2019-07-01

Proactive Energy Optimization in Residential Buildings with Weather and Market Forecasts

Cody Ryan Simmons
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/7594

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
This work explores the development of a home energy management system (HEMS) that uses weather and market forecasts to optimize the usage of home appliances and to manage battery usage and solar power production. A Moving Horizon Estimation (MHE) application is used to find the unknown home model parameters. These parameters are then updated in a Model Predictive Controller (MPC) which optimizes and balances competing comfort and economic objectives. Combining MHE and MPC applications alleviates model complexity commonly seen in HEMS by using a lumped parameter model that is adapted to fit a high-fidelity model. HVAC on/off behaviors are simulated by using Mathematical Program with Complementary Constraints (MPCCs) and solved in near real-time with a nonlinear solver. Removing HVAC on/off as a discrete variable decreases potential solutions and consequently reduces solve time and increases the probability of reaching a more optimal solution. The results of this work indicate that energy management optimization significantly decreases energy costs and balances energy usage more effectively throughout the day compared to a home with regular temperature control. A case study for Phoenix, Arizona shows an energy reduction of 21% and a cost reduction of 40%. Homes using this home energy optimization will contribute less to the grid peak load and therefore, improve grid stability and reduce the amplitude of load following cycles for utilities. This case study combines renewable energy, energy storage, forecasts, cooling system, variable rate electricity plan and a multi-objective function allowing for a complete home energy optimization assessment. There remain several challenges, including improved forecast models, improved computational performance to allow the algorithms to run in real-time, and mixed empirical / first principles machine learning methods to guide the model structure.

Keywords: model predictive control, moving horizons estimation, home energy optimization, forecast, thermal modeling, HEMS, energy storage, solar generation
ACKNOWLEDGMENTS

I would like to acknowledge the Utah Science Technology and Research (USTAR) and Brigham Young University for providing financial support in this project. I would also like to express my appreciation to those that have assisted in making this project possible: Kody Powell, Stella Nickerson for their guidance as committee members, Damon Petersen for help with the literature review and editing, and Derek Prestwich for assistance with programming and editing. I would also like to thank my thesis advisor, Dr. John Hedengren, for his expertise and mentoring over the course of this project. I am very grateful for the encouragement and continual support from my family not only for this project, but also for my education.
# TABLE OF CONTENTS

LIST OF TABLES ................................................................. vi

LIST OF FIGURES .............................................................. vii

Chapter 1 Introduction ...................................................... 1
  1.1 Grid Stabilization .................................................... 1
  1.2 Demand Side Management .......................................... 3
  1.3 Forecasting Methodology ........................................... 4
  1.4 Model Predictive Control in Building Energy Management ....... 5
  1.5 Accounting for Forecast Uncertainty and System Disturbances .. 6
  1.6 Contributions ....................................................... 7

Chapter 2 Theory and Methods used in Developing Energy Management Software ........................................ 10
  2.1 Methods ............................................................... 10
  2.2 Theory/Overview .................................................... 10
  2.3 Simulation ........................................................... 11
  2.4 Building House Model .............................................. 12
    2.4.1 Initial Model .................................................. 12
    2.4.2 Improved Model ............................................... 15
  2.5 Air Conditioner Load Model ....................................... 17
  2.6 Battery Model ....................................................... 18
  2.7 Ambient Temperature Prediction Model .......................... 20
  2.8 Miscellaneous Building Power Prediction Model ................. 23
  2.9 Solar Power Production Prediction Model ......................... 25
  2.10 Moving Horizon Estimation and Model Predictive Control Theory .. 28
  2.11 Moving Horizon Estimation with Lumped Parameters .......... 31
  2.12 Model Predictive Controller with Forecasts .................. 33

Chapter 3 Case Studies ..................................................... 40
  3.1 Base Case .......................................................... 40
  3.2 Optimized Case .................................................... 42
    3.2.1 Comparison to Base Case ................................... 44
  3.3 MPC only HEMS Case study ....................................... 45
    3.3.1 Comparison to full HEMS application ....................... 47

Chapter 4 Conclusions and Future Work ................................ 48
  4.1 Future work ....................................................... 49

REFERENCES ................................................................................. 51

Appendix A Software Interfacing for Advanced Control ............. 57
  A.1 Software Interfacing for Advanced Control ..................... 57
Appendix B  Source Code  .................................................. 59
  B.1  Source Code  .................................................. 59

Appendix C  EnergyPlus configuration file  .................................. 84
  C.1  EnergyPlus .idf file  ............................................. 84
# LIST OF TABLES

2.1 Battery Model Nomenclature ............................................. 19  
2.2 Commercial Home Energy Storage and Lithium-ion cell specifications ....... 19  
2.3 Objective function terms .................................................. 30
LIST OF FIGURES

1.1 HEMS Data flow ................................................................. 9
2.1 Simulation Test Results ...................................................... 11
2.2 Diagram of the Lumped Parameter Home Model ...................... 13
2.3 Original Model vs. Improved Model ...................................... 16
2.4 Actual vs. Predicted for Air Conditioner Load ....................... 17
2.5 Actual vs. Predicted for Ambient Temperature 10 minutes ahead .... 21
2.6 Actual vs. Predicted for Ambient Temperature 24 hours ahead .... 22
2.7 Actual vs. Predicted for Misc. Building Power 10 minutes ahead .... 24
2.8 Actual vs. Predicted for Misc. Building Power 24 hours ahead .... 25
2.9 Actual vs. Predicted for Solar Power Production 10 minutes ahead ... 26
2.10 Actual vs. Predicted for Solar Power Production 24 hours ahead ... 27
2.11 MHE results during typical temperature control scheme ............ 32
2.12 MHE results during complex simulation ................................ 33
2.13 Air Conditioning Function .................................................. 35
2.14 MPC sample Solution ....................................................... 38
3.1 Phoenix, AZ house with no optimization ............................... 41
3.2 Optimized House in Phoenix, AZ .......................................... 43
3.3 Optimized House in Phoenix, AZ .......................................... 46
CHAPTER 1. INTRODUCTION

The state of Utah is a leader in solar energy installations, ranking second nationally in 2016 with a cumulative total of over 1500 MW installed, enough power for 300,000 homes [1]. This investment helps reduce air quality problems, but also results in several substantial technological challenges. Alternative energy sources such as solar and wind power have inherent variability which has been shown to destabilize the power grid as the market penetration increases [2]. The move toward alternative energy and the associated problems have motivated development on predictive control methodologies for building energy management. Demand response optimization mitigates the problems by shifting demand to match production and shaving peak demand. Proactive and predictive methods also restore some grid stability, especially when coupled with energy storage technologies which act as buffers to the time variability of alternative energy sources [1]. In this work, a proactive demand response energy system management algorithm incorporates a reduced-order building model with model predictive control (MPC) and moving horizon estimation (MHE). Conventional energy management solutions currently in implementation are predominantly reactive, meaning that they only respond to external stimuli as they happen. The proposed system incorporates machine-learned forecasts of future conditions (e.g., forecast supply, demand, and price) to determine when to store, consume, or generate energy and modify the model dynamically based on system measurements in a specified past time horizon (MHE). While the research focuses on specific energy systems where data is readily available, the resulting approach has application at a wide range of scales including single homes and larger buildings such as apartment complexes.

1.1 Grid Stabilization

Due to the time dependency of solar and wind, when energy is used becomes as important as how energy is used, if not more so. Electricity is a commodity that must be generated
and consumed simultaneously. Both solar and wind energy are inherently intermittent, causing issues with simultaneous production and consumption [3]. The introduction of intermittent energy sources creates a grid management problem that is currently being addressed with a reactive operating strategy: rapidly ramping fuel-fired power plants to accommodate ever-changing solar and wind [3]. This is an inefficient use of capital and energy. Reactive strategies create a practical limit to the amount of renewable power that is accommodated by grid infrastructure. Much of the variability in the grid is predictable given weather forecasts and knowledge of consumer habits [4]. While commercially viable storage technologies are emerging, none have been married to predictive energy management tools that leverage existing techniques for forecasting energy supply and demand. By implementing predictive algorithms that use machine-learned forecasts, energy systems are responsive, charging storage ahead of a shortage event to alleviate future demand.

Powell et al. demonstrate this concept in a paper demonstrating an energy system that uses energy storage systems to match energy production with energy demand, using a solar power plant as a case study [3]. The goal of the study is to proactively optimize the system to respond to changes in the environment rather than reactively responding to changes. Powell et al. demonstrate that proactive dynamic optimization increases the percentage of incident energy collected by the solar power plant in all scenarios, but most significantly on days when cloud cover exacerbates the intermittent nature of the power source. Powell et al. conclude that different weather conditions correspond with different optimal operating strategies, further demonstrating that proactive control utilizing forecast data improves the efficiency of operation of renewable energy sources. This demonstrates how proactive energy management leads to increased grid stability.

Smart metering is another strategy for maintaining grid stability. Smart meters allow utilities to receive real time data of consumer energy usage. When the majority of consumers have smart meters, a smart grid is formed. Smart grids allow seamless integration of renewable energy sources due to increased knowledge of the system [5, 6]. Utility companies react to changes because the smart grid provides real-time data on consumers’ energy usage. If consumers generate power using solar or wind, then there is a reduced demand seen by the utility company. The use of smart meters allows the utility company to see this reduced demand in real time and make decisions accordingly. Smart metering also provides the opportunity for dynamic energy pricing. Dynamic energy pricing allows the utility to manage the electricity on the demand side. With variable utility
rates, residential and industrial consumers are able to react to changing electricity prices. This leads to an increase in grid reliability and efficiency as it reduces peak loads [7]. An important observation is that these pricing strategies often do not reduce the amount of energy consumed during the entire day, but instead, smooth out the demand throughout the day.

1.2 Demand Side Management

Heating, ventilation, and air conditioning (HVAC) systems are among the largest electrical energy consumers and large contributors to peak demand in the United States [8]. One major topic of recent research is demand side management or demand response optimization. Demand response optimization focuses on control strategies for consumers of electrical energy, often responding to dynamic pricing structures set up by utility companies. Sheha and Powell show that a HEMS with a PV system along with battery energy storage can be economically feasible if coupled with the right utility rate structure [9]. Sheha and Powell also review how rate structures can incentivize or de-incentivize energy storage and other peak shaving technologies [9]. With the right rate structure, consumers can implement strategies to reduce their electricity consumption and reduce costs. Simultaneously, this demand side response helps the utility company and enhances grid stability by smoothing out energy demand throughout the day and reducing peak demand, often through energy storage.

Multiple review articles have been published on demand response [10–15]. Aghaei and Alizadeh review demand response, showing how demand response leads to greater grid stability with renewable energy penetration [10]. Shariatzadeh et al. review demand response implementation and methods and suggest how demand response may be implemented in smart grids [14]. Wang et al. and Brahman et al. review and describe methods for integrated demand response in smart energy hubs. These hubs combine local energy generation sources (i.e. solar power, natural gas turbines) and energy storage (thermal energy storage, battery) and allow them to manage sources of consumed power, acting as semi-independent energy systems interacting with the grid [15, 16]. Good et al. describe the barriers to implementation of demand response including regulatory barriers, market design, physical network barriers, anthropogenic barriers, and other human factors such as lack of understanding [11]. O’Connell et al. review many of the challenges to demand response such as lack of experience, inherent system uncertainty (i.e. weather and occupancy),
reliable control strategies, and market frameworks [17]. Paterakis et al. review demand enabling technologies for demand response and provide an overview of the global status quo of demand response implementation [12]. Yoon et al. investigated the effects of using a demand response controller (DRC) to control the temperature set-points in a home to reduce peak load and energy cost. They demonstrate that the DRC is capable of decreasing the peak load by 24.7%, annual electricity HVAC uses by 4.0%, and cost of electricity by 7.7% to 10.8% based on the pricing structure [8]. This study shows that there is potential for manipulating home energy consumption by adjusting temperature set-points. Controlling set-points with the addition of market and weather forecasts is predicted to provide even better reductions in cost, consumption, and peak loads.

1.3 Forecasting Methodology

Forecasting is one of the most important tools for proactive energy management systems. Proactive systems inherently require predictions of future system behavior to mitigate foreseen issues rather than reacting to them. A significant amount of research has been done to optimize and forecast HVAC systems as they are the largest consumer of building electricity and consequently cause a majority of peak loads [18–20]. Weather forecasting is also an active research topic because of the increasing penetration of renewable energy sources and the inherent dependency on weather conditions for power generation. It is crucial to incorporate weather and HVAC forecasting because they are intrinsically interrelated. For example, if it is a hot and sunny day, solar energy generation increases and the HVAC system requires more energy to cool the home. The energy management system may sub-cool the home or allow the temperature set-point to increase if there is insufficient solar power with peak energy prices. The energy management software may also reduce energy consumption at peak energy price times to sell electricity back to the grid. This is the underlying reason proactive building energy management systems are useful.

A significant amount of literature has been published on the development of efficient and accurate forecasting algorithms [21]. Bilbao et al. present a machine-learning (ML) based regression forecasting method using an artificial neural network (ANN) with Bayesian regulation back-propagation and test this method on a building at the University of New South Wales [22]. Daut et al. use a hybrid swarm intelligence algorithm and support vector machine to achieve superior performance for building electrical energy consumption forecasting [23]. Deb et al. perform
a review on time series energy consumption forecasting for building energy optimization by comparing methods based on historical data (black box) to those based on first-principles simulation (white box) [24]. Tanveer et al. and Wei et al. both review data-driven methods for prediction of large scale building energy consumption with a focus on artificial neural networks, clustering, and support vector machines [4, 25]. Wang et al. review artificial intelligence methods for building energy prediction with an emphasis on the difference between single and ensemble methods [26]. Foucquier et al. review first-principles physics based building modeling and energy performance predictions [27]. Amasyali et al. review building energy consumption prediction methods with a focus on data pre-processing, data processing, and prediction itself [28]. Fumo et al. provide an overview of prediction models which work with entire building energy simulation software packages [29]. Lazos et al. review common forecasting methods, discussing the difference between the short prediction horizons typical of statistical or data-driven methods compared to the longer time horizons typical of physics-based methods [30].

1.4 Model Predictive Control in Building Energy Management

Although many control strategies are employed for proactive energy management of building energy systems, this work’s scope is limited to MPC. MPC is a well-known control method which utilizes a process model to predict future values given a set of control moves over a future time horizon. The set of control moves over the future horizon is optimized using dynamic optimization methods which are well-documented in literature [31,32]. When integrated with forecasting, dynamic optimization allows the control strategy to proactively mitigate for disturbances prior to their occurrence when they are predicted to occur during the future time horizon. MPC for building energy optimization has been an active topic for several decades, with multiple review articles written on the topic [33–36]. One important aspect of MPC is the model used for optimization and prediction. Model complexity has a significant impact on computational power and time required to solve the dynamic optimization problem, sometimes making on-line implementation of the control strategy infeasible. Several articles identify methodologies to reduce computational complexity of the building energy MPC problem. Picard et al. present a linear time invariant (LTI) state space model for the problem which allows the model complexity to be decoupled from the required computational time, although at the expense of system nonlinearities [37]. Ruiz et al.
introduce a genetic algorithm (NGSA-II) as an optimization method in an effort to reduce the computational burden of the problem [38]. Sangi et al. suggest a model based on exergy (usable work) rather than on energy consumption and an agent-based hybrid model predictive controller [39]. Santoro et al. introduce a nonlinear model predictive control (NMPC) formulation with a nonlinear model to describe the building system [40].

Another important aspect of MPC in relation to building energy optimization and grid stabilization is the integration of energy storage such as thermal energy storage (TES) and batteries. Yu et al. conduct a comprehensive review of control strategies to integrate thermal energy storage to shift peak load and allow for greater renewable energy penetration in the grid, including an overview of MPC’s space [33]. Khakimova et al. present a method for utilizing MPC on a smart house equipped with photovoltaic (PV) power, thermal energy storage, and an HVAC system with the objective to minimize energy purchased from the grid [41]. Multiple objectives also commonly exist in building energy optimization as occupant thermal comfort and energy efficiency are typically conflicting objectives. This concern is inherently covered in all works in this space, but is more explicitly addressed by Ascione et al. [42]. Ascione et al. use a multi-objective dynamic optimization with a genetic algorithm, hourly set-points, and weather and occupancy forecasts to generate a Pareto front from which the user chooses an optimal solution based on comfort and efficiency preferences. Oldewurtel et al. developed a MPC framework for integrated room automation to control thermal comfort, maximize energy efficiency, as well as control luminance and room carbon dioxide levels [43]. Touretzky tackle the issue of time-scales by implementing a long-time scale HVAC scheduling formulation with forecasts and a short-time scale model predictive controller to meet the proactive schedule [44]. Touretzky et al. introduce an economic MPC formulation for an overall economic objective in the same work. Afram et al. provides a review of for building energy optimization, addressing differences among literature in many aspects of MPC [34]. Killian et al. provide answers to many practical and technical questions regarding MPC for building energy optimization [35].

1.5 Accounting for Forecast Uncertainty and System Disturbances

MPC provides an effective means for proactive control and optimization of building HVAC systems to reduce energy consumption and maximize efficiency, all while enhancing grid stabi-
lization. However, energy consumption and weather forecasts which allow for effective predictive and proactive control are often inaccurate, which leads to sub-optimal control [24, 45, 46]. In addition to uncertainty in building energy forecasts and weather forecasts, system disturbances can occur which could be unpredicted by the proactive controller [47]. There have been many methods presented to deal with these issues in the academic literature. Santoro et al. introduce system disturbances and model mismatch into their simulations with NMPC of a building energy system, accounting for uncertainty in model mismatch and demonstrating robust performance [40]. Oldewurtel et al. present a stochastic model predictive control (SMPC) approach to account for uncertainty in weather forecasts which outperforms the control in their study [43]. Vahid-Pakdel et al. use a stochastic mixed-integer linear programming (MILP) methodology to account for uncertainties such as in demands, prices, and wind speed in a smart grid optimization, reducing costs by 5% [48]. Kim et al. use adaptive MPC and multiple distributed simple system models to both alleviate computational burden and address system disturbances in building energy control [47]. Zhang et al. introduce randomized model predictive control (RMPC) which samples multiple possible scenarios within an uncertainty window to implement stochastic methods in the absence of a probabilistic disturbance model [49]. Kwak et al. introduce methods to incorporate real-time weather data and building energy consumption data into MPC to mitigate error due to uncertainty and inaccuracy in forecasts [46, 50]. Ebrahimpour and Santoro develop a methodology for utilizing MHE in conjunction with MPC with a simplified model to account for uncertainties and model measurement mismatch due to load and occupancy [51]. In this work we present a methodology to utilize MHE to incorporate real-time weather and building energy consumption data into MPC via dynamic parameter estimation.

1.6 Contributions

The novel contributions of this work include the following:

1. A home energy management system (HEMS) that uses a combined MHE and MPC approach that estimates residential home building parameters and optimizes home energy in real-time. Mathematical building models are often very complicated and computationally expensive.
This work overcomes this obstacle by using a lumped parameter model that is adapted to fit a high-fidelity model.

2. A hybrid approach manages the on/off behavior of air conditioners as a continuous function using a mathematical program with complementarity constraints (MPCC). Discrete variables increase the number of potential solutions which require more computation resources and time. By using MPCC’s, the discrete behaviors are converted into a continuous function that is solved with continuous optimization.

3. A combined renewable energy, energy storage, forecasts, cooling system, variable rate electricity plan, and multi-objective function residential house model is presented and tested on a simulated home in Phoenix, Arizona. This combination includes all major energy flows within a home as well as important outside disturbances. Accounting for these factors allows for a complete home energy optimization assessment with controlled conditions to assess the potential benefit from the HEMS approach.

This work provides a framework to develop and optimize a HEMS with forecasting. Figure 1.1 shows how each of the individual components fit together to form the complete HEMS framework. The process begins with simulated home data. This data is sent to the MHE application where the home parameters values are estimated. Simulation data is also sent to the forecasting models where future values are predicted. The home parameters and future predictions are sent to the MPC application where climate control, battery usage, and energy flows in the home are optimized. Optimized decisions are sent back to the simulation and implemented into the HEMS. This cycles repeats every 10 minutes to allow time for the optimizer to converge to a solution.

This work is useful to researchers and HEMS developers interested in home modeling, automation, forecasting, and energy usage optimization. The following sections of the paper describe the methods used to model and optimize homes using the energy management software. The concluding sections quantify and discuss the effect of implementing the optimization software. They also provide suggestions to improve and continue the work discussed in this paper. Appendix A, B, and C include the source code required to run the HEMS and simulation.
Figure 1.1: HEMS Data flow
CHAPTER 2. THEORY AND METHODS USED IN DEVELOPING ENERGY MANAGEMENT SOFTWARE

2.1 Methods

This chapter discusses the reduced-order and EnergyPlus models, HVAC thermal and electrical systems, and weather forecast models. It also describes the mathematical formulation of MHE and MPC methods used. Lastly, the results of the MHE and MPC applications are included and discussed.

2.2 Theory/Overview

A simulator is used to allow practical and efficient software testing. EnergyPlus is a console-based program funded by the Department of Energy (DOE). Energy is used to simulate whole building energy systems and allows users to completely customize the building parameters. EnergyPlus provides energy consumption data for the whole home including lights, plugs, appliances, heating and cooling. EnergyPlus also allows users to manipulate time step resolution, HVAC configurations, heat transfer calculation strategies, etc. Using the data from the simulator, an MHE is used to fit parameters in a heat transfer equation that makes a correlation between ambient temperature and the inside home temperature. This correlation is then combined with forecasts to optimize home energy flows by manipulating variables such as temperature set-points. In addition to changing the home temperature set-points, the controller makes decisions on whether to charge or discharge a battery used for energy storage and whether to buy or sell electricity to the grid. These decisions are determined by minimizing the total cost of the energy system. Profits can be gained at individual time steps if net metering is available and there is an excess amount of energy in the home system. Ultimately, the solver is minimizing and shifting energy loads which results in minimizing energy costs when properly utilizing variable rate pricing structures.
2.3 Simulation

Figure 2.1 shows the results of an example simulation in EnergyPlus. The blue line is the amount of electricity used by the HVAC system to cool the house and the red line is the ambient temperature outside of the home. A correlation is seen between the ambient temperature and the energy required by the HVAC system.

![Figure 2.1: Simulation Test Results](image)

As discussed in the introduction, air conditioning units are one of the main electricity-demand contributors. As a result, the initial focus is to build an energy management software that encapsulates the correlation seen in Figure 2.1 to reduce electricity costs. Summer months have
the highest cooling demand and have the highest potential to improve energy usage. Consequently, this work focuses on optimization during summer months.

2.4 Building House Model

2.4.1 Initial Model

An accurate house model is needed to optimize the energy usage. A reduced first principles based model is developed to accurately model the thermodynamics of the house. The basis of the house model is derived from the total energy equation shown in Equation 2.1.

\[
\frac{\partial}{\partial t} \left( \frac{1}{2} \rho v^2 + \rho \dot{U} \right) = - \left( \nabla \cdot \left( \frac{1}{2} \rho v^2 + \rho \dot{U} \right) v \right) - \left( \nabla \cdot q \right) - \left( \nabla \cdot p v \right) - \left( \nabla \cdot \tau \cdot v \right) + \rho \left( v \cdot g \right) \tag{2.1}
\]

One of the main goals of this work is to develop a model that is accurate but also simple. Although Equation 2.1 does not include nuclear, radiative, electromagnetic, or chemical forms of energy, it is still too complex to be optimized efficiently in a real time HEMS. Equation 2.1 is simplified with a few assumptions. The first assumption is that energy transfer is dominated by convective and conductive heat transfer. This assumption eliminates the last three terms of Equation 2.1. The next assumption is that there is no mechanical energy transfer. This assumption eliminates \( \frac{1}{2} \rho v^2 \) from the Equation 2.1. With these two assumptions the model is reduced down to Equation 2.2.

\[
\frac{\partial}{\partial t} \rho \dot{U} = - \left( \nabla \cdot \rho \dot{U} v \right) - \left( \nabla \cdot q \right) \tag{2.2}
\]

Equation 2.2 is a simplified version of the total energy balance and contains the information needed for an effective house model. Next, it is helpful to switch Equation 2.2 from the internal energy form to a temperature form. This is done by first converting internal energy to enthalpy and then enthalpy to a function of temperature. The resulting equation is shown in Equation 2.3.

\[
V \rho C_p \frac{dT}{dt} = \left( h_{\text{conv}} - \frac{k_{\text{cond}}}{L} \right) \Delta T \tag{2.3}
\]
The final transformation to the energy balance is to lump the parameters together. Parameter lumping is helpful in simplifying a model because one parameter contains the information of many constant values instead of estimating each one individually. This concept is particularly helpful when estimating parameter values in an MHE application because it simplifies the solution and also decreases the degrees of freedom. Equation 2.4 shows the final lumped parameter house model which closely resembles Newton’s Law of Cooling. Equation 2.5 shows how the parameters are lumped.

\[
\frac{dT}{dt} = \lambda \Delta T \tag{2.4}
\]

\[
\lambda = \frac{(hA_{\text{conv}} - kA_{\text{cond}})}{V \rho \hat{C}_p} \tag{2.5}
\]

Equation 2.4 is now applied specifically to the HEMS. The goal is to have a mathematical relationship between the outside temperature and the temperature of the air inside the home. To achieve this relationship, the house is divided into parts. These parts include, the outside surface of the wall, the inside surface of the wall, and the air contained in the home itself. This is done because the transfer of energy and heat capacitance of these sections are different and must be separated to model the home accurately. Figure 2.2 shows a diagram of how the house is divided into different parts.

![Diagram of the Lumped Parameter Home Model](image-url)

Figure 2.2: Diagram of the Lumped Parameter Home Model
The first part of the energy balance is the heat transfer through the outer wall which is represented by Equation 2.6. This equation is displayed in the same order as the generic balance equation seen in Equation 2.2. The left side of Equation 2.6 is the accumulation of total energy contained in the outer wall. The parameter $A$ contains information about the density, heat capacity, and area of the outer wall. This value is estimated so the exact values of density, heat capacity, and area are not needed. The accumulation term also contains the time differential of the outside wall temperature. This allows the model to adjust for changes in the total energy in the outer wall over time. The right side of Equation 2.6 contains terms that account for the input and output flows to and from the outside wall. The first term is the flow of energy by convection due to the ambient temperature around the outside of the home. The second term is the flow of convection due to the inner wall temperature. If the difference in temperatures is positive, it means the energy is flowing into the home and vice versa for a negative difference.

$$A \frac{dT_{\text{wall,outer}}}{dt} = B (T_{\text{wall,outer}} - T_{\text{air,ambient}}) - C (T_{\text{wall,outer}} - T_{\text{wall,inner}}) \quad (2.6)$$

Equation 2.7 describes the energy balance around the inner wall. The concepts are the same as the outer wall. Energy flows in and out by convection from the outer wall and the air inside the home.

$$D \frac{dT_{\text{wall,inner}}}{dt} = E (T_{\text{wall,inner}} - T_{\text{wall,outer}}) - F (T_{\text{wall,inner}} - T_{\text{air,inside}}) \quad (2.7)$$

Finally, the energy balance of the air inside the home needs to be developed. As before, there is a convection term that describes the energy flow between the inner wall and the air inside the home. Equation 2.8 includes a new term, $Q_{HVAC}$, which is the amount of energy flow from the HVAC system. $Q_{HVAC}$ is positive for an air-conditioner and negative for a furnace or heater. Lastly, HVAC I/O is whether the HVAC system is on or off. This is a binary variable represented as a 0 or a 1.

$$G \frac{dT_{\text{air,inside}}}{dt} = H (T_{\text{air,inside}} - T_{\text{wall,inner}}) - Q_{HVAC} (HVAC I/O) \quad (2.8)$$

Equations 2.6 - 2.8 combine to form a reduced-order model capable of representing the principle temperature dynamics of the home. The energy balances neglect radiation and conduc-
tion influences. These are negligible compared to convection at ambient temperatures so they are eliminated from the energy balance.

### 2.4.2 Improved Model

A deficiency in the lumped parameter model described by Equations 2.6 - 2.8 is exposed when testing with the EnergyPlus simulator. The lumped parameter model above fits the house data well when the air conditioner is only on for short periods of time, but fails to capture the correct behavior if the air conditioner is on for extended periods. As a result, the optimizer is unable to predict the temperature of the home appropriately and a new model is developed.

The energy balance described in Equation 2.2 still holds true, but the model is augmented. The augmented model equations are shown below in Equations 2.9 and 2.10.

\[
\frac{dT_{\text{air,inside}}}{dt} = A(T_{\text{ambient}} - T_{\text{air,inside}}) + B(T_{\text{ground}} - T_{\text{air,inside}}) - C(HVAC_{I/O}) - D(HVAC_{Status})
\]  

(2.9)

\[
\frac{dHVAC_{I/O}}{dt} = HVAC_{Status}
\]  

(2.10)

Two major additions are incorporated in this model which are the heat transfer effects of the ground temperature and an additional term for the HVAC system. The ground temperature influence is added because it improved the model accuracy when applying it to a variety of different climates.

The last term of Equation 2.9 is what ultimately improved the model. Equation 2.10 shows that \( HVAC_{Status} \) is just the change in \( HVAC_{I/O} \). This allows the model to track what state the HVAC system is in. For example, when the air conditioner turns on, the status goes from 0 to 1 and results in a \( HVAC_{Status} \) value of 1. On the other hand, if the air conditioner turns off, then the status goes from 1 to 0 and results in a \( HVAC_{Status} \) value of -1. This is important to capture in the model because HVAC systems have different behaviors when the system turns on, has been on for a period of time, and turns off. With the combination of the terms \( C(HVAC_{I/O}) \) and \( D(HVAC_{Status}) \), the house model accurately represent these effects.
Figure 2.3 shows the results of the change. The first subplot shows the original model. This model was developed when the air conditioner status was changed rapidly in a cyclic manner. This subplot shows the result when the status is more variable. There is significant deviation from the actual data even though the model seems to match the points of inflection. The second subplot of Figure 2.3 shows the results of the improved model. The fit is not perfect but does match the data closer than the original model. These subplots show that the improvement was critical to the success of the HEMS because the optimizer will not likely provide a cyclic solution.

It is important to notice that the number of balance equations in the previous model is reduced from three down to one. The addition of Equation 2.10 complicates the model and increases the time to converge. This complication arises because the model is discretized over a long time horizon and is non-linear. To maintain fast and reliable solutions, the energy balance equations for $T_{wall,inner}$ and $T_{wall,outer}$ are removed by setting the inner wall, outer wall, and inside air temperature equal. The results and comparison of the models are discussed in more detail in further sections.
2.5 Air Conditioner Load Model

Another aspect of the system that needs to be modeled is the air conditioner load. Although the air conditioners used in this work are On/Off, power consumption varies based on the efficiency of the unit. This efficiency is a function of many external influences such as humidity, temperature, age of the unit, and so on. Through careful consideration of many of these influences, the ambient air temperature is found to have a dominating impact on the efficiency and that the other effects are negligible. As a result, Equation 2.11 is developed to model the load of the air conditioner unit. The constant $A$ is obtained by using MHE. Figure 2.4 shows the accuracy of the model which has an average $R^2$ value greater than .99.

$$HVAC_{load} = A \frac{HVAC_{1/0}}{T_{air,ambient}}$$  \hspace{1cm} (2.11)

This is a crucial addition because it allows the optimization algorithm to perform better than if a constant load is assumed. This is especially true because the efficiency is worse during
the middle of the day and results in a higher load as shown below in Figure 2.4. This is commonly when air conditioners are on and when energy prices are at their highest. Having a model that explains these trends is essential to finding the true optimum.

2.6 Battery Model

Energy storage supplements a HEMS. Having the ability to store and use energy at different times then when it is produced allows MPC to have more degrees of freedom to optimize the energy system. To be able to do this, an accurate battery model is needed so that the system can anticipate storage usage and efficiency.

The model used in this work is a commercial home energy storage battery system as seen in Table 2.2. Equations 2.12 - 2.13 convert the total battery unit into a system of lithium-ion battery cells. The individual lithium-ion cell properties are found in Table 2.2.

\[ N_{Cells_{series}} = \frac{V_{battery}}{V_{cell}} \]  
\[ N_{Cells_{parallel}} = \frac{Q_{battery}}{Q_{cell}} \]

Equations 2.14 - 2.18 are the remainder of the battery model equations. The first equation calculates the power going into or out of the battery after accounting for AC to DC, or DC to AC inversion losses. Equations 2.15 and 2.16 convert the power into the current in or out of the individual battery cells. Equations 2.17 and 2.18 track the battery state of charge (SOC).

\[ P_{inv} = \frac{P_{in/out}}{P_{inv}} \]
\[ I_{battery} = \frac{P_{inv}}{V_{battery}} \]
\[ I_{cell} = \frac{I_{battery}}{N_{Cells_{parallel}}} \]
\[ \frac{dQ_{discharged}}{dt} = I_{cell} \]
\[ SOC_{battery} = \frac{Q_{battery} - Q_{discharged}}{Q_{battery}} \]
Table 2.1: Battery Model Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{battery}$</td>
<td>Current In/Out of Battery</td>
</tr>
<tr>
<td>$I_{cell}$</td>
<td>Current In/Out of Cell</td>
</tr>
<tr>
<td>$N_{Cells_{parallel}}$</td>
<td>Number of Cells in parallel</td>
</tr>
<tr>
<td>$N_{Cells_{series}}$</td>
<td>Number of Cells in Series</td>
</tr>
<tr>
<td>$P_{inv}$</td>
<td>DC Power</td>
</tr>
<tr>
<td>$inv_{eff}$</td>
<td>Efficiency of the Inverter</td>
</tr>
<tr>
<td>$P_{in/out}$</td>
<td>AC Power</td>
</tr>
<tr>
<td>$V_{battery}$</td>
<td>Battery Voltage</td>
</tr>
<tr>
<td>$V_{cell}$</td>
<td>Cell Voltage</td>
</tr>
<tr>
<td>$Q_{discharged}$</td>
<td>Cell capacity discharged</td>
</tr>
<tr>
<td>$Q_{battery}$</td>
<td>Battery capacity</td>
</tr>
<tr>
<td>$Q_{cell}$</td>
<td>Cell capacity</td>
</tr>
<tr>
<td>$SOC_{battery}$</td>
<td>State of Charge of the Battery</td>
</tr>
</tbody>
</table>

The combination of Equations 2.12 - 2.18 allows the MPC to optimally select energy storage and discharge.

Table 2.2: Commercial Home Energy Storage and Lithium-ion cell specifications

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery Voltage</td>
<td>50 V</td>
</tr>
<tr>
<td>Battery Usable Capacity</td>
<td>13.5 kWh</td>
</tr>
<tr>
<td>Round Trip Efficiency</td>
<td>90%</td>
</tr>
<tr>
<td>Maximum charge/discharge Power</td>
<td>5 kW</td>
</tr>
<tr>
<td>Lithium-ion Cell Capacity</td>
<td>1 Ah</td>
</tr>
<tr>
<td>Lithium-ion Cell Voltage</td>
<td>3.6 V</td>
</tr>
</tbody>
</table>
2.7 Ambient Temperature Prediction Model

The most important forecast variable is the ambient temperature. This is due to the major effect on the heat transfer to the home, and consequently, the power needed to keep the home within a comfortable range of temperatures. If ambient temperatures are predicted, the house model described above predicts when the air conditioner needs to run. The MPC optimizes the start time and duration of cooling load. Bad predictions cause bad solutions, even solutions that can have worse effects than no optimization at all.

This work uses an empirical forecasting model. One reason for choosing an empirical model is that weather data is widely collected and recorded which provides large data sets to train and validate a model. The data used for training the models in this work are from multiple years and only includes summer months. A forecast model is created for every time point in the MPC horizon. For example, if the MPC is optimizing a point 12 hours in the future, there is a model developed specifically for predicting the ambient temperature 12 hours in the future. This is found to be most effective because the data used to train the model had varying influences. For example, if the ambient temperature ten minutes from now is predicted, then the p-values for the data within the last hour are found to be the only statistically significant variables. However, 24 hour predictions require the use of a model that contains temperature data from hours to several days in the past according to the variable p-values in the regression model. The models for predicting the temperature 10 minutes and 24 hours ahead are discussed in more detail below. Other models are developed for each of the remaining MPC horizon points, but they are not included because the results are similar to the 10 minute and 24 hour models.

\[
T_{\text{future}} = \text{Intercept} + \sum_{i} \alpha_i \hat{T}_a + \sum_{i}^{24} \beta_i Hr_i
\]  

(2.19)

The general form of the models is seen in Equation 2.19 as a time series model. The inputs to the model are \( \hat{T}_a \) which is an array of historical temperature data and \( Hr \) which is an array to indicate the hour of the day corresponding to the point being predicted. As mentioned above, the historical data is reduced down to the data that best captures the trends. This process is done by keeping the variables with significant p-values. The historical data array varies in size depending...
on how many parameters are found to be significant. The parameters are fit using linear regression in Python with the scipy.minimize package.

Figure 2.5 shows the actual value versus the predicted value for ambient temperature 10 minutes ahead. The $R^2$ value is greater than 0.99 and the root mean square error is 0.0499. An accurate 10 minute forecast is important for near-term directional weather predictions, but does not give the optimizer enough data about the system to be able store energy hours in advance if needed. The longer predictions are more critical for exploiting time-of-day pricing and making long-term proactive decisions.

![Figure 2.5: Actual vs. Predicted for Ambient Temperature 10 minutes ahead](image-url)
Figure 2.6 shows the result of the model that predicts temperature 24 hours in the future. The fit matches the model well with an $R^2$ value of 0.85 and a root mean square error of 1.845. These values along with Figure 2.6 indicate that the model is valid and can be used for optimization. It is important to notice that predictions are more uncertain further into the future as explained by the figures and statistical values.

Figure 2.6: Actual vs. Predicted for Ambient Temperature 24 hours ahead

These models are used to send forecasting data to the MPC application. Using accurate models allows the MPC application to better predict the status of the home in the future and al-
allows the optimizer to proactively manage the energy flows to accommodate changes in ambient temperature.

2.8 Miscellaneous Building Power Prediction Model

Another important process variable to predict is the miscellaneous building power requirement. Miscellaneous building power is a variable inside EnergyPlus that accounts for all of the building power usage minus what is needed for the HVAC system. This includes the power used by lights, appliances, and electronics plugged into outlets. Although the individual components mentioned may have very sporadic usage and trends, the combination of all of them into one variable stabilizes the dynamics into repetitive trends. It is important to be able to predict this value because it has an impact on the energy demand of the home at different times throughout the day and affects how the HEMS optimizes the energy flow in the home.

An empirical model is fit to EnergyPlus simulation data which provide large data sets that are used to train and validate the model. As with the ambient temperature forecasting, a model is created for each time-step of the MPC application. The general model equations are also very similar to Equation 2.19, but with one major difference as shown in Equation 2.20.

\[
Emisc_{\text{future}} = \text{Intercept} + Emisc_{\text{current}} + \sum_{i} \alpha_i Hr_i
\]  

(2.20)

Instead of using historical data to predict the future value, the model only uses the current miscellaneous power value and the hour indicator value. These are determined by examining the p-values associated to each variable included in the historical data sets and only keeping ones that are statistically significant. The prediction model for predicting 10 minutes and 24 hours in advance are discussed in greater detail below.

The results for the 10 minutes ahead model are seen in Figure 2.7. The majority of the points are close to the identity line, but there are a few outliers. This is explained by the sporadic use of appliances and outlets. The fit has an \(R^2\) value of 0.948 and a root mean square error of 57.7 Watts. This indicates that the fit does well at representing the majority of data but some of the data points deviate from the model fit. Air conditioning loads are around 5,000 to 7,000 Watts so a root mean square error of 57.7 Watts has a negligible effect on the overall optimization.
The results of the model predicting 24 hours in advance are shown below in Figure 2.8. The fit has an $R^2$ value of 0.885 and a root mean square error of 86.1. Both values are worse compared to the 10 minutes ahead model. This is most likely explained because the model has to extrapolate further from the training data. An interesting observation to make is that there are a large number of outliers below the identity line, indicating that those points in the model under-predict the amount of power usage. This negatively affects the HEMS because it does not account for the correct amount of energy consumption at those time steps which results in model mismatch. Model mismatch leads to sub optimal solutions and reduced cost savings.
The models discussed above allow the MPC application to gain information about outlet and appliance power usage. The MPC is able to make informed decisions based off of these energy flows and is able to more fully optimize the system.

2.9 Solar Power Production Prediction Model

Solar Power production varies in magnitude and timing throughout the day. The power production is largely influenced by the size of generation system and the time of day. Properly sized equipment has the potential to eliminate the need to buy electricity from the grid if managed appropriately. As a result, it is crucial that this variable is predicted as accurately as possible to
enable maximum energy management benefits. The models created in this work are empirical models trained and validated by EnergyPlus simulation data. The general model equation takes a similar form to the ambient temperature model. The general model equation is seen in Equation 2.21.

$$P_{solar\ future} = \sum_{i} P_{\alpha_i} H_{r_i} \quad (2.21)$$

Figure 2.9: Actual vs. Predicted for Solar Power Production 10 minutes ahead

The major difference between the solar power production model and the ambient temperature model is the historical data inputs. Solar power production relies on the surface area of the
solar panels and the amount of solar energy. In this work the solar generation system size is a constant. The amount of light received from the sun varies but follows a diurnal trend. As a result, the model equation only needs the time of day to efficiently predict the maximum amount of generated power. This conclusion is also confirmed by looking at the p-values of each potential parameter used to train the regression model. The prediction results for ten minute and twenty-four hour forecasts are discussed in detail below.

Figure 2.10: Actual vs. Predicted for Solar Power Production 24 hours ahead
Figure 2.9 shows the result of the model fit for the 10 minutes ahead predictions. The fit has an $R^2$ value greater than .99 and a root mean square error of 33.5 W. The $R^2$ value and the graphical fit in Figure 2.9 indicate a good model fit.

The results of the model predicting 24 hours in advance are shown in Figure 2.10. These results have an $R^2$ value of .978 and a root mean square error of 183.4 Watts. Compared to the 10 min ahead model, this fit is worse and has a larger spread from the actual values. This is due to the outliers seen in Figure 2.10. This model is trained on more than 54,000 data points and the majority of the points lie along the line which give the high $R^2$ but the increased number of outliers cause a larger root mean square error. As with the other prediction models, this is explained by the predicted points being extrapolated further away from the training data.

The models developed for predicting solar power production are simple in that they only have one variable but they fit the data well based off of the statistical values and graphical fit. The MPC application uses these predictions to proactively reduce or shift energy flows to maintain home temperature comfort and lower energy costs.

2.10 Moving Horizon Estimation and Model Predictive Control Theory

MHE is an established parameter estimation method that utilizes dynamic optimization over a fixed time horizon of recent measurements to regress current model parameters [31]. This work utilizes the GEKKO optimization suite [32] for MHE to regress system parameters. Two common MHE objective function forms are the least squares objective and $l_1$-norm objective. The least squares objective is influenced by bad data and outliers. The home system may experience issues with the sensors or incorrect data recording so the $l_1$-norm objective is used instead to improve parameters estimates while eliminating outliers and measurement noise [31]. The general form of the MHE $l_1$-norm objective is shown in Equations 2.22 - 2.30. The nomenclature for these equations are listed in Table 2.3. The $l_1$-norm objective uses a dead-band to avoid over-fitting the parameters to noise. The $l_1$-norm objective also has other benefits such as a penalty for manipulating parameters when there is little or no benefit. MHE only adjusts the model parameters if they are outside of the dead-band. This is particularly helpful if a system is at a steady state, when the parameters are weakly observable, or when there is no information content in the data.
that would necessitate a parameter adjustment.

\[
\begin{align*}
\min_{x,y,p,d} \mathbf{\Phi} &= W_{m}^{T}(e_{U} + e_{L}) + W_{p}^{T}(c_{U} + c_{L}) + \Delta p^{T}c_{\Delta p} \\
\text{s.t.} \quad 0 &= f \left( \frac{dx}{dt}, x, y, p, d, u \right) \\
0 &= g(x, y, p, d, u) \\
0 &\leq h(x, y, p, d, u) \\
e_{U} &\geq y - y_{x} + \frac{d_{b}}{2} \\
e_{L} &\geq y_{x} - \frac{d_{b}}{2} - y \\
c_{U} &\geq y - \hat{y} \\
c_{L} &\geq \hat{y} - y \\
0 &\geq e_{u}, \quad e_{L}, \quad c_{U}, \quad c_{L}
\end{align*}
\]

\[
\begin{align*}
\min_{x,y,p,d} \mathbf{\Phi} &= W_{\text{hi}}^{T}e_{\text{hi}} + W_{\text{lo}}^{T}e_{\text{lo}} + y^{T}c_{y} + u^{T}c_{u} + \Delta u^{T}c_{\Delta u} \\
\text{s.t.} \quad 0 &= f \left( \frac{dx}{dt}, x, y, p, d, u \right) \\
0 &= g(x, y, p, d, u) \\
0 &\leq h(x, y, p, d, u) \\
\tau_{c} \frac{dy_{t,\text{hi}}}{dt} + y_{t,\text{hi}} &= sp_{\text{hi}} \\
\tau_{c} \frac{dy_{t,\text{lo}}}{dt} + y_{t,\text{lo}} &= sp_{\text{lo}} \\
e_{\text{hi}} &\geq y - y_{t,\text{hi}} \\
e_{\text{lo}} &\geq y_{t,\text{lo}} - y
\end{align*}
\]

MHE is often utilized in conjunction with MPC, which uses the current system parameters regressed by MHE to predict future values given a set of control moves [31]. Dynamic Optimization, MPC, and MHE have wide application across a broad range of industries including continuous chemical process optimization [52–54], cryogenic carbon capture [55–59], energy system capacity
planning [60], and drilling automation [61–63]. The optimal control over the future prediction horizon is determined by dynamic optimization. This work utilizes a MHE to determine current building model parameters to use with MPC. In conjunction with weather forecasting, the estimation provides a higher level of accuracy in the predictive control and is a contribution of this work. While these approaches are not individually new, the combination of all elements in a real-time demonstration with a high-fidelity simulator is novel.

Table 2.3: Objective Function Terms from [31] for MHE and MPC

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Φ</td>
<td>objective function</td>
</tr>
<tr>
<td>yx</td>
<td>measurements $(y_{x0}, ..., y_{zn})^T$</td>
</tr>
<tr>
<td>y</td>
<td>model values $(y_0, ..., y_n)^T$</td>
</tr>
<tr>
<td>wm, Wm</td>
<td>measurement deviation penalty</td>
</tr>
<tr>
<td>wp, Wp</td>
<td>penalty from the prior solution</td>
</tr>
<tr>
<td>cΔp</td>
<td>penalty from the prior parameter values</td>
</tr>
<tr>
<td>db</td>
<td>dead-band for noise rejection</td>
</tr>
<tr>
<td>x, u, p, d</td>
<td>states $(x)$, inputs $(u)$, parameters $(p)$, or disturbances $(d)$</td>
</tr>
<tr>
<td>Δp</td>
<td>change in parameters</td>
</tr>
<tr>
<td>f, g, h</td>
<td>equation residuals, output fraction, and inequality constraints</td>
</tr>
<tr>
<td>eu, el</td>
<td>slack variable above and below dead-band measurement</td>
</tr>
<tr>
<td>cU, cL</td>
<td>slack variable above and below a previous model value</td>
</tr>
<tr>
<td>y1, y1hi, y1lo</td>
<td>desired trajectory target or dead-band</td>
</tr>
<tr>
<td>Whi, Wlo</td>
<td>penalty outside trajectory dead-band</td>
</tr>
<tr>
<td>cy, cu, cΔu</td>
<td>cost of $y$, $u$ and $Δu$, respectively</td>
</tr>
<tr>
<td>τc</td>
<td>time constant of desired controlled variable response</td>
</tr>
<tr>
<td>e1lo, e1hi</td>
<td>slack variable below or above the trajectory dead-band</td>
</tr>
<tr>
<td>Sp, Sp1lo, Sp1hi</td>
<td>target, lower, and upper bounds to final set point dead-band</td>
</tr>
</tbody>
</table>
Model predictive control seeks to optimize an objective such as minimizing energy consumption or maximizing profit by manipulating controllable system variables at discrete time points in a future time horizon. Dynamic optimization uses a system model, which in this work is an online MHE model. The MPC application in this work uses the control $l_1$-norm objective function. The $l_1$-norm objective form is seen in Equations 2.31 - 2.38. Terms in these equations are listed in Table 2.3. The control $l_1$-norm objective shares many benefits as discussed above in the MHE $l_1$-norm discussion. The major difference is that instead of bad data and outlier rejection, the objective form is able to add priority to different objective functions. This permits multiple objectives which are solved simultaneously in one optimization problem. That is essential for this work because cost and comfort are both objectives and are prioritized with a hierarchy.

2.11 Moving Horizon Estimation with Lumped Parameters

A MHE is used to calculate the model parameters for the home. MHE is used because the model is updated to available data and accounts for changes made to the home. MHE optimally adjusts a model to align with data, but has not been commonly implemented in the process of modeling and optimizing in HEMS. Most home models involve complex systems of equations or complex models that have been trained on large data sets. Those models are often accurate, but require substantial computational time to develop and solve. This is a major constraint for HEMS and is why this work focuses on simplifying the model so that it can be manipulated and solved in real-time while maintaining as much accuracy as possible. The model equations are listed above in section 2.4.

The MHE horizon is 36 hours long with a one minute time step. This allows the estimator to fit the model to trends that happen daily while also capturing the short-term dynamics of heat transfer through the walls.

Figure 2.11 shows the MHE solution of the system when the MPC is not active. Values for $T_{\text{air,ambient}}$, $T_{\text{air,inside}}$, and $Q_{\text{HVAC}}$ are obtained from the EnergyPlus simulator. During this period of time, a thermostat is controlling the temperature with a dead-band. If the temperature goes above the set-point, the air conditioner turns on until it reaches the lower set-point and turns off. This explains the cyclic behavior seen in Figure 2.11.
The subplot on the top displays the temperatures of the ambient air, set-points, the air inside the home, and the predicted temperature from the MHE model. The second subplot displays when the air conditioner is on or off and the power consumption of the air conditioner. This plot shows that the MHE predictions are correct in matching the actual indoor temperature. This model accuracy is essential for the rest of the energy management software because the air conditioner has such a large influence on power consumption.

As discussed in the building house model section, the original model does not extrapolate when the air conditioner is on for longer periods of time. Consequently, a new model is created based off of the same principles as the initial model but with extra terms added to account for model mismatch error. These new terms account for the heat transfer from the ground and the air conditioner on/off status changes. The results of the improved model are shown in Figure 2.12.

The model fit is less exact compared to the first, however, it is able to represent the air conditioning system within a reasonable amount of error. The simulation results in Figure 2.12 are less cyclic and show the air conditioner behavior outside of the normal cooling cycles. This development is important for the HEMS because there are times where it is beneficial to keep the air conditioner on for extended periods and break the cyclic behavior.
One of the benefits of MHE is that when new data is available, the solution time shifts to the next solve time and uses the previous solution to initialize the next solve (i.e. Bayesian approach). This allows the system to be updated continuously while also keeping the solve times short which has been a significant barrier in the HEMS research.

Another important factor in keeping the solution times short is the use of scaling in the solver. Scaling is crucial when solving this model because the magnitudes of the parameters differ on the magnitude of $1e7$ or greater. This causes the solver to take longer to converge and potentially fail.

### 2.12 Model Predictive Controller with Forecasts

MPC is used to optimize and make changes to the system. MPC has been used in HEMS before, but hasn’t included the combination of weather forecasts, market forecasts, and an economic objective function while also being solved in synchronization with an MHE application. Due to the simplicity of the MHE model described above, the MPC model consequently remains simple. This is important because, as stated earlier, models used in HEMS can be very complex and can require long computational times to solve.
One of the first things to consider when building an MPC application is the horizon length. In this work, a horizon length of 24 hours is chosen. A horizon of this length allows the optimizer to more efficiently react to changes in ambient temperature, solar power production, changes in electricity prices, and captures one full diurnal cycle.

An additional part of the MPC is the home air conditioning model. Section 2.4 discusses the home model equations and Section 2.11 discusses the estimation of the home model parameters. Together these combine to form the complete home model which is used in the MPC. Every time the MPC solves, the new parameter solutions from the MHE are sent to the MPC to update the model. This updated model is important for a successful MPC because it allows the controller to know how the ambient temperature and air conditioner affect the temperature inside the home. These relationships also allow the MPC to implement comfort into the objective function. This is accomplished by setting high and low temperature set-points. The priority of this objective function is manipulated to achieve different results. If the priority is high, then the comfort is be the main goal of the optimizer. However, if the priority is reduced, then the optimizer makes trade-offs between saving money and reducing the occupants comfort.

Another aspect of building the MPC model is the air conditioner on/off behavior. Most air conditioners have set speeds. For example, a one-speed air conditioner is only on or off. Adding a discrete on off variable is difficult to optimize because it is a non-continuous function which is hard for solvers to handle and are computationally expensive. Even with these drawbacks, it is crucial that the MPC model follows the on off behavior as closely as possible. In an effort to avoid discrete variables, a hybrid approach is used in this study.

The key to the hybrid approach is to keep the $HVAC_{I/0}$ variable continuous, but to force it to either be one or zero. Equation 2.39 and Figure 2.13 show a function that achieves this goal. In the equation, the parameter $A$ is be adjusted to change the height of the parabola. In Figure 2.13, several different values of $A$ are plotted. As $A$ decreases, the height of the parabola decreases as well. This is important because Equation 2.39 is setting this function to be less than or equal to zero. With $HVAC_{I/0}$ being bounded at zero and one, this expression is only valid when $HVAC_{I/0}$ equals zero or one and therefore, giving us the discrete-like behavior needed. As a result, if $A$ is too large, then the height of the parabola is too high and the optimizer gets stuck at the value of zero or one. Alternatively, if $A$ is too low, then the function fails to constrain $HVAC_{I/0}$ and it is allowed
to be any value between one and zero. $A$ is consequently optimized to be at the value where it constrains $HVAC_{I/0}$ to zero or one and so that optimizer effectively explores the design space.

$$0 \geq -A \left( HVAC_{I/0} - 0.5 \right)^2 + \frac{A}{4}$$

(2.39)

![Air Conditioning Function](image)

**Figure 2.13: Air Conditioning Function**

In the process of developing the above function, the first idea explored was to add the constraint into the objective function so that the right side of Equation 2.39 would be minimized in the objective function. This minimized the function successfully, but dominated the other objectives in the optimization. This caused the results to be sub optimal with respect to optimal energy con-
sumption. As a result, Equation 2.39 was developed to use MPCC’s which achieves the same goal without affecting the comfort and economic objective functions.

The next component of the MPC is the battery model. The battery model is discussed in greater detail in Section 2.6. This battery model is used in MPC for several reasons. The first is that the battery model is continuous and continuously differentiable. Battery models are often undefined when the state of charge reaches zero or one. This model, however, is continuous at zero, one, and in-between. The second reason is because the model simplifies a larger battery into a system of smaller battery cells. This enables the user to change out the battery specifications inside the MPC application.

At this point, the MPC application has a model of the house as well as a battery model. Before any optimization happens, these models need to be correlated. This is done through a system of energy balances.

The first of these equations is a calculation of the total power demand as seen in Equation 2.40. Demand is a positive or negative quantity depending on how the quantity solar production compares to the house energy loads.

\[
\text{Demand} = P_{\text{misc}} + P_{\text{HVAC}} - P_{\text{solar}} 
\]  

(2.40)

If the demand is negative, then the power is either sold back to the grid (if allowed) or put in the battery to charge it. Conversely, if the demand is positive, then the house needs to be supplied with power purchased from the grid or discharge the battery. The cost of buying electricity and the amount gained from selling electricity to the grid is often different. Consequently, the energy balance treats the energy sold and the energy purchased as two different variables instead of just one that is positive or negative. This adds a slight complication because power is only bought or sold but both should not happen at the same time. This requires model constraints. Slack variables are added to handle these complementarity constraints. The switching conditions are displayed below in Equations 2.41 - 2.45.

\[
\text{Grid}_{\text{buy}} + P_{\text{battery}} - \text{Grid}_{\text{sell}} \geq \text{Demand} 
\]  

(2.41)

\[
\text{Grid}_{\text{buy}} = \text{Demand} - P_{\text{battery}} + SV_1 
\]  

(2.42)
\[ Grid_{sell} = P_{battery} - Demand + SV_2 \]  
\[ P_{battery} - Demand = SV_1 - SV_2 \]  
\[ SV_1 \cdot SV_2 \leq 0 \]

Although, the equations look simple, they accomplish a complex goal. Equation 2.41 ensures that the power difference between what is purchased or sold from the grid and charged or discharged from the battery is always greater than the demand requirement. Equations 2.42 - 2.45 constrain the model to only buy or sell electricity. The first equation to consider is Equation 2.45 which forces the product of the two-state variable to be less than or equal to zero. This is important for Equation 2.44 because it forces one to be zero if the other is non-zero. For example, if \( P_{battery} - Demand \) is positive, then \( SV_1 = P_{battery} - Demand \) and \( SV_2 \) has to equal 0 to obey the constraint in Equation 2.45. This observation is critical when looking at Equations 2.42 and 2.43. To continue with the example, if \( SV_1 = P_{battery} - Demand \), then Equation 2.42 reduces down to \( Grid_{buy} = 0 \). Alternatively, if \( SV_2 = 0 \), then Equation 2.43 reduces down to \( Grid_{sell} = P_{battery} - Demand \). Therefore, the model is successfully constrained to only buy or sell electricity. As a last check of consistency, \( P_{battery} - Demand \) is set to be positive, indicating that the power in the system is greater than what is required by the home. As a result, this extra power is sold back to the grid which is explained by Equation 2.43 because it reduces down to \( Grid_{sell} = P_{battery} - Demand \).

Lastly, the end goal of the MPC application in the HEMS is to reduce costs and increase revenue if possible. To do this, the model needs to relate energy to a dollar amount as seen in Equations 2.46 - 2.48.

\[ Revenue = Grid_{sell} \cdot Price_{Net-metering} \]  
\[ Cost = Grid_{buy} \cdot Price_{electricity} \]  
\[ Profit = Revenue - Cost \]
Figure 2.14: MPC sample Solution
Equation 2.48 brings all of the MPC models together. The final addition in the MPC is to maximize Profit (e.g. minimize cost). As stated above, the priority of the objective function components are adjusted to accomplish different levels of comfort and savings.

All of the discussion to this point is enough for the optimizer to make improvements to the system. However, one goal of the energy management system is to make proactive changes to account for weather and market forecasts. This is accomplished by using the forecasting models discussed in the above sections to inform the MPC about changes to variables in the future. Before every new MPC solve, the forecasting models calculate new predictions which are then used by the MPC application to make proactive optimization decisions.

With any optimization, it is important to validate the models. Validation typically is done by comparing results to experimental data. Experimental data is not readily available, so validation will be done with the use of first principles knowledge. The above results show a few indications that the model used is working. First, when the air conditioner is on, the house temperature drops. Second, the price of electricity increases for a few hours during the MPC horizon and during this period, the MPC uses the air conditioning as little as possible in an effort to minimize costs. Lastly, when the amount of solar power production is greater than the demand of the home, the extra energy from the system is being sent into the battery and the SOC increases.

An example MPC solution is shown in Figure 2.14. The first subplot displays the predicted temperatures from the house air conditioning model. The second subplot shows when the air conditioner is on or off, as well as the battery state of charge. In the third subplot, the pricing schemes for buying and selling electricity are shown. Lastly, the fourth subplot shows the predicted solar power produced and the predicted house power demand.
CHAPTER 3. CASE STUDIES

The following case study shows the results of applying the HEMS to manage the energy in a home with battery storage and a PV system. These results are then compared to identical homes and climates so that the improvements are quantified.

Sections 3.1 - 3.3 show the results of a one day simulation of a 3350 square foot home in Phoenix, Arizona. Appendix B contains the simulation and optimization code and Appendix C contains the EnergyPlus home configuration .idf file. This house is equipped with a 5.0 kW crystalline silicon rooftop PV system, a 14,000 kWh battery based on commercially available home batteries, and a 5 ton central air conditioning system. Other home properties are based on a typical home and are described in greater detail in the configuration file found in Appendix C.

3.1 Base Case

Before the results of the HEMS are discussed, it is important to first look at the case where no energy management software is present.

The first plot shows the ambient temperature, air temperature inside the home, and the temperature set points. As expected, the air conditioner goes through repetitive cycles to maintain the temperature inside the home. During the hottest period of the day, the air conditioner has to stay on for a long period to maintain a comfortable atmosphere. This behavior is also seen in the second subplot which shows the status of the HVAC system. The third subplot shows the pricing structure for purchasing energy and for selling it through net metering. In the last subplot, energy flows are shown for the home. This includes the production of solar power, the amount of energy purchased from the grid and the amount of energy sent back to the grid. Energy on the positive half of the plot help reduce the optimizer objective function whereas the negative half will increase the objective function.
Figure 3.1: Phoenix, AZ house with no optimization
The first important trend to notice is that purchasing price is at its highest when the air conditioner runs the most. This is caused because energy prices are set higher during peak hours of the day due to the increase in demand. Therefore the energy load is what ultimately drives the prices. Successful energy management software should shift or lower the load peak so that penalty for buying energy during high prices is minimized. This is accomplished by pre-cooling the home or using energy storage to run the air conditioner.

Another important trend to recognize is that the solar power production follows the same pattern as the changes in ambient temperature. Energy demand is usually at its highest during the hottest time of the day, however with a solar kit, energy demand can be lowered significantly. Is is particularly important for a house with no optimization because it helps lower energy costs without the need of process decisions. An optimizer however, should be able to manage the energy production more effectively with the use of energy storage.

3.2 Optimized Case

Once the base case is understood, it is easier to realize the improvements achieved due to the HEMS. Figure 3.2 below shows the plots of the optimized home. The simulation inputs are identical to those used in the base case. This means the house, weather data, occupancy data, and appliance schedules are all identical. The main difference is that the optimized case has the option to use a battery to store and discharge energy. To make sure that the net energy in both cases are equal, the battery is constrained to start and end with the simulation day with the same SOC.

The first subplot in Figure 3.2 shows the actual ambient temperature and temperature inside the home. It shows that the temperature in the home stays near the comfort range chosen by the user which is critical to the overall success of the HEMS. It is important to notice that around 6 pm, the house temperature actually goes beyond the high comfort set point. This results from balancing the multi-objective optimization problem. In this situation there is heavy weighting on the economic objective which allows the house temperature to rise above the set point. The weight is adjusted to keep the temperature below this limit if that is the desired result.
Figure 3.2: Optimized House in Phoenix, AZ
The second subplot of 3.2, shows the status of the air conditioner and the battery. This subplot is significantly different compared to Figure 3.1. In the base case, the air conditioner cycles on and off regularly. The optimized case has a more methodical approach to turning the air conditioner on and off. The main reason for this change is that a typical thermostat operates between a cooling temperature and a shutoff temperature whereas the optimized case has the freedom to move freely as long as it is within the comfort range set by the user. As a result the air conditioner can turn on for longer periods of time and eliminate unnecessary cycling. Another difference is that in the optimized case, the air conditioner is scheduled to come on at better times. The price of electricity increases at 2 pm and as a result the optimizer starts to proactively cool the home to avoid the additional costs.

SOC of the home battery system is also displayed on the second subplot of 3.2. The battery remains relatively inactive until the solar panels begin to produce power. At that point the battery builds up its charge until the increase in price for electricity. If the air conditioner needs to be run during this high price period, then the battery discharges to lower the demand require by the grid. This allows the system to stay in the comfort limits at all times of the day while also reducing the cost of electricity.

Lastly, there are important improvements seen in the fourth subplot in Figure 3.2. This plot shows the evidence that system changes make a difference. The power needed from the grid is not completely mitigated but it is reduced compared to the base case. The amount of energy sold back to the grid in both cases differs by only 1.21% different.

These results indicate that the savings come from scheduling appliances and reducing consumption rather than selling energy back to the grid for profit. This result may differ if the pricing structures changes, allowing the net metering to be more profitable.

### 3.2.1 Comparison to Base Case

As with any optimization problem, it is important to be able to quantify the improvements. The two competing objectives in the HEMS presented in this work are comfort and the cost of electricity. The previous sections show that the comfort constraints are maintained but give little information about the actual economic improvements. The cost of electricity for the base case simulation day is $5.12. On the other hand the cost of electricity for the optimized energy system
is $3.05. That equates to a savings of $2.07 or a 40% decrease in energy cost. The optimizer also reduces the number of times the air conditioner unit cycles on/off which decreases the amount of damage to the system. Decreasing damage will lower long-term maintenance costs, however this is not explored in detail in this work. This is a significant improvement and shows that there is value in using a proactive energy management system to handle residential energy usage in Phoenix, Arizona.

These results are from a simulation run on one of the hottest days of the year in Phoenix. The results during other times of the year will vary. Phoenix, Arizona is warm year round which suggests that these improvements could be beneficial year round.

3.3 MPC only HEMS Case study

To show the benefits of combining MHE and MPC, a simulation is run with the MHE model updates turned off. The MHE application is on for one hour before the MPC application turns on so that model parameter estimations are stable enough to be used in the MPC application. After the one hour, the MHE parameter estimates are held constant and remain static within the MPC application. The MPC uses those estimates for the remainder of the day to optimize the home energy system. The results of this simulation are shown in Figure 3.3.

The temperature subplot shows that the house temperature fluctuates often. This is due to the optimizer turning the air conditioner on and off repeatedly, and consequently, increases the amount of power consumption. The last subplot of Figure 3.3 has a significant amount of red compared to the green which indicates that there is more power being consumed than is being generated or saved. The optimizer is converging to a solution but it is worse than if no optimization occurred at all due to model mismatch. Using a dynamic model that adjusts through MHE allows the model mismatch to be minimized and consequently allows the optimizer to reach an improved optimal solution as discussed in the following section.
Figure 3.3: Optimized House in Phoenix, AZ
3.3.1 Comparison to full HEMS application

To put the effects of removing the MHE application from the HEMS, it is helpful to compare the costs in the optimized and base cases. The HEMS without MHE increased the cost by 155% when compared to the HEMS with both MHE and MPC. Additionally, removing the MHE has an adverse effect on the cost savings that it increases costs by 41% when comparing it to the base case.

These results are convincing that aligning the model parameters to the simulated home model is critical to the HEMS developed in this work. As discussed in the Introduction, there are HEMS applications that rely solely on MPC to optimize a home. This is done by developing a robust and comprehensive model. However, these models are more computationally expensive to optimize. This work uses a model that uses the benefits of MHE and MPC which allows the model to remain simple. This is accomplished as the MHE application dynamically changes the model parameters as the system changes. There are many thermodynamic properties that are temperature dependent such as density, heat capacity, etc. As the temperature in the home and the air outside the home change, these parameters also change. By using MHE, these changes are handled appropriately by making new parameter estimations. For the MPC only HEMS, equations would need to be included in the model to help account for these dynamics which explains the additional model complexity.

Another reason MHE is critical to the HEMS in this work is that it estimates parameters based on the system changes made by the MPC. When the MPC optimizes the system, changes to the system may be made. These changes cannot always be predicted and vary from time to time. The MHE application is running in synchronization with the MPC, so it adjusts to the model to match in real-time to what the changes made by the MPC.
CHAPTER 4. CONCLUSIONS AND FUTURE WORK

This work demonstrates the ability to use MPC and MHE as a home energy management system in residential homes. The home energy management approach combines weather forecasts, energy consumption forecasts, solar production forecasts, a battery, an air conditioner, and house thermodynamics. All of these models are combined to be used in closed loop optimization. MHE estimates uncertain house thermodynamic and air conditioner model parameters. The MHE application is developed to be able to solve for a variety of different homes and air conditioner units allowing for a broad range of applications. This minimizes the time needed to fit the models to every individual house. This work reaches optimal operation by considering all major elements of a home energy system simultaneously. Lower energy costs and improved grid energy utilization are achieved by automatically controlling air conditioner and battery usage.

Another innovation of this work is the use of MPCCs to model the on/off behavior of an air conditioning unit, allowing the system to be solved with gradient-based solvers. In most cases, gradient-based solvers are faster and more reliable than genetic solving algorithms or mixed-integer methods. Genetic solving algorithms are commonly used when solving a model with one or more discrete variables. This work shows that by using the MPCCs to change the on/off behavior from discrete to continuous, dynamic optimization is used to control a home energy system in close to real-time speeds.

EnergyPlus is used to simulate a 3350 square foot home in Phoenix, Arizona. The initial test uses a typical temperature control scheme with no energy management. Another test is run which implements the HEMS developed in this work. Results of the comparison between these two tests show a cost and energy reduction of 40% and 21% respectively. These improvements have a positive impact on grid stability and peak load because the energy consumption is spread out more evenly throughout the day.
Computational speed and model accuracy is also considered in this work. For the above case study, there are 23,749 equations and 4 degrees of freedom for every MHE solution. The problem is large due to the 36 hour time horizon. The average solution takes 2.755 clock seconds and around 5 iterations to solve. These results are a direct impact of a simple model. The MPC application is much more complex due to the additional models and model interactions. The MPC has 10,656 equations and 11,231 variables to optimize over the 24 hour horizon. The initial MPC solution time is 776 clock seconds and 3954 iterations. Subsequent MPC solutions average 318 seconds and 1735 iterations. There is a major difference because the first solution has to completely initialize and subsequent solutions use the previous solution to initialize the optimizer and converge to the new solution about 59% faster.

Lastly, the benefits of including both MHE and MPC in the HEMS are analyzed. For each MHE plus MPC solution it takes roughly 5.3 clock minutes. Each solution time step is 10 minutes apart so there is sufficient amount of time for HEMS application to run before the next system change. Using only MPC in the HEMS with the models developed in this work increase the costs of electricity by 41% compared to the base case or 155% when compared to the HEMS that includes both MHE and MPC. Using both MHE and MPC allows models to be simple while also flexible to adjust to changes to the system. Using this approach allows HEMS applications to solve quickly and retain enough accuracy to decrease electricity costs.

4.1 Future work

Future work should expand these methods to include the following: 1) additional testing in different climates and seasons, 2) additional testing on various pricing schemes, 3) more elaborate forecasting models by including additional parameters and non-linear terms, and 4) ways to investigate and improve solve time and solver reliability to access the viability of using on an actual home energy system.

This work tests the optimization software on a residential home in Phoenix, Arizona. Future work should explore the benefits of applying the home energy management software developed in this work to different climates. Phoenix, Arizona is warm year-round and requires significant use of an air conditioner. Other climates have a smaller dependency on air conditioners which affects the total effectiveness of the software. This testing should also expand to testing during different
seasons throughout the year for similar reasons. Future work can also benefit from incorporating a furnace model for colder climates so the energy consumption can be optimized for all weather conditions.

Pricing schemes vary based on location and utility. This work focuses on a basic on-peak, off-peak pricing structure. Future work should explore real-time pricing (RTP) schemes, more complex (TOU) schemes, and other pricing schemes to see which schemes are more conducive for the home energy management system efficiency. The research can be expanded to develop a more optimal pricing scheme that could be presented to utility companies.

The forecasting models in this work are accurate, but basic and location-specific. For example, this work trains the forecasting models to data specific to Phoenix, Arizona. Although these models can be re-trained for new locations, it is a tedious and time-consuming task. Future work can develop these forecasts to be more robust and flexible which would allow this software to be easily implemented into wider range of homes and locations.

Finally, these results need to be implemented and tested on a real home in real-time. Currently, these optimization algorithms have only been tested on a simulated house with a simulated time frame. Future work should expand the study to test the energy management system in an actual home to see if the results can produce similar findings to those found during simulation. Field testing would help determine if the energy management system developed in this work is accurate and flexible enough to be implemented into real residential energy systems.
REFERENCES

Accessed: 06-12-2017. 1

1

energy storage system over a 24 hour period using weather forecasts.” In 2013 American
Control Conference, pp. 2946–2951. 2

overview on the data driven and large scale based approaches for forecasting of building

solutions.” In 2012 IEEE 5th India International Conference on Power Electronics (IICPE),
pp. 1–6. 2

ials for subzero applications.” Energy, 51, pp. 1–17. 2

ing, dynamic pricing and dsm in smart grid: A review.” Renewable and Sustainable Energy
Reviews, 54, pp. 1311–1322. 3

ings based on dynamic price of electricity.” Energy and Buildings, 80, pp. 531–541. 3, 4

energy management systems with photovoltaics and batteries.” The Electricity Journal, 32(1),
pp. 6 – 12. 3

with renewable energy sources : A review.” Renewable and Sustainable Energy Reviews, 18,
pp. 64–72. 3

enablers of demand response in the smart grid.” Renewable and Sustainable Energy Reviews,
72(November 2016), pp. 57–72. 3


Appendix A. Software Interfacing for Advanced Control

A.1 Software Interfacing for Advanced Control

One practical hurdle for building energy optimization is software interfacing with building energy simulators for testing. A Python script was developed for this work. The first step of the process produces an XML file that contains the building information. BEopt has a graphical user interface (GUI) for creating residential building models. However, it is beneficial to be able to modify the XML files programmatically. In the BEopt installation file, a Python program called create_xml_file.py is included which accepts arguments that will allow the user to modify an XML file with new building parameters. The next step in the process is to produce an IDF file which is the file that EnergyPlus uses to run the simulation. The BEopt installation also comes with a program that assists in this process. The XML file is the input and the user specifies the output location of the newly created IDF file.

At this point EnergyPlus has the appropriate file to run. However, there are additional ways to improve the interaction with the simulator. The IDF file contains schedules of when the appliances in the home are used, temperature set-points, and many more parameters to inform the simulator of what to do. Therefore, it is crucial that an optimizer can manipulate these. This can be done by setting up the IDF file to receive external changes. There are two major steps to accomplish this. The first is to modify and map the IDF file to understand external commands. EPPY is a python package developed just for this task. With this package, the user can set up the variables for the simulator to output to the optimizer, as well as set up the variables that the optimizer will input back into EnergyPlus. The exact details of this procedure can be found in the source code, as well as BEopt documentation.

The last step of the process is to interact with the simulation program. Fortunately, there is another Python package developed for this task called pyEp. This package was developed to allow Python programs to access the simulation outputs setup in the last step. It also allows users to
send back inputs to the simulator, allowing the user or optimizer to make changes to temperature set-points or appliance schedules.

With the combination of all of these tools, a program can be created to fully interact with the EnergyPlus simulator. These developments are crucial to allow an energy management software to interface with the simulations. It also gives large amount of cases and data to run and test the software to ensure proper behavior.
APPENDIX B. SOURCE CODE

B.1 Source Code

```python
import pyEp
import os
import matplotlib.pyplot as plt
from matplotlib import dates
import time
import numpy as np
import pandas as pd
import datet ime
from gekko import GEKKO
import initialize_sim_files
import APS_data as APS
import simulation_functions as simf
import pickle
from variable_forecasting_with_data gen import pred_N_min_ahead_point
import shutil
from matplot lib import animation
from powerwallBattery_Model import Battery

Start_date = '7/1/2013' # must be 2013
End_date = '7/3/2013' # must be 2013
Summer = True

Timestep_size = 1 # interval (min apart) must go into 60 evenly (60,30,20,15,12,10,6,5,4,3,2,1)

Cooling_SP_T = simf.f_to_c(80) #69-80 F
Heating_SP_T = simf.f_to_c(60) #63-75 F

T_max_movement = 3 # max number of degrees(F) the temperature can change in 1 hour

Include_solar = True
Include_battery = True
Net_metering = True

MHE_estimation = True
mhe_start_time = 1440 + 1440 # start after 1 day (1440 min)
```
MPC_control = True
mpc_start_time = 1500 + 1440 # start after 1 day 60 min (1500 min)
Forecasting = True
Real_time_Plotting = False
remote_solve = True
remote_server = 'http://machinelearning.byu.edu'
AC_force_off = False
# file locations

if location_name == 'phoenix':
    weather_name = 'USA_AZ_Phoenix-Sky Harbor Intl_API-722780_TMY3' # must be in your Eplus weather folder # change xml
if location_name == 'atlanta':
    weather_name = 'USA_GA_Atlanta-Hartsfield-Jackson Intl_API-722190_TMY3' # must be in your Eplus weather folder # change xml

house_label = 'cody_house'
house_hvac_load = 5590

Eplus_file_location = "C:\EnergyPlusV8-8-0"
BEopt_installation_path = 'C:\Program Files (x86)\NREL\BEopt_2.8.0'
current_dir = os.getcwd()
path_to_buildings = current_dir + '\\Buildings' # must be full path to work with ep input
pyEp.set_eplus_dir(Eplus_file_location)  # sets energyplus path
builder = pyEp.socket_builder(path_to_buildings)  # creates exe path
cfgs = builder.build()  # Configs is [port, building_folder_path, idf]

for file in os.listdir('%s\\raw_files' % current_dir):
    os.remove('%s\\raw_files\\%s' %(current_dir, file))
shutil.copy2('%s\\houses\\%s\\default.xml' %(current_dir, house_label), '%s\\raw_files\\%s' % (current_dir, house_label))

if not os.path.isdir('%s\\plots\\%s\\%s' %(location_name, house_label)):
    # os.mkdir('%s\\plots\\%s\\%s' % (location_name, house_label))
    os.mkdir('%s\\plots\\%s\\%s\\mhe_plots' % (location_name, house_label))
    os.mkdir('%s\\plots\\%s\\%s\\mpc_plots' % (location_name, house_label))

# Misc Parameters
MHE_Debug = False
MPC_Debug = True
MPC_Debug_break = False
HVAC_STATUS = 0  # initialize variable

# setup time
Start_date = datetime.datetime.strptime(Start_date, '%m/%d/%Y')
End_date = datetime.datetime.strptime(End_date, '%m/%d/%Y')
End_date = End_date + datetime.timedelta(days = 1)
# time simulation
start_time = time.time()  # record start time of simulation for simulation time calculation

# setup time
Timesteps = int(60/Timestep_size)  # do a multiple of 5 to work with RTP pricing (timesteps in an hour so 4 = 15 min)
sim_timestep_tot = int((End_date - Start_date).days) * 24) / (Timestep_size/60)  # calculate total timesteps of simulation
109 sim_timestep_per_day = int(sim_timestep_tot / ((End_date - Start_date).days)) # calculate total timesteps in 1 day of simulation
110
time_df = pd.DataFrame(
    {
        'Time (Minutes)': np.arange(0, sim_timestep_tot),
        'Time (Days)': np.arange(0, (End_date - Start_date).days / 1440),
        'Time (hours)': np.arange(0, 24 * (End_date - Start_date).days / 60)
    }
)
111
# Setup Pricing (pulls RTO pricing from API)
Capacity_Charge = 7 # cents/kWh # this is the same over all time so it is not included in calc for cost
net_meter_price = 0.1196 * 100
112
# Combine this with output_df?
plotting_df = pd.DataFrame(
    {
        'Cooling SP': np.ones(sim_timestep_tot),
        'Heating SP': np.ones(sim_timestep_tot),
        'Capacity Charge': np.ones(sim_timestep_tot) * Capacity_Charge,
        'TOD Pricing': np.ones(sim_timestep_tot) * Total_price_per_timestep_df['price']
    }
)

# Initialize plot and storage variables
results_df = pd.DataFrame(
    {
        'actual profit with solar': np.zeros(sim_timestep_tot),
        'optimized profit': np.zeros(sim_timestep_tot),
        'actual profit with solar normal temp control': np.zeros(sim_timestep_tot),
        'actual profit no batt or solar': np.zeros(sim_timestep_tot)
    }
)

# MPC forecasting index array
mpc = np.arange(0, sim_timestep_tot)

# Testing better training day
T_SP_df['Cooling SP'] = np.ones(sim_timestep_tot) * (Heating_SP_T - 2) # change the array to constant value so it doesn't interfere with mpc changes, add 2 to increase driving force for thermostat
T_SP_df['Heating SP'] = np.ones(sim_timestep_tot) * (Heating_SP_T - 10)

# MPC
forecasting_index_array_mpc = np.array([
```python
[0,1,2,3,4,5,6,7,8,9,10,20,30],
[0,1,2,3,4,5,6,7,8,9,10,20,30],
[0,1,2,3,4,5,6,7,8,9,10,20,30,40],
[0,1,2,3,4,5,6,7,8,9,10,20,30,40,50],
[0,10,20,30,40,60],
[0,10,20,30,40,50,60,120],
[0,10,20,30,40,50,60,120],
[0,10,20,30,40,50,60,120],
[0,10,20,30,40,50,60,120],
[0,10,20,30,40,50,60,120],
[0,10,20,30,40,50,60,120],
[0,10,20,30,40,50,60,120],
[0,10,20,30,40,50,60,120],
[0,30,60,120,180,240,300,360],
[0,30,60,120,180,240,300,360],
[0,30,60,120,180,240,300,360],
[0,60,120,180,240,300,360,360,420],
[0,60,120,180,240,300,360,420,480,540],
[0,120,180,240,300,360,420,480,540,600,720],
[0,120,180,240,300,360,420,480,540,600,720,780],
[0,180,300,420,540,720,840,960,1080,1200,1320,1440],
[0,180,300,420,540,720,840,960,1080,1200,1320,1440],
[0,180,300,420,540,720,840,960,1080,1200,1320,1440]
)

different_times = []
for i in range(len(forecasting_index_array_mpc)):
    different_times = np.append(different_times, forecasting_index_array_mpc[i])
    different_times = set(different_times)
different_times = [int(i) for i in different_times]

mpc_time = np.append(np.arange(0, 120, 10), np.arange(120, 1460, 20))
mpc_time = np.arange(0, 1450, 10)
mpc_time = mpc_time/60

if Real_time_Plotting == True:
    fig, (ax1, ax2) = plt.subplots(2, 1)
    fig.tight_layout()
    fig.set_figheight(8)
    fig.set_figwidth(15)
    ax2_2 = ax2.twinx()
    hour_locator = dates.HourLocator()
    hour_formatter = dates.DateFormatter('%I %p')
    min_locator = dates.MinuteLocator(interval = 5)
    mhe_input_array = args_input = np.ones([10,3])
    mhe_input_df = pd.DataFrame(mhe_input_array)
    #global mhe_input_df
```
def frames():
    while True:
        yield mhe_input_df

def subplot_1(args):
    # clear plot
    ax1 cla()

    # set data
    t = mhe_input_df.iloc[0].values
    Tha = mhe_input_df.iloc[1].values
    Ta = mhe_input_df.iloc[2].values
    SP_hi = mhe_input_df.iloc[5].values
    SP_low = mhe_input_df.iloc[6].values
    Tha_mhe = mhe_input_df.iloc[7].values

    # setup subplots
    ax1.set_title('Temperatures')
    ax1.set_ylabel('Temp (F)')
    for tick in ax1.get_xticklabels():
        tick.set_rotation(45)
    ax1.xaxis.set_major_locator(hour_locator)
    ax1.xaxis.set_major_formatter(hour_formatter)
    ax1.xaxis.set_minor_locator(min_locator)

    # plot data
    ax1.plot(t, Tha, color='xkcd: black', label=r'$T_{\text{air, inside}}$')
    ax1.plot(t, Ta, color='xkcd: red', label=r'$T_{\text{ambient}}$')
    ax1.plot(t, SP_hi, color='b', linestyle='--', label=r'$SP_{HI}$')
    ax1.plot(t, SP_low, color='r', linestyle='--', label=r'$SP_{LO}$')
    ax1.plot(t, Tha_mhe, color='xkcd: fuchsia', Marker='x', linestyle='--', label=r'$T_{\text{predicted}}$ $T_{\text{air, inside}}$', markersize=1)

    # post legend
    return ax1.legend()

def subplot_2(args):
    # clear plot
    ax2 cla()

    # set data
    t = mhe_input_df.iloc[0].values
    HVAC = mhe_input_df.iloc[8].values
    HVAC = mhe_input_df.iloc[9].values

    # setup subplots
    ax2.set_title('AC Energy and Status')
    ax2.set_ylabel('Energy Usage (W)')
    ax2.set_ylim([0, 10000])
    for tick in ax2.get_xticklabels():
        tick.set_rotation(45)
    ax2.xaxis.set_major_locator(hour_locator)
    ax2.xaxis.set_major_formatter(hour_formatter)
    ax2.xaxis.set_minor_locator(min_locator)

    ax2,2 = ax2.twinx()
    ax2,2.set_xlabel('AC On/OFF', rotation=270, labelpad=20)
    ax2,2.set_xlim([-5.1, 5])
```python
# plot data
ax2.plot(t, P_HVAC, color='g', label=r'$Q_{HVAC}$')  # PHVAC line
mhe.setData(t, P_HVAC).
ax2.plot(t, I0_HVAC, color='r', label=r'$HVAC$ $I/O$', ls='steps')  # HVACIO line
mhe.setData(t, I0_HVAC)

# post legend
return ax2.legend(loc='lower left'), ax2.legend(loc='upper left')

def animate(args):
    return subplot1(args), subplot2(args), fig.tight_layout()

anim = animation.FuncAnimation(fig, animate, frames=frames, interval=1000, repeat=False, blit=False)

# mpc plot
fig2, (ax21, ax22, ax23, ax24) = plt.subplots(4, 1)
# setup figure
fig2.tight_layout()
fig2.set_figheight(8)
fig2.set_figwidth(15)
ax22 = ax22.twinx()

mpc_hour_locator = dates.HourLocator(interval=2)
mpc_hour_formatter = dates.DateField('%I %p')
mpc_min_locator = dates.MinuteLocator(interval=30)

mpc_input_array = args['input'] = np.ones((15, 3))
mpc_input_df = pd.DataFrame(mpc_input_array)

def mpc_frames():
    while True:
        yield mpc_input_df

def subplot21(args):
    # clear plot
    ax21.cla()

    # set data
t = mpc_input_df.iloc[0].values
Tha_pred = mpc_input_df.iloc[1].values
Ta_pred = mpc_input_df.iloc[2].values
SP_HI = mpc_input_df.iloc[7].values
SP_LO = mpc_input_df.iloc[8].values

# setup subplots
ax21.set_title('Temperatures')
ax21.set_ylabel('Temp (F)')
for tick in ax21.get_xticklabels():
    tick.set_rotation(45)
ax21.xaxis.set_major_locator(mpc_hour_locator)
ax21.xaxis.set_major_formatter(mpc_hour_formatter)
ax21.xaxis.set_minor_locator(mpc_min_locator)

# plot data
ax21.plot(t, Tha_pred, color='xkcd: black', Marker='o', markersize=2.5, label=r'$SP_{HI}$')
ax21.plot(t, Ta_pred, color='xkcd: red', Marker='o', markersize=2.5, label=r'$SP_{LO}$')
ax21.plot(t, SP_HI, color='b', linestyle='--', label=r'$SP_{HI}$')
ax21.plot(t, SP_LO, color='r', linestyle='--', label=r'$SP_{LO}$')

# post legend
return ax21.legend()
```
def subplot_22(args):
    # clear plot
    ax22.cla()
    ax22_2.cla()

    # set data
    t = mpc_input_df.iloc[0].values
    Batt_SOC = mpc_input_df.iloc[3].values
    IOHVAC = mpc_input_df.iloc[4].values

    # setup subplots
    ax22.set_title('AC Status and Battery SOC')
    ax22.set_xlabel('AC STATUS')
    ax22.set_ylim((0, 1.1))
    for tick in ax22.get_xticklabels():
        tick.set_rotation(45)
    ax22.xaxis.set_major_locator(mpc_hour_locator)
    ax22.xaxis.set_major_formatter(mpc_hour_formatter)
    ax22.xaxis.set_minor_locator(mpc_minute_locator)

    # ax22_2 = ax22.twinx()
    ax22_2.set_ylabel('BATT SOC', rotation=270, labelpad=20)
    ax22_2.set_ylim((0, 1.1))
    for tick in ax22_2.get_xticklabels():
        tick.set_rotation(45)
    ax22_2.xaxis.set_major_locator(mpc_hour_locator)
    ax22_2.xaxis.set_major_formatter(mpc_hour_formatter)
    ax22_2.xaxis.set_minor_locator(mpc_minute_locator)

    # plot data
    ax22.plot(t, Batt_SOC, color='g', label=r'$SOC$', ls='steps')
    ax22_2.plot(t, IOHVAC, color='r', label=r'$HVAC I/O$', ls='steps')

    # post legend
    return ax22.legend(loc='upper left'), ax22_2.legend(loc='upper right')

def subplot_23(args):
    # clear plot
    ax23.cla()

    # set data
    t = mpc_input_df.iloc[0].values
    TOD_price = mpc_input_df.iloc[5].values
    Netmetering_price = mpc_input_df.iloc[6].values

    # setup subplots
    ax23.set_title('Pricing')
    ax23.set_xlabel('Cents/kWh')
    for tick in ax23.get_xticklabels():
        tick.set_rotation(45)
    ax23.xaxis.set_major_locator(mpc_hour_locator)
    ax23.xaxis.set_major_formatter(mpc_hour_formatter)
    ax23.xaxis.set_minor_locator(mpc_minute_locator)

    # plot data
    ax23.plot(t, TOD_price, color='r', label=r'$TOD \$$', ls='steps')
    ax23.plot(t, Netmetering_price, color='g', label=r'$Netmetering \$$', ls='steps')
    return ax23.legend()
```python
def subplot24(args):
    # Clear plot
    ax24.clf()

    # Set data
    t = mpc_input_df.iloc[0].values
    PV_prod = mpc_input_df.iloc[11].values
    PJHVAC = mpc_input_df.iloc[13].values

    # Setup subplots
    ax24.set_title('Energy Predictions')
    ax24.set_xlabel('Usage/Production (W)')
    ax24.set_ylim([0, 8000])
    for tick in ax24.get_xticklabels():
        tick.set_rotation(45)
    ax24.xaxis.set_major_locator(mpc_hour_locator)
    ax24.xaxis.set_major_formatter(mpc_hour_formatter)
    ax24.xaxis.set_minor_locator(mpc_min_locator)

    # Plot data
    ax24.plot(t, PV_prod, color='xkcd:orange', label=r'$S$ Solar $Power$ Produced', ls='steps')
    ax24.plot(t, PJHVAC, color='xkcd:blue', label=r'$T$ Total $Power$ Required', ls='steps')
    return ax24.legend()

def animate2(args):
    return subplot21(args), subplot22(args), subplot23(args), subplot24(args), fig2.tight_layout()

anim2 = animation.FuncAnimation(fig2, animate2, frames=mpc_frames, save_count=0, interval=1000, repeat=False, blit=False)
plt.pause(4)
# Run simulator
while True:
    try:
        ep = pyEp.ep_process('localhost', configs[0][0], configs[0][1], weather_name)  # Runs exe
        break
    except IndexError:
        builder = pyEp.socket_builder(path_to_buildings)  # Creates exe path
        configs = builder.build()  # Configs is [port, building_folder_path, idf]

# Cosimulation
outputs = []  # Initialize array

EPTimeStep = Timessteps
SimDays = ((End_date - Start_date).days)
kStep = 1
MAXSTEPS = int(SimDays*EPTimeStep)  #
DeltaT = (60/EPTimeStep)*60; # seconds per timestep

try:
    print("Running Cosimulation with Total Steps " + str(MAXSTEPS))
    while (kStep < MAXSTEPS):
        i_array = kStep - 1
        print('')
        print(str(kStep) + '/' + str(MAXSTEPS))
        sim_time = (kStep - 1) + DeltaT
        dayTime = sim_time % 86400
```

```python
# parse output array
output = ep.decode_packet_simple(ep.read())
outputs.append(output)

output_df = pd.DataFrame.from_records(outputs, columns=
    ['Drybulb Temp (C)', 'Zone Air Temp (C)',
     'Cooling Temp SP (C)', 'Heating Temp SP (C)',
     'Air System Total Cooling Energy (J)', 'Total Facility Power Demand (W)',
     'HVAC Power (W)', 'Total Power Generated (W)',
     'site diffuse solar radiation', 'site direct solar radiation',
     'site ground reflected solar radiation', 'pv array eff.',
     'pv cell temp.', 'pv short circuit current', 'open circuit voltage',
     'Relative Humidity', 'Wind Speed', 'Site Ground Temp (C)', 'Precip Depth',
     'Rain Status', 'Sky Clearness'])

output_df['Misc building Power (W)'] = output_df['Total Facility Power Demand (W)'] - output_df['HVAC Power (W)']
output_df['Water VP (hPa)'] = (output_df['Relative Humidity'] / 100) * 6.105 * np.exp((17.27 * output_df['Drybulb Temp (C)']) / (237.7 + output_df['Drybulb Temp (C)']))
output_df['Apparent Temp (C)'] = output_df['Drybulb Temp (C)'] + 0.348 * output_df['Water VP (hPa)'] - 0.7 * output_df['Wind Speed']

output_df['HVAC I/O'] = output_df['HVAC Power (W)'] != 0
output_df['HVAC I/O'] = output_df['HVAC I/O'].astype(bool).astype(int)
output_df['Times step'] = range(kStep)
output_df['Date and Time'] = Start_date + datetime.timedelta(seconds = 60 * output_df['Times step'])

output_df['Act Demand'] = output_df['Misc building Power (W)'] + output_df['HVAC Power (W)']
AC DC_invert_eff = np.sqrt(0.9) # this is to account for energy in and energy out losses
output_df['Grid Requirement (no battery)'] = output_df['Act Demand'] - output_df['Total Power Generated (W)']

output_df['TOD PRICING'] = Total_price_per_timestep_df['price'][len(output_df):]
output_df['NETMETERING PRICING'] = np.ones(len(output_df)) * net_meter_price

# outputs list and order can be found in the variables file inside the building folder

# MHE during loop
if MHE_estimation == True and i_array > mhe_start_time and i_array <= mpe_start_time: # skip a day so that weather can be predicted #TODO check to make sure this isn\'t breaking things
    # start values
    if i_array == mhe_start_time:
        A_storage = {}
        B_storage = {}
        C_storage = {}
        D_storage = {}
        solve_time_mhe = []
        # MHE model
        mhe = GEKKO(name = 'MHE', remote = False) # remote, solve, server = remote_server)

    # in seconds
    mhe_time = np.arange(0.36 * 60, 1 / 60) # must equal units of mpc time

    mhe_time = mhe_time
    # parameters to estimate
    A_mhe = mhe.FV(value = 2.696117, lb = 0, name = 'A') # value = A_av) # lb = 0)
    B_mhe = mhe.FV(value = 0.08670757, lb = 0, name = 'B') # value = B_av) # lb = 0)
    C_mhe = mhe.FV(value = 0.2234747, lb = 0, name = 'C') # value = C_av) # lb = 0)
    D_mhe = mhe.FV(value = 2.89168, lb = 0, name = 'D') # value = D_av) # lb = 0)
    A_mhe.STATUS = 1
    B_mhe.STATUS = 1
```

67
# input variables
TaKmhe = mhe.MV(name = 'TaKmhe', value = output_df['Drybulb Temp (C)'].values[-len(mhe.time)])

# state variables
u_change_mhe = mhe.SV(name = 'uchange')

# control variables
Tha_pred_Kmhe = mhe.CV(value = output_df['Zone Air Temp (C)'].values[-len(mhe.time)], name = 'Tha_pred_Kmhe')

# equations
mhe.Equation(u_change_mhe == HVAC_IO_mhe)

# Global Options
mhe.options.IMODE = 5 # MHE

mhe.options.SOLVETIME = 0
mhe.options.DAYS = 0
mhe.options.MAX_ITER = 1000 # max number of iterations
mhe.options.TIME_SHIFT = 10

MHE.STATUS = False
mhe.sol_storage = {}

# output_df['mhe_sol'] = np.zeros(len(output_df))

if i_array >= mhe.start_time and i_array % 10 == 0: # sim timestep per day:
    Tha_pred_Kmhe.VALUE = output_df['Zone Air Temp (C)'].values[-len(mhe.time)]
    TaKmhe.VALUE = output_df['Drybulb Temp (C)'].values[-len(mhe.time)]
    HVAC_IO_mhe.VALUE = output_df['HVAC I/O'].values[-len(mhe.time)]
    TgKmhe.VALUE = output_df['Site Ground Temp (C)'].values[-len(mhe.time)]

print('solving mhe')

mhe.solve(debug = MHEDebug.debug = MHEDebug)

A_storage['Timestep%s solved%s' % (iStep, MHE.STATUS)] = [A_mhe.value[0]]
B_storage['Timestep%s solved%s' % (iStep, MHE.STATUS)] = [B_mhe.value[0]]
C_storage['Timestep%s solved%s' % (iStep, MHE.STATUS)] = [C_mhe.value[0]]
D_storage['Timestep%s solved%s' % (iStep, MHE.STATUS)] = [D_mhe.value[0]]
if MHE Debug == True:
    input("Press Enter to Continue")

print('MHE Solved = ' + str(MHE STATUS))
print( 'Objective function = %s' %mhe.options.OBJFCNVAL)

# store solutions in dataframe
mhe solvent = mhe.load_results()
mhe solvent = pd.DataFrame.from_dict(mhe solvent)
mhe solvent_storage['Time step:%s solved:%s' %kStep,MHE STATUS] = mhe solvent
solve_time_mhe.append(mhe.options.SOLVETIME)

if array == mpc start time:
solve_time_mpc = []
iterations_mpc = []
ACmhe = GEKKO(name = 'ACmhe ', remote = False)
ACmhe = np.linspace(0, mpc start time - 1, mpc start time)
ACmhe = ACmhe time

# parameters to estimate
HVAC load ACmhe = ACmhe.FV(name = 'HVAC load ACmhe ') # in joules
HVAC load ACmhe. STATUS = 1

# inputs
E HVAC per on ACmhe = ACmhe.Param(name = 'E HVAC per on')
Ta ACmhe = ACmhe.Param(name = 'Ta ACmhe ')

# variables to match data
E HVAC consumption ACmhe = ACmhe.CV(value = 3000, name = 'E HVAC consumption')
E HVAC consumption ACmhe. STATUS = 1
E HVAC consumption ACmhe. FSTATUS = 1

# Equations
ACmhe. Equation(E HVAC consumption ACmhe == E HVAC per on ACmhe * HVAC load ACmhe * Ta ACmhe)

# Global Options
ACmhe.options.IMODE = 5 # ACmhe
ACmhe.options.EV_TYPE = 2 # Objective type we a
ACmhe.options.NODS = 2 # Collocation nodes
ACmhe.options.MAX_ITER = 1000 # max number of iterations
ACmhe.options.MAX_TIME = 60+4 # seconds

if array >= mpc start time and array % 10 == 0:
    E HVAC per on ACmhe.VALUE = output_df[ 'HVAC I/O'][i_array−mpc start time−1 array]
    Ta ACmhe.VALUE = output_df[ 'Drybulb Temp (C)'][i_array−mpc start time−1 array]
    E HVAC consumption ACmhe.VALUE = output_df[ 'HVAC Power (W)'][i_array−mpc start time−1 array]/1e3
ACmhe.solve(disp = False)

# create weather prediction array
if Forecasting == True and i_array >= mpc start time:
    # initialize dicts for data
    past Drybulb Temp_C dict = {}
past Misc building Power dict = {}
past Total Power Generated dict = {}

    # change mpc array to array in minutes
    mpc min array = np.array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,]...
```python
mpc_empty_array_mpc = np.zeros([len(mpc_min_array[1:])])
P_m misc_input_array = np.zeros([len(mpc_min_array[1:])])
for i in range(len(P_m misc_input_array)):
P_m misc_input_array[i] = 0

# create dictionary of data for forecasting
for time_ag in different times:
past_Drybulb_Temp_c_dict['temp_%s_minutes_ago' % time_ag] = output_df['Drybulb Temp (C)'][i_array - time_ag]  # simf
future_c_to_k:
past_Misc_building_Power_dict['misc_power_%s_minutes_ago' % time_ag] = output_df['Misc building Power (W)'][i_array - time_ag]
past_Total_Power_Generated_dict['generated_power_%s_minutes_ago' % time_ag] = output_df['Total Power Generated (W)'][i_array - time_ag]

# initialize arrays for predicted data to go into
Drybulb_Temp_c_future = np.zeros(len(mpc_min_array[1:])), Misc_building_Power_future = np.zeros(len(mpc_min_array[1:])), Total_Power_Generated_W_future = np.zeros(len(mpc_min_array[1:])),

# make misc power prediction array
for k, i in zip(range(len(Misc_building_Power_future)), mpc_min_array[1:]): # loop through whole array and every point in mpc array(min) except for the current point
# make array for formula input
Misc_building_Power_hist_array = np.zeros(len(P_m misc_input_array[k]))
for i, j in zip(range(len(P_m misc_input_array[k])), P_m misc_input_array[k]):
    Misc_building_Power_hist_array[i] = past_Misc_building_Power_dict['misc_power_%s_minutes_ago' % int(j)]

# indicate hour of the predicted point
future_time = output_df['Date and Time'][i_array] + datetime.timedelta(seconds = 60)*1
hour_indicator_param = future_time.hour = 1

Misc_building_Power_params = Misc_building_Power_forecasting_param_dict['predict %s minutes ahead' % 1]
Misc_building_Power_future[k] = pred_N_min_ahead_point(Misc_building_Power_params, Misc_building_Power_hist_array, hour_indicator_param)

Misc_building_Power_future[Misc_building_Power_future < 0] = 0

# make ta prediction array
for k, i in zip(range(len(Drybulb_Temp_c_future)), mpc_min_array[1:]), # loop through whole array and every point in mpc array(min) except for the current point
# make array for formula input
Drybulb_Temp_c_hist_array = np.zeros(len( forecasting_index_array_mpc[k] ) )
for i, j in zip(range(len( forecasting_index_array_mpc[k] ) ), forecasting_index_array_mpc[k]):
    Drybulb_Temp_c_hist_array[i] = past_Drybulb_Temp_c_dict['temp_%s_minutes_ago' % j]

# indicate hour of the predicted point
future_time = output_df['Date and Time'][i_array] + datetime.timedelta(seconds = 60)*1
hour_indicator_param = future_time.hour = 1

Drybulb_Temp_c_params = Drybulb_Temp_c_forecasting_param_dict['predict %s minutes ahead' % 1]
Drybulb_Temp_c_future[k] = simf.c_to_k(pred_N_min_ahead_point(Drybulb_Temp_c_params, Drybulb_Temp_c_hist_array, hour_indicator_param))

# make pv power prediction array
for k, l in zip(range(len(Total_Power_Generated_W_future)), mpc_min_array[1:]), # loop through whole array and every point in mpc array(min) except for the current point
```
# make array for formula input
Total_power_generated_hist_array = mpc_empty_array

# indicate hour of the predicted point
hour_indicator_param = np.zeros(24)
future_time = output_df["Date and Time"][i_array] + datetime.timedelta(seconds = 60)*1
hour_indicator_param[future_time.hour] = 1

Total_power_generated_params = Total_power_generated_W_forecasting_param_dict["predict %s minutes ahead" %l]
Total_power_generated_W_future[k] = pred_X_min_head_point(Total_power_generated_params, Total_power_generated_hist_array, hour_indicator_param)

Total_power_generated_W_future[Total_power_generated_W_future < 0] = 0

# from scipy.interpolate import interp1d
f_drybulb = interp1d(mpc_min_array, np.append(output_df["Drybulb Temp (C)"][i_array], simf.k_to_c(Drybulb_Temp_C future ), 'cubic')
f_ppv = interp1d(mpc_min_array, np.append(output_df["Total Power Generated (W)"][i_array], Total_power_generated_W_future ))
f_pmisc = interp1d(mpc_min_array, np.append(output_df["Misc building Power (W)"][i_array], Misc_building_Power_future ), 'cubic')

# MPC
if i_array >= mpc_start_time and MPC_control == True: # skip a day so that weather can be predicted
    # start values
    if i_array == mpc_start_time: # start 60 minutes after mhe start
        # model initialization
        battery_state_storage = {}
        mpc = GEKKO(name = 'MPC', remote = False)#remote_solve, server = remote_server)
        mpc.time = mpc_time
        # parameters to estimate
        A_mpc = mpc.FV(value = A_mhe.NEWVAL, name = 'A')#value = A_av)#lb=0)
        B_mpc = mpc.FV(value = B_mhe.NEWVAL, name = 'B')#value = B_av)#lb=0)
        C_mpc = mpc.FV(value = C_mhe.NEWVAL, name = 'C')#value = C_av)#lb=0)
        D_mpc = mpc.FV(value = D_mhe.NEWVAL, name = 'D')
        A_mpc.FSTATUS = 1
        B_mpc.FSTATUS = 1
        C_mpc.FSTATUS = 1
        D_mpc.FSTATUS = 1
        Ta_K_mpc = mpc.MV(name = 'T_K_mpc', value = output_df["Drybulb Temp (C)"][values][i_array])
        Tg_K_mpc = mpc.MV(name = 'Tg_K_mpc', value = output_df["Site Ground Temp (C)"][values][i_array])
        Ta_K_mpc.FSTATUS = 1
        Tg_K_mpc.FSTATUS = 1
        # manipulated variables
        HVAC_Q_mpc = mpc.MV(name = 'E_HVAC_on', lb = 0, ub = 1) # add initial point?
        HVAC_Q_mpc.FSTATUS = 1
        HVAC_Q_mpc.STATUS = 1
        # state variables
        u_change_mpc = mpc.SV(name = 'uchange')
        u_tracker_mpc = mpc.SV(name = 'u_tracker')
        # control variables
```python
Tha_pred_K_mpc = mpc.CV(value = output_df["Zone Air Temp (C)"].values[1], array, name = 'Tha_pred_K_mpc') # lb = Heating_SP_T, ub = Cooling_SP_T
Tha_pred_K_mpc.STATUS = 1
Tha_pred_K_mpc.PSTATUS = 1
Tha_pred_K_mpc.SPHI = Cooling_SP_T
Tha_pred_K_mpc.WPHI = 1000
Tha_pred_K_mpc.SPLLO = Heating_SP_T
Tha_pred_K_mpc.WPLLO = 1000

# equations
mpc.Equation(u_change_mpc == HVAC_IO_mpc)
mpc.Equation(u_change_mpc.dt() == u_tracker_mpc)
mpc.Equation(Tha_pred_K_mpc.dt() == B_mpc*(T_a_mpc- Tha_pred_K_mpc) + C_mpc*(T_g_mpc- Tha_pred_K_mpc) - HVAC_IO_mpc*D_mpc - u_tracker_mpc * A_mpc)

# push HVAC_IO_mpc to 0 or 1
pusher_x = 9e-7  # works with step pricing

mpc.Equation(-pusher_x*(HVAC_IO_mpc-0.5)**2 + pusher_x/4 <= 1e-8)

# Battery Model

# whole battery parameters
SOC_initial = 0.001
# initial state of charge
V_battery = 50  # battery voltage
total_batt_energy = 14*1000  # total battery energy in Wh
round_trip_efficiency = .90  # efficiency of battery in and out
max_draw = 5000  #W limit on charge and discharge

# Individual Lithium–ion battery parameters
E0_batt = 3.7348  # (V) [battery constant voltage]
R_batt = 0.09  # (Ohms) [internal resistance]
K_batt = 0.00876  # (V) [polarization voltage]
A_batt = 0.468  # (V) [exponential zone amplitude]
B_batt = 3.5294  # (Ah)^-1 [exponential zone time constant inverse]
Q_batt = 1  # (As) [battery capacity]
V_lithium_batt = 3.6  # (V) [electrochemical potential of lithium]

N_series = np.ceil(V_battery/V_lithium_batt)
capacity_N_series = total_batt_energy/(N_series*V_lithium_batt)
N_cells = np.ceil(capacity_N_series/Q_batt)
Capacity_discharged_0 = -(SOC_initial + Q_batt)-Q_batt

# battery pre-calcualtions
if Include_battery == True:
    charge_batter = mpc.SV(value = 0, lb = 0, name = 'charge_batter')

p_AC_DC_inv = mpc.SV(value = 0, name = 'p_AC_DC_inv')

current_into_batt = mpc.SV(value = 0, name = 'current_into_batt') #Amps

indv_cell_current_into_batt = mpc.SV(value = 0, name='indv_cell_current_into_batt')#Amps

Capacity_discharged = mpc.SV(name = 'Capacity_discharged', value = Capacity_discharged_0, lb = 0, ub = Q_batt)

SOC_mpc = mpc.SV(value = SOC_initial, lb=0, ub=1, name='SOC')

indv_cell_current_outof_batt = mpc.SV(value = 0, name='indv_cell_current_outof_batt ')#Amps

current_outof_batt = mpc.SV(value = 0, name = 'current_outof_batt') #Amps
```

72
p_DC_AC_inv = mpc.SV(value = 0, name = 'p_DC_AC_inv')

discharge_battery = mpc.SV(value = 0, lb = 0, name = 'discharge_battery')

P_batt_storage_mpc = mpc.MV(name = 'P_batt_storage_mpc') #, lb = -5000, ub = 5000) # positive means discharging

# Watts

P_batt_storage_mpc.STATUS = 1
P_batt_storage_mpc.FSTATUS = 1
P_batt_storage_mpc.LOWER = -max_draw/1e3
P_batt_storage_mpc.UPPER = max_draw/1e3

S3_mpc = mpc.SV(lb = 0, name = 'Slack var3')
S4_mpc = mpc.SV(lb = 0, name = 'Slack var4')

mpc.Equation(discharge_battery == P_batt_storage_mpc + S3_mpc)
mpc.Equation(charge_battery == -P_batt_storage_mpc + S4_mpc)
mpc.Equation((P_batt_storage_mpc == S4_mpc - S3_mpc)
mpc.Equation((p_DC_AC_inv == charge_battery + np.sqrt(.9))
mpc.Equation(current_into_batt == p_DC_AC_inv/V_batt) # calculate current going into battery
mpc.Equation(indv_cell_current_into_batt == current_into_batt*1e3/N_cells)
mpc.Equation(Capacity_discharged_.dt() == indv_cell_current_into_batt - indv_cell_current_into_batt)
mpc.Equation(SOC_mpc == (Q_batt - Capacity_discharged) / Q_batt) # soc is the capacity of the battery minuse what is discharged

else:
    P_batt_storage_mpc = mpc.Const(value = 0, name = 'P_batt_storage_mpc')

##### added mpc equations and variables

Demand_mpc = mpc.SV(name = 'Demand')

Grid_to_Demand_mpc = mpc.SV(name = 'Grid_to_Demand_mpc') #, lb = 0)

Grid_to_Demand_mpc.LOWER = 0

Sell_to_grid = mpc.SV(value = 0, name = 'Sell back to grid') #, lb = 0)
Sell_to_grid.LOWER = 0

Profit_mpc = mpc.CV(value = .001, name = 'profit_mpc')
Profit_mpc.STATUS = 1
Profit_mpc.COST = 10 # negative maximize

S1_mpc = mpc.SV(lb = 0, name = 'Slack var1')

S2_mpc = mpc.SV(lb = 0, name = 'Slack var2')

P_misc_mpc = mpc.MV(value = 3, name = 'P_misc_mpc')

P_misc_mpc.FSTATUS = 1
P_misc_mpc STATUS = 0

# TOD_Pricing_mpc = mpc.Param(name = 'TOD_Pricing_mpc') # cents/kWh
net_meter_pricing_mpc = mpc.Param(name = 'net_meter_pricing_mpc') # cents/kWh

optimize_profit = mpc.Param(name = 'optimize profit')
Utilities_charge_mpc = mpc.SV(value = 0, name = 'Utilities_charge_mpc', lb = 0) # dont name it cost--it causes problems with gekko code

if Net_metering == True:
    Revenue_mpc = mpc.SV(name = 'Revenue_mpc', lb = 0)
else:
    Revenue_mpc = mpc.Const(name = 'Revenue_mpc', value = 0)

if Include_solar == True:
P_PV_mpc = mpc.MV(name = 'P_PV_mpc', value = 0).initialize_df['Total Power Generated (W)'][0])
P_PV_mpc.FSTATUS = 1 # receive measurement
P_PV_mpc.STATUS = 0
else:
P_PV_mpc = mpc.Const(value = 0, name = 'P_PV_mpc')

P_HVAC_load_mpc = mpc.FV(value = HVAC_load_ACmhe.NEWVAL, name = 'P_HVAC_load_mpc')
P_HVAC_load_mpc.FSTATUS = 1

# Energy Balances
mpc.Equation(Demand_mpc == P_mise_mpc + (HVAC_Q_mpc+P_HVAC_load_mpc+ToK_mpc) - P_PV_mpc)
mpc.Equation(Grid_to_Demand_mpc + P_batt_storage_mpc - Sell_to_grid == Demand_mpc)
mpc.Equation(Grid_to_Demand_mpc == Demand_mpc - P_batt_storage_mpc + S1_mpc)#P Demand_mpc
mpc.Equation(Sell_to_grid == P_batt_storage_mpc - Demand_mpc + S2_mpc)
mpc.Equation(P_batt_storage_mpc - Demand_mpc == S1_mpc - S2_mpc)
mpc.Equation(S1_mpc <= 0)

# Financial calculations
mpc.Equation(Revenue_mpc == Sell_to_grid*1/60 * net_meter_pricing_mpc/100)
mpc.Equation(Utility_charge_mpc == Grid_to_Demand_mpc*1/60 + TOD_pricing_mpc/100) # this value magnitude is important for solving

mpc.Equation(Profit_mpc == (Revenue_mpc - Utilities_charge_mpc)-optimize_profit)

# Global Options
mpc.options.IMODE = 6 # MPC
mpc.options.CV_TYPE = 1 # Objective type # use with sphi and sploo (1) or splo (2)
mpc.options.NODES = 2 # Collocation nodes # causing errors
mpc.options.SOLVER = 3 # 1=APOPT, 3=IPOPT
mpc.options.MAX_ITER = 5000
mpc.options.EV_TYPE = 2
mpc.options.TIME_SHIFT = 1
MPC.STATUS = False
mpc.sol_storage = {}

mpc.options.TIME_SHIFT = 0
mpc.options.COLDSTART = 2
mpc.solve(disp = MPC_Debug, debug = MPC_Debug_break)
mpc.options.TIME_SHIFT = 1

#% MPC during loop
if i_array % 10 == 0:
    # Update parameters from mhe
    if Include_battery == True:
        if i_array >= mpc_start_time + 10:
            current_soc = battery_state_storage['Time step %s' % (i_array - 10)]['SOC'][49]
            Capacity_discharged.MEAS = -(current_soc * Q_batt) - Q_batt
        else:
            Capacity_discharged.MEAS = -(SOC_initial * Q_batt - Q_batt)
    A_mpc.MEAS = A_mhe.NEWVAL
B_mpc.MEAS = B_mhe.NEWVAL
C_mpc.MEAS = C_mhe.NEWVAL
D_mpc.MEAS = D_mhe.NEWVAL
P_HVAC_load_mpc.MEAS = HVAC_load_AC_mhe.NEWVAL

# variables that should be predicted and sent to the mpc
price_time_array = [i*60+60*(i_array) for i in mpc_time]  # TODO check to see if this really should be 0 after
sim period ends
TOD_Pricing_array = Total_price_per_timestep_df['price'].loc[Total_price_per_timestep_df['time (sec)'].isin(
price_time_array)]  # cents/KWh
TOD_Pricing_mpc.VALUE = TOD_Pricing_array
net_meter_pricing_array = TOD_Pricing_array != 0
net_meter_pricing_array = net_meter_pricing_array.astype(bool).astype(int) * net_meter_price
net_meter_pricing_mpc.VALUE = net_meter_pricing_array
optimize_profit_array = (TOD_Pricing_array != 0).astype(int)

if Forecasting == True:
    Ta_K_mpc.MEAS = f_drybulb(mpc_time*60)
P_PV_mpc.MEAS = np.round(f_ppv(mpc_time*60)/1e3, 3)
P_misc_mpc.MEAS = f_ppmisc(mpc_time*60)/1e3
Tg_K_mpc.MEAS = output_df['Site Ground Temp (C)'][i_array]*np.ones(len(mpc_time))
else:
    Ta_K_mpc.MEAS = output_df['Drybulb Temp (C)'][i_array]
P_PV_mpc.MEAS = output_df['Total Power Generated (W)'][i_array]
P_misc_mpc.MEAS = output_df['Misc building Power (W)'][i_array]

# variables that should be predicted by mpc into the future (create bias)
HVAC_IO_mpc.MEAS = output_df['HVAC I/O'][i_array]
Tha_pred_mpc.MEAS = output_df['Zone Air Temp (C)'][i_array]

if i_array > sim_timestep_tot - (1440 - mpc_start_time % 1440) and Include_battery == True:
    end_of_day = (sim_timestep_tot - mpc_start_time)/60 - (i_array - mpc_start_time)/60
    fix_loc = np.where(np.round(mpc_time, 3) == round(end_of_day, 3))
    mpc.fix(SOC_mpc, fix_loc[0][0], SOC_inital)

print('solving MPC')
mpc.solve(disps=MPD_Debug, debug=MPD_Debug_break)

###make changes to actual process
if mpc.options.SOLVSTATUS == 1:
    MPC_STATUS = True
    if HVAC_IO_mpc.NEWVAL >= .5:
        HVAC_STATUS = 2  # turn hvac cycle on (still stays within temp limits)
    else:
        HVAC_STATUS = 1  # turn hvac off
else:
    if mpc.options.APPINFO == -1:
        MPC_STATUS = False
        print('Stopping from Max Iterations')
        if HVAC_IO_mpc.NEWVAL >= .5:
            HVAC_STATUS = 2  # turn hvac cycle on (still stays within temp limits)
        else:
            HVAC_STATUS = 1  # turn hvac off
    else:
        MPC_STATUS = False
print('MPC Solved = ' + str(MPC_STATUS))
print('Objective function = %s' %mpc.options.OBJFCNVAL)

# store solutions in dataframe
mpc_sol = mpc.load_results()
mpc_sol = pd.DataFrame.from_dict(mpc_sol)

solve_time = mpc.append(mpc.options.SOLVETIME)
iterations = mpc.append(mpc.options.ITERATIONS)

# regular thermostat control
elif i_array >= mpc_start_time - 300 and MPC_control == True:  # control temp if mpc is not turned on
    if output_df['Zone Air Temp (C)'][i_array] > Cooling_SP_T:
        HVAC_STATUS = 2  # turn hvac cycle on (still stays within temp limits)
    elif output_df['Zone Air Temp (C)'][i_array] < Cooling_SP_T - 3.5/9:  # Heating_SP_T:
        HVAC_STATUS = 1  # turn hvac off

else:  # control temp if mpc is not turned on
    if output_df['Zone Air Temp (C)'][i_array] > Cooling_SP_T:
        HVAC_STATUS = 2  # turn hvac cycle on (still stays within temp limits)
    elif output_df['Zone Air Temp (C)'][i_array] < Cooling_SP_T - 2.5/9:  # Heating_SP_T:
        HVAC_STATUS = 1  # turn hvac off

# calculate what actually happens #TODO fix after scaling
if i_array >= mpc_start_time and MPC_control == True:  # if mpc has solved at least once
    if include_battery == True:
        Real_power_storage = P_batt_storage_mpc.NEWVAL * 1e3  # positive if discharging
    else:
        Real_power_storage = 0

# profit with battery and pv
if output_df['Grid Requirement (no battery)'][i_array] > Real_power_storage:
    real_P_from_grid = output_df['Grid Requirement (no battery)'][i_array] - Real_power_storage
else:
    real_P_sold_to_grid = Real_power_storage - output_df['Grid Requirement (no battery)'][i_array]
    real_P_from_grid = 0

ActRevenue_mpc = real_P_sold_to_grid/1000 + (1/60) * output_df['NETMETERING PRICING'][i_array]/100
ActUtilities_charge_mpc = real_P_from_grid/1000 + (1/60) * output_df['TOD PRICING'][i_array]/100
Act_profit = ActRevenue_mpc - ActUtilities_charge_mpc

results_df['optimized profit'][i_array] = Act_profit

# profit of house optimized but no battery
if output_df['Grid Requirement (no battery)'][i_array] > 0:
    real_P_from_grid = output_df['Grid Requirement (no battery)'][i_array]
    real_P_sold_to_grid = 0
else:
    real_P_sold_to_grid = - output_df['Grid Requirement (no battery)'][i_array]
    real_P_from_grid = 0

ActRevenue_mpc = real_P_sold_to_grid/1000 + (1/60) * output_df['NETMETERING PRICING'][i_array]/100
ActUtilities_charge_mpc = (real_P_from_grid/1000 + (1/60) * output_df['TOD PRICING'][i_array]/100
Act_profit = ActRevenue_mpc - ActUtilities_charge_mpc

results_df['actual profit with solar'][i_array] = Act_profit
# profit of house optimized no pv or battery

```
# Utilities, charger, mpc = output_df['Act Demand'] / i_array / 1000 * (1/60) + output_df['TOD PRICING'] / i_array / 100
Act_profit = - Act_Utilities, charger, mpc
results_df[actual profit no batt or solar][i_array] = Act_profit
```

Realt ime plotting

```
if Real_time_Plotting == True:
i_plot = kStep - 1
```

if MHE_estimation == True:
```python
if i_array % 10 == 0:
    if kStep > mhe_start_time:
        mhe_input_array = args_input = np.ones([10, len(mhe_time)])
        mhe_input_df = pd.DataFrame(mhe_input_array)
        mhe_input_df.iloc[0] = dates.date2num(output_df['Date and Time'][i_array])
        mhe_input_df.iloc[1] = simf_c_to_f(output_df['Zone Air Temp (C)'][i_array])
        mhe_input_df.iloc[2] = simf_c_to_f(output_df['Drybulb Temp (C)'][i_array])
        if i_array < mpc_start_time:
            mhe_input_df.iloc[5] = simf_c_to_f(Cooling_SP_T) + np.ones(len(mhe_time))
            mhe_input_df.iloc[6] = simf_c_to_f(Heating_SP_T) + np.ones(len(mhe_time))
        else:
            mhe_input_df.iloc[5] = simf_c_to_f(Cooling_SP_T) + np.ones(len(mhe_time))
            mhe_input_df.iloc[6] = simf_c_to_f(Heating_SP_T) + np.ones(len(mhe_time))
        mhe_input_df.iloc[7] = simf_c_to_f(np.array(Tha[pred_K_mhe.VALUE])
        mhe_input_df.iloc[8] = output_df['HVAC Power (W)'][i_array]
        mhe_input_df.iloc[9] = output_df['HVAC 1/0'][i_array]
        plt.pause(1)
        fig.savefig('plots/%s\%s\mhe_plots\mhe_timestep_%s_tod_ops\%location_name, house_label, i_array)
```

if MPC_control == True:
```python
if i_array % 10 == 0:
    if kStep > mpc_start_time:
        mpc_input_array = args_input = np.ones([15, len(mpc_time)])
        mpc_input_df = pd.DataFrame(mpc_input_array)
        # change mpc array to datetimes
        for i in range(len(mpc_time)):
            mpc_input_df.iloc[0][i] = output_df['Date and Time'][i_array]
            mpc_input_df.iloc[1][i] = simf_c_to_f(np.array(Tha[pred_K_mpc.VALUE])
            mpc_input_df.iloc[2][i] = simf_c_to_f(np.array(Ta_K_mpc.VALUE))
        if Include_battery == True:
            mpc_input_df.iloc[3] = np.array(SOCmpc.VALUE)
            mpc_input_df.iloc[4] = np.array(HVAC10mpc.VALUE) #np.round()
            mpc_input_df.iloc[5] = np.array(TOD_Pricing_mpc.VALUE)
            mpc_input_df.iloc[6] = np.round(net_meter_pricing_mpc.VALUE)
            mpc_input_df.iloc[7] = simf_c_to_f(Cooling_SP_T) + np.ones(len(mpc_time))
            mpc_input_df.iloc[8] = simf_c_to_f(Heating_SP_T) + np.ones(len(mpc_time))
            mpc_input_df.iloc[12] = np.array(Grid_to_Demand_mpc.VALUE) * 1e3
            mpc_input_df.iloc[13] = np.array(P_HVAC_load_mpc.VALUE) + np.array(HVAC10mpc.VALUE) + np.array(P_misc_mpc.VALUE) * 1e3
            mpc_input_df.iloc[14] = np.array(Sell_to_grid.VALUE) * 1e3
```

plt.pause(1)
fig2_savefig('plots\%s\%s\mpc\plots\mpc\timestep\%s\t_id\_eps' % (location_name, house_label, i_array))

#% track changes in real system
if i_array >= mpc_start_time and MPCcontrol == True and i_array % 10 == 0 and Include_battery == True:
    if i_array == mpc_start_time:
        SOC_battery = SOC_initial
    else:
        SOC_battery = battery_state_storage['Timestep:%s' % (int(mpc_time[1]*60))] # remember soc

if P_batt_storage_mpc.NEWVAL*1e3 >= 0:
    batt_efficiency = 1/np.sqrt(0.9)
else:
    batt_efficiency = np.sqrt(0.9)

battery_model_res = 50
battery_current = P_batt_storage_mpc.NEWVAL*1e3/50*np.ones(battery_model_res)*batt_efficiency
battery_time = np.linspace(0, mpc_time[1]*60, battery_model_res)
battery_real = Battery(battery_current, battery_time, total_batt_energy, V_batter, SOC_battery, 1.1, 1.0)

battery_state_storage['Timestep:%s' % (i_array)] = battery_real

if AC_force_off == True:
    HVACSTATUS = 1 # turn hvac off

if kStep != sim_timesstep_tot:
    SP_LO = T_SP_df['Heating SP'][i_array]
    SP_HI = T_SP_df['Cooling SP'][i_array]

    # these are HVAC io, hvac status
    setpoints = [SP_LO, SP_HI, HVACIO[i_array], HVACSTATUS] # third entry is to control the HVAC system NoAction (0), ForceOff (1), CycleOn (2), and CycleOnZoneFanesOnly (3)
    ep = ep.write(ep.encode_packet_simple(setpoints, i_array + deltaT)) # write new value to energyplus simulator

    kStep = kStep + 1 # add one to iteration

except ImportError:
    print('solver results error shutting down ep')
    ep.close()

except KeyboardInterrupt:
    print('shutting down ep')
    ep.close()

# run base case for comparison
print("")
pyEp.set_eplus_dir(Eplus_file_location) # sets energyplus path
builder = pyEp.socket_builder(path_to_buildings) # creates exe path
configs = builder.build() # Configs is [port, building_folder_path, idf]

initialize_sim_files (in seperate .py file)
initialize_sim_files.initialize(EB2opt_installation_path, Eplus_file_location, path_to_buildings, Start_date.month, End_date.month, Start_date.day + 2, End_date.day, Timesteps, location_name, Cooling_SP_T, 0)

# run simulator
ep.base = pyEp.ep.process('localhost', configs[0][0], configs[0][1], weather.name)

outputs_base = []
EPTimeStep = TimeSteps
SimDays = ((End_date - Start_date).days) - 2
KStep = 1
MAXSTEPS = int(SimDays * TimeSteps) #
deltaT = (60 / EPTimeStep) * 60; # seconds per timestep
while (KStep - 1 < MAXSTEPS):
  i_array = KStep - 1
  output_base = ep_base.decode_packet_simple(ep_base.read())
  outputs_base.append(output_base)
  output_base_df = pd.DataFrame.from_records(outputs_base, columns =
  [ 'Drybulb Temp (C)' , 'Zone Air Temp (C)' ,
    'Cooling Temp SP (C)' , 'Heating Temp SP (C)' ,
    'Air System Total Cooling Energy (J)' , 'Total Facility Power Demand (W)' ,
    'HVAC Power (W)' , 'Total Power Generated (W)' ,
    'site diffuse solar radiation' , 'site direct solar radiation' ,
    'site ground reflected solar radiation' , 'pv array eff' ,
    'pv cell temp' , 'pv short circuit current' , 'open circuit voltage' ,
    'Relative Humidity' , 'Wind Speed' , 'Dewpoint' , 'Precip Depth' ,
    'Rain Status' , 'Sky Clearness' ])
  output_base_df[ 'Misc building Power (W)'] = output_base_df[ 'Total Facility Power Demand (W)'] - output_base_df[ 'HVAC Power (W)']
  output_base_df[ 'Water VP (hPa)'] = (output_base_df[ 'Relative Humidity']/100) * 6.105 * np.exp((17.27 * output_base_df[ 'Drybulb Temp (C)']) / (237.7 + output_base_df[ 'Drybulb Temp (C)']))
  output_base_df[ 'Apparent Temp (C)'] = output_base_df[ 'Drybulb Temp (C)'] + 0.4 * output_base_df[ 'Water VP (hPa)'] - 0.7 * output_base_df[ 'Wind Speed']
  output_base_df[ 'Actual Requirement (no battery)'] = output_base_df[ 'Act Demand'] - output_base_df[ 'Total Power Generated (W)'] * AC_DC_invert_eff
  output_base_df[ 'TOD PRICING'] = Total_price_per_timestep_df[ 'price'][:len(output_base_df)]
  output_base_df[ 'NETMETERING PRICING'] = np.ones(len(output_base_df)) * net_meter_price
  if output_base_df[ 'Grid Requirement (no battery)'][i_array] > 0:
    real_P_from_grid = output_base_df[ 'Grid Requirement (no battery)'][i_array]
    real_P_sold_to_grid = 0
  else:
    real_P_sold_to_grid = - output_base_df[ 'Grid Requirement (no battery)'][i_array]
    real_P_from_grid = 0
  Act_Revenue_mpc = real_P_sold_to_grid / 1000 + (Timestep_size/60) * output_base_df[ 'NETMETERING PRICING'][i_array]/100
  Act_Utilities_charge_mpc = real_P_from_grid / 1000 + (Timestep_size/60) * output_base_df[ 'TOD PRICING'][i_array]/100
  Act_profit = Act_Revenue_mpc - Act_Utilities_charge_mpc
  results_df['actual profit with solar normal temp control'][i_array] = Act_profit
if output_base_df[ 'Zone Air Temp (C)'][i_array] > Cooling_SP_T:
  HVACSTATUS = 2 # turn hvac cycle on (still stays within temp limits)
1225     elif output_base_df['Zone Air Temp (C)'][i_array] < Cooling_SP[i_array] - 5.5:
1226         Heating_STATUS = 1  # turn hvac off
1227
1228     if kStep != sim_time_steps:
1229         if i_array % 50 == 0:
1230             print('Step %s complete' %i_array)
1231         setpoints_base = [SP_LO, SP_HI, Heating_SP[i_array], HVAC_STATUS]
1232         ep_base.write(ep_base.encode_packet_simple(setpoints_base, (kStep - 1) * deltaT))
1233         kStep = kStep + 1  
1234     ep_base.close()  
1235     plt.close('all')
1236
1237 fig3, (ax31, ax32, ax33, ax34) = plt.subplots(4, 1)
1238 #setup figure
1239 fig3.tight_layout()
1240 fig3.set_figheight(8)
1241 fig3.set_figwidth(15)
1242 ax32,2 = ax32.twins()
1243
1244 mpc_hour_locator = dates.HourLocator(interval=2)
1245 mpc_hour_formatter = dates.DateFormatter(' %I %p')
1246 mpc_min_locator = dates.MinuteLocator(interval=30)
1247
complete_input_array = args_input = np.ones([15,3])
complete_input_df = pd.DataFrame(complete_input_array)

def complete_frames():
    while True:
        yield complete_input_df

def subplot_31(args):
    #clear plot
    ax31.cla()

    #set data
    t = complete_input_df.ioc[0].values
    Theta_pred = complete_input_df.ioc[1].values
    TA_pred = complete_input_df.ioc[2].values
    SP_HI = complete_input_df.ioc[7].values
    SP_LO = complete_input_df.ioc[8].values

    #setup subplots
    ax31.set_title('Temperatures')
    ax31.set_xlabel('Temp (F)')
    for tick in ax31.get_xticklabels():
        tick.set_rotation(45)
    ax31.xaxis.set_major_locator(mpc_hour_locator)
    ax31.xaxis.set_major_formatter(mpc_hour_formatter)
    ax31.xaxis.set_minor_locator(mpc_min_locator)

    #plot data
    ax31.plot(t, Theta_pred, color='skew: black', Marker = 'o', markersize = 2.5, label = r'T_{air,inside}S')
    ax31.plot(t, TA_pred, color='skew: red', Marker = 'o', markersize = 2.5, label = r'T_{air,ambient}S')
    ax31.plot(t, SP_HI, color='b', linestyle = '--', label = r'SP_{HI}S')
    ax31.plot(t, SP_LO, color='r', linestyle = '--', label = r'SP_{LO}S')

80
def subplot32(args):
    # clear plot
    ax32 cla()
    ax32_2 cla()

    # set data
    t = complete_input_df.iloc[0].values
    Batt_SOC = complete_input_df.iloc[3].values
    IO_HVAC = complete_input_df.iloc[4].values

    # setup subplots
    ax32 set_title('AC Status and Battery SOC')
    ax32 set_xlabel('AC STATUS')
    ax32 set ylim([0, 1.1])
    for tick in ax32 get_xticklabels():
        tick set_rotation(45)
    ax32.xaxis set_major_locator(mpc_hour_locator)
    ax32.xaxis set_major_formatter(mpc_hour_formatter)
    ax32.xaxis set_minor_locator(mpc_min_locator)
    ax32_2 set_ylabel('BATT SOC', rotation=270, labelpad=20)
    ax32_2 set_ylim([0, 1.1])
    for tick in ax32_2 get_xticklabels():
        tick set_rotation(45)
    ax32_2.xaxis set_major_locator(mpc_hour_locator)
    ax32_2.xaxis set_major_formatter(mpc_hour_formatter)
    ax32_2.xaxis set_minor_locator(mpc_min_locator)

    # plot data
    ax32.plot(t[0:10], Batt_SOC[0:10], color='xkcd: green', label=r'$SOC$', ls='steps')
    ax32_2.plot(t, IO_HVAC, color='xkcd: red', label=r'$HVACS 51/05$', ls='steps')
    ax32.fill_between(t[0:10], 0, Batt_SOC[0:10], color='sked: green', alpha=.1)
    ax32_2.fill_between(t, 0, IO_HVAC, color='sked: red', alpha=.1)

    h1, l1 = ax32.get_legend_handles_labels()
    h2, l2 = ax32_2.get_legend_handles_labels()

    # post legend
    return ax32.legend(h1+h2, l1+l2, loc='center left', bbox_to_anchor=(1.055, 0.5))

def subplot33(args):
    # clear plot
    ax33 cla()

    # set data
    t = complete_input_df.iloc[0].values
    TOD_price = complete_input_df.iloc[5].values
    Netmetering_price = complete_input_df.iloc[6].values

    # setup subplots
    ax33 set_title('Pricing')
    ax33 set_ylabel('Cents/kWh')
    for tick in ax33.get_xticklabels():
        tick set_rotation(45)
    ax33.xaxis set_major_locator(mpc_hour_locator)
    ax33.xaxis set_major_formatter(mpc_hour_formatter)
def subplot34(args):
    # clear plot
    ax34.cla()

    # set data
    t = complete_input_df.iloc[0].values
    PV_prod = complete_input_df.iloc[11].values
    P_from_grid = complete_input_df.iloc[12].values
    P_to_grid = complete_input_df.iloc[14].values

    # setup subplots
    ax34.set_title("Net Energy")
    ax34.set_xlabel("Usage/Production (W)")
    ax34.set_xlim((-2000, 6500))
    for tick in ax34.get_xticklabels():
        tick.set_rotation(45)
    ax34.xaxis.set_major_locator(mpc_hour_locator)
    ax34.xaxis.set_major_formatter(mpc_hour_formatter)
    ax34.xaxis.set_minor_locator(mpc_min_locator)

    # plot data
    ax34.plot(t, PV_prod, color='skc orange', label=r'$$Power$$ $$Produced$$', ls='steps')
    ax34.plot(t, P_from_grid, color='skc red', label='Power from grid', ls='steps')
    ax34.plot(t, P_to_grid, color='skc green', label='Power sold back to grid', ls='steps')
    ax34.plot(t, np.zeros(len(t)), color='skc black')
    ax34.fill_between(t, 0, -P_from_grid, color='skc red', alpha=1)
    ax34.fill_between(t, 0, -P_to_grid, color='skc green', alpha=1)
    return ax34.legend(loc='center left', bbox_to_anchor=(1, 0.5))

def animate3(args):
    return subplot31(args), subplot32(args), subplot33(args), subplot34(args), fig3.tight_layout()
complete_input_df.iloc[13] = output_base_df["Act Demand"].values
plt.pause(3)
fig3.savefig('plots/%s\%s\%s\actual_base_case_tod.eps' % (location_name, house_label))

complete_input_array = args_input = np.ones([15, len(output_df[mpc_start_time :].values)])
complete_input_df = pd.DataFrame(complete_input_array)

# change mpc array to datetimes
complete_input_df.iloc[0] = dates.date2num(output_df["Date and Time"].mpc_start_time.values)
complete_input_df.iloc[1] = simf.ctoof(output_df["Zone Air Temp (C)"].mpc_start_time.values)
complete_input_df.iloc[2] = simf.ctoof(output_df["Drybulb Temp (C)"].mpc_start_time.values)

mpc_sol_storage["Timestep:%s" %i_array] = mpc_sol

soc_actual_array = np.zeros(len(mpc_sol_storage))
for i, j in zip(mpc_sol_storage, range(len(mpc_sol_storage))):
soc_actual_array[j] = mpc_sol_storage["%s %s"]['soc'][0]

for i in range(len(output_df[mpc_start_time :].values)):
    if i % 10 == 0:
        complete_input_df.iloc[3, i] = soc_actual_array[int(i/10)]

complete_input_df.iloc[4] = output_df["HVAC I/O"].mpc_start_time.values
complete_input_df.iloc[5] = output_df["TOD PRICING"].mpc_start_time.values
complete_input_df.iloc[6] = output_df["NETMETERING PRICING"].mpc_start_time.values

complete_input_df.iloc[7] = simf.ctoof(CoolingSP_T) * np.ones(len(output_df[mpc_start_time :].values))
complete_input_df.iloc[8] = simf.ctoof(HeatingSP_T) * np.ones(len(output_df[mpc_start_time :].values))

complete_input_df.iloc[11] = output_df["Total Power Generated (W)"].mpc_start_time.values
complete_input_df.iloc[12] = output_df["Grid Requirement (no battery)"].mpc_start_time.clip(lower = 0).values
complete_input_df.iloc[14] = output_df["Grid Requirement (no battery)"].mpc_start_time.clip(upper = 0).values

complete_input_df.iloc[13] = output_df["Act Demand"].mpc_start_time.values
plt.pause(3)
fig3.savefig('plots/%s\%s\%s\actual_optimized_case_tod.eps' % (location_name, house_label))
APPENDIX C. ENERGYPLUS CONFIGURATION FILE

C.1 EnergyPlus .idf file

Version,
8.8;  !− Version Identifier

SimulationControl,
Yes,  !− Do Zone Sizing Calculation
No,  !− Do System Sizing Calculation
Yes,  !− Do Plant Sizing Calculation
No,  !− Run Simulation for Sizing Periods
Yes;  !− Run Simulation for Weather File Run Periods

Building,
Phoenix Sky Harbor Int’l Ap AZ 3350 sqft,  !− Name
180,  !− North Axis {deg}
Suburbs,  !− Terrain
0.05,  !− Loads Convergence Tolerance Value
0.05,  !− Temperature Convergence Tolerance Value {deltaC}
FullExterior,  !− Solar Distribution {FullExterior = Shading from/on exterior surfaces. All interior direct solar falls on the floor and is absorbed according to the floor’s absorptivity.}
50;  !− Maximum Number of Warmup Days

ShadowCalculation,
AverageOverDaysInFrequency,  !− Calculation Method
20,  !− Calculation Frequency
200;  !− Maximum Figures in Shadow Overlap Calculations

Output: Diagnostics,
DisplayAdvancedReportVariables;  !− Key 1

SurfaceConvectionAlgorithm: Inside,  !− Algorithm
TARP;  !− Algorithm

SurfaceConvectionAlgorithm: Outside,
DOE−2;  !− Algorithm

HeatBalanceAlgorithm,
ConductionTransferFunction,  !− Algorithm
200;  !− Surface Temperature Upper Limit {degC}

ZoneCapacitanceMultiplier: ResearchSpecial,
Multiplier,  !− Name
.  !− Zone or ZoneList Name
1;  !− Temperature Capacitance Multiplier
15;  !− Humidity Capacitance Multiplier
1;  !− Carbon Dioxide Capacity Multiplier

Timestep,
60;  !− Number of Timesteps per Hour
Convergence Limits,
1. Minimum System Timestep {min}
20. Maximum HVAC Iterations

Site: Location,
Phoenix Sky Harbor Intl Ap AZ, Name
33.45, Latitude {deg}
-111.98, Longitude {deg}
-7, Time Zone {hr}
336.9999864; Elevation {m}

Foundation: Kiva: Settings,
1.731, Soil Conductivity [W/m-K]
1842.3, Soil Density [kg/m3]
418.70000000000005, Soil Specific Heat [J/kg-K]
0.9, Ground Solar Absorptivity
0.9, Ground Thermal Absorptivity
0.03, Ground Surface Roughness [m]
25, Far-Field Width [m]
ZeroFlux, Deep-Ground Boundary Condition
40, Deep-Ground Depth [m]
0.02, Minimum Cell Dimension [m]
1.5, Maximum Cell Growth Coefficient Hourly
Simulation Timestep

Site: GroundTemperature: BuildingSurface,
19.507568933771957, Jan Ground Temperature [degC]
17.68388763001127, Feb Ground Temperature [degC]
17.537840691095834, Mar Ground Temperature [degC]
18.403270653280558, Apr Ground Temperature [degC]
21.877489695966784, May Ground Temperature [degC]
25.335524352510895, Jun Ground Temperature [degC]
28.33455587669853, Jul Ground Temperature [degC]
30.23512925260984, Aug Ground Temperature [degC]
30.3931602755722, Sep Ground Temperature [degC]
28.849500174639487, Oct Ground Temperature [degC]
25.92781794610709, Nov Ground Temperature [degC]
22.57586170800307, Dec Ground Temperature [degC]

Site: GroundTemperature: Deep,
23.8, Jan Deep Ground Temperature [C]
23.8, Feb Deep Ground Temperature [C]
23.8, Mar Deep Ground Temperature [C]
23.8, Apr Deep Ground Temperature [C]
23.8, May Deep Ground Temperature [C]
23.8, Jun Deep Ground Temperature [C]
23.8, Jul Deep Ground Temperature [C]
23.8, Aug Deep Ground Temperature [C]
23.8, Sep Deep Ground Temperature [C]
23.8, Oct Deep Ground Temperature [C]
23.8, Nov Deep Ground Temperature [C]
23.8, Dec Deep Ground Temperature [C]

Site: WaterMainsTemperature,
Correlation,
23.8, Annual Average Outdoor Air Temperature [C]
23.84444444444445, Maximum Difference In Monthly Average Outdoor Air Temperature [deltaC]

The full EnergyPlus config.idf file can be found at https://github.com/CSimsRun/Masters_Thesis.git