Prescient

Release 2.0.2

Prescient Developers

Jul 26, 2022

CONTENTS:

Prescient is a python library that provides production cost modeling capabilities for power generation and distribution networks.

CHAPTER

ONE

USING PRESCIENT

1.1 Installation

The Prescient python package can be installed using pip, or it can be installed from source. Python and a linear solver are prerequisites for either installation method.

To install Prescient, follow these steps:

1.1.1 Install python

Prescient requires python 3.7 or later. We recommend installing Anaconda to manage python and other dependencies.

1.1.2 Install a linear solver

Prescient requires a mixed-integer linear programming (MILP) solver that is compatible with Pyomo. Options include open source solvers such as CBC or GLPK, and commercial solvers such as CPLEX, Gurobi, or Xpress.

The specific mechanics of installing a solver is specific to the solver and/or the platform. An easy way to install an open source solver on Windows, Linux, and Mac is to install the CBC Anaconda package into the current conda environment:

```
conda install -c conda-forge coincbc
```

Tip: Be sure to activate the correct python environment before running the command above.

Note that the CBC solver is used in most Prescient tests, so you may want to install it even if you intend to use another solver in your own runs.

1.1.3 Install Using Pip

Prescient is available as a python package that can be installed using pip. To install the latest release of Prescient use the following command:

> pip install gridx-prescient

Be sure the intended python environment is active before issuing the command above.

1.1.4 Install From Source

You may want to install from source if you want to use the latest pre-release version of the code, or if you want to modify/contribute to the code yourself. The steps required to install Prescient from source are described below:

Get Prescient source code

The latest version of Prescient can be acquired as source from the Prescient github project, either by downloading a zip file of the source code or by cloning the *main* branch of the github repository.

Install Python Dependencies

The python environment where you run Prescient must include a number of prerequisites. You may want to create a python environment specifically for Prescient. To create a new Anaconda environment and install Prescient's prerequisites into the new environment, issue the following command from the root folder of the Prescient source code:

```
conda env create -f environment.yml
```

The command above will create an environment named *prescient*. To use a different name for the environment, add the *-n* option to the command above:

```
conda env create -n nameOfYourChoice -f environment.yml
```

Once you have create the new environment, make it the active environment:

```
conda activate prescient
```

If you are using something other than Anaconda to manage your python environment, use the information in *environment.yml* to identify which packages to install.

Install Egret

When installing Prescient from the latest version of the source code, Egret may need to be installed manually because pre-release versions of Prescient sometimes depend on pre-release versions of EGRET. Install EGRET from source according to the instructions *here <https://github.com/grid-parity-exchange/Egret/blob/main/README.md>*.

Install the Prescient python package

The steps above configure a python environment with Prescient's prerequisites. Now we must install Prescient itself. From the prescient python environment, issue the following command:

```
pip install -e .
```

This will update the active python environment to include Prescient's source code. Any changes to Prescient source code will take affect each time Prescient is run.

This command will also install a few utilities that Prescient users may find useful, including *runner.py* (see *Running Prescient*).

Verify your installation

Prescient is packaged with tests to verify it has been set up correctly. To execute the tests, issue the following command:

```
python -m unittest tests/simulator_tests/test_sim_rts_mod.py
```

This command runs the tests using the CBC solver and will fail if you haven't installed CBC. The tests can take as long as 30 minutes to run, depending on your machine. If Prescient was installed correctly then all tests should pass.

1.2 Running Prescient

There are three ways to launch and run Prescient:

- With a configuration file, *using runner.py*
- With command line options, *using the prescient.simulator module*
- From python code, *using in-code configuration*

In all three cases, the analyst supplies configuration values that identify input data and dictate which options to use during the Prescient simulation. Configuration options can be specified in a configuration file, on the command line, in-code, or a combination of these methods, depending on how Prescient is launched.

To see what configuration options are available, see *Configuration Options*.

1.2.1 Launch with runner.py

Prescient can be run using *runner.py*, a utility which is installed along with Prescient (see *Install Egret*). Before executing *runner.py*, you must create a configuration file indicating how Prescient should be run. Here is an example of a configuration file that can be used with *runner.py*:

```
command/exec simulator.py
--data-directory=example_scenario_input
--output-directory=example_scenario_output
--input-format=rts-gmlc
--run-sced-with-persistent-forecast-errors
--start-date=07-11-2024
--num-days=7
--sced-horizon=1
--sced-frequency-minutes=10
--ruc-horizon=36
```

Because runner.py can potentially be used for more than launching Prescient, the first line of the configuration file must match the line shown in the example above. Otherwise runner.py won't know that you intend to run Prescient.

All subsequent lines set the value of a configuration option. Configuration options are described in *Configuration Options*.

Once you have the configuration file prepared, you can launch Prescient using the following command:

```
runner.py config.txt
```

where *config.txt* should be replaced with the name of your configuration file.

1.2.2 Launch with the *prescient.simulator* module

Another way to run Prescient is to execute the *prescient.simulator* module:

```
python -m prescient.simulator <options>
```

where *options* specifies the configuration options for the run. An example might be something like this:

```
python -m prescient.simulator --data-directory=example_scenario_input --output-
↪directory=example_scenario_output --input-format=rts-gmlc --run-sced-with-persistent-
↪forecast-errors --start-date=07-11-2024 --num-days=7 --sced-horizon=1 --sced-frequency-
↪minutes=10 --ruc-horizon=36
```

Configuration options can also be specified in a configuration file:

```
python -m prescient.simulator --config-file=config.txt
```

You can combine the *–config-file* option with other command line options. The contents of the configuration file are effectively inserted into the command line at the location of the *–config-file* option. You can override values in a configuration file by repeating the option at some point after the *–config-file* option.

Running the *prescient.simulator* module allows you to run Prescient without explicitly installing it, as long as Prescient is found in the python module search path.

1.2.3 Running Prescient from python code

Prescient can be configured and launched from python code:

```python
from prescient.simulator import Prescient

Prescient().simulate(
        data_path='deterministic_scenarios',
        simulate_out_of_sample=True,
        run_sced_with_persistent_forecast_errors=True,
        output_directory='deterministic_simulation_output',
        start_date='07-10-2020',
        num_days=7,
        sced_horizon=4,
        reserve_factor=0.0,
        deterministic_ruc_solver='cbc',
        sced_solver='cbc',
        sced_frequency_minutes=60,
```

(continues on next page)

(continued from previous page)

```
        ruc_horizon=36,
        enforce_sced_shutdown_ramprate=True,
        no_startup_shutdown_curves=True)
```

The code example above creates an instance of the Prescient class and passes configuration options to its *simulate()* method. An alternative is to set values on a configuration object, and then run the simulation after configuration is done:

```
from prescient.simulator import Prescient

p = Prescient()

config = p.config
config.data_path='deterministic_scenarios'
config.simulate_out_of_sample=True
config.run_sced_with_persistent_forecast_errors=True
config.output_directory='deterministic_simulation_output'
config.start_date='07-10-2020'
config.num_days=7
config.sced_horizon=4
config.reserve_factor=0.0
config.deterministic_ruc_solver='cbc'
config.sced_solver='cbc'
config.sced_frequency_minutes=60
config.ruc_horizon=36
config.enforce_sced_shutdown_ramprate=True
config.no_startup_shutdown_curves=True

p.simulate()
```

A third option is to store configuration values in a *dict*, which can potentially be shared among multiple runs:

```
from prescient.simulator import Prescient

options = {
    'data_path':'deterministic_scenarios',
    'simulate_out_of_sample':True,
    'run_sced_with_persistent_forecast_errors':True,
    'output_directory':'deterministic_simulation_output'
}

Prescient().simulate(**options)
```

These three methods can be used together quite flexibly. The example below demonstrates a combination of approaches to configuring a prescient run:

```
from prescient.simulator import Prescient

simulator = Prescient()

# Set some configuration options using the simulator's config object
config = simulator.config
config.data_path='deterministic_scenarios'
```

(continues on next page)

(continued from previous page)

```
config.simulate_out_of_sample=True
config.run_sced_with_persistent_forecast_errors=True
config.output_directory='deterministic_simulation_output'

# Others will be stored in a dictionary that can
# potentially be shared among multiple prescient runs
options = {
    'start_date':'07-10-2020',
    'sced_horizon':4,
    'reserve_factor':0.0,
    'deterministic_ruc_solver':'cbc',
    'sced_solver':'cbc',
    'sced_frequency_minutes':60,
    'ruc_horizon':36,
    'enforce_sced_shutdown_ramprate':True,
    'no_startup_shutdown_curves':True,
}

# And finally, pass the dictionary to the simulate() method,
# along with an additional function argument.
simulator.simulate(**options, num_days=7)
```

1.3 Configuration Options

1.3.1 Overview

Prescient configuration options are used to indicate how the Prescient simulation should be run. Configuration options can be specified on the command line, in a text configuration file, or in code, depending on how Prescient is launched (see *Running Prescient*).

Each configuration option has a name, a data type, and a default value. The name used on the command line and the name used in code vary slightly. For example, the number of days to simulate is specified as *--num-days* on the command line, and *num_days* in code.

1.3.2 Option Data Types

Most options use self-explanatory data types like *String*, *Integer*, and *Float*, but some data types require more explanation and may be specified in code in ways that are unavailable on the command line:

Table 1: Configuration Data Types

Data type	Command-line/config file usage	In-code usage
Path	A text string that refers to a file or folder. Can be relative or absolute, and may include special characters such as ~.	Same as command-line
Date	A string that can be converted to a date, such as *1776-07-04*.	Either a string or a datetime object.
Flag	Simply include the option to set it to true. For example, the command below sets *simulate_out_of_sample* to true: `runner.py --simulate-out-of-sample`	Set the option by assigning True or False: `config.simulate_out_of_sample = True`

Module	Refer to a python module in one of the following ways:
	• The name of a python module (such as *prescient.simulator.prescient*)
	• The path to a python file (such as *prescient/simulator/prescient.py*)
		In addition to the two string options available to the command-line, code may also use a python module object. For example:
		`import my_custom_data_provider`
		`config.data_provider = my_custom_data_provider`

1.3.3 List of Configuration Options

The table below describes all available configuration options.

Table 2: Configuration Options

Command-line Option	In-Code Configuration Property	Argument	Description

--config-file	config_file	Path. Default=None.	Path to a file holding configuration options. Can be absolute or relative. Cannot be set in code directly on a configuration object, but can be passed to a configuration object's *parse_args()* function:
			`p = Prescient()`
			`p.config.parse_args(["--config-file", "my-config.txt"])`
			See *Launch with runner.py* for a description of configuration file syntax.

General Options			

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
--start-date	start_date	Date. Default=2020-01-01.	The start date for the simulation.
--num-days	num_days	Integer. Default=7	The number of days to simulate.
Data Options			
--data-path or --data-directory	data_path	Path. Default=input_data.	Path to a file or folder where input data is located. Whether it should be a file or a folder depends on the input format. See *Input Data*.
--input-format	input_format	String. Default=dat.	The format of the input data. Valid values are *dat* and *rts_gmlc*. Ignored when using a custom data provider. See *Input Data*.
--data-provider	data_provider	Module. Default=No custom data provider.	A python module with a custom data provider that will supply data to Prescient during the simulation. Don't specify this option unless you are using a custom data provider; use data_path and input_format instead. See *Custom Data Providers*.
--output-directory	output_directory	Path. Default=outdir.	The path to the root directory to which all generated simulation output files and associated data are written.
RUC Options			
--ruc_every-hours	ruc_every_hours	Integer. Default=24	How often a RUC is executed, in hours. Default is 24. Must be a divisor of 24.
--ruc-execution-hour	ruc_execution_hour	Integer. Default=16	Specifies an hour of the day the RUC process is executed. If multiple RUCs are executed each day (because *ruc_every_hours* is less than 24), any of the execution times may be specified. Negative values indicate hours before midnight, positive after.
--ruc-horizon	ruc_horizon	Integer. Default=48	The number of hours to include in each RUC. Must be >= *ruc_every_hours* and <= 48.
--ruc-prescience-hour	ruc_prescience_hour	Integer. Default=0.	The number of initial hours of each RUC in which linear blending of forecasts and actual values is done, making some near-term forecasts more accurate.
--run-ruc-with-next-day-data	run_ruc_with_next_day_data	Flag. Default=false.	If false (the default), never use more than 24 hours of forecast data even if the RUC horizon is longer than 24 hours. Instead, infer values beyond 24 hours. If true, use forecast data for the full RUC horizon.

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
--simulate-out-of-sample	simulate_out_of_sample	Flag. Default=false.	If false, use forecast input data as both forecasts and actual values; the actual value input data is ignored. If true, values for the current simulation time are taken from the actual value input, and actual values are used to blend near-term values if *ruc_prescience_hour* is non-zero.
--ruc-network-type	ruc_network_type	String. Default=ptdf.	Specifies how the network is represented in RUC models. Choices are: * ptdf – power transfer distribution factor representation * btheta – b-theta representation
--ruc-slack-type	ruc_slack_type	String. Default=every-bus.	Specifies the type of slack variables to use in the RUC model formulation. Choices are: * every-bus – slack variables at every system bus * ref-bus-and-branches – slack variables at only reference bus and each system branch
--deterministic-ruc-solver	deterministic_ruc_solver	String. Default=cbc.	The name of the solver to use for RUCs.
--deterministic-ruc-solver-options	deterministic_ruc_solver_options	String. Default=None.	Solver options applied to all RUC solves.
--ruc-mipgap	ruc_mipgap	Float. Default=0.01.	The mipgap for all deterministic RUC solves.
--output-ruc-initial-conditions	output_ruc_initial_conditions	Flag. Default=false.	Print initial conditions to stdout prior to each RUC solve.
--output-ruc-solutions	output_ruc_solutions	Flag. Default=false.	Print RUC solution to stdout after each RUC solve.
--write-deterministic-ruc-instances	write_deterministic_ruc_instances	Flag. Default=false.	Save each individual RUC model to a file. The date and time the RUC was executed is indicated in the file name.
--deterministic-ruc-solver-plugin	deterministic_ruc_solver_plugin	Module. Default=None.	If the user has an alternative method to solve RUCs, it should be specified here, e.g., my_special_plugin.py. **Note:** This option is ignored if --*simulator-plugin* is used.
SCED Options			
--sced-frequency-minutes	sced_frequency_minutes	Integer. Default=60.	How often a SCED will be run, in minutes. Must divide evenly into 60, or be a multiple of 60.
--sced-horizon	sced_horizon	Integer. Default=1	The number of time periods to include in each SCED. Must be at least 1.

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
--run-sced-with-persistent-forecast-errors	run_sced_with_persistent_forecast_errors	Flag. Default=false.	If true, then values in SCEDs use persistent forecast errors. If false, all values in SCEDs use actual values for all time periods, including future time periods. See *Forecast Smoothing*.
--enforce-sced-shutdown-ramprate	enforce_sced_shutdown_ramprate	Flag. Default=false.	Enforces shutdown ramp-rate constraints in the SCED. Enabling this option requires a long SCED look-ahead (at least an hour) to ensure the shutdown ramp-rate constraints can be statisfied.
--sced-network-type	sced_network_type	String. Default=ptdf.	Specifies how the network is represented in SCED models. Choices are: * ptdf – power transfer distribution factor representation * btheta – b-theta representation
--sced-slack-type	sced_slack_type	String. Default=every-bus.	Specifies the type of slack variables to use in SCED models. Choices are: * every-bus – slack variables at every system bus * ref-bus-and-branches – slack variables at only reference bus and each system branch
--sced-solver	sced_solver	String. Default=cbc.	The name of the solver to use for SCEDs.
--sced-solver-options	sced_solver_options	String. Default=None.	Solver options applied to all SCED solves.
--print-sced	print_sced	Flag. Default=false.	Print results from SCED solves to stdout.
--output-sced-initial-conditions	output_sced_initial_conditions	Flag. Default=false.	Print SCED initial conditions to stdout prior to each solve.
--output-sced-loads	output_sced_loads	Flag. Default=false.	Print SCED loads to stdout prior to each solve.
--write-sced-instances	write_sced_instances	Flag. Default=false.	Save each individual SCED model to a file. The date and time the SCED was executed is indicated in the file name.
Output Options			
--disable-stackgraphs	disable_stackgraphs	Flag. Default=false.	Disable stackgraph generation.
--output-max-decimal-places	output_max_decimal_places	Integer. Default=6.	The number of decimal places to output to summary files. Output is rounded to the specified accuracy.
--output-solver-logs	output_solver_logs	Flag. Default=false.	Whether to print solver logs to stdout during execution.
Miscellaneous Options			

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
--reserve-factor	reserve_factor	Float. Default=0.0.	The reserve factor, expressed as a constant fraction of demand, for spinning reserves at each time period of the simulation. Applies to both RUC and SCED models.
--no-startup-shutdown-curves	no_startup_shutdown_curves	Flag. Default=False.	If true, then do not infer startup/shutdown ramping curves when starting-up and shutting-down thermal generators.
--symbolic-solver-labels	symbolic_solver_labels	Flag. Default=False.	Whether to use symbol names derived from the model when interfacing with the solver.
--enable-quick-start-generator-commitment	enable_quick_start_generator_commitment	Flag. Default=False.	Whether to allow quick start generators to be committed if load shedding would otherwise occur.
Market and Pricing Options			
--compute-market-settlements	compute_market_settlements	Flag. Default=False.	Whether to solve a day-ahead market as well as real-time market and report the daily profit for each generator based on the computed prices.
--day-ahead-pricing	day_ahead_pricing	String. Default=aCHP.	The pricing mechanism to use for the day-ahead market. Choices are: * LMP – locational marginal price * ELMP – enhanced locational marginal price * aCHP – approximated convex hull price.
--price-threshold	price_threshold	Float. Default=10000.0.	Maximum possible value the price can take. If the price exceeds this value due to Load Mismatch, then it is set to this value.
--reserve-price-threshold	reserve_price_threshold	Float. Default=10000.0.	Maximum possible value the reserve price can take. If the reserve price exceeds this value, then it is set to this value.
Plugin Options			
--plugin	plugin	Module. Default=None.	Python plugins are analyst-provided code that Prescient calls at various points in the simulation process. See *Customizing Prescient with Plugins* for details. After Prescient has been initialized, the configuration object's *plugin* property holds plugin-specific setting values.
--simulator-plugin	simulator_plugin	Module. Default=None.	A module that implements the engine interface. Use this option to replace methods that setup and solve RUC and SCED models with custom implementations.

1.4 Input Data

1.4.1 Custom Data Providers

1.5 Results and Statistics Output

Under Construction

Documentation coming soon

1.6 Customizing Prescient with Plugins

Under Construction

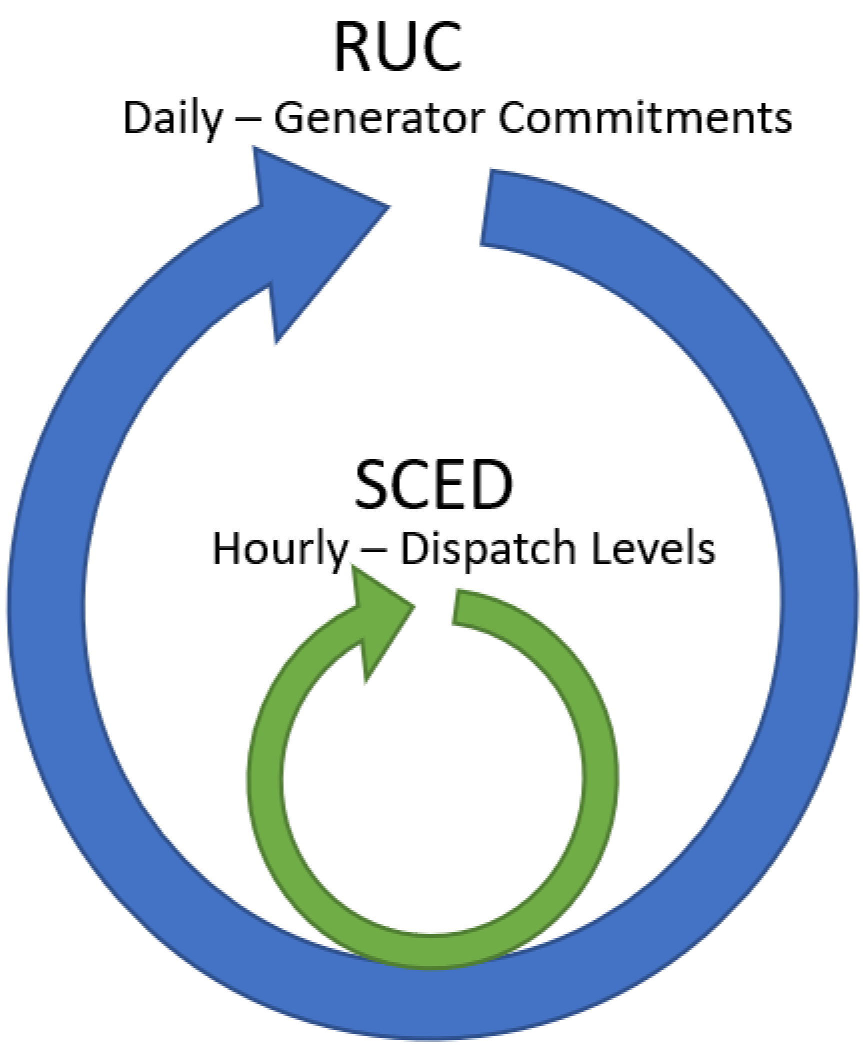
Documentation coming soon

CHAPTER

TWO

MODELING CONCEPTS

2.1 The Prescient Simulation Cycle



Prescient simulates the operation of a power generation network throughout a study horizon, finding the set of operational choices that satisfy demand at the lowest possible cost.

A Prescient simulation consists of two repeating cycles, one nested in the other. The outer cycle is the Reliability Unit Commitment (RUC) planning cycle, which schedules changes in dispatchable generators' online status during the cycle's period. The inner, more frequent cycle is the Security Constrained Economic Dispatch (SCED) cycle, which determines dispatch levels for dispatchable generators.

2.1.1 The RUC Cycle

The RUC cycle periodically generates a RUC plan. A RUC plan consists of two types of data: a unit commitment schedule and, optionally, a pricing schedule (when *compute-market-settlements* is True). The unit commitment schedule indicates which dispatchable generators should be activated or deactivated during upcoming time periods. The pricing schedule sets the contract price for expected power delivery and for reserves (ancillary service products). The RUC plan reflects the least expensive way to satisfy predicted loads while honoring system constraints.

A new RUC plan is generated at regular intervals, at least once per day. A new RUC plan always goes into effect at midnight of each day. If more than one RUC plan is generated each day, then additional RUC plans take effect at equally spaced intervals. For example, if 3 RUC plans are generated each day, then one will go into effect at midnight, one at 8:00 a.m., and one at 4:00 p.m. Each RUC plan covers the time period that starts when it goes into effect and ends just as the next RUC plan becomes active.

A RUC plan is based on the current state of the system at the time the plan is generated (particularly the current dispatch and up- or down-time for dispatchable generators), and on forecasts for a number of upcoming time periods. The forecasts considered when forming a RUC plan must extend at least to the end of the RUC's planning period, but typically extend further into the future in order to avoid poor choices at the end of the plan ("end effects"). The amount of time to consider when generating a RUC plan is known as the RUC horizon. A commonly used RUC horizon is 48 hours.

The simulation can be configured to generate RUC plans some number of hours before they take effect. This is done by specifying a time of day for one of the plans to be generated. The gap between the specified generation time and the next time a RUC plan is scheduled to take effect is called the RUC gap. Each RUC plan still covers the expected time period, from the time the plan takes effect until the next RUC plan takes effect, but its decisions will be based on what is known at the time the RUC plan is generated.

2.1.2 The SCED Cycle

The SCED process selects dispatch levels for all active dispatchable generators in the current simulation time period. Dispatch levels are determined using a process that is very similar to that used to build a RUC plan. The current state of the system, together with forecasts for a number of future time periods, are examined to select dispatch levels that satisfy current loads and forecasted future loads at the lowest possible cost.

The SCED cycle is more frequent than the RUC cycle, with new dispatch levels selected at least once an hour. The SCED honors unit commitment decisions made in the RUC plan; whether each generator is committed or not is dictatated by the RUC schedule currently in effect.

Costs are also determined with each SCED, based on dispatchable generation selected by the SCED process, the commitment and start-up costs as selected by the associated RUC process, as well as current actual demands and non-dispatchable generation levels.

2.2 Time Series Data Streams

Prescient uses time series data from two data streams, the real-time stream (i.e., actuals) and the forecast stream. As their names imply, the real-time stream includes data that the simulation should treat as actual values that occur at specific times in the simulation, and the forecast stream includes forecasts for time periods that have not yet occured in the simulation.

Both streams consist of time-stamped values for loads and non-dispatchable generation data.

2.2.1 Real-Time Data (Actuals)

The real-time data stream provides data that the simulation should treat as actual values. Real-time values are typically used only when the simulation reaches the corresponding simulation time.

Real-time data can be provided at any time interval. The real-time data interval generally matches the SCED interval (see *sced-frequency-minutes*), but this is not a requirement. If the SCED interval does not match the real-time interval then real-time data will be interpolated or discarded as needed to match the SCED interval.

2.2.2 Forecasts

Forecast data are provided by the forecast data stream. The frequency of data provided through the forecast stream must be hourly.

New forecasts are retrieved each time a new RUC plan is generated. The forecasts retrieved in a given batch are those required to satisfy the RUC horizon (see *ruc-horizon*), starting with the RUC activation time.

Forecast Smoothing

As forecasts are retrieved from the forecast data stream, they may be adjusted so that near-term forecasts are more accurate than forecasts further into the future. This serves two purposes: first, to avoid large jumps in timeseries values due to inaccurate forecasts; and second, to model how forecasts become more accurate as their time approaches.

The number of forecasts to be smoothed is determined by the *ruc-prescience-hour* configuration option. Values for the current simulation time are set equal to their actual value, ignoring data read from the forecast stream. Values for `ruc-prescience-hour` hours after the current simulation time are set equal to data read from the forecast stream. Between these two times, values are a weighted average of the values provided by the actuals and forecast data streams. The weights vary linearly with where the time falls between the current time and the ruc prescience hour. For example, if `ruc-prescience-hour` is 8, then the adjusted forecast for 2 hours after the current simulation time will be `0.25*forecast + 0.75*actual`.

Note that blending weights are determined relative to the current simulation time when the RUC is generated, not relative to the time the RUC goes into effect.

Real-Time Forecast Adjustments

Forecasts are adjusted further each time a SCED is run. This is done by comparing the forecast for the current time with the actual value for the current time. The ratio of these two values is calculated, then used as a scaling factor for forecast values. For example, if the forecast for a value was 10% too high, all future forecasts for the same value are reduced by 10%.

Note: If *run-sced-with-persistent-forecast-errors* is false, then SCEDs will use actual values for all time periods. Forecasts will still be used for RUCs, but SCEDs will be based entirely on actual values, even for future time periods.

2.3 Reserves and Ancillary Services

2.4 Energy Markets and Pricing

CHAPTER

THREE

EXAMPLES AND TUTORIALS

CHAPTER

FOUR

REFERENCE

4.1 File Formats

4.1.1 RTS-GMLC

This is the main input format.

4.1.2 Pyomo DAT Files

Old way to do it.

4.2 Python Classes and Functions

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search