Efficient Allocation of Grid Energy Resources including Storage (EAGERS)

Economic dispatch for micro-grids and district energy systems presents a highly constrained non-linear, mixed-integer optimization problem that scales exponentially with the number of systems. Energy storage technologies compound the mixed-integer or unit-commitment problem by necessitating simultaneous optimization over the applicable time horizon of the energy storage. EAGERS greatly reduces, and under some conditions eliminates, the mixed-integer aspect of the problem using complementary convex quadratic optimizations. The generalized method applies to grid-connected or islanded district energy systems comprised of any variety of electric or combined heat and power generators, electric chillers, heaters, and all varieties of energy storage systems. The approach satisfies local spinning reserve constraints and represents participation in demand response and spinning reserve markets. It incorporates constraints for generator operating bounds, ramping limitations, network line losses, transmission constraints, and energy storage inefficiencies.

Table of Contents

[II. Overview 1](#_Toc505880759)

[i. Getting Started 1](#_Toc505880760)

[ii. Starting a New Project 1](#_Toc505880761)

[III. Planning Tool 1](#_Toc505880762)

[i. The Planning Tool Interface 2](#_Toc505880763)

[IV. Control Tool 6](#_Toc505880764)

[i. The Control Tool Interface 7](#_Toc505880765)

[ii. Forecasting 10](#_Toc505880766)

[iii. Dispatch Scheduling 15](#_Toc505880767)

[iv. Reduced Mixed-integer Problem 32](#_Toc505880768)

[v. Component Model Descriptions 36](#_Toc505880769)

[vi. Real-time Model Predictive Control 38](#_Toc505880770)

[V. Simulation Tool (STRIDES) 39](#_Toc505880771)

[i. Simulation Tool Interface 39](#_Toc505880772)

[ii. System Model Files 39](#_Toc505880773)

[iii. Component Functions 43](#_Toc505880774)

[iv. Controller Functions 43](#_Toc505880775)

[v. Additional Reference Functions 43](#_Toc505880776)

[vi. Description of Specific Components 43](#_Toc505880777)

[vii. Description of Specific Models 43](#_Toc505880778)

[VI. Glossary of Important Variables 43](#_Toc505880779)

[i. Project Variables 43](#_Toc505880780)

[ii. Planning Tool Variables 43](#_Toc505880781)

[iii. Control Tool Variables 43](#_Toc505880782)

[iv. Simulation Tool Variables 43](#_Toc505880783)

[VII. Interface Flow Diagrams 43](#_Toc505880784)

[v. 1. EAGERS Interface 43](#_Toc505880785)

[VIII. Function Summaries 43](#_Toc505880786)

[IX. Appendix 43](#_Toc505880787)

# Overview

The EAGERS platform is an open source tool for designing, controlling, and simulating energy systems, with a focus on optimal control of distributed energy resources.

## Getting Started

Open Matlab and navigate to the EAGERS/main directory. Type the command ‘EAGERS’to launch window of Figure 1. From this window you can select from a pre-saved project, or start a new project, then launch directly into either the planning, optimization, or simulation tools.

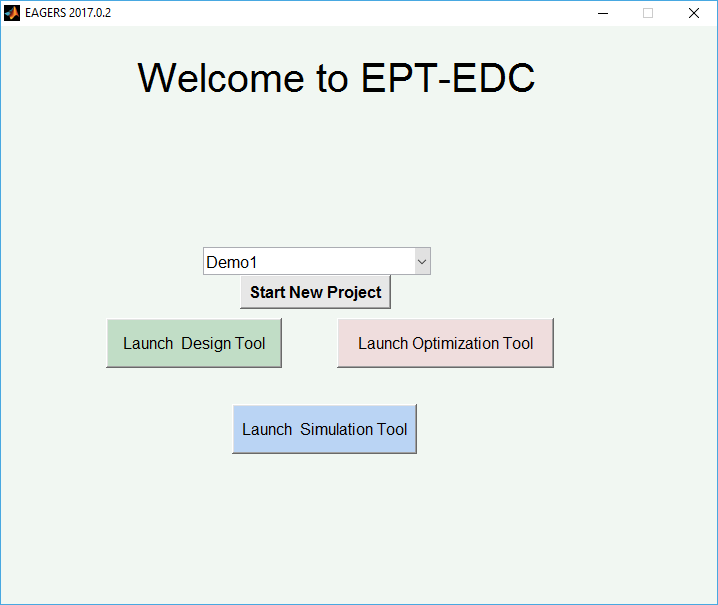


Figure 1 EAGERS welcome screen

## Starting a New Project

If Start New Project is selected the user can either load data files, or construct a building demand from the prototypical building models available. Buildings can be added to the microgrid by highlighting the building type, climate zone/city, and vintage desired, then selecting Add under ‘Current Load Profile/s.’ To create data for heating and cooling demands, the District Cooling and District Heating boxes must be checked.

# Planning Tool

The EAGERS planning tool has five tabs: Main Window, Building Spec, System Spec, Network Spec, and Settings. The purpose of the planning tool is to identify the optimal configuration and size of distributed energy equipment for the users’ needs and preferences. If the user has building data for the site of interest, they can upload that actual historical data. Otherwise, they can calibrate one of the prototypical buildings to match their occupant use profiles. Different equipment can be readily added, resized and removed to test different system configurations.

## The Planning Tool Interface

### Main Window

The main window allows you to view data, i.e. electric, heating and cooling demands, for a project, and view the resulting cost/benefit analysis for meeting those demands with the system specified on the System Spec tab. The user can scroll through the simulated or actual building data, viewing a single day, a week, a month, or a year’s data at a time. The histogram to the right illustrates the distribution of the demand. Multiple projects can be added within this window to compare the cost/benefit analysis for different designs. The cost for individual equipment is input in the table near the bottom.

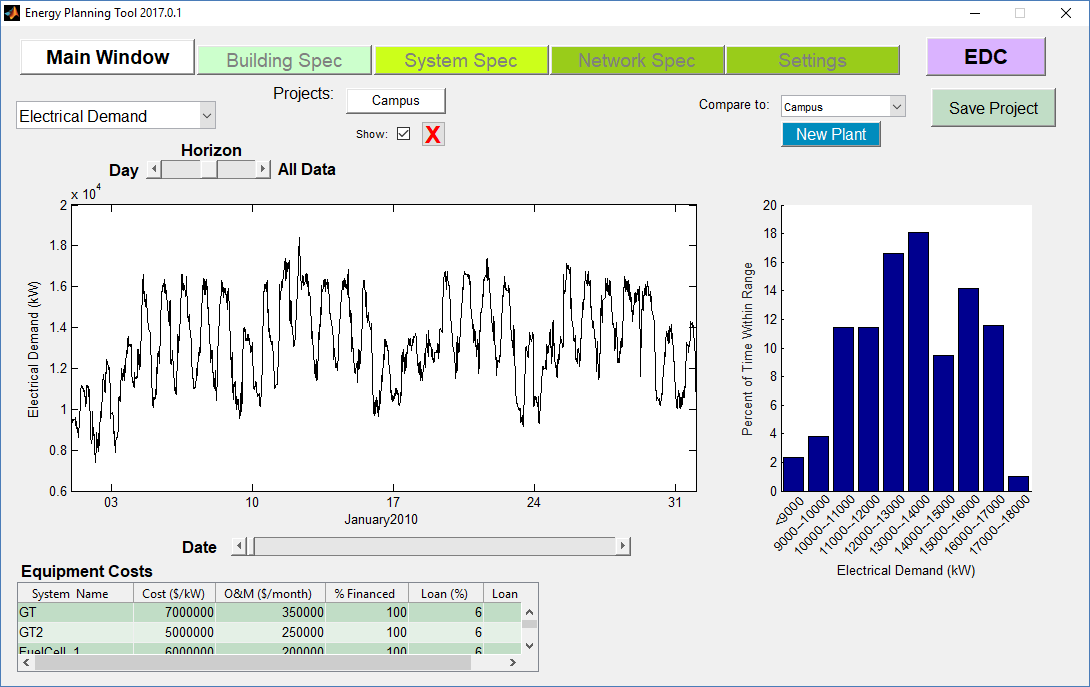


Figure 2 Primary planning tool GUI

### Building Spec

The Building specification tab allows the user to upload historical building use data for their site, or configure a prototypical building. Pressing any one of the Schedule buttons opens a new window that allows the user to define building occupancy, lighting or equipment schedules in a similar fashion to EnergyPlus. The user can click on the window to drag and edit points, and choose to be step changes or linear interpolation between the specified points.

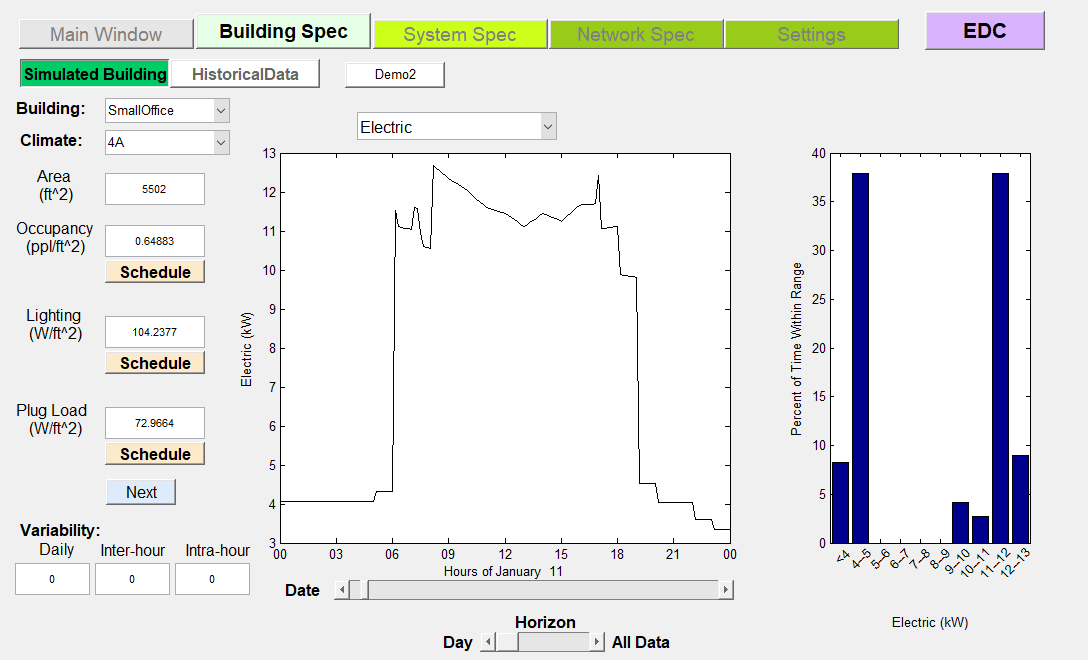


Figure 3 Building Spec Tab in the Planning Tool

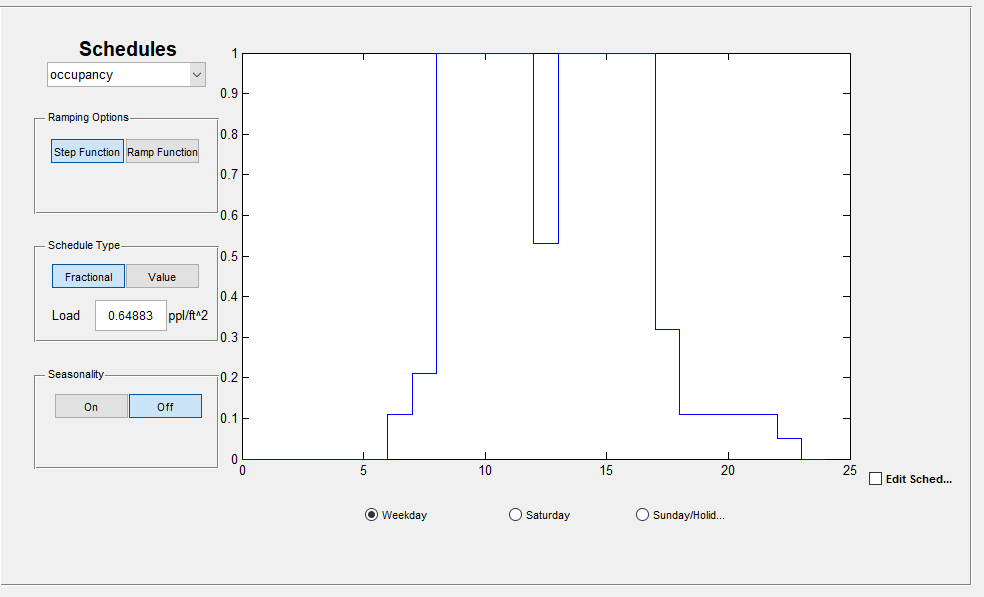


Figure 4 Schedule Editor Window

### System Spec

The System Spec tab defines the equipment and utilities connected in the project. There are 5 bus bars shown on the left side of the window: AC Bus, DC bus, Heating (hot air), Hot Water, and Cooling (cold water). Individual systems can be connected to one or two busses. If the project does not include any items of a particular category, then those items do not appear connected to the bus bars. The library allows users to add equipment from a default or user-defined library of equipment. If the user wishes to define multiple items of the same type, e.g. two chillers, both would be accessible from the same button on the left had side of the tab. The user can click on any of the items in the project, and the right side will allow the parameters of that object to be edited.

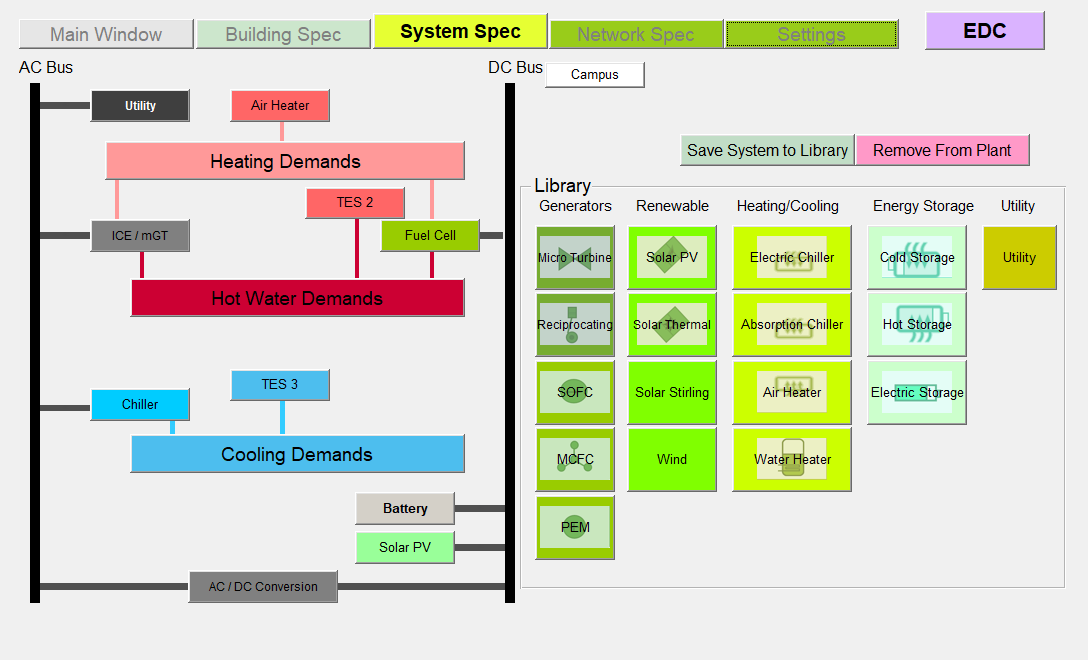
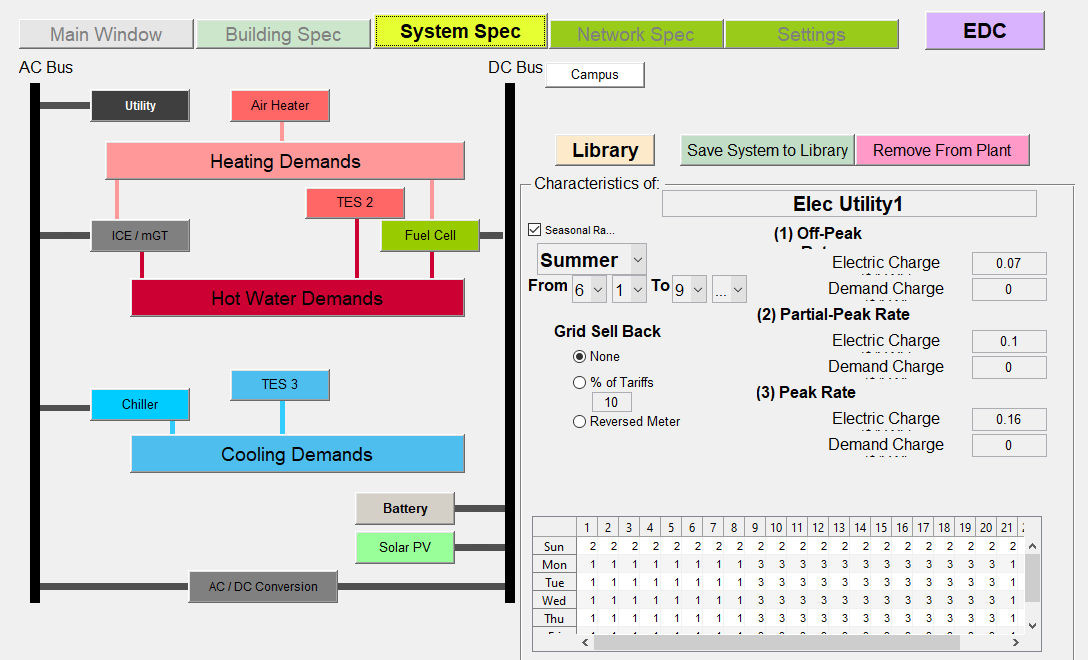
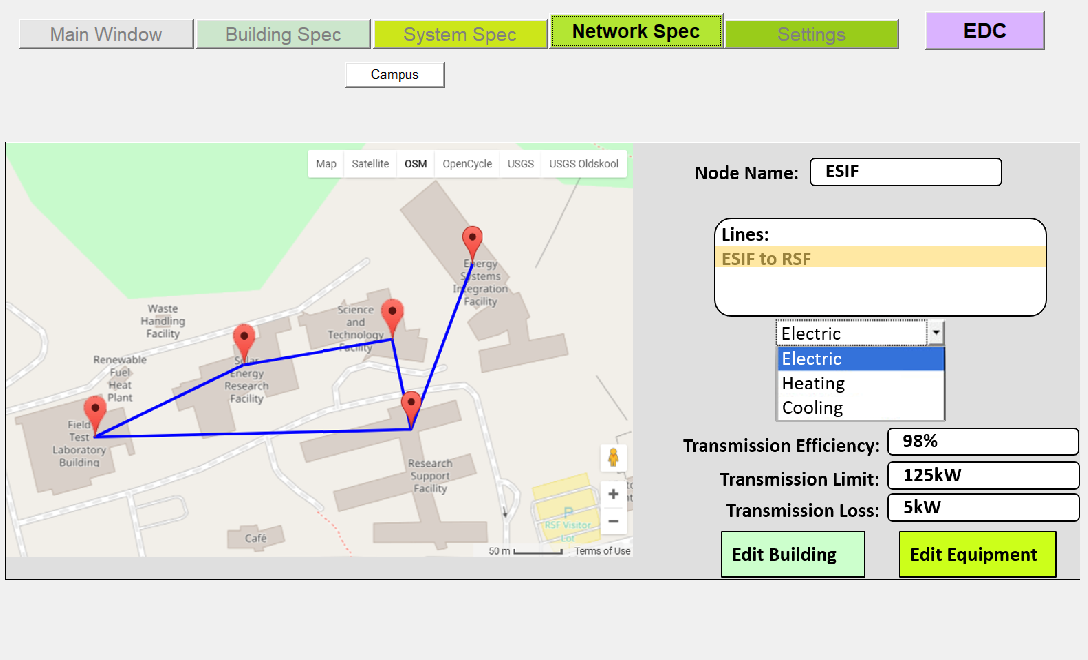


Figure 5 System specification tab

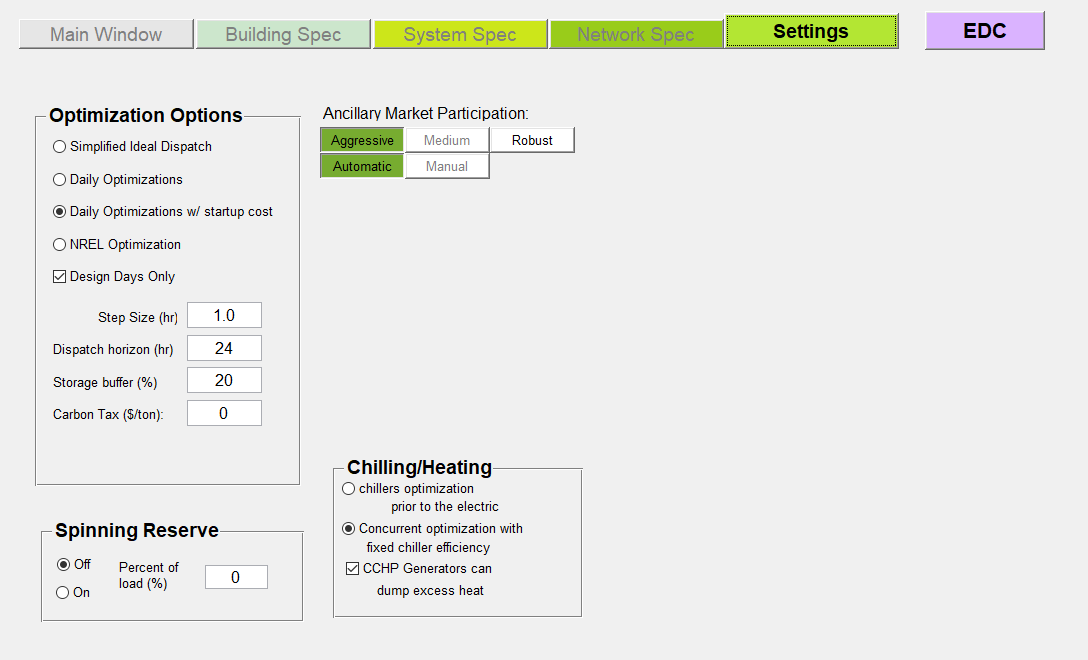


### Network Spec

For some DER equipment, location maters. A district heating or district cooling loop can be specified, and equipment placed at the correct node on that network. In this manner local conditions, e.g. flow and temperature, are seen by the appropriate equipment.



### Settings



# Control Tool

The objective of the control tool is to optimize the dispatch of resources for a given forecast, then control the plant in real-time to approximate the optimal dispatch given the uncertainty in the forecast. The control tool uses the set of energy resources and energy demands assembled in the planning tool, and links to the non-linear systems models of the simulation tool. Figure 6 outlines in block diagram the information pathways between the forecasting, dispatch, control and simulation components of the control tool.

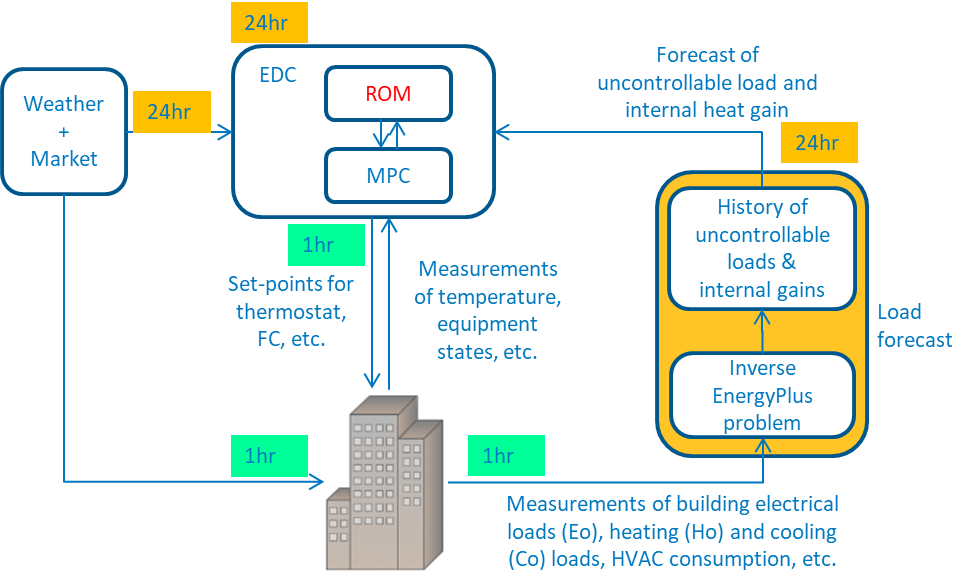


Figure 6 Overview of control tool structure

The approximate steps taken by the controller to simulate the control of a building are:

1. Forecast entire building demand (with stochastic probabilities?)
2. Optimize dispatch over entire horizon (on/off & approximate SOC)
3. Apply MPC over shorter horizon using system identification, SID, model with target SOC from ii).
4. Evaluate with non-linear energy systems models & non-linear SID + load uncertainty + energy system model error + measurement noise

## The Control Tool Interface

### Main Window

The main interface with the control tool illustrates the planned dispatch for the building and DER equipment. The available equipment is listed on the left. The status lights indicate the availability: green is on, yellow is standby, and red is offline. Clicking any of these items will update the status in the lower left. In the example shown in Figure 7 there are electrical demands, heating demands, and cooling demands that must be met. Clicking the buttons at the top of each graph will switch which graph is the large primary graph. Additional graphs are possible, e.g. building zone temperature. The charts can switch between line graphs, and the stacked bar charts shown below. The Start/Stop buttons begin or end a simulation based on the initial slider position for date at the bottom.

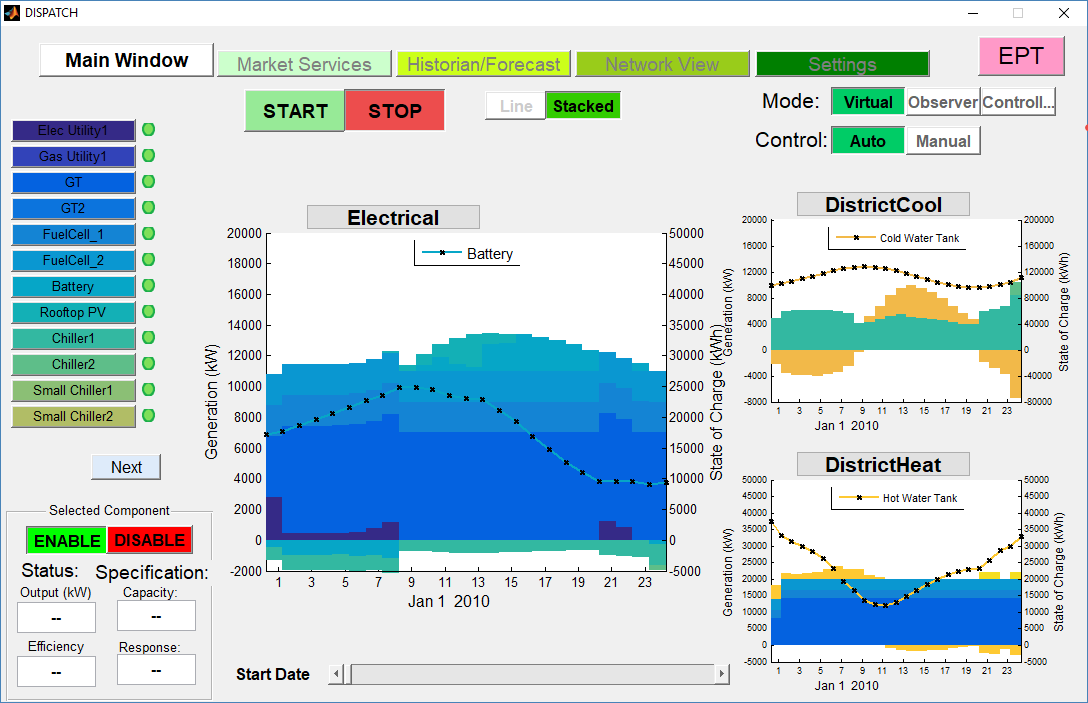
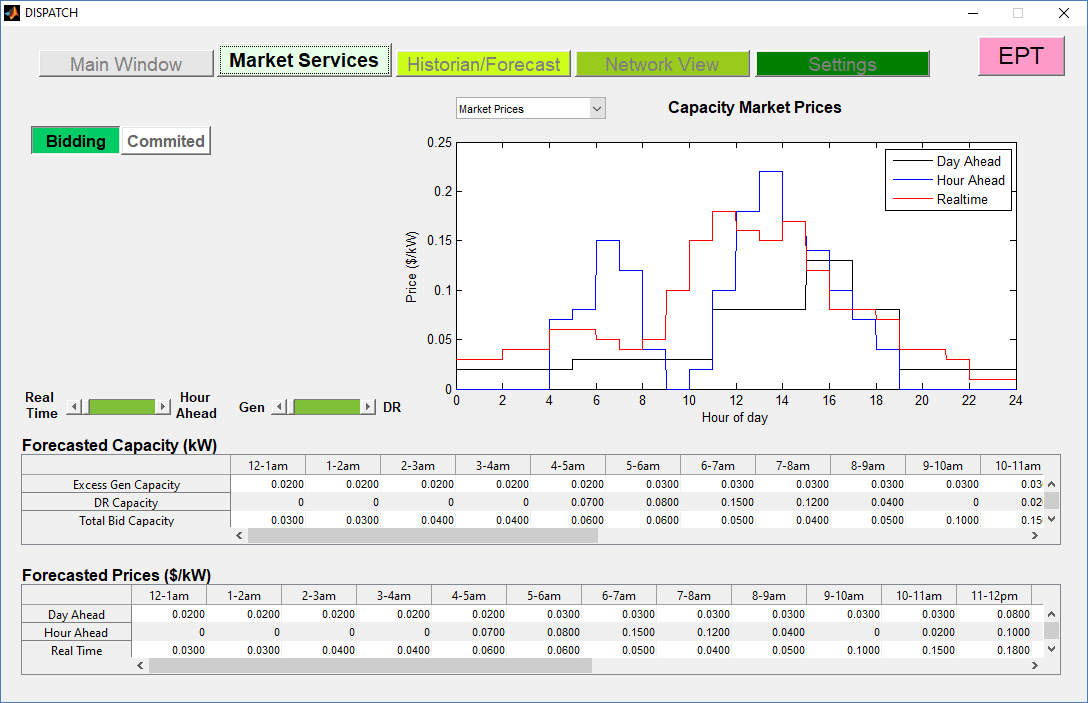


Figure 7 Main control GUI window

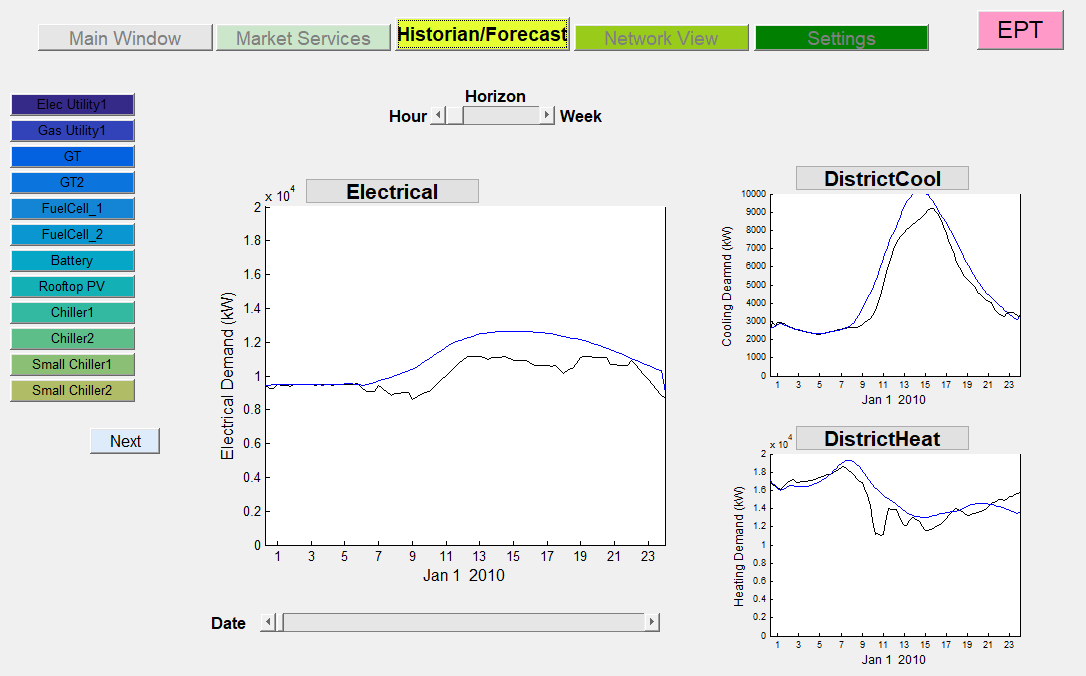
### Market Services

The market services tab illustrates the forecasted ancillary market prces for diferent services, and allows the user to specify their participation in each.



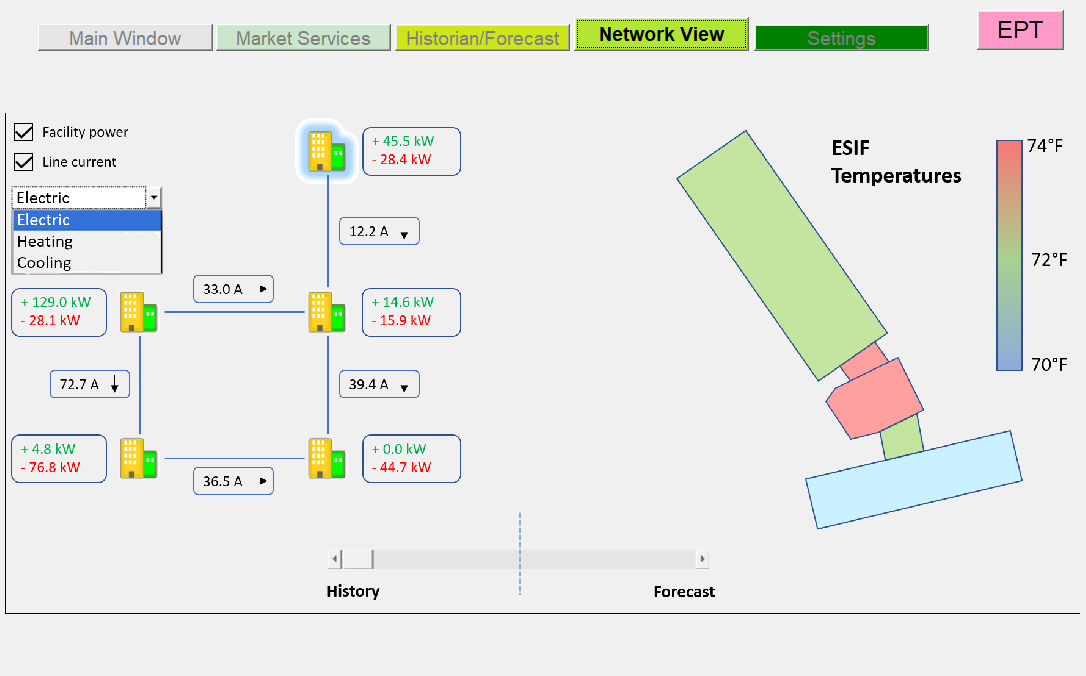
### Historian/Forecast

The Historian/Forecast tab allows the user to scroll through historical data for building demand, equipment use, or weather.



### Network View

The network view tab allows the user to see the realtime status of energy transfer within the microgrid, district heating, or district cooling loop. This can also visually represent the current building temperature.



### Settings

The control specification tab offers the user a suite of options for the control optimization.

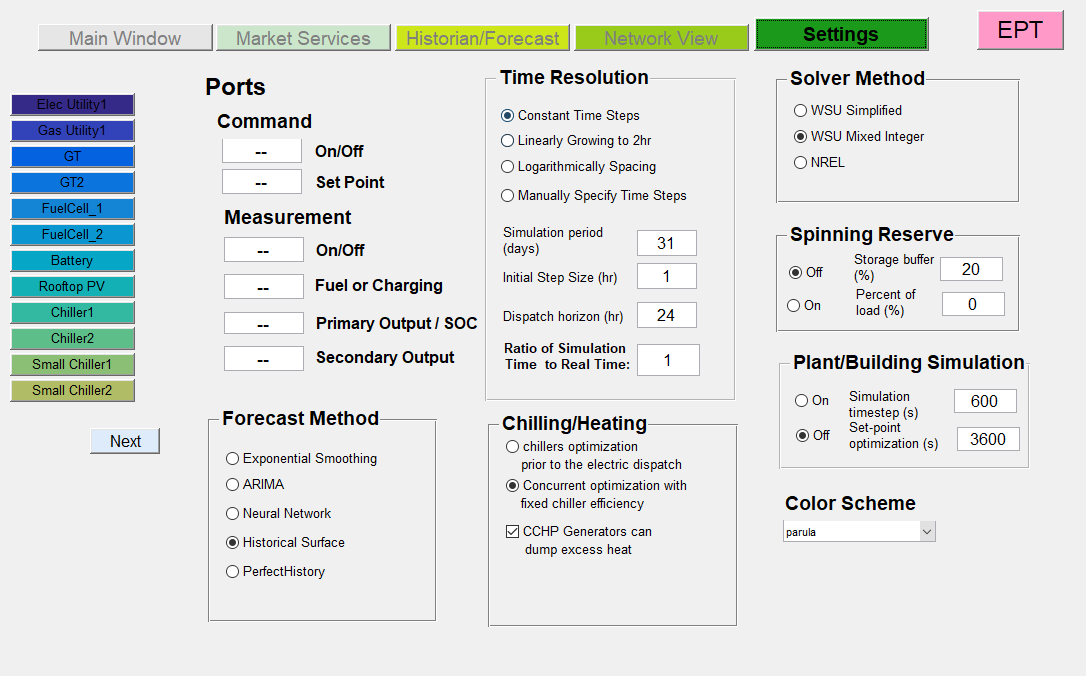


Figure 8 Optimization specification GUI

Selecting Optimization Options allows the user to characterize the optimization. A new window opens with the options of a fast simulation or a simulation that is run at real time. If a Fast Simulation is selected, then the ratio of simulation time to real time must be specified; this ratio can be ignored if running in real time. By changing the ratio at which the simulations run in relation to real time, this variable scales the capacity at which a storage device performs. The power output for the storage devices is calculated to be the change in state of charge multiplied by the amount of time it took to make that change; since the time is now running at a faster/slower speed, the capacity must be scaled to account for this change. The ‘Meet heat demand within \_\_ hr’ option lets the user designate any delay that might occur while trying to meet demand. This heat demand tolerance allows for the heat demand to be met without the inclusion of district heating. It also helps prevent CHP (combined heat and power) generators from being controlled by heat demand instead of electric demand.

The Optimized Timestep Resolution allows the user to pick between one of 3 options. Constant time steps uses time intervals equal to the initial step size selected in the Dispatch Parameters. Logarithmically spacing uses a set of 8 time intervals of growing size between the initial time and the end of the dispatch horizon. Manually specify time steps, allows the user to specify the time intervals of the dispatch optimization.

The Chiller Optimization options will dictate whether the optimization will be run before electric dispatch or concurrently with the electric dispatch. Running the optimization before the dispatch will add the power used to run the chillers to the electric demand. An assumed fixed chiller efficiency will be required if the user chooses to run them concurrently.

The Dispatch Parameters govern the various modifications of the optimization frequencies in the dispatch. The simulation period dictates the length of the simulation. The initial step size is the duration of time at which the dispatch is optimized. The dispatch horizon determines how far into the future the dispatch predicts demand and generator dispatch. The set-point frequency regulates how often (within an optimization step) the online generators are given a new optimized set-point to meet demand. The MPC frequency is how often the grid balance is checked and adjusted to make sure demands are met. Selecting the option in the bottom corner ‘CCHP Generators can dump excess heat’ allows for the generators to produce more heat than is needed to meet the demand and dump the excess. This option helps prevent generators from being limited by the heat demand.

## Forecasting

### Perfect Forecasting

This option interpolates historical data in the same way that the test data is created. It is akin to having perfect foreknowledge of the demands.

### Simple Exponential Smoothing

The simple exponential smoothing (SES) algorithm forecasts future data as the sum of past data points with weights. The weight given to past data points decreases as you go further into the past and by a factor alpha (α). This single weighting factor can be “trained” with a relatively small historical data set. The forecast equation described in (1)

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Clearly larger alpha’s place greater emphasis on the most recent point, while smaller alpha’s will more closely approximate the long-term moving average. Figure 5 illustrates the historical weighting as a function of alpha. The effect of data beyond the seventh value into the past on the next forecast is less than 5% for any alpha of 0.2 or greater. Looking at equation (1) it may be apparent that the forecast at t+1 is simply (1-α) multiplied by the previous forecast. Our current forecasting method exploits this to only require the most recent forecast.

The SESTrain(data) function takes in a list of data for training the simple exponential smoothing (SES) level smoothing parameter, alpha. It assumes 24-hour periodicity in the training data. Given a list of training data, it will output the optimal value of alpha for that data. It will also display some plots:

1. R-squared Values for Corresponding Alphas – This plot displays the R-squared values for each alpha value that was tested in the optimization for both basic and corrected forecasts.
2. Forecast – Displays the forecasted data on the same set of axes as the training data using the optimized alpha for the forecast.
3. Corrected Forecast – Displays the data forecasted using a corrected forecast that takes some error in the last forecast into account.
4. Forecast Error – Displays the error in the basic forecast for each hour where the forecast and training data overlap.

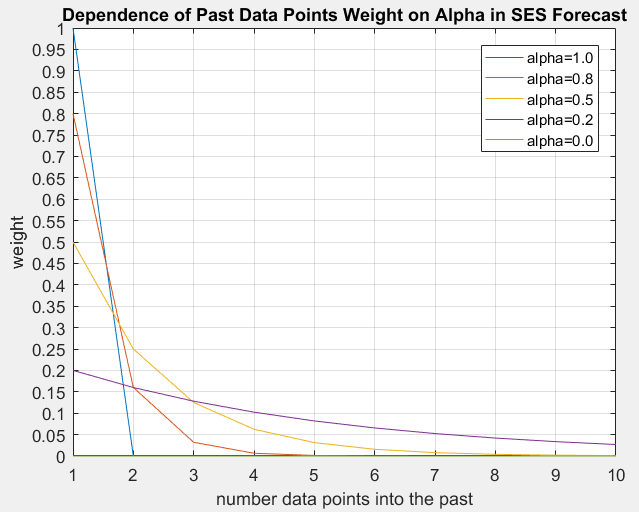


Figure 9 a) Historical weighting of data in SES method, b) R2 values as a function of alpha

The SESForecast(histData, histFcast, alpha) function takes in two lists – histData and histFcast – and one numerical value between 0 and 1 – alpha. There will need to be some initial values input histFcast. These might be all zeros, or the same values as contained in the input histData.

### Auto-regressive integrated moving average (ARIMA):

### Neural Network:

This approach uses historical data for demand and temperature to train an artificial neural network. Once trained, the forecasting is quite quick and accurate for real building applications. It is less accurate for EnergyPlus buildings due to the sudden ramping periods. The drawback of this approach is the significant historical data that is required to properly train the network, e.g. several months. The advantage of this method is the well-defined method for on-line updating of parameters, e.g. machine learning, that can be applied in a real-time environment. can be trained to forecast demand for a horizon when given the temperature forecast, the day of the year, and the time of day for each forecast.

The supplied method uses a one layer ANN with weights and biases with a sigmoid activation function. The network is trained using cross entropy and momentum techniques. This network also employs L2 regularization with lambda of 0.5 to prevent overfitting. The training inputs are: Demand, Time of day, Day of year, and Ambient Temperature. The forecasting inputs are thus: Time of day, Day of year, and Forecasted Ambient Temperature.

The sample results below were generated with a single building from the NYSERDA database. The network well predicts the electric demand, but struggles with heat demand due to the low quality of the heat demand data. It is more difficult to filter heat demand because zero demand does not imply an incorrect data value, yet all incorrect data values appear as zero. Training with nearly 1 year of hourly data required significant computational time, but forecasting proceeds quickly. As with most forecasting techniques, the neural network is a better predictor of moderate behavior near the mean, and does not capture many of the outliers.

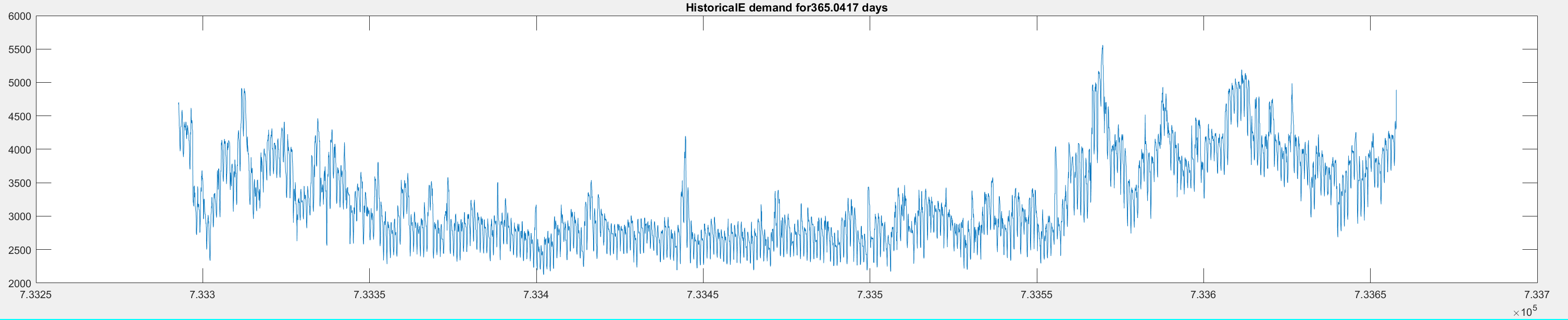


Figure 10 Historical electric data used for training

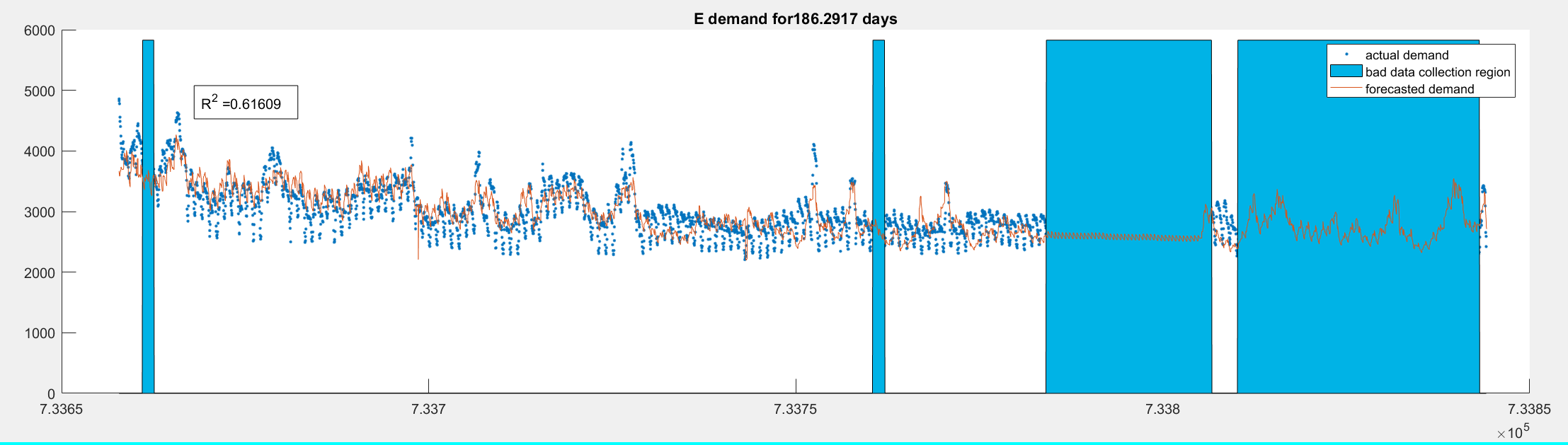


Figure 11 Data and network dispatch for electric demand from December to March.

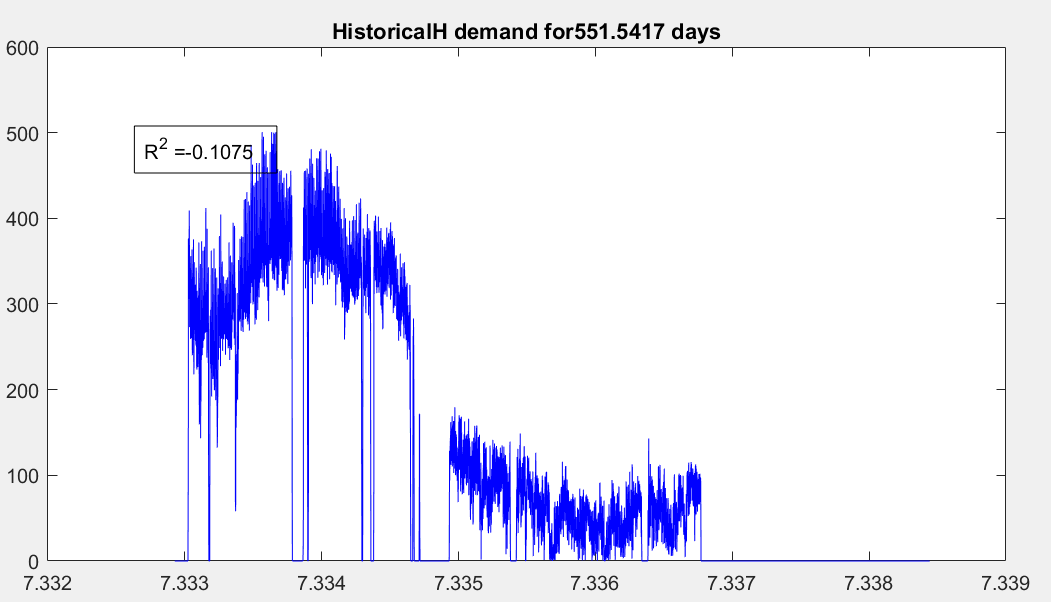


Figure 12 Heating demand, filtered data

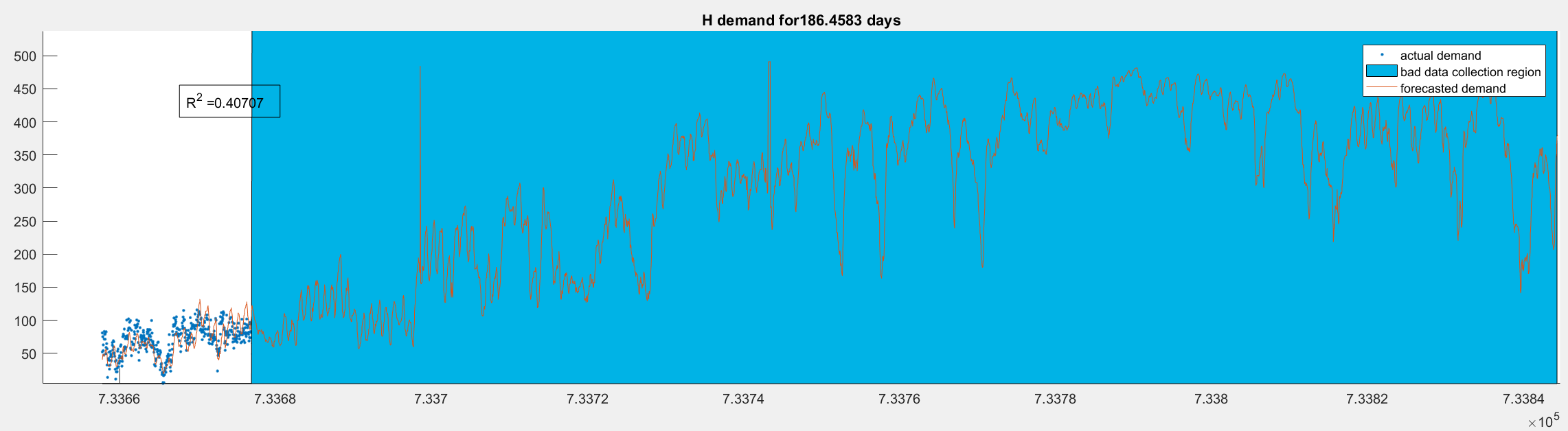


Figure 13 Heating demand forecasting

### Historical Surface Fits

Forecasting temperature is done by averaging the values from the prior day with the historical data for the current day and region. This average is then smoothed to produce the projected temperatures for the next 24 hours, Figure 14.To avoid any discontinuities in the weather pattern, the first forecasted temperature will always match the last actual temperature from the previous day; this provides the baseline for the smoothed temperature for the next horizon, Figure 15.

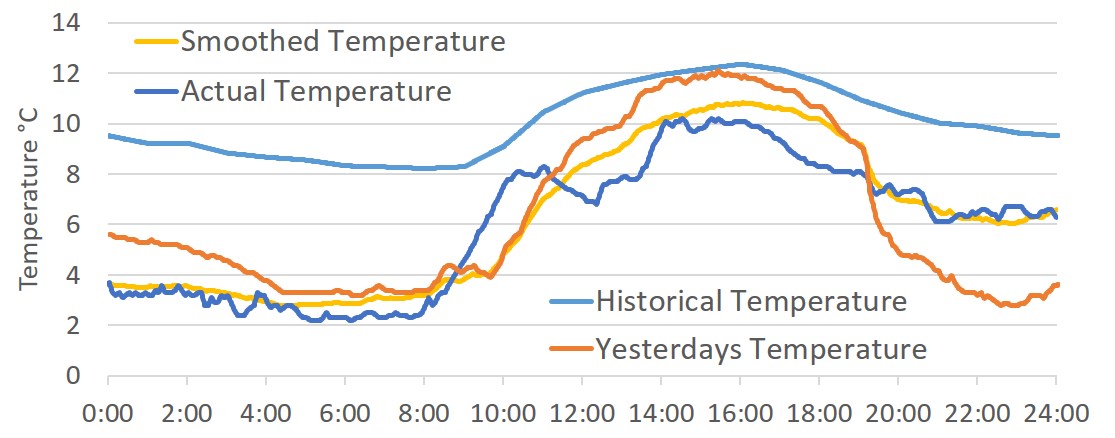


Figure 14 The temperature from yesterday has been averaged with the historical, then smoothed to form a prediction. This prediction is then compared to the actual temperature.

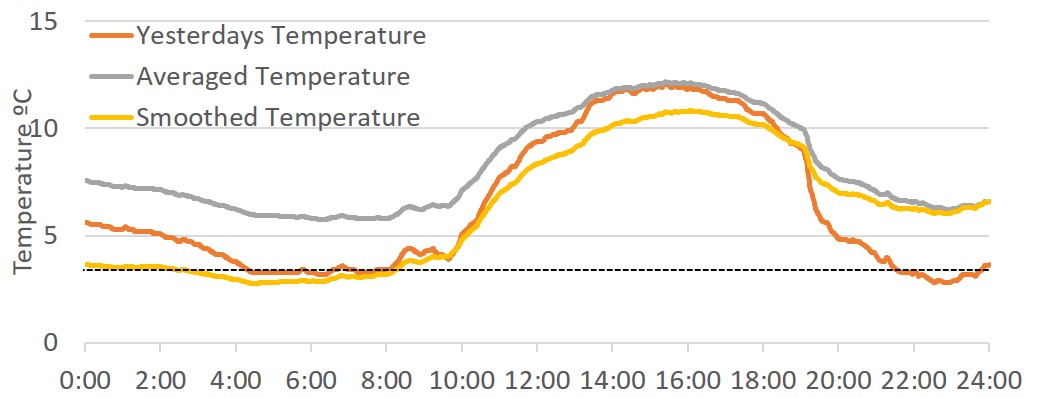


Figure 15 The last recorded temperature from yesterday is used to the determine the base for the smoothed temperature, which is then fit with the average of yesterday and the historical.

The demand of each component in a generator set is dependent on the actual temperature, therefore forecasting is crucial to accurately predicting usage. In the same way temperature is forecasted, the loads for electricity, heating and cooling are calculated. An average from the previous day’s load and the historical load for the forecasted temperature are used to predict a surface fit for the load. Different surfaces are used for weekdays and weekends/holidays to ensure that the most accurate prediction is being made.

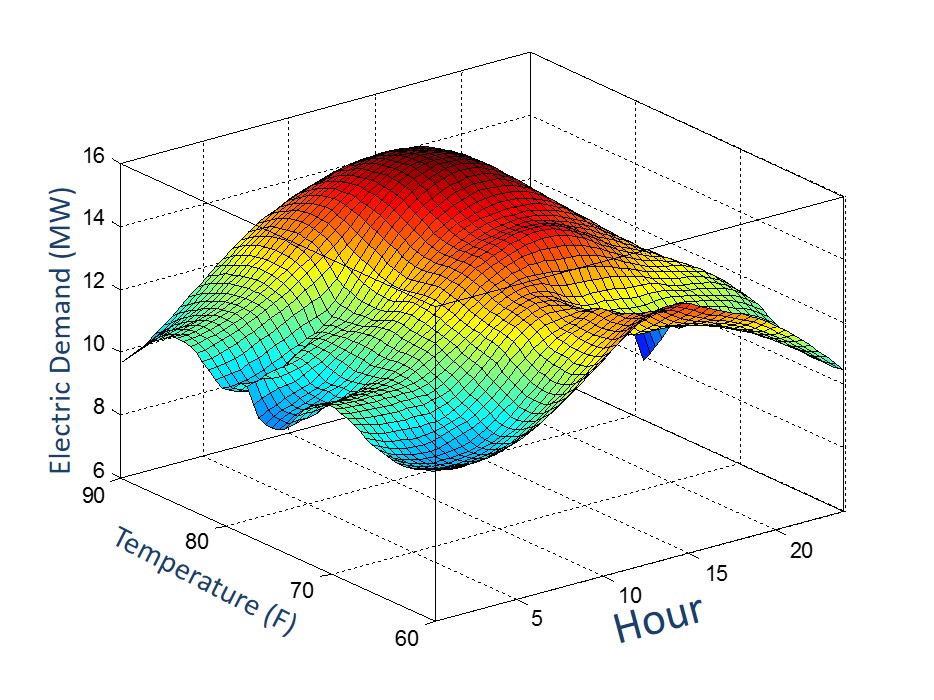


Figure 16 The surface fit made by forecasting electric load from averaging yesterday's and historical loads

For any dispatch with long timesteps (larger than 1 hour), power will be forecasted. This is done by looking at KWh over the last hour and predicting the power for the next hour, using left handed trapezoid integration.

## Dispatch Scheduling

The disparate time scales of equipment involved suggest a multi-tiered or hierarchal optimization approach would be best suited. A high level optimization over a 24+ hour time horizon, for which certain assumptions become valid, can determine the boundary conditions for a short-term, <1hr, optimization of equipment and building HVAC set-points. Assumptions at the 24-hour time scale include:

1. A ‘smoother’ building energy use profile by aggregating variable or cyclic loads into 1-hr blocks of energy use.
2. Fast responding equipment can operate at any point within its allowable range.
3. Slow responding equipment can be approximated with a linear ramp rate, avoiding the complexity of a 1st or 2nd order response model.
4. Building temperature can achieve precisely the desired set-point, eliminating the need for a dynamic building model.

The short-term optimization makes only three assumptions, but critically these enable formulation as a standard ‘Model Predictive Control’ problem (MPC) and avoid any mixed-integer elements.

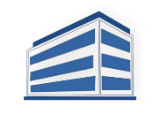
1. The state (on/off) of each system is known for the current time horizon (1hr), and any changes of state occur at a linear combination of known thresholds in time, demand, or SOC.
2. The target conditions are known for the end of the current horizon (on/off and SOC for storage).
3. The costs/value of different systems and outputs are known over the current horizon.

Connecting the hourly economic dispatch with its associated uncertainty to the real-time MPC requires two intermediate steps: determination of the switching thresholds, and a re-optimization of the active generator set points. Figure 17 illustrates the connection between the optimization and the individual equipment.

**Forecast**

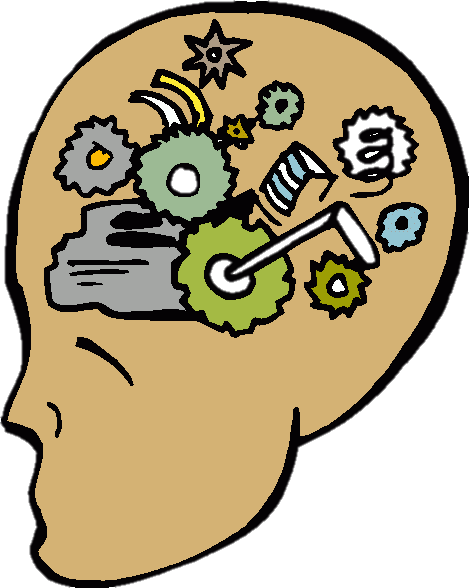
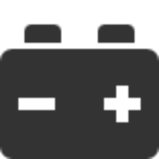
**(Load &prices)**

**On-Line Optimization** (<1min)

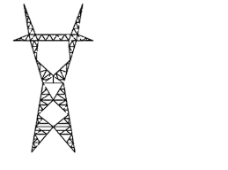


**Economic Dispatch**

(1 hour)



**ON**



**OFF**



**Threshold**

Figure 17 Hierarchal Control for Micro-Grid

### Description of the Micro-Grid Cost Function

The basic cost function to be minimized is defined by (2)and subject to energy balance constraints of the form of (3) or (4) & (5), and range constraints (6) and (7).

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

In (2) there are *N* time steps, *k* = 1, 2, 3,…, and *G* dispatchable generators whose cost, *F(Pi)*, is a function of their power output, *Pi*. Connection to an external electric grid, represented by *Pgrid*, is assigned a time dependent price for either purchasing or selling power, *F(Pgrid).* The specifics of the cost functions for each generator determine the type of problem, i.e. convex or non-convex. Generally this is a mixed-integer problem with 2N·G states for the generators to be on or off at each time step. The number of on/off decision variables quickly increases beyond what is practical to solve. The EAGERS approach develops a convex problem that can be solved with standard interior-point techniques. The minimization of (2) is constrained by a hard constraint (energy balance) for each demand category, at every time step, *k* = 1,2,3….

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

The load, ***Lk***, and any uncontrollable power generation, ***(Punctl)k***, such as solar PV is placed on the right side of the equation to correspond with the quadratic programing form *Aeq·x=beq*. Each energy demand category, e.g. AC power, DC power, heating, cooling, or steam production, has a separate energy balance. The hard constraint can be made ‘soft’ in one direction, e.g. excess heat production can be vented, with the inclusion of an additional term, ***Ploss***, and a single constraint. Eq (3) thus becomes:

|  |  |  |
| --- | --- | --- |
|  |  | (4) |
|  |  | (5) |

Each dispatchable generator is limited in its operation to some specific range. The range may be discontinuous, as some generators may be off, ***(Pi)k = 0***, or on within the specified bounds, ***Pimin*** and ***Pimax***. The grid power may be limited by the supplied power or in the power that can be returned.

|  |  |  |
| --- | --- | --- |
|  |  | (6) |
|  |  | (7) |

### Individual Equipment Cost Functions

The cost function of each generator, *F(Pi)*, can take a number of forms depending upon the solution methodology selected. Generally speaking, the cost functions for most systems, e.g. generators, chillers, and cooling towers is non-linear and possibly non-convex. Non-convex cost functions imply multiple local minima, which introduces instability in a receding horizon control problem because the solution may rapidly oscillate between local minima with only slightly changing initial conditions and costs.

The EAGERS approach solves a least-squares problem to fit a convex piecewise quadratic polynomial to a set of measured data. These piecewise quadratics are well suited for interior-point search methods. It is common practice in optimization approaches to estimate convex functions with a series of linear segments. However, when using an interior point method it is generally faster and more accurate to use a few quadratic segments than a multitude of linear segments. The quadratic costs have the benefit of smoother transitions in the solution, i.e. a generator does not alternate from one end of a segment to the other due to a $0.000001/kWh change in the fuel cost.

**Fit A** represents the best possible piecewise convex quadratic that avoids the lower bound discontinuity and has zero cost at zero output. **Fit B** includes the discontinuity and has a non-zero initial cost.

Generator Output (kW)

Fit A

Fit B

UB

LB

B)

Discontinuity

η

$

kWh

LB

UB

Generator Output (kW)

A)

$/hr

Figure 18 Conceptual depiction of generator performance and cost functions.

A) Typical electric generator efficiency (η), specific cost of generation ($/kWh), and non-linear operating cost curve ($/hr).

B) Piecewise convex quadratic cost functions.

Fit A and Fit B are determined by a least squares problem, eq (8). The objective of the least squares problem is to find the curve fitting coefficients, ***aj*** and ***bj***, that minimize the sum of errors between the fit cost and the recorded data in set ***D***. The data used to calibrate the individual generator cost functions, i.e. ***F(Pi)*** for generator ***i*** at power ***Pi***, consists of individual data points, ***d***, for power output, ***(Pi)d***, and fuel input, ***Yd***. The piecewise quadratic cost function is described by equations (9) and (10). There are *m* segments, ***Sj***, each with a corresponding length, ***Smaxj***. The output, ***(Pi)d***, is divided into these segments in order, such that ***S1*** = ***Smax1*** before ***S2*** has a value greater than zero. Equation (11) converts the optimization of (8) into the quadratic programing formulation, *min C = x’Hx + f’x*. Constraint equation (12) ensures a convex fit, while constraint equation (13) ensures a tight fit at maximum output, i.e. ***Fitmax =*** ***Ymax***, where ***Ymax*** corresponds to the cost at ***max(Xd)***.

|  |  |  |
| --- | --- | --- |
|  |  | (8) |
| 1. **FitA:** |  | (9) |
|  |  | (10) |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | | (11) |
|  | |  | (12) |
|  | |  | (13) |

**Fit B** adds an additional constant, ***a0***, to represent the y-intercept by pre-appending a row and column to the matrix in eqn (11), as shown in eq (15).

|  |  |  |  |
| --- | --- | --- | --- |
| 1. **FitB:** | |  | (14) |
|  |  | | (15) |

A separate optimization can determine the optimal segment lengths. Most generators have peak operational efficiencies at or near rated capacity, in which case a linear approximation is equivalent to **Fit A**. However, chillers, fuel cells and other distributed energy systems operate more efficiently at part load. In these instances a piecewise quadratic cost drives the solution towards these optimal operating conditions, where a linear fit would not. **Fit B** provides a significant improvement in the accuracy of the fit, particularly at part load. However, using **Fit B** requires a binary variable to avoid incurring a constant cost even when the generator is off.

### Ramping Constraints

Some generators response is limited between successive time intervals. It is straightforward to implement an additional constraint (16) through inequalities that applies to all types of generators/chillers/heaters.

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

### Start-up costs:

Ideally, the optimization should avoid wear on the equipment associated with re-starts as well as the non-steady-state performance during start-up or shut-down by limiting the re-starts. The cost associated with re-starts cannot be evenly distributed as an operating cost. Several approaches are available to limit the re-start occurrences for generators.

**The ‘Locked’ approach**: This approach mimics the standard operator procedure of only turning systems on/off if the operator knows it will operate for XX hours. At each step where there is a change in state, i.e. *Bi,k-1* ≠ *Bi,k*, it checks that the state (on/off) at all of the immediately previous steps is the same. This approach introduces *2·N·G* additional constraints of the form (17) and (18). Special attention needs to be taken with the initial state. If the initial state is ‘on’, then constraint (17) needs to be modified depending upon how long the generator has been ‘on’ prior to the start of the optimization. Similarly for constraint (18) if the initial state is ‘off’.

|  |  |  |
| --- | --- | --- |
|  |  | (17) |
|  |  | (18) |

The primary issue with this approach is the determination of the duration, XX, over which the generator should be locked on or off. The optimization is solving for cost, but this enforces a cost using constraints.

**The ‘window’ approach**: In some problems, e.g. chillers with cold-water storage, it sometimes possible to impose no-start or no-shutdown windows. In a no-start window the constraint (19) applies and prevents the system from starting during peak hours for example.

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

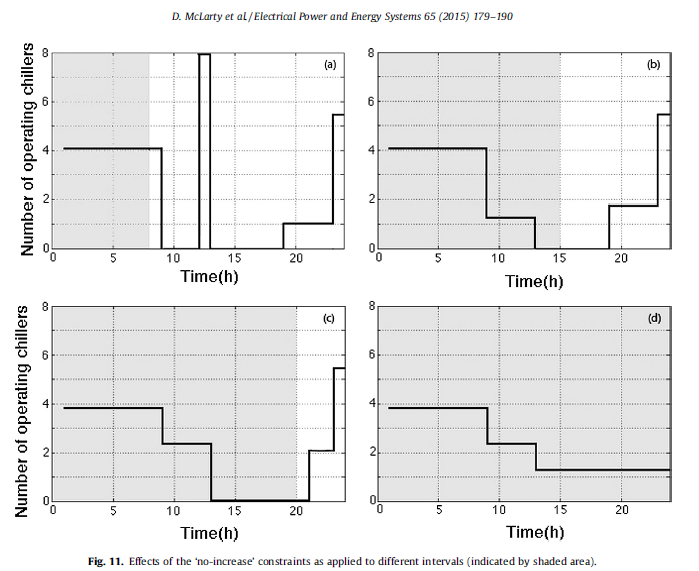
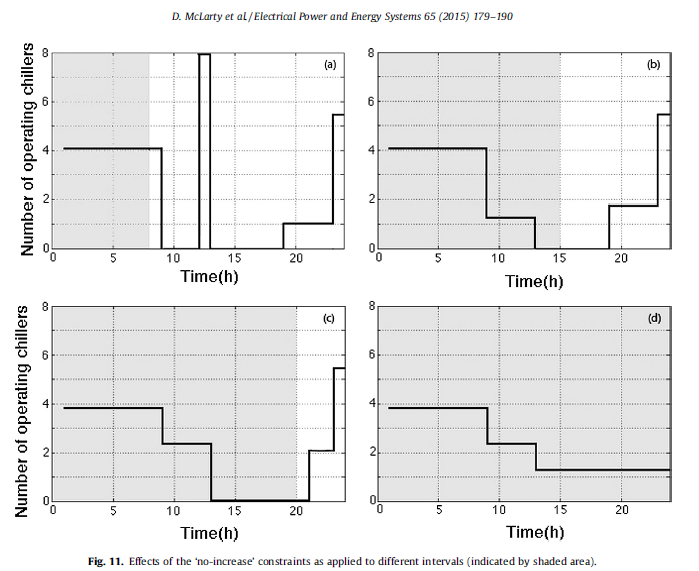


Figure 19 Evolving chiller dispatch with moving 'window' approach

**The single start/stop approach**: Some systems should not start any more frequently than the horizon over which the optimization is computed. The constraint (20), in combination with (21), constrains the binary variables to be a function of a continuous decision variable, *Tstart*. This can effectively eliminate the mixed integer problem, but can introduce ‘flat’ sections the cost function. If the cost function cannot distinguish between *Tstart* being 12.0 and 12.1, it can cause early termination in some solver techniques.

|  |  |  |
| --- | --- | --- |
|  |  | (20) |
|  |  | (21) |

**EAGERS approach:** If the cost of each feasible case is tracked during the mixed-integer problem, a secondary problem can quickly filter out segments when a system is operated for a short period, or turned off for a short period. This approach is discussed in greater detail in a later section. The approach has the benefit of allowing generators to operate for a short period when it saves a lot of money, perhaps by avoiding a peak demand charge, but still encourages a minimal number of re-starts.

### Incorporating Energy Storage

**State-of-Charge (SOC) and Power**: The cost function (22) is modified by adding a value, *F(SOCr)N*, representing the value of useful stored energy remaining in the battery or thermal storage device at the end of the optimization horizon. The energy balance (23), gains a term representing the power withdrawn from or supplied to the energy storage device, *(Pr – φr)*. This includes a penalty term for the charging/discharging inefficiency, which can be neglected when considering an ideal storage device.

|  |  |  |
| --- | --- | --- |
|  |  | (22) |
|  |  | (23) |

The useful power supplied to or extracted from the energy storage device, *Pr*, is related to the change in stored energy by equation (24). The additional energy put into charging the storage device is captured by φr in equation (25). The indirect cost of producing additional energy to satisfy the energy balance (23) ensures this charging loss is equal to, not greater than, the actual round-trip energy losses. The energy storage charging and discharging efficiencies, represented by *ηc* and *ηd*, are constant.

|  |  |  |
| --- | --- | --- |
|  |  | (24) |
|  |  | (25) |

The capacity constraint (26) and the charging/discharging constraint (27) bound the dispatch of the storage system.

|  |  |  |
| --- | --- | --- |
|  |  | (26) |
|  |  | (27) |

If the final state of the storage system is not constrained or valued the optimal solution will always deplete the storage, leading to less than optimal solutions in a receding horizon control problem. The methods for ensuring charge remains at the end of the optimization are:

1. Apply a hard constraint that the SOC at *Thorizon* is equal to a fixed value, e.g. 50%.
2. Apply a hard constraint that the SOC at *Thorizon* is equal to the SOC at *T0*.
3. Apply a final state value, as shown in (22), with the value calculated according to (28), (29), and (30)

Power put into or coming from energy storage devices, *S*, is assigned cost only at the time it was generated, *Pi*, or purchased, *Pgrid*. Any residual state-of-charge, *SOC*, must lower the net cost; otherwise the *SOC* would always be driven to zero at the end of the dispatch horizon. The function describing the value of this residual charge is a convex quadratic such that the first kWh of storage is valued slightly more, *1+δ*, than the highest marginal cost dispatchable generation, and the last kWh of storage is valued less than, *1/(1+δ)*, the smallest marginal cost of generation. The discharge efficiency, *ηd*, is included because only the energy that can be extracted has value.

|  |  |  |
| --- | --- | --- |
|  |  | (28) |
|  |  | (29) |
|  |  | (30) |

**Self-discharging losses**: Most storage devices are unable to store energy indefinitely, and thus some amount of self-discharge applies. Self-discharge can be either constant, *κ*, or proportional to the state of charge, *κ\*.* A constant discharge rate, e.g. 1kWh per hour or 1kW can be pre-calculated and added to the demand, *L*, in the energy balance. The value *κ\** in (31)is equal to the fraction of storage lost per hour from full charge. For example if a fully charged battery would lose 20% charge in the first hour *κ\** = 0.2

|  |  |  |
| --- | --- | --- |
|  |  | (31) |

**SOC Buffers**: Because of uncertainty in the forecasting it is important, particularly in an islanded microgrid, to avoid fully charging or fully discharging the energy storage. The buffer is proportional, Π, to the maximum capacity of the storage. Two pseudo-states, *l* and *u*, are given quadratic costs, the severity of which determines the relative ‘softness’ of the boundary. The soft constraints (32) and (33) can be in addition to the hard capacity constraint (26). Soft constraints act as buffers in the receding horizon control by placing a thumb on the scale in the cost function (34) as the storage approaches full or empty.

|  |  |  |
| --- | --- | --- |
|  |  | (32) |
|  |  | (33) |
|  |  | (34) |

### Spinning Reserve and Ancillary Services

Spinning reserves ensure that the energy dispatch is robust to uncertainty in the forecast. The spinning reserve target, *SRtarget*, plus any spinning reserve sold on the ancillary market, *SRancillary*, must be less than the total spinning reserve of all generators in the network at each time step. Constraint (35) imposes this as a hard constraint while (36) demonstrates how a ‘soft’ constraint can be applied in this situation. The ‘soft’ constraint can maintain a higher spinning reserve at most time steps, but allow for short periods of low spinning reserves when it is cost prohibitive to provide additional reserves. Using both constraints in conjunction can maintain higher reserves while setting an absolute floor on the tolerable reserves. The soft constraint calculates the spinning reserve shortfall, *εSR*. which is given a cost in the cost function (37). The inequality is important in order not to penalize excess spinning reserve, but only a shortage of spinning reserve. Excess spinning reserve is not penalized by making *εSR* strictly non-negative.

|  |  |  |
| --- | --- | --- |
|  |  | (35) |
|  |  | (36) |
|  |  | (37) |

The spinning reserve of each generator is constrained by the ramping (38) and capacity constraints (39) of the device.

|  |  |  |
| --- | --- | --- |
|  |  | (38) |
|  |  | (39) |

Similarly energy storage systems provide spinning reserve limited by the peak discharge rate (40) and the remaining stored energy, (41).

|  |  |  |
| --- | --- | --- |
|  |  | (40) |
|  |  | (41) |

### Modeling Buildings

Standard 1st order building RC models take the form (42), where the temperature increase depends upon the heat entering the zone through conduction, *(TOA – Tk-1)/R*, the internal heat added, *Q̇intGain*, and the heat added/removed by the HVAC system, *ṁCP·(Ts-Tk-1)*. The net energy imbalance is multiplied by time, *Δt*, and divided by capacitance, *C*, to get the change in temperature. The outside air temperature, *TOA*, and the HVAC supply temperature, Ts, may vary with each step, *k*.

|  |  |  |
| --- | --- | --- |
|  |  | (42) |
|  |  | (43) |

The intermediate variable *T\** is introduced because depending upon the step size and response time of the building it may be better to estimate the heat gain through conduction and the cooling with *Tk* or *(Tk-1+Tk)/2* or *Tk-1*. If adjusting a building from 69°F to 70°F happens quickly, then the energy balance used for the next hour should use *Tk*. If the dynamics are slow relative to the step size, it is more economical computationally to use *Tk-1*.

The primary issue with the standard 1st order RC model is the non-linear term *ṁ·Ts*. Since generally we wish to control both the flow and supply temperature, this non-linearity causes a problem. On approach is to assume a fixed temperature difference between the supply air and room temperature, and add a separate term for the heating as in (44). In this arrangement *ṁ* controls the cooling, while *Q̇reheat* independently controls the heating.

|  |  |  |
| --- | --- | --- |
|  |  | (44) |

The 1st order response model can be divided into two zones, the air zone *TZ* in (45), and the wall zone, *TW* in (46). Each heat transfer boundary has a different resistance, though they can be the same. The addition of a second temperature state incorporates some of the temporal relationship between outdoor air temperature and the heat gain through conduction to the air zone, e.g. the zone gains more heat through conduction at 3PM when it is 90°F outside, than at 11AM when it is also 90°F outside because it was hot all day.

|  |  |  |
| --- | --- | --- |
|  |  | (45) |
|  |  | (46) |

The EAGERS approach re-arranges the energy balance of (42) to solve for the HVAC power, as shown in eq (47).

|  |  |  |
| --- | --- | --- |
|  |  | (47) |

Next, the energy balance is split into two segments: the energy needed to reach a static target, *Tset*, i.e. 70°F, and the additional energy required to reach a dynamic target, the optimization decision variable, *Tk*. Equation (48) has now identified a constant term, *Q̇HVAC0* which can be pre-solved outside the optimization. This term can thus include heuristics and non-linearities specific to the building and HVAC system. The remaining term *ΔQ̇HVAC* represents a local linearization for deviations in temperature from the nominal temperature profile, *Tset*.

|  |  |  |
| --- | --- | --- |
|  |  | (48) |

Assuming the internal air zone temperature dynamics are sufficiently fast relative to the 1-hr time step justifies using *Tk* in the heat transfer energy balance in place of *T\**. Within the small range of variation, e.g. 2-4 degrees, this assumption is valid. Replacing *C* with *Cap* and *1/R* with *UA* arrives at the formulation (49).

|  |  |  |
| --- | --- | --- |
|  |  | (49) |

This equality form is still limited to a one-for one trade-off of heating for cooling. EnergyPlus uses a schedule to set the temperature boundaries, and controls the HVAC system such that the building tries not to exceed those thresholds. This boundary control is fundamentally different from tracking a single thermostat set point, ie. 70°F. There are strong non-linear relationships within the multi-zone building energy balance and the heuristics which determine air mass flow, HVAC supply temperature, and damper or economizer position. The next steps allow simultaneous non-zero heating and cooling, or a deadband range in which there is not heating or cooling. The first step splits the energy balance into a heating equality, eq (50), and a cooling equality, eq (51), so that separate variables represent the heating and cooling. This allows them to be inserted into the correct system energy balance.

|  |  |  |
| --- | --- | --- |
|  |  | (50) |
|  |  | (51) |

These equalities include a constant term, e.g. H0 and C0, and a linear inequality term described in (52) and (53). The new terms, and represent the temperature at which heating and cooling begin to linearly increase. Without considering fresh air entering the building, the transition from heating to cooling will occur at a single temperature setpoint. The constant values, H0 and C0, typically represent the heating and cooling treatment of the minimum fresh air introduced to the building. Changing the damper position can partially cool the building on certain days, creating the deadband shown in Figure 20.

|  |  |  |
| --- | --- | --- |
|  |  | (52) |
|  |  | (53) |

Since the temperature state *Tk* is bound, it is practical to limit the range of these new terms as well using (54) and (55).

|  |  |  |
| --- | --- | --- |
|  |  | (54) |
|  |  | (55) |

Figure 20 describes the equilibrium heating requirements to achieve a temperature set point. Equilibrium heating implies the building is at steady state, and thus the heating is balancing the heat loss to the environment. On a hot day this transition point may be well above the maximum allowable temperature, *Tb,kmax*. Thus the heating will remain H0 throughout the range of the optimization variable *Tk*. H0 is likely the re-heat necessary for dehumidification. On a cold day, there will always be heating so is set equal to the lower limit, and H0 is the equilibrium heating at *Tb,kmin.* For days in which the heating starts within the range of the optimization variable, the inequality allows it to remain at H0 until additional heating is required.

*H0*

*C0*

Figure 20 Diagram on non-linear equilibrium heating

The approach allows for heuristic decision processes for HVAC supply temperature, air mass flow, and damper/economizer positioning when calculating H0 and C0, prior to the optimization.

Note that this approach is readily extendable to include wall temperature states in a similar fashion to equation (46). However, due to the significantly slower transients of the wall temperature states, e.g. 6 to 12 hours, it is more practical to contain the wall temperature state calculations within the pre-solution for , and . Thus the secondary heat transfer equations are still applied, but are not included as additional states within the optimization, reducing overall complexity.

With the heating and cooling loads defined, they can be placed in their respective energy balances. In (56) the cooling load of each building, *Cb*, is subtracted from the cooling power output of each chiller, *Ci*, and the cooling power withdrawn from the reserves of any cold energy thermal storage, *Cr*.

|  |  |  |
| --- | --- | --- |
|  |  | (56) |

The cooling load met by electric chillers appears in the electric energy balance. A piecewise convex fit (57) of the chiller efficiency, *β*, determines the electricity consumption per unit of cooling.

|  |  |  |
| --- | --- | --- |
|  |  | (57) |

There may be an additional electric loads associated with the HVAC cooling system, such as the fans needed to move the air around. If these additional loads are assumed linearly proportional to the cooling state, they can be added to the electric energy balance (58) as *γCb*.

|  |  |  |
| --- | --- | --- |
|  |  | (58) |

The heating energy balance (59) subtracts the heat load of each building, *Hb*, from the heat output of all CHP generators, heaters, *Hi*, and heat withdrawn from the reserves of any hot thermal storage, *Hr*. If heat can be arbitrarily rejected to the atmosphere, e.g. from excess CHP heat recovery, a *Hloss* term is included to balance the heat equation. The equality constraint (50) can be combined with the total energy balance equality as shown in (59)The heat output from CHP generators depends upon the effectiveness, *ε*, of the heat recovery system and the waste heat produced at the electric power output setting, *Pi*.

|  |  |  |
| --- | --- | --- |
|  |  | (59) |

Instead of placing hard boundaries on the building zone temperature ‘soft’ constraints, (60) and (61), are applied. Both *Texcess,k* and *Tshortfall,k* are given a cost in the cost function (62). *Tideal* and *ΔTcomfort* are fixed values, though *ΔTcomfort* can increase during off-peak hours.

|  |  |  |
| --- | --- | --- |
|  |  | (60) |
|  |  | (61) |
|  |  | (62) |

### Cooling Towers

Each cooling tower fan has a piecewise convex linear fit (63) to approximate the heat rejection to the atmosphere for a given power input. The coefficients *αj* can be a function of the ambient temperature and humidity, and updated with the weather forecast of each step.

|  |  |  |
| --- | --- | --- |
|  |  | (63) |

The performance of cooling towers and chillers have a strong dependence on the temperature of the cooling tower return water temperature, *WTk*. Lowering this temperature set point increases the electric power consumed by the fans to reject the heat, *E(HRf)*, as shown in (64**)**. Conversely, lowering the water temperature decreases the power consumed by the chillers, *E(Ci)* as shown in (65). Chillers typically perform better with lower return water temperature, thus an opportunity for optimization arises on cool or dry weather days.

|  |  |  |
| --- | --- | --- |
|  |  | (64**)** |
|  |  | (65) |

The cooling tower return loop water temperature is modeled as the 1st order response to an energy balance between the chillers and the cooling tower fans heat rejection, (66). The heat that must be rejected by the chillers is equal to their cooling output plus the electric power they consume to account for the heat added to the refrigerant by the compressors.

|  |  |  |
| --- | --- | --- |
|  |  | (66) |

The electric energy balance can now also include the loads from the cooling tower fans (67).

|  |  |  |
| --- | --- | --- |
|  |  | (67) |

### Networked Distributed Energy Resources

When systems of distributed resources are connected on an electric, district heating, or district cooling network it imposes additional constraints and losses that are not captured in the optimization described thus far. The losses may be fixed or proportional to the amount of energy transfer. The fixed losses are readily added to the load term, *L*, at the appropriate node. Nodes with ideal (no proportional loss) energy transfer can be aggregated even if they are spatially separated. It is possible for the network map to be different for electric, heating, and cooling. For example on a microgrid there may be no constraining transport limits for electricity, and the losses may be negligible, thus the electric network is modeled as a single node. The district heating network may be constrained by the size of the water pipes and have significant heat loss.

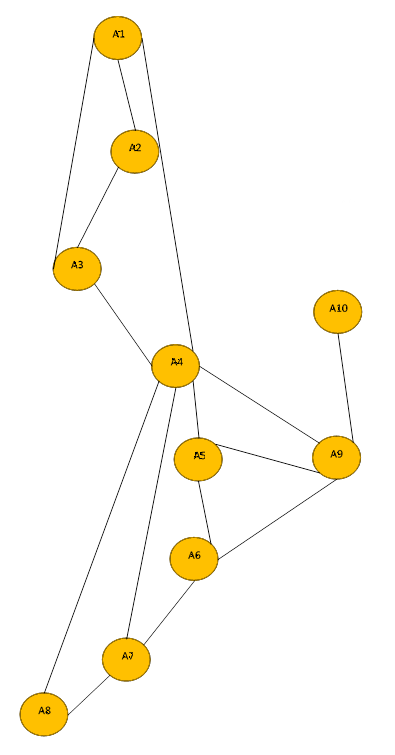
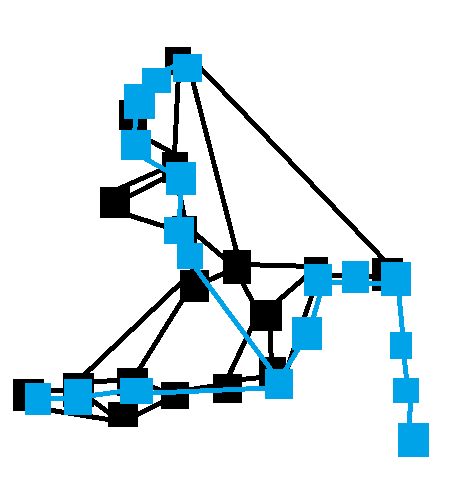
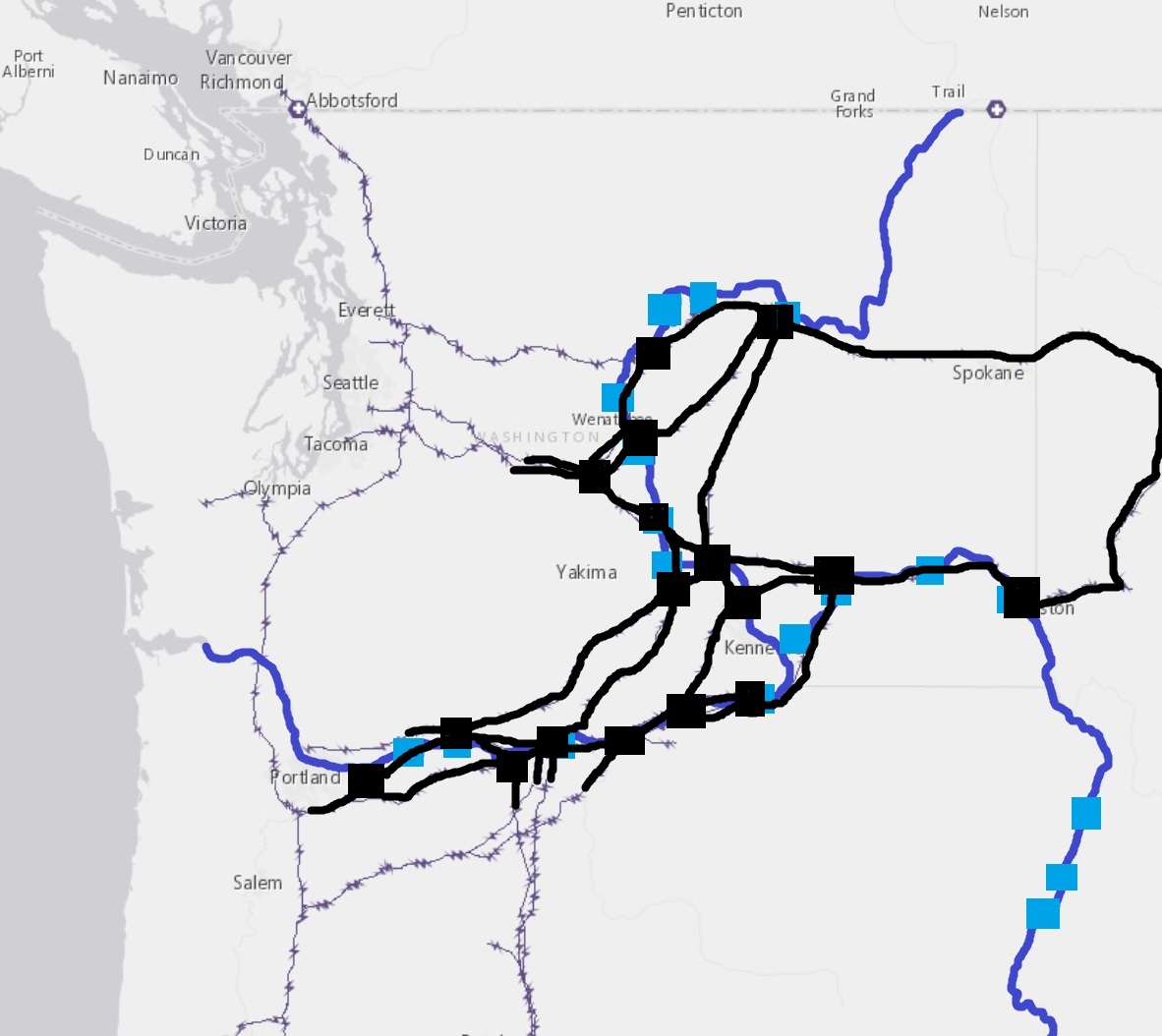


Figure 21 a) Overlay of a 17-node electric network (black) with an 18 dam river network (blue), b) a simplified visual representation of the network, c) reduced order 10-node network

For an energy network the energy must be balance at each node, *j*, in the network. The balance at each node (62) now includes only a subset *Gj* of the generators and *Sj* of the storage systems that are co-located at that node. The demand, *Lj*, the uncontrollable power generation, i.e. wind and solar, and the dissipated excess energy, *Ploss*, are also unique to each node. Each node may have multiple connections, or lines, coming into or out of it, represented by the subset of all transmission lines in the network, *Tj*.

|  |  |  |
| --- | --- | --- |
|  |  | (68) |

In creating a network it is important to distinguish between lines that are capable of bi-directional energy transfer, vs. single direction energy transfer. Systems with single direction energy transfer can be described by (62) where *Pl* is the power transferred on the line, and the multiplier ξ equals -1 in the energy balance of the upstream node, and equal to the transfer efficiency, *η🡪*, at the downstream node. Similar to an electric network model, a district heating and district cooling energy balance can be defined for each node, as described in (63) and (64).

|  |  |  |
| --- | --- | --- |
|  |  | (69) |

|  |  |  |
| --- | --- | --- |
|  |  | (70) |

Transmission losses proportional to the power transferred on a bi-directional power line are included in (65) and calculated via (66) and (67). Separate strictly non-negative penalties, represented by *σ🡨* and *σ🡪*, balance energy at the nominally upstream and nominally downstream nodes. Similar to the charging loss term, the indirect cost of additional energy generation ensures the equality holds when the line transfers power in that direction.

|  |  |  |
| --- | --- | --- |
|  |  | (71) |
|  |  | (72) |
|  |  | (73) |

### Hydroelectric generation, water reservoirs and river systems:

The river network stock and flow problem is directly analogous to the electric/heating/cooling networks described previously, with a few key differences.

1. Mass must be conserved. There may be some losses to infiltration or evaporation, but generally, the outflow of a reservoir will travel downstream to the next reservoir.
2. The transport of the mass from one reservoir to the next takes some finite amount of time. Thus the mass balance connects across time steps in a way the electric energy balance does not.
3. The source terms, i.e. precipitation and base flow, cannot be controlled.

Each reservoir or irrigation district acts as a ‘node’ in the water network. Each node of a water network will have a mass balance. Precipitation, snow melt and base flow act as ‘generators’, irrigation districts act as ‘loads’, and river segments act as unidirectional transfer ‘lines’. A key difference in connecting the nodes is the rate of mass transfer along a river segment. While electric power transfer occurs instantaneously, it may take several hours for water to transit a river section to the next dam. A hydroelectric power plant lies at the intersection of an electric network and a water network. The hydroelectric dams appear in both the mass and energy balances of the node. Each hydroelectric dam can be modeled with 3 states at each time step: the SOC, the downstream flow, and the spill flow.



Figure 22 Conceptual model of hydroelectric dam/water reservoir

The mass balance for each node *j* = 1, 2, 3, …, and every time step, *k* = 1,2,3…, is shown in (20). Where the transient flow time from the upstream reservoir, *T*, relates the outflows of the upstream reservoirs *Qi*, in 1000 cfs, to the change in storage, *SOC*, in 1000 acre-ft. Since an acre-ft is 43559.9 ft3, and there are 3600 seconds in an hour, the volume per hour is 12.1/Δt. The net source/sinks between reservoirs resulting from side-streams, evaporation, and infiltration are lumped into a single term, *φ*.

|  |  |  |
| --- | --- | --- |
|  |  | (74) |

*The Energy Balance* specified by (5) requires the power produced at each dam, in kW, as a function of the states. Most dams have a spillway option so that the downstream flow is not always equal to the water flow through the turbines, (21). The power produced, shown in equation (22), is directly proportional to the flow through the turbines, *Qdownriver* – *Qspill*, the reservoir height, *Hd*, efficiency of the turbines and generators, ηd, and a conversion factor, 84.674 kJ/1000ft3·ft. The values of gravity and water density are considered within the conversion factor.

|  |  |  |
| --- | --- | --- |
|  |  | (75) |
|  |  | (76) |

*Constraints* are imposed on the optimization of (1) through equalities, inequalities, and cost terms. Each dam in the water network is constrained by a) an instream flow requirement, b) a maximum generator flow, c) maximum increase in generator power, and d) maximum decrease in generator power.

|  |  |  |
| --- | --- | --- |
|  |  | (77) |
|  |  | (78) |
|  |  | (79) |
|  |  | (80) |

The complexity of any network can be reduced when ‘ideal’ transport can be assumed between adjacent nodes. If the transmission line connecting two adjacent electric nodes has more than sufficient capacity for energy transfer in either direction, and the energy losses of transmission are negligible, it is then possible to aggregate the two nodes into a single node. It is not possible to aggregate the river ‘nodes’ in the same way. In this example it was assumed that the transfer lines between each hydroelectric dam and the nearest electric node were ‘ideal’, as were some shorter line segments. The resulting 10-node electric network is shown in figure \_\_. The remaining, non-aggregated, transmission lines each have a transfer efficiency <1. The transmission loss terms of (13) and (14) are used to subtract this lost energy depending upon which direction the energy is flowing.

## Reduced Mixed-integer Problem

### Non-mixed integer implementation:

Solving the optimization (1) with **Fit A** results in a close approximation of the true optimal operation. It is likely only one generator is dispatched in the linear region of **Fit A**, since the slope of each generators linear segment is unique. At times, the ramping constraints may force two or more generators into this region.

The part-loaded generator/s may be operating in the discontinuity between off and the lower bound. Either the part loaded generator/s must shut down and allow other systems to pick up the slack, or the part loaded generator/s stay on with other systems operating at part-load to accommodate the extra capacity. A quick feasibility check, and consideration of start-up costs can help make this determination at each time step. With the resulting on/off schedule of generator operation known, the optimization (1) can be re-solved using **Fit B** to better approximate the marginal cost of each generator.

Solving (1) with **Fit A**, checking the feasibility of the part-loaded generator/s, then solving (1) with **Fit B** replaces the mixed-integer optimization with two straightforward quadratic optimizations. This approach is valid for most simple arrangements of generators and storage devices. Arrangements that are more complex may still require solving a portion of the mixed-integer problem

### Standard mixed-integer approach:

If the cost function of a single generator includes a discontinuity, i.e. zero cost when off, and some finite non-zero cost when operating, the problem becomes a mixed integer problem. If zero output corresponds to an idling mode that still incurs a constant fuel cost, and this constant cost is roughly equal to the y-intercept of the fit cost function, then mixed-integer problem is not necessary. Solving the mixed integer problem with the given formulation requires binary variables, *Bi,k*, implemented in the cost function (68) and either the constraint function (69), or the energy balance equality constraint (70).

|  |  |  |
| --- | --- | --- |
|  |  | (81) |
|  |  | (82) |
|  |  | (83) |

### EAGERS simplified mixed-integer approach:

This approach combines the two convex quadratic programming problems of the non-mixed integer problem with an intermediate mixed-Integer sub-optimization and some logical conditioning**.** The error between **Fit A** and the actual cost at part-load, varies by generator. The start-up and re-start costs are not captured by the cost function, and vastly varying equipment sizes may mean that accommodating a large generator may mean turning off one or more smaller units.

The sub-optimization problem formulates a separate optimization for each time step, (71). The optimization problem formulation is very similar as the multi-step optimization with a similar energy balance constraint (72), and constraints for individual generator operating ranges. The single step optimization omits the ramping constraints of each generator, and modifies each energy storage system according to (72), (73), (74) and (75).

When optimizing a single time step using the cost function of (71) the energy storage lacks the ‘big-picture’ perspective of the simultaneous optimization. EAGERS approach is to apply a quadratic cost to deviations from this initially planned power output determined by the first optimization, *Pr0*. The quadratic cost constrains the storage profile to be similar to the first optimization, but allows for deviations when significant savings can accrue. The energy balance at each step thus becomes (72). The planned power output from the storage is placed on the right-hand-side as a constant. The deviation from this power, and the charging penalty remain on the left hand side.

|  |  |  |
| --- | --- | --- |
|  |  | (84) |
|  |  | (85) |

The planned power output, *Pr0*, can be calculated based on the SOC states from the first optimization, (73), including any proportional or fixed losses, κ\* or κ. The allowable range of the deviation is thus the nominal power output range of the storage device shifted by *Pr0* as per (74). The power output might be further constrained by the available stored energy or remaining storage capacity if that is more restrictive given the current SOC.

|  |  |  |
| --- | --- | --- |
|  |  | (86) |
|  |  | (87) |

The charging penalty must similarly be offset by the planned power output, as per (75).

|  |  |  |
| --- | --- | --- |
|  |  | (88) |

At each step there exist **2G** combinations of generators, not all of which are feasible. The binary on/off criteria is organized into *Bi,k*. Equation (76) is used to determine if a particular set, A, of generators (A⊆B) is feasible at that step. The test of (76) is applied to each energy balance that is part of the optimization.

|  |  |  |
| --- | --- | --- |
|  |  | (89) |

After eliminating infeasible options. All remaining combinations are tested, and the cost of each scenario is recorded. The combinatorial problem is possibly of the order *N·2G*, but significant reductions can be made by predetermining infeasible combinations and partially separating the cooling optimization from the electric optimization when there is sufficient storage available.

After testing the feasible options at each step, the lowest cost option for each step is selected and an initial binary on/off decision matrix is formed, *Bi,k*. The storage deviation from the lowest cost option is used to update the SOC profile using (77) and (23).

|  |  |  |
| --- | --- | --- |
|  |  | (90) |

This on/off matrix may not be a stable generator dispatch. The steps described in Figure 23 outline the overall procedure for systematically testing the feasible subset of optimizations, recording the configuration, cost, and dispatch for each tested scenario, and then applying additional heuristics to improve the binary on/off decision matrix for the entire set of generators.

Initial Energy Dispatch using Fit A

Calculate Marginal Costs of Power in each category

Are there two or more chillers?

Check start-up costs vs. marginal cost of alternate options at each step

Determine feasible combinations

Repeat for each time step ***k***

Test feasible options

Convert state of charge to power

Update energy storage state for k+1

Determine feasible combinations

Repeat for each time step ***k***

Test feasible options

Convert state of charge to power

Update energy storage state for k+1

Final Energy Dispatch using Fit B

Check start-up costs vs. marginal cost of alternate options at each step

Check ramp rate limitations

Best chillers for step ***k***

Yes

Figure 23 Mixed-integer sub-optimization outline

The first heuristic function, depicted in Figure 24, evaluates the cost differential between each feasible alternative and compares the marginal cost to any start-up costs avoided by changing the binary on/off schedule.

Initial Dispatch On/Off Schedule, *Bi,k*

Determine Shortest Segment

Is there a feasible alternate option with lower start-up cost?

Update Dispatch Power and Recalculate *SOCr,k*

Ignore this segment

Generator stops, then starts

Yes

Update Dispatch On/Off Schedule, *Bi,k*

Would it be feasible & cheaper to idle or keep running this generator?

Yes

No

Can the segment simply be avoided?

Is there sufficient storage capacity to shift the energy to when the generator previously shut down?

No

No

No

Yes

Yes

Ignore this segment

Figure 24 Method for avoiding costly re-starts

The second heuristic function determines if adjustments need to be made to accommodate output constraints due to ramp rate limitations. This can imply turning on generators earlier, leaving them on longer, or changing the schedule of a different generator to accommodate the shortfall.

## Component Model Descriptions

### Utilities: Electric, Gas, District Heating or Cooling

The options for specifying an electric utility include: i) Summer and winter rate tables, ii) On-peak, mid-peak, and off-peak energy costs and demand charges, iii) sell back options, and iv) a minimum import threshold.

Two 7x24 tables describe the hourly rate for Sunday – Saturday where 1 represents off-peak, 2 represents mid-peak and 3 represents on-peak. The first table is for winter schedules and the other for summer schedules. The user can specify when the summer/winter seasons start and end. The off-peak, mid-peak, and on-peak energy rates are defined separately for each season as well. The demand charges are calculated based on the peak hourly use during the month. Separate demand charges can be specified for each rate period.

The Grid Sell-Back refers to the ability of the utility to sell back to the grid, and for what rate. The user can either select none, a percent of the tariffs, or a reversed meter for the grid sell-back. The Minimum Import Threshold refers to the minimum the utility can buy at the purchase rate. This number can be negative if selling back at the same purchase rate is possible, in which case it represents the maximum sell back.

Gas utilities are specified as either a constant cost ($/mmbtu) or a variable cost which the user must provide as a vector of costs of the same length as the building data.

### Generators, Chillers, Boilers, and CHP systems

These types of prime-mover devices are specified via: i) a nominal capacity, ii) a turn-down ratio, iii) an efficiency curve, and iv) a response rate. The nominal capacity may be in terms of kW or Tons for chillers. The turn down ratio is the maximum allowable power divided by the minimum power. The efficiency curve is specified in terms of normalized capacity. A system may have more than 1 output, i.e. CHP generator, in which case the efficiency table has an additional column for each output. The response rate can be specified as a linear slew rate, a 2nd order dynamic model, or a continuous time state-space model.

### Energy storage systems

Thermal storage refers to either hot or cold thermal storage, typically of water. The size of the storage system can be quantified in volume and temperature difference (water) or in kWh of storage capacity. Charging rate limits and efficiencies are also specified.

Electric storage refers to any battery within the microgrid. All batteries are specified by size, peak charge/discharge rates, charge/discharge resistances and a voltage vs. state of charge curve. Similar to the generator, the electric storage also has the option to ‘Specify Communication Ports’. The ‘Self Discharge Rate’ considers the innate loss of energy storage associated with batteries.

For every storage device, both electric and thermal, there is a value of self-discharge that must be accounted for. Self-discharge refers to the characteristic loss of stored energy over time. This is important to consider as all storage devices experience this loss in some degree. Thermal storage experiences self-discharge at a much greater rate than electric storage, so it is especially important that it is considered. In EAGERS, self-discharge is understood to be a constant that is added to the overall demand of the storage. This constant is represented by the loss (in percent) multiplied by the upper bound of the device, then divided by the charge and discharge efficiencies.

### Hydro Power

### Wind and Solar

The user has the choice of either wind or solar to be placed under the renewable category. Renewables offset the demand of the generators, as they provide energy to the plant.

When editing solar, several specifications such as location, size, angle, type and tracking must be input to best identify the contribution the resource is making to the microgrid. To the right of the solar setup interface is a DC-AC Conversion chart for all important values associated with the solar panel.

### Building

## Real-time Model Predictive Control

The purpose of the cQP method is to develop a fast, deterministic solution to the scheduling problem that is stable when implemented in a receding horizon control strategy. Connecting the long-term optimization, i.e. 24-hour, to the short-term control that responds to quickly changing demands requires a smooth hand-off from the big-picture solution to the near term decision-making.

Spinning reserve plays a key role in accommodating the uncertainty in the load between successive optimizations of the long-term forecast. Sufficient spinning reserve allows a generator to turn off or on precisely as scheduled by the long-term forecast despite any load deviations that occur. The long-term forecast may schedule a generator to be off by the start of the next optimization, but does not optimize precisely when within the current period it should shut down. Nor does it specify conditions, load or generation deviations, for which the generator should not shut down, or conditions for which a different generator should be started. These omissions likely results in sub-optimal behavior in response to load deviations from the forecast used.

# Simulation Tool (STRIDES)

The simulation tool enables non-linear simulation of energy system components with their respective local controllers. Components can be readily linked into systems and larger networks of systems. Individual components, e.g. heat exchangers, batteries, inverters, fuel cells, etc., can be spatially resolved physical models, or simple reduced-order models.

## Simulation Tool Interface

There is currently no GUI interface akin to the planning and control tool interfaces. A series of menus walk you through the available options. The function *STRIDES* is a script placeholder for a future GUI that walks through the user options described previously. The primary functions it calls are *BuildModel*, *RunBlocks*, *CreateLinModel*, and *RunLinSystem* that are described in the next section.

The first options is to initialize a model. This is necessary after edits have been made to the model file or any component/controller functions that this model calls upon. Selecting this option will initialize a model to a steady-state operating condition.

After initializing, or loading a pre-initialized model, the user has the option to develop a linearized version of their model. The linear representation shown below considers ẋ as the change in states of the model, *ẏ* the outputs of the model, and *u* the control inputs. This will create a set of A, B, C, D, matrices linearized around the operating points selected by the user. This allows for linear simulations to interpolate between multiple linear models to better approximate the non-linear behavior.

|  |  |  |
| --- | --- | --- |
|  |  | (91) |
|  |  | (92) |

After the linear model has been created, or if this step were skipped, the user has the option to simulate a response with the dynamic system. This can be done for either the non-linear or linear models, or both for comparison. The user specifies the dynamic with a load profile comprised of two vectors, i) a vector of time (in seconds) corresponding to ii) a vector of % of nominal power.

## System Model Files

Each system model is a function describing the user defined specifications of each component in the system. It is possible to program user options into these model functions that change the arrangement of components, i.e. internal or external reforming fuel cell system. Each model file groups the parameters associated with each component. Due to the way it initializes, it is generally helpful for convergence if the components are listed in an order similar to the flow of gases. This is not always possible with feedback loops and recirculation, and the initialization should handle most variations.

Each component must have specified i) type, ii) name, iii) connections, iv) TagInf. The type determines what component function will be called, e.g. ‘Blower’, ‘FuelCell’. This does not have to match the name. This was done to allow multiples of each component. For example a system with two heat exchangers would both be given the type ‘HeatExchanger’, but would be given names HX1, HX2. In the model function each component must be put into the Plant.Components directory with the component name. For example all specifications for HX1 must be put into Plant.Components.HX1. This includes type, name, connections and TagInf, e.g. Plant.Components.HX1.type = ‘HeatExchanger’, Plant.Components.HX1.name = ‘HX1’.

Connections is an array of string variables (text) that specifies what other block, function, or Tag is connected to each inlet. The order must match the definition of component inlets in the component initialization function. A connection to another component is specified by the component name and outlet port name. If nothing is connected to that inlet an empty string is specified, i.e. ‘’. As an example a heat exchanger has 4 inlet ports, i) Flow1, ii) Flow2, iii) Flow1Pout, and iv) Flow2Pout. In this example Plant.Components.HX1.connections = {‘Comp.Flow’; ‘Turb.Outlet’; ‘Comb.Pin’; ‘’;}. This would connect the Flow1 port to a compressor named Comp, connect the Flow2 port to a turbine named Turb, and connect the Flow1Pout to a combustor named Comb. The Flow2Pout port is left unconnected and will remain at its initialization value, 101kPa. This could be connected to a function of ambient pressure by specifying the name of a function that connects to a model or database of ambient pressure.

TagInf is a list (cell array of strings) of parameters, e.g. RPM, that are to be recorded at each time step. The component function must be set up to put the value into the structure Tags.(block.name).RPM. The parameter name specified in TagInf must match exactly what is recorded in the component, i.e. *Plant.Components.Shaft.TagInf = {‘RPM’}*.

The system model function can include any other parameters that the user would likely use to specify its characteristics. These would be specific to the component type, and greater detail is provided in the description of each component type.

After specifying each component it is necessary to specify the controller for the system. Controllers are structured very similar to components, but they are distinguished separately to enable the linearization of the components separately from the controller. The controller output ports become *u* in the state-space linear model. This means that the controller will still need to specify type, name, connections, and *TagInf*. The only difference is that they will be put in the directory *Plant.Controls.ControllerName*. It should be possible, eventually, to connect multiple controllers to a single model, but for the time being a model should have a single controller function. The function can still be a multiple input multiple output controller, i.e. shaft speed and power, but it should be in a single m-file function.

At the bottom of the model function the user can specify any parameters they would like to see graphed as the model is simulated, and any additional parameters they would like plotted when the simulation finishes. *Plant.Scope* defines variables that are plotted as the simulation progresses, and *Plant.Plot* lists parameters to be displayed upon completion.

### Build Model

*BuildModel* takes the set of components assembled in the *Plant* variable by the model function, and initializes those components to a steady-state operating condition. It will save everything into the global variable *modelParam*. The objective is to organize the states of each component into a single vector of states, *Y*, to be used by ODE15s. The ordered states associated with each component, e.g. 13 through 34, are recorded under *modelParam.ComponentName.States*. To avoid numerical issues, nearly all variables are scaled to be ~1. Certain variables representing valve position are normalized to be between 0 and 1 and the upper/lower bound is recorded as such. The purpose is to avoid computations with variables of 10e12 magnitude along with 10e-8 magnitude as the matrix math leads to rounding errors. The scaling factor of each variable is captured in *modelParam.Scale*. After initializing the steady-state condition is saved in *modelParam.IC*.

The first section initializes each component independently of the others, then connects outlet ports to inlet ports for the second step. The second step converges component initializations to an approximation of steady-state operation. During this step, the controller initialization function plays a key role in making adjustments that converge the model. Finally the non-linear model is run to steady state by simulating 24 hours at constant demand. It does this using ODE15s which in turn calls the next function, RunBlocks.

### Run Blocks

Each ODE solver, e.g. ODE15s, solves a non-liner model by calling a function dY = function(t,Y) where Y is a vector representing the states of the model. Thus RunBlocks aggregates the output of each component model, i.e. the change in states, to arrive at a net function for the entire model.

The first portion is to determine what time-step the solver is on, or if it is re-calculating the model Jacobian. Some variable time-step solvers will occasionally reverse direction, recalculate the Jacobian, then proceed. This first portion ensures those reversed time steps are overwritten.

The second portion updates the pressure states. Because these are often back propogated through the components it is important to update these first.

The third portion runs through the list of components in the model, and converges the inlets of each block. This addresses any issues where by a inlet value from one block is passed through another block without being represented by a state. For example, the fuel cell controller may specify the fuel inlet temperature that the fuel source block outputs. There is no state representing the temperature in the source block, and this temperature must be seen immediately by the reformer or mixing block that is connected to the source. This is the reason for part of the structure of the component functions that will be discussed later.

Next the components are run individually, with their correct inlets, to determine the rate of change of each state, dY. These are aggregated into a single vector dY for the entire model. During this step it records any parameters specified in the component variable TagInf. Finally, the RunBlocks function plots any parameters specified in the Plant.Scope within the model function.

### CreateLinModel

### RunLinSystem

### Example System model Gas Turbine:

This models a recuperated turbine system. There are 8 components and 1 controller. The components arranged per Figure 25 are: a fuel source, and air source, a compressor, a heat exchanger, a combustor, a mixing volume, a turbine, and a shaft. Details on the mathematical representation of these components can be found later in this section.

**HX**

Compressor

Turbine

**Air**

Combustor

**Fuel**

Generator

Figure 25 Recuperated micro-turbine system

The controller for this system, *RecouperatedGasTurbine*, has two outputs, the power extracted by the generator and the fuel supplied to the combustor. The controller uses measurements of the turbine exhaust temperature (TET) and the RPM to control the operation of the turbine under changing load conditions. The controller tracks a desired power output by changing the fuel flow into the combustor. More fuel flow raises the temperature and increase power output. The controller also aims to hold TET constant by changing the RPM, thereby changing the mass flow of air. Changing the RPM requires extracting more or less power with the generator, thus causing temporary deviations between the desired power output and the actual power output. These controls work in tandem to control the turbine.

### Example System model SOFCsystem:

The modular system modeling approach described in this manual can be used to represent a few high temperature fuel cell configurations shown in Figure 26, Figure 28, and Figure 29. Each of the systems described make use of a similar controller for the blower power, recirculation valve position, air pre-heater bypass valve position and stack current, though the PI gains may need to be adjusted for different configurations and sizes.

Selecting the internal or direct reforming option results in the system configuration of Figure 26. The ‘direct’ reformer sends the fuel directly to the anode, where much of the reforming would occur near the fuel entrance of the fuel cell. Some designs use an indirect internal reformer to separate the steam reforming reaction from the active SOFC electrolyte. Selecting the ‘internal’ option simulates this. Figure 27 illustrates how this thermally coupled arrangement pre-reforms the fuel within the stack, but not within the anode.

Blower

Oxidizer

**Fuel**

Cathode

Anode

Figure 26 SOFC with anode recirculation and internal reforming

**H2O**

**O2, N2**



**heat**

**CH4**

**+**

**-**

**H2O,H2,CO2**

**CO, H2**

**H2**

**CO2**

Figure 27 Arrangement of "internal" fuel reformer

Selecting the ‘external’ reforming option results in the arrangement of Figure 28 where exhaust heat is recovered in an external reformer before the exhaust heat is used to pre-heat the air. There are some limitations in this design, particularly at high utilizations when there is insufficient combustion heat to supply energy to both the external reformer and the air preheater. The air flow requirements are higher, for a given stack temperature gradient, since there is less internal cooling. Thus the heat transfer requirement of the air pre-heater is higher. System efficiencies are significantly lower in this ‘external’ arrangement.

Blower

Oxidizer

**Fuel**

Cathode

Anode

Figure 28 SOFC with anode recirculation and external reforming

A slightly different arrangement uses an adiabatic reformer to pre-reform a portion of the incoming fuel. The system diagram becomes that of Figure 29. The anode recirculation provides the humidity and energy to reform a portion of the incoming fuel.

Blower

Oxidizer

**Fuel**

Cathode

Anode

Figure 29 SOFC with anode recirculation and adiabatic reforming

The controller measures the blower speed, *N*, the cathode inlet and exhaust temperatures, *Tin* and *Tout*, and the stack voltage, *Vstack*. The controller uses two cascade PI control loops to track power while maintaining a constant operating temperature, *Target1*, and fuel cell temperature gradient, *Target2*. Feed-forward controllers determine the fuel flow, *ṅCH4*, operating current, *Jsetpoint*, and anode recirculation, *r*. The first cascade PI controller (32) determines the target cathode inlet temperature, *Tin,set*. This target temperature determines the error in the heater bypass position (33).

|  |  |  |
| --- | --- | --- |
|  |  | (93) |
|  |  | (94) |

Under certain transient or off-design conditions the bypass controller may saturate at zero, implying additional heat needs to be injected into the system. This heat is supplied in the form of additional fuel that is unused by the SOFC, and thus generates heat in the oxidizer. A complementary control loop (34) lowers the fuel utilization when the Bypass controller saturates. Similarly the bypass will not open until the SOFC has returned to it nominal fuel utilization.

|  |  |  |
| --- | --- | --- |
|  |  | (95) |

The second cascade PI controller determines a target speed for the blower, and uses the difference between the target speed and the measured speed to adjust the power supplied to the blower.

|  |  |  |
| --- | --- | --- |
|  |  | (96) |
|  |  | (97) |

The net stack current is calculated from the target power supply in kW, the blower power supply in kW, and the measured voltage according to (37).

|  |  |  |
| --- | --- | --- |
|  |  | (98) |

The fuel flow rate is calculated from a known composition of the fuel and a desired operating fuel utilization according to (38) and (39).

|  |  |  |
| --- | --- | --- |
|  |  | (99) |
|  |  | (100) |

The anode recirculation is found by implicitly solving expression (40) describing the inlet steam to carbon ratio, *S2C*.

|  |  |  |
| --- | --- | --- |
|  |  | (101) |

Table 1 presents the nominal control gain values used for transient load following control of with an SOFC system.

Table 1. Controller PI control parameters for cascade SOFC control

|  |  |  |  |
| --- | --- | --- | --- |
| Control State | Nominal Value (\_)0 | Proportional Gain *KP* | Integral Gain *KI* |
| *Tin,set* | 725 Kelvin | .5 | 1e-4 |
| *Bypass* | 5% | 2 | 2e-2 |
| *Utilization* | 80% | .2 | 2e-2 |
| *Ntarget* | 5000 | .4 | 1e-3 |
| *BlowerPower* | 3kW | 5 | 2e-2 |

### Example System Model oxySOFC:

### Example System model rSOFC:

H2O

O2

**SOFC**

**WGS / H2 recovery**

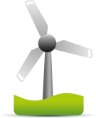
H2O

CO2

CH4

H2

Forward: Power Production



**Steam Reforming**

**SOEC**

**Partial Methanation**

**CH4 / H2 separation**

Reverse: Fuel Production

H2

## Component Functions

Each component function can represent a different piece of physical hardware. Any equations can be used to represent the physical hardware. All component functions must adhere to the following form and include specific elements in addition to the 4 variables that must be in the model function, i.e. i) type, ii) name, iii) connections, iv) TagInf. The name of the operation function defines the “type” used in the model function. For example, *function Out = FuelCell(varargin)* is the function for the FuelCell type component.

### Component function structure

Each component function has three sections, i) the very first initialization, ii) initialization to steady-state, and iii) the dynamic model. These are separated by the if, elseif, else statements:

global Tags

**if length(varargin)==1**

% First Initalization

* All fixed parameters of the component that only need to be initialized, i.e. they are not a function of the inlet values or operational conditions. These are saved in the structure *block.Parameter = value.*
* An estimate of all states and the appropriate scaling factor that will result in a value of order of magnitude 1. Scaling factors are put in the structure *block.Scale*. The initial values, divided by *block.Scale*, are put in *block.IC*. The upper and lower bounds must be specified for each state. This will prevent states that should never be negative from going negative, or valves to never open past fully open. If the state can increase or decrease indefinitely they should be specified as inf or –inf respectively. *block.UpperBound* represents the upper limit of the normalized state IC, while *block.LowerBound* represents the lower limit. Most states, such as pressure, temperature, and mass flow, have a lower limit of zero. A reversible fuel cell may have a negative lower limit to represent the negative current when it is in electrolyzer mode. It is important to have correct scaling values, i.e. non-zero, for all states in the component. The order of states specified by the vectors *block.Scale* and *block.IC* determine the order of states for the model. Once initialized these don’t change. Thus the operation function must use the same order of states. This order can be updated in the 2nd part of the initialization function if for example additional inlet species must be kept track of with states.
* A list of inlet and outlet port names saved into the structures *block.InletPorts* and *block.OutletPorts*, e.g. *block.InletPorts = {'NetCurrent','Flow1','Flow2'}*. The order in this list defines the order of the inlet ports when specifying connections in the model function, i.e. *Plant.Components.FC1.connections*.
* Each inlet and outlet must be given an initial condition. For inlet ports connected to another component, this initial value will be overwritten. If this port is unconnected, this initial condition will be held constant as the inlet to the block. The initial value can also be updated in the 2nd part of the initialization. For example *block.NetCurrent.IC = expected value of this port;*
* To help converge the mass flow/ pressure loop all components that input or output pressure must also do the following.
  + *block.PortName.Pstate = [];* if it is an inlet port receiving a pressure value from a different component
  + *block.PortName.Pstate = state#;* if it is an outlet port. The state number is the index in the vector of initial conditions associated with this pressure state.
  + *block.P\_Difference = {'Pout','Pin'};* list the port name pairs corresponding to the inlet & outlet of each stream.
  + *block.*dMdP = [ dMdP, C] according to the slope & intercept form dM/dP\*Pout - C\*Pin = mdot. If this variable does not exist, it assumes a constant pressure drop from the difference in initial condition of the inlet and outlet ports.

**elseif length(varargin)==2**

% Initalization to steady state

Inlet = varargin{2};

* This part of the function is called after all components have been through their first initialization. At this point you have an initial guess for all of the inlet ports that came from any connected components.
* This portion of the function updates the values of the states in *block.IC* and *block.Scale*, any outlet port initial conditions, *block.PortName.IC*, and any othe component parameters that depend on inlet conditions, *block.Parameter*.

**else**

% Dynamic model

t = varargin{1};

Y = varargin{2};

Inlet = varargin{3};

block = varargin{4};

string1 = varargin{5};

These inputs represent i) the current time of the simulation, ii) a vector of the current states (normalized near 1), iii) outputs of any connected functions that feed into this component, iv) the stored constant parameters of this component and v) a variable specifying one of two options.

**if strcmp(string1,'Outlet')**

* All math relating inlets and current states to the outlet values
* Out.PortName = value for all ports
* Tags.(block.name).Parameter = Parameter value for all variables you want to keep track of

**elseif strcmp(string1,'dY')**

* All math relating change in states to the inlet and current state values
* Out = dY

**end**

**end**

## Controller Functions

Controller functions have all of the same requirements and structure as component functions, with the added specification of targets. These control targets are treated as inlets, thus the set-point can be determined externally. The targets also specify around what conditions the model can be linearized. The controller is initialized after all other components, and thus has the ability to call upon parameters of any of the components during its first initialization. The other purpose for making control functions a unique category, is the way they are treated during a system linearization. The linear model matrices, A, B, C, and D, are made for the ‘plant’, i.e. the system components, with the original controller left in place. This allows for optimal MPC controller to be readily developed so long as they patch the input/output ports of the original controller.

## Additional Reference Functions

## Description of Specific Components

### Battery

### Building

### Blower

### Capacitor

### Combustor

### Compressor

### DCDCConverter

### FuelCell/Electrolyzer

Understanding of the dynamic performance characteristics of SOFC and SOEC systems is essential to design, even if the system is intended to operate at steady-state (base loaded). Transient events, such as start-up and shut-down, are inevitable for all systems. Maintaining operation of the entire system within desired constraints using the limited actuation available, requires control systems developed upon a fundamental understanding of the component dynamics and coupled relationships within the system.

While the simplicity of base loaded operation is preferred from a controls and engineering perspective, SOFC and SOEC systems are increasingly being designed for transient and load-following operation in response to greater adoption and accommodation of intermittent renewable power. The fast response characteristics of emerging SOFC and SOEC systems are well suited to meet the dynamic loads of off-grid applications, and support the electrical grid through spot-pricing and ancillary service markets. Displacing inefficient legacy generators operating on more carbon intensive fuels, SOFC and SOEC systems provide a highly responsive complement to intermittent renewable generation, thus leveraging the grid decarbonization at higher renewable market penetration.

The dynamic simulation approach presented applies to a variety of energy conversion systems and devices including fuel cells and electrolyzers. Some simplifications are required to solve the dynamic conservation equations (mass, energy, momentum) with transient responses ranging from milliseconds to hours within the highly coupled integrated systems with reasonable computational effort. Typical bulk model methodologies, e.g. representing each component as a single node with a single set of uniform conditions, avoids much of the computational rigor, but misses key system interactions and underlying constraints. Many bulk models attempt to address the non-uniform distribution of reactions, temperatures, and gas composition with linearization approximating steady operation. These approximations, typically made at nominal operating conditions, are a poor proxy for the non-uniform distributions at part load, and are particularly inadequate to represent the non-linear transient responses that must be addressed with integrated control schemes. The physics, chemistry and electrochemistry of SOFC and SOEC operation cannot be well captured or characterized with a bulk model.

SOFC and SOEC performance is inherently spatially dependent, that is, the major performance characteristics (e.g., temperature and current density) cannot be well predicted without knowledge of the spatial variations in temperature, species concentrations, etc. As a result, some degree of spatial resolution is required in the major components of SOFC and SOEC dynamic models. The first key simplification is determination of the limited spatial resolution to resolve the geometry in such a way as to capture only the directions in which major parameters that govern performance change significantly. The model must sufficiently capture the significant physics, chemistry and electrochemistry spatial dependence without overburdening the computational effort. When applicable, symmetry within the stack and within individual repeating units of the stack can reduce the modeling scope.

For example, in an SOFC or SOEC stack, the cell-to-cell variations are typically very small. Assuming a vertical stacking orientation, stacks comprised of a limited number of repeating cell units, i.e. bipolar plate and cell, between successive end plates, the heat transfer boundary conditions in the vertical direction can represent the nth cell from the end plate. For well-insulated stacks, those in which significantly more heat is removed via the gas flows than through conduction or radiation to the enclosure, the lateral heat transfer boundary condition can be set to zero, or some other small constant heat leakage.

Symmetry within the repeating cell unit can further reduce the spatial modeling requirement. Co-flow or counter-flow configurations present symmetric temperature and concentration gradients when assuming a uniform distribution of gas flow. Thus, the most significant spatial variations of the physics, chemistry, and electrochemistry governing performance are one-dimensional (1-D), and representing a single channel or flow path may be sufficient. On the other hand, cross-flow or serpentine flow patterns, or significant heat loss near the cell edges necessitates a two-dimensional (2-D) model.

The resolution of the 1-D or 2-D model depends upon a number of factors that affect the linearity of the concentration, temperature and reaction gradients. Specifically flow direction, internal reforming, fuel utilization, and the magnitude of lateral heat flux through conduction within the cell or bipolar plate relative to convective heat transfer between the gases and solid surfaces. Co-flow operation at low utilization without internal reforming and with low relative thermal conductivity presents the most linear performance gradients. Such operation could be well represented with as few as five nodes. Counter-flow geometry presents non-linear temperature profiles near the fuel and air inlets due to the dissimilar temperature of the gases. Typically, the higher cathode flow rates dominate the temperature profile, with the anode inlet gases slightly reducing the temperature gradient near the fuel entrance. Anode utilizations in excess of 75% typically exhibit a non-linear reduction in current density towards the fuel exit region due to a depletion of the reacting species concentrations. The reduced electrochemical activity reduces the heat generation, and thus reduces the temperature gradients in this region.

Internal reforming can result in significant cooling near the fuel inlet, resulting in a non-linear profile that is highly dependent upon flow rates, operating temperature, and inlet methane concentrations. Internal stresses, being highly correlated with thermal gradients, are most concentrated in the fuel inlet region of an internally reforming cell. Lower temperature operation reduces the kinetics of the reforming reactions and distributes the cooling effect along the fuel flow direction, reducing the stress concentration.

The disparate time scales of interest, specifically the electrochemical reaction time scale of sub-milliseconds, the bulk gas flow resident time scale, the control actuation time scale, and the thermal capacitance time scale of minutes to hours, can be leveraged to reduce the computational burden when simulating transient responses. At the fastest time scale, the most significant constraint is the equipotential constraint of the cell. This algebraic constraint forces an iterative solution for the current density within each step of the ODE solution for mass and energy equations. A small relaxation of the equipotential constraint can drastically reduce the computational burden, while still ensuring the constraint is held within a specified tolerance, i.e. 1e-6 V, at the smallest time scale of interest, i.e. milliseconds. An appropriate variable time step stiff ODE solver can partially separate the solution of the bulk flow and temperature states from the faster equipotential ODE determining the current distribution. Likewise, the stiff-ODE solver can solve for the transient concentrations within the bulk flow with a linear approximation of the slower temperature dynamics.

The variety of components and component arrangements within commercial and prototype SOFC and SOEC systems lends itself towards a modular modeling approach. The equations governing the dynamic performance of each sub-system can be arranged in a series of non-linear ODE equations conforming to (1). The linkages between components replicates the physical links of gas flow rates, pressures, and temperatures. All of the models are based on the fundamental mass, momentum, and energy conservation equations plus detailed solutions of electrochemical, chemical, and heat transfer processes in a nodal model with limited geometric spatial resolution.

|  |  |  |
| --- | --- | --- |
|  |  | (102) |

The governing equations for a spatially resolved SOFC or SOEC model closely resemble those of most bulk models. In some respects, the spatially resolved models are simpler, or at least have fewer calibration coefficients, because terms which address gradient-based factors such as concentration losses, are implicitly included due to the calculation of local concentrations. One approach, depicted in Figure V‑30, for adapting a bulk system model into a spatially discretized model of *n* control volumes, places *n* bulk models in series and adds equations representing the lateral heat transfer between nodes.

An alternative option, depicted in Figure V‑31, formulates the model as a system of equations solved with vectors of length *n*. The *i*th index of each vector represents the state of the *i*th node. A transformation matrix *T*, maps the outlet gas flow states of one node to the inlets of the subsequent node. The conductive heat transfer coefficients, *k·A*, can be multiplied by the temperature differences in adjacent nodes and supplied to the model as a vector of net heat gain into each node, ***q̇***. In this fashion a different flow geometry is readily represented by changing the transformation matrix, *T*. The second method, which will be employed for all subsequent work, provides several benefits and challenges.

* The spatial discretization can be generalized to *n* control volumes allowing a single model to achieve any desired spatial resolution.
* The model script size is reduced allowing shorter load times and faster simulations
* The vector mathematics are readily solved by most computational engines faster than *n* separate sets of equations.
* 1-D, 2-D or 3-D discretization is readily possible with changes to the transformation matrix, *T*.
* Varying flow geometries (co-flow, counter-flow, cross-flow…) can be accommodated through changes to the transformation matrix, *T*..

**Node 1**

**Node 2**

**Node *n***

Gas Flow

Gas Flow

Heat Transfer

Heat Transfer

Flow Input

Flow Output

Figure V‑30 Series connected bulk models with intermodal heat transfer

**Nodes 1, 2,…*n***

***T***

***nth states***

All States

Input 2, …*n*

Heat Transfer

Flow Input 1

Flow Output

Figure V‑31 Vector based nodal model with transformation and heat transfer matrix *T*

The model equations consist of an energy balance for each control volume, a mass balance for each gas species, an equipotential ODE for the local current flow, and a net mass balance for the pressure state of the gaseous control volume. These equations will be written in the form of (1) where the differential of the state variable is on the left-hand-side, and the function of the current state variables is on the right-hand-side. This model is applicable to both SOFC and SOEC operation. The energy balance for the bi-polar plate (2), cathode (3), SOFC or SOEC electrode and electrolyte assembly (4), and anode gas (5) follow a similar pattern of an energy balance divided by a time constant, *tCi*.

|  |  |  |
| --- | --- | --- |
|  |  | (103) |
|  |  | (104) |
|  |  | (105) |
|  |  | (106) |

Each flow, *Zi*, is represented by a structured variable consisting of the temperature in Kelvin, *Ti*, and *m* non-zero species flow rates, *(ṅj)i*. Equation (6) represents an SOFC anode operating solely on humidified hydrogen. Additional species can easily be considered with additional mass balances to the set of ODE’s being solved.

|  |  |  |
| --- | --- | --- |
|  |  | (107) |

A function *H(Zi)* calculates the total enthalpy of each flow (7) by summing the flow rate of each species multiplied by its enthalpy of formation and sensible enthalpy. The Schomate equation (8) with coefficients taken from NIST’s online chemistry Webbook is used to estimate sensible enthalpy of each species as a function of the local temperature divided by 1000, i.e. *t = Ti/1000*. Specific heat and entropy are calculated using the same reference data and equation (9) and (10).

|  |  |  |
| --- | --- | --- |
|  |  | (108) |
|  |  | (109) |
|  |  | (110) |
|  |  | (111) |

The *Q̇gen* term in equation (4) represents the heat produced by the electrochemical reaction. The heat produced is the enthalpy of the reaction less any energy extracted as electric power. The absolute value is placed around the voltage term to make this expression valid for both fuel cell and electrolyzer operations, assuming a system where negative currents correspond to the electrolyzer operation. The rate of hydrogen electrochemistry in kmol/s is equivalent to the current attributed to the hydrogen electrochemistry, *JH2*, divided by 2000 times Faraday’s constant, *F*. The hydrogen electrochemistry reaction (12) measures the change in enthalpy of the reactants. A similar expression (13) is applied for the carbon monoxide electrochemistry.

|  |  |  |
| --- | --- | --- |
|  |  | (112) |
|  |  | (113) |
|  |  | (114) |

The sensible energy crossing the electrolyte with the oxygen ion is captured in the *Q̇ion* term described in (14).

|  |  |  |
| --- | --- | --- |
|  |  | (115) |

Each species has a mass conservation equation to determine the change in the flow rate state, *(ṅj)i*. The flow rate state represents the flow of species *j* at the outlet of node *i*. The inlet is represented by node *i-1*, although the index of the inlet flow can change according to the transformation matrix *T*. The reaction rate of the individual species, *Rj*, depends upon the electrochemistry and the bulk chemistry of steam reforming, *SR*, and water gas shift, *WGS*, reaction rates.

|  |  |  |
| --- | --- | --- |
|  |  | (116) |

The reaction rates of the primary species of interest are detailed in (16-21), though additional species and reactions can be readily added.

|  |  |  |
| --- | --- | --- |
|  |  | (117) |
|  |  | (118) |
|  |  | (119) |
|  |  | (120) |
|  |  | (121) |
|  |  | (122) |

Resolution of the local current distribution depends upon the total current set point determined by the system controller, *Jsetpoint*, the local temperature and concentration conditions of the node, and the intrinsic material properties of the SOFC or SOEC. The temporal relaxation of the equipotential constraint is done using the ODE of (22). The equation combines the error in total current with the local voltage error. The difference between the local voltage, *Vi*, and the average node voltage, *Vavg*, is divided by the effective resistance of the node to approximate a local current error. The time constant of this ODE is set small enough that the ODE approximates an algebraic constraint on the millisecond time scale. The absolute value terms for the voltages allow equation (22) to apply in both fuel cell and electrolyzer operation.

|  |  |  |
| --- | --- | --- |
|  |  | (123) |

The local voltage is calculated from the Nernst potential, the local current density, the membrane thickness, *tM*, an electrolyte constant, *EC*, and the exponential of an activation energy, *ΔGact*, and the local temperature, *Ti*.

|  |  |  |
| --- | --- | --- |
|  |  | (124) |

The Nernst voltage is determined from a reference voltage, the local temperature, and the local reactant partial pressures.

|  |  |  |
| --- | --- | --- |
|  |  | (125) |

The local reactant partial pressures are taken to be the average of the node inlet and exit concentration by summing the flow rates for the node of interest and the upstream node.

|  |  |  |
| --- | --- | --- |
|  |  | (126) |

The final ODE equations solve for the pressure state of the anode and cathode compartments by solving a mass storage equation. The linear friction factor, *ffanode*, represents the linear approximation of the flow vs. pressure drop relationship across the anode. The downstream node pressure is represented by the index *i-1*, although the index may change with the transformation matrix *T*.

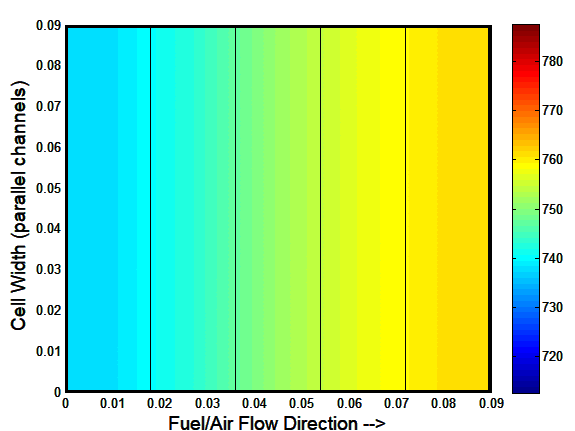
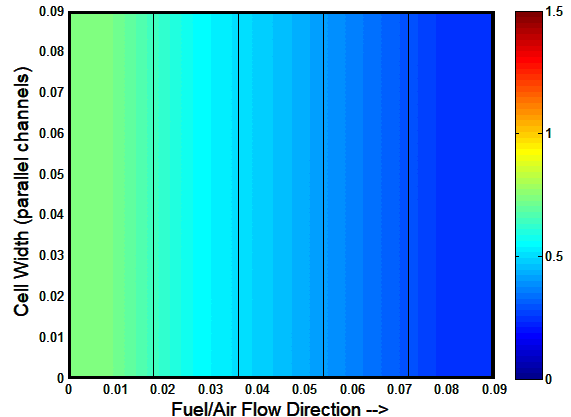
|  |  |  |
| --- | --- | --- |
|  |  | (127) |
|  |  | (128) |

The modeling coefficients used are detailed in Table 2.

Table 2 Parameters of nominal SOFC or SOEC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component** | **Parameter** | **Variable** | **Unit** | **Value** |
| **Stack** | Length | L | m | .25 |
|  | Width | W | m | .25 |
| **Bipolar Plate** | Thickness | tB | m | 0.006 |
|  | Density | ρ | kg/m3 | 1975 |
|  | Specific Heat | C | J/kg\*K | 611 |
|  | Conductivity | K | W/m\*K | 25.23 |
| **Anode** | Channel Height | hA | m | 0.002 |
|  | Channel Width | wA | m | 0.005 |
|  | Wall Thickness | tWA | m | 0.002 |
| **Cathode** | Wall Thickness | hC | m | 0.002 |
|  | Channel Height | wC | m | 0.002 |
|  | Channel Width | tWC | m | 0.005 |
| **Electrolyte** | Membrane Thickness | tM | m | 18e-6 |
|  | Electrolyte Constant | EC | K/Ω·m | 2e3 |
|  | Activation Energy | tWA | kJ/kmol | 8e3 |
|  | Density | ρ | kg/m3 | 375 |
|  | Specific Heat | C | J/kg\*K | 800 |
|  | Conductivity | k | W/m\*K | 6.19 |

An implementation of the model described can readily compute the steady-state and dynamic distribution of temperature, current, and reactant species. The simplest case of a symmetric co-flow cell with a hydrogen anode operating with a hydrogen utilization of 80% results in the linear temperature and current distribution shown in Figure V‑32. The average temperature of the cell is 750°C, the cathode inlet to outlet temperature difference is 50°C and the average current density is 0.5 A/cm2.

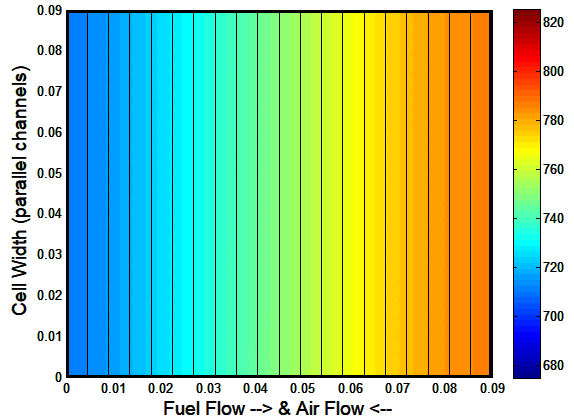
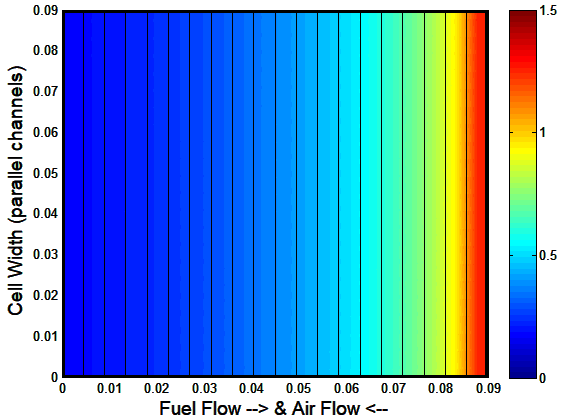
 

**Electrolyte Current Density (A/cm2)**

**Electrolyte Temperature (°C)**

Figure V‑32 Electrolyte temperature profile (a) and current density profile (b) of a co-flow hydrogen fueled SOFC

The counter-flow arrangement shown in Figure V‑33 was controlled to maintain the 750°C operation with the cathode inlet to outlet temperature difference remaining at 50°C and the average current density remaining at 0.5 A/cm2. The temperature gradient of the cell increases, particularly in the fuel entrance region, necessitating a higher spatial resolution. Twenty nodes were modeled. There is an increase in the current concentration near the fuel entrance of the cell, and as a result the temperature in that region rises. This lowers local resistance and further increases the current density in the fuel entrance region.

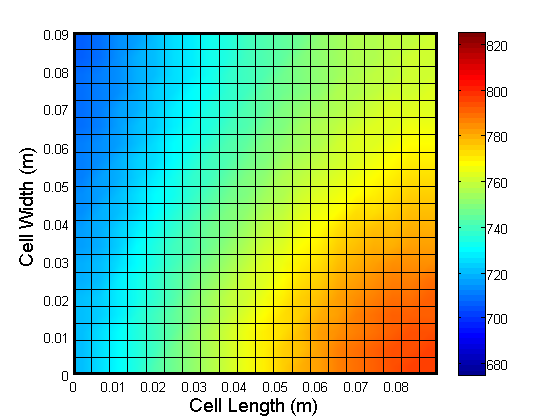
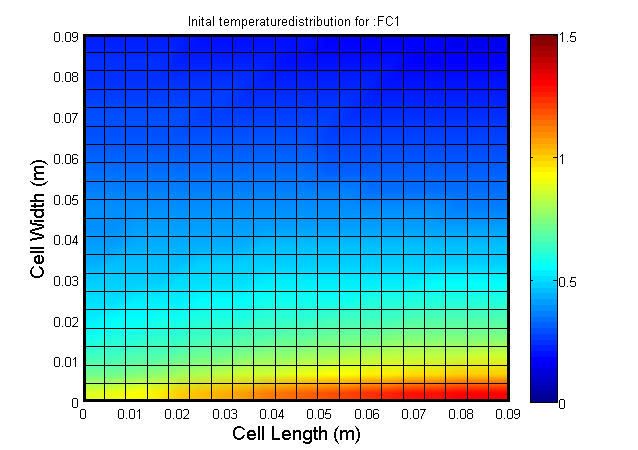
 

**Electrolyte Temperature (K)**

**Electrolyte Current Density (A/cm2)**

Figure V‑33 Electrolyte temperature profile (a) and current density profile (b) of a counter-flow hydrogen fueled SOFC

The temperature and current distributions become even more interesting when simulating a cross-flow or serpentine flow pattern with a 20 by 20 node spatial resolution. In the cross-flow arrangement shown in Figure V‑35, the operation was controlled to maintain the 750°C operation with the cathode inlet to outlet temperature difference remaining at 50°C and the average current density remaining at 0.5 A/cm2. Air still flows from left to right, but fuel flows from bottom to top of the figure. Again the high current density arises in conjunction with a high temperature region in the lower right hand corner.

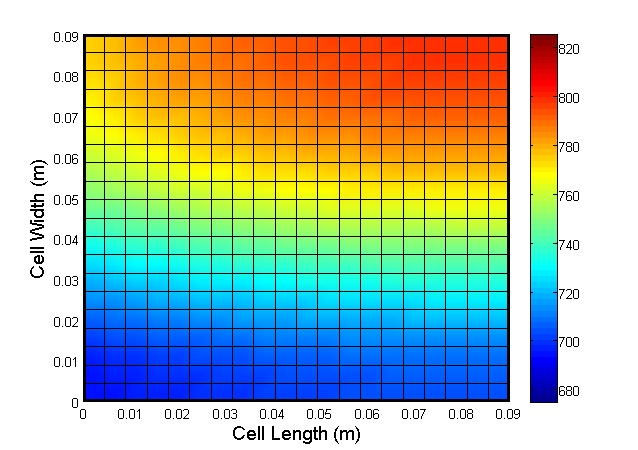
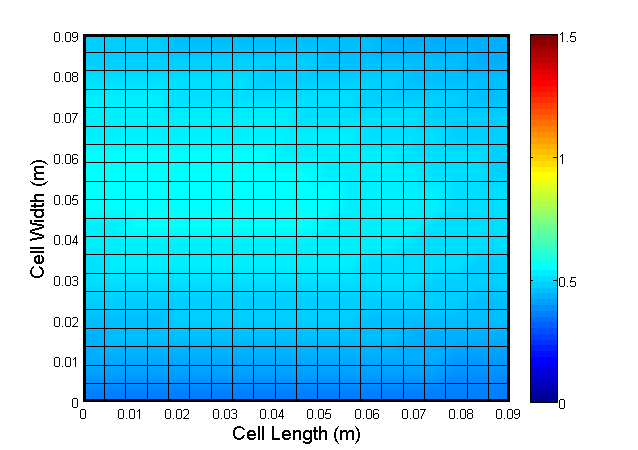
 

**Electrolyte Temperature (K)**

**Electrolyte Current Density (A/cm2)**

Figure V‑10 Electrolyte temperature profile (a) and current density profile (b) of a cross-flow hydrogen fueled SOFC

Internal reforming can also have a significant impact on the spatial temperature profile, as shown in Figure V‑35. Changing the fuel from hydrogen to a natural gas mixture approximately 90% CH4, 4% CO, 4% CO2, and 2% N2, and enabling direct internal reforming shifts the hottest portion of the cell from the fuel inlet to the fuel exit. Anode recirculation provides anode fuel humidification with a steam-to-carbon ratio of 3. The cathode flow rate is reduced to maintain the same 50°C temperature difference from cathode inlet to exhaust. The average current density is maintained at 0.5A/cm2. Due to the counterbalancing rates of natural gas reforming and hydrogen consumption, the hydrogen concentration is more uniform across the cell, resulting in a more uniform current distribution. Due to the internal reforming the temperature gradients are significantly more severe, leading to additional thermal stress.

**Electrolyte Current Density (A/cm2)**

**Electrolyte Temperature (K)**

Figure V‑35 Electrolyte temperature profile (a) and current density profile (b) of a cross-flow internally reforming SOFC

The heat exchanger and reformer components can be spatially resolved in nearly identical fashion with a few notable differences as discussed in the next section.

### HeatExchanger

The heat exchanger and reformer components can be spatially resolved in nearly identical fashion with a few notable differences.

The most commonly modeled device improved by spatial discretization is the heat exchanger. Bulk heat exchanger models ineffectively capture transient or off-design performance due to complex geometry as well as localized physics of heat transfer and mass flow. Most heat exchangers are comprised of two fluids moving past opposite sides of a solid divider. A simple model comprised of three control volumes per node can be calibrated to represent any variety of common heat exchanger designs. With sufficient spatial resolution the temperature profile within a single control volume can be assumed linear, thus avoiding the additional calculations of the NTU of LMTD methods. Consistent with the ODE form of (1), equations (28), (29), and (30) represent the energy balance ODE for the cold gas temperature, *TC*, the solid heat exchanger temperature, *TS*, and the hot gas temperature, *TH*. These equations only consider convective heat transfer between the gases and the solid surface, and conductive heat transfer between nodes of the solid heat exchanger material, *q̇cond*. Equation (31) solves for the conductive heat transfer into a node by summing the heat transfer from andacent nodes which have a common edge. The format is extendable to include radiative heat transfer if the geometric form and local view factors are known.

|  |  |  |
| --- | --- | --- |
|  |  | (129) |
|  |  | (130) |
|  |  | (131) |
|  |  | (132) |

### Reformer

A spatially resolved reformer model follows the format of the heat exchanger, but adds the individual species ODE’s from the fuel cell / electrolyzer modeling to properly account for the changing species across the reformer.

### LeakageValve

### MixingVolume

### Oxidizer

### Shaft

### SimpleMix

### Source

### Turbine

### Valve3Way

### ZonalBuilding

## Description of Specific Models

### SOFCstack

### SOFCsystem

### SOECstack

### GasTurbine

### OxyFC

### MCFCsystem

# Glossary of Functions

## Project Variables

Model\_dir: The saved directory that EAGERS resides in (should have a way in Python to just call up this directory without having global variable)

Plant: Structured variable with the following fields:

Necessary Fields:

Name: Unique name for the project

Generator: Structure containing all of the generation equipment & utilities

optimoptions: structure containing all of the options (see detailed section later)

Network: Structure outlining the location of each node, its connection to other nodes

and the equipment at that node

Optional Fields:

Building: Structure listing all of the buildings in the project

Weather: Data file for historical weather

Market: Information regarding ancillary service market

Costs: Structure of equipment and operating costs

Data: Historical data of Demand, Weather, Hydro etc.

Fields added during initialization & simulation

subNet: Re-organization of Network to facilitate building the optimization matrices

OpMatA: First optimization with no lower bounds and zero intercept

OpMatB: Second optimization with all constraints

OneStep: Intermediate optimization for a single time step

Dispatch: Structure storing the current dispatch as the simulation is run

Predicted: Structure storing each predicted dispatch for the entire horizon

WYForecast: Annual river network dispatch

TestData: Structured variable with following fields

Timestamp: a vector of dates coordinating the time each row of the data applies to

Weather: A structured variable containing dry bulb, relative humidity, and irradiance

Demand: A structured variable with each demand type. Columns of the same demand can represent loads at different nodes (python directly associates of load-node pairs)

Building: A structure containing the nonHVAC, DCloads, and internalGains of a building. Each column represents a different building.

Hydro: A structure containing outflow, and source/sink data for each node in the river network.

Also includes its own timestamp field to allow different time intervals from other data/demand categories

Plant.Generator has all the information of each generator necessary to complete the GUI’s and construct the optimization matrices.

Generator.Type: defines what category the generator is in, electric, CHP, battery, chiller, storage, grid….

Generator.Name: gives the generator a name

Generator.Source: specifies what the input is, i.e. fuel, bio-gas, waste heat. \*\*It is important to note what can have a quadratic vs. linear relationship. If we relate the input of a generator to its fuel, and only give the fuel a cost, the cost-curve of the generator can only be linear or varying in time. This makes splitting up costs between different fuels i.e. natural gas, bio-fuel, difficult, but may work well for hydro-power or other cases. For electric generators with non-linear cost-curves it is better to convert to $ when building the optimization matrix.

Generator.Output: specifies what the generator generates, ie. Heat, electricity, cooling, hydrogen…, and gives the output in terms of energy efficiency as a function of capacity. \*note that storage devices i.e. batteries hot water tank, do not have an output. Their Type determines what is stored and hence what is output. This may need to be revisited if we consider storage devices that output two things (heat+electricity)

Generator.Size: The capacity is in kW for generators and kWh for storage devices

Generator.Enabled: whether the device is operational and should be included in the dispatch or not. \*\*\*\*\*This may need to be revisited as some different structure if we want to include planned or unplanned shutdowns in the optimization.

Generator.VariableStruct: This contains characteristics specific to this type of generator. This information is editable in the GUI and used to build the Generator.OpMatA and OpMatB structures.

Generator.QPform: This structure helps in the construction of the optimization matrices associated with the multi-time-step quadratic programming

QPform.states: a string listing the states used to represent the generator i.e. x, y, & z. The states are each fields of QPform and contain the matrix values that should be associated with this state during the optimization.

QPform.cost: identifies the input cost used to convert input to $’s, so that the cost variables can be readily scaled with changing fuel costs. If the input is linked to a separate state for fuel consumption, its costs are zero and the value here does not do anything.

QPform.link.eq: a vector showing the values of each state in a row linking the states at a specific time-step. Paired with OpMat.A.link.beq.

QPform.link.ineq: same as above, but in the inequality matrix. A vector showing the values of each state in a row linking the states at a specific time-step. Paired with OpMat.A.link.b.

QPform.X.output.\_\_\_\_\_: Categories are same as Generator.Output (electricity, steam, heat, hydrogen…) \*\* note these are shortened to E, S, H, H2 …, if the field exists it has a value that should be associated with this state for the row relating generator outputs in this category to demand in this category.

QPform.X.Ramp: If this state has a ramping constraint its values are placed in Ramp.A and Ramp.b. Ramp.A can be a vector with two values if it is a single direction constraint, or a 2x2 matrix if it is constrained in ramping up & down.

QPform.X.H: The quadratic component of the cost, [] is interpreted as 0.

QPform.X.f: The quadratic component of the cost, [] is interpreted as 0.

QPform.X.ub: upper bound associated with this state.

QPform.X.lb: lower bound associated with this state.

List of constraints:

|  |  |  |  |
| --- | --- | --- | --- |
| Type of output | Electrical output | QPform.X.output.electricity | [] or 1 |
|  | Heat output | QPform.X.output.heat | [] or 1 or Hratio |
|  | Cooling output | QPform.X.output.cooling | [] or 1 or Cratio |
| Ramp Rates | Ramp up & down | QPform.X.Ramp.A | [-1,1;1,-1;] |
|  |  | QPform.X.Ramp.b | [rampupvalue, (-rampdownvalue)] |
| State splitting | How do the different states relate | QPform.link.eq | For a gen [1,-1,-1]  For grid [1,-1]  For storage [1,-1/roundtrip efficiency] |
|  |  | QPform.link.beq | 0 |
| Boundaries | Upper bound | QPform.X.ub | Generators(i).Size, inf, usablesize |
|  | Lower Bound | QPform.X.lb |  |
| Charging Rates | Handled in Ramp |  |  |
| Depth of Discharge | Handled in lower bound |  |  |
| Self Discharge | Can either be a constant or ratio of state of charge |  |  |
| Buffer | ? |  |  |

## Functions

## Planning Tool Variables

## Control Tool Variables

## Simulation Tool Variables

# Interface Flow Diagrams

## 1. EAGERS Interface

### Opening EAGERS

# Function Summaries

|  |  |
| --- | --- |
| **Design** | |
| BuildingProfile | * Estimates the energy profile of a building for each type of demand (E,H,C) * Accepts building characteristics (Type, Area, Schedules, etc.), weather profile for region, and date of simulation (possibly location if user specified) |
| SolarCalc | * Uses location of the plant & date to determine solar behavior * uses NOAA solar calculations |
| **Emissions** | |
| AvgEmissionProfile | * Finds the average emissions per month for a year for a plant |
| EmissionProfile | * Creates an emission profile for a plant |
| EmissionsCalculated | * Creates the baseline and dispatched emissions based on location and boilers |
| EmissionZone | * Gives emissions region based on location |
| **GUI** | |
| WelcomeScreen | * Creates a welcome splash screen (WelcomeScreen.fig) * Allows user to choose plant, and which tool to enter |
| Plant | * Defines the class “Plant” with all necessary variables |
| center\_menu | * Function that creates a popup menu GUI that is in the center of the screen |
| *Design* | |
| DesignCosts | * Calculates monthly and net present costs of the plant, based on the generation combinations * Incorporates O&M, fuel, and monthly financing from loans * Determines data to be plotted in the Design tool, monthly costs tab |
| EditSystem | * Functions needed to modify/update the system specifications * Populates a component tab with all variables associated to that type of generation/storage |
| MainScreen1 | * Handles and directs all Design tool GUI functionality (MainScreen1.fig) * Handles callbacks/redirection for all interactions in the planning tool (EPT) |
| ScheduleEditor | * Separate GUI interface (ScheduleEditor.fig) that allows the user to visualize and modify the uncontrolled loads within the simulated building * Can change ramp constraints, seasonality, & loads/temperature control set points |
| updateSystemRep | * Handles the creation & modifications to the plant by adding/removing a component in the GUI |
| *Graphics* | |
| * All visual graphics that are used in EAGERS | |
| *Mapping* | |
| * Functions from the mapping toolbox (DG-BEAT) | |
| *Optimization* | |
| DISPATCH | * Handles & directs all GUI functionality for the tabs of the optimization tool (DISPATCH.fig) * Handles callbacks & redirection for all interations in the dispatch control tool (EDC) |
| plotDispatch | * Function to plot on the GUI main screen in dispatch |
| plotDemand | * Function to plot forecasted and actual demand on GUI forecasting tab |
| plotLines | * Plots line power transfers and losses |
| plotSchedule | * Function to plot forecasted, actual and historical schedules in the forecasted tab |
| updateGUIstatus | * Updates the status indicators in the GUI of each component in the optimization |
| *Simulation* | |
| * empty | |
| **Model Library** | |
| * MATLAB scripts that load component models with all necessary variables needed for the Simulation tool | |
| **Optimization** | |
| *Basic Functions* | |
| agregateSSmodle | * Builds the state space model for the optimization * aggregates the similar matrices for each generator into 1 large matrix * Ex. large A,B, C,D matrices for ẋ = Ax + Bu, y = Cx + Du |
| automaticInitialConditon | * Given the Plant data, the function determines the initial condition for the optimization |
| buildTimeVector | * Creates the time vector based on the type of spacing specified by the user * Also based off of horizon, resolution, etc. defined in optimoptions |
| calculateHistoricalFit | * Uses existing building data to make a surface fit of demand vs. time vs. temperature |
| calculateHydroSOC | * Given power for each dam, transmission lines, and outflow of each dam, calculates the “state of charge” of the hydro components for a specified forecasted time |
| callQPsolver | * Takes in the QP & equality matrices for a generator and the specified type of solver to direct the optimization to the correct method * Outputs the solution (generator state) to the minimization strategy as well as the feasibility and cost associated with that solution |
| changeTimestep | * Changes the state space model to a new MPC frequency |
| createCombinations | * Creates all feasible combinations of components to meet demand |
| findbuffer | * Creates upper and lower buffer for storage |
| GenCosts | * Creates the cost curves (Fit A and Fit B) for a generator * Outputs all cost terms associated with the curves |
| GRIDandSTORAGE | * Function to decide whether the grid or storage (or a mixture of both) will make up the difference between generation and demand |
| initializeOptimization | * Script to get the plant ready for optimization * Guides through the functions to create the optimization matrices and to calculate the correct temperature/demand profiles |
| instantMarginalCost | * Outputs the marginal cost of each type of generation in the Plant |
| interpolateData | * Given the frequency of the optimization and timestamps of the data, the function creates data to match the frequency |
| lagd | * Creates the matrices A & L0 for the old MPC |
| loadBuilding | * Creates the QP form structure for a Building optimization |
| loadGenerator | * Loads the QP form for all “generators” in the plant * Calls the aggregate models |
| manualInitialCondition | * Sets the initial conditions based on manually defined values for the optimization |
| mccQPgen | * Solves QP minimization in an open source function (multiple centrality corrections) |
| MPCsolve | * Sets up and solves the original MPC problem for the next timestep * Uses a discrete time state space model |
| NetCostCalc | * Creates a cost for the next horizon based on cost of generation * Storage has no cost in this function |
| pcQPgen | * Solves QP minimization in an open source function (predictor corrector method) |
| RampRateCalc | * Given the state space, upper & lower bound, and Hratio, the function determines the ramp rate for each generator (dx\_dt) |
| RenewableOutput | * Determines the power output for renewable generation * Wind in still being developed |
| RunBaseline | * Old function to determine a baseline economic dispatch based on Plant |
| Timers | * Makes timers for different parts of the optimization * Ex. MPC, optimization, and dispatch timers |
| TypicalDay | * Given a temperature profile and a list of dates, the function creates a typical date of weather for each month in the date list |
| updateGeneratorCost | * Creates new updated generator costs given a new timestamp based on the rates of fuel sources |
| updateMarginalCost | * Creates a minimum and maximum marginal cost for the next timestep for each type of output |
| *Communications* | |
| closePorts | * Closes all ports that were established for the optimization |
| ExportResult | * Exports results to excel (xls file) |
| GetCurrentData | * Given a date, the function gets the Real Time Data for that time |
| GetCurrentState | * Opens all ports and gets the state of each connection point |
| GetCurrentTime | * Gives the time of the virtual or real optimization |
| GetGeneratorOnOff | * Given a measure, opens (then closes) a port to get a generator state |
| GetHistoricalData | * Given a date, this function finds the desired data at that time |
| getHydroFlows | * Given the date and location, the function finds flow associated with that node |
| LogData | * Records data for the tested plant, including fan, ambient temperature, and state of charge |
| openPorts | * Creates/opens communication ports for a plant to be tested/optimized |
| SetGeneratorOnOff | * Communication necessary to turn a generator on or off through ports |
| SetGeneratorOutput | * Changes the setpoint of a generator through communication ports |
| writeThermalLoad | * Simulates a thermal load with the fans |
| *Forecasting* | |
| ARIMAForecast | * Forecasting technique using 1st differenced values from 1 horizon ago and 1 timestep ago to predict the next 24 hours |
| ForecastBuilding | * Estimates the energy profile for a building for a building given features of the building/location * Uses schedules and loads |
| ForecastHydro | * Forecasts the source/sink at each timestep |
| ForecastZonalBuilding | * Forecsts heating and cooling needed to reach the temperature setpoint for a building |
| SNIWPEForecast | * Forecasting technique using simple exponential smoothing, based on previous forecasts and a correction error |
| SurfaceForecast | * Forecasting technique that creates a surface fit based on historical data |
| updateForecast | * Creates a new forecast depending on the selected technique |
| *HydroQP* | |
| build\_hydro | * Creates a hydro plant with all necessary variables to run in an optimization |
| build\_hydro\_Basin | * Creates a hydro basin plant with all necessary variables to run in an optimization |
| hydro\_diffTime | * Establishes the time it takes for flow from upriver to downriver |
| HydroTest | * Function used for testing plant structure * Build hydro is actually used |
| SOCchange | * Script to run to calculate inflow/outflow and yearly totals |
| *NetworkQP* | |
| build\_subNet | * Creates an organized group of subnetworks based on their position in the network * Includes the connection between nodes if there are more than one |
| buildMatrices | * Function to create the matrices needed to optimize the Plant in multistep |
| buildMatrices1step | * Builds optimization matrices without ramping constraints and storages in treated as a generator |
| checkStartupCosts | * Compares generator on/off with the startup cost and decides whats the best configuration |
| chillerOnly1step | * Creates dispatch options to include only the generation and storage related to cooling * Used if 3 or more chillers in a Plant |
| disableGenerators | * Updates the QP matrices with only enabled generators |
| DispatchLoop | * Given date, forecast, and possibly the last dispatch if it exists, directs the optimization through EAGERS to find the optimal dispatch (and make sure it’s feasible) |
| DispatchQP | * Visualization of the dispatch |
| eliminateCombinations | * After feasible generator combinations are created, this function removes all overly-costly combinations and returns the most efficient combinations and their cost * Number of combinations is decided by the “n\_best” variable |
| FilterGenerators | * Takes the first guess for the dispatch and filters generation to follow the rules that provide a more efficient solution * Rule ex. If there is a generator that is off at the start, trying turning off for a longer time |
| FindFeasible | * If the optimal dispatch is found infeasible, this function checks the dispatch without ramp rates * Isolates what constraint is making it infeasible |
| StepByStepDispatch | * Determines the generator outputs to meet the plant demand and optimize cost (without time constraints) |
| StepDispatchForward | * Steps the dispatch forward and stores the previous dispatch |
| updateMatrices | * Given the old QP matrices for the previous optimizations, updates all new values * Done without having to rebuild entire matrices |
| updateMatrices1step | * Updates the QP matrices with no time dependent variables, since there is only a single step |

# Appendix

|  |  |
| --- | --- |
| **Opening a Plant in the Dispatch Controller** | |
| **1)** Select a Plant and Optimization Tool from Welcome Screen | **DISPATCH**(SliderStartDate\_Callback) |
| initializeOptimization |
|  | |
| **2)** Locate the values needed to represent each generator (and building) in the quadratic optimizations, and create the corresponding state-space model | *loadGenerator* |
| *agregateSSmodel* |
| *loadBuilding* |
|  | |
| **3)** Organize the values from Step 2 into a structure that reflects the states of generation, storage and the building, that will also be used for the quadratic optimization | *build\_subNet* |
| *findbuffer* |
| *buildTimeVector* |
| *buildMatrices* |
| *buildMatrices1step* |
|  | |
| **4)** Determine an initial condition for the plant, forecast one day forward, and calculate the optimal dispatch over the forecast horizon | automaticInitialCondition |
| updateForecast |
| DispatchLoop |
|  | |
| **5)** Plot forecasted dispatch on GUI | plotDispatch |

|  |  |
| --- | --- |
| **Running a Virtual Optimization** | |
|  | **DISPATCH** (Start\_Callback)  \* |
| **1)** Once Start is clicked in the EDC, the start date is determined as well as the timestamp, temperature and demand for that day | *(get slider start date and optimization option values)* |
| *GetCurrentData* |
|  |  |
| **2)** Determine the initial conditions for the dispatch, as well as the time vector with the correct frequency | *automaticInitialCondition*  *(or manualInitialCondition)* |
| *buildTimeVector* |
|  |  |
| **3)** For every timestep of the dispatch, record the previous dispatch, then update, optimize, plot and step forward.  \*Continue this loop until the optimization is stopped. | *GetCurrentData* |
| *updateForecast* |
| *DispatchLoop* |
| *plotDispatch* |
| *StepDispatchForward* |

|  |  |
| --- | --- |
| **DispatchLoop for non-Mixed Integer** | |
| **1)** Assume the predicted dispatch is either 1) exactly the current state (if it’s the first dispatch) or 2) whatever has been dispatched so far | **Dispatch Loop** |
|  |  |
| **2)** Update the scale cost with the utility prices for the horizon (since all costs were assumed to be 1 initially) and Update the marginal cost with the predicted dispatch. | updateGeneratorCost |
| updateMarginalCost |
|  |  |
| **3)** Using these updated costs, update the QP matrices for fit A, then use them to determine the initial dispatch (which is also the “optimal dispatch”) | updateMatrices |
| callQPsolver |
|  |  |
| **4)** Update marginal cost with optimal dispatch. | updateMarginalCost |
|  |  |
| **5)** Using the new marginal costs, update the QP matrices for fit B, then use them to determine the generator dispatch. | updateMatrices |
| callQPsolver |
|  |  |
| **6)** Make sure this is the most efficient dispatch | FilterGenerators |