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](#_Toc481569895)

[i. Getting Started 1](#_Toc481569896)

[ii. Starting a New Project 1](#_Toc481569897)

[III. Planning Tool 2](#_Toc481569898)

[i. The Planning Tool Interface 2](#_Toc481569899)

[ii. Component Selection and Sizing 3](#_Toc481569900)

[iii. Automated Optimization of Plant Arrangement and Sizing 3](#_Toc481569901)

[IV. Control Tool 3](#_Toc481569902)

[i. The Control Tool Interface 4](#_Toc481569903)

[a. Optimization Options 5](#_Toc481569904)

[ii. Forecasting 6](#_Toc481569905)

[a. Simple Exponential Smoothing 6](#_Toc481569906)

[b. Auto-regressive integrated moving average (ARIMA): 6](#_Toc481569907)

[c. Neural Network: 6](#_Toc481569908)

[d. Historical Surface Fits 6](#_Toc481569909)

[iii. Dispatch Scheduling 8](#_Toc481569910)

[iv. District Energy Systems and Electric Networks 13](#_Toc481569911)

[v. Component Descriptions 13](#_Toc481569912)

[a. Utilities: Electric, Gas, District Heating or Cooling 13](#_Toc481569913)

[b. Generators, Chillers, Boilers, and CHP systems 13](#_Toc481569914)

[c. Energy storage systems 14](#_Toc481569915)

[d. Hydro Power 14](#_Toc481569916)

[e. Wind and Solar 14](#_Toc481569917)

[f. HVAC 15](#_Toc481569918)

[vi. Real-time Model Predictive Control 15](#_Toc481569919)

[V. Simulation Tool (STRIDES) 15](#_Toc481569920)

[i. Simulation Tool Interface 15](#_Toc481569921)

[ii. System Model Files 16](#_Toc481569922)

[a. Build Model 17](#_Toc481569923)

[b. Run Blocks 18](#_Toc481569924)

[c. CreateLinModel 18](#_Toc481569925)

[d. RunLinSystem 18](#_Toc481569926)

[e. Example System model Gas Turbine: 18](#_Toc481569927)

[f. Example System model SOFCsystem: 19](#_Toc481569928)

[g. Example System Model oxySOFC: 20](#_Toc481569929)

[h. Example System model rSOFC: 21](#_Toc481569930)

[iii. Component Functions 21](#_Toc481569931)

[a. Operation function arrangement 21](#_Toc481569932)

[b. Initialization function arrangement 22](#_Toc481569933)

[iv. Controller Functions 23](#_Toc481569934)

[v. Additional Reference Functions 23](#_Toc481569935)

[vi. Description of Specific Components 23](#_Toc481569936)

[a. Battery 23](#_Toc481569937)

[b. Blower 23](#_Toc481569938)

[c. Capacitor 23](#_Toc481569939)

[d. Combustor 23](#_Toc481569940)

[e. Compressor 23](#_Toc481569941)

[f. DCDCConverter 23](#_Toc481569942)

[g. Electrolyzer 23](#_Toc481569943)

[h. FuelCell 23](#_Toc481569944)

[i. HeatExchanger 24](#_Toc481569945)

[j. LeakageValve 24](#_Toc481569946)

[k. MixingVolume 24](#_Toc481569947)

[l. Oxidizer 24](#_Toc481569948)

[m. Reformer 24](#_Toc481569949)

[n. Shaft 24](#_Toc481569950)

[o. SimpleMix 24](#_Toc481569951)

[p. Source 24](#_Toc481569952)

[q. Turbine 24](#_Toc481569953)

[r. Valve3Way 24](#_Toc481569954)

[vii. Description of Specific Models 24](#_Toc481569955)

[a. SOFCstack 24](#_Toc481569956)

[b. SOFCsystem 24](#_Toc481569957)

[a. SOECstack 24](#_Toc481569958)

[b. GasTurbine 24](#_Toc481569959)

[c. OxyFC 24](#_Toc481569960)

[d. MCFCsystem 24](#_Toc481569961)

[VI. Glossary of Important Variables 24](#_Toc481569962)

[i. Project Variables 24](#_Toc481569963)

[ii. Planning Tool Variables 27](#_Toc481569964)

[iii. Control Tool Variables 27](#_Toc481569965)

[iv. Simulation Tool Variables 27](#_Toc481569966)

[VII. Interface Flow Diagrams 27](#_Toc481569967)

[v. 1. EAGERS Interface 27](#_Toc481569968)

[e. Opening EAGERS 27](#_Toc481569969)

[VIII. Appendix 27](#_Toc481569970)

# 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.

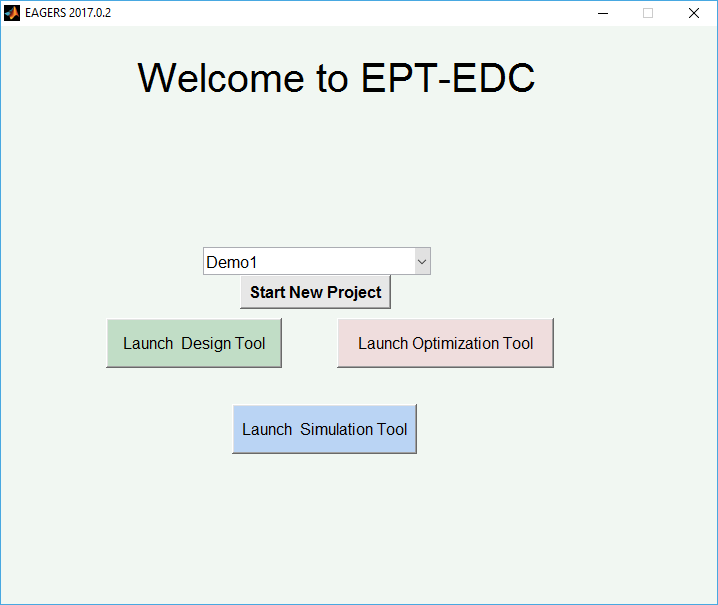


Figure 1 EAGERS welcome screen

## 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 Planning Tool Interface



Figure 2 Primary planning tool GUI

## Component Selection and Sizing

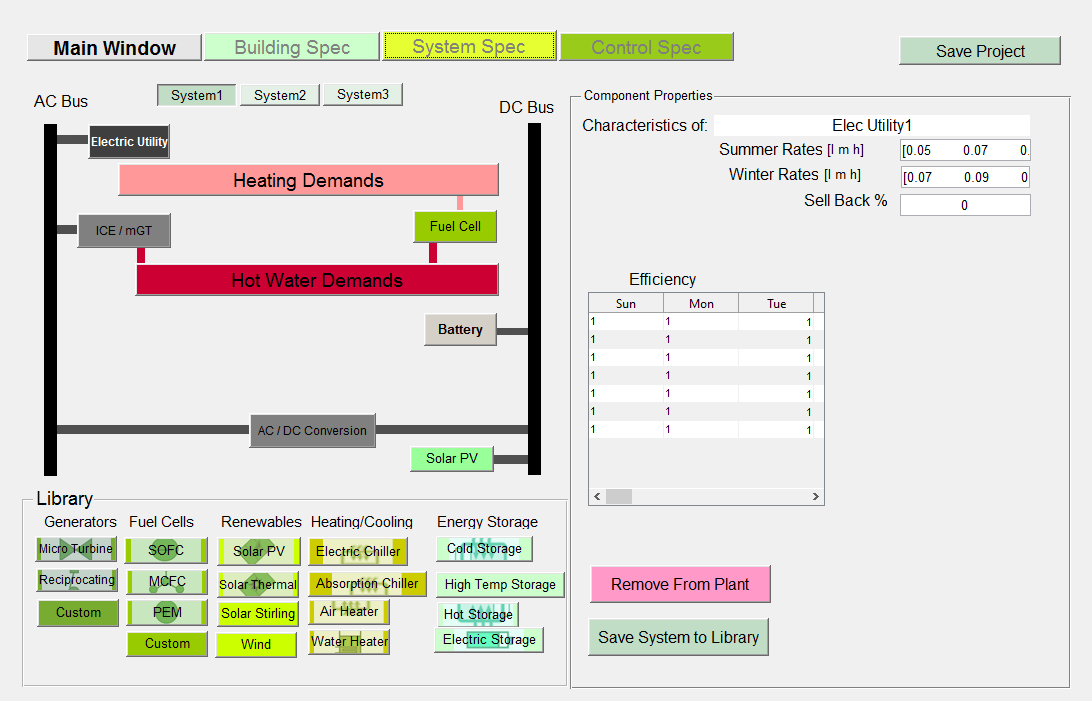


Figure 3 System specification tab

## Automated Optimization of Plant Arrangement and Sizing

# 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 4 outlines in block diagram the information pathways between the forecasting, dispatch, control and simulation components of the control tool.

**Fuel/Grid costs**

**Non-linear Building SID**

**measurements**

**Time/Date**

**Typical occupant behavior**

**Desired set-points**

**Forecast entire building**

**+**

**Controller (1hr)**

**Set-points**

**Optimal Control**

**Linear SID building**

**-**

**+**

**+**

**Non-linear FC/GT, etc.**

**+**

**Stochastic uncertainty**

**targets**

**cQP Dispatch Optimization (24hr)**

Figure 4 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

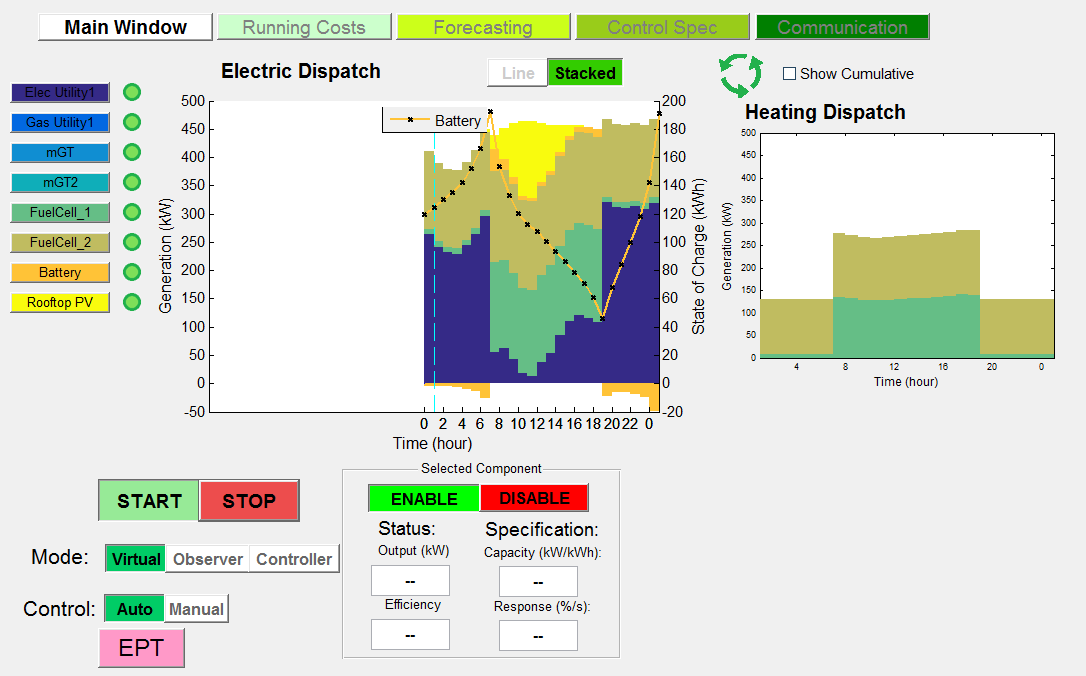


Figure 5 Main control GUI window

### Optimization Options

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

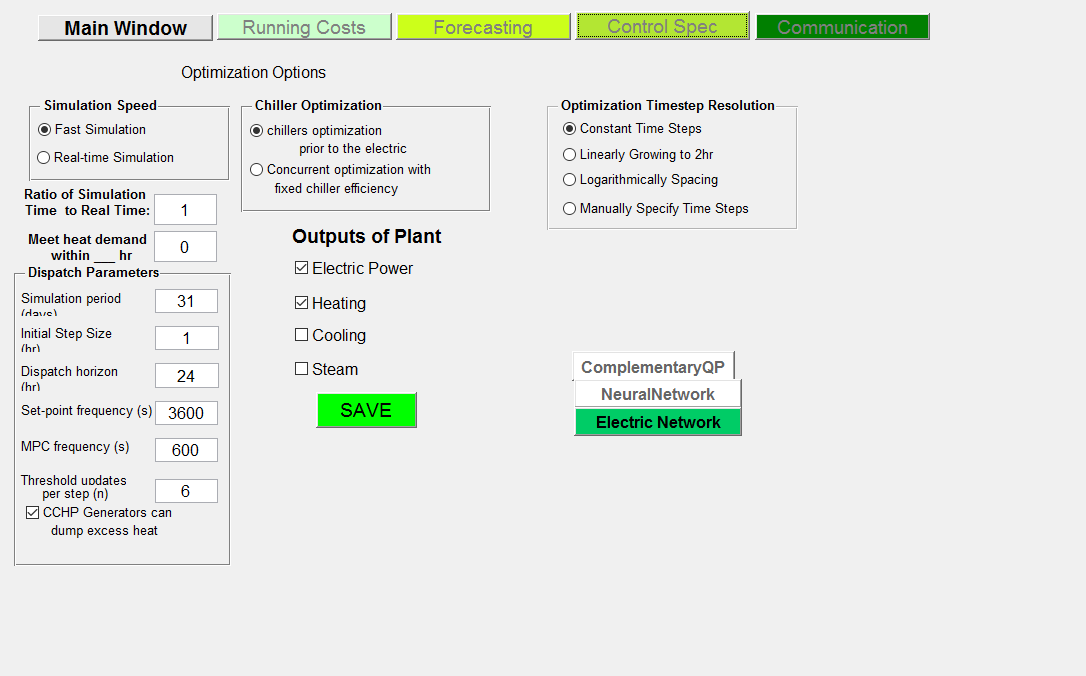


Figure 6 Optimization specification GUI

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 is:

|  |  |  |
| --- | --- | --- |
|  |  | (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.

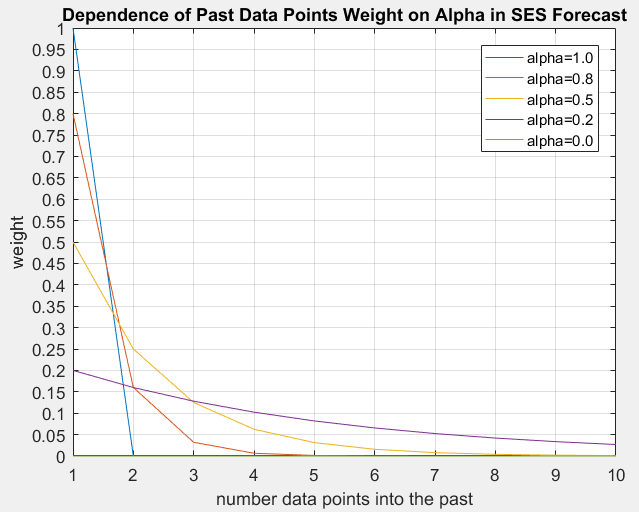


Figure 7 a) Historical weighting of data in SES method, b) R2 values as a function of alpha

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.

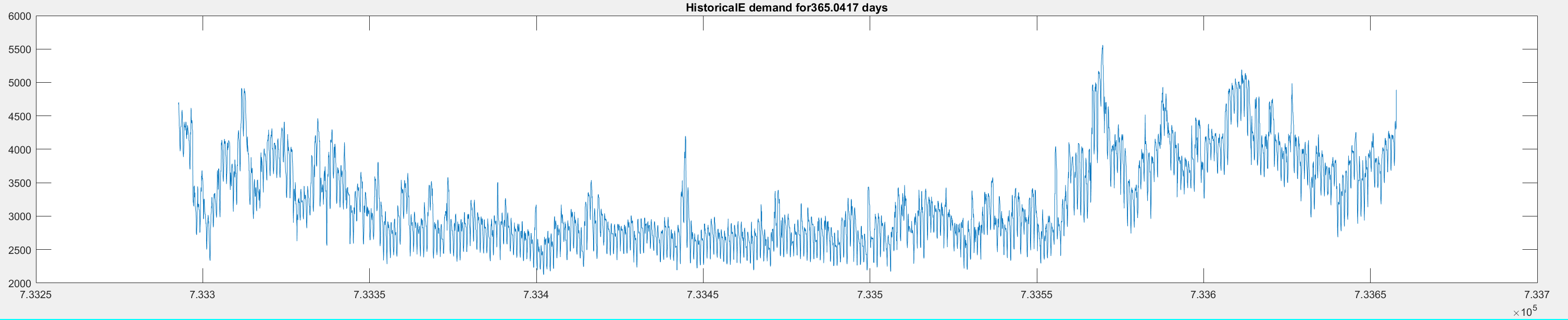


Figure 8 Historical electric data used for training

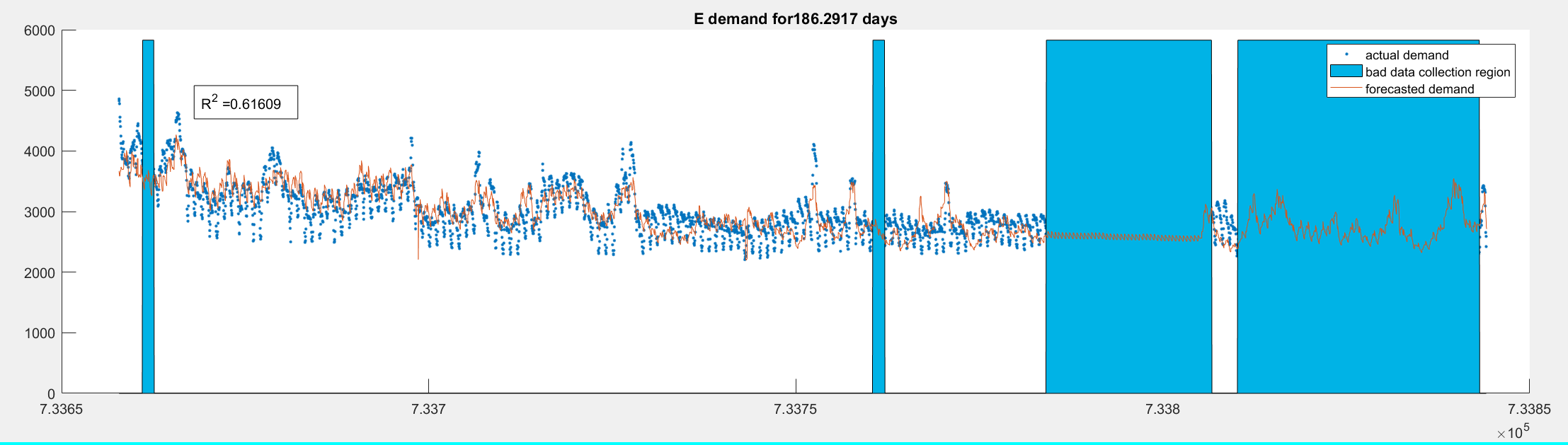


Figure 9 Data and network dispatch for electric demand from December to March.

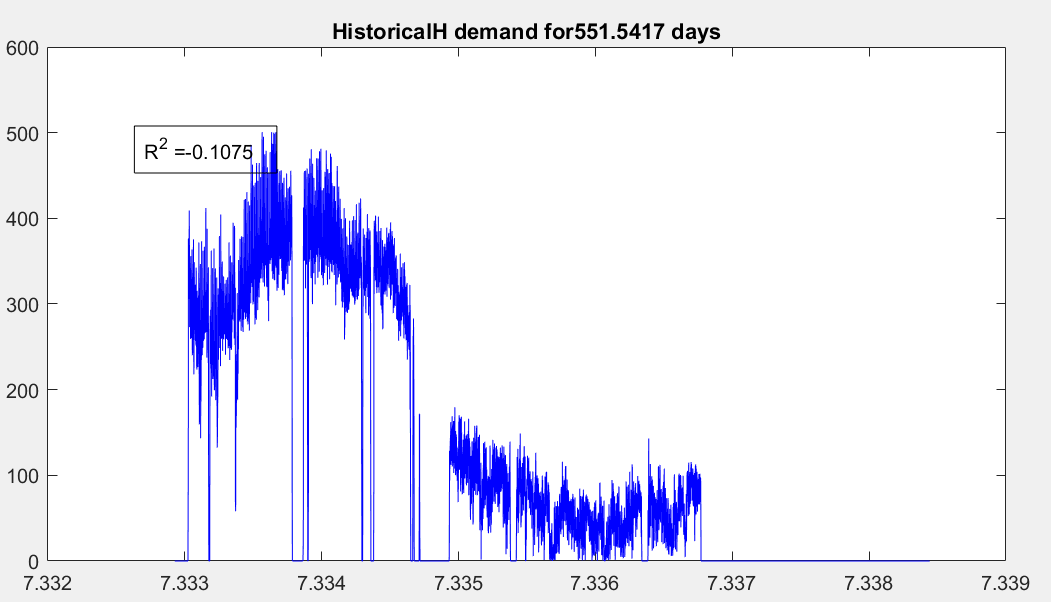


Figure 10 Heating demand, filtered data

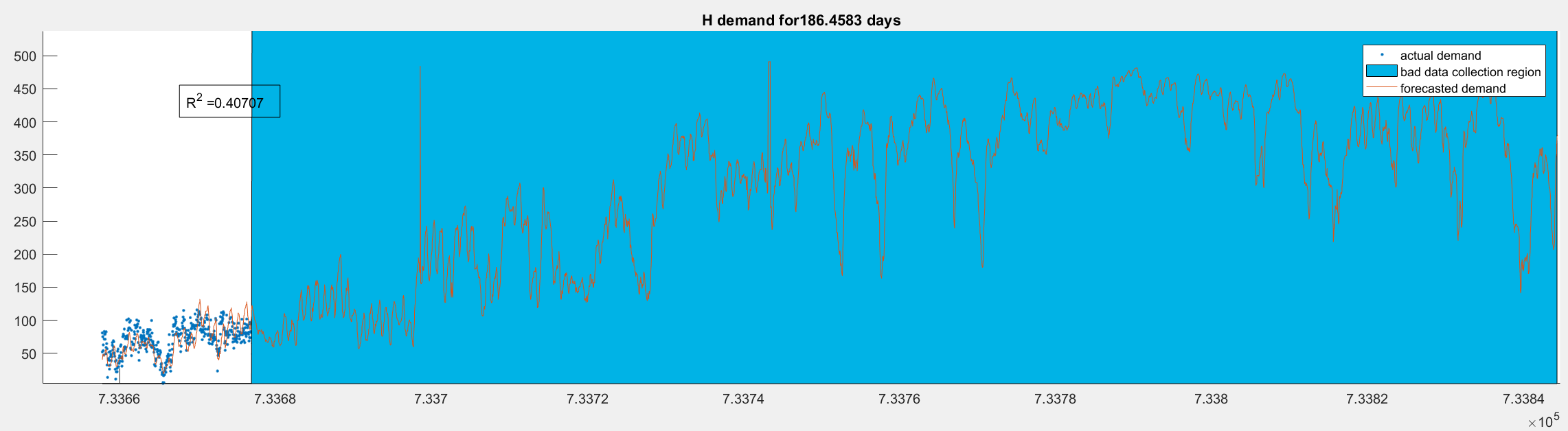


Figure 11 Heating demand forecasting

### 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 7.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 8.

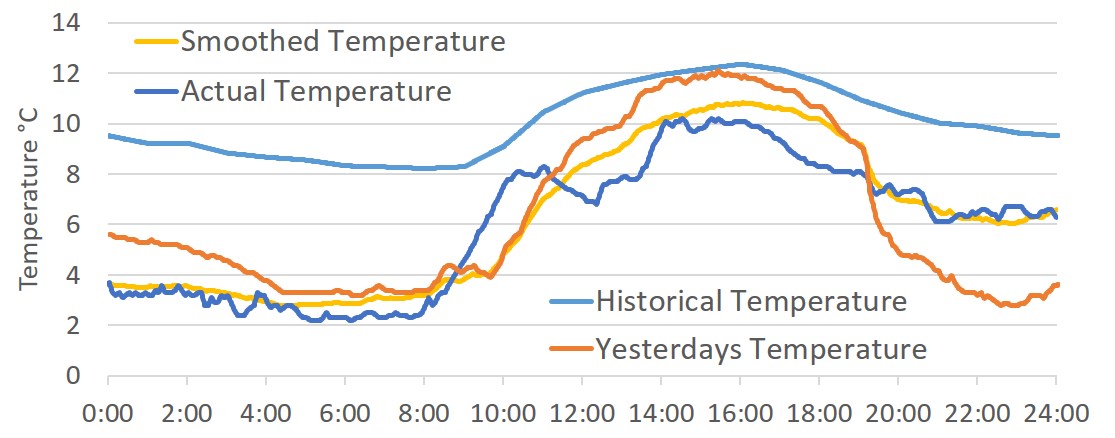


Figure 12 The temperature from yesterday has been averaged with the historical, then smoothed to form a prediction. This prediction is then compared to the actual temperature.

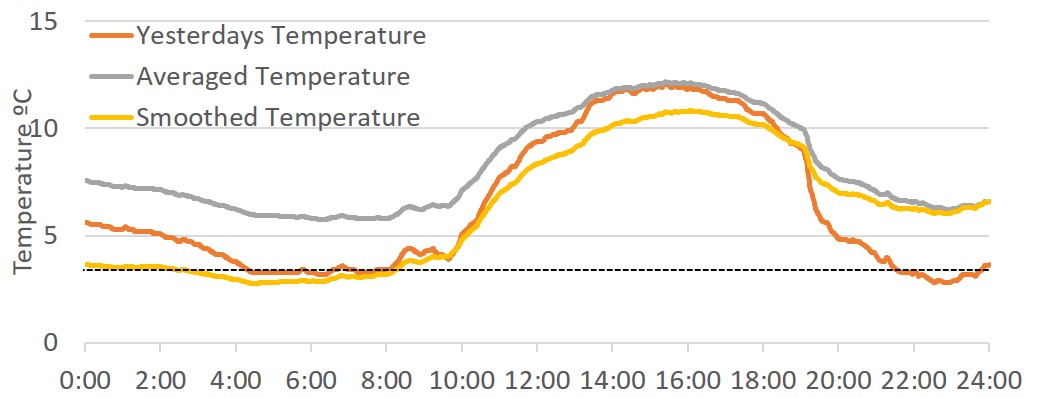


Figure 13 The last recorded temperature from yesterday is used to the determine the base for the smoothed temperature, which is then fit with the average of yesterday and the historical.

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.

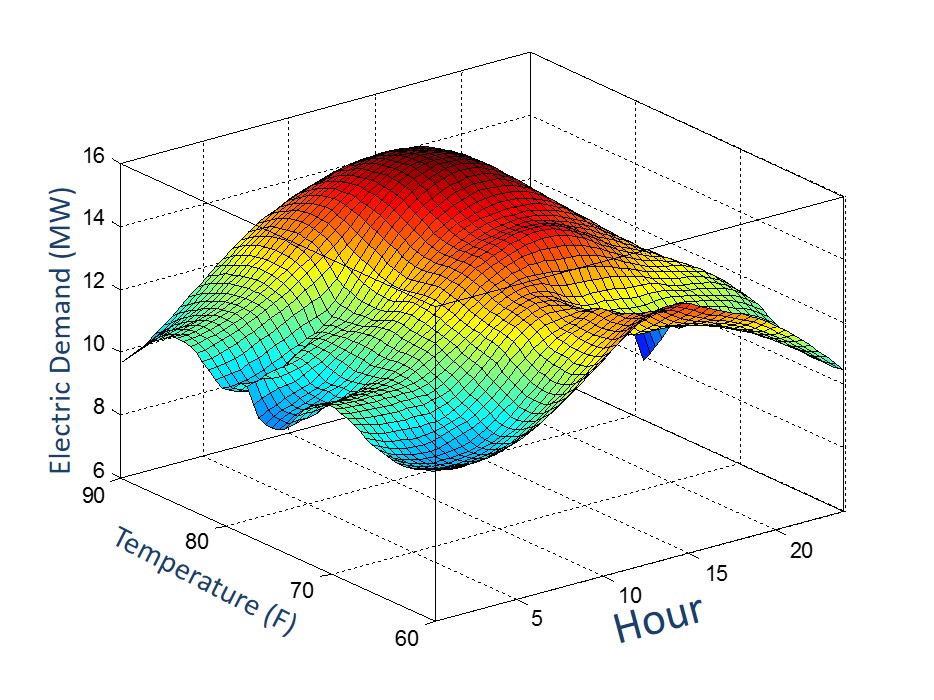


Figure 14 The surface fit made by forecasting electric load from averaging yesterday's and historical loads

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.

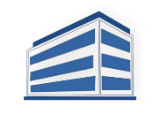
## Dispatch Scheduling

The process of linking the long-term (24-hour) scheduling to the short term control problem requires a number of steps to ensure the best possible plant operation. The hierarchal arrangement envisioned uses complementary quadratic programming described below to do long term forecasting, and a similar mathematical formulation to solve the intermediate-term on/off decision making. A forecast with a probability distribution is given to the long-term optimization. The best-guess and boundary cases are evaluated and any generator start/stops in the next time step are noted. The forecasted SOC at step *t+1* in the best-guess optimization is used as the target condition of the intermediate dispatch. A Newtonian search method finds the optimal time to start/stop a generator. This optimization is repeated until such time as the generator is started/stopped. The output from this continually updated optimization provides the set-point for a low-level MPC applied to each dispatchable generator.

**Forecast**

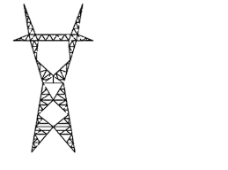
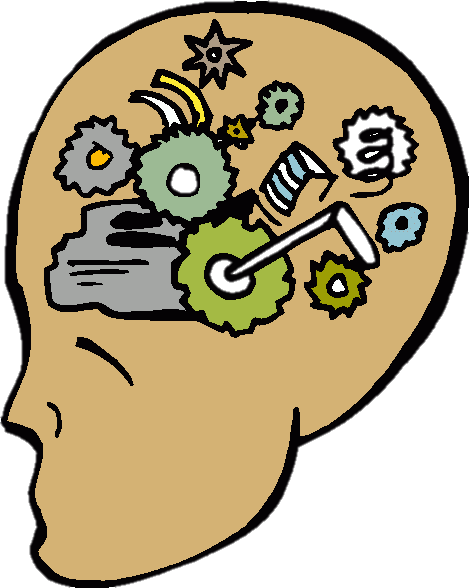
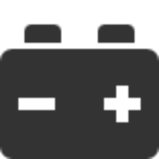
**(Load &prices)**

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



**Economic Dispatch**

(15min)



**ON**



**OFF**

Major Challenges: non-linear cost curve and a discontinuity: zero cost at zero power, finite cost at lower operating condition. This results in a mixed-integer problem to determine on/off, e.g. 2m possible combinations. Taken over a 24 hour horizon this becomes 2m·24 possible combinations!

The following methodology describes a modified dynamic economic dispatch formulation applicable to district energy systems with significant amounts of energy storage. The approach significantly reduces the burden of the mixed-integer unit commitment problem and is applicable to a receding horizon control approach. The cost function to be minimized is defined as:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Where 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 spinning reserve shortfall, *εSR*, represents the difference between a target spinning reserve capacity, and the actual spinning reserve capacity. It is assigned a cost, *F(εSR)*, representing the risk of not meeting the power demand requirements within each interval. 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 approach described greatly reduces or eliminates the mixed-integer aspect of the optimization problem.

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, (2), 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.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1. Where: |  |  |
|  |  |  |

The steep negative slope, *a1*, at zero *SOC* implies a preference to use the most expensive generator before fully depleting the storage. Similarly, the less negative, or possibly positive, slope at full *SOC*, *a1 + a2*, suggests a preference to discharge storage before using the least expensive dispatchable generator. This method does not assign cost or value to stored energy at intermediate time steps, thus ensuring maximizing utilization within the dispatch horizon.

The minimization of (1) is constrained by:

*Energy balance:* for each energy demand category, at each node *j* = 1, 2, 3, …, and every time step, *k* = 1,2,3….

|  |  |  |
| --- | --- | --- |
|  |  |  |

Each energy demand category, e.g. AC power, DC power, heating, cooling, or steam production, has a separate energy balance. At each node, *j*, there is a subset of generators, *Gj*, storage devices, *Sj*, and transmission lines, *Tj*. The power supplied to or extracted from energy storage devices, (6), includes the round-trip energy loss, *ϕr*, (3). The discharging power of the storage system, *Pr*, is calculated from the change in state-of-charge, *SOC*. The charging loss term, *ϕ*, accounts for both charging and discharging losses and is strictly non-negative. The indirect cost of producing additional energy to satisfy the energy balance (5) 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.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

Each transmission line has a unidirectional or bidirectional power transfer, *Pl*, with a proportional line loss, *σ🡨🡪*. The load, *Lj*. determines the net sink of power from the generators and transmission lines at each node. Any uncontrollable power generation, such as rooftop solar PV, is captured in the term *Punctrl j*. The *Ploss* term captures any excess production of heat, cooling, or steam that can be vented. Linear conversion from one energy category to another, e.g. DC power to AC power, is represented as a negative generator in the source energy balance, *-Pi*, and a positive term in the converted energy category, *Pi·β*, where *β* represents the conversion efficiency.

*Capacity constraints* on dispatchable energy systems, energy storage systems, and grid connections respectively.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

*Ramping constraints* on dispatchable energy systems and charging/discharging limits on energy storage systems.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

*Transmission losses* are calculated proportional to the power transferred on a bi-directional power line. Separate strictly non-negative penalties, represented by *σ🡨* and *σ🡪*, are used in the energy balance 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. Unidirectional energy transfer has no transmission loss term in the supplying node, and linearly scales the power in the receiving node, i.e. *(1–η)·Pl*.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

*Spinning reserves* are necessary to ensure robustness of the optimal solution to uncertainty in the forecast. The spinning reserve shortfall, *εSR*, is the target cumulative spinning reserve plus any reserves sold on the ancillary service market, *SRancillary*, and minus the sum of actual spinning reserves. Reserve shortfall is penalized with a quadratic cost when the actual spinning reserves dip below the target. Excess spinning reserve is not penalized by making *εSR* strictly non-negative. The spinning reserve of each generator and storage device is constrained by the ramping and capacity constraints of the device per (14-17). A weighting factor, *wi*, is used based on the average transmission losses between the generator and the load.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
| 1. Where: | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |

*Generator cost functions*, *F(Pi)*, determine the complexity of the optimization problem. The input-to-output conversion efficiency (η) may be a non-linear function of output, depicted in Figure 10A. The standard unit commitment problem inverts efficiency to find the specific cost of generation ($/kWh), which is typically convex, and solves for the appropriate cost of energy that balances supply and demand. Doing so requires multiplying the cost of energy, ($/kWh), by the energy delivered, kWh, which results in the non-convex operating cost ($/hr), shown in Fig. 10A. The unit-commitment problem typically solves for a balanced supply and demand at a single moment in time, and thus must give an equivalant cost to energy drawn from a storage system. Typically cost estimates divide the average cost of energy generation by the round-trip efficiency of the storage system. This fails to account for the degrading round-trip efficiency with the duration of storage. The mixed-integer aspect of the problem arises from the discontinuity between a generator’s minimum operating condition and its off-line state.

It is common practice in optimization approaches to estimate convex functions with a series of linear segments in order to linearize the optimization. The methodology described in this paper optimizes the cost function (1) representing each generator operating cost, *F(Pi)*, with a piecewise convex quadratic function. **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. Limiting the cost functions to convex quadratics enables a gradient based interior-point search method to quickly converge on a global minimum cost for the entire time horizon.

Fit A

Fit B

I

D

UB

Generator Output (kW)

LB

B)

Discontinuity

η

$

kWh

LB

UB

Generator Output (kW)

A)

$/hr

Figure 15 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 is linear from 0 to peak efficiency, D, and quadratic from D to the upper bound, UB. Fit B is discontinuous from 0 to the lower bound, LB, linear from LB to the cost curve inflection point, I, and quadratic from I to UB.

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. 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

*Mixed-Integer sub-optimization:* For complex or highly constrained district energy systems it may be beneficial to check a broader set of feasible operating conditions between the optimizations with **Fit A** and **Fit B**. 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 in (1). Vastly varying equipment sizes may mean that accommodating a large generator may mean turning off one or more smaller units. Each of these circumstances can be accommodated with a reduced mixed-integer problem, of order *N·2G*, introduced between the two successive optimizations of (1) solves the combinatorial problem.

The intermediate step formulates a set of optimizations of (1) at each discrete time step. Each optimization considers a feasible arrangement of the 2m generator combinations available at that time. The cost differential between these feasible alternatives and the original combination of generators is then compared to any start-up costs avoided by changing the on/off schedule from the first optimization of (1).

When optimizing a single time step, energy storage lacks the ‘big-picture’ perspective of the simultaneous optimization. This perspective is incorporated by using the *SOC* determined by the first optimization as the nominal target. Deviations from this nominal *SOC* are penalized with a quadratic cost, ensuring the storage profile remains similar to the first optimization, while accommodating changes in generator ramping and boundary constraints.

*Combined heating and power generators, CHP:* produce output appearing in two energy balances (3), for a single input feedstock. The secondary heating output does not alter the cost of the generator, and must be linearly proportional to the primary output, i.e. *Pi·β*. This may over or under represent actual heat co-production in the case of a partially loaded CHP unit. Generally, there is a greater tolerance for variance in heating than electricity. Thus, an increase or decrease in demand during the subsequent forecast optimization accommodates any deviation in heat supplied.

Electric chillers typically represent a non-linear conversion of electrical power to cooling power not captured by a constant coefficient of performance, COP. Without cold thermal storage, there is little flexibility in meeting the thermal demand, and chillers are often run at non-optimal performance. In this scenario, it is preferable to first optimize the chiller dispatch independent of other systems, where the linear and quadratic cost terms represents the non-linear electric power consumption, then add the resulting electric demand to net electric load and proceed with the optimization of the remaining energy systems.

With cold energy storage, it becomes feasible to use chiller loads to balance the electric demand, and thus the dispatch all systems concurrently. In this scenario, the chillers have no direct costs, i.e. *F(Pi) = 0*. They appear in the electric energy balance as a load, *-Pi*, and in the cooling energy balance as a generator, *Pi ·COP*. Given the flexibility in dispatch afforded by the thermal storage, it is generally preferable to operate all chillers at their design condition, thus justifying the assumption of constant COP.

*Hydroelectric generation, water reservoirs and river systems:* represent a similar stock and flow problem 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 16 Conceptual model of hydroelectric dam/water reservior

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*.

|  |  |  |
| --- | --- | --- |
|  |  |  |

*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.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

*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.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Electric Networks, District Energy Systems and River Systems

Equation (5) described an energy balance in terms of multiple ‘nodes’ with electric or other energy transmission ‘lines’ connecting the nodes. This framework for describing an energy network is similar to that of any stock and flow model, including that multiple networks of different species can be overlaid with specific nodes providing the interconnection where some energy conversion occurs. Additionally adjacent nodes may be aggregated to reduce the system complexity. Let’s take the example of the Columbia Basin which overlays a river network and an electric network.

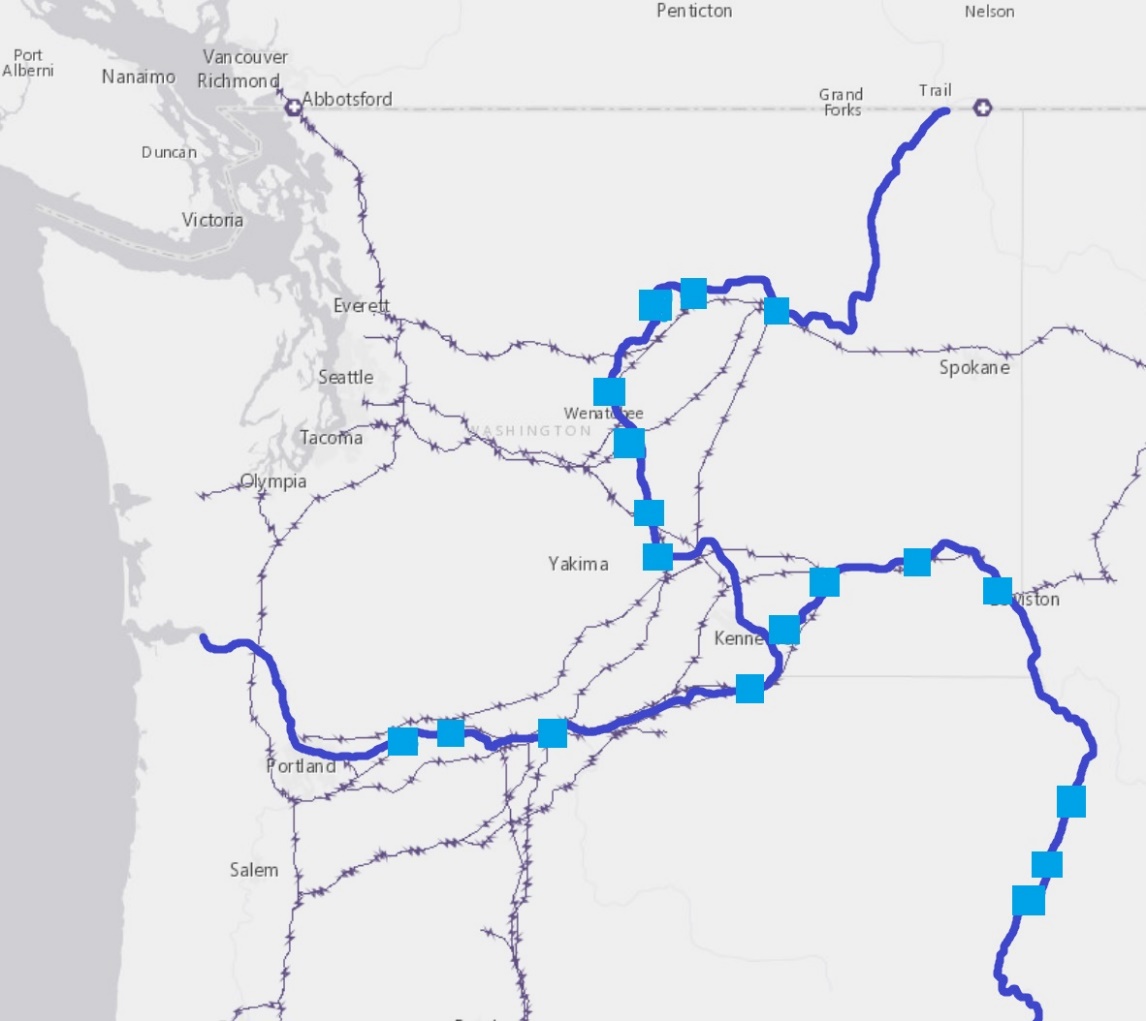


Figure 17 Overlay of the primary transmission line (>345kV), the Columbia and Snake Rivers with their respective hydroelectric dams

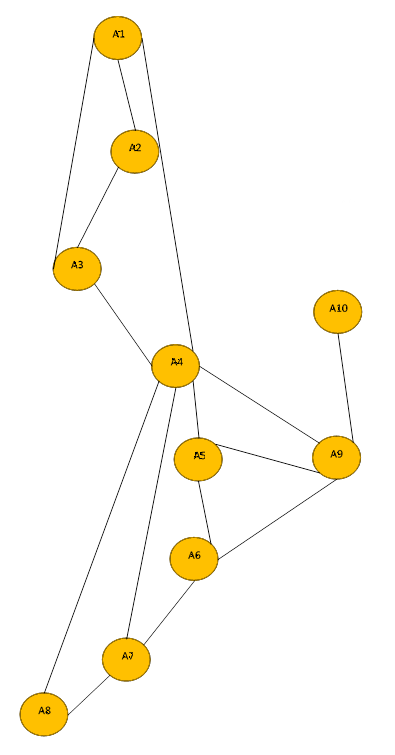
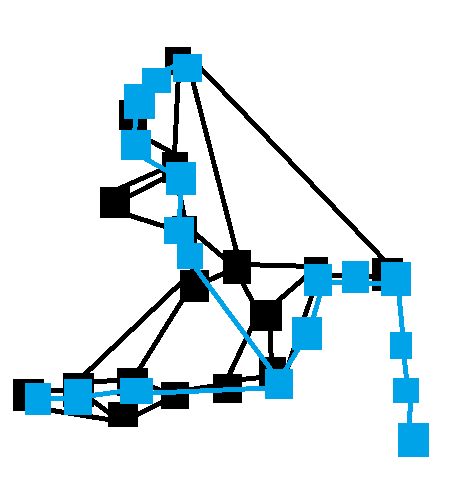
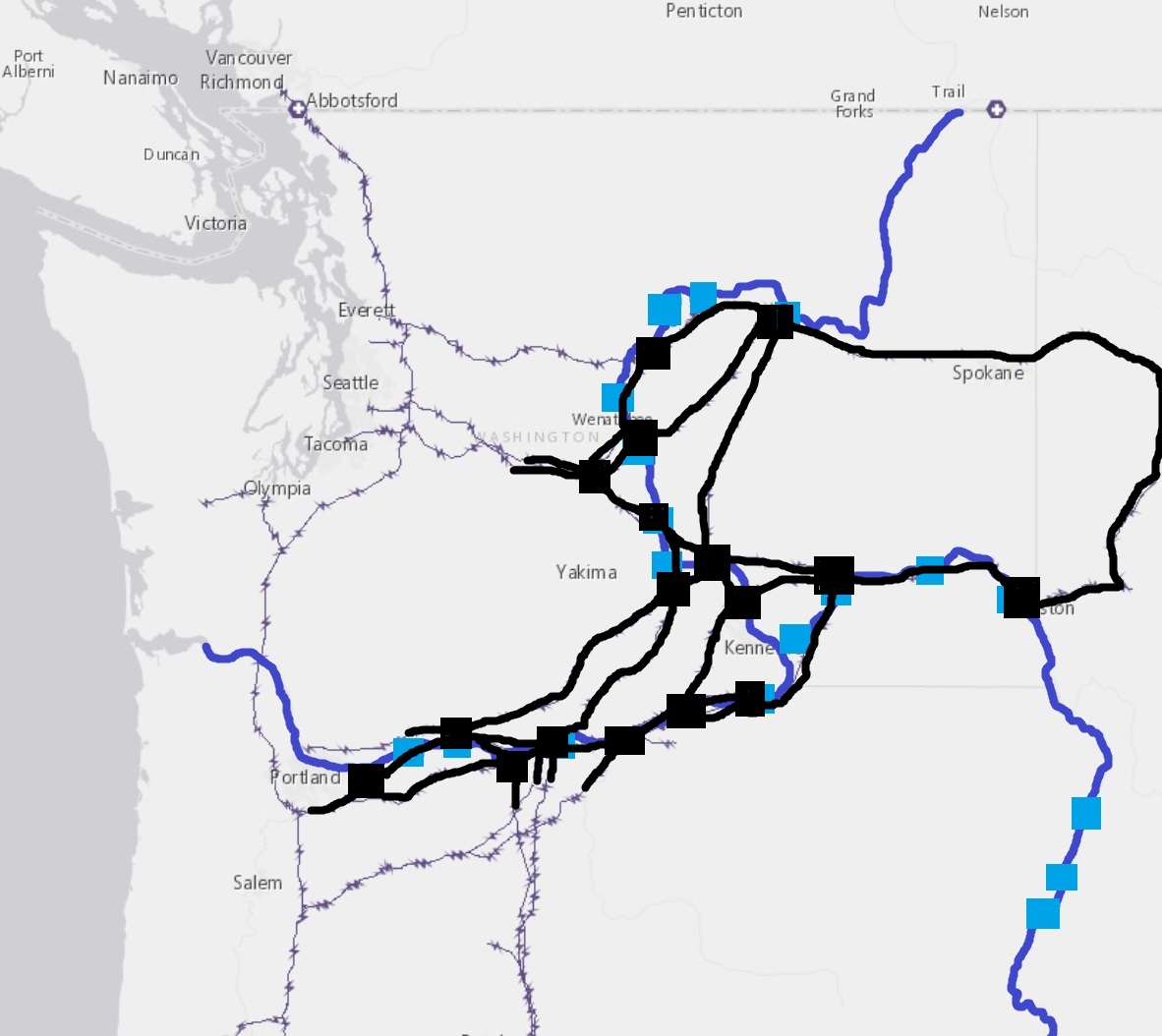


Figure 18 a) Overlay of a 17-node electric network (black) with an 18 dam river network (blue), b) a simplified visual representation of the network, c) reduced order 10-node network

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.

## Component 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.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

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 10 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 19 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:

This model captures a few high temperature fuel cell configurations shown in Figure 11, Figure 13, and Figure 14. 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 11. 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 12 illustrates how this thermally coupled arrangement pre-reforms the fuel within the stack, but not within the anode.

Blower

Oxidizer

**Fuel**

Cathode

Anode

Figure 20 SOFC with anode recirculation and internal reforming

**H2O**

**O2, N2**



**heat**

**CH4**

**+**

**-**

**H2O,H2,CO2**

**CO, H2**

**H2**

**CO2**

Figure 21 Arrangement of "internal" fuel reformer

Selecting the ‘external’ reforming option results in the arrangement of Figure 13 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 22 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 14. The anode recirculation provides the humidity and energy to reform a portion of the incoming fuel.

Blower

Oxidizer

**Fuel**

Cathode

Anode

Figure 23 SOFC with anode recirculation and adiabatic reforming

### Example System Model oxySOFC:

**H2O**

**O2**



**heat**

**CH4**

**+**

**-**

**H2O,H2,CO2**

**CO, H2**

**H2**

**CO2**

### Example System model rSOFC:

H2O

O2

**SOFC**

**WGS / H2 recovery**

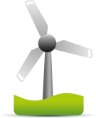
H2O

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

### Electrolyzer

### FuelCell



### HeatExchanger

### LeakageValve

### MixingVolume

### Oxidizer

### Reformer

### Shaft

### SimpleMix

### Source

### Turbine

### Valve3Way

### ZonalBuilding

## Description of Specific Models

### SOFCstack

### SOFCsystem

### SOECstack

### GasTurbine

### OxyFC

### MCFCsystem

# Glossary of Important Variables

Global variable Plant contains all information on the generators, demand, (we will add details of grid i.e. network constraints & losses here) and dispatch results.

## Project Variables

Plant

Plant.Data

Plant.Building

Plant.Generator

Plant.Network

Plant.subNet

Plant.Plotting

Plant.Cost

Plant.subNet

Plant.OpMatA

Plant.OpMatB

Plant.OneStep

Plant.Online

Plant.Design

Plant.Dispatch

Plant.Predicted

Plant.RunData

Plant.Baseline

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 | ? |  |  |

## 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 |