controller_usage_demonstration

May 11, 2017

This Jupyter Notebook demonstrates how to enable the GridBallast controller for a load (in this case a water heater) by feeding a stored grid frequency time series contained in an external file to the simulator.

To run this notebook, please make sure you are in a UNIX based environment and have all the necessary python packages installed (plotly, matplotlib, numpy, pandas).

```
In [1]: !ls
```

```
controller_usage_demonstration.ipynb smSingle_base.glm
controller_usage_demonstration.pdf   smSingle_lenient_freq.glm
fan1.csv                             smSingle_strict_freq.glm
fan2.csv                             smSingle_strict_freq_jitter.glm
frequency.PLAYER                     wh1_base.csv
hot_water_demand.glm                 wh1_lenient_freq.csv
local_gd                             wh1_strict_freq.csv
smSingle.glm                         wh1_strict_freq_jitter.csv
```

We need to re-configure the path for GridLAB-D such that the binary can locate the path of the library. If you are using macOS, please make sure you have installed the GNU version of sed, e.g.,

```
brew install gnu-sed
```

```
In [2]: %%bash
        oldpath='/tmp/temp'
        newpath=`pwd`'/local_gd'
        gsed -i "s#$oldpath#$newpath#" local_gd/bin/gridlabd
        gsed -i "s#$oldpath#$newpath#" local_gd/lib/gridlabd/glxengine.la
```

```
In [3]: !local_gd/bin/gridlabd --version
```

```
GridLAB-D 4.0.0-17296 (feature/730:17296) 64-bit MACOSX RELEASE
```

The above listed **local_gd/bin/gridlabd** is the binary version of the gridlab-d software with controlling functionality. In addition to that, we have **.glm** files and generated **.csv** files. We also have a **frequency.PLAYER** containing the 1-second resolution frequency information.

The version of the gridlab-d binary file and the content of the frequency.PLAYER can be seen below.

If the version of the gridlab-d does not work, we can disable the comments below and run the command to compile the source and install the gridlab-d to the machine.

```
In [4]: # %%bash
        # cd ~
        # git clone -b feature/730 https://github.com/jingkungao/gridlab-d.git
        # cd gridlab-d
        # cd third_party
        # chmod +x install_xercesc
        # . install_xercesc
        # tar -xvf cppunit-1.12.0.tar.gz
        # cd cppunit-1.12.0
        # ./configure LDFLAGS="-ldl"
        # make
        # sudo make install
        # cd ../..
        # autoreconf -isf
        # ./configure
        # make
        # sudo make install
```

```
In [5]: # !head -10 frequency.PLAYER
```

We can further plot the frequency data to get a better sense of it.

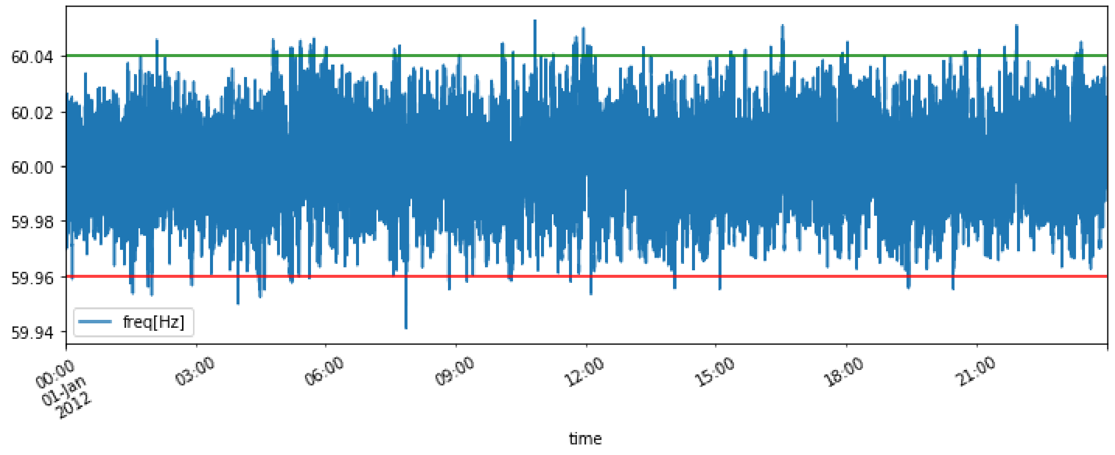
```
In [6]: # install necessary packages if the system does not have them
        !pip3 install numpy
        !pip3 install pandas
        !pip3 install plotly
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: pandas in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: python-dateutil>=2 in /usr/local/lib/python3.6/site-packages (fro
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/site-packages (from panda
Requirement already satisfied: numpy>=1.7.0 in /usr/local/lib/python3.6/site-packages (from pand
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/site-packages (from python-d
Requirement already satisfied: plotly in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: requests in /usr/local/lib/python3.6/site-packages (from plotly)
Requirement already satisfied: decorator in /usr/local/lib/python3.6/site-packages (from plotly)
Requirement already satisfied: six in /usr/local/lib/python3.6/site-packages (from plotly)
Requirement already satisfied: pytz in /usr/local/lib/python3.6/site-packages (from plotly)
Requirement already satisfied: nbformat>=4.2 in /usr/local/lib/python3.6/site-packages (from plo
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.6/site-packages (from nbfo
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/site-packages (from 
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /usr/local/lib/python3.6/site-packages
Requirement already satisfied: traitlets>=4.1 in /usr/local/lib/python3.6/site-packages (from nb
```

```
In [7]: %matplotlib inline

        import numpy as np
        import pandas as pd
```

```
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly.graph_objs as go
init_notebook_mode(connected=True)

raw_freq = pd.read_csv('frequency.PLAYER',index_col=0,names=['time','freq[Hz]'],
                       parse_dates=True,infer_datetime_format=True)

freq_low = 59.96
freq_high = 60.04

ax = raw_freq.plot(figsize=(12,4),rot=30)
ax.axhline(y=freq_low, c='red')
ax.axhline(y=freq_high, c='green')
```

```
Out[7]: <matplotlib.lines.Line2D at 0x11134ae48>
```



Next, we will run **local_gd/bin/gridlabd** on different **.glm** files and plot the outputs showing the difference with and without controllers.

We start with running **smSingle_base.glm**, which is almost same as the original **smSingle.glm** provided by NRECA to us with the main difference being that we changed the simulation clock and added a recoreder for waterheater1 at the end.

1 Base case

We begin with the same circuit provided by NRECA (smSingle.glm), and modify it slightly as follows:

- We change the simulation time to match the time of **frequency.PLAYER** and add a recorder to record the waterheater measurements. Note that we record data for waterheater1 as an example but it could be used for any waterheater.
- We also set the timestep to 1 second instead of 60 seconds.

- For a more realistic water draw schedule, we include a **hot_water_demand.glm** which exhibits typical the weekday and weekend water demand usage patterns.

Below we illustrate some of those changes made to the glm file:

```
In [8]: # from 2012-01-01 to 2012-01-02
        !head -9 smSingle_base.glm
```

```
clock {
        timezone PST+8PDT;
        starttime '2012-01-01 00:00:00';
        stoptime '2012-01-02 00:00:00';
};

#include "hot_water_demand.glm";

#set minimum_timestep=1;
```

```
In [9]: # record data for waterheater1 at 1s resolution
        !tail -8 smSingle_base.glm
```

```


object recorder {
        interval 1;
        property measured_frequency,temperature,actual_load,is_waterheater_on,water_demand;
                // current_tank_status,waterheater_model,heatgain,power_state;
        file wh1_base.csv;
        parent waterheater1;
};
```

We are now ready to run a simulation with the base case (no control).

```
In [10]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_base.glm
```

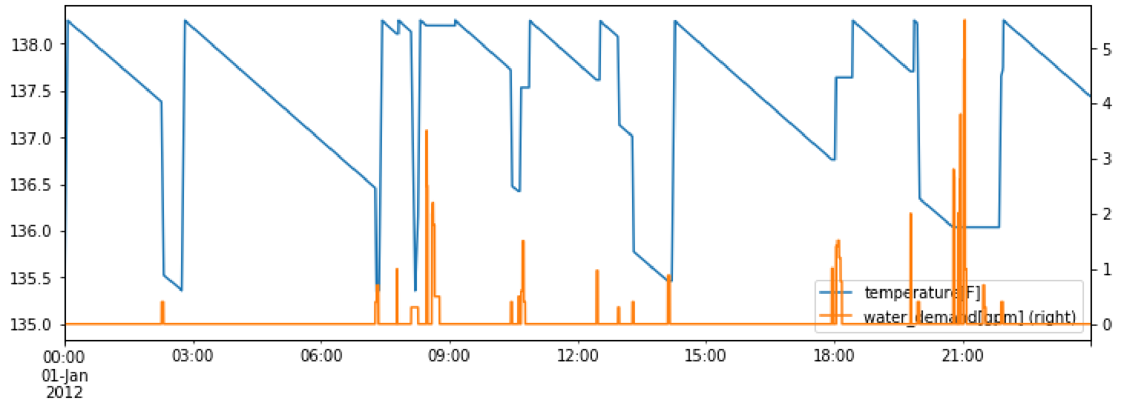
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 35 objects
Parallelism                    1 thread
Total time                  18.0 seconds
  Core time                  2.4 seconds (13.5%)
    Compiler                 1.0 seconds (5.8%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.9%)
  Model time                15.6 seconds/thread (86.5%)
Simulation time                1 days
Simulation speed              47 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate            4800 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node                9.422    60.5%   4711.0
triplex_meter       0.941     6.0%    313.7
recorder            0.802     5.2%    267.3
house               0.800     5.1%    400.0
ZIPload             0.770     4.9%     96.2
transformer         0.684     4.4%    342.0
waterheater         0.650     4.2%    325.0
triplex_line        0.554     3.6%    277.0
regulator           0.372     2.4%    372.0
triplex_node        0.311     2.0%    311.0
auction             0.168     1.1%    168.0
climate             0.093     0.6%     93.0
================ ======== ======== ========
Total              15.567   100.0%    444.8

WARNING  [2012-01-02 00:00:00 PST] : last warning message was repeated 1 times
```

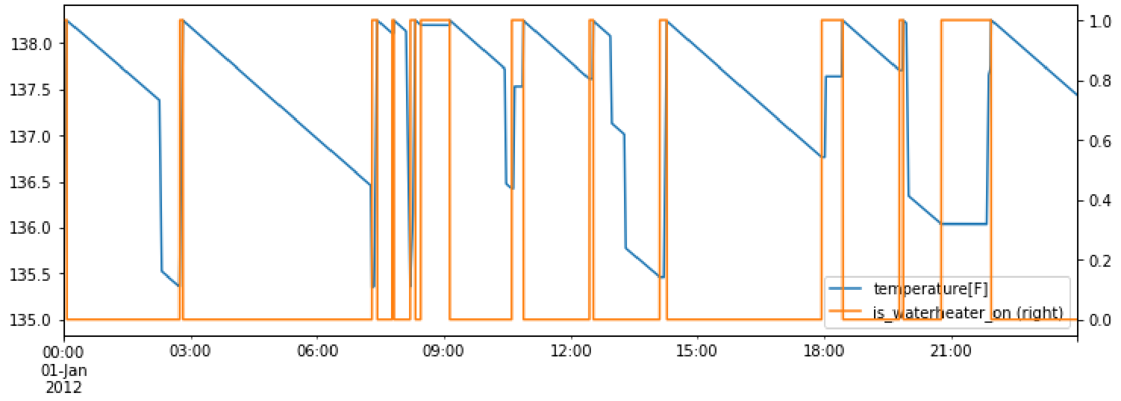
Now, we plot the generated waterheater data stored in **wh1_base.csv** from the simulation.

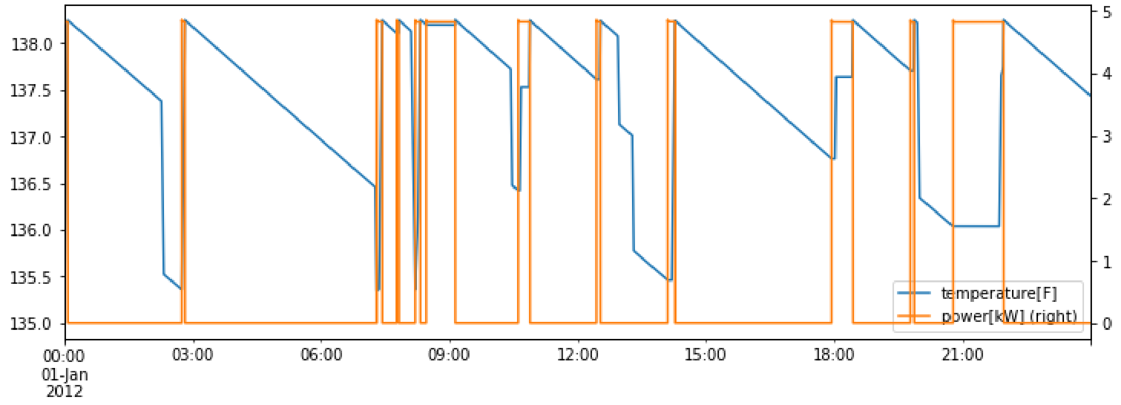
```
In [11]: df_base = pd.read_csv('wh1_base.csv',sep=',',header=8,
                               index_col=0,parse_dates=True,infer_datetime_format=True,
                               names=['freq[Hz]','temperature[F]','power[kW]','is_waterheater_on',
                                       'water_demand[gpm]'])

         df_base[['temperature[F]', 'water_demand[gpm]']].plot(figsize=(12,4),
                                                             secondary_y='water_demand[gpm]')
```

```
        df_base[['temperature[F]','is_waterheater_on']].plot(figsize=(12,4),
                                                      secondary_y='is_waterheater_on')

        df_base[['temperature[F]','power[kW]']].plot(figsize=(12,4),
                                                      secondary_y='power[kW]')
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x11782b470>
```







```
In [12]: # We can also plot the interactive version of the plot during certain period
         def plotly_plotdf(df,title='Interactive plot of column variables'):
             if len(df)>20000:
                 print('Too many points, please reduce number of points! (less than 20000)')
                 return
             data = []
             for i in df.columns:
                 trace = go.Scatter(
                     name = i,
                     x = df.index,
                     y = df[i]
                 )
                 data.append(trace)
             fig = go.Figure(
                 data = data,
                 layout = go.Layout(showlegend=True,
                                   title=title)
             )
             iplot(fig)
```

```
In [13]: # we can toggle the varialbe to visualize each of them
         plotly_plotdf(df_base.resample('1min').mean())
```

2 Lenient Frequency Control

To configure the GridBallast controller, we set specific properties of the waterheater object in the glm file. The properties corresponding to the controller include:

- enable_freq_control [boolean]
- freq_lowlimit [float]
- freq_uplimit [float]
- enable_jitter [boolean]
- average_delay_time [integer]

For this test we modify waterheater 1 to enable the frequency control and set a wide frequency dead-band (59.9Hz - 60.1Hz). We expect the GridBallast controller to be rarely triggered.

```
In [14]: !head -611 smSingle_lenient_freq.glm|tail -21
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
    };
        enable_freq_control true;
        freq_lowlimit 59.9;
        freq_uplimit 60.1;
        heat_mode ELECTRIC;
};
```

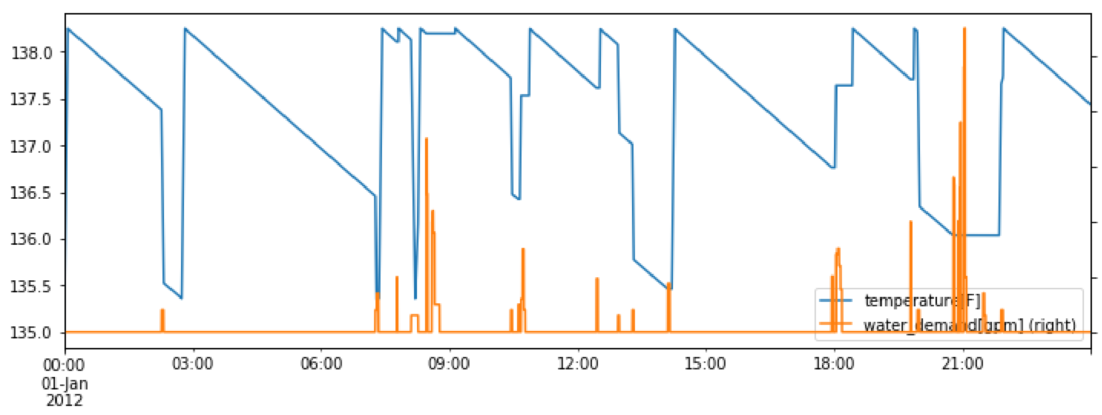
```
In [15]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_lenient_freq.glm
```

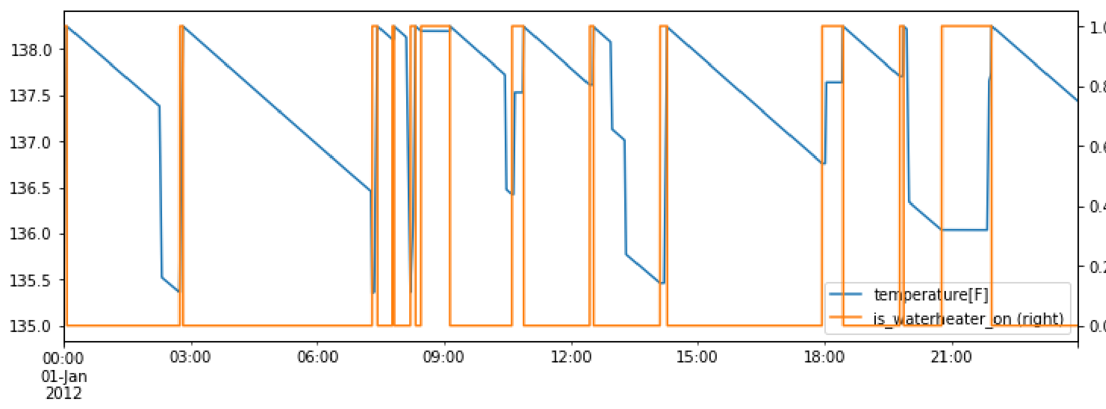
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 36 objects
Parallelism                    1 thread
Total time                  20.0 seconds
  Core time                  2.8 seconds (14.1%)
    Compiler                 1.2 seconds (6.2%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.8%)
  Model time                17.2 seconds/thread (85.9%)
Simulation time                1 days
Simulation speed              43 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate             4320 x realtime
```

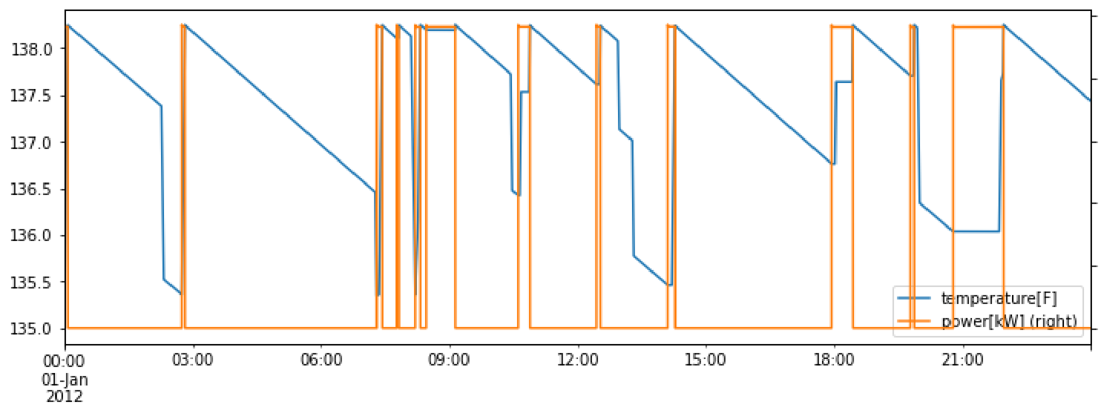
```
Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               10.110    58.9%   5055.0
triplex_meter       1.042     6.1%    347.3
house               0.829     4.8%    414.5
recorder            0.828     4.8%    276.0
ZIPload             0.811     4.7%    101.4
waterheater         0.728     4.2%    364.0
transformer         0.655     3.8%    327.5
triplex_line        0.570     3.3%    285.0
player              0.521     3.0%    521.0
regulator           0.417     2.4%    417.0
triplex_node        0.364     2.1%    364.0
auction             0.190     1.1%    190.0
climate             0.113     0.7%    113.0
================ ======== ======== ========
Total              17.178   100.0%    477.2

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

Now, we plot the generated waterheater data stored in **wh1_lenient_freq.csv** from the simulation.

```
In [16]: # We save data to wh1_lenient_freq.csv and plot the results
         df_lenient_freq = pd.read_csv('wh1_lenient_freq.csv',sep=',',header=8,
                           index_col=0,parse_dates=True,infer_datetime_format=True,
                           names=['freq[Hz]','temperature[F]','power[kW]','is_waterheater_on',
                                   'water_demand[gpm]'])

         df_lenient_freq[['temperature[F]', 'water_demand[gpm]']].plot(figsize=(12,4),
                                                         secondary_y='water_demand[gpm]')

         df_lenient_freq[['temperature[F]','is_waterheater_on']].plot(figsize=(12,4),
                                                        secondary_y='is_waterheater_on')

         df_lenient_freq[['temperature[F]','power[kW]']].plot(figsize=(12,4),
                                                        secondary_y='power[kW]')

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x115fc1198>
```







```
In [17]: # we can toggle the varialbe to visualize others
         plotly_plotdf(df_lenient_freq.resample('1min').mean())
```

3 Strict Frequency Control

We modify waterheater 1 to enable the frequency control, but we impose a tighter frequency deadband (59.97Hz - 60.03Hz). In other words, the gridballast controller should be triggered very often.

```
In [18]: !head -611 smSingle_strict_freq.glm|tail -21
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
};
```

```
In [19]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_strict_freq.glm
```

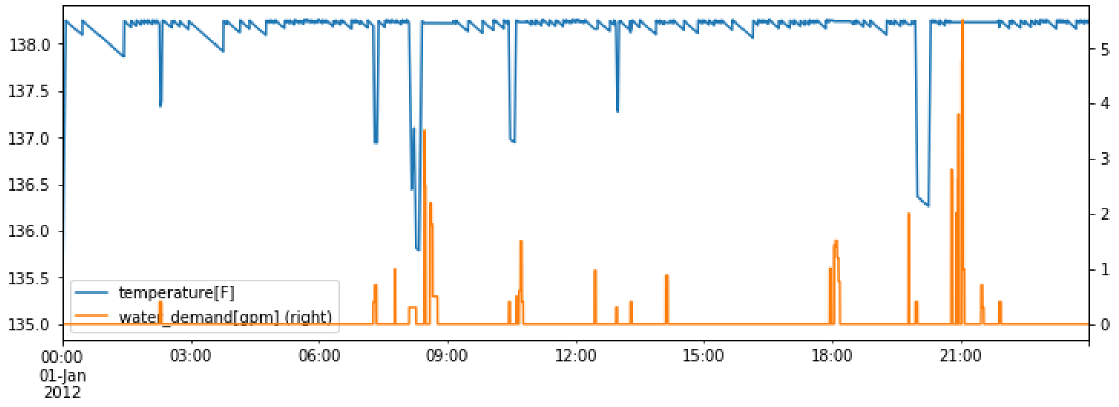
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 36 objects
Parallelism                    1 thread
Total time                  21.0 seconds
  Core time                  3.0 seconds (14.2%)
    Compiler                 1.3 seconds (6.3%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.1%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.9%)
  Model time                18.0 seconds/thread (85.8%)
Simulation time                1 days
Simulation speed              41 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate            4114 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               10.687    59.3%   5343.5
triplex_meter       1.069     5.9%    356.3
recorder            0.936     5.2%    312.0
house               0.870     4.8%    435.0
ZIPload             0.852     4.7%    106.5
waterheater         0.723     4.0%    361.5
transformer         0.619     3.4%    309.5
triplex_line        0.564     3.1%    282.0
player              0.513     2.8%    513.0
regulator           0.428     2.4%    428.0
triplex_node        0.428     2.4%    428.0
auction             0.208     1.2%    208.0
climate             0.112     0.6%    112.0
================ ======== ======== ========
Total              18.009   100.0%    500.2

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

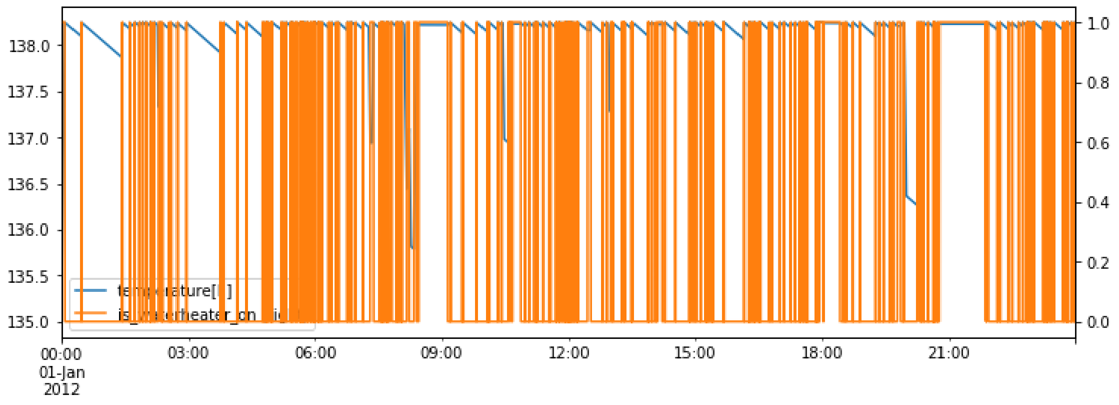
Now, we plot the generated waterheater data stored in **wh1_strict_freq.csv** from the simulation.

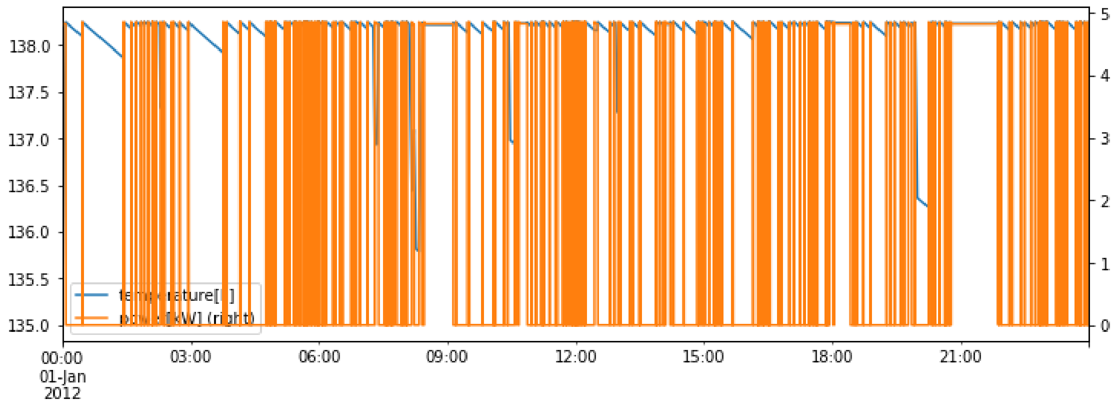
```
In [20]: # We save data to wh1_lenient_freq.csv and plot the results
         df_strict_freq = pd.read_csv('wh1_strict_freq.csv',sep=',',header=8,
                              index_col=0,parse_dates=True,infer_datetime_format=True,
                              names=['freq[Hz]','temperature[F]','power[kW]','is_waterheater_on',
                                     'water_demand[gpm]'])

         df_strict_freq[['temperature[F]', 'water_demand[gpm]']].plot(figsize=(12,4),
                                                               secondary_y='water_demand[gpm]')
```

```
df_strict_freq[['temperature[F]','is_waterheater_on']].plot(figsize=(12,4),
                                                            secondary_y='is_waterheater_on')

df_strict_freq[['temperature[F]','power[kW]']].plot(figsize=(12,4),
                                                    secondary_y='power[kW]')
```

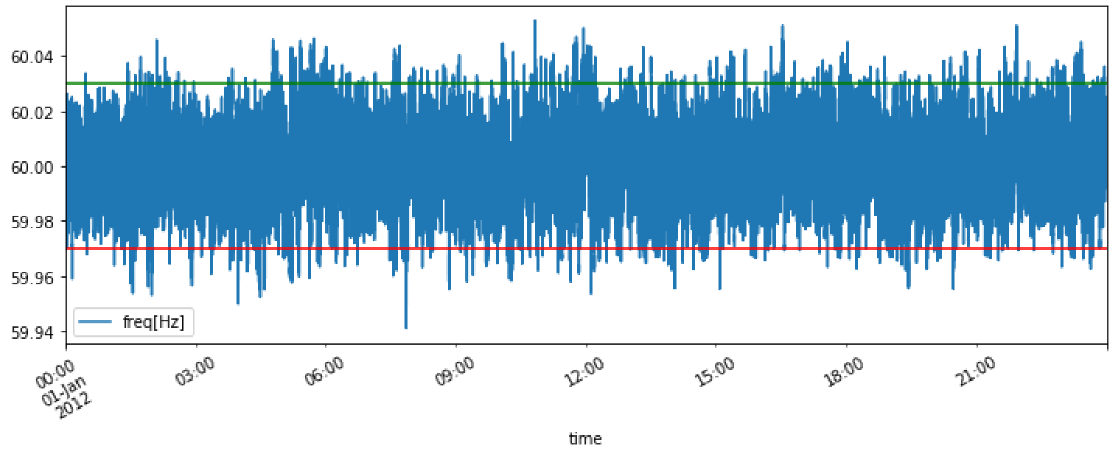
```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x115fc9080>
```







```
In [21]: # we can toggle the varialbe to visualize others
         plotly_plotdf(df_strict_freq.resample('1min').mean())

In [22]: # we can plot the frequency and the lower/upper limit again
         freq_low = 59.97
         freq_high = 60.03

         ax = raw_freq.plot(figsize=(12,4),rot=30)
         ax.axhline(y=freq_low, c='red')
         ax.axhline(y=freq_high, c='green')

Out[22]: <matplotlib.lines.Line2D at 0x11c36e2e8>
```



4 Strict Frequency Control with Jitter

We now modify the previous case (with a tight frequency deadband) and add a jitter to the response of the waterheater, such that the start of GridBallast event will delay randomly with an expected value of 600 seconds. Internally, the controller delay follows a uniform distribution over the interval [1,2*average_delay_time].

We use 600 seconds to clearly illustrate the difference in the power consumption patterns of the water heater previously illustrated and this one with jitter control enabled. Needless to say, users can set these values differently depending on how many water heaters are connected to the network or other considerations.

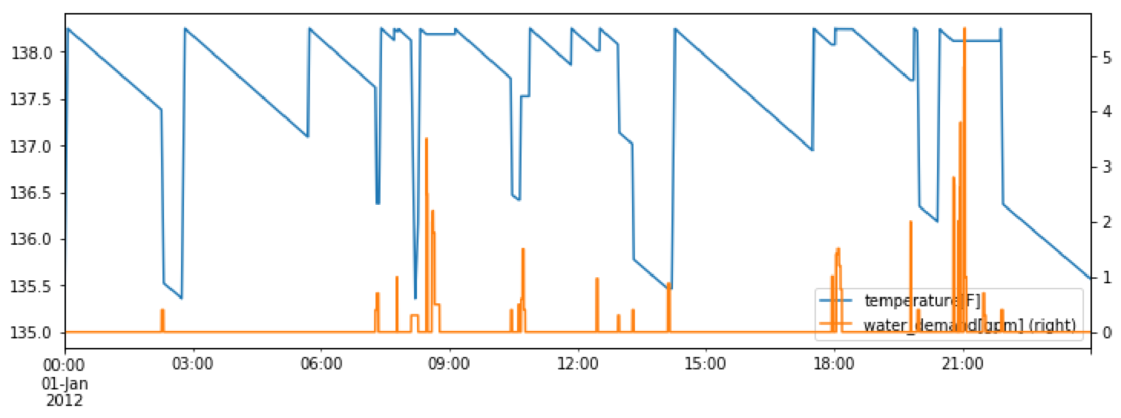
```
In [23]: !head -613 smSingle_strict_freq_jitter.glm|tail -23

object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
        enable_jitter true;
        average_delay_time 600;
};
```

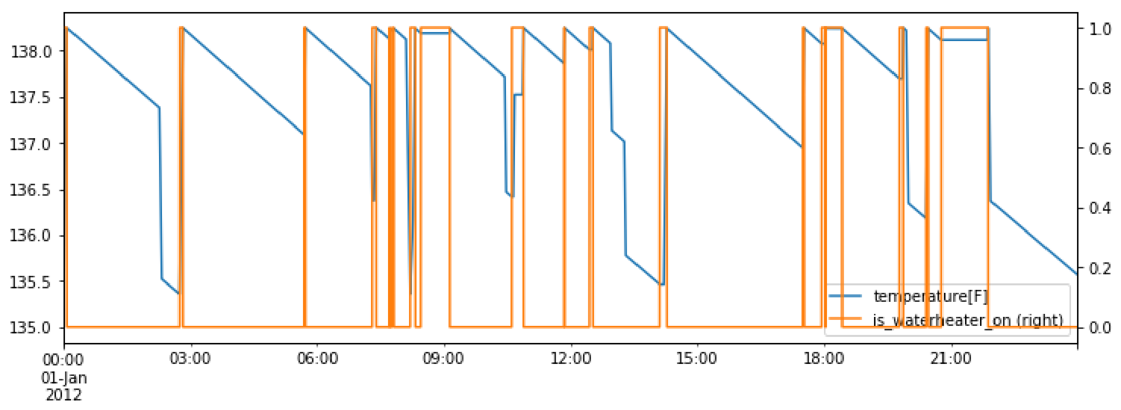
```
In [24]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_strict_freq_jitter.glm
```

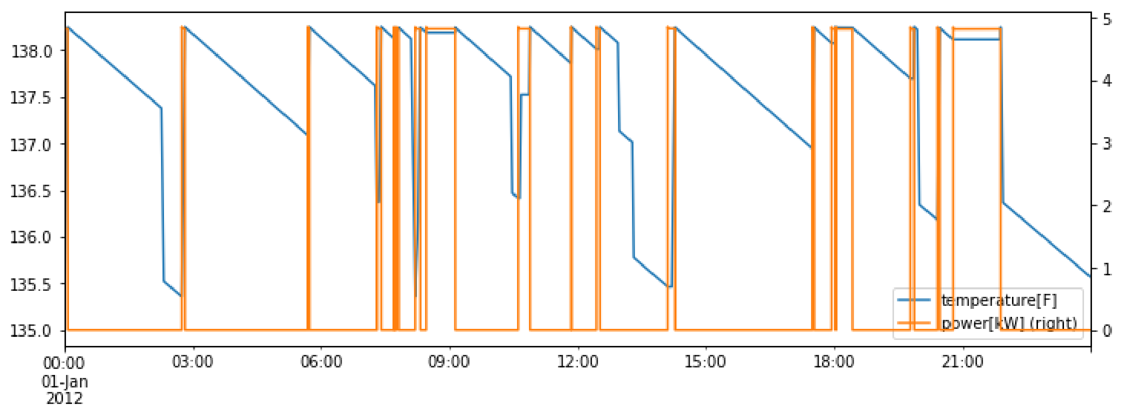
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, defaulting to 3.

Core profiler results
======================

Total objects                 36 objects
Parallelism                    1 thread
Total time                  19.0 seconds
  Core time                  2.2 seconds (11.5%)
    Compiler                 1.2 seconds (6.5%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.2%)
    Transforms               0.2 seconds (0.9%)
  Model time                16.8 seconds/thread (88.5%)
Simulation time                1 days
Simulation speed              45 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
```

```
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep              1 seconds/timestep
Simulation rate            4547 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node              10.020     59.6%   5010.0
triplex_meter      1.026      6.1%    342.0
recorder           0.877      5.2%    292.3
house              0.768      4.6%    384.0
waterheater        0.711      4.2%    355.5
ZIPload            0.709      4.2%     88.6
transformer        0.607      3.6%    303.5
triplex_line       0.556      3.3%    278.0
player             0.460      2.7%    460.0
regulator          0.400      2.4%    400.0
triplex_node       0.353      2.1%    353.0
auction            0.208      1.2%    208.0
climate            0.111      0.7%    111.0
================ ======== ======== ========
Total             16.806    100.0%    466.8

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

```
In [25]: # We save data to wh1_lenient_freq.csv and plot the results
         df_strict_freq = pd.read_csv('wh1_strict_freq_jitter.csv',sep=',',header=8,
                          index_col=0,parse_dates=True,infer_datetime_format=True,
                          names=['freq[Hz]','temperature[F]','power[kW]','is_waterheater_on',
                                 'water_demand[gpm]'])

         df_strict_freq[['temperature[F]', 'water_demand[gpm]']].plot(figsize=(12,4),
                                                           secondary_y='water_demand[gpm]')

         df_strict_freq[['temperature[F]','is_waterheater_on']].plot(figsize=(12,4),
                                                          secondary_y='is_waterheater_on')

         df_strict_freq[['temperature[F]','power[kW]']].plot(figsize=(12,4),
                                                          secondary_y='power[kW]')

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x11d5172e8>
```







As we can see, after applying the jitter, the water heater is engaged less often than in the previous experiment without jitter.