Energy Plus Occupancy + Optimization 2-Federate Simulation

Version 5.01 - Stable feature and bug fixing release
Brian Woo-Shem
Last updated: 2021-07-26

This is a major update; downloading and using the new federation files is recommended, rather than selectively swapping on an older version.

After version 5.0, no new updates are expected from Brian until mid-August.

New Optimization & Occupancy Modes:

The federation now supports 12 different operation modes:

1. Occupancy probability & current status + Optimization: the primary operation mode. Optimization combining probability data and current occupancy status
2. Occupancy probability & current status (no optimization)
3. Occupancy probability + optimization: optimization with only occupancy probability (NOT current status)
4. Occupancy probability (no optimization)
5. Occupancy current status + optimization: optimization with only occupancy sensor data for current occupancy status
6. Occupancy current status (no optimization)
7. Adaptive Setpoints + Optimization: optimization with adaptive setpoints where 90% people are comfortable. No occupancy
8. Adaptive Setpoints (no optimization)
9. Fixed + Optimization: optimization with fixed setpoints. No occupancy.
10. Fixed (no optimization)
11. Occupancy Precognition + Occupancy: Optimize if occupancy status for entire prediction period (2 hrs into future) is known. A joke!?
12. Occupancy Precognition (no optimization)

Easy Configuration:

- config.txt is now located in the deployment folder.
- Previously, to change the mode, heating vs. cooling, date range, and optimization settings, the Controller.java and energyOptTset2hr.py files had to be edited and recompiled with bash build-all.sh. Now, the code files stay the same and parameters are set in a single config.txt
- See instructions below

```
                                    config.txt
  Open    ▼   ⨍⊞        ~/Desktop/EnergyPlusOccOpt2Fed_4-6/EnergyPlusOpt2Fed_deployment        Save    ☰   _   □   ✕

  1 ip_address: <Options: 127.0.0.1 for VirtualBox, 10.16.130.### for Cluster>
  2 10.16.130.151
  3 port_number: <Socket 0 use 6789, +1 for each additional simulation>
  4 6789
  5 MODE: < occupancy, occupancy_prob, occupancy_sensor, adaptive90, fixed, occupancy_precognition >
  6 occupancy
  7 heatorcool: < heat, cool >
  8 cool
  9 date_range: < Match EP. Options: Jan1thru7, Feb12thru19, Sept27thruOct3, July1thru7 >
 10 Sept27thruOct3
 11 optimize: < true, false >
 12 true
```

Note: do NOT include quote marks '' around strings

MODE:

\#   'occupancy': the primary operation mode. Optimization combining probability
data and current occupancy status

\#   'occupancy_prob': optimization with only occupancy probability (NOT current
status)

\#   'occupancy_sensor': optimization with only occupancy sensor data for
current occupancy status

\#   'adaptive90': optimization with adaptive setpoints where 90% people are
comfortable. No occupancy

\#   'fixed': optimization with fixed setpoints. No occupancy.

\#   'occupancy_precognition': Optimize if occupancy status for entire
prediction period (2 hrs into future) is known. A joke!?

heatorcool:

\#   'heat': only heater, use in winter

\#   'cool': only AC, use in summer

date_range:

\#   'Jan1thru7'

\#   'Feb12thru19'

\#   'Sep27-Oct3_SJ'

\#   'July1thru7'

\#   'bugoff': For debugging and testing specific inputs. Hot w rapid cool off.
Run with day = 1, hour = 0.

\#   'bugfreeze': Extreme cold values, rapidly gets colder, for testing. Run
with day = 1, hour = 0.

\#   'bugcook': Extreme hot values, cools briefly then gets hotter for testing.
Run with day = 1, hour = 0.

\#   'bugsnug': Comfortable 18-23°C for testing both heat and cool.

\#   'bugwarm': Less extreme hot than bugcook mode

\#   'bugAC': Figure out why cool mode keeps failing if the price changes

\#   'bughprice': Analogous to bugAC but for heating with price change

optimize:

true: Use optimization code, energyOptTset2hr.py and try to optimize usage
false: Use the previously specified mode, but do not optimize

Well-Behaved File Paths

File paths in energyOptTset2hr.py, Controller.java, and Socket.java are now routed to the deployment folder. There is no need to change filepaths in the code when moving to a new computer.

Data output files are labeled with settings, date, and time so that they don't overwrite each other and it is easier to determine what each one is for afterward.

DataCVXOPT_occupancy_heat_2021-07-16_15-31.txt	116.8 kB	15:44
DataCVXOPT_occupancy_heat_2021-07-16_16-28.txt	76.4 kB	16:41
DataEP_occupancy_heat_2021-07-16_15-31.txt	149.4 kB	15:44
DataEP_occupancy_heat_2021-07-16_16-28.txt	144.5 kB	16:41

energyOptTset2hr.py Improvements

Occupancy Integration: Can now optimize for four occupancy modes (see 1, 3, 5, 11 under "New Optimization & Occupancy modes)
- Gets occupancy data from occupancy_1hr.csv
- Optimizes using setpoint bounds determined by occupancy
- WARNING: old versions of the occupancy_1hr.csv may not work correctly

New Input Parameters Method: date range, heat or cool, mode, number of timesteps to optimize, and number of timesteps to return can all be set either as values at the start of the program, or through additional inputs when running the code. This is meant for future controller improvements such that optimization can be run more frequently, and heating vs. cooling is set automatically by the program during operation. Including these parameters are optional and they can still be set within the python script as before.
The following formats all will work:

```
python energyOptTset2hr.py day hour temp_indoor_initial
python energyOptTset2hr.py day hour temp_indoor_initial n nt
python energyOptTset2hr.py day hour temp_indoor_initial n nt heatorcool
python energyOptTset2hr.py day hour temp_indoor_initial n nt heatorcool MODE
python energyOptTset2hr.py day hour temp_indoor_initial n nt heatorcool MODE
date_range
python energyOptTset2hr.py day hour temp_indoor_initial heatorcool
python energyOptTset2hr.py day hour temp_indoor_initial heatorcool MODE
python energyOptTset2hr.py day hour temp_indoor_initial heatorcool MODE date_range
python energyOptTset2hr.py day hour temp_indoor_initial heatorcool nt
```

```
python energyOptTset2hr.py day hour temp_indoor_initial n MODE
python energyOptTset2hr.py day hour temp_indoor_initial n MODE date_range
```
Note: Linux may use 'python3' or 'python3.9' instead of 'python'

   Windows use 'py'

day = [int] day number in simulation. 1 =< day =< [Number of days in simulation]

hour = [int] hour of the day. 12am = 0, 12pm = 11, 11pm = 23

temp_indoor_initial = [float] the initial indoor temperature in °C

n = [int] number of 5 minute timesteps to use in optimization. Requirement: n >= nt

nt = [int] number of 5 minute timesteps to return prediction for. Requirement: n >= nt

heatorcool = [str] see ===> SET HEATING VS COOLING! <===

MODE = [str] see ===> MODE <===

date_range = [str] see ===> WHEN TO RUN <===

Other improvements
- Added switching for adaptive vs fixed vs occupancy control - Brian
- Added occupancy optimization - Brian
- Fixed optimizer should not know occupancy status beyond current time
- Merging 2 codes, keeping the best characteristics of each
- Base code is energyOptTset2hr_kaleb.py, with stuff from the PJ/Brian one added
- Runtime reduced from 2.0 +/- 0.5s to 1.0 +/- 0.5s by moving mode-specific steps inside "if" statements and taking dataframe subsets before transformation to matrices
- Both heat and cool modes can optimize without out of bounds errors.
- Fixed off by one error where indoor temperature prediction is one step behind energy, causing it to miss some energy savings

occupancyAdaptSetpoints.py

New code that returns the non-optimized setpoints for any of the occupancy, adaptive, or fixed modes. It is a subset fork of the optimizer code, and it functions the same way but does not do any of the optimization stuff. Theoretically it can replace the section in Controller.java that computes adaptive and fixed un-optimized setpoints, but the simulation would run slower in these cases because it has to send to/from the Python.

Run it the same way as energyOptTset2hr.py

Controller.java
- Inputs for the heat or cool, mode, date range, and optimization now come from config.txt, see above.
- Added compatibility with occupancyAdaptSetpoints.py
- Added use config file instead of changing variables in here to reduce recompiling

- Fixed datastring receiver from Python so that datastrings do not continually append with data for the next datastring, and entire process is much faster and more reliable
- Cleaned up code
- Has variables needed for more frequent optimization calls but does not work because of energyOptTset2hr limitations
- No more long filepaths to change! Instead it saves to the run directory (usually deployment) by default!
- EP and CVXOPT Data Summary file paths have a date + time format so they don't overwrite previous runs.
- Includes more coherent error messages for common issues:
  - Python crash
  - Lost socket connection
  - Failed config input
  - Failed write to file

Socket.java

The filepath for the config.txt goes to deployment. No need to change and recompile on new computers.

Auto Shut-down in UCEF Federates

Socket and controller previously would print "Waiting for __ interaction" forever until they crash if the other federate closes. Now, they do 200 attempts over about 80 seconds to wait for the other's interaction; if they fail to respond after this long, it usually means the other is closed or crashed. After the wait period, the remaining federate also shuts down.

Tips & Tricks

Date ranges in EP are inclusive.

	Jan1thru7	Feb12thru19	Sept27thruOct3
Start Month	Jan	Feb	Sept
Start Day	1	12	27
End Month	Jan	Feb	Oct
End Day	1	18	3

If end day Feb 19 is used in Feb12thru19, EP will try to use an eighth day and the weather, WholesalePrice, Solar files in UCEF will not have enough data so either Controller or the Python will crash over a data not found error.

If it crashes on an optimization or occupancy mode, check the last lines of the controller log. If you see something like this:

```
114 ZoneTemp String: 20.892595
115 Run:  python3 ./energyOptTset2hr.py 1 0 20.892595 24 12 cool occupancy Sept27thruOct3
116 10:00:31.106 [org.webgme.guest.controller.Controller.main()] ERROR org.webgme.guest.controller.Controller -
    java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
```

It indicates that Python likely crashed and did not return values, so when Java tries to use the array of values sent, it returns null and Java crashes. The problem is in Python.

To debug, take the line after "Run:" and paste it into the same terminal where you ran the simulation. Use the Python output to debug the Python code. When the command runs without crashing, then the simulation will probably work.

```
vagrant@vagrant-virtual-machine:~/Desktop/EnergyPlusOccOpt2Fed_4-6/EnergyPlusOpt2Fed_deployment
$ bash run-default.sh ../EnergyPlusOpt2Fed_generated
Waiting for the federation manager to come online.....
Waiting for 1 instances of Controller to join....
Waiting for 1 instances of Socket to join....
Press any key to terminate the federation execution...
Waiting for the federation manager to terminate.....
vagrant@vagrant-virtual-machine:~/Desktop/EnergyPlusOccOpt2Fed_4-6/EnergyPlusOpt2Fed_deployment
$ python3 ./energyOptTset2hr.py 1 0 20.892595 24 12 cool occupancy Sept27thruOct3
Traceback (most recent call last):
  File "./energyOptTset2hr.py", line 174, in <module>
    price_df = pd.read_excel('WholesalePrice.xlsx', sheet_name=date_range)
  File "/home/vagrant/.local/lib/python3.8/site-packages/pandas/util/_decorators.py", line 299,
in wrapper
    return func(*args, **kwargs)
  File "/home/vagrant/.local/lib/python3.8/site-packages/pandas/io/excel/_base.py"  line 344, i
```

Typical run times: Occupancy Optimization, 1 week

52 minutes on VirtualBox

15 minutes on Cluster