Efficient Allocation of Grid Energy Resources including Storage (EAGERS)

Economic dispatch for micro-grids presents a highly constrained non-linear optimization problem. A common issue faced by search or gradient based optimization algorithms is the discontinuity between a generator’s minimum operating condition and its off-line state, resulting in a non-continuous solution space with many local cost minima. This challenge is compounded by the dispatch flexibility afforded by energy storage technologies, which require simultaneous optimization over the energy storage horizon. This document outlines an open-source platform, EAGERS developed to address these challenges using complementary convex quadratic optimizations. The approach applies to grid-connected or islanded micro-grids comprised of any variety of electric or combined heat and power generators, electric chillers, heaters, and all varieties of energy storage systems. It incorporates constraints for generator operating bounds, ramping limitations, and energy storage inefficiencies. Please refer to the appendix for a detailed explanation of the optimization strategy.

Table of Contents

[II. Overview 2](#_Toc479236385)

[i. Getting Started 2](#_Toc479236386)

[ii. Starting a New Project 2](#_Toc479236387)

[III. Planning Tool 2](#_Toc479236388)

[i. The Planning Tool Interface 2](#_Toc479236389)

[ii. Component Selection and Sizing 2](#_Toc479236390)

[iii. Automated Optimization of Plant Arrangement and Sizing 2](#_Toc479236391)

[IV. Control Tool 2](#_Toc479236392)

[i. The Control Tool Interface 2](#_Toc479236393)

[a. Optimization Options 2](#_Toc479236394)

[ii. Forecasting 2](#_Toc479236395)

[a. Historical Surface Fits 2](#_Toc479236396)

[iii. Dispatch Scheduling 2](#_Toc479236397)

[iv. District Energy Systems and Electric Networks 2](#_Toc479236398)

[v. Component Descriptions 2](#_Toc479236399)

[a. Utilities: Electric, Gas, District Heating or Cooling 2](#_Toc479236400)

[b. Generators, Chillers, Boilers, and CHP systems 2](#_Toc479236401)

[c. Energy storage systems 2](#_Toc479236402)

[d. Hydro Power 2](#_Toc479236403)

[e. Wind and Solar 2](#_Toc479236404)

[f. HVAC 2](#_Toc479236405)

[vi. Real-time Model Predictive Control 2](#_Toc479236406)

[V. Simulation Tool (STRIDES) 2](#_Toc479236407)

[i. System Model Files 2](#_Toc479236408)

[ii. Component Functions 2](#_Toc479236409)

[iii. Controller Functions 2](#_Toc479236410)

[iv. Initialization and Simulation Functions **Error! Bookmark not defined.**](#_Toc479236411)

[v. Additional Reference Functions 2](#_Toc479236412)

[VI. Glossary of Variables 2](#_Toc479236413)

[i. Project Variables 2](#_Toc479236414)

[ii. Planning Tool Variables 2](#_Toc479236415)

[iii. Control Tool Variables 2](#_Toc479236416)

[iv. Simulation Tool Variables 2](#_Toc479236417)

[VII. Interface Flow Diagrams 2](#_Toc479236418)

[v. 1. EAGERS Interface 2](#_Toc479236419)

[a. Opening EAGERS 2](#_Toc479236420)

[VIII. Appendix 2](#_Toc479236421)

# 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.

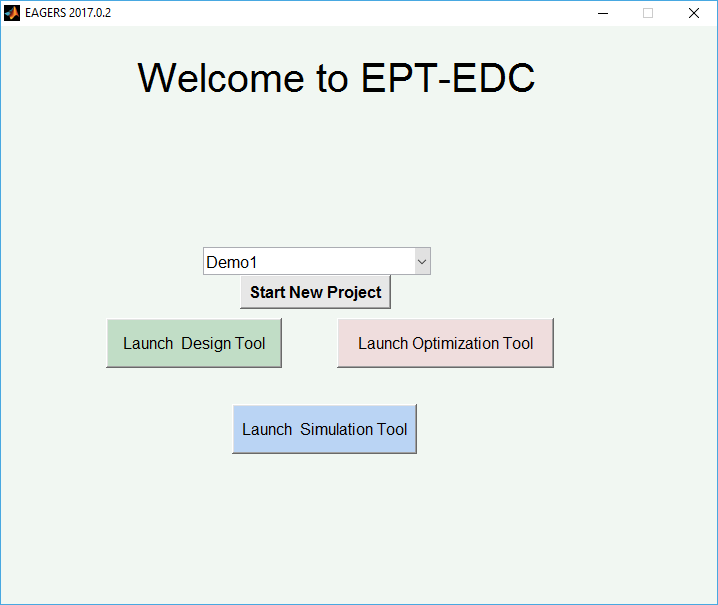


Figure 1 EAGERS welcome screen

## 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

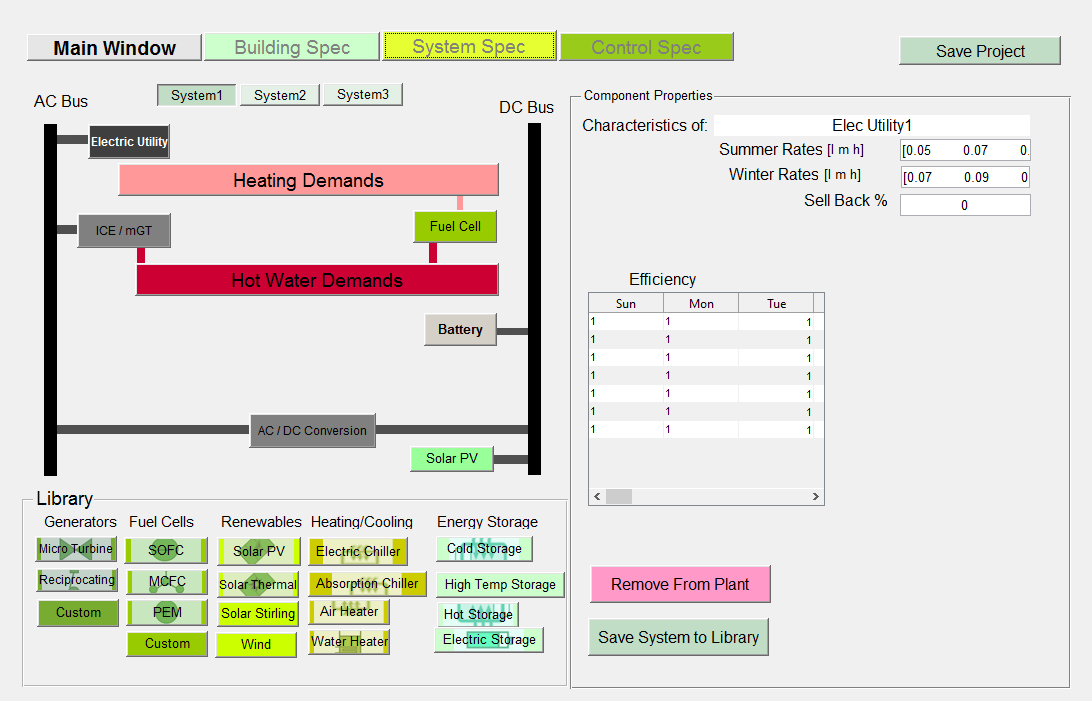


Figure 3 System specification tab

## 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

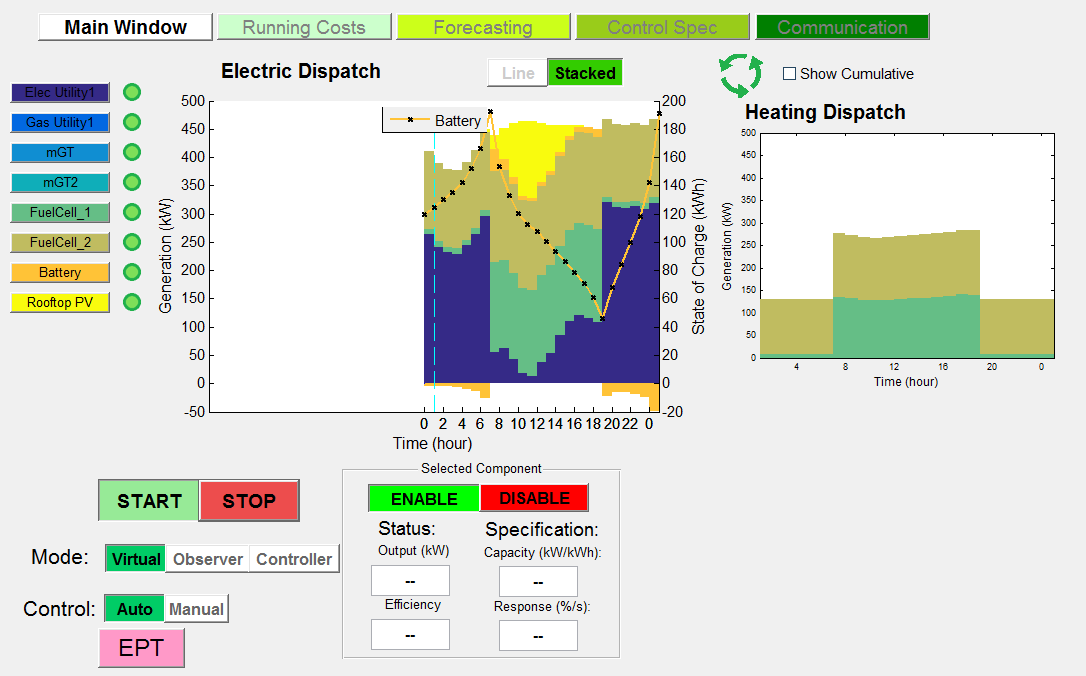


Figure 5 Main control GUI window

### Optimization Options

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

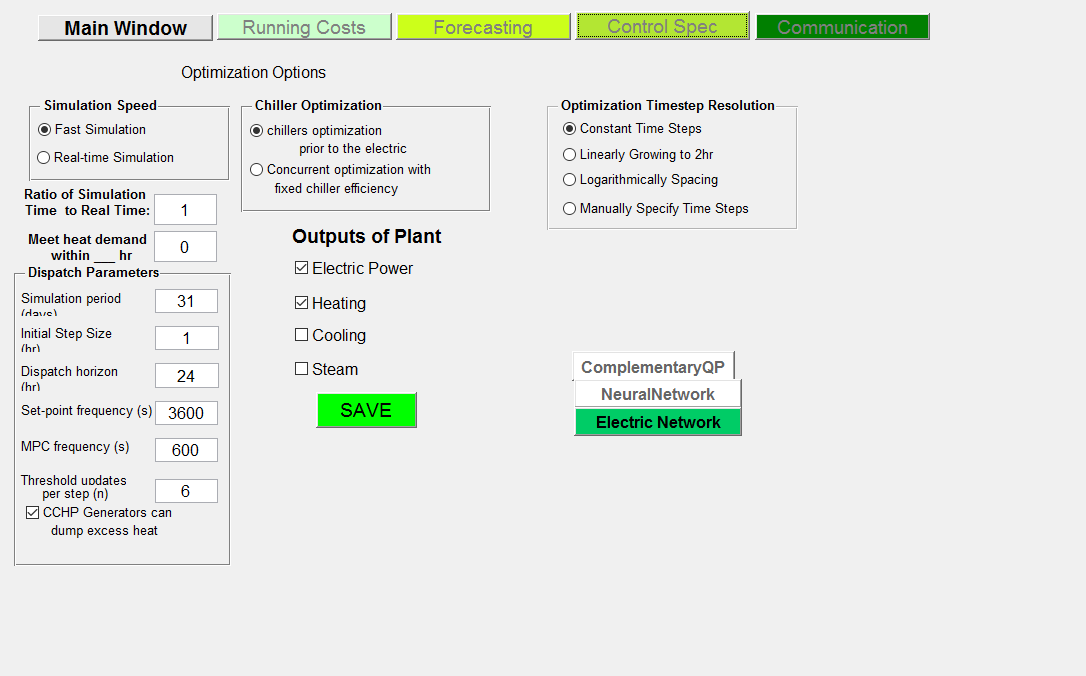


Figure 6 Optimization specification GUI

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

### Simple Exponential Smoothing

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

### Neural Network:

### 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.

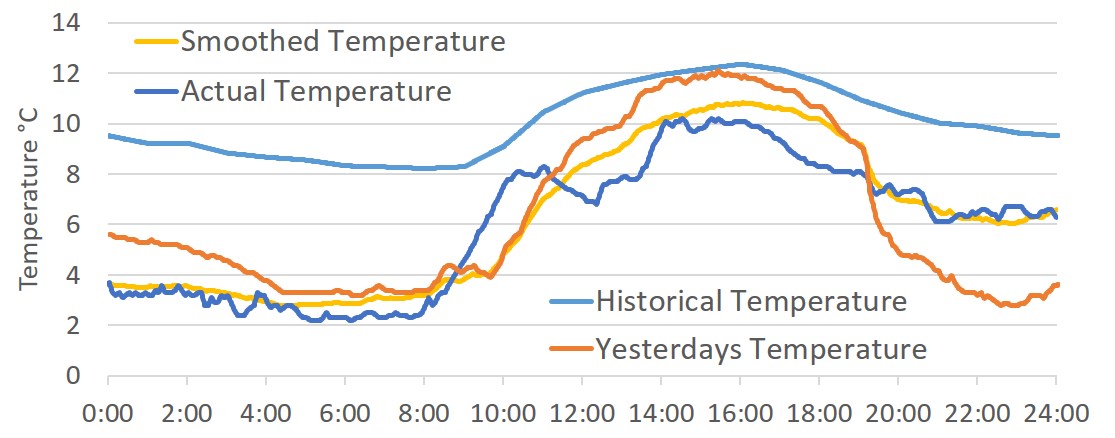


Figure 7 The temperature from yesterday has been averaged with the historical, then smoothed to form a prediction. This prediction is then compared to the actual temperature.

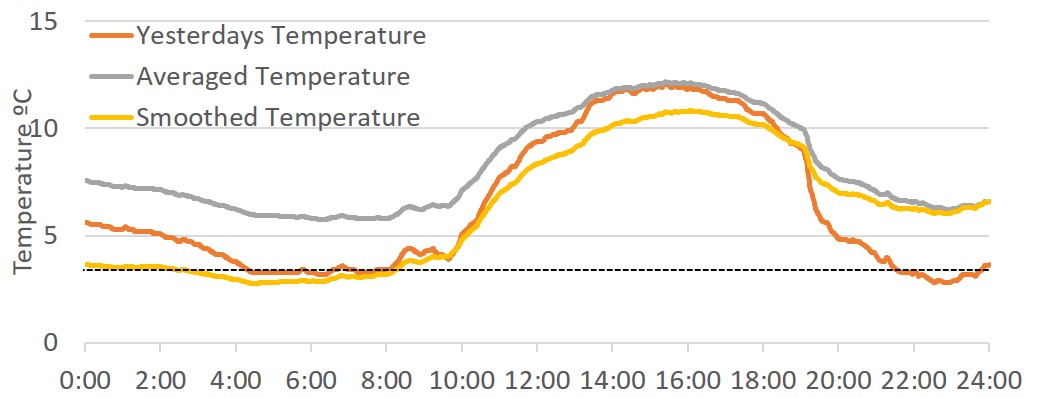


Figure 8 The last recorded temperature from yesterday is used to the determine the base for the smoothed temperature, which is then fit with the average of yesterday and the historical.

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.

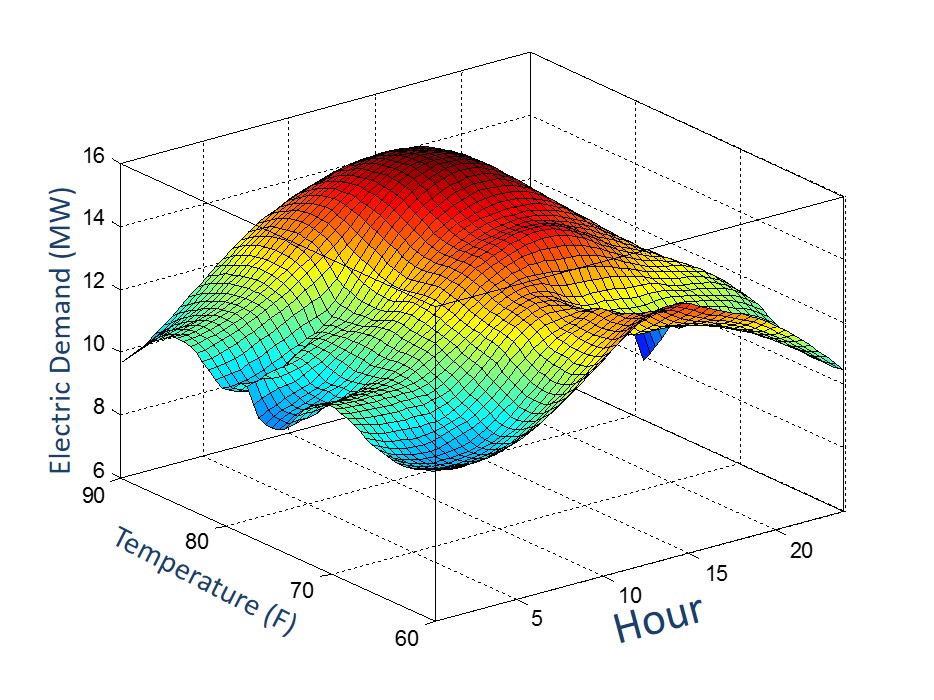


Figure 9 The surface fit made by forecasting electric load from averaging yesterday's and historical loads

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.

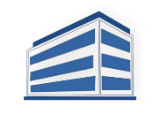
## Dispatch Scheduling

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**

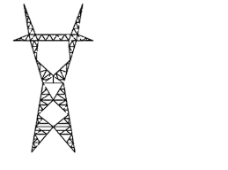
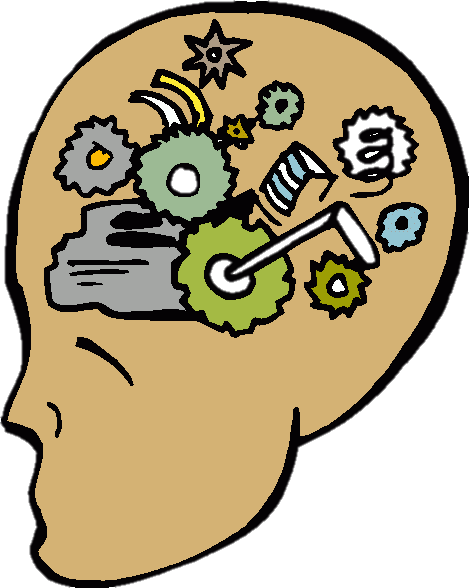
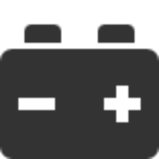
**(Load &prices)**

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



**Economic Dispatch**

(15min)

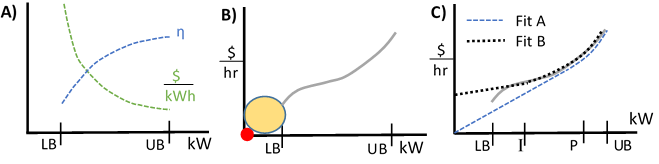
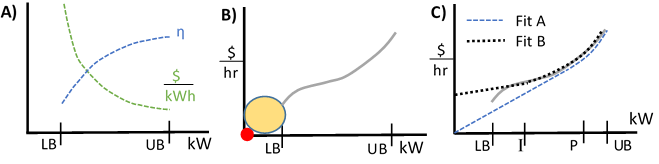


**ON**



**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!



cQP avoids the exponential increase in on/off states by breaking the problem into 3 steps using two different piecewise quadratic convex fits of the generator cost curves; one with a zero y-intercept, the other with a non-zero y-intercept.

Step 1: Solve the entire optimization with all constraints except the lower boundary constraint (i.e. *Ph,kmin=0* and *Fh,k(0)=0*) and use **fit A**.

Step 2: Solve each time step in sequence. Solve the combinatorial problem to find the optimal set of generators / set-points for each time period. Use **fit B** and enforce the lower boundary constraints, but not the ramping constraints. Deviations from the SOC profile of step 1 are penalized. The SOC at t-1 is updated to reflect the best solution of the previous time step.

Step 3: Optimize the entire horizon once again using the schedule (on/off) determined by step 2. All constraints, i.e. both the lower bound and ramping constraints, are enforced. The more accurate **fit B** is used. This step determines set-points for energy storage throughout the horizon.

The mathematical optimization solves the convex quadratic minimization of the total cost of operation over the forecast horizon:

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

Where there are *N* time steps: *k* = 1, 2, 3,…

Where there are *G* dispatchable generators whose cost is a function of their power output, *Ph,k*. The costs being the best possible piecewise convex quadratic fit of the generator cost calculated from the energy conversion efficiency curve.

Where there are *S* storage devices with a value assigned to the final state-of-charge, SOC. The final value is quadratic such that the first kWh of storage is valued slightly less than the lowest marginal cost dispatchable generation, and the last kWh of storage is valued less than the smallest marginal cost of generation.

The minimization of (1) is constrained by:

1. *Energy balance:* at each node *j* = 1, 2, 3, … a separate balance for each type of demand, i.e. electric, heating, cooling…

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

Where at each node, *j*, there is a subset of generators, *Gj*, a subset of storage devices, *Sj*, with round-trip energy loss *ϕr,k*, and a subset of the transmission line or water pipes, *Tj*, with power transfer *Pl,k*. Line losses proportional to the power transfer are captured in *σl,k*. The load at each node at each time step is specified by *Lj,k*. Any uncontrollable power generation, such as rooftop solar PV, are captured in the term *Punctrl j,k*. A separate energy balance is computed for each type of demand, i.e. AC power, DC power, heat, cooling, or steam.

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

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

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

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

1. *Charging/discharging losses* for storage systems are accounted for when the system is charging. The SOC in the equations represents the actual SOC scaled by the discharging efficiency. This way a reduction in SOC is directly proportional to the usable energy extracted during discharge, and the charging loss term, ϕ, accounts for both losses when putting energy back into the storage system. The storage penalty is non-negative. The indirect cost of producing additional energy to satisfy the energy balance (2) ensures this charging loss is always proportional to the charging power.

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

1. *Transmission losses* are calculated proportional to the power transferred on a line. If bi-directional power is possible, there are separate penalties, σ, for the transfer from node A 🡪 B and from B 🡪 A. Both penalties are greater than or equal to zero. Similar to the charging loss, the indirect cost of additional energy ensures the proportionality with the line transfer power.

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

1. *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. An additional state is needed at each time step for each active generator and available storage device. A weighting factor, *wi*, is used based on the average transmission losses between the generator and the load.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
| Where: | |  |  |
|  | |  |  |
|  | |  |  |
|  | |  |  |

1. *Non-Spinning reserves* are necessary to ensure robustness of the optimal solution to uncertainty in the forecast. The difference between the target cumulative spinning reserve and the sum of actual spinning reserves is penalized with a quadratic cost. An additional state is needed at each time step for each active generator and available storage device. A weighting factor is used based on the average transmission losses between the generator and the load.

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

1. *Congestion penalties* may be implemented to lower the risk of overloading a single transmission line. These are implemented in a similar fashion to spinning reserves with a penalty on the difference between the desired and actual excess line capacity.

## District Energy Systems and Electric Networks

## Component Descriptions

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

When editing an electric utility, the user will be faced with 4 options to adjust the way the utility is being perceived by the program. The first option is a ‘Peak/Off-Peak Rate’ which, when selected, presents the user with 4 inputs that further describe the rates. The user can also choose to load 15 minutes of data or load an existing utility, as well as modify the structure.

When modifying or setting up an electric utility, you will be prompted to further describe the unit, such as naming the unit and identifying key features. 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. 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. On the left of the interface are two identical tables; one for winter and one for summer. These tables identify what hour and day the utility experiences a peak (3), partial peak (2) and when its off-peak (1). Each of these three intervals is related to both the energy charge and the demand charge. The user can define the peaks however they see fit.

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

The generator setup menu is applied to both electric and CHP generators, as well as a variety of other components that behave similarly to generators in the dispatch. These other components include boilers, chillers, and heaters. When modifying a generator, a window will appear with several different aspects to adjust to better define what role the “generator” plays in the microgrid. On the edit screen, the user has the opportunity to ‘Specify Communication Ports’. This feature will allow the user to further describe the communication from the generator to a desired controller. The output type in combination with the energy source are used to best identify what exactly the component is within the microgrid. Several output types can be selected, but only one energy source can be chosen. For example, a CHP generator would output electricity and heat, while using natural gas as the energy source. To better navigate through these options, the flowchart for generators could be utilized.

### 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.

### HVAC

The Heating, Ventilation, and Air Conditioning (HVAC) system for the microgrid can be classified as one of three different types: ‘Smart Air Conditioning Only’, ‘Smart Heating Only’ or ‘Smart AC and Heat’. A warning is displayed at the bottom right of the setup menu to remind the user that a chiller and/or heater must be used to simulate the smart HVAC system in the grid. The size of the building the HVAC is working within needs to specified, as well as the amount of energy displaced by the system to better evaluate the way the HVAC operates as a component of the microgrid.

## 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. 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 10 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 11 SOFC with anode recirculation and internal reforming

**H2O**

**O2, N2**



**heat**

**CH4**

**+**

**-**

**H2O,H2,CO2**

**CO, H2**

**H2**

**CO2**

Figure 12 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 13 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 14 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**

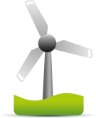
H2O

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. There must be two separate functions, one for initialization, one for operation. The name of the operation function defines the “type” used in the model function. For example, *function Out = FuelCell(t,Y, Inlet,block,string1)* is the operation function for the FuelCell type component. The corresponding initialization function must append “initialize” to the front of the function name, i.e. *function block = FuelCell(varargin)*.

### Operation function arrangement

Each operation function must accept the inputs *t*, *Y*, *Inlet*, *block*, and *string1*. 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. The function arrangement must include:

global Tags

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

### Initialization function arrangement

block = varargin{1};

if length(varargin)==1

* 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*. 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

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*.

end

## Controller Functions

Controller functions have all of the same requirements and structure as component functions. Ther must be an operational and initializing control function with the structure described previously. One not of difference is that the controller is initialized last, 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

### Blower

### Capacitor

### Combustor

### Compressor

### DCDCConverter

### Electrolyzer

### FuelCell



### HeatExchanger

### LeakageValve

### MixingVolume

### Oxidizer

### Reformer

### Shaft

### SimpleMix

### Source

### Turbine

### Valve3Way

## 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.GUIhandles

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.OpMatA: This structure helps in the construction of the optimization matrices associated with the multi-time-step quadratic programming where it is unknown which generators are on or off, FitA, and is detailed below.

Generator.OpMatB: This structure helps in the construction of the optimization matrices associated with the multi-time-step quadratic programming where it is known which generators are on or off, FitB, and is similar to what is detailed for OpMatA below. The difference is that the number of states is likely reduced.

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

OpMatA.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.

OpMatA.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.

OpMatA.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.

OpMatA.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.

OpMatA.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.

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

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

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

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

List of constraints:

|  |  |  |  |
| --- | --- | --- | --- |
| Type of output | Electrical output | OpMatA.X.output.electricity | [] or 1 |
|  | Heat output | OpMatA.X.output.heat | [] or 1 or Hratio |
|  | Cooling output | OpMatA.X.output.cooling | [] or 1 or Cratio |
| Ramp Rates | Ramp up & down | OpMatA.X.Ramp.A | [-1,1;1,-1;] |
|  |  | OpMatA.X.Ramp.b | [rampupvalue, (-rampdownvalue)] |
| State splitting | How do the different states relate | OpMatA.link.eq | For a gen [1,-1,-1]  For grid [1,-1]  For storage [1,-1/roundtrip efficiency] |
|  |  | OpMatA.link.beq | 0 |
| Boundaries | Upper bound | OpMatA.X.ub | Generators(i).Size, inf, usablesize |
|  | Lower Bound | OpMatA.X.lb | Usually 0 for OpMatA and LB for OpMatB |
| 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

# Appendix