The Switch Model:

Installation, Usage and Documentation

by Rodrigo Henríquez

OCM Lab
Departament of Electrical Engineering,
Pontificia Universidad Católica de Chile.

June 30, 2017

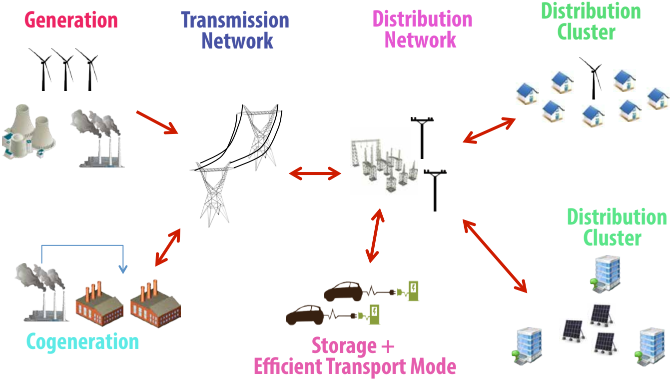
Outline

Outline

Introduction

Global goals and evolution of systems

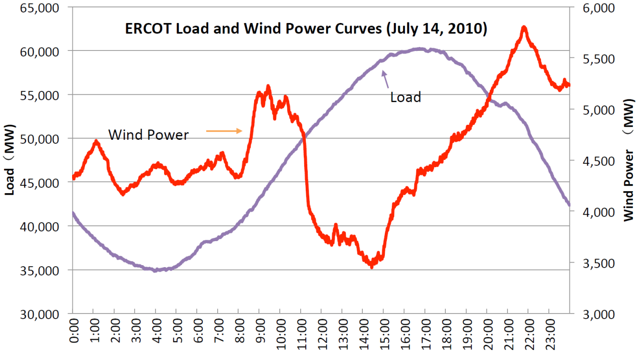
- There is a need for moving towards energy systems less dependant on fossil fuels.
- Power and energy systems are quickly evolving towards that goal.



Introduction

Global goals and evolution of systems

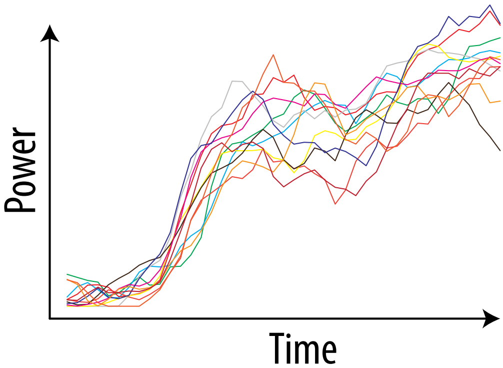
- New technologies and policies are required to fight against climate change.
- Understanding the potential impact of policy changes and new technological paradigms is of key interest for regulators, industry stakeholders and researchers.



Introduction

New tools for energy modelling

- Evolution of electricity systems is projected using capacity-expansion models.
- New important technical and economic challenges due to the volatile and hard-to-predict nature of Renewable Energy Sources.

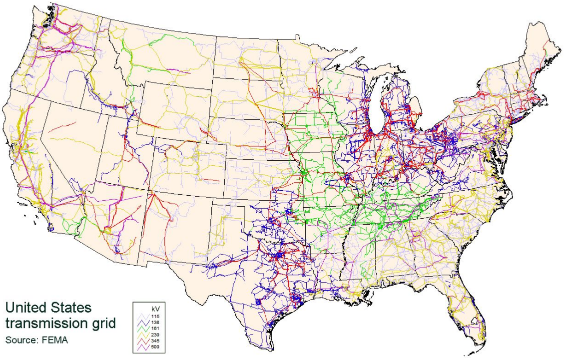


Introduction

New tools for energy modelling

Models must handle large amounts of data and output scenarios, that creates several challenges:

- Tuning time and space scales
- Representing transmission network
- Considering uncertainty sources
- Address the human dimension



Outline

The framework

The Switch platform

- Switch is a platform freely distributed online as a Python package, written in Pyomo language, that allows users to build and solve power systems models to perform investment planning, production cost simulations and economic and policy analyses.
- It is available as a GitHub repository `https://github.com/switch-model/switch` and listed on the Python Package Index (PyPI) as *switch-model*.
- It has a Modular structure, in which analysts can choose the appropriate level of complexity and specific features for their studies.

The framework

The Switch platform: Modular structure

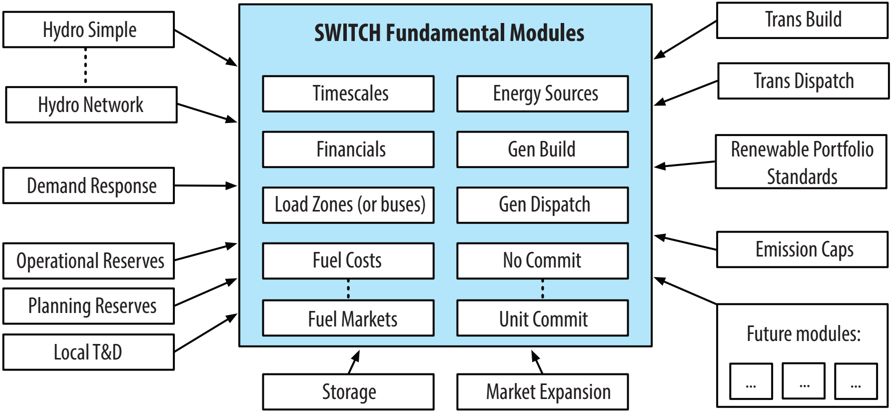


Figure: Modular formulation of Switch. Dotted lines represents either-or choices.

Installing Switch

Before using Switch, these softwares must be installed:

- **Python 2.7:** It is recommended installing Anaconda `https://www.continuum.io/downloads`. Be sure to choose Anaconda 2, since Switch is not compatible with Python 3.6.
- **MILP Solver:** A solver like GLPK, CPLEX, Gurobi or others must be added to the system path in order to solve optimization problems.
- In Windows, it is recommended to install **git** `https://git-for-windows.github.io/`, in order to obtain Switch updates directly from the github and using **Git Bash** as a terminal instead of `cmd`.

Installing Switch

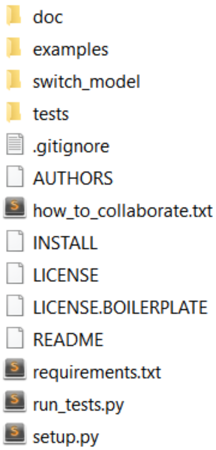
There are two ways for installing Switch, but the most recommended is the following one:

- Go to the Github package: https://github.com/switch-model/switch and clone the repository (download in zip).
- Extract in a folder the zip file (recommended in your User folder)
- Use the following commands in a terminal:

```
cd switch-master
pip install --upgrade --editable .
```

- The previous pip install command adds the switch_model package to your Python installation. It also installs the Pyomo Python package, which Switch uses to define and solve optimization models.

File structure



- Folder **examples** contains 17 different examples useful to understand the usage of different modules and input files.
- Folder **switch_model** contains all the scripts that define every module of Switch.
- Other files are for installation, documentation and testing.

Checking Installation

The Switch model has available a bunch of tests that will check if your installation was succesful. For that purpose in the root folder (switch-master) run the following command:

```
python run_tests.py
```

If everything went fine, you should see that 40 tests were ran, and ended with an OK.

Running examples

To run an example, navigate to an example directory and run the command:

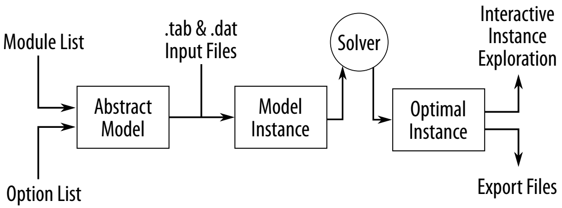
```
switch solve --verbose --log-run
```

For example, running 3zone_toy problem:

```
Rodrigo@DESKTOP-HGEFF59 MINGW64 ~/switch-fork/switch/examples/3zone_toy (master)
$ switch solve --verbose --log-run
logging run to logs\2017-06-22_16-26-53.log

=======================================================================
SWITCH model created in 0.02 s.
Arguments:
verbose=True, outputs_dir='outputs', logs_dir='logs', solver='glpk', inputs_dir='inputs', solver_manager='serial', log_run_to_file=
True
Modules:
switch_model, switch_model.timescales, switch_model.financials, switch_model.balancing.load_zones, switch_model.energy_sources.prop
erties, switch_model.generators.core.build, switch_model.generators.core.dispatch, switch_model.reporting, switch_model.transmissio
n.local_td, switch_model.generators.core.no_commit, switch_model.energy_sources.fuel_costs.markets, switch_model.transmission.trans
port.build, switch_model.transmission.transport.dispatch, switch_model.solve
=======================================================================

Loading inputs...
Inputs loaded in 0.09 s.


Solving model...
Solved model. Total time spent in solver: 0.098000 s.
Optimization termination condition was optimal.

Executing post solve functions...
Post solve processing completed in 0.03 s.
```

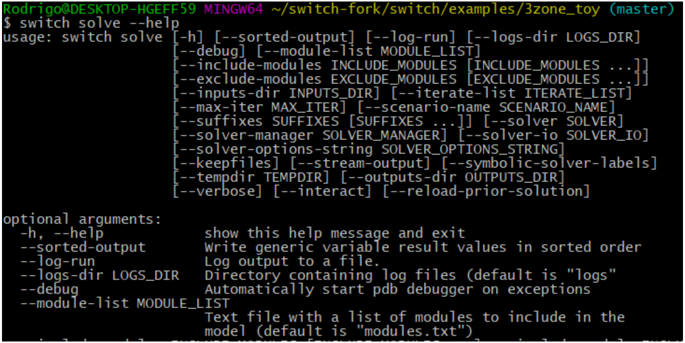
Constructing a scenario

1. To solve a scenario, Switch defines an *abstract model* based on the selected module list (located on inputs/modules.txt) and other options provided from command line.
2. Next, each module in the list is provided an opportunity to read input data. These are used to instantiate the abstract model into a *model instance*.
3. Then Pyomo generates an optimization problem in standard matrix form, which is passed to the solver along with any user-defined solver options.



Checking options of Switch

Running the command: `switch solve --help` provides information on how to configure several options of Switch:



```
Rodrigo@DESKTOP-HGEFF59 MINGW64 ~/switch-fork/switch/examples/3zone_toy (master)
$ switch solve --help
usage: switch solve [-h] [--sorted-output] [--log-run] [--logs-dir LOGS_DIR]
                    [--debug] [--module-list MODULE_LIST]
                    [--include-modules INCLUDE_MODULES [INCLUDE_MODULES ...]]
                    [--exclude-modules EXCLUDE_MODULES [EXCLUDE_MODULES ...]]
                    [--inputs-dir INPUTS_DIR] [--iterate-list ITERATE_LIST]
                    [--max-iter MAX_ITER] [--scenario-name SCENARIO_NAME]
                    [--suffixes SUFFIXES [SUFFIXES ...]] [--solver SOLVER]
                    [--solver-manager SOLVER_MANAGER] [--solver-io SOLVER_IO]
                    [--solver-options-string SOLVER_OPTIONS_STRING]
                    [--keepfiles] [--stream-output] [--symbolic-solver-labels]
                    [--tempdir TEMPDIR] [--outputs-dir OUTPUTS_DIR]
                    [--verbose] [--interact] [--reload-prior-solution]

optional arguments:
  -h, --help            show this help message and exit
  --sorted-output       Write generic variable result values in sorted order
  --log-run             Log output to a file.
  --logs-dir LOGS_DIR   Directory containing log files (default is "logs"
  --debug               Automatically start pdb debugger on exceptions
  --module-list MODULE_LIST
                        Text file with a list of modules to include in the
                        model (default is "modules.txt")
```

Constructing your own scenario

In order to evaluate your own scenario you need to decide:

- Features you want to use: Select the appropriate modules for that purpose.
- Configuring your inputs. Those will depend on what modules you want to use. These will be `.dat` for single parameters or `.tab` files for indexed parameters.
- Configure your solver and other options through the command line.

Outline

Modules in Switch

- Switch have 26 modules that are used for defining several characteristics in a power system study.
- The core modules defines essential parts of the model:
  - Time scales
  - Objective function and financial characteristics
  - Load zones
  - Generators characteristics
- Other regional modules (Hawaii) are available to use in case of being necessary.

Timescales

In this module, different timescales are defined to be used on the model. Switch use 3 different timescales to take in account temporal dimension in various scales in the problem. These are:

- **Periods:** Multi-year time scales to describe investment decisions.
- **Timeseries:** Block of consecutive time points within a period. Usually represnt a single day, a week, a month or a entire year.
- **Timepoints:** Describe unique time points within a time series. Usually represents an hour and are used to index parameters such as electricity demand and renewable availability.

These sets are loaded in three inputs files: `periods.tab`, `timeseries.tab` and `timepoints.tab`.

Financials

This module defines financial parameters and functions and the objective function. These are used to:

- Bring future costs to present.
- Calculate annualized cost of investments.
- Convert future to present value.
- Determine the capital recovery factor, among others.

To support a dynamic objective function that can be defined at runtime, two dynamic lists are created:

- **Fixed costs components:** Model components that contribute to overall system costs in a period basis.
- **Variable costs components:** Model components that contribute to overall system costs in a time point basis.

Discount and interest rates are defined in the `financials.dat` input file.

Balancing: Load Zones

This module defines the set of load zones. Each load zone can represent an electric bus, or a local area that can be approximated to one central bus.

Two dynamic lists, indexed by zones and timepoints, are created:

- Components that inject power.
- Components that withdraw power.

The power balance equation is defined in this module. The sum of all elements in a list must be equal of the sum of all elements of the other list, at each load zone and timepoint.

The set of load zones is defined on the input file `load_zones.tab`, the electric demand on `loads.tab` and the expected peak demand on `zone_coincident_peak_demand.tab`.

Energy Sources: Properties

In this module, the set of all energy sources are defined for the Switch model. All generation projects will be required to define their energy source.

This will be useful to obtain the final energy mix and ensure the requirements of renewable portfolio standards.

- The set of non fueled energy sources are defined on `non_fuel_energy_sources.tab` input file.
- The set of fueled energy sources, with its CO_2 intensity are defined on `fuels.tab` input file.

Generators: Core.Build

In this module all generation projects that have been built, could be expanded or could potentially be built are defined.

A linear formulation is used to decide how much capacity is built per period, and a maximum capacity constraint is used to enforce limitations on land or resource availability.

Input data is loaded from `generation_projects_info.tab`, `gen_build_predetermined.tab` and `gen_build_costs.tab`.

Generators: Core.Gen_Discrete_Build

This module allows users to define discrete builds of generation technologies that have a discrete unit size specified.

With this scheme is possible to define specific projects if the discrete unit size is equal to the maximum available capacity of a specific project.

The unit size is specified on `generation_projects_info.tab` input file.

Generators: Core.Dispatch

This module defines model components to describe generation projects.

Variables such as the dispatch and fuel use rate and parameters such as the ratio of total capacity available for renewable projects. This module requires either `no_commit` or `unitcommit` in order to constraint projects's dispatch.

- Expected availability of thermal projects and other parameters are defined on `generation_projects_info.tab` input file.
- Ratio of total capacity for renewable projects at each time point are defined on `variable_capacity_factors.tab` input file.

Energy Sources: Fuel_Costs.Simple

This module defines a simple description of flat fuel costs for Switch.

- Allows to define the cost of the fuel in MMBTU/h for each fuel, load zone and period.
- Allows to define a set of load zones and periods in which a fuel will not be available.

Fuel costs are specified on `fuel_cost.tab` input file.

Energy Sources: Fuel_Costs.Markets

This module defines a fuel markets for the Switch model.

- Each regional fuel market has a supply curve with discrete tiers of escalating costs.
- Tiered supply curves are flexible format that allows anything from a flat cost in every load zone with no limits on consumption, to a detailed supply curve of fuel for each load zone.

Fuel markets inputs are defined on `regional_fuel_markets.tab`, `fuel_supply_curves.tab`, `zone_to_regional_fuel_market.tab` and `zone_fuel_cost_diff.tab`.

Transmission: Transport.Build

This module defines bulk transmission lines of an electric grid, that connect different load zones.

- Each line will connect two load zones, defining the distance between them and the efficiency of the line.
- The efficiency parameter can be used to model losses in power flows.
- Maximum invested capacity per period can be considered to limit the expansion of transmission lines.

Transmission lines inputs are defined on `transmission_lines.tab`, `trans_optional_params.tab` and `trans_params.dat`.

Transmission: Transport.Dispatch

This module defines a Transportation Network Model, for defining the dispatch and its limits of each transmission line.

- A set of directed transmission line is constructed with the ordered combinations of load zones.
- For each transmission line there are two variables for dispatching flows in each direction.
- Receiving flows are the only ones affected by the transmission efficiency.

Generators: Core.No_Commit

This module defines simple limitations on project dispatch, without considering a detailed unit commitment.

- Limits the dispatch subject to the available capacity of each project.
- Available capacity will be different at each time point for variable renewable projects.
- Fuel usage is based on a full load heat rate.

Generators: Core.Commit.Operate

This module describes a linear Unit Commitment (UC) of generation projects. This module is mutually exclusive with the `no_commit` module.

- Variables for commit, start-up and shut-down.
- Fuel costs for starting up capacity.
- Dispatch will be limited by the commit decision. Commit decision will be limited by the available capacity.
- Consistency between commit, start-up and shut-down variables.
- Minimum up and down times rules are available.

Parameters are defined on `generation_projects_info.tab` and `gen_timepoint_commit_bounds.tab` input files

Generators: Core.Commit.Discrete

This module allows users to define a discrete unit commitment.

- Defines an integer variable and a discrete unit size for each generator.
- The commit variable is forced to be a multiple of its unit size.
- If the unit size is defined as the same as the maximum capacity, then a standard UC is implemented.

Generators: Core.Commit.Fuel_Use

This module describes the fuel usage with consideration of a UC and incremental heat rates using piecewise linear expressions.

- Each line segment of fuel usage has an increasing slope in units MMBTU/MWh when more power is being generated.
- Fuel usage variable is greater than or equal to each segment line.
- In a cost minimizing problem, the fuel usage variable will be pushed till it touch one of the segments.
- It is required that heat rates increase with the energy production, so segments form a convex boundary.

Input data is defined on the `gen_inc_heat_rates.tab` file.

Transmission: Local_TD

This module defines model components to describe local transmission and distribution (T&D) build-outs.

- Adds a virtual “distribution node” to each load zone, that is connected to the zone’s central node.
- Each distribution pathway has a efficiency that represents distribution losses.
- DERs can be included inside the distribution node, that will impact the energy balance at the distribution node, avoiding losses from the central network.
- For now, the model is not allowed to inject power from the distribution node to the central grid.

Input data is also defined on the `load_zones.tab` file.

Balancing: Demand_Response.Simple

This modules describes a simple Demand Response (DR) Shift Service for Switch.

- A maximum increase and decrease in demand must be specified per load zone and time point.
- The total energy shifted at each timeseries must sum zero.
- No cost is considered for shifting energy.

Input data is defined on the `dr_data.tab` file.

Generators: Extensions.Hydro_Simple

This module defines a simple hydro electric model to dispatch reservoir-based hydro plants.

- Dispatch must be greater than a minimum level.
- The average dispatch at each time series must be equal to a predefined parameter.

Input data is defined on the `hydro_timeseries.tab` file.

Generators: Extensions.Hydro_System

This module creates a hydraulic system that works in parallel with the electric one. Both systems are linked through the power generation process at hydro-based generators.

- The water network topology must be specified. This includes water nodes, reservoirs, water connection and hydro generation projects.
- Nodes can have inflows and consumption that are independent of the electrical system.
- Connections can control flow between nodes, limited by flow constrains or geological filtration.
- Cascading generation and spilling are allowed in the formulation.

Parameters are defined on `water_nodes.tab`, `reservoirs.tab`, `water_connections.tab`, `hydro_generation_projects.tab`, `water_node_tp_flows`, `reservoir_tp_data.tab` and `min_eco_flows.tab` input files.

Generators: Extensions.Storage

This module defines storage technologies that builds on top of generic generators.

Adds component for each generator for deciding:

- How much energy to build into storage.
- When to charge and discharge.
- Limits on charging and discharging.
- Energy accounting.

Parameters are defined on the same files of generators. These are `generation_projects_info.tab` and `gen_build_costs.tab`.

Policies: RPS_Simple

This module defines a simple Renewable Portfolio Standard (RPS) policy scheme for Switch model.

- All non-fuel energy sources are assumed to be RPS-eligible.
- Dispatched electricity that is generated by RPS-eligible sources must meet the energy goal.
- The energy goal is set as a required percentage of the total demand in that period.

Parameters are defined on `rps_targets.tab` and `fuels.tab` input files.

Policies: Carbon_Policies

This module adds emission policies to the Switch model.

- It could be in the form of emission caps
- It could be in the form of added cost, that could represent social cost of carbon, clearing price of a cap-and-trade carbon market or a carbon tax.

Input data is defined on `carbon_policies.tab` input file.

Balancing: Unserved_Load

This module defines components to allow leaving some load unserved at each load zone.

A cost penalty of unserved load must be defined in units of \$/MWh. It can be defined on the `lost_load_cost.dat` and defaults to a value of 500 \$/MWh if it is not specified.

Balancing: Planning_Reserves

This module defines planning reserve requirements (prr) to support resource adequacy requirements.

- Different elements contributes to available capacity or requirements for capacity.
- Generation projects and transmission flows contributes to available capacity.
- Demand contributes to requirements for capacity.
- The available capacity at each timepoint has to be greater than the requirements for capacity.

Parameters are defined on `reserve_capacity_value.tab`, `planning_reserve_requirements_zones.tab`, `generation_projects_info.tab` and `planning_reserve_requirements_zones.tab` input files.

Balancing: Operating_Reserves.Areas

This module defines the balancing areas in which operating reserve requirements will be considered.

A balancing area can represent a single load zone, or a collection of them, where operating reserves are considered.

Balancing areas are defined on `load_zones.tab` input file.

Balancing: Operating_Reserves.Spinning_Reserves

This module defines a set of rules for considering spinning reserves for up and down requirements.

- Demand and variable renewable projects contributes to requirements for spinning reserves (3%+5%).
- $N-1$ contingencies (maximum contingency variable) contributes to requirements for up spinning reserves.
- Slack capacity (difference between capacity and commit) of conventional generation projects contributes to available spinning reserves.

Parameters are defined on `generation_projects_info.tab` and `spinning_reserve_params.dat` input files.