Prescient

Release 2.0.2

Prescient Developers

Dec 15, 2021

CONTENTS:

CHAPTER

ONE

USING PRESCIENT

1.1 Installation

Prescient is a python package with a number of dependencies and prerequisites.

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 Get Prescient source code

The latest stable 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.

1.1.3 Install dependencies

Prescient runs in a python environment that must include a number of python prerequisites. You may want to create a python environment specifically for Prescient. To create a new Anaconda environment that includes Prescient's prerequisites, 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.

1.1.4 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 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.

1.1.5 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*).

1.1.6 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 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 the Prescient python package*). 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* module

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

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

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

```
python -m prescient.simulator.prescient --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
```

(continues on next page)

(continued from previous page)

Configuration options can also be specified in a configuration file:

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

Note that if you use the *–config-file* option, it must be the only option on the command line.

Running the *prescient* 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:

```
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,
        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. Another option 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
```

(continues on next page)

(continued from previous page)

```
config.ruc_horizon=36
config.enforce_sced_shutdown_ramprate=True
config.no_startup_shutdown_curves=True

p.simulate()
```

Managing configuration in code is very flexible. 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'
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	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. If specified, no other command line options or function arguments are allowed.
General Options			
--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.

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
--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.
--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			

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
--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.
--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 *Future Values in SCEDs*.
--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.

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
Miscellaneous Options			
--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.

continues on next page

Table 2 – continued from previous page

Command-line Option	In-Code Configuration Property	Argument	Description
--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

Note: This was taken from a previous write-up and needs to be revisited.

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

Prescient loops through two repeating phases, the reliability unit commitment (RUC) phase and the security constrained economic dispatch (SCED) phase. The RUC phase determines which dispatchable generators will be active in upcoming operational time periods. For each operational period within a RUC cycle, the SCED phase selects the dispatch level of each committed thermal generator.

The RUC phase occurs one or more times per day. Each time the RUC phase occurs, Prescient generates a unit commitment schedule that indicates which generators will be brought online or taken offline within the RUC's time horizon. The SCED phase occurs one or more times per hour. Each SCED selects a thermal dispatch level for each committed generator.

2.1.1 The RUC Phase

More detailed description of the RUC...

The RUC phase occurs one or more times per day. Each time the RUC phase occurs, Prescient generates a unit commitment schedule that indicates which generators will be brought online or taken offline within the RUC's time horizon. The RUC schedule may begin immediately, or it may begin a number of hours after the RUC is generated.

2.1.2 The SCED Phase

More detailed description of the SCED, including a high level description of the optimization problem being solved, and possibly a conversational description of some things that can be tweaked (such as how often a SCED runs).

Future Values in SCEDs

Warning: Coming soon.

2.2 Reserves and Ancillary Services

2.3 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