FlexAssist Documentation

Jared Langevin

Oct 08, 2021

Contents

FlexAssist is a recommender engine that helps commercial building owners and operators effectively participate in demand response (DR) programs. To learn more about what FlexAssist can do, check out *An Overview of FlexAssist*.

Before using FlexAssist, users should consult the *Software Requirements* section to ensure that prerequisite programs and packages are installed on their computer. With the required software installed, users can proceed to the *Execution Guide* section for guidance on how to run the FlexAssist algorithm in various modes.

FlexAssist is freely available for public use on GitHub.

CHAPTER 1

Documentation Contents

1.1 An Overview of FlexAssist

1.1.1 What is FlexAssist and why is it needed?

- FlexAssist is a recommender engine that helps commercial building owners and operators of office and retail building types effectively participate in demand response (DR) programs.
- Given a set of contextual conditions forecasted for DR events on a day-ahead basis and owner/operator valuations of building services, FlexAssist determines which in a set of candidate building control strategies is likely to strike the best balance between the economic benefits of DR event participation and the risk of building service losses.
- FlexAssist addresses the need to provide building operators with decision-support resources as utilities seek to expand commerical DR programs and leverage greater flexibility in energy demand to help integrate variable renewable energy supply and improve the resilience of the electric grid. Expanded DR programs offer increasing opportunities for commercial buildings to participate in DR – e.g., through the services of load aggregators – yet, potential participants often lack the resources needed to assess the benefits and risks of responding to frequent DR event calls from the grid. FlexAssist fills this gap by automating the process of conducting a benefit-risk assessment in advance of an event and determining how best to respond, and by learning from information collected during each event to continually improve the underlying prediction framework.

1.1.2 Operator preferences are directly accounted for

- Discrete choice modeling translates predicted changes in building demand and services under candidate DR strategies into an operator utility score.
- Choice model coefficients are drawn from discrete choice experiments that infer operator weightings of various load adjustments (e.g. temperature, lighting, etc.) against potential economic benefits.
- Currently considered inputs to the choice model include the expected economic benefit of implementing a candidate DR strategy during a given event, as well as the expected maximum increase in temperature, lighting reduction, and reduction in plug load power availability from implementing the candidate DR strategy.

1.1.3 Recommendations reflect decision-making uncertainties

- Underlying models estimating the change in building demand and services under DR and operator utility are implemented using probabilistic programming techniques that directly represent the uncertainty in model predictions.
- Model outputs are provided as a range of values, rather than single point values, and communicate the likelihood of service losses and economic benefits under candidate DR strategies.
- Attaching operator valuations from the discrete choice experiments to probable service losses yields a powerful risk assessment framework to guide DR participation in a particular context and under a particular set of conditions.

1.1.4 Underlying models can be updated with new DR event data

- FlexAssist is conceptualized as a Bayesian decision network that merges prior expectations about key modeled relationships (e.g., between change in thermostat set point from a DR strategy and demand reduction) with observed evidence on these relationships.
- Under this framework, model parameters are considered random and are assigned a probability distribution that is easily updated with new evidence.
- In practice, this means that FlexAssist's predictions may be automatically re-calibrated with evidence of the actual effects of DR strategies in real-world settings, given the collection of relevant event data (e.g., metered demand, indoor environmental conditions, weather, occupancy).

1.2 Software Requirements

FlexAssist runs on Python and several of its supporting packages, as outlined below.

Prerequisites*

- Python 3
- Python packages: pymc3, theano, numpy, scipy, arviz, matplotlib

Instructions follow for installing Python and these supporting packages on Mac OS and Windows.

1.2.1 Mac OS installation

0. (Optional) Install a package manager

The Mac OS ships with Python already installed. Installing and using a package manager will make it easy to ensure that any additional installations of Python do not interfere with the version of Python included with the operating system. Homebrew is a popular package manager, but there are other options, such as MacPorts and Fink.

Note: While this step is optional, subsequent instructions are written with the assumption that you have installed Homebrew as your package manager.

To install Homebrew, open Terminal (found in Applications/Utilities, or trigger Spotlight with -space and type "Terminal"). Visit the Homebrew website and copy the installation command text on the page. Paste the text into the Terminal application window and press Return. If you encounter problems with the installation, return to the Homebrew website for help or search online for troubleshooting assistance.

If you are using a package manager other than Homebrew, follow the documentation for that package manager to install Python 3. If you have chosen to not install a package manager, you may use the Python Software Foundation installer for the latest version of Python 3.

1. Install Python 3

In a Terminal window, at the command prompt (a line terminated with a $ character and a flashing cursor), type:

```
brew install python3
```

2. Install required Python packages

Once Python 3 is fully installed, pip3 is the tool you will use to install add-ons specific to Python 3. Six packages, pymc3, theano, numpy, scipy, arviz, and matplotlib, are required for Scout. To install them, at the command prompt in Terminal, type:

```
pip3 install pymc3 theano numpy scipy arviz matplotlib
```

If you'd like to confirm that the packages were installed successfully, you can start Python from the command prompt in Terminal by typing:

```
python3
```

and import the packages (within the Python interactive shell, indicated by the >>> prompt).

```
import theano, numpy, scipy, arviz, matplotlib
```

If no error or warning messages appear, then the installation was successful and you can exit Python by typing `quit()`.

1.2.2 Windows installation

1. Install Python 3

Tip: If you have 64-bit Windows installed on your computer, downloading and installing the 64-bit version of Python is recommended.

Download the executable installer for Windows available on the Python Software Foundation downloads page. Run the installer and follow the on-screen prompts as you would with any other software installer. Be sure that the option in the installer "Add Python 3.x to PATH," where x denotes the current version of Python 3, is checked.

2. Install required Python packages

Once Python 3 installation is complete, six Python packages need to be installed: pymc3, theano, numpy, scipy, arviz, and matplotlib. pip is the tool you will use to install add-ons specific to Python 3. Begin by opening a command prompt window. At the prompt (a line of text with a file path terminated by a greater than symbol, such as `C:\>`), type:

```
py -3 -m pip install pymc3 theano numpy scipy arviz matplotlib
```

If you would like to confirm that the packages were installed successfully, you can open an interactive session of Python in a command prompt window by typing:

```
py -3
```

and then importing the packages (within the Python interactive session, indicated by a >>> prompt):

```
import pymc3, theano, numpy, scipy, arviz, matplotlib
```

If no error or warning messages appear, the packages were installed successfully. Exit the interactive session of Python by typing:

```
quit()
```

1.3 Execution Guide

FlexAssist supports multiple execution modes:

- Selecting from a set of candidate DR response strategies.
- Generating predictions of baseline demand (without implementation of DR strategies).
- Generating new models of building demand/services under DR and baseline conditions.
- Updating existing models of building demand/services given new data.

Guidance on preparing inputs, running the model, and interpreting output for each of these execution modes is provided below.

1.3.1 Selecting from set of DR response strategies

Given a set of candidate DR strategies and day-ahead DR event conditions as an input, this model execution mode will determine which candidate strategy is most likely to maximize operator utility and provide that information as an output. Selection likelihoods are generated based on the predicted change in demand and building services under each of the candidate DR strategies and DR event conditions, and considering operator valuations of various service loss risks.

Required input files

The input file for this execution mode is found in `./data/test_predict.csv.`. Table 1.1 outlines the columns that must be present in this file.

Table 1.1: Description of the input file for generating the recommendations.

Column name	Description
Name	Name of the DR strategy
Hr	Hours into the DR event, starting from 1
OAT	Outdoor air temperature (ºF)
RH	Outdoor relative humidity (%)
Lt_Nat	Outdoor natural illuminance (lux)
Lt_Base	Indoor lighting schedule fraction
Occ_Frac	Occupant schedule fraction
Delt_Price_kWh	Utility incentive for DR
h_DR_Start	Hours into the DR window
h_DR_End	Hours since DR window ended
h_PCool_Start	Hours since pre-cooling started
h_PCool_End	Hours since pre-cooling ended
Delt_CoolSP	Change in temperature set point (ºF)
Delt_LgtPct	Reduction in lighting power (%)
Delt_OAVent_Pct	Reduction in outdoor air fraction
Delt_PL_Pct	Reduction in plug load power (%)
Delt_CoolSP_Lag	Change in temperature set point since previous hour (ºF)
Delt_LgtPct_Lag	Change in lighting power since previous hour (%)
Delt_OAVent_Pct-Lag	Change in outdoor air fraction since previous hour (%)
Delt_PL_Pct_Lag	Change in plugload power since previous hour (%)
Pcool_Mag	Magnitude of pre-cooling before event started (ºF temperature decrease)
Pcool_Dur	Duration of precooling before DR event started

An example of this input file is available that reflects the candidate DR strategies shown in Table Table 1.2.

Note: Plug load reduction strategies are currently only supported when running simulations for office building types; when running simulations for retail buildings, all settings in `./data/test_predict.csv.` related to plug loads should be set to zero.

Table 1.2: Example set of DR strategy characteristics.

DR strategy name	Description	Magnitude
GTA-Low	Global temperature adjustment	+2ºF
GTA-Moderate	Global temperature adjustment	+3ºF
GTA-High	Global temperature adjustment	+5ºF
Precool-Low	Global temp. adjustment Precooling temperature change	+2ºF -2ºF
Precool-Moderate	Global temp. adjustment Precooling temperature change	+3ºF -5ºF
Precool-High	Global temp. adjustment Precooling temperature change	+5ºF -5ºF
Dimming-Low	Lighting dimming	-20%
Dimming-Moderate	Lighting dimming	-40%
Dimming-High	Lighting dimming	-60%
Plug Load-Low	Plug load reduction	-10%
Plug Load-Moderate	Plug load reduction	-30%
Plug Load-High	Plug load reduction	-50%
Package-Low	Global temp. adjustment Lighting dimming Plug load reduction	+2ºF -10% -10%
Package-Moderate	Global temp. adjustment Lighting dimming Plug load reduction	+4ºF -30% -30%
Package-High	Global temp. adjustment Lighting dimming Plug load reduction	+6ºF -30% -30%
Package-Low-Precool	Global temp. adjustment Lighting dimming Plug load reduction Precooling temperature change	+2ºF -20% -10% -2ºF
Package-Moderate-Precool	Global temp. adjustment Lighting dimming Plug load reduction Precooling temperature change	+3ºF -40% -30% -5ºF
Package-High-Precool	Global temp. adjustment Lighting dimming Plug load reduction Precooling temperature change	+5ºF -60% -50% -5ºF

Running the model

DR strategy recommendations and associated predictions are generated with the following command line/Terminal inputs on Windows and MacOS, respectively:

Windows

```
cd Documents\projects\flex-bldgs
py -3 flex.py --bldg_type <insert building type/vintage name> --bldg_sf <insert␣
↪square footage> <additional options>
```

Mac

```
cd Documents/projects/flex-bldgs
python3 flex.py --bldg_type <insert buildling type/vintage name> --bldg_sf <insert␣
↪square footage> <additional options>
```

Where `--bldg_type` options include `mediumofficenew` (~50K square foot medium office post-2004), `mediumofficeold` (~50K square foot medium office pre-2004, `stdaloneretailnew` (~25K standalone retail building post-2004), `stdaloneretailold` (~25K standalone retail building pre-2004), `largeofficenew` (~500K square foot large office pre-2004), `largeofficeold` (~500K square foot large office post-2004), `largeofficenew_elec` (all-electric version of `largeofficenew` using water-source heat pump and dedicated outdoor air system (DOAS)), `largeofficeold_elec` (all-electric version of `largeofficeold`), and `bigboxretail` (~100K square foot big box retail building, 2004 vintage). Values for `--bldg_sf` should be entered as-is (e.g., 50,000 for a 50K square foot building)

Additional run options (represented in the command above by `<additional options>`) include `--null_strategy`, which will include the option of taking no actions in the candidate strategies; `--dmd_thres <insert value>`, which will impose a threshold on maximum during-event hourly demand reduction (in W) across strategies and remove strategies that do not meet the threshold from consideration on an event-by-event basis; and `--daylt_pct <insert value>` which will adjust the standard 30% daylit floorspace assumed in making predictions to the user-specified value (in %).

The model will load the input data and begin predicting the changes in demand and indoor services during each of the event hours reflected in the input file, drawing upon previously initialized models of building demand and services (see *Selecting from set of DR response strategies*).

Interpretation of outputs

Outputs from execution of this mode are stored in the file `./data/recommendations.json`. The file has the following structure:

```
{
 "notes": <notes about the contents of the file>,
 "predictions": {
     "DR strategy name 1": <Percentage of simulations in which DR strategy 1 was␣
↪selected>, ...
     "DR strategy name N": <Percentage of simulations in which DR strategy N was␣
↪selected>, ...
  },
 "input output data": {
      "demand": {
         "DR strategy name 1": [<All sampled maximum hourly demand reduction values␣
↪(W/sf) for DR strategy name 1>],
         "DR strategy name N": [<All sampled maximum hourly demand reduction values␣
↪(W/sf) for DR strategy name N>]},
      "demand precool": {
         "DR strategy name 1": [<All sampled maximum hourly demand increase from␣
↪precooling values (W/sf) for DR strategy name 1>],
         "DR strategy name N": [<All sampled maximum hourly demand increase from␣
↪precooling values (W/sf) for DR strategy name N>]},
      "cost": {
         "DR strategy name 1": [<All sampled total economic benefit values ($) for DR␣
↪strategy name 1>],
         "DR strategy name N": [<All sampled total economic benefit values ($) for DR␣
↪strategy name N>]},
      "cost precool": {
         "DR strategy name 1": [<All sampled total economic loss from precooling␣
↪values ($) for DR strategy name 1>],
         "DR strategy name N": [<All sampled total economic loss from precooling␣
↪values ($) for DR strategy name N>]},
      "temperature": {
         "DR strategy name 1": [<All sampled maximum hourly temperature increase␣
↪values (°F) for DR strategy name 1>],
         "DR strategy name N": [<All sampled maximum hourly temperature increase␣
↪values (°F) for DR strategy name N>]},
      "lighting": {
         "DR strategy name 1": [<Input lighting reduction fraction for DR strategy␣
↪name 1>],
         "DR strategy name N": [<Input lighting reduction fraction for DR strategy␣
↪name N>]},
      "plug loads": {
         "DR strategy name 1": [<Input plug load reduction fraction for DR strategy␣
↪name 1>],
```

(continues on next page)

(continued from previous page)

```
        "DR strategy name N": [<Input plug load reduction fraction for DR strategy␣
↪name N>]}
  }
}
```

1.3.2 Generating baseline demand predictions

One of the functions of FlexAssist is to generate the baseline demand value given certain conditions, such as weather and occupancy, and building characteristics such as type and vintage. Models of baseline demand follow the same approach as those that were fit to predict the changes in demand and building servies under candidate DR strategies, which are used in *Generating new models*.

Required input files

The input file for this execution mode is found in `./data/test_predict_bl.csv`. Table 1.3 outlines the columns that must be present in this file.

Table 1.3: Description of the input file for generating the baseline demand prediction.

Column name	Description
Hr	Hour into the DR event, starting from 1
Vintage	Four vintages are considered within the scope: 1980, 1980-2004, 2004, 2010
Hour_number	Actual time based on 24-hour military time
Climate	Climate zone where the building is located, followed by IECC climate zone map
OAT	Outdoor air temperature (ºF)
RH	Outdoor relative humidity (%)
Occ_Frac	Occupancy schedule fraction
V1980 … V19802004	Binary check box
CZ.2A … CZ.7A	Binary check box

Running the model

Baseline demand predictions are generated using the `--base_pred` option as below:

Windows

```
cd Documents\projects\flex-bldgs
py -3 flex.py --base_pred --bldg_type <insert bldg name> --bldg_sf <insert sf>
```

Mac

```
cd Documents/projects/flex-bldgs
python3 flex.py  --base_pred --bldg_type <insert bldg name> --bldg_sf <insert sf>
```

The model will automatically load in the input data and start calculating the hourly baseline demand values given the input information.

Interpretation of outputs

Predicted hourly baseline demand values are reported in `./data/base_predict_summary.csv`. For each predicted hour, there will be 1) mean value (W/sf), and 2) standard deviation together indicating the predicted results. By default, the sample number for generating these results is set to 1000.

1.3.3 Generating new models

Users can use this mode to initialize/re-initialize all the models of building demand/services and operator utility that underly FlexAssist's predictions, given input CSV data that follows a certain data structure. The model list includes the following 6 regression models:

- Baseline demand value
- Thermally-driven demand changes during the DR period (for strategies involving temperature adjustment)
- Non thermally-driven demand changes during the DR period (for strategies not involving temperature adjustment)
- Demand changes during the pre-cooling period
- Indoor temperature changes during the DR period (relevant only to strategies involving temperature adjustment)
- Operator utility

Required input files

Input files for this execution mode are found in the `./data` directory. Current CSV files underlying the models of building demand and services were generated from a batch of simulations in EnergyPlus, where four scenarios of building types and vintages were considered. Another file with training data for the operator choice model was developed from discrete choice experiments with building operators. If users want to re-initialize the models using their own data, the format of their CSV files must be consistent with these current files. Table Table 1.4 shows example CSV file names underlying demand and service models sfor the medium office new vintage (post-2004); these example CSVs may serve as useful references for formatting and content.

Table 1.4: Input files for generating new regression models.

Input file	Regression model
MO_B.csv	Baseline demand prediction
MO_Thermal_Demand_new.csv	Thermally-driven demand changes during the DR period
MO_Nonthermal_Demand_new.csv	Non thermally-driven demand changes during the DR period
MO_PC_Demand_new.csv	Demand changes during the pre-cooling period
MO_Temperature_new.csv	Indoor temperature changes during DR

Running the model

New model initialization is executed using the `--mod_init` option as below:

Windows

```
cd Documents\projects\flex-bldgs
py -3 flex.py --mod_init --bldg_type <insert bldg name>
```

Mac

```
cd Documents/projects/flex-bldgs
python3 flex.py --mod_init --bldg_type <insert bldg name>
```

The model will start loading input data and initializing the variables for each regression model.

Interpretation of outputs

Model coefficient samples from the Bayesian inference framework are saved as pickled files (`.pkl`) to the `./model_stored` directory. For example, for the medium office building type and new vintage (post-2004), each file represents specific model(s) as shown in Table 1.5.

Table 1.5: Coefficient data for generated regression models.

Output file	Regression model
dmd_mo_b.pkl	Baseline demand prediction
dmd_therm_mo_n.pkl	Thermally-driven demand changes during the DR event period
dmd_ntherm_mo_n.pkl	Non thermally-driven demand changes during the DR period
pc_dmd_mo_n.pkl	Demand changes during the pre-cooling period
tmp_mo_n.pkl	Indoor temperature changes during DR
dce.pkl	Operator utility

These stored moddel coefficient data are then drawn from in the other execution modes.

1.3.4 Updating existing models

FlexAssist enables updating of previously initialized models of change in building demand and services based on new data observations. Existing models are stored in `.pkl` files in the `./model_stored` folder, as outlined in Table 1.5.

Required input files

The input file for this execution mode is found in `./data/test_update.csv` for updating the models of demand/services under DR strategies, and in `./data/test_update_bl.csv` for updating the models of baseline demand. Table 1.6 outlines the columns that must be present in the formert file; example files are found here for models of demand/services and here for models of baseline demand.

Table 1.6: Description of the input file for estimating the existing models.

Column name	Description
Vintage	Four vintages are considered within the scope: 1980, 1980-2004, 2004, 2010
Climate.zone	Climate zone where the building is located, followed by IECC climate zone map
Hour.number	Actual time based on 24-hour military time
Outdoor.Temp.F	Outdoor air temperature (ºF)
Outdoor.Humid.	Outdoor relative humidity (%)
Occ.Fraction.	Occupancy schedule fraction (<=1)
Demand.Power.Diff.sf.	Whole building reduction in electricity demand from baseline (W/sf)
Indoor.Temp.Diff.F.	Indoor temperature difference from baseline (ºF)
Indoor.Humid.Diff.F.	Indoor RH difference from baseline (ºF)
Lighting.Power.Diff.pct.	Reduction in lighting power (%)
MELs.Diff.pct.	Reduction in plug load power (%)
Cooling.Setpoint.Diff.	Change in temperature set point (ºF)
Cooling.Setpoint.Diff. One.Step.	Change in temperature set point since previous hour (ºF)
Since.DR.Started	Hours into the DR window
Since.DR.Ended	Hours since DR window ended
Pre.cooling.Temp. Increase.	Magnitude of pre-cooling before event started (ºF temperature decrease)
Pre.cooling.Duration	Duration of precooling before DR event started
Since.Pre.cooling. Started.	Hours since pre-cooling started
Since.Pre.cooling.Ended.	Hours since pre-cooling ended

Running the model

Model updates are executed using the `--mod_est` option as below:

Windows

```
cd Documents\projects\flex-bldgs
py -3 flex.py --mod_est --bldg_type <insert bldg name>
```

Mac

```
cd Documents/projects/flex-bldgs
python3 flex.py --mod_est --bldg_type <insert bldg name>
```

The model will automatically load in the input data and start updating the previously initialized models.

Interpretation of outputs

Updated model parameter coefficient distribution estimates are written to the `.pkl` files in the `./model_stored` folder. The different `.pkl` model types are enumerated in Table 1.5.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search