLAB using MATLAB’s `csvread()` function.
2.5.2 Simulating BIS behavior

I organized the BIS data into an array similar to my EWH array. The array shows the energy level and status of the batteries at every 5 minute interval. To get there though, I needed to simulate the battery usage based on the RegA signal data. These data occurred every 2 seconds. I built a 43200 x 6 array to simulate an individual battery over the course of a year. The battery’s state of charge determines its behavior, with five statuses defining these behavioral strategies. These statuses are summarized in Table 2.3.

Table 2.3: Statuses of EWH and BESS assets.

<table>
<thead>
<tr>
<th>No.</th>
<th>Asset</th>
<th>Status</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EWH</td>
<td>DERAS Shed</td>
<td>Respond to DERAS shed commands.</td>
</tr>
<tr>
<td>2</td>
<td>EWH</td>
<td>Local Heat</td>
<td>Import at deadband temperature.</td>
</tr>
<tr>
<td>3</td>
<td>EWH</td>
<td>DERAS Heat</td>
<td>Respond to DERAS import commands.</td>
</tr>
<tr>
<td>11</td>
<td>BESS</td>
<td>DERAS</td>
<td>Respond to DERAS commands.</td>
</tr>
<tr>
<td>12</td>
<td>BESS</td>
<td>Charge or DERAS</td>
<td>Charge firsts if power is cheap, else DERAS.</td>
</tr>
<tr>
<td>13</td>
<td>BESS</td>
<td>Charge</td>
<td>Charge if power is cheap, else idle.</td>
</tr>
<tr>
<td>14</td>
<td>BESS</td>
<td>Discharge or DERAS</td>
<td>Discharge if power is expensive, else DERAS.</td>
</tr>
<tr>
<td>15</td>
<td>BESS</td>
<td>Discharge or Idle</td>
<td>Discharge if power is expensive, else idle.</td>
</tr>
</tbody>
</table>

Batteries in the DERAS mode, control state 11, respond to DERAS commands exclusively, as long as they remain between their high energy and low energy thresholds. If a battery in DERAS mode drops below its low energy threshold, it turns to status 12, charge or DERAS. In this state it prefers to discharge if power is cheap, or otherwise it follows DERAS commands. If a battery in DERAS mode, status 11, rises above its high energy threshold, it turns to the discharge or DERAS mode, status 14. Here it prefers to discharge its energy if power is expensive, or otherwise it follows DERAS commands. These states
and transitions are shown in Figure 2.3.

I defined expensive power as power above 85% of an average summer load provided by the Florida PUC [7]. This meant power between 11:30 am and 9:30 pm would be considered expensive power. Batteries wait to discharge until power is expensive. At other times, power would be considered inexpensive. Batteries wait to charge until power is inexpensive.

I calculate the battery state of charge and new status every 2 seconds. I assume that the batteries instantly followed the RegA signal with no delay in response. I determined whether the battery would charge, discharge, or follow RegA based on the time of day and the battery’s status. The calculations are summarized in Table 2.5. For an individual BIS, I calculate these at every two seconds storing it in my 43200 x 6 matrix. Then I would take the state of charge and status at each EIM interval change, and store it in a matrix with rows of individual battery units and columns of states of charge and statuses at each EIM interval. This data could then be stacked on top of my BIS data in a single matrix.
Figure 2.3: State diagram for battery inverter system model.
I set variables based on state of charge to define when the BIS would move from one logic state to another logic state. These logic states, coupled with the time of day considerations, determine what actions the batteries take. Table 2.4 shows how batteries in one state will move to a new state based on state of charge.

Table 2.4: Statuses of EWH and BESS assets.

<table>
<thead>
<tr>
<th>Curr\New</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>N/A</td>
<td>10%</td>
<td>0%</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>12</td>
<td>80%</td>
<td>N/A</td>
<td>0%</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>13</td>
<td>80%</td>
<td>20%</td>
<td>N/A</td>
<td>90%</td>
<td>100%</td>
</tr>
<tr>
<td>14</td>
<td>85%</td>
<td>10%</td>
<td>0%</td>
<td>N/A</td>
<td>100%</td>
</tr>
<tr>
<td>15</td>
<td>85%</td>
<td>10%</td>
<td>0%</td>
<td>90%</td>
<td>N/A</td>
</tr>
</tbody>
</table>

2.5.3 Simulating a Battery Bid

To simulate a battery bid, I added another state to the battery logic, state 16. State 16 calls on the battery to charge until completely full. This accomplishes a decrement bid, by cancelling out generation with load.

I took the aggregation of 10,000 batteries, and sorted the BIS by energy capacity from highest to lowest. Then I assigned BIS to cover parts of a bid until it is completely covered by the assets in DERAS.

I used two models. During peak hours between 11:30 and 9:30, I used the batteries from lowest to highest energy states. During non-peak hours, I assumed the batteries that were in states 12 and 13 would be charging anyway, making themselves potentially part of a forecasted load. Therefore, I ignored batteries in states 12 and 13 during off-peak hours,
Table 2.5: BIS calculations at each state

<table>
<thead>
<tr>
<th>State 11, DERAS Controlled</th>
<th>$E_k = E_{k-1} + E_{RegA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if $E_k \leq E_{Low}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 12</td>
</tr>
<tr>
<td></td>
<td>else if $E_k \geq E_{High}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 14</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>Set State = 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State 12, Charge or DERAS</th>
<th>if Power inexpensive, $E_k = E_{k-1} + P_{Max}(2 \text{ sec})$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>else, $E_k = E_{k-1} + E_{RegA}$</td>
</tr>
<tr>
<td></td>
<td>if $E_k \leq E_{Min}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 13</td>
</tr>
<tr>
<td></td>
<td>else if $E_k \leq E_{Charge}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 12</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>Set State = 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State 13, Charge or Idle</th>
<th>if Power inexpensive, $E_k = E_{k-1} + P_{Max}(2 \text{ sec})$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>else, $E_k = E_{k-1}$</td>
</tr>
<tr>
<td></td>
<td>if $E_k \leq E_{MandatoryCharge}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 13</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>Set State = 12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State 14, Discharge or DERAS</th>
<th>if Power expensive, $E_k = E_{k-1} - P_{Max}(2 \text{ sec})$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>else, $E_k = E_{k-1} + E_{RegA}$</td>
</tr>
<tr>
<td></td>
<td>if $E_k \leq E_{Discharge}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 11</td>
</tr>
<tr>
<td></td>
<td>else if $E_k \leq E_{Max}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 14</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>Set State = 15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State 15, Discharge or Idle</th>
<th>if Power expensive, $E_k = E_{k-1} - P_{Max}(2 \text{ sec})$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>else, $E_k = E_{k-1}$</td>
</tr>
<tr>
<td></td>
<td>if $E_k \leq E_{MandatoryDischarge}$</td>
</tr>
<tr>
<td></td>
<td>Set State = 14</td>
</tr>
<tr>
<td></td>
<td>else</td>
</tr>
<tr>
<td></td>
<td>Set State = 15</td>
</tr>
</tbody>
</table>
and instead met my load by converting batteries in state 11 and also potentially state 14 as needed (state 14 can charge for a limited amount, but prefers discharge). When assigned units to a bid, I changed their status to 16.

State 16 is the aggregator controlled decrement state. In state 16 the batteries import power from the electric grid to effect a generation decrement. After each 5 minute interval, the state 16 BIS return to state 11, and they can immediately be reassigned to state 16 for the next interval. The EIM operates on 5 minute intervals whereas the RegA and arbitrage bids operate every 2 seconds. I simulated every 2 seconds, but I only stored data every 5 minutes for energy level and state.

Figure 2.4: Once assigned BISs go to state 16. After each interval they revert to state 11.

After completing the bid, I resimulated the battery aggregation over the remainder of the day and into the next. By doing this, I aimed to reach a steady-state condition where the battery aggregation returns to normal after the bid. These resimulations also updated every 2 seconds and only saved data every 5 minutes.
3 Results & Analysis

3.1 Results of simulating EWH assets without bids

I simulated the EWH water heaters without bids as a base case. I created a plotting script in MATLAB to plot the simulated characteristics of a 10,000 unit aggregation of EWHs. Figure 3.1 shows the results for a single day, June 6th, 2007. I marked the energy takes if all units were at their deadband or at their low limit temperature point in yellow and red, respectively. I marked the median of these energy takes in purple. As expected, the energy take fluctuates around the median energy take of the units.
Figure 3.1: The average Energy Take hovers around the median between the dead band and the low temperature cutoff.

The percent of units on at a given time is a key metric for measuring the capacity of the aggregator, and the primary reason these units turn on is because a draw occurred. Not every draw causes a water heater to turn on, and an idle water heater will turn on eventually due to thermal losses. However, we would expect to see a strong correlation between the percent of households experiencing draws and the percent of EWH assets heating. Figure 3.2 shows the percent of the units turned on coming into the 5 minute EIM market interval and the percent of units that experienced a draw during the current EIM interval. As expected, the number of units turned on slightly lags the draws.
3.2 Results of the EWH on Assets with bids

I simulated bids by specifying the EIM interval or intervals over which the bid was occurring and the number of megawatts I would be decrementing.

I first simulated an hour long bid from 02:00 to 03:00 of 5 MW. A bid like this could occur due to an unexpectedly gusty night, with additional wind generation that would force a decrement. The aggregator absorbs this energy into its aggregation of 10,000 EWHs. Figure 3.3 shows the overall energy take of the aggregator decreasing over this hour, and the
system returns to normal. The blue line shows the energy take when the bid is simulated and the dotted black line shows energy take with no bid simulated. A large linear dive in energy take occurs between 0200 and 0300. This dive in energy take comes from the units drawing 5 MW for one hour to meet the energy requirements of the bid.

Figure 3.3: To cover the bid, extra units turn on and a drop in energy take occurs.

Figure 3.4 shows the percent of water heaters that are turned on, blue representing the bid and dotted-black representing no bid. The number of water heaters turned on spikes from 02:00 to 03:00. After this period, the number of units heating due to a bid is less than the expected number of units heating when no bid is acted upon. At 09:40, the aggregator
energy take with bid and aggregator energy take differ by no more than 2%. By 10:00 the number of units heating with bid and without bid differ by no more than 2%. These can be thought of as settling times. The system takes 6 hours and 40 minutes to settle for energy take and 7 hours to settle for the number of units on.

Figure 3.4: The energy take dives to nearly 0, and it still does not cover the 6 MW, 1 hour bid

I then simulated a bid that the aggregator could not handle. I tested a 6 MW decrement bid from 0200 to 0300. Here the aggregator tries to assign units to get the necessary megawatts, but in the last two intervals fails to meet the bid. All units are turned on, and the aggregator still fails to meet the bid. Figure 3.5 shows the energy take falling to nearly 36
zero. Figure 3.6 shows the percentage of units heating. In the last two intervals, all units are
turned on, yet the aggregator still cannot meet its bid. Figure 3.7 shows the power output of
the 5 MW and 6 MW bids.

Figure 3.5: The energy take dives to nearly 0, and it still does not cover the 6 MW, 1 hour bid.
Figure 3.6: In the 2:50 and 2:55 intervals, the aggregator turns all units on, and still does not meet its bid.

Figure 3.8 shows how the aggregation handling a variety of two hour decrement bids ranging from 1 to 7 MW. Initially the aggregator provides power, and then the aggregation simply runs out of energy take. The bids do not go to zero, due to some amount of water draws and ambient losses over each interval.
Figure 3.7: In the 2:50 and 2:55 intervals, the 6 MW bid starts running out of energy, and fails to meet the power commitment of its bid.
I tested another bid at lower power, but over a much longer time period, in order to characterize the limitations of the aggregator. Figures 3.9 and 3.10 show the effects on the energy take. The bid ends at 22:00 on the first day and the energy take settles to within 2% of the non-bid simulation by 03:00 the next day for a 5 hour settling time. Figures 3.11 and 3.12
Figure 3.9: The energy take after a 1.5 MW, 18 hour bid, starting at 02:00 and ending at 20:00.
Figure 3.10: The day after energy take effects of the 1.5 MW, 18 hour bid.
Figure 3.11: Percent of units turned on for the day the of 1.5 MW, 18 hour bid from 02:00 to 20:00.
When I raised the bid to 1.6 MW over the same 18 hour period, the aggregator fails to meet the bid during certain intervals. Figures 3.13 and 3.14 show the energy takes and percent of units heating for the bid. Between 16:00 and 17:00, the aggregator fails to meet the bid during five of the five minute intervals.
Figure 3.13: The aggregator fails to meet the bid five times between 16:00-17:00.
Figure 3.14: The aggregator cannot supply the necessary power across seven of the intervals from 16:00 to 17:00.

I went further to plot a 2.0 MW bid over the same 18 hour period. The aggregator fails to meet the bid across 52 intervals of the 216 total intervals. Failures now occur from about 14:00 to 18:00, and another failure period opens up from around 4:30 to 6:00. Figure 3.15 shows the energy take falling to almost nothing during this periods. Figure 3.16 shows the units heating, with spikes to 100% during these periods.
Figure 3.15: The aggregator starts failing from 4:30 to 6:00.
Figure 3.16: The aggregator cannot supply all the energy to meet the bids, despite all units heating.

In Figure 3.17, I plotted the power absorbed by these units for bids from 02:00 to 20:00 for bids ranging from 1.5 MW to 2.0 MW. The bids first start failing in the 16:00 to 17:00 period, then intervals around 5:00 to 6:00 starting showing deficiencies. The higher bids lead to wider spans of failed intervals as well as greater deficiencies in actual power absorbed. The aggregator recovers after these intervals of failure. Upticks in hot water draws occurring around 6:00 and around 18:00 provide the extra energy take necessary to cover future intervals.
Figure 3.17: Increasing power leads to increasing number of intervals with insufficient power.

Both the 1.6 MW bid and the previous 6 MW bid represent bids failing due to lack of energy take in the units. The electric water heaters do not have the ability to store the extra energy without going past their setpoints.

The aggregator could also fail to meet a bid due to lack of power, rather than energy. In this case a large amount of power would be called on in a short interval. To test this I tried a case of large power draw over a single 5 minute interval. Figure 3.18 shows how a 43 MW bid over a single interval effects the energy take of the aggregator. After this bid, roughly 1.5 MWh of capacity still remain available. However, looking at Figure 3.19, we see that nearly all the units are called (99.3%). The aggregator actually meets this bid, but increasing
this to even 44 MW will result in failure. Here the aggregator should have roughly 5 MWh of energy take available. In theory it could be able to serve 60 MW over a 5 minute period. The aggregator is limited by power here. It does not have enough power in its aggregated electric heating coils to produce 60 MW.

Figure 3.18: 99.7% of all units turn on, barely covering this 43 MW bid.
To characterize the aggregator’s maximum energy, I wrote a MATLAB script simulating the aggregator’s response to a bid over a number of intervals at n-amount of megawatts. With this script I can pinpoint the maximum energy the aggregator can provide over a given interval.
The maximum decrement bid across a number of intervals is plotted in Figure 3.20. While the lines may look continuous, the points correspond to a specific number of 5 minute EIM intervals. Figure 3.21 shows a bid from 1 to 24 EIM intervals, which corresponds to 5 minutes to 2 hours worth of resource commitment. Table 3.1 further illustrates this point with specific power levels for each of the first 12 intervals. For the 10,000 EWH aggregation, only the first interval is power limited, while the remaining intervals are all energy limited.
I also tested my simulation on a smaller aggregation of 100 EWH. All these EWH were from 5 bedroom households with 80 gallon water tanks. I tested a 50 kW decrement bid to see the similarities to the larger aggregation. Figure 3.22 shows the effects of the 50 kW bid on energy take. Once again the black dotted line shows the non-bid case. The bids look comparable, however the settling is considerably slower. This is due to the water heaters being either 100% on or 100% off at any given time. With greater numbers, these effects even out, but with a smaller aggregation they are more noticeable.
Table 3.1: Maximum power the aggregator can provide across a number of intervals starting at 02:00 on June 6, 2007.

<table>
<thead>
<tr>
<th>Intervals</th>
<th>Max Power (MW)</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>43.2</td>
<td>Power Limited</td>
</tr>
<tr>
<td>2</td>
<td>29.3</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>3</td>
<td>20.1</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>4</td>
<td>15.3</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>5</td>
<td>12.3</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>6</td>
<td>10.3</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>7</td>
<td>8.9</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>8</td>
<td>7.8</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>9</td>
<td>6.9</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>10</td>
<td>6.3</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>11</td>
<td>5.7</td>
<td>Energy Limited</td>
</tr>
<tr>
<td>12</td>
<td>5.2</td>
<td>Energy Limited</td>
</tr>
</tbody>
</table>

Figure 3.22: Maximum biddable energy up to 2 hours.
Similarly Figure 3.23 shows the effects of the 50 kW bid on the amount of units heating. Again, slower settling is apparent.

Figure 3.23: 100 EWH aggregate energy take bidding 50 kW.

Figure 3.24 shows the effects that bids have on power absorbed by the aggregation. Here the granularity of water heaters is most visible. The algorithm commits water heater to cover the bid by turning at 100%. The units are bit until the power provided is at the bid level or higher, so with a small aggregation the power is noticeably greater than bid. Across 10,000 units this slight overage is unnoticeable on a graph, yet across only 100 units the overage can be seen on a graph. Comparing the failure of the small aggregation’s 100 kW overbid
to the 6 MW overbid bid from Figure 3.7, the 10,000 unit aggregation loses power more smoothly and linearly than the 100 unit aggregation.

![Power Absorption from Dec Bids on 100 EWH Aggregation](image)

Figure 3.24: 100 EWH aggregate power absorption over 1 hour.

### 3.3 Results of Battery-Inverter Systems without bids

I simulated 10,000 battery-inverter assets, assuming that they would follow PJM’s RegA signal. The first simulation established a base case. DERAS has batteries as part of its aggregation. These batteries are likely to be doing something else when not participating with the EIM, so I modelled them to regulate frequency based on the logic in Section 2. The main purpose of this simulation is to offer a baseline for BIS use to compare against when I
assign EIM bids later.

Figure 3.25 shows the state of charge of the batteries as a whole. The red line at the top shows the energy the batteries could store if all were fully charged. The 10,000 batteries are an amalgamation of 2, 4, 6, 8, and 10 hour systems, nominally rated with up to 20% of their capacity derated. Based on their state of charge, the batteries have different logic modes.

Figure 3.25: State of charge for the aggregate BIS, not bidding into EIM.

Figure 3.26 shows the logic states of the batteries. At the start of the day about half of the batteries are following the RegA signal, and the other half of batteries are in the prefer to
charge or else follow RegA state. As the batteries start depleting over the peak hours, many of them move into the prefer to charge or else idle state.

![Logic States of 10,000 Simulated Batteries on June, 6th, 2007](image)

Figure 3.26: Logic states for each of the BIS not bidding into the EIM.

The logic states in Figure 3.26 manifest themselves as four actions that batteries can take: follow RegA, charge, discharge, and idle. Figure 3.27 shows the actions that the batteries are taking. In the early hours when power is cheap, the batteries choose to charge. Once sufficiently charge they again start following RegA as seen from 00:00 to 10:00. Once power becomes expensive at 11:30, the batteries all switch to follow RegA, and as time passes and more batteries deplete, they go idle rather than discharge to the grid. Once 22:30 hits, and
power is cheap again, the batteries switch to charging.

Figure 3.27: Actions the BIS take based on their logic states.

Figure 3.28 shows PJM’s RegA signal that these batteries are trying to follow. The RegA signal calls for the battery-inverter systems to draw anywhere from up to 100% of the capacity as import and export power. RegA leans far more export than import, and this day is no exception.
3.4 Results of Battery-Inverter Systems with EIM Bids

The batteries have significantly more energy than the EWHs. In this model, a 2 hr battery (16 kWh) has enough storage availability to continuously draw 8000 Watts over 2 hours, while an 80 gallon EWH at 6 degrees below setpoint can only draw 4500 Watts over 15 minutes (1.5 kWh). I tested a 20 MW bid from 02:00 to 04:00 (40 MWh of energy). Figure 3.29 shows the energy effects of the bid. Note that here I am measuring state of charge energy, so as the battery draws power the energy should increase. From 02:00 to 04:00 additional
batteries charge to cover the decrement bid. The energy stored in the batteries increases over this period. This would be expected from a decrement bid. The increased load that the batteries draw will cancel generation.

Figure 3.29: Aggregate state of charge profile after a 20 MW bid from 02:00 to 04:00.

Figure 3.30 shows the logic states of the BIS. From 02:00 to 04:00, anywhere from 25 to 50% of the BIS charge to cover the deck the decrement bid. Figure 3.31 shows the resulting actions.
Figure 3.30: Logic states of batteries after a 20 MW bid from 02:00 to 04:00.
Looking closely at Figure 3.29, from 9:30 to 11:30, the bid BIS aggregation loses charge while the base case BIS aggregation gain charge. This can be understood by looking at the logic states of the batteries in the decrement bid case shown in Figure 3.30. Many of the batteries have higher energy states, which leads them to not charge before during the 09:30 to 11:30, as shown in Figure 3.31. These graphs can be contrasted against the logic and action states of the base case, Figure 3.26 and Figure 3.27. Roughly 10% of units turn on in the bid case, while roughly 30% of the units charge in the base case. This is due to the model I used as I set logic states based on state of charge (see Section 2.5).
I also tested a 50 MW overbid for the period from 02:00 to 04:00 (100 MWh). Figure 3.32 shows the state of charge of the battery aggregation. Over the first hour, DERAS supplies the 50 MW decrement bid. This can be seen in the logic states of the aggregation in Figure 3.33. Between 03:00 and 4:00, all batteries switch to charging mode, but the batteries cannot cover the bid. Figure 3.34 shows the BIS actions for the aggregation.
Figure 3.33: Logic states of batteries after a 50 MW bid from 02:00 to 04:00.
Figure 3.34: BIS actions after a 50 MW bid from 02:00 to 04:00.

Figure 3.35 shows the power charged by the BIS aggregation for bids ranging from 20 to 80 MW. As batteries become fully charged, DERAS loses its ability to offer power. Incidentally, the 80 MW is the absolute maximum power of the battery aggregation based on 10,000 batteries each providing a maximum of 8,000 watts.
Figure 3.35: Power drawn by the aggregator in response to various decrement bids from 02:00 to 04:00.

I also performed a simulation using a much longer 18 hour time frame for a bid. In this case, I bid 15 MW across 216 EIM 5 minute intervals (270 MWh). Figure 3.36 shows the energy across the start to approach maximum charge. Figure 3.37 and Figure 3.38 show the battery logic states and actions respectively. At the final intervals, almost 100% of the battery aggregation is charging.
Figure 3.36: State of charge profile after a 15 MW bid from 02:00 to 20:00.
Figure 3.37: Logic states of batteries after a 15 MW bid from 02:00 to 20:00.
When I simulate a 20 MW overbid across 18 hours (360 MWh), DERAS maxes out both its energy availability and power. Figure 3.39 shows the battery states of charge for the aggregation. Between the hours of 17:00 and 20:00, the battery aggregation reaches its maximum charge.
Figure 3.39: State of charge profile after a 20 MW bid from 02:00 to 20:00. The aggregation becomes 100% fully charged.
Figure 3.40: Logic states of batteries after a 20 MW bid from 02:00 to 20:00.

The logic states and battery actions are shown on figure 3.40 and figure 3.41 respectively. These show that the 100% of the units are called on starting at 13:00. This again demonstrates a power limited bid, in that it fails due to lack of power before it fails due to lack of energy.
The power absorption by DERAS for 18 hour bids ranging from 15 to 30 MW is shown in Figure 3.42. For higher bids, the power absorption loses momentum and quickly crashes to zero.
Additionally I simulated the maximum decrement bids available across for a given number of 5 minute intervals. Figure 3.43 shows the maximum available bids for anywhere from 1 interval (5 minutes) to 36 intervals (3 hours). For short stints, the battery aggregation can supply 80 MW continuously, but as the number of intervals increases, the 2 hour batteries (16 kWhs) and the 4 hour batteries (24 kWhs) start filling to capacity. Once a battery is filled to capacity, it can no longer charge so the power from its inverter becomes unavailable.
Figure 3.43: Power output from aggregation drops as batteries become fully charged.
4 Discussion

The results of these simulated bids show how aggregations of EWHs and aggregations of BISs can bid into an EIM through decrement bids.

4.1 EWH Aggregations

EWHs have a primary purpose to provide hot water to their owners. An EWH aggregation could take advantage of a grid use, but this would be secondary to their providing hot water. The model I used assumed that the EWH would be available for hot water at all times and that any grid functionality came secondary at lower priority.

The simulations of the 10,000 EWH aggregation demonstrated the ability of an aggregator to follow EIM decrement bids. The model used 39.5% of the households with 40 gallon units, 39.6% with 50 gallon units, and 20.9% with 80 gallon units. The model assumes that an aggregator would find approximately this makeup of CTA-2045 assets among its EWHs. The model also assumed assets that had turned on before the bid or would turn on due to ambient heat losses could not be used to supply power as part of the aggregation, as these assets could be considered part of the load forecast. The model additionally assumed that the power availability of the units was 4500 Watts, which is not an unreasonable assumption as most water heaters have power ratings around this value.
With these assumptions in mind, a 10,000 unit EWH aggregation would have around 5 or 6 MWh of energy available at any given time. The available power would be related to the units that were not heating or forecasted to be heating. The maximum available power would be the sum of all the power in each unit, or 45 MW. However, since some of these would already be heating, this could be down rated as much of 20%, or around 36 MW.

The EWH aggregations could provide high power bids over short time periods or lower power bids across longer intervals. I found bids of 43 MW would be successful over a 5 minute period. I found that bids of 5 MW would be successful over a period of an hour. I found that 1.5 MW bids would be successful over periods of 18 hours. In each of these cases, when bids exceeded the available energy or power the bids would fail to provide the correct amounts of megawatts over the given bid periods. I found that bids were limited by energy across all numbers of five minute intervals except the first five minute interval. Across the first 5 minute interval energy levels with the ability to provide roughly 60 MW across 5 minutes was limited by a maximum power of 45 MW by the electric coils. After that first interval, the EWH aggregation was limited by the energy take of the system. These details are summarized in Table 3.1.

When overbidding occurred, the aggregation was unable to deliver the necessary power over the required time period. This led to the power drawn by the aggregation diminishing in the later intervals. For short, high power, bids, the system did not recover, and the power diminished to about 0.5 MW, or the continuous power coming from the hot water draws as in Figure 3.8. However, when fewer megawatts were bid across longer intervals, the hot
water draws could allow the system to recover back to full power, except for a few bad 5 minute intervals as shown in Figure 3.17.

While overbidding would result in insufficient power over certain intervals, the DERAS dispatch algorithm could be questioned. I dispatched units to cover bids until the bids could no longer be covered and the system started to fail. An alternative way would be to recognize first that DERAS could not cover the bid, and then even out deficiencies in the bid over the entire time interval. For short bids, like a 1 hour, 6 MW bid, the solution is to simply offer 5.2 MW over the entire hour. This can be solved simply using the maximum power algorithm as shown in Table 3.1. However for longer intervals like the 18 hour, 2 MW bid, a dispatch that minimizes disturbances might be better than a dispatch that tries to assign units to cover every single interval until failure.

After nearly depleting themselves, EWHs took roughly 7 hours to recover back to their non-bid states. Here there are two settling times, the energy settling time for when energy takes recover to within 2% of the non-bid levels and the power settling time for when the number of units heating returns to within 2% of the non-bid levels. After these settling times, the aggregate system effectively returns to its normal energy state as if no bid had occurred.

4.2 BIS Aggregations

BIS for the most part have a primary purpose of making transactions on the electric grid. Certain customers may have a BIS to provide backup power, but failures tend to be rare on the modern electric grid. For this reason, grid transactions can be viewed as the primary
purpose of the BIS aggregation.

Since the batteries can be providing other services, the question then becomes what those services should be. The action I chose for the battery aggregation was to follow a frequency regulation signal first, and secondarily play economic arbitrage when needing to discharge or recharge. The important take away of this model is, the BIS are performing some task, and I am going to take control of part of the BIS aggregation and have it perform another task. In this case, the task I am having them perform is a decrement to an EIM.

The BIS aggregation was able to make a 40 MW bid over the course of two hours without losing power across any of its 5 minute intervals. The BIS aggregation was also able to supply 15 MW of power across 18 hours, while still maintaining sufficient power draw. As bids increased, the power absorption from the aggregation of batteries would fail. The aggregation would not be able to absorb enough power across the later intervals.

The aggregate response to an overbid might be improved by the dispatch algorithm. The current dispatch algorithm dispatches enough BIS units to cover each bid interval until the system runs out of units. A smarter algorithm might be designed to provide a smaller insufficient amount of energy across each of the bid intervals to minimize the shock to the the system once the aggregator could no longer absorb power. With BIS, where the primary grid function is serving the grid, a better strategy might be to bid the maximum amount of power based on the maximum bid algorithm.

The BIS aggregated bids were entirely power limited. Over a 24 hour period, bids failed due to lack of power rather than lack of energy. This is perhaps due to the DERAS dispatch
algorithm, which finds units to cover the bid, and then operates them at 100% of full power. As the batteries start charging, they reach their maximum state of charge. At this point, the battery can no longer charge, so the overall power available to the system diminishes. An algorithm that utilized batteries at less than full power could more efficiently access its energy take. The highest available maximum bid would theoretically fail by both power and energy take at the same time.

The battery model used a very generic control logic. Batteries took grid actions based on a logic state that was based upon their state of charge. These led to results where a decrement bid increased the short term charge of the system, while decreasing the charge later on due to changes in logic states from the base system. Figure 3.29 demonstrates this. The battery aggregation charges between 02:00 and 04:00, but then around 09:30 to 11:30, the battery aggregations follow the RegA signal because they have a relatively high charge. The battery aggregation that did not charge between 02:00 and 04:00 became so depleted, that many of its BIS charged between 09:30 and 11:30 rather than follow the RegA signal. This resulted in the system that fulfilled the decrement bid by charging from 02:00 to 04:00, having a lower state of charge around 20:00 than the system that did not make a decrement bid.

This is a result of the BIS logic model. On the one hand, the aggregator could judge that income it makes from the decrement bid, and the subsequent regulation following, justifies this decision over the no-bid situation. Alternatively, an aggregator could choose to control the system differently and simply tell the BIS aggregation to charge between 09:30 and
11:30.

The decisions on how to model the control and dispatch algorithms for the BIS aggregations are more flexible than the EWHs, because the BIS aggregation is primarily transaction with the electric grid. This leaves more flexibility in deciding how to decide when to charge or discharge, and when to make EIM bids.
This research shows the potential for aggregated EWH assets and BIS assets for providing energy to an EIM. A 10,000 unit aggregation EWH was able to provide roughly 43 MW of instantaneous power over a single 5 minute interval and roughly 5 to 6 MW over an hour, depending on time of day. This research shows that aggregations of EWHs can be used to provide significant decrementing power to an EIM market while still maintaining their primary function of providing hot water.

This research also examined the ability of aggregated BISs to offer decrementing bids to an EIM while also providing frequency regulation and performing economic arbitrage. To this end, the simulated BIS aggregations provided significant decrementing support, as much as 40 MW in a 2 hour period and 15 MW in an 18 hour period.

Further research may focus on a smarter dispatch algorithm. The water heater algorithm of picking the assets with highest energy takes and forcing them to heat is a simple straight forward model, which could be adapted for short intervals by simply reducing power in the case of an overbid. However, the longer running bids might be better dispatched to increase the number of insufficient intervals while reducing the insufficiency across these intervals. For batteries, figuring out the use case is most important. Batteries following an economic arbitrage model will necessarily want to follow different strategies than those following a
regulation signal with EIM as a secondary purpose. Dispatch algorithms can be tailored to the purpose of the BIS aggregation.

Further research could also focus on better modeling data. As DERAS grows larger and can control tens of thousands of DER assets, the ability to track real assets and build digital twins of these assets for forecasting decisions would allow for a model that can predict power availability of real assets in real time. This is the goal of a fully mature DER aggregation.


[8] Tasnimun Faika, Taesic Kim, and Maleq Khan. An Internet of Things (IoT)-based network for dispersed and decentralized wireless battery management systems. 2018


Appendix A: MATLAB Scripts
% Thesis %
% Data Input Portion %
% Duration and time are easy. They come in as double arrays. Time has to % enter as a table, and then be converted to an array to keep time date % formatting.
tic
% Five Bedrooms

dur5 = csvread('Dur5Bed440.csv');     % Duration for SKHomes x 1000
hot5 = csvread('Hot5Bed440.csv');     % Hot Draws for SKHomes x 1000
time5 = readtable('Time5Bed440.csv'); % Time/Dates for SKHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time5 = table2array(time5);
full5 = tch1(time5,dur5,hot5);
%toc

% Four Bedrooms Part 1
%tic
dur41 = csvread('Dur4Bed1000.csv');  % Duration for SKHomes x 1000
hot41 = csvread('Hot4Bed1000.csv');  % Hot Draws for SKHomes x 1000
time41 = readtable('Time4Bed1000.csv'); % Time/Dates for SKHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time41 = table2array(time41);
full41 = tch1(time41,dur41,hot41);
%toc

% Four Bedrooms Part 2
%tic
dur42 = csvread('Dur4Bed650.csv');  % Duration for SKHomes x 1000
hot42 = csvread('Hot4Bed650.csv');  % Hot Draws for SKHomes x 1000
time42 = readtable('Time4Bed650.csv'); % Time/Dates for SKHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time42 = table2array(time42);
full42 = tch1(time42,dur42,hot42);
%toc

% Three Bedrooms Part 1
%tic
dur31 = csvread('Dur3Bed1800.csv');  % Duration for SKHomes x 1000
hot31 = csvread('Hot3Bed1800.csv');  % Hot Draws for SKHomes x 1000
time31 = readtable('Time3Bed1800.csv'); % Time/Dates for SKHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time31 = table2array(time31);
full31 = tch1(time31,dur31,hot31);
%toc

% Three Bedrooms Part 2
%tic
dur32 = csvread('Dur3Bed2000.csv');  % Duration for 5BRHomes x 1000
hot32 = csvread('Hot3Bed2000.csv');  % Hot Draws for 5BRHomes x 1000
time32 = readtable('Time3Bed2000.csv');  % Time/Dates for 5BRHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time32 = table2array(time32);
full32 = tchi(time32, dur32, hot32);
%toc

% Three Bedrooms Part 3
%tic
dur33 = csvread('Dur3Bed60.csv');  % Duration for 5BRHomes x 1000
hot33 = csvread('Hot3Bed60.csv');  % Hot Draws for 5BRHomes x 1000
time33 = readtable('Time3Bed60.csv');  % Time/Dates for 5BRHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time33 = table2array(time33);
full33 = tchi(time33, dur33, hot33);
%toc

% Two Bedrooms Part 1
%tic
dur21 = csvread('Dur2Bed630.csv');  % Duration for 5BRHomes x 1000
hot21 = csvread('Hot2Bed630.csv');  % Hot Draws for 5BRHomes x 1000
time21 = readtable('Time2Bed630.csv');  % Time/Dates for 5BRHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time21 = table2array(time21);
full21 = tchi(time21, dur21, hot21);
%toc

% Two Bedrooms Part 2
%tic
dur22 = csvread('Dur2Bed2000.csv');  % Duration for 5BRHomes x 1000
hot22 = csvread('Hot2Bed2000.csv');  % Hot Draws for 5BRHomes x 1000
time22 = readtable('Time2Bed2000.csv');  % Time/Dates for 5BRHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time22 = table2array(time22);
full22 = tchi(time22, dur22, hot22);
%toc

% One Bedrooms Part 1
%tic
dur1 = csvread('Dur1Bed1320.csv');  % Duration for 5BRHomes x 1000
hot1 = csvread('Hot1Bed1320.csv');  % Hot Draws for 5BRHomes x 1000
time1 = readtable('Time1Bed1320.csv');  % Time/Dates for 5BRHomes x 1000
% Takes ~7 minutes for 20E5x10E4
time1 = table2array(time1);
full1 = tchi(time1, dur1, hot1);
clear time1 hot1 duri time22 hot22 dur22 time21 hot21 dur21 time32 hot32
clear dur32 time31 hot31 dur31 time41 hot41 dur41 time5 hot5 dur5
clear time42 hot42 dur42 time33 hot33 dur33

% Concatenating all datasets.
raw = [full5; full41; full42; full31; full32; full33; full21; full22; full1];

clear full5 full41 full42 full31 full32 full33 full21 full22 full1
toc
tic

% Assigning ID#'s
SIZE = size(raw);
k=1;
raw(1,9) = k;
for n=2:SIZE(1)
    if ( raw(n,2) < raw(n-1,2) ) % If the month has cycled from Dec to Jan.
        k =k + 1;
    end
    raw(n,9) = k;
end
toc
tic
[etakeInit, draws] = DataInit2(raw, k);
toc
tic
clear raw
tic

% Calculating Energy Takes for Each 5 minute period by simulation.
ETAKE = SimETAKE2(etakeInit, draws);
toc
clear etakeInit
function final=tdhi(t,d,h)
% Time Duration Hot function

% Inputs t, d, and h
% All input data is an NxM matrix where M is the number of datasets, and N are the the individual points in the the dataset.
% t = time matrix, a matrix of datetime values which could be UTC'd.
% d = duration matrix, a matrix of duration of draw values as doubles.
% h = hot draw matrix, a matrix of hot draw volume as doubles.

% Purpose
% Converting Raw Data to Useful Arrays.

% Method
% I want to turn the raw data in the squarish looking arrays into data in long vertical arrays. Hot and Duration are easy, and can be done in a loop in about 15 seconds. I convert the timedate information in the time arrays into timedate vectors, having the form [yr,mo,day,hr,min,sec]. From here, I save those into a vertical array of data points with six columns for the date vector data. Finally I combine all the data sets into an (N*M) x 9 matrix.

% Output (N*M) x 9 matrix, with rows representing data point columns stored in the form [yr,mo,day,hr,min,sec,dur,hotdraw,id].

tv = 0; % Vertical reorganization of t matrix with final size (N*M)x1.
dv = 0; % Vertical reorganization of d matrix with final size (N*M)x1.
SIZE = size(d); % This is for calculating my end points for the nested 2-dimensional for loop.

% This for loop converts the hot and duration data in about 15 seconds.
k=1; % This variable counts through the vertical matrix that I'm saving to.
for n=1:SIZE(2)
    for n=i:SIZE(1)
        tv(k,1) = h(n,m);
        dv(k,1) = d(n,m);
        k=k+1;
    end
end

% The time array should have the same vertical dimension as the duration hot arrays. I use this dimension to precision the time matrix based for faster processing.
SIZE = size(dv);
tv = zeros(SIZE(1),6); % Presized, vertical reorganization of t matrix.
dv = zeros(SIZE(1),1); % Presized column for ID's to be filled in later.
SIZE = size(t); % For calculating for-loop limits.

% This for loop converts the datetime data and takes about 45 minutes for
% a 20000x1000 input matrix of datetime entries.

k=1; % This variable counts through the vertical matrix I'm saving to.
for n=1:SIZE(l)
    for m=1:SIZE(l)
        temp = datevec(t(n,m)); % Converts datetime to datetime vector.
        tv(k,1) = temp(1); % Stores the year from datetime vector.
        tv(k,2) = temp(2); % Stores the month.
        tv(k,3) = temp(3); % Stores the day.
        tv(k,4) = temp(4); % Stores the hour.
        tv(k,5) = temp(5); % Stores the minute.
        tv(k,6) = temp(6); % Stores the second.
        k=k+1;
    end
end

% This concatenates the vertical time, duration and hotdraw arrays into
% an (N x 9) matrix, [yr,mo,day,hr,min,sec,dur,hotdraw, id#].
raw = [tv, dv, hv, idv];

% Counting the NaN rows.
SIZE = size(raw);
counter = 0;
for n=1:SIZE(1)
    if(isnan(raw(n,1)))
        counter = counter + 1;
    end
end

SIZE = SIZE(1) - counter;
final = zeros(SIZE, 9);

% Deleting the NaN rows.
SIZE = size(raw);
k=1;
for n=1:SIZE(1)
    if(~isnan(raw(n,1)))
        final(k,1) = raw(n,1);
        final(k,2) = raw(n,2);
        final(k,3) = raw(n,3);
        final(k,4) = raw(n,4);
        final(k,5) = raw(n,5);
        final(k,6) = raw(n,6);
        final(k,7) = raw(n,7);
        final(k,8) = raw(n,8);
        final(k,9) = raw(n,9);
        % final(k,10) = raw(n,10);
        % final(k,11) = raw(n,11);
        % final(k,12) = raw(n,12);
        % final(k,13) = raw(n,13);
\[ k = k - 1; \]
\end{verbatim}

end

end

% The second column is not used, because of where I got my data rounds to
% the nearest minute, so I'm assigning interval numbers to the data in
% column 6.

\begin{verbatim}
SIZE = size(final);
for n=1:SIZE(1)
      interval = 1;
      if final(n,2) == 2
          interval = interval + 31*288;
      elseif final(n,2) == 3
          interval = interval + (31+28)*288;
      elseif final(n,2) == 4
          interval = interval + (31+28+31)*288;
      elseif final(n,2) == 5
          interval = interval + (31+28+31+30)*288;
      elseif final(n,2) == 6
          interval = interval + (31+28+31+30+31)*288;
      elseif final(n,2) == 7
          interval = interval + (31+28+31+30+31+31)*288;
      elseif final(n,2) == 8
          interval = interval + (31+28+31+30+31+30+31)*288;
      elseif final(n,2) == 9
          interval = interval + (31+28+31+30+31+30+31+31)*288;
      elseif final(n,2) == 10
          interval = interval + (31+28+31+30+31+30+31+31+31)*288;
      elseif final(n,2) == 11
          interval = interval + (31+28+31+30+31+30+31+31+31+31)*288;
      elseif final(n,2) == 12
          interval = interval + (31+28+31+30+31+30+31+31+31+31+31)*288;
end

% Now adjust for full days (days start at 1).
interval = interval + (final(n,3) - 1)*288;
% Now adjust for full hours (hours start at 0). 12 intervals in an hr.
interval = interval + final(n,4)*12;
% Now adjust by minutes
counter = 0;
minutes = final(n,5);
while(minutes >= 5)
    minutes = minutes - 5;
    counter = counter + 1;
end
interval = interval + counter;
% Now find fractional adjustment.
if(minutes == 1)
    interval = interval + 1/5;
\end{verbatim}
elseif(minutes == 2)
    interval = interval + 2/5;
elseif(minutes == 3)
    interval = interval + 3/5;
elseif(minutes == 4)
    interval = interval + 4/5;
end
final(n,6) = interval;
end

% The "final" array is an (N x 9) matrix with the following columns.
% [year, month, day, hour, minute, interval#, duration, hotdraw, id#].

end
% Data Initialize
function [etakeInit, draws] = DataInit2(full,k)

% Purpose: Pulls the important data related to the specific unit ID for
% calculating energy take. Calculates an initial energy take.

% Inputs
% full: a data array with every piece of information, ordered by unit# and
% then chronologically by hot draw times.
% k: the number of units.

% Outputs
% etakeInit: A matrix k x 9 matrix with initial data for each EWH. Data is
% stored as [ID#, #Bedrooms, EWH Liters, OFF E-Take, ON E-Take, Losses,
% Coil Power, Randomized Initial E-Take, Randomized Initial Status].
% draws: An k x (288*365) matrix. Each event is mapped to a specific 5 min
% periods throughout the year.

% Making an ID#, Bedrooms, Size, Initial Energy Take, and Initial Status
etakeInit = zeros(k,9);
% Assumptions
deadband = 3; % Degrees F that the water heater can fall before turning on.
sendband = 6; % Degrees F that the water heater can fall on a shed cmd.
setpoint = 120; % Degrees F
losses = 36.3; % Watts if power and Watt-hrs per hour if energy.
% This was based on a 50 gallon tank, so a correction
% factor based on surface area is employed.
coil = 4500; % Watts if power and Watt-hrs per hour if energy.
% Time spent's losing temperature.

% Surface area coordination (inches, but they divide out for calculations)
% This for correcting the losses at different surface areas.
% SA = [Gallons, Height(in), Diameter(in), Surface Area (To be calculated)]
% SIZE = [50, 61.5, 26, 0; 50, 60.5, 22, 0; 49, 60.25, 20, 0; 30, 46.5, 19, 0];
for n=1:SIZE(1)
    SA(n,4) = 2*pi() * SA(n,2) * SA(n,3) + 2*pi() * SA(n,3)^2;
end

% Creating Interval Table of Energy Take increases due to hot water draws.
% This is an array with columns corresponding to time intervals
% rows corresponding to ID#'s (variable k). This will be an N x M matrix with
% N = ID#'s or variable k, and M = # or intervals or 288 * 365.
draws = zeros(k,(288*365));
SIZE = size(full);
for n=1:SIZE(1)
    if(full(n,3) > 0)
ID = cast(full(n, 9), 'uint32');
interval = cast(floor(full(n, 6)), 'uint32');
remainder = full(n, 6) - interval; % Actual minutes of draw in current interval.
offset = 5 - (remainder * 5); % Minutes over which event occurs.
duration = full(n, 7)/60; % Duration of draw in minutes.
draw = full(n, 8) * 3.78541; % GPM of hot draw converted to LPM.
if (offset >=(duration)) % Draw occurs during the current interval
    D = (duration * draw); % Liters of draw, but also kg since sg=1
    draws(ID, interval) = draws(ID, interval) + D; % Draw at interval
    % equals the current number there plus % any additional amount D that I'm % adding due to this current draw.
else % The draw bleeds into other intervals intervals.
    D = (offset * draw); % This time I use the remainder of the % time in this interval.
    draws(ID, interval) = draws(ID, interval) + D; % Add this to the % current interval.
    leftover = duration - offset; % Time bleeding into add'l % intervals.
    h = 1; % Interval counter.

    % Count down the leftover time duration in 5 minute chunks.
    while (leftover > 0)
        if((interval + h) > (200*365)) % Stop if out of bounds.
            leftover = 0;
            break
        end
        if(leftover <=5)
            break
        end

        leftover = leftover - 5;
        D = 5*draw;
        draws(ID, interval+h) = draws(ID, interval+h) + D;
        h = h+1;
    end

    % Take care of the last bit of leftover.
    if (leftover > 0)
        D = leftover*draw;
        draws(ID, interval+h) = draws(ID, interval+h) + D;
    end
end
for n=1:k
  % Column 1: ID. Assigning ID's
  etakeInit(n,1) = n; % Count up ID #'s.

  % Column 2: Bedrooms. Assigning number of bedrooms consistent with data
  % Column 3: Volume. Assigning gallons immediately converted to Liters.
  % Column 4: Energy take necessary to turn off.
  % Column 5: Energy take necessary to turn on.
  % Column 6: Adjusted losses based on surface area (default is 50 gal).
  % Column 7: Power of heating coil on unit in Watts.
  if (n <= 500) % If five bedrooms.
    etakeInit(n,2) = 5; % Then assign five bedrooms.
    etakeInit(n,3) = 80 * 3.78541; % Size assignment in Liters
    etakeInit(n,4) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-OFF
    etakeInit(n,5) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-ON
    etakeInit(n,6) = losses*(SA(1,4)/SA(2,4)); % Correction for 50 to 50 gal.
    etakeInit(n,7) = coil;
  elseif (n <= 1500) % If four bedrooms.
    etakeInit(n,2) = 4; % Then assign four bedrooms.
    etakeInit(n,3) = 80 * 3.78541; % Size assignment in Liters
    etakeInit(n,4) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-OFF
    etakeInit(n,5) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-ON
    etakeInit(n,6) = losses*(SA(2,4)/SA(2,4)); % Correction for 50 to 50 gal.
    etakeInit(n,7) = coil;
  elseif (n <= 5000) % If three bedrooms.
    etakeInit(n,2) = 3;
    etakeInit(n,3) = 50 * 3.78541;
    etakeInit(n,4) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-OFF
    etakeInit(n,5) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-ON
    etakeInit(n,6) = losses*(SA(3,4)/SA(2,4)); % Correction for 50 to 50 gal.
    etakeInit(n,7) = coil;
  elseif (n <= 8400) % If two bedrooms.
    etakeInit(n,2) = 2;
    etakeInit(n,3) = 40 * 3.78541;
    etakeInit(n,4) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-OFF
    etakeInit(n,5) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-ON
    etakeInit(n,6) = losses*(SA(3,4)/SA(2,4)); % Correction for 50 to 50 gal.
    etakeInit(n,7) = coil;
  else % If one bedroom.
    etakeInit(n,2) = 1;
    etakeInit(n,3) = 40 * 3.78541;
    etakeInit(n,4) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-OFF
    etakeInit(n,5) = (deadband)/1.0 * 4184 * etakeInit(n,3) / 3600; % E-ON
    etakeInit(n,6) = losses*(SA(3,4)/SA(2,4)); % Correction for 50 to 50 gal.
    etakeInit(n,7) = coil;
  end

  % Column 8: Energy Take. Assigning initial Energy Take in W-hrs.
  % Randomize an initial temperature of unit to get an initial E-Take.
  etakeInit(n,8) = rand()*(etakeInit(n,5)-etakeInit(n,4)) + etakeInit(n,4);
% Column 9: Oper Status. Assigning Initial Condition (Idle = 1; On = 2).
% Energy between turn off and turn on.
Egap = etakeInit(n,5)-etakeInit(n,4); % Watt-hr
% Time idling.
t_idle = (Egap / etakeInit(n,6)); % Time to go from off to on in seconds.
% Time importing energy.
t_imp = (Egap / etakeInit(n,7)); % Time to go from on to off in seconds.
idlefrac = t_idle / (t_idle + t_imp);
% I calculated the chance that the EWH is importing or idling.
% Now, I will test a rand(). If less than idlefrac, then idle, else on.
test = rand();
if(test < idlefrac)
etakeInit(n,9) = 1; % Status for idle.
else
etakeInit(n,9) = 2; % Status for importing.
end
end
end
%Simulate Energy Takes

function [ETAKE] = SimETAKE(etaknInit, draws)

% Purpose: Calculates energy takes at each 5 minute time interval.
% Includes initial data at front of array.

% Inputs
% etakeInit: Includes the initial data at the front of the array like
% bedroom number, ID#,
% draws: The energy takes amassed by each hot water draw for each 5 min
% period. N x M matrix with N rows of EWH units and M columns of 5 min
% intervals and statuses. M over a year is typically 2*288*365.

% Outputs
% ETAKE: An N x (M+7) matrix, with N rows of EWH units and M columns of 5
% minute intervals plus 7 columns of initial data.
% Column 1: ID#.
% Column 2: # of Bedrooms.
% Column 3: Volume of EWH in Liters. (Also mass in kg because sg=1)
% Column 4: Energy takes at which point the unit turns off.
% Column 5: Energy take at which point the unit turns on.
% Column 6: Ambient losses of the unit.
% Column 7: EWH coil power.

%Assumptions
Tamb = 51; % Ambient water temperature in degrees.
Tset = 120; % Setpoint of water heater.

% Pre sizing ETAKE Matrix
SIZE = size(etakeInit);
ETAKE = zeros(SIZE(1), 7+(2*288*365));

% Transferring initial data from etakeInit to ETAKE.
SIZE = size(ETAKE);
for n=1:SIZE(1)
    ETAKE(n,1)=etakeInit(n,1);
    ETAKE(n,2)=etakeInit(n,2);
    ETAKE(n,3)=etakeInit(n,3);
    ETAKE(n,4)=etakeInit(n,4);
    ETAKE(n,5)=etakeInit(n,5);
    ETAKE(n,6)=etakeInit(n,6);
    ETAKE(n,7)=etakeInit(n,7);
    ETAKE(n,8)=etakeInit(n,8);
    ETAKE(n,9)=etakeInit(n,9);
end

% Simulating EWH Behavior to Calculate Energy Take at each interval.
for n=5:(SIZE(2)/2) % For each column, starting at 10 and counting by 2's.
    for m=1:SIZE(1) % For each row.
        ELoss = ETAKE(m,n)/12; % Losses in Watt-hrs distributed over 5 min.
        EGain = ETAKE(m,n+7)/12; % Gains distrubted over 5 min.
E\text{Gap} = \text{ETAKE}(n, 5) - \text{ETAKE}(n, 4); \quad \% \text{Difference between ON \text{ETAKE} and OFF \text{ETAKE}}.

\text{ID} = \text{cast}(\text{ETAKE}(n, 1), 'uint32');
\text{DrawInterval} = \text{cast}(\text{m-4}, 'uint32');
\text{T}_{\text{tank}} = \text{T}_{\text{set}} - (\text{ETAKE}(n, 2, m-2) * 1.8 * 3600) / (\text{ETAKE}(n, 3) * 4184);
\text{D} = \text{draws}(\text{ID}, \text{DrawInterval}); \quad \% \text{kg of draw in the interval.}
\text{E}\text{Draw} = (D * 4184 * ((\text{T}_{\text{tank}} - \text{T}_{\text{Amb}})/1.8)) / 3600; \quad \% \text{Amount of energy lost in draw in W-hrs.}

\text{if}(\text{ETAKE}(n, 2, m-1) == 1) \% \text{If unit is IDLE coming into 5 min period.}
\begin{align*}
\text{E} &= \text{ETAKE}(n, 2, m-2) + \text{E}\text{Loss} + \text{E}\text{Draw}; \quad \% \text{Add losses to prior energy take, and check.}
\end{align*}
\begin{align*}
\text{if}(\text{E} < \text{ETAKE}(n, 5)) \quad \% \text{Verify this new energy take would not cause unit to turn on.}
\text{ETAKE}(n, 2, m) &= \text{E}; \quad \% \text{Set current \text{ETAKE} to E, because unit did not turn on.}
\text{ETAKE}(n, 2, m-1) &= 1; \quad \% \text{Set current status to 1 (IDLE) because unit did not turn on.}
\end{align*}
\begin{align*}
\text{else} \quad \% \text{Unit goes past ON energy take, and is turned on.}
\text{X}_1 &= \text{ETAKE}(n, 5) - \text{ETAKE}(n, 2, m-2); \quad \% \text{Gap between ON \text{ETAKE} and prior \text{ETAKE}.}
\text{X}_2 &= \text{E} - \text{ETAKE}(n, 2, m-2); \quad \% \text{Gap between new \text{ETAKE} and prior \text{ETAKE}.}
\% \text{I am assuming linearity of loss on the individual interval.}
\text{R} &= 1 - (\text{X}_1/\text{X}_2); \quad \% \text{R is the remaining fraction of the interval over which the heating start.}
\% \text{This is the time after the energy take reaches the energy take necessary to turn the unit on. I assumed the energy takes linearly over each time interval.}
\text{G} &= (\text{R} * \text{EGain}); \quad \% \text{G = change in energy take if coil were to heat remainder of five minute time interval.}
\text{E}_1 &= \text{E} - \text{G}; \quad \% \text{E}_1 \text{ would be the new energy take, unless the heating would cause the energy take to go past the OFF \text{ETAKE}.}
\text{E}_2 &= \text{ETAKE}(n, 4); \quad \% \text{E}_2 \text{ represents the minimum \text{ETAKE possible, at which point the coil turns off, and the EWH idles.}
\text{if}(\text{E}_1 > \text{E}_2) \% \text{If gains are not maxed out, then subtract G from E for new energy take, and set status to importing (status = 2 for importing.).}
\begin{align*}
\text{ETAKE}(n, 2, m) &= \text{E} - \text{G}; \quad \% \text{Set current \text{ETAKE}.}
\text{ETAKE}(n, 2, m+1) &= \text{E}; \quad \% \text{Set status to 2 (IMPORTING).}
\end{align*}
\text{else} \quad \% \text{Else, EWH heats up and turns off. Need to calculate remaining time period for losses.}
\% \text{Again, I assumed linearity.}
\text{X}_1 &= \text{E} - \text{E}_2;
\text{X}_2 &= \text{E} - \text{E}_1;
\text{R}_2 &= \text{R} * (1 - \text{X}_1/\text{X}_2); \quad \% \text{R}_2 \text{ is the remaining fraction of}
% the five minute interval, where
% energy take increases due to
% losses and possible draws.
E3 = ETAKE(n, 4) + R2*(Eloss + Edraw); % Adding the
% ambient losses and draw losses
% over the fractional time
% remaining in the 5 min interval.
% These losses are added to the
% OFF energy take.

% I assume here that the fractional ambient losses and
% draw losses are not enough to drag the energy take
% below the ON energy take.
EIAKE(n, 2*m) = E3; % Recalculated from the OFF E-TAKE
% plus the losses in the remaining
% fraction of the interval.

ETAKE(n, 2*m+1) = 1; % Set current status to 1 (IDLE)
% because unit ends in idle.

end

elseif(ETAKE(n, 2*m-1) == 2) % If the unit starts in IMPORT mode.
    E = ETAKE(n, 2*m-2) + Eloss + Edraw - EGain; % Here I add the
    % ambient losses, the draw losses, and
    % subtract the energy gains from the
    % coil.
    if(E > ETAKE(n, 4)) % If the new energy, does not cause the unit
        % to switch to idle mode.
        ETAKE(n, 2*m) = E; % Then the new E-TAKE is calculated.
        ETAKE(n, 2*m+1) = 2; % The status remains 2 (IMPORTING).
    else
        % The EWH reaches the minimum energy take, and
        % goes into idle mode, turning its coil off.
        X1 = ETAKE(n, 2*m-2) - ETAKE(n, 4); % This is where the coil
        % switches to idle mode.
        X2 = ETAKE(n, 2*m-2) - E; % This is the energy take that
        % would have occurred had the coil not switched
        % off mid interval.
        R = 1 - (X1/X2); % Remaining fraction of interval over which
        % the coil is turned off, and only the ambient
        % losses and draw losses affect the energy
        % take.
        ETAKE(n, 2*m) = ETAKE(n, 4) + R*(Eloss + Edraw); % At the OFF
        % energy take, the coil turns off. Then the
        % ambient and draw losses add to the OFF
        % energy take for the remainder of the
        % interval.
        ETAKE(n, 2*m+1) = 1; % Status returns to 1 (IDLE).
    end
else
disp('Error in initial state setting');
end
end

end
function [X,I,power] = SimDec10(month, day, hour, min, duration, MW, ETAKE, draws)

% Purpose: Analyzes the effects of making a decrement bid into the EIM. A
% decrement bid is akin to reducing generation. By increasing load, DERAS
% effects a decrement bid.

% Inputs: Month, day, hour, and minute determine the interval at which the
% bid is occurring. Duration determines the length of time the bid is for.
% MW is the megawatts promised over that duration. The draws matrix is
% developed from the NIST data on hot water draws. The ETAKE matrix is the
% simulated data that this program comb through, assigns the decrement bid
% and then simulates the effects.

% Outputs: The program simulates a bid in the EIM market, and resimulates
% the energy take data. The output matrix, X, represents the resimulated
% data in light of the EIM bid over the course of a the day and the next
% day. (Potentially another day if I determine that the transient effects
% of a bid can last into more days.

% Assumptions
Tset = 120; % Setpoint of water in degrees.
Tamb = 51; % Ambient temperature of water in degrees.

% Determining the starting interval for the day and the starting interval
% for the bid.

bidStartInt = time2interval(month, day, hour, min);
dayStartInt = day2interval(month, day);
durInt = duration / 5;
finalInt = dayStartInt + 287 + 288;
delayInt = bidStartInt - dayStartInt;

SIZE1=size(ETAKE);
SIZE2=size(draws);
X = zeros(SIZE1(1),7+200*2*2); % 2 Days and 2 Entries (Status and E-Take).
% 7 is for the initial data in ETAKE array.
if(SIZE1(1) ~= SIZE2(1))
    disp('Error. Mismatched number of units for draw and energy takes')
    return
elseif(mod(duration,5) ~= 0)
    disp('Error. Invalid duration. Duration must be a multiple of 5.')
    disp('EIM markets run every 5 minutes')
    return
end

% Copying the preliminary data in stored in the first 7 columns.
for n = 1:SIZE1(1)
    X(n,1) = ETAKE(n,1);
    X(n,2) = ETAKE(n,2);
X(n,3) = ETAKE(n,3);  
X(n,4) = ETAKE(n,4);  
X(n,5) = ETAKE(n,5);  
X(n,6) = ETAKE(n,6);  
X(n,7) = ETAKE(n,7);  
end

% Copying data that occurs before the bid, and needs no resimulation.

if (delayInt > 0)
   %counter = 1;
   for m = i:delayInt%(6 + 2*dayStartInt): (6+2*(dayStartInt+delayInt)
      for n = i:SIZEE(1)
         X(n,6+2*m) = ETAKE(n,4+2*(dayStartInt)+2*m);
         X(n,7+2*m) = ETAKE(n,5+2*(dayStartInt)+2*m);
      end
   end
end

% Sorting rows by energy take in the interval before the bid.
%y = 6+2*delayInt
sorted = sortrows(X,6+2*delayInt,'descend');
X = sorted;

% Assigning units to cover the bid in initial time interval.
W = MWh * 10^6;     % Total Power Owed across each interval.
Whr = MWh * 10^6 * (duration/60);  % Total watt hours owed by our bid.
intWhr = MWh * (10^6) * 1/12;       % Energy in W-hrs per interval owed.

intAcct = zeros(1,durInt);  % This tracks the periods over which the unit may
for n = 1:durInt
   intAcct(n) = intWhr;       % have turned on due to losses, and therefore
   % would have been part of the forecasted load. In this case I can include the heating from
   % deadband to setpoint, but not from shedband to deadband, because the unit would have turned on
   % anyway. I account for this by immediately
   % counting the full amount of power, but I return % the shedband to deadband energy to the period
   % over which the unit would have turned on.
   %y = 6 + delayInt * 2

% Assigning Unit Statuses and Performing Preliminary Accounting
unitCtrl = 0;
for n = i:SIZEE(1)
   if(X(n,(7 + delayInt*2))==2)  % If EWH is already ON, then skip.
      continue
   end
   % CtrlON mode = 3. This means we have assigned the EWH to cover the bid
   X(n,(7+delayInt*2)) = 3;       % Set to CtrlON mode.
unitCtr = unitCtr + 1; % Track number of units I turn ON.
ETotal = X(n,(6 + delayInt*2)); % Total Energy Take Available.
EMax = X(n,7) / 1/12; % Maximum Energy in W-hrs that can be
% consumed during next time period.

% Account for energy removal.
if (EMax < ETotal)
    intAcct(1) = intAcct(1) - EMax;
else
    intAcct(1) = intAcct(1) - ETotal;
end
% Check to see if unit would have ordinarily turned ON over duration.
ELoss = X(n,6) * 1/12; % Ambient losses over an interval.
EON = X(n,5); % Energy take when unit turns ON.
Diff = EON - ETotal; % Energy loss necessary to turn the unit
% ON.
ERest = X(n,5) - X(n,4); % Amount of energy owed back to account
% for the load coming on anyway.
ETest = 0; % Counted variable for testing if unit
% would turn on.

flag = 0; % A flag for checking if I already paid back the energy.
for x = 1:durInt % Checking each interval to see if the unit turns ON.
    ETest = ETest + ELoss; % Increment the losses each period.
    if (ETest > Diff) % If the EH turns on during this interval...
        ctr = 0; % Counter for counting across extra intervals.
        flag = 1; % Mark that the unit would have turned on.
        % Pay back the energy to later intervals.
        while (ERest > 0)
            if ((x + ctr) > durInt) % If the energy restitution would
                break % fall outside of the bounded time period. Then
                break, because we do not owe this energy back.
            if (EMax <= ERest) % If bounded by max energy.
                intAcct(x+ctr) = intAcct(x+ctr) + EMax; % Acct for EMax.
                ERest = ERest - EMax; % Remove EMax from restitution.
            else % Else, we return the balance of the restitution.
                intAcct(x+ctr) = intAcct(x+ctr) + ERest; % ERest back.
                ERest = 0;
                break % Break, because we do not owe anything more.
            end
            ctr = ctr + 1;
        end
    end
    if (flag == 1)
        break
    end
end
if (intAcct(1) < 0)
    %unitCtr; %display number of units I turn on for now
    break
end
% Running simulation for the bid only.
% test1 = 6 + delayInt*2
% test = 6 + 2*(delayInt+durInt)
% intAcct
% m = (6 + 2*(delayInt+1)):2:(6 + 2*(delayInt+durInt))
% SIZE1
% SIZE2
% dayStartInt

% AcctIndex = 1:
% SIZE3 = size(intAcct)
% addedUnits = 0;
% subbedUnits = 0;
% m = (6 + 2*(delayInt+1)):2:(6 + 2*(delayInt+durInt))
% n = 1:SIZE1(1)
% m = (6 + 2*(delayInt+1)):2:(6 + 2*(delayInt+durInt));
lastIntFlag = 0;
for m = (6 + 2*(delayInt+1)):2:(6 + 2*(delayInt+durInt)) % from starting
  % interval of bid, to finish interval of bid.

% First Simulate Current Interval
for n = 1:SIZE1(1) % from first unit to last unit (sorted by
  % energy take, with highest being on top.

  ELoss = ETAKE(n,6)/12; % Losses in Watt-hrs distributed over 5 min
  EGain = ETAKE(n,7)/12; % Gains distributed over 5 min
  EGap = ETAKE(n,5) - ETAKE(n,4); % Difference between ON ETAKE and
  % OFF ETAKE.

  ID = cast(ETAKE(n,1), 'uint32'): % Looking up ID from ETAKE matrix.
  % NOTE: ID's have been shuffled due % due to the sorting by E-take's.

dayInt = (m - 6) / 2; % Calculating the daily interval in terms % of m.
DrawInterval = cast(dayInt + dayStartInt - 1, 'uint32'): % Correcting % to the interval across the year.
Itank = max - (ETAKE(n,m-2)*1.8*3600)/(ETAKE(n,3)*4184); D = draw(1, DrawInterval); % kg of draw in the interval.
EDraw = (D * 4184 * ((Itank - Tank)/1.8))/3600; % Amount of energy

% Now test the three cases, 1 = IDLE, 2 = IMPORT, 3 = CtrlIMPORT
% If IDLE...
n
% if X(n,m-1) == 1 % If unit is IDLE coming into 5 min period.
  E = X(n,m-2) + ELoss + EDraw; % Add losses to prior % energy take, and check.
  if(E < X(n,5)) % Verify this new energy take % would not cause unit to turn on.
\( X(n,m) = E; \) % Set E-TAKE in current interval to
\( E, \) because unit did not turn on.
\( X(n,m+1) = 1; \) % Set current status to 1 (IDLE)
\( \) because unit did not turn on.

`else`
% Unit goes past ON energy take, and is turned on.
\( X_1 = X(n,5) - X(n,m-2); \) % Gap between ON E-TAKE
\( \) and prior E-TAKE.
\( X_2 = E - X(n,m-2); \) % Gap between new E-TAKE and
\( \) Prior E-TAKE.

% I am assuming linearity of loss on the individual
% interval.
\( R = 1 - (X_1/X_2); \) % R is the remaining fraction of the
% interval over which the heating occurs.
% This is the time after the energy take
% reaches the energy take necessary to
% turn the unit on. I assumed the energy
% take increases linearly over each time
% interval.
\( G = (R * E\text{Gain}); \) % G = change in energy take if coil were
% to heat remainder of five minute time
% interval.
\( E_1 = E - G; \) % E1 would be the new energy take, unless
% the heating would cause the energy take
% to go past the OFF E-TAKE.
\( E_2 = X(n,4); \) % E2, represents the minimum E-TAKE
% possible, at which point the coil
% turns off, and the EWH idles.

`if(E_1 > E_2)` % If gains are not maxed out, then subtract G
% from E for new energy take, and set status
% to importing (status = 2 for importing).
\( X(n,m) = E - G; \) % Set current E-Take.
\( X(n,m+1) = 2; \) % Set status to 2 (IMPORT).

`else`
% Else, EWH heats up and turns off. Need to
% calculate remaining time period for losses.
% Again, I assumed linearity.
\( X_1 = E - E_2; \)
\( X_2 = E - E_1; \)
\( R_2 = R * (1 - X_1/X_2); \) % R2 is the remaining fraction of
% the five minute interval, where
% energy take increases due to
% losses and possible draws.
\( E_3 = X(n,4) + R_2*(E\text{Loss} - E\text{Draw}); \) % Adding the
% ambient losses and draw losses
% over the fractional time
% remaining in the 5 min interval.
% These losses are added to the
% OFF energy take.

% I assume here that the fractional ambient losses and
% draw losses are not enough to drag the energy take
% below the ON energy take.
\( X(n,m) = E_3; \) % Recalculated from the OFF E-TAKE
% plus the losses in the remaining
% fraction of the interval.

X(n,m+1) = 1; % Set current status to 1 (IDLE)
% because unit ends in idle.
end
end

% If Importing because the unit reached it's maximum energy take...
elseif(X(n,m-1) == 2) % EWH in IMPORT mode by an internal decision
    E = X(n,m-2) + ELoss + EDraw - EGain; % Here I add the
    % ambient losses, the draw losses, and
    % subtract the energy gains from the
    % coil.
    if(E>X(n,4)) % If the new energy, does not cause the unit
    % to switch to idle mode.
        X(n,m) = E; % Then the new E-TAKE is calculated.
        X(n,m+1) = 2; % The status remains 2 (IMPORT).
    else % The EWH reaches the minimum energy take, and
        % goes into idle mode, turning its coil off.
        X1 = X(n,m-2) - X(n,4); % This is where the coil
        % switches to idle mode.
        X2 = X(n,m-2) - E; % This is the energy take that
        % would have occurred had the coil not switched
        % off mid interval.
        R = 1 - (X1/X2); % Remaining fraction of interval over which
        % the coil is turned off, and only the ambient
        % losses and draw losses affect the energy
        % take.
        X(n,m) = X(n,4) + R*(ELoss + EDraw); % At the OFF energy
        % take, the coil turns off. Then the ambient
        % and draw losses add to the OFF energy take
        % for the remainder of the interval.
        X(n,m+1) = 1; % Status returns to 1 (IDLE).
    end
end

% If Importing because DERAS took over control...
elseif(X(n,m-1) == 3) % EWH in CtrlIMPORT mode, controlled by DERAS
    E = X(n,m-2) + ELoss + EDraw - EGain; % Here I add the
    % ambient losses, the draw losses, and
    % subtract the energy gains from the
    % coil. Same as for state 2.
    if(E>0) % If the new energy, does not cause the unit
    % to switch to idle mode. Same as state 2,
    % except that the idle mode switch occurs at
    % the set point, where energy take equals 0.
        X(n,m) = E; % Then the new E-TAKE is calculated.
        X(n,m+1) = 3; % The status remains 3 (CtrlIMPORT).
    else % The EWH reaches the minimum energy take (0), and
        % goes into idle mode, turning its coil off.
        X1 = X(n,m-2) - 0; % This is where the coil
        % switches to idle mode.
X1 = X(n,n-2) - E; % This is the energy that
% would have occurred had the coil not switched
% off mid interval.
R = 1 - (X1/X2); % Remaining fraction of interval over which
% the coil is turned off, and only the ambient
% losses and draw losses affect the energy
% take.
X(n,m) = 0 + R*(ELoss + EDraw); % At the OFF energy
% take, the coil turns off. Then the ambient
% and draw losses add to the OFF energy take
% for the remainder of the interval.
X(n,m+1) = 1; % Status returns to 1 (IDLE).
else
    disp('Error in initial state setting');
X(n,m-1)
end

% Check which units are still in CtrlIMPORT (3) in next interval.
% CtrlCntr = 0;
stillONCntr = 0;
ECtr = 0;
for n = 1:SIZE(1)
    if (X(n,m+1) == 3)
        stillONCntr = stillONCntr + 1;
        ETotal = X(n,m);
        EMax = X(n,7) * 1/12; % Maximum Energy in N-hrs that can
                                % be consumed during next interval.
        % Account for energy removal.
        if (EMax < ETotal)
            ECtr = ECtr + EMax;
        else
            ECtr = ECtr + ETotal;
        end
    end
end

% Check the amount of energy owed on the next interval.
AcctIndex = (((m-6)-(2*DelayInt))/2) + 1; % Determines the acct index
                                                % based on m and delayInt.
if (AcctIndex > durInt) % Checking to see we aren't going past the
                        % bounds of the IntAcct matrix.
    lastIntFlag = 1; % Set the flag to non-zero if we do.
end

% Compare next acct with the units already in CtrlIMPORT mode (3).
if (lastIntFlag == 0) % Verify I'm not out of bounds.
    if (IntAcct(AcctIndex) > ECtr)
        intAcct(AcctIndex) = intAcct(AcctIndex) - ECtr; % Energy owed.
sorted = sortrows(X,m,'descend');
X = sorted;
% Assigning Units and Accounting for Units that would turn ON.
for n = 1:SIZE(1)
    if(X(n,m+1) == 2) % If EWH is already IMPORT, then skip.
        continue
    elseif(X(n,m+1) == 3) % If CtrlIMPORT, then skip.
        continue
    else % If IDLE, then convert to CtrlIMPORT.
        X(n,m+1) = 3; % Set to CtrlIMPORT mode.
        ETotal = X(n,m); % Total energy take available.
        EMax = X(n,7) ' 1/12; % Maximum energy take available.

        % Account for energy removal.
        if (EMax < ETotal)
            intAcct(AcctIndex) = intAcct(AcctIndex) - EMax;
        else
            intAcct(AcctIndex) = intAcct(AcctIndex) - ETotal;
        end

        % Check to see if unit would have ordinarily turned ON
        % over the bid duration.
        ELoss = X(n,6) ' 1/12; % Ambient losses over interval.
        EON = X(n,5); % E-take when unit turns ON.
        Diff = EON - ETotal; % Energy loss necessary to turn
        % the unit ON.
        ERest = X(n,5) - X(n,4); % Amount of energy owed back to
        % account for the load coming
        % on anyway.
        ETest = 0; % Counted variable for testing
        % if unit would turn on.
        flag = 0; % A flag for checking if I
        % already paid back the energy.
        for x = AcctIndex:durInt % Checking each interval to
        % see if the unit turns ON.
            ETest = ETest + ELoss; % Increment the losses each
            % period.
            if (ETest > Diff) % If the EWH turns on during
            % this interval...
                ctr = 0; % Counter for counting across
                % extra intervals.
                flag = 1; % Mark that the unit would have
                % turned on.
            % Pay back the energy to later intervals.
                while (ERest > 0)
                    if ((x + ctr) > durInt) % If the energy
                        % restitution would fall outside
                        break % of the bounded time period. Then
                    end % break, because we do not owe
                    % this energy back.
                    if (EMax < ERest) % If bounded by max
% energy.
intAcoct(x+ctr) = intAcoct(x+ctr) + EMax;
% Acct for EMax.
ERest = ERest - EMax; % Remove EMax
%from restitution tally.
else
% Else, we return the balance of
% the restitution.
intAcoct(x+ctr) = intAcoct(x+ctr) + ERest;
% Balance of ERest paid back.
ERest = 0;
break % Break, because we do not owe
% anything more.
end
ctr = ctr + 1;
end
end
if (flag == 1)
break
end
end
if (intAcoct(AcctIndex) < 0)
%unitCtrl; %display number of units I turn on for now
break
end
end
end

elseif (intAcoct(AcctIndex) < (ETr - 4500/12))
% Then subtract units.
intAcoct(AcctIndex) = intAcoct(AcctIndex) - ETr;
% This will be a negative number, the magnitude indicating how
% energy take we must get to by turning units off.
sorted = sortrows(X,m,'ascend'); % Sort from lowest to highest
% energy takes such that we
% turn lowest energy take
X = sorted; % units off first.

for n = 1:size(1)
if(X(n,m+1) == 2) % If in uncontrolled IMPORT, then skip.
continue
elseif(X(n,m+1) == 1) % If IDLE, then skip.
continue
else % If in CtrlIMPORT mode, then turn OFF.
X(n,m+1) = 1; % Set to CtrlIMPORT mode.
ETotal = X(n,m); % Total energy take available.
EMax = X(n,7) * 1/12; % Maximum energy take available.
% Account for energy addition.
if (EMax < ETotal)
    intAcct(intAcctIndex) = intAcct(intAcctIndex) + EMax;
else
    intAcct(intAcctIndex) = intAcct(intAcctIndex) + ETotal;
end

if (intAcct(intAcctIndex) > -(4500/12))
    unitCtr;% display number of units I turn on for now
    break
end

end

sorted = sortrows(X,m,'descend');% Sort from highest to lowest
    % so that everything is back
    % to normal.

% else, we fall into a band range of a single water heater, so do
% nothing. We are close enough.
else
    % I'm at the end of the bid interval. Convert units in mode 3
    % back to modes 1 and 2.

    % Cleaning up if on the last interval.
    for n = 1:SIZE1(1)
        if (m == (6 + 2*(delayInt+durInt)) ) % If on the last interval.
            if (X(n,m+1) == 3) % And if the unit was in CtrlON mode (3).
                if (X(n,m) >= X(n,5)) % Convert to IMPORT mode if above
                    X(n,m+1) = 2; % max idle energy take.
                else
                    X(n,m+1) = 1; % Convert to IDLE mode if below
                end
            end
    end
    % Cycle through all units until modes are 1 or 2.
end

end

% Now simulate the rest of intervals where no bid is being acted upon.
sorted = sortrows(X,i,'ascend');% Sort from lowest to highest ID# so that
    % so that everything is back to normal.
for n = (6+2*(delayInt+durInt+1)):2:((6+2*(285*2))) % Cycles after the bid.

    % Simulate Current Interval
    for n = 1:SIZE1(1) % From first unit to last unit (sorted by
        % energy take, with highest being on top.

        ELoss = X(n,6)/12; % Losses in Watt-hrs distributed over 5 min
        EGain = X(n,7)/12; % Gains distributed over 5 min
Egap = X(n,5) - X(n,4); % Difference between ON E-TAKE and OFF E-TAKE.
ID = cast(X(n,1), 'uint32'); % Looking up ID from E-TAKE matrix.
% NOTE: ID's have been shuffled due to the sorting by E-take's.
dayInt = (m - e) / 2; % Calculating the daily interval in terms of m.
DrawInterval = cast(dayInt + dayStartInt - 2, 'uint32'); % Correcting to the interval across the year.
Itank = Itank - (X(n,m-2)*1.8*3600)/(X(n,3)*4184);
D = draws(ID, DrawInterval); % kg of draw in the interval.
EDraw = (D * 4184 * ((Itank - Tamb)/1.8))/3600; % Amount of energy

% Now test the two cases, 1 = IDLE and 2 = IMPORT. CtrlIMPORT (3) % does not exist anymore, because we are outside of the bid % intervals.

% If IDLE...
if(X(n,m-1) == 1) % If unit is IDLE coming into 5 min period.
E = X(n,m-2) + ELoss + EDraw; % Add losses to prior energy take, and check.
if(E < X(n,5)) % Verify this new energy take would not cause unit to turn on.
X(n,m) = E; % Set E-TAKE in current interval to E, because unit did not turn on.
X(n,m+1) = 1; % Set current status to 1 (IDLE) because unit did not turn on.
else % Unit goes past ON energy take, and is turned on.
X1 = X(n,5) - X(n,m-2); % Gap between ON E-TAKE and prior E-TAKE.
X2 = E - X(n,m-2); % Gap between new E-TAKE and prior E-TAKE.
% I am assuming linearity of loss on the individual interval.
R = 1 - (X1/X2); % R is the remaining fraction of the interval over which the heating occurs.
% This is the time after the energy take reaches the energy take necessary to turn the unit on. I assumed the energy take increases linearly over each time interval.
G = (R * EGain); % G = change in energy take if coil were to heat remainder of five minute time interval.
E1 = E - G; % E1 would be the new energy take, unless the heating would cause the energy take to go past the OFF E-TAKE.
E2 = X(n,4); % E2, represents the minimum E-TAKE possible, at which point the coil turns off, and the EWH idles.
if(E1 > E2) % If gains are not maxed out, then subtract G
% from E for new energy take, and set status
to importing (status = 2 for importing).
\[ X(n,m) = E - G; \] % Set current E-Take.
\[ X(n,m+1) = 2; \] % Set status to 2 (IMPORT).
else
% Else, EWH heats up and turns off. Need to
calculate remaining time period for losses.
% Again, I assumed linearity.
\[ X1 = E - E2; \]
\[ X2 = E - E1; \]
\[ R2 = R \ast (1 - X1/X2); \] % R2 is the remaining fraction of
% the five minute interval, where
% energy take increases due to
% losses and possible draws.
\[ E3 = X(n,4) + R2 \ast (Eloss + EDraw); \] % Adding the
% ambient losses and draw losses
% over the fractional time
% remaining in the 5 min interval.
% These losses are added to the
% OFF energy take.
% I assume here that the fractional ambient losses and
draw losses are not enough to drag the energy take
% below the ON energy take.
\[ X(n,m) = E3; \] % Recalculated from the OFF E-TAKE
% plus the losses in the remaining
% fraction of the interval.
\[ X(n,m+1) = 1; \] % Set current status to 1 (IDLE)
% because unit ends in idle.
end
end

% If Importing because the unit reached it's maximum energy take...
elseif(X(n,m-1) == 2) % EWH in IMPORT mode by an internal decision
\[ E = X(n,m-2) + Eloss + EDraw - EGain; \] % Here I add the
% ambient losses, the draw losses, and
% subtract the energy gains from the
% coil.
if(E>X(n,4)) % If the new energy, does not cause the unit
to switch to idle mode.
\[ X(n,m) = E; \] % Then the new E-TAKE is calculated.
\[ X(n,m+1) = 2; \] % The status remains 2 (IMPORT).
else
% The EWH reaches the minimum energy take, and
goes into idle mode, turning its coil off.
\[ X1 = X(n,m-2) - X(n,4); \] % This is where the coil
% switches to idle mode.
\[ X2 = X(n,m-2) - E; \] % This is the energy take that
% would have occurred had the coil not switched
% off mid interval.
\[ R = 1 - (X1/X2); \] % Remaining fraction of interval over which
% the coil is turned off, and only the ambient
% losses and draw losses affect the energy
% take.
\[ X(n, m) = X(n, 1) + R^* (E_{\text{Loss}} + E_{\text{Draw}}); \] % At the OFF energy
% take, the coil turns off. Then the ambient
% and draw losses add to the OFF energy take
% for the remainder of the interval.
\[ X(n, m+1) = 1; \] % Status returns to 1 (IDLE).

end
else
    disp('Error in initial state setting');
    X(n, m-1)
end
end % End n for loop for number of units.
end % End m for loop for number of intervals.

\texttt{@intAcct}
SIZE = size(intAcct);
power = zeros(1, SIZE(2));
failctr = 0;
for n=1:SIZE(2)
    intAcct(n);
    power(n) = (-intAcct(n)) / (10^6) + 15/12; % Technically counts energy.
    if (intAcct(n) > 0)
        failctr = failctr+1;
        disp('Yep');
    end
end

total_power = sum(power) * 12; % Now, I multiply the energy in Watt-hrs by 12/perm hr.
total_power = total_power
end % End function.
function Energy = DayPlotter(mo, day, ETAKE)

% Purpose: Plots.

% Inputs
% Date in the form of month and day.

% Outputs
% Plot of energy usage by interval for each 5 minute interval.

SIZE = size(ETAKE);

interval = day2interval(mo, day);
Energy = zeros(1, 128);
UnitsON = zeros(1, 128);
counter = 1;
for m=interval:(interval+237)
    E = 0;
    U = 0;
    for n=1:SIZE(1)
        E = E + ETAKE(n, 6+2*m);
        if (ETAKE(n, 7+2*m) == 2)
            U = U + 1;
        end
    end
    Energy(counter) = E;
    UnitsON(counter) = U;
    counter = counter + 1;
end

x = 1:238;

figure(1)
plot(x.*(1/12),Energy.*(1/1000000))
xlabel('Hour')
ylabel('Energy Take (MWh)')
axis([0 24 0 10])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24])

figure(2)
plot(x.*(1/12),UnitsON.*(1./SIZE(1)).*100)
xlabel('Hour')
ylabel('% of Units Turned On')
axis([0 24 0 40])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24])
% DayPlotter7, newest plotting tool for Dec Bids with power calculator
% function Energy = DayPlotter4(X)
% Purpose: Plots.
mc = 6;
day = 6;
hr = 2; % 0 through 23
minInt = 0; % 0 through 11.
duration = 12*5*1; % must be multiples of 5.
amount = 6; % MegaWatts

[X,f,power]=SimDec10(mci,day,hr,minInt,duration,amount,ETAKE,draws);
SIZE = size(X);

% E1 = zeros(1,288);
E2 = zeros(1,288);
% E3 = zeros(1,288);
STAT11 = zeros(1,288);
STAT21 = zeros(1,288);
STAT31 = zeros(1,288);
STAT12 = zeros(1,288);
STAT22 = zeros(1,288);
STAT32 = zeros(1,288);
ON1 = zeros(1,288);
ON2 = zeros(1,288);
Max = zeros(1,200);
Min = zeros(1,200);

for n=1:SIZE
    for m=1:SIZE(1)
        E1(i,m) = E1(i,m) + X(n,6+m*2);
        E2(i,m) = E2(i,m) + X(n,(6+m*2 + 288*2));
        % E3(i,m) = E3(i,m) + X(n,(6+m*2 + 288*4));
        if (X(n,7+m*2) == 1)
            STAT11(i,m) = STAT11(i,m) + 1;
        elseif (X(n,7+m*2) == 2)
            STAT21(i,m) = STAT21(i,m) + 1;
            ON1(i,m) = ON1(i,m) + 1;
        else
            STAT31(i,m) = STAT31(i,m) + 1;
            ON1(i,m) = ON1(i,m) + 1;
        end
        if (X(n,7+m*2+288*2) == 1)
            STAT12(i,m) = STAT12(i,m) + 1;
        elseif (X(n,7+m*2+288*2) == 2)
            STAT22(i,m) = STAT22(i,m) + 1;
            ON2(i,m) = ON2(i,m) + 1;
        else
            STAT32(i,m) = STAT32(i,m) + 1;
        end
    end
end
ON2(1,m) = ON2(1,m) + 1;
end
Max(1,m) = Max(1,m) + X(n,5);
Min(1,m) = Min(1,m) + X(n,4);
end

E1(55)

AVG = (MAX - MIN)./Z + MIN;
x = 1:288;
%Energy = DayPlotter2(mo, day, ETAKE, draws);

figure(5)
plot(x.'(1/12),100.*(ON2./SIZE(1)))
xlabel('Hour')
ylabel('%% of Units Heating')
axis([0 24 0 40])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);

[Energy, UnitsON] = DayPlotterSub(mo, day, ETAKE, draws);
%Energy = Energy(2:289);
%Energy = [Energy,0];
figure(1)
plot(x.'(1/12),E1.*(1/1000000))
xlabel('Hour')
ylabel('Energy Taken (MWh-hr)')
axis([0 24 0 10])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);
hold on
plot(x.'(1/12),Max.*(1/1000000))
plot(x.'(1/12),Min.*(1/1000000))
plot(x.'(1/12),AVG.*(1/1000000))
%Energy = DayPlotterSub(mo, day-i, ETAKE, draws);
plot(x.'(1/12),Energy.*(1/1000000), 'k:')
hold off

figure(2)
plot(x.'(1/12),100.*(ON1./SIZE(1)))
xlabel('Hour')
ylabel('%% of Units Heating')
axis([0 24 0 40])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);
hold on
plot(x.'(1/12),100.*(UnitsON./SIZE(1)), 'k:')
hold off

Ucheck = (UnitsON - ON1) ./ UnitsON;
Echeck = (Energy - Ei) / Energy;

[Energy, UnitsON] = DayPlotterSub(mo, day+1, ETAKE, draws);
%Energy = Energy(2:286);
%Energy = [Energy, 0];
figure(3)
plot(x.*(1/12),E2.*(1/1000000))
xlabel('Hour')
ylabel('% of Units ON')
axis([0 24 0 10])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);
hold on
plot(x.*(1/12),Max.*(1/1000000))
plot(x.*(1/12),Min.*(1/1000000))
plot(x.*(1/12),AVG.*(1/1000000))
%Energy = DayPlotterSub(mo, day+1, ETAKE, draws);
plot(x.*(1/12),Energy.*(1/1000000), 'k:')
hold off

figure(4)
plot(x.*(1/12),100.*(ON2./SIZE(1)))
xlabel('Hour')
ylabel('% of Units Heating')
axis([0 24 0 20])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);
hold on
plot(x.*(1/12),100.*(UnitsON./SIZE(1)), 'k:')
hold off

Ucheck2 = (UnitsON - ON2) ./ UnitsON;
Echeck2 = (Energy - E2) / Energy;
function [Energy, UnitsON] = DayPlotterSub(mo, day, ETAKE, draws)

% PURPOSE: Plots.

% Inputs
% Date in the form of month and day, and ETAKE matrix and draws matrix.

% Outputs
% Plot of energy usage by interval for each 5 minute interval.

SIZE = size(ETAKE);

interval = day2interval(mo, day);
Energy = zeros(1, 288);
UnitsON = zeros(1, 288);
DrawON = zeros(1, 288);
Max = zeros(1, 288);
Min = zeros(1, 288);
counter = 1;
for m = interval:(interval+287)
    E = 0;
    U = 0;
    for n = 1:SIZE(1)
        E = E + ETAKE(n, 6+2*m);
        if (ETAKE(n, 7+2*m) == 2)
            U = U + 1;
        end
        Max(counter) = Max(counter) + ETAKE(n, 5);
        Min(counter) = Min(counter) + ETAKE(n, 4);
    end
    Energy(counter) = E;
    UnitsON(counter) = U;
    counter = counter + 1;
end
end
% Power Plotter

mo = 6;
day = 6;
hr = 2; % 0 through 23
minInt = 0; % 0 through 11.
duration = 5*12*2; % must be multiples of 5.
amount = 1; % MegaWatts
[X,f,power1] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);
amount = 1.5;
[X,f,power15] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);
amount = 2;
[X,f,power2] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);
amount = 3;
[X,f,power3] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);
amount = 4;
[X,f,power4] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);
amount = 5;
[X,f,power5] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);
amount = 6;
[X,f,power6] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);
amount = 7;
[X,f,power7] = SimDec10(mo,day,hr,minInt,duration,amount,ETAKE,draws);

SIZE = size(power1);
x = 1:SIZE(2);

figure(13)
plot(x,power1)
axis([1,SIZE(2),0,0])
xlabel('EIM Interval')
ylabel('Power (MW)')
hold on
plot(x,power15)
plot(x,power2)
plot(x,power3)
plot(x,power4)
plot(x,power5)
plot(x,power6)
plot(x,power7)
hold off
max = zeros(1,288);

x = 0;
for m = 1:288
  for n = .5:.1:50
    f = 0;
    [X,f] = SimDec10(6,6,2,0,5*m,n,ETAKE,draws);
    if(f>0)
      break
    else
      max(1,m)=n;
      %continue
    end
  end
end
% Battery Initialization

tic
% RegA data from PJM
RegA = csvread('RegA.csv');
SIZE = size(RegA);
Days = zeros(1,SIZE(2));
tic
posCount = 0;
egCount = 0;
for m = 1:SIZE(2)
    sum = 0;
    for n = 1:SIZE(1)
        sum = sum + RegA(n,m);
    end
    Days(m) = sum;
    if (sum >= 0)
        posCount = posCount + 1;
    else
        negCount = negCount + 1;
    end
end
tic
% Inverter Data, Outback 6046A
VDC_IN = 48; % VDC
P = 8000; % VA, continuous. Seems like max power is actually 24000 VA.
VAC_OUT = 240; % VAC or 120
FREQ_OUT = 60; % Hz
IDLE_LOSS = 34; % W
EFF_IN = 0.93;
EFF_OUT = 0.93;
IAC_MAX = 100; % AAC to grid. OR 200 if 120V.
BATT_OUT = 115; % ADC to battery, continuous.
% Randomizing battery energy.
energy_min = 0; % This is just the shut off
energy_max = zeros(10000,1); % Array of max energies.
for k = 1:10000 % Randomly assign the designed W-hrs of batteries.
    if (STORAGE <= 0.20)
        emax = 2*P; % 2-hr battery. W-hrs.
    elseif(STORAGE <= 0.40)
        emax = 4*P; % 4-hr battery.
    elseif(STORAGE <= 0.60)
        emax = 6*P; % 6-hr battery.
    elseif(STORAGE <= 0.80)
else
    emax = 10^5P; % 10-hour battery.
end

LOSS = rand() * 0.10000000; % Batteries lose capacity as they age.
% Up to 20%, but 0.10000000 makes a later
% calculation simpler.

emax = emax - emax*LOSS; % Randomly remove up to 20% of capacity.
energy_max(k,1) = emax;
end

% Battery operating model. These numbers are fractions of energy_max.
MaxE = 1; % Stop responding to RegA commands.
HiE = 0.9; % Discharge when convenient.
LoE = .1; % Recharge when convenient chance.
MinE = 0; % Stop RegA commands when at this point.
ReE = 0.3; % Voluntarily recharging, continue until 60% of energy_max.
DisE = 0.85; % Voluntarily discharging, continue until 85% of energy_max.
MRes = 0.2; % Mandatory recharging, continue until 20% of energy_max.
MDisE = 0.9; % Mandatory discharging, continue until 90% of energy_max.
toc

tic

batts = zeros(365*1000*24, 6); % All data for a single battery over a year.
% Scoring data for an individual battery. The battery will follow RegA for
% the entire year, and update it’s energy every 2 seconds.
% batts stores this data as an array with rows representing each two
% second interval and 6 columns. Column 1 = UnitID, Column2 = 2-sec
% interval, Column3 = 5-minute interval, Column4 = RegA command, Column5 =
% Energy of Battery, Column6 = Status of Battery.

% Battery Stauses
% 11 = following regA instructions.
% 12 = following regA instructions, but looking for good time to charge.
% 13 = waiting for good time to charge.
% 14 = following regA instructions, but looking for time to discharge.
% 15 = waiting for a good time to discharge.

batts = zeros(10000, 7+(2*288*365)); % Summarized data for all batteries
% in 5-min intervals.
% To reduce memory and storage needs, I simulate every 2 seconds, but only
% every 5 minutes. The batts array is generated for each individual battery
% and then overwritten. The batts array stores the energy and status of
% the batteries at each 5 minute interval. This data is stored in an array
% similar to that of the EWH, where each row is a different battery and the
% first 7 columns store identifying information on the battery unit itself.
% Column1 = UnitID, Column2 = Nominal Capacity, Column3 = empty, Column4 =
% Minimum Energy Capacity, Column5 = Maximum Energy Capacity, Column6 =
% PowerMinimum, and Column7 = PowerMax. The initial energy levels of the
% batteries are randomized and stored in column 8, while the initial
% statuses are stored in column 9. Higher columns show energy takes and
% statuses at higher 5-minute EIM intervals.

too

% Status
% 11 = following regA instructions.
% 12 = following regA instructions, but looking for good time to charge.
% 13 = waiting for good time to charge.
% 14 = following regA instructions, but looking for time to discharge.
% 15 = waiting for a good time to discharge.
% miscels = 0;
startEx = 23/48; % Expensive power starts at 11:30 am.
endEx = 43/48; % Expensive power ends at 9:30 pm.
tic
for k=1:10000  % Count through all 10,000 batteries.
   entryctr = 1; % Counts through the individual 2 sec intervals in RegA.
   intctr = 1; % Counts through the individual 5 min intervals for EIM.
   % Initializing first energy take and status randomly.
   batts(entryctr,1) = k; % Unit number
   batts(entryctr,2) = 1; % 2-second interval number (RegA)
   batts(entryctr,3) = 1; % 5-min interval number (EIM)
   batts(entryctr,4) = RegA(i,1); % Signal for RegA interval 1.
   batts(entryctr,5) = rand()*energy_max(k,1); % Randomized initial energy
      % based max energy.
   % This status assigning assumes that the battery was not already in
   % recharge mode or discharge mode.
   if  (batts(entryctr,5)) == (energy_max(k,1) ' MinE)  
      batts(entryctr,6) = 13; % 13 = charging or idle.
   elseif  (batts(entryctr,5)) <= (energy_max(k,1) ' LoE)  
      batts(entryctr,6) = 12; % 12 = charging or following regA.
   elseif  (batts(entryctr,5)) < (energy_max(k,1) ' HiE)  
      batts(entryctr,6) = 11; % 11 = following regA instructions.
   elseif  (batts(entryctr,5)) < (energy_max(k,1) ' MaxE)  
      batts(entryctr,6) = 14; % 14 = discharging or following RegA
   else
      batts(entryctr,6) = 15; % 15 = discharging or idle.
   end

batts(k,1) = k; % Unit ID
if(energy_max(k,1) > 0*P)  % Assigning nominal capacity.
   batssum(k,2) = 10;
elseif(energy_max(k,1) > 6*P)
   batssum(k,2) = 8;
elseif(energy_max(k,1) > 4*P)
   batssum(k,2) = 6;
elseif(energy_max(k,1) > 2*P)
battsum(k, 2) = 4;
else
  battsum(k, 2) = 2;
end

battsum(k, 4) = energy_max(k, 1) * MinE; % Assigning minimum capacity.
battsum(k, 5) = energy_max(k, 1) * MaxE; % Assigning maximum capacity.
battsum(k, 6) = -P; % Minimum power.
battsum(k, 7) = P; % Maximum power.
battsum(k, 8) = batts(entryctr, 5); % Energy at first 2-sec interval.
battsum(k, 9) = batts(entryctr, 6); % Status at first 2-sec interval.

% Loop, what am I using
% int counter, counts the intervals, updates every day (or m)
% entry counter, counts the individual entries (updates every EWH or k)
% m runs from 1-365
% n runs from 1-43200
% When m and n are 1, that's the first data set, I already set this.
% so increment the entry counter and continue.
%
for m=1:SIZE(2)
  intctr = 1;
  for n=1:SIZE(1)
    % Ignore the first data set, that one is already set.
    if ((n == 1) && (m == 1))
      entryctr = entryctr + 1;
      continue
    end
    % Non-Simulated Variables.
    batts(entryctr, 1) = k;
    batts(entryctr, 2) = n + (m-1)*43200;
    if (entryctr == batts(entryctr, 2))
      miscalos = miscalos+1;
    end
    batts(entryctr, 3) = intctr;
    batts(entryctr, 4) = RegA(n, m);
    % Simulated Variables.
    Ereg = batts(entryctr-1, 5) + batts(entryctr, 4)*P*(1/1800);
    Echa = batts(entryctr-1, 5) + P * (1/1800);
    Edis = batts(entryctr-1, 5) - P * (1/1800);
    Eidle = batts(entryctr-1, 5);
    % Status 11 (Reg B)
    if (batts(entryctr - 1, 6) == 11)
      if (Ereg <= MinE*battsum(k, 5))
        batts(entryctr, 5) = battsum(k, 4); % Min Energy
batts(entryctr,6) = 13;
else if (Ereg <= LoE*battsum(k,5))
    batts(entryctr,5) = Ereg;
batts(entryctr,6) = 12;
else if (Ereg < HiE*battsum(k,5))
    batts(entryctr,5) = Ereg;
batts(entryctr,6) = 11;
else if (Ereg < MaxE*battsum(k,5))
    batts(entryctr,5) = Ereg;
batts(entryctr,6) = 14;
else
    batts(entryctr,5) = battsum(k,5); % Max Energy
    batts(entryctr,6) = 15;
end

% Status 12 (Prefer to charge, else RegA)
elseif (batts(entryctr - 1,6) == 12)
    if ((n > (SIZE(l)*startEx+1)) && (n < (SIZE(l)*endEx+1)))
        % If power is expensive-ish, then RegA.
        if (Ereg <= MinE*battsum(k,5))
            batts(entryctr,5) = battsum(k,4); % Min Energy
            batts(entryctr,6) = 13;
        elseif (Ereg < ReE*battsum(k,5))
            batts(entryctr,5) = Ereg;
batts(entryctr,6) = 12;
        elseif (Ereg < HiE*battsum(k,5))
            batts(entryctr,5) = Ereg;
batts(entryctr,6) = 11;
        elseif (Ereg < MaxE*battsum(k,5))
            batts(entryctr,5) = Ereg;
batts(entryctr,6) = 14;
        else
            batts(entryctr,5) = battsum(k,5); % Max Energy
            batts(entryctr,6) = 15;
        end
    else % else power is inexpensive-ish, so charge.
        %E = batts(entryctr-1,5) + P * (1/1800);
        if (Echa <= ReE*battsum(k,5))
            batts(entryctr,5) = Echa;
batts(entryctr,6) = 12;
        elseif (Echa < HiE*battsum(k,5))
            batts(entryctr,5) = Echa;
batts(entryctr,6) = 11;
        elseif (Echa < MaxE*battsum(k,5))
            batts(entryctr,5) = Echa;
batts(entryctr,6) = 14;
        else
            batts(entryctr,5) = battsum(k,5); % Max Energy
            batts(entryctr,6) = 15;
        end
    end
end
% Status 13 (Must charge or idle)
else if (batts(entryctr - 1, 6) == 13)
    if ((n > (SIZE(1)*startEx+1)) && (n < (SIZE(1)*endEx+1)))
% If power is expensive-ish, then idle.
        batts(entryctr, 5) = Eidle;
        batts(entryctr, 6) = 13;
    else % Power is cheap-ish, so charge up.
        if (Echa <= MReE*battsum(k, 5))
            batts(entryctr, 5) = Echa;
            batts(entryctr, 6) = 13;
        elseif (Echa <= ReE*battsum(k, 5))
            batts(entryctr, 5) = Echa;
            batts(entryctr, 6) = 12;
        elseif (Echa < HiE*battsum(k, 5))
            batts(entryctr, 5) = Echa;
            batts(entryctr, 6) = 11;
        elseif (Echa < MaxE*battsum(k, 5))
            batts(entryctr, 5) = Echa;
            batts(entryctr, 6) = 14;
        else
            batts(entryctr, 5) = battsum(k, 5); % Max Energy
            batts(entryctr, 6) = 15;
        end
    end
end

% Status 14 ( Prefer to discharge, else RegA)
else if (batts(entryctr - 1, 6) == 14)
    if ((n > (SIZE(1)*startEx+1)) && (n < (SIZE(1)*endEx+1)))
% If power is expensive-ish, then discharge.
        if (Edis < MEdE*battsum(k, 5))
            batts(entryctr, 5) = battsum(k, 4);
            batts(entryctr, 6) = 13;
        elseif (Edis <= LEdE*battsum(k, 5))
            batts(entryctr, 5) = Edis;
            batts(entryctr, 6) = 12;
        elseif (Edis <= DisE*battsum(k, 5))
            batts(entryctr, 5) = Edis;
            batts(entryctr, 6) = 11;
        else
            batts(entryctr, 5) = Edis;
            batts(entryctr, 6) = 14;
        end
    else % Power is cheap-ish, so follow RegA.
        if (Ereg <= MinE*battsum(k, 5))
            batts(entryctr, 5) = battsum(k, 4);
            batts(entryctr, 6) = 13;
        elseif (Ereg <= LorE*battsum(k, 5))
            batts(entryctr, 5) = Ereg;
            batts(entryctr, 6) = 12;
        elseif (Ereg < DIsE*battsum(k, 5))
            batts(entryctr, 5) = Ereg;
batts(entryctr, 5) = Ereg;
batts(entryctr, 6) = 11;
else if (Ereg < MaxE * battsumm(k, 5))
batts(entryctr, 5) = Ereg;
batts(entryctr, 6) = 14;
else
  batts(entryctr, 5) = battsumm(k, 5);
batts(entryctr, 6) = 15;
end
end

% Status 15 (Must discharge or idle)
else if (batts(entryctr - 1, 6) == 15)

  if ((n > (SIZE(1) * startEx + 1)) && (n < (SIZE(1) * endEx + 1)))
    % If power is expensive-ish, then discharge.
    if (Edis < MinE * battsumm(k, 5))
      batts(entryctr, 5) = battsum(k, 4);
batts(entryctr, 6) = 13;
    elseif (Edis <= LcE * battsumm(k, 5))
      batts(entryctr, 5) = Edis;
batts(entryctr, 6) = 12;
    elseif (Edis <= DisE * battsumm(k, 5))
      batts(entryctr, 5) = Edis;
batts(entryctr, 6) = 11;
    elseif (Edis <= MIDisE * battsumm(k, 5))
      batts(entryctr, 5) = Edis;
batts(entryctr, 6) = 14;
    else
      batts(entryctr, 5) = Edis;
batts(entryctr, 6) = 15;
    end
  else % Else power is inexpensive, so idle.
    batts(entryctr, 5) = Eidle;
batts(entryctr, 6) = 15;
  end

% Unknown battery state.
else
  disp('Error, unknown battery state')
batts(entryctr - 1, 6) = 10;
break
end

% End of loop housekeeping.
if (mod(n, 150) == 0)
  intctr = intctr + 1;
  % battsumm(k, (7 + intctr * 2 + 288 * (m - 1))) = batts(entryctr, 5);
  % battsumm(k, (8 + intctr * 2 + 288 * (m - 1))) = batts(entryctr, 6);
end
entryctr = entryctr + 1;
end
end
% Now I should have a bats array that I can move into the battsumm
% array with summary values.
SIZE2 = size(batts);
colctr=0;
for m = 2:SIZE2(1)
    if(batts(m-1,3) == batts(m,3))
        battsumm(k, colctr+10) = batts(m,5);
        battsumm(k, colctr+11) = batts(m,6);
        colctr=colctr+2;
    end
end
end

tic clear BATT_OUT bats colctr Days DisE E Echa Edis EFF_IN EFF_OUT Eidle emax
clear endEx energy_max energy_min entryctr Freq FREQ_OUT HiE IAC_MAX
clear IDLE_LOSS intctr k LoE LOSS m MaxE MDisE MinE miscalcs MReE n
clear negCount P posCount ReE RegA SIZE SIZE2 startEx STORAGE sum VAC_OUT
clear VDC_IN
toc
function [X,f,power] = BattBid4(month, day, hour, min, duration, MW, battsumm)

% Describing RegA signal %
RegA = csvread('RegA.csv');
% Determine which column of RegA array to use.
monthAdd = 0; % Figure out how many columns to add onto the day.
  if (months start at 1)
    if (month == 2)
        monthAdd = 31;
    elseif (month == 3)
        monthAdd = (31+28);
    elseif (month == 4)
        monthAdd = (31+28+31);
    elseif (month == 5)
        monthAdd = (31+28+31+30);
    elseif (month == 6)
        monthAdd = (31+28+31+30+31);
    elseif (month == 7)
        monthAdd = (31+28+31+30+31+30);
    elseif (month == 8)
        monthAdd = (31+28+31+30+31+30+31);
    elseif (month == 9)
        monthAdd = (31+28+31+30+31+30+31+31);
    elseif (month == 10)
        monthAdd = (31+28+31+30+31+30+31+31+30);
    elseif (month == 11)
        monthAdd = (31+28+31+30+31+30+31+31+30+31);
    elseif (month == 12)
        monthAdd = (31+28+31+30+31+30+31+31+30+31+30);
  end

  % Now adjust for full days (days start at 1).
  column = day + monthAdd; % Column Determined.
  % Trim RegA to only the data on current day and next day.
  RegA = [RegA(:,column), RegA(:,column+1)];

% Important interval numbers.
bidStartInt = time2interval(month, day, hour, min);
dayStartInt = day2interval(month, day);
durInt = duration / 5;
finalInt = dayStartInt + 287 + 288;
delayInt = bidStartInt - dayStartInt;

% Matrix Sizing.
SIZE1 = size(battsumm);
SIZE2 = size(RegA);
X = zeros(SIZE1(1),7+288*2*2); % 2 Days and 2 Entries (Status and Energy).
    % 7 is for the initial data in ETAKE array.
if (mod(duration, 5) ~= 0)
  disp('Error. Invalid duration. Duration must be a multiple of 5.')
  disp('EMT markets run every 5 minutes')
  return
end

% Copying the preliminary data in stored in the first 7 columns.
for n = 1:SIZE(1)
  X(n,1) = battsumm(n,1);
  X(n,2) = battsumm(n,2);
  X(n,3) = battsumm(n,3);
  X(n,4) = battsumm(n,4);
  X(n,5) = battsumm(n,5);
  X(n,6) = battsumm(n,6);
  X(n,7) = battsumm(n,7);
end

% Copying data that occurs before the bid, and needs no resimulation.
if (delayInt > 0)
  %counter = 1;
  for m = 1:delayInt:(6+2*dayStartInt):(6+2*dayStartInt+delayInt)
    for n = 1:SIZE(1)
      X(n,6+2*m) = battsumm(n,6+2*(dayStartInt)+2*m);
      X(n,7+2*m) = battsumm(n,5+2*(dayStartInt)+2*m);
    end
  end
end

% Sorting rows by energy in the interval before the bid.
% y = 6+2*(delayInt)
sorted = sortrows(X,6+2*delayInt,'ascend');
X = sorted;
%f = RegA;

% Assigning units to cover the bid in initial time interval.
W = MW * 10^6; % Total Power owed across each interval.
Wthr = MW * 10^6 * (duration/60); % Total watt hours owed by our bid.
intWthr = MW * (10^6) * 1/12; % Energy in W-hrs per interval owed.

intAcct = zeros(1,durInt);% This tracks the owed energy across each
for n = 1:durInt % interval.
  intAcct(n) = intWthr;
end

% Battery operating model. These numbers are fractions of energy_max.
MaxE = 1; % Stop responding to RegA commands.
% Discharge when convenient.
LoE = .1;   % Recharge when convenient chance.
MinE = 0;   % Stop RegA commands when at this point.
ReE = 0.6;  % Voluntarily recharging, continue until 80% of energy_max.
DisE = 0.85; % Voluntarily discharging, continue until 55% of energy_max.
MReE = 0.2;  % Mandatory recharging, continue until 20% of energy_max.
MDisE = 0.9; % Mandatory discharging, continue until 90% of energy_max.

% batt = [RegARow, RegAColumn, Energy, Status, RegA]
batt = zeros(5,151);
MaxIndex = size(intAgt);
MaxIndex = MaxIndex(2);
AcctIndex = 1;
for m = (6 + 2*(delayInt+1));2:(6 + 2*(delayInt+durInt)) % From starting
   interval of bid, to finish interval of bid.

   % Preparing the next interval.
   if(AcctIndex > MaxIndex) % If we are past the last interval then stop.
      disp('Possible Error: Stopped due to intAgt & MaxIndex Exceeded')
      sorted = sortrows(X,1,'descend');
      X = sorted;
      break
   else % Else assign units to the next interval.
      sorted = sortrows(X,m,'ascend');
      X = sorted;
   end

   % Skip if the unit was already on.
   for n = 1:SIZEI(1)
      % First check time price of power.
      % if expensive
      if ( ( (hour > li1) || (hour == li1) && (min >= 30) ) && (hour < li2) )
         % what a deal, dec'ing when power is cheap. This means
         % the battery charges for free.
         else % power is cheap
            if (X(n,m) == li2)
               continue % Ignore the ones that would be charging anyway.
               % As these might be expected load.
            elseif (X(n,m) == li3)
               continue % same story.
            end
         end
   end

   % CtrlON mode = 16. This means we have assigned the batt to dec.
   X(n,m-1) = 16; % Set to CtrlDec Mode.
   Etotal = X(n,5) - X(n,m-2); % Total Energy Take Available.
   % is full capacity - actual charge.
EMax = X(n,7) * 1/12; % Maximum Energy in W-hrs that can be
% consumed during next time period.

% Account for energy removal.
if (EMax < ETotal)
    intAcct(AcctIndex) = intAcct(AcctIndex) - EMax;
else
    intAcct(AcctIndex) = intAcct(AcctIndex) - ETotal;
end

% Not checking to see if the unit would have ordinarily come on.
% That is more an ENH thing.
if (intAcct(AcctIndex) < 0)
    unitCtr: %display number of units I turn on for now
        break
end

% First Simulate Current Interval
for n = 1:SIZE1(I) % From first unit to last unit (sorted by
    % energy, with lowest being on top).
    if ( (X(n,m-1) < 16) && (X(n,m-1) > 10) )
        % if a normal, uncontrolled battery state, resimulate.
        battmax = X(n,5);
        E = X(n,m-2);
        S = X(n,m-1);
        currInt = (m-6)/2;
    if (currInt < 289)
        RegCol = 1;
        RegRow = round( ((currInt - 1)/288)*SIZE2(1) + 1 );
    else
        RegCol = 2;
        RegRow = round( (((currInt - 1)-289)/288)*SIZE2(1) + 1 );
    end
    if ( (currInt > 136) && (currInt < 259) )
        TOU = 1; %Expensive Power
    elseif( (currInt > (136+200)) && (currInt < 259+200) )
        TOU = 1;
    else
        TOU = 0; % Cheap power.
    end
    batt(1,1) = RegRow;
    batt(1,2) = RegCol;
end
batt(1,3) = E;
batt(1,4) = S;
batt(1,5) = RegA(RegRow,RegCol);

% Simulating the power for regular units.
for k = 1:150

    % Debug line, If m = 540, n = 9998, and k = 149, pause.
    if(m==540 && n==9998 && k==149)
        % bbb = 1;
        %end

% Simulated Variables. % Looking at k, writing to k+1.
Ereg = batt(k,3) + RegA(RegRow,RegCol)*X(n,7)*(1/1800);
EchA = batt(k,3) + X(n,7) * (1/1800);
Edis = batt(k,3) - X(n,7) * (1/1800);
Eidle = batt(k,3);
Emin = X(n,4);
Emax = X(n,5);

if(batt(k,4) == 11)
    if(Ereg <= battmax’MinE)
        batt(k+1,3) = Emin;
batt(k+1,4) = 13;
    elseif(Ereg <= battmax’LoE)
        batt(k+1,3) = Ereg;
batt(k+1,4) = 12;
    elseif(Ereg <= battmax’HiE)
        batt(k+1,3) = Ereg;
batt(k+1,4) = 11;
    elseif(Ereg <= battmax’MaxE)
        batt(k+1,3) = Ereg;
batt(k+1,4) = 14;
    else
        batt(k+1,3) = Emax;
batt(k+1,4) = 15;
    end

elseif(batt(k,4) == 12)
    if(100) % Power is expensive, so RegA
        if(Ereg <= battmax’MinE)
            batt(k-1,3) = Emin;
batt(k-1,4) = 13;
        elseif(Ereg <= battmax’ReE)
            batt(k-1,3) = Ereg;
batt(k-1,4) = 12;
        elseif(Ereg <= battmax’HiE)
            batt(k-1,3) = Ereg;
batt(k-1,4) = 11;
        elseif(Ereg <= battmax’MaxE)
            batt(k-1,3) = Ereg;
batt(k-1,4) = 14;
        else
            batt(k+1,3) = Emax;
batt(k+1,4) = 15;
        end
    end
batt(k+1,3) = Ereg;
batt(k+1,4) = 14;

else
    batt(k+1,3) = Emax;
batt(k+1,4) = 15;
end

else % Power is cheap, so charge.
    if(Echa <= battmax*ReE)
        batt(k+1,3) = Echa;
batt(k+1,4) = 12;
    elseif(Echa <= battmax*HiE)
        batt(k+1,3) = Echa;
batt(k+1,4) = 11;
    elseif(Echa <= battmax*MaxE)
        batt(k+1,3) = Echa;
batt(k+1,4) = 14;
    else
        batt(k+1,3) = Emax;
batt(k+1,4) = 15;
    end
end

elseif(batt(k,4) == 13)
    if(TOU) % Power is expensive, so Idle
        batt(k+1,3) = Eidle;
batt(k+1,4) = 13;
    elseif(TOU) % Power is cheap, so charge.
        if(Echa <= battmax*MRReE)
            batt(k+1,3) = Echa;
batt(k+1,4) = 13;
        elseif(Echa <= battmax*ReE)
            batt(k+1,3) = Echa;
batt(k+1,4) = 12;
        elseif(Echa <= battmax*HiE)
            batt(k+1,3) = Echa;
batt(k+1,4) = 11;
        elseif(Echa <= battmax*MaxE)
            batt(k+1,3) = Echa;
batt(k+1,4) = 14;
        else
            batt(k+1,3) = Emax;
batt(k+1,4) = 15;
        end
    end

elseif(batt(k,4) == 14)
    if(TOU) % Power is expensive, so Discharge
        if(Edis <= battmax*MinE)
            batt(k+1,3) = Emin;
batt(k+1,4) = 13;
        elseif(Edis <= battmax*LoE)
batt(k+1,3) = Edis;
batt(k+1,4) = 12;
elseif(Edis <= battmax*DisE)
batt(k+1,3) = Edis;
batt(k+1,4) = 11;
else
batt(k+1,3) = Edis;
batt(k+1,4) = 14;
end
else % Power is cheap, so RegA.
if(Ereg <= battmax*MinE)
batt(k+1,3) = Ereg;
batt(k+1,4) = 13;
elseif(Ereg <= battmax*LoE)
batt(k+1,3) = Ereg;
batt(k+1,4) = 12;
elseif(Ereg <= battmax*DisE)
batt(k+1,3) = Ereg;
batt(k+1,4) = 11;
elseif(Ereg <= battmax*MaxE)
batt(k+1,3) = Ereg;
batt(k+1,4) = 14;
else
batt(k+1,3) = Emax;
batt(k+1,4) = 15;
end
end
elseif(batt(k,4) == 15)
if(TOU) % Power is expensive, so Discharge
if(Edis <= battmax*MinE)
batt(k+1,3) = Emin;
batt(k+1,4) = 13;
elseif(Edis <= battmax*LoE)
batt(k+1,3) = Edis;
batt(k+1,4) = 12;
elseif(Edis <= battmax*ReE)
batt(k+1,3) = Edis;
batt(k+1,4) = 11;
elseif(Edis <= battmax*MeE)
batt(k+1,3) = Edis;
batt(k+1,4) = 14;
else
batt(k+1,3) = Edis;
batt(k+1,4) = 15;
end
else % Power is cheap, so Idle.
batt(k+1,3) = Eidle;
batt(k+1,4) = 15;
end
end
end
X(n,m) = batt(151,3);
X(n,m+1) = batt(151,4);

elseif (X(n,n-1) >= 16)
  if (X(n,n-1) == 16) \% Charge per DERAS command.
    Echa = X(n,n-2) + X(n,7)^*(1/12);
    Enax = X(n,5);
    if (Echa >= Enax)
      X(n,m) = X(n,5);
      X(n,m+1) = 15;
    elseif (Echa <= 0)
      X(n,m) = X(n,4);
      X(n,m+1) = 13;
    else
      X(n,m) = Echa;
      X(n,m+1) = 11; \% (most flexible, it can figure out its
                     \% best status 2 seconds later.
  end

\% Leaving a little room for a status 17, DERAS Increment.

else \% Some error occurred.
  disp('Error, unknown or unexpected status (inner branch)')
  X(n,m-1)
  m
  n
end

else \% Probably got an EWH mixed in here, or status = 0 error.
  disp('Error, unknown or unexpected status (footer branch)')
  X(n,m-1)
  m
  n
end

end
AcctIndex = AcctIndex + 1; % Increment the account index.

end

\% \% \% I think it's good below here. \% \% \% \% \% \%

\% Now simulate from current interval to last interval (2 full days of
\% intervals are simulated across this entire script).
batt = zeros(5,151);
ctest = 0;
for m=(6 + 2*(delayInt+durInt+1)):2:(6+2*2*288) % current int to last.

    for n = 1:SIZE1(1) % From unit 1 to unit final.
        battmax = X(n,5);
        E = X(n,m-2);
        S = X(n,m-1);
        currInt = (m-6)/2;
        if (currInt < 209)
            RegCol = 1;
            RegRow = round( ((currInt - 1)/288)*SIZE2(1) + 1 , 0);
        else
            RegCol = 2;
            RegRow = round( ((currInt - 1)/288)*SIZE2(1) + 1 );
        end
        if ( (currInt > 138) && (currInt < 259) )
            TOU = 1; %Expensive Power
        elseif( (currInt > (138+288)) && (currInt < 259+288) )
            TOU = 1;
        else
            TOU = 0; % Cheap power.
        end
        batt(1,1) = RegRow;
        batt(1,2) = RegCol;
        batt(1,3) = E;
        batt(1,4) = S;
        cttst = cttst + 1;
        batt(1,5) = RegA(RegRow,RegCol);
    % Simulating the power for regular units.
    for k = 1:150
        % Debug Line, If m = 540, n = 9999, and k = 146, pause.
        if(m==540 && n==9999 && k==146)
            bbb=1;
            %end
        
        % Simulated Variables. % Looking at k, writing to k+1.
        Ereg = batt(k,3) + RegA(RegRow,RegCol)*X(n,7)*(1/1800);
        Echa = batt(k,3) + X(n,7) * (1/1800);
        Edis = batt(k,3) - X(n,7) * (1/1800);
        Eidle = batt(k,3);
        Emin = X(n,4);
        Emax = X(n,5);
        if(batt(k,4) == 11)
            if(Ereg <= battmax*MinE)
                batt(k+1,3) = Emin;
            end
        end
    end
end
batt(k+1,4) = 13;
elseif(Ereg <= battmax*LoE)
    batt(k+1,3) = Ereg;
batt(k+1,4) = 12;
elseif(Ereg <= battmax*HiE)
    batt(k+1,3) = Ereg;
batt(k+1,4) = 11;
elseif(Ereg <= battmax*MaxE)
    batt(k+1,3) = Ereg;
batt(k+1,4) = 14;
else
    batt(k+1,3) = Emax;
batt(k+1,4) = 15;
end

elseif(batt(k,4) == 12)
    if(TOU) & Power is expensive, so RegA
        if(Ereg <= battmax*MinE)
            batt(k+1,3) = Emin;
batt(k+1,4) = 13;
        elseif(Ereg <= battmax*ReE)
            batt(k+1,3) = Ereg;
batt(k+1,4) = 12;
        elseif(Ereg <= battmax*HiE)
            batt(k+1,3) = Ereg;
batt(k+1,4) = 11;
        elseif(Ereg <= battmax*MaxE)
            batt(k+1,3) = Ereg;
batt(k+1,4) = 14;
        else
            batt(k+1,3) = Emax;
batt(k+1,4) = 15;
        end
    else % Power is cheap, so charge.
        if(Echa <= battmax*ReE)
            batt(k+1,3) = Echa;
batt(k+1,4) = 12;
        elseif(Echa <= battmax*HiE)
            batt(k+1,3) = Echa;
batt(k+1,4) = 11;
        elseif(Echa <= battmax*MaxE)
            batt(k+1,3) = Echa;
batt(k+1,4) = 14;
        else
            batt(k+1,3) = Emax;
batt(k+1,4) = 15;
        end
    end
end

elseif(batt(k,4) == 13)
    if(TOU) & Power is expensive, so Idle
batt(k+1,3) = Eidle;
batt(k+1,4) = 13;
else % Power is cheap, so charge.
  if(Echa <= battmax'MReE)
    batt(k+1,3) = Echa;
batt(k+1,4) = 13;
  elseif(Echa <= battmax'ReE)
    batt(k+1,3) = Echa;
batt(k+1,4) = 12;
  elseif(Echa <= battmax'HiE)
    batt(k+1,3) = Echa;
batt(k+1,4) = 11;
  elseif(Echa <= battmax'MaxE)
    batt(k+1,3) = Echa;
batt(k+1,4) = 14;
else
  batt(k+1,3) = Emax;
batt(k+1,4) = 15;
end
end

elseif(batt(k,4) == 14)
  if(TOU) % Power is expensive, so Discharge
    if(Edis <= battmax'MinE)
      batt(k+1,3) = Emin;
batt(k+1,4) = 13;
    elseif(Edis <= battmax'LoE)
      batt(k+1,3) = Edis;
batt(k+1,4) = 12;
    elseif(Edis <= battmax'DisE)
      batt(k+1,3) = Edis;
batt(k+1,4) = 11;
    else
      batt(k+1,3) = Edis;
batt(k+1,4) = 14;
    end
  else % Power is cheap, so RegA.
    if(Ereg <= battmax'MinE)
      batt(k+1,3) = Ereg;
batt(k+1,4) = 13;
    elseif(Ereg <= battmax'LoE)
      batt(k+1,3) = Ereg;
batt(k+1,4) = 12;
    elseif(Ereg <= battmax'DisE)
      batt(k+1,3) = Ereg;
batt(k+1,4) = 11;
    elseif(Ereg <= battmax'MaxE)
      batt(k+1,3) = Ereg;
batt(k+1,4) = 14;
    else
      batt(k+1,3) = Emax;
  end
end
batt(k+1,4) = 15;

end

end

elseif(batt(k,4) == 15)
    if(TOU) % Power is expensive, so Discharge
        if(Edis <= battmax'MinE)
            batt(k+1,3) = EMIN;
            batt(k+1,4) = 13;
        elseif(Edis <= battmax'LoE)
            batt(k+1,3) = Edis;
            batt(k+1,4) = 12;
        elseif(Edis <= battmax'ReE)
            batt(k+1,3) = Edis;
            batt(k+1,4) = 11;
        elseif(Edis <= battmax'MReE)
            batt(k+1,3) = Edis;
            batt(k+1,4) = 10;
        else
            batt(k+1,3) = Edis;
            batt(k+1,4) = 15;
        end
    else % Power is cheap, so Idle.
        batt(k+1,3) = Edile;
        batt(k+1,4) = 15;
    end

else % Bad status somewhere.
    disp('Error, unknown or unexpected status (after-dec)')
    X(n,m-1)
    m
    n
end % End decision branch
end % End 5-minute interval loop.
X(n,m) = batt(i51,3);
X(n,m+1) = batt(i51,4);
end

%intAcct
%intAcct
SIZE = size(intAcct);
power = zeros(1,SIZE(2));
failott = 0;
for n=1:SIZE(2)
    intAcct(n):
        power(n) = (-intAcct(n))/(10^6) + MN/12; % Technically counts energy.
        if (intAcct(n) < 0)
failctr = failctr+1;
end
end
power = 12 .* power; % Now, I multiply the energy in Watt-hrs by 12/per hr.
% to get the power.
f=failctr;
end % End function.
function [Energy] = BattPlotter3(mo,day,battsumm)

% Purpose: Plots.

% Inputs
% Date in the form of month and day, and ETARE matrix and draws matrix.

% Outputs
% Plot of energy usage by interval for each 5 minute interval.

% RegA = csvread('RegA.csv');

SIZE = size(battsumm);

interval = day2interval(mo,day);
Energy = zeros(1,288);
RegON = zeros(1,288);
CharON = zeros(1,288);
DisON = zeros(1,288);
Idle = zeros(1,288);
Checksum = zeros(1,288);
excctr = 0;
dailyInt = 0;
Max = zeros(1,200);
S1AT11 = zeros(1,200);
S1AT12 = zeros(1,288);
S1AT13 = zeros(1,288);
S1AT14 = zeros(1,288);
S1AT15 = zeros(1,288);
Checksum2 = zeros(1,288);
%Min = zeros(1,288);
counter = 1;
for n=interval:(interval+287)
    E = 0;
    R = 0;
    C = 0;
    D = 0;
    I = 0;
    M = 0;
    S11 = 0;
    S12 = 0;
    S13 = 0;
    S14 = 0;
    S15 = 0;
    dailyInt = dailyInt + 1;
    for n=1:SIZE(1)
        E = E + battsumm(n,6+2*m);
        M = M + battsumm(n,5);
        if(battsumm(n,7+2*m) == 11) % RegA
            % Continue with the rest of the code
        end
    end
end
R = R + 1;
S11 = S11 + 1;
end
if(battsum(n,7+2*m) == 12) % Charge or RegA
    S12 = S12 + 1;
    if ( (dailyInt < 259) || (dailyInt > 139) ) % If power !cheap.
        R = R + 1; % So RegA.
    else % Power is cheap, so charging.
        C = C + 1;
    end
end
if(battsum(n,7+2*m) == 13) % Charge or Idle.
    S13 = S13 + 1;
    if ( (dailyInt < 259) && (dailyInt > 139) ) % If power !cheap.
        I = I + 1; % So Idle.
    else % Power is cheap, so charging.
        C = C + 1;
    end
end
if(battsum(n,7+2*m) == 14) % Discharge or RegA.
    S14 = S14 + 1;
    if ( (dailyInt < 259) && (dailyInt > 139) ) % If power !cheap.
        D = D + 1; % So discharge.
    else % Power is cheap, so RegA.
        R = R + 1;
    end
end
if(battsum(n,7+2*m) == 15) % Discharge or Idle.
    S15 = S15 + 1;
    if ( (dailyInt < 259) && (dailyInt > 139) ) % If power !cheap.
        D = D + 1; % So discharge.
    else % Power is cheap, so RegA.
        I = I + 1;
    end
end
Energy(counter) = E;
RegON(counter) = R;
ChaON(counter) = C;
DisON(counter) = D;
Idle(counter) = I;
Max(counter) = M;
Checksum(counter) = R+C+D+I;
STAT11(counter) = S11;
STAT12(counter) = S12;
STAT13(counter) = S13;
STAT14(counter) = S14;
STAT15(counter) = S15;
Checksum2(counter) = S11 + S12 + S13 + S14 + S15;
counter = counter + 1;
end
\[
\text{Avg} = (\text{Max} - \text{Min}) / 2 + \text{Min};
\]

\[x = 1:288;\]

```matlab
figure(11)
plot(x.*(1/12),Energy.*(1/1000000))
xlabel('Hour')
ylabel('Energy (MW-hr)')
axis([0 24 0 500])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24])
hold on
plot(x.*(1/12),Max.*(1/1000000))
%plot(x.*(1/12),Min.*(1/1000000))
%plot(x.*(1/12),Avg.*(1/1000000))
hold off
```

```matlab
figure(12)
plot(x.*(1/12),RegON.*(1/SIZE(1)).*100)
xlabel('Hour')
ylabel('\% of Units')
axis([0 24 0 100])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24])
hold on
plot(x.*(1/12),ChaON.*(1/SIZE(1)).*100)
plot(x.*(1/12),DisON.*(1/SIZE(1)).*100)
plot(x.*(1/12),Idle.*(1/SIZE(1)).*100)
hold off
```

```matlab
figure(13)
plot(x.*(1/12),STAT1L.*(1/SIZE(1)).*100)
xlabel('Hour')
ylabel('\% of Units')
axis([0 24 0 100])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24])
hold on
plot(x.*(1/12),STAT1L.*(1/SIZE(1)).*100)
plot(x.*(1/12),STAT1L.*(1/SIZE(1)).*100)
plot(x.*(1/12),STAT1L.*(1/SIZE(1)).*100)
plot(x.*(1/12),STAT1L.*(1/SIZE(1)).*100)
hold off
```

\% Describing RegA signal \%
\% First determine which column of RegA array to use.

```matlab
    monthAdd = 0;  % Figure out how many columns to add onto the day.
```
% (months start at 1)
if (mo == 2)
    monthAdd = 31;
elseif (mo == 3)
    monthAdd = (31+28);
elseif (mo == 4)
    monthAdd = (31+28+31);
elseif (mo == 5)
    monthAdd = (31+28+31+30);
elseif (mo == 6)
    monthAdd = (31+28+31+30+31);
elseif (mo == 7)
    monthAdd = (31+28+31+30+31+30);
elseif (mo == 8)
    monthAdd = (31+28+31+30+31+30+31);
elseif (mo == 9)
    monthAdd = (31+28+31+30+31+30+31+31);
elseif (mo == 10)
    monthAdd = (31+28+31+30+31+30+31+31+30);
elseif (mo == 11)
    monthAdd = (31+28+31+30+31+30+31+31+30+31);
elseif (mo == 12)
    monthAdd = (31+28+31+30+31+30+31+31+30+31+30);
end
% Now adjust for full days (days start at 1)
column = day + monthAdd; % Column Determined.

% Now Plot against x.
SIZE2 = size(RegA);
x2 = 1:1:SIZE2(1);
figure(14)
%SIZE3 = size(x2)
%SIZE4 = size(RegA(:,column))
x2 = x2 ./ 1000;
plot(x2, RegA(:,column))
xlabel('Hour')
ylabel('RegA Setpoint as Fraction of Max Power')
axis([0 24 -1 1])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);

if (false)
    figure(2)
    plot(x.*(1/12),UnitsON.*(1./SIZE(1)).*100)
xlabel('Hour')
ylabel('% of Units Turned On')
axis([0 24 0 40])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);
SIZE = size(draws);
counter = 1;
for m=interval:(interval+287)
    DrawCount = 0;
    for n=1:SIZE
        if (draws(n,m) > 0)
            DrawCount = DrawCount + 1;
        end
    end
    DrawON(counter) = DrawCount;
    counter = counter + 1;
end

figure(3)
plot(x.*(1/12),DrawON.*(1/SIZE(1)).*100)
xlabel('Hour')
ylabel('% Households Experiencing a draw')
axis([0 24 0 40])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);

figure(4)
plot(x.*(1/12),DrawON.*(1/SIZE(1)).*100, 'Color', 'b')
hold on
plot(x.*(1/12),UnitsON.*(1/SIZE(1)).*100, 'Color', 'r')
xlabel('Hour')
ylabel('%')
axis([0 24 0 40])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]);
hold off

% Maximum Megawatts
MW = UnitsON .* (4500 * 10^(-6));

figure(5)
plot(x.*(1/12),MW, 'Color', 'b')
xlabel('Hour')
ylabel('MW')
end
end
% BattPlotter5
month = 6;
day = 6;
hour = 2;
min = 0;
duration = 5*12*2;
MW = 80;

[X,i,power] = BattBid4(month, day, hour, min, duration, MW, battsumm);
OldEnergy = BattPlotter3(month, day, battsumm);

% Purpose: Flots.

SIZE = size(X);

Energy = zeros(1,288);
RegON = zeros(1,289);
ChaON = zeros(1,289);
DisON = zeros(1,289);
Idle = zeros(1,200);
Checksum = zeros(1,200);
Max = zeros(1,285);
STAT11 = zeros(1,288);
STAT12 = zeros(1,288);
STAT13 = zeros(1,288);
STAT14 = zeros(1,288);
STAT15 = zeros(1,288);
STAT16 = zeros(1,288);
Checksum2 = zeros(1,288);
Teston = zeros(1,289);
%Hin = zeros(1,289);
dailyInt = 0;
counter = 1;
for n=1:286
    E = 0;
    R = 0;
    C = 0;
    D = 0;
    I = 0;
    M = 0;
    J = 0;
    S11 = 0;
    S12 = 0;
    S13 = 0;
    S14 = 0;
    S15 = 0;
    S16 = 0;
dailyInt = dailyInt + 1;
    % (dailyInt < 259) && (dailyInt >= 139)
    for n=1:SIZE(1)
        E = E + X(n, 6+2*m);
    end
end
\[ M = M + X(n,5); \]
\[
\text{if}(X(n,7+2^m) == 11) \quad \% \quad \text{RegA} \\
R = R + 1; \\
S11 = S11 + 1; \\
\text{end} \\
\text{if}(X(n,7+2^m) == 12) \quad \% \quad \text{Charge or RegA} \\
S12 = S12 + 1; \\
\quad \text{if} \quad (\text{dailyInt} < 258) \quad \& \quad (\text{dailyInt} >= 130) \quad \% \quad \text{If power \!cheap.} \\
\quad R = R + 1; \quad \% \quad \text{So RegA.} \\
\quad \text{else} \quad \% \quad \text{Power is cheap, \!so charging.} \\
\quad C = C + 1; \\
\text{end} \\
\text{end} \\
\text{if}(X(n,7+2^m) == 13) \quad \% \quad \text{Charge or Idle.} \\
\quad S13 = S13 + 1; \\
\text{if} \quad (\text{dailyInt} < 257) \quad \& \quad (\text{dailyInt} >= 138) \quad \% \quad \text{If power \!cheap.} \\
\quad I = I + 1; \quad \% \quad \text{So Idle.} \\
\quad \text{else} \quad \% \quad \text{Power is cheap, \!so charging.} \\
\quad C = C + 1; \\
\text{end} \\
\text{end} \\
\text{if}(X(n,7+2^m) == 14) \quad \% \quad \text{Discharge or RegA.} \\
\quad S14 = S14 + 1; \\
\quad J = J + 1; \\
\text{if} \quad (\text{dailyInt} < 258) \quad \& \quad (\text{dailyInt} >= 138) \quad \% \quad \text{If power \!cheap.} \\
\quad D = D + 1; \quad \% \quad \text{So discharge.} \\
\quad \text{else} \quad \% \quad \text{Power is cheap, \!so RegA.} \\
\quad R = R + 1; \\
\text{end} \\
\text{end} \\
\text{if}(X(n,7+2^m) == 15) \quad \% \quad \text{Discharge or Idle.} \\
\quad S15 = S15 + 1; \\
\quad J = J + 1; \\
\text{if} \quad (\text{dailyInt} < 258) \quad \& \quad (\text{dailyInt} >= 138) \quad \% \quad \text{If power \!cheap.} \\
\quad D = D + 1; \quad \% \quad \text{So discharge.} \\
\quad \text{else} \quad \% \quad \text{Power is cheap, \!so RegA.} \\
\quad I = I + 1; \\
\text{end} \\
\text{end} \\
\text{if}(X(n,7+2^m) == 16) \quad \% \quad \text{DERAS controlled dec (charge).} \\
\quad S16 = S16 + 1; \\
\quad C = C + 1; \quad \% \quad \text{DERAS said charge, \!so charge.} \\
\text{end} \\
\text{end} \\
\% \quad \text{D} \\
\text{Energy(counter)} = E; \\
\text{RegCN(counter)} = R; \\
\text{CharCN(counter)} = C; \\
\text{DisCN(counter)} = D; \\
\text{Idle(counter)} = I; \\
\text{Max(counter)} = M;
Tester(counter) = J;
Checksum(counter) = R+C+D+I;
STAT11(counter) = S11;
STAT12(counter) = S12;
STAT13(counter) = S13;
STAT14(counter) = S14;
STAT15(counter) = S15;
STAT16(counter) = S16;
Checksum2(counter) = S11 + S12 + S13 + S14 + S15 + S16;
counter = counter + 1;
end

% Avg = (Max - Min) / 2 + Min;

% Tester
% Checksum2
% Checksum
% DisON

x = 1:288;

figure(1)
plot(x.*(1/12),Energy.*(1/1000000))
xlabel('Hour')
ylabel('Energy (MN-hr)')
axis([0 24 0 500])
xticks([0,2,4,6,0,10,12,14,16,18,20,22,24]);
hold on
plot(x.*(1/12),Max.*(1/1000000))
plot(x.*(1/12),OldEnergy.*(1/1000000))
plot(x.*(1/12),AVG.*(1/1000000))
hold off

figure(2)
plot(x.*(1/12),RegON.*(1/size(1)).*100)
xlabel('Hour')
ylabel('% of Units')
axis([0 24 0 100])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24])
hold on
plot(x.*(1/12),ChaON.*(1/size(1)).*100)
plot(x.*(1/12),DisON.*(1/size(1)).*100)
plot(x.*(1/12),Idle.*(1/size(1)).*100)
hold off

figure(3)
plot(x.*(1/12),STAT11.*(1/size(1)).*100)
xlabel('Hour')
ylabel('\% of Units')
axis([0 24 0 100])
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24])
hold on
plot(x.*(1/12),STAT12.*(1/SIZE(1)).*100)
plot(x.*(1/12),STAT13.*(1/SIZE(1)).*100)
plot(x.*(1/12),STAT14.*(1/SIZE(1)).*100)
plot(x.*(1/12),STAT15.*(1/SIZE(1)).*100)
plot(x.*(1/12),STAT16.*(1/SIZE(1)).*100)
hold off

if(false)
    % Describing RegA signal %
    % First determine which column of RegA array to use.
    monthAdd = 0; % Figure out how many columns to add onto the day.
    % (months start at 1)
    if(mo == 2)
        monthAdd = 31;
    elseif(mo == 3)
        monthAdd = (31+28);
    elseif(mo == 4)
        monthAdd = (31+28+31);
    elseif(mo == 5)
        monthAdd = (31+28+31+30);
    elseif(mo == 6)
        monthAdd = (31+28+31+30+31);
    elseif(mo == 7)
        monthAdd = (31+28+31+30+31+30);
    elseif(mo == 8)
        monthAdd = (31+28+31+30+31+30+31);
    elseif(mo == 9)
        monthAdd = (31+28+31+30+31+30+31+31);
    elseif(mo == 10)
        monthAdd = (31+28+31+30+31+30+31+31+30);
    elseif(mo == 11)
        monthAdd = (31+28+31+30+31+30+31+31+30+31);
    elseif(mo == 12)
        monthAdd = (31+28+31+30+31+30+31+31+30+31+30);
    end
    % Now adjust for full days (days start at 1)
    column = day + monthAdd; % Column Determined.

    % NOW Plot against x.
    SIZE2 = size(RegA);
    x2 = 1:1:SIZE2(1);
    figure(4)
    %SIZE3 = size(x2)
    %SIZE4 = size(RegA(:,column))
x2 = x2 ./ 1800;  
plot(x2, RegA(:,column)) 
xlabel('Hour') 
ylabel('RegA Setpoint as Fraction of Max Power') 
axis([0 24 -1 1]) 
xticks([0,2,4,6,8,10,12,14,16,18,20,22,24]):
```matlab
mo = 6;
day = 6;
hour = 2; % 0 through 23
minInt = 0; % 0 through 11.
duration = 5*12*2; % must be multiples of 5.
amount = 20; % MegaWatts
[X,f,power1]=BattBid4(mo,day,hour,minInt,duration,amount,battsum);
amount = 30;
[X,f,power2]=BattBid4(mo,day,hour,minInt,duration,amount,battsum);
amount = 40;
[X,f,power3]=BattBid4(mo,day,hour,minInt,duration,amount,battsum);
amount = 50;
[X,f,power4]=BattBid4(mo,day,hour,minInt,duration,amount,battsum);
amount = 60;
[X,f,power5]=BattBid4(mo,day,hour,minInt,duration,amount,battsum);
amount = 70;
[X,f,power6]=BattBid4(mo,day,hour,minInt,duration,amount,battsum);
amount = 80;
[X,f,power7]=BattBid4(mo,day,hour,minInt,duration,amount,battsum);

SIZE = size(power1);
x = 1:SIZE(2);

figure(13)
plot(x,power1)
axis([1,SIZE(2),0,100])
xlabel('5 Minute EIM Intervals')
ylabel('Power (MW)')
title('Power Absorbed by DERAS for Simulated Bids from 02:00 to 04:00')
hold on
plot(x,power15)
plot(x,power2)
plot(x,power3)
plot(x,power4)
plot(x,power5)
plot(x,power6)
plot(x,power7)
hold off
```
% maxNewBatt
max = zeros(1,200);
%new = zeros(1,50*10);

%for n = 1:500
% new(n+1) = 0;
%end

x = 0;
for m = 1:200
    for n = 12:5:35
        f = 0;
        [X,I,intAcct] = BattBid4(6,6,2,0,5*m,n,battsumm);
        if(f>0)
            break
        else
            max(i,m)=n;
        %continue
        end
    end
end