Sustainable Energy Systems Integration & Transitions (SESIT) group

Strategic Integration of Large-capacity Variable Energy Resources (SILVER)

User Manual

Table of Contents

Abbreviations

CF – Capacity factor
CSV – Comma-separated values
DR – Demand response
ED – Economic dispatch
EV – Electric vehicle
HDF – Hierarchical data format
MW – Megawatt
MWh – Megawatt-hour
OPF – Optimal power flow
PHS – Pumped hydro storage
PV – Solar photovoltaics
RT_OPF – Real-time optimal power flow
SILVER – Strategic Integration of Large-capacity Variable Energy Resources
UC – Unit commitment
VRE – Variable renewable energy

1. Introduction

SILVER is a new scenario-based electricity system model that explores the trade-offs among alternative balancing strategies for high variable renewable energy (VRE) electricity grids. SILVER optimizes the asset dispatch for a user-defined electricity system configuration that specifies demand response availability, generation assets, storage assets, and transmission infrastructure. The SILVER model comprises four modules: the long-term scenario planning framework, the day-ahead network-constrained price-setting dispatch, day-ahead unit commitment, and real-time optimal power flow dispatch [1]. In the first step, different system design scenarios are generated considering network and generation expansion, and renewable energy resources integration. Second step is performed to determine the marginal price of system operation. In this step, an optimal power flow (OPF) problem is solved to find the marginal price [2]. The marginal price is used in the third step. Based on system design (first step output) and marginal prices (second step output), a unit commitment (UC) problem is solved in the third step to find the optimal operation of the system with the aim of cost minimization [2]. The output of this step is optimal scheduling of units (generators, responsive loads, electric vehicles and storages). In the fourth step, a real time OPF problem is implemented to compensate for the difference between real time and forecasted amount of renewable energy resources output power and loads.

SILVER has been applied to twelve Ontario-based scenarios to explore the implications of high VRE penetrations in a 100% renewable scenario [1]. The role that storage, demand response, electric vehicles, and transmission expansion might play in balancing VRE is analyzed. The results highlight the operational differences between balancing options: demand response, with a fixed price remuneration, is dispatched in proportion to net load curve variability, whereas storage technologies which take advantage of price arbitrage, are dispatched in proportion to marginal cost variability. GHG emissions fall with increasing VRE penetrations; however, there are local maxima when natural gas replaces nuclear generation, which interrupt this trend. Weak wind resources in the summer necessitate significant non-VRE capacity, which increases capital costs for higher VRE penetrations [1].

SILVER has also been applied to evaluate deploying storage assets to facilitate variable renewable energy integration. This study assessed the impacts of grid flexibility, renewable penetration, and market structure [3]. Finally, SILVER was implemented for a case study in Lusaka, Zambia to study planning for variable renewable energy and EV integration under varying degrees of decentralization [4].

The SILVER model has been written in the Python programming language. Python is an interpreted, high-level, general-purpose programming language. The language constructs an object-oriented approach which aims to help programmers write clear, logical code for small and large-scale projects.

This document provides a general description of SILVER code and its implementation.

2. Implementation of the SILVER code

2.1 Prerequisites

To run SILVER, there are some prerequisites. In what follows, these prerequisites include:

1. Anaconda (if you are comfortable managing your own python environments this is not required)
2. Python 3.7
3. Solver (GPLK or CPLEX)
4. Excel or equivalent software to view and edit .csv and .xlsx files

2.1.1 Python3 installation

To run the SILVER framework in Windows, it is necessary that **python version 3.7** be installed.

2.1.2 Windows Installation Instructions

Install Anaconda

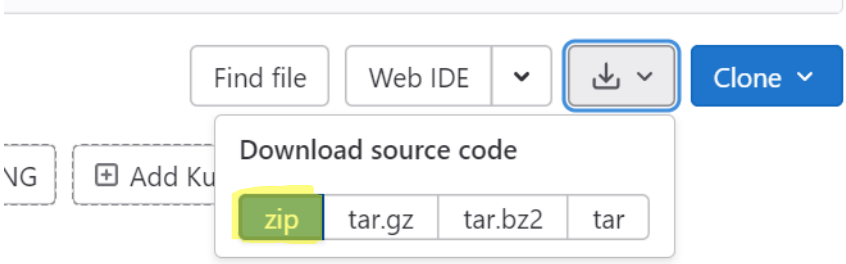
1. Install either the Anaconda or Miniconda environment.

   What is Anaconda?
   Anaconda is a distribution tool for Python and R that simplifies package management. The environment includes data science packages suitable for Windows, Linux, and macOS.

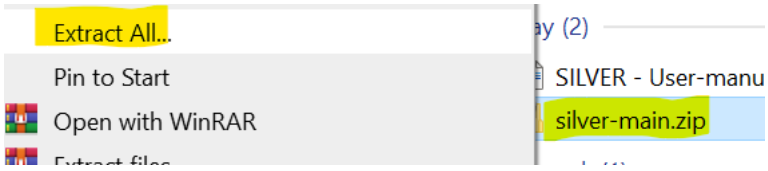
   What is Miniconda?
   Miniconda is merely a simplified bootstrap version of Anaconda.

Download the SILVER Repository

1. Go to https://gitlab.com/McPherson/SILVER
2. Download the repo in a .zip format



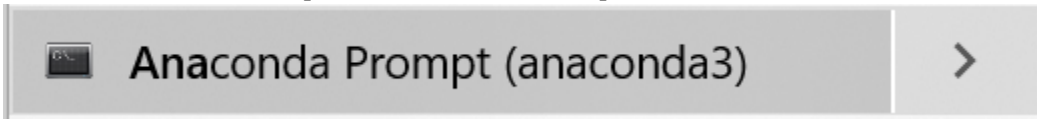
3. From there, you may export the contents to where you would like them to reside on your computer



Note: you will need to access this folder via command line for running the python executables and you will need to access this folder for configuring the

Create the Environment and Install the Requirements

1. Now that you have anaconda installed. use the windows search bar in the bottom left of your screen to search for and open the "Anaconda Prompt."



2. like you would a regular terminal navigate to the folder you just extracted using the "cd" commands. It should look something like this:

```
(base) C:\Users\evand>cd Documents

(base) C:\Users\evand\Documents>cd silver-main

(base) C:\Users\evand\Documents\silver-main>dir
 Volume in drive C is OS
 Volume Serial Number is ECE7-53F1

 Directory of C:\Users\evand\Documents\silver-main

2022-11-25  06:01 PM    <DIR>          .
2022-11-25  06:01 PM    <DIR>          ..
2022-11-25  06:01 PM             3,296 .gitignore
2022-11-25  06:01 PM               745 configCreate.py
2022-11-25  06:01 PM             1,109 LICENSE
2022-11-25  06:01 PM             1,074 README.md
2022-11-25  06:01 PM             2,115 requirements.txt
2022-11-25  06:01 PM    <DIR>          silver
2022-11-25  06:01 PM           501,575 SILVER - User-manual - Open Access Version -25 NOV 2022.pdf
2022-11-25  06:01 PM            19,815 SILVER_Common_Errors_V0-1.docx
2022-11-25  06:01 PM             4,101 silver_env.yaml
               8 File(s)        533,830 bytes
               3 Dir(s)  291,945,648,128 bytes free

(base) C:\Users\evand\Documents\silver-main>
```

3. In this directory you should see a file called **silver_env.yaml** This file will be used to configure the required python dependencies to run SILVER. To create this environment, enter the command below into the anaconda prompt

   `conda env create -f silver_env.yaml`

4. Once the environment setup is complete, Activate the new environment.
   `conda activate silver_env`

5. Anytime you want to run SILVER you will do so from this environment

Install CPLEX

Note: this software isn't required to run silver as it requires an additional license. However, it is significantly more efficient than other free to use solvers such as GLPK. Further, some scenarios can be solved by CPLEX that cannot be solved by GLPK due to complexity.
Note: Installation of this solver may require JAVA to be setup on your machine.

1. Go to https://www.ibm.com/academic/technology/data-science

2. Scroll down and go to the Software tab
3. Click Download under ILOG CPLEX Optimization Studio
4. Scroll down and check the box for IBM ILOG CPLEX Optimization Studio 20.10 for Windows x86-64 (CC8ASML)
5. Scroll further down and click I agree then Download Now.
6. You may have to install the additional IBM Download Director. You will be given instructions if that is the case.
7. Once the Download Director is open, click launch.
8. Keep the default settings the same and proceed through the pages.
9. Enter the following commands into the Anaconda prompt
    1. python "C:\Program Files\IBM\ILOG\CPLEX_Studio201\python\setup.py" build
    2. python "C:\Program Files\IBM\ILOG\CPLEX_Studio201\python\setup.py" install
10. Go back to the Environment Variables window visited earlier.
11. Add a new path to the Path variable
    1. C:\Program Files\IBM\ILOG\CPLEX_Studio201\cplex\bin\x64_win64

    2. Note for Mac users – You set the environment variables by typing:
        1. nano ~/.zshrc
            i. This opens a text file. Within that file, you add:
            ii. export PATH="/Applications/CPLEX_Studio201/cplex/bin/x86-64_osx:${PATH}"
                1. Or the path to your CPLEX installation

Fix a PYOMO solver bug

Note: if you are not using CPLEX solver you can skip this step.
Note: if CPLEX.py cannot be found in either of the locations mentioned below you may need to search elsewhere on your machine for the environment location containing the Pyomo library you installed using the yaml file.

1. Open the CPLEX.py file in a text editor
    1. Located at
       C:\Users\[your_user]\AppData\Roaming\Python\Python37\site-packages\pyomo\solvers\plugins\solvers\CPLEX.py
       Or
       C:\Users\[your_user]\AppData\Local\Programs\Python\Python37\site-packages\pyomo\solvers\plugins\solvers\CPLEX.py
2. Add the code from this GitHub issue https://github.com/Pyomo/pyomo/pull/1779/files

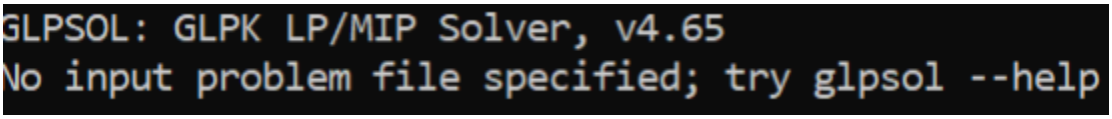
GLPK Solver for Windows 10

1. Download the latest version of the GLPK Solver
   https://sourceforge.net/projects/winglpk/
2. Extract the folder to your C: drive
3. Open the glpk folder
4. Depending on your operating system type, you will open either the 'w32' or 'w64' folder
5. Copy the address of the folder as text

6. Use the Windows search bar to find the “Edit the system environment variables” window
7. Open the “Environment Variables...” window under the “Advanced” tab
8. Select the “Path” variable under “System variables” and click “Edit...”
9. Select “New” then add the “w64” address to the bottom
10. Select “OK”
11. Open the Command Prompt
12. Enter the following command:

```
glpsol
```

If you see the following, the solver was installed successfully:



Run SILVER

1. Navigate back to the SILVER-Code folder
   cd [location of folder]\SILVER\SILVER\SILVER-CODE
2. Run the python file
   python SILVER_VER_18.py

2.1.3 Linux Installation Instructions

Download Linux Ubuntu Virtual Machine

Follow the link below for instructions on setting up a Linux Ubuntu Virtual Machine using Oracle VirtualBox:
https://brb.nci.nih.gov/seqtools/installUbuntu.html

Follow the link below to download VirtualBox by Oracle:
https://www.virtualbox.org/wiki/Downloads

Follow the link below to download Ubuntu Desktop (ISO file)
https://ubuntu.com/download/desktop

```
$ sudo apt update
$ sudo apt install build-essential zlib1g-dev \
libncurses5-dev libgdbm-dev libnss3-dev \
libssl-dev libreadline-dev libffi-dev curl
sudo apt-get install libbz2-dev
```

Download Python 3.7.3

Download Python 3.7.3 by following the link below:
https://www.python.org/downloads/release/python-373/

Scroll down to Files and click version "Gzipped source tarball"

Extract the tarball for Python installation into the Documents folder --> Should be a file named "Python-3.7.3"

Configure the script
$ cd /home/user_name/Documents/Python-3.7.3
$ ./configure

Start the build process
$ sudo make install
Verify the installation
$ python3 --version

Download the SILVER Repository

1. Go to https://gitlab.com/McPherson/SILVER
2. Download the repo in a .zip format
3. From there, you may export the contents to where you desire

Install the Requirements

It should be noted that the --user argument used with pip3 install runs the installation as an admin.
$ pip3 install --user future
$ pip3 install --user pandas

Change to the directory that contains the requirements.txt file, for myself I did the following:
$ cd /home/jgmonroe/Documents
$ pip3 install --user -r requirements.txt

Install the last remaining dependencies
$ pip3 install --user xlrd==1.2.0
$ pip3 install --user pyutilib==5.8.0

Install CPLEX

Create a folder "IBM" in your home directory --> /home/jgmonroe/IBM

Download the CPLEX Optimization Studio Free Academic Edition, not the Free Community Edition.

https://www.ibm.com/analytics/cplex-optimizer

Navigate to the folder that contains the bin file.
$ cd /home/jgmonroe/Documents/DownloadDirector
$ chmod +x ILOG_COS_20.10_LINUX_X86_64.bin
$ ./ILOG_COS_20.10_LINUX_X86_64.bin
$ sudo ./ILOG_COS_20.10_LINUX_X86_64.bin --> If you need to run it as a root (admin)

Now build and install the cplex package.
$ python3 /home/jgmonroe/IBM/python/setup.py build
$ sudo python3 /home/jgmonroe/IBM/python/setup.py install

Export the path of the CPLEX executable so the SILVER framework can identify the location of the correct solver.
$ export PATH=$PATH:/home/jgmonroe/IBM/cplex/bin/x86-64_linux

Fix a CPLEX bug

Open the CPLEX.py file and update it with the fix at the link below
$ xdg-open ~/.local/lib/python3.7/site-packages/pyomo/solvers/plugins/solvers/CPLEX.py

https://github.com/Pyomo/pyomo/pull/1779/files

Install GLPK Solver for Linux

1. Download the latest version with the .tar.gz extension
   http://ftp.gnu.org/gnu/glpk/
2. Unzip the folder
   $ gzip -d glpk-X.Y.tar.gz
3. Unarchive the distribution file
   $ tar -x < glpk-X.Y.tar
4. Navigate to the glpk folder
5. Run the configuration script
   $ ./configure
   a. If you run into issues, try this command instead
      $ ./configure --disable-shared
6. Compile the package
   $ make
7. Check the package
   $ make check
8. Install the GLPK package
   $ make install

Run SILVER

Now change the directory to the folder that executes the program and then run the program.
$ cd /home/jgmonroe/SILVER/SILVER-Code
$ python3 SILVER_VER_18.py

2.2 General description of the master SILVER framework components

The SILVER framework contains two folders: 1) SILVER-Code and 2) SILVER_Data. The SILVER-Code folder contains all the programming modules that are used to run the SILVER framework, including the Minpower package (https://pypi.org/project/minpower/). The SILVER_Data folder contains user input data files, intermediary data files that are produced and processed during model execution, and output data files including model results.

2.3 Input data

All the necessary data and parameters for the SILVER model are provided by the user. Input data can be passed to Python through Excel files and parameter setting the file configVariables.ini.

2.3.1 Base file structure (located in SILVER_Data)

```
user_inputs/
├ Hydro_Data-<scenario name>/
|  ├ hydro_daily.csv
|  ├ hydro_hourly.csv
|  ├ hydro_monthly.csv
|  └ hydro_cascade.csv
├ ImportExport_Data-<scenario name>/
|  └ importexport_hourly.csv
├ VRE_Resource_Analysis-<scenario name>/
|  ├ Solar_Generation_Data/
|  |  └ 282-96/
|  |    └ Solar_Generation_Data_282-96_2018.csv
|  └ Wind_Generation_Data/
|    └ 288-72/
|      └ Wind_Generation_Data_288-72_2018.csv
├ configVariables.ini                          [necessary]
├ model inputs - <scenario name>.xlsx          [necessary]
└ <scenario name>_Demand_Real_Forecasted.xlsx  [necessary]
```

2.3.2 Excel files (located in SILVER_Data)

These Excel files include input data to build the SILVER model for a case study. Generally, input data includes information for the transmission network, generators and loads.

Descriptions of the input files are provided below:

2.3.2.1 hydro_hourly - Excel CSV

The hydro_hourly input file contains hour by hour (8760 hours total) generation potential for each hydro_hourly generator in the network configuration in units of MWh. It is important to remember that the generator IDs shown in the column headers match the IDs that are represented in the non-vre sheet in the modeled attributes spreadsheet.

date	hydro_hourly_BC1
1/1/2018 0:00	2.152
1/1/2018 1:00	2.152

2.3.2.2 hydro_daily - Excel CSV

The hydro_daily input file contains day by day (365 days total) generation potential for each hydro_daily generator in the network configuration in units of MWh. It is important to remember that the generator IDs shown in the column headers match the IDs that are represented in the non-vre sheet in the modeled attributes spreadsheet.

date	hydro_daily_AB1
day_1	232.8
day_2	232.8

2.3.2.3 hydro_monthly - Excel CSV

The hydro_monthly input file contains month by month (12 months total) generation potential for each hydro_monthly generator in the network configuration in units of MWh. It is important to remember that the generator IDs shown in the column headers match the IDs that are represented in the non-vre sheet in the modeled attributes spreadsheet.

date	hydro_monthly_BC1

month_1	40027
month_2	37632

2.3.2.4 importexport_hourly - Excel CSV

This file includes hourly Imported/Exported energy (in MW) for each import/export point (substation) in the case study. Imported power is demonstrated with a positive value, while exported power is demonstrated with a negative value.

2.3.2.5 VRE_Resource_Analysis

Files in these folders include hourly capacity factor (cf) data for each wind and solar unit.

2.3.2.6 power curves - Excel Worksheet

These sheets in this document include technical information of different VRE technologies. The information is used to calculate the output power of VREs units.

2.3.2.7 power curves_2 - Excel Worksheet

The sheets in this document include technical information of different VRE technologies. The information is used to calculate the output power of VREs units.

2.3.2.8 XX_Demand_Real_Forecasted - Excel Worksheet

Generally, this file includes necessary data in order to calculate nodal real/forecasted load based on population fraction. Note that the "XX" in the beginning of the file name stands for the province, such as "BC" for British Columbia.

Zonal_Demand_Forecasted

This sheet contains the forecasted load schedule for each region represented in the network configuration. The sheet contains hour by hour forecasted demand data which is calculated using values in the Province_Total_Forecasted and dr_potential sheets. The units for the cell values are in MWh. The name of each region must correspond to the name that is represented in the model inputs spreadsheet in the demand centres sheet.

date	AB.a	BC.a
2018-01-01 00:00:00	100	100
2018-01-01 01:00:00	100	100
2018-01-01 02:00:00	100	100

Province_Total_Forecasted

This sheet contains the aggregated forecasted load schedule for the entire network configuration. The sheet contains hour by hour forecasted demand data **which is input by the user before simulation**. The units for the cell values are in MWh.

date	demand_fc
2018-01-01 00:00:00	200
2018-01-01 01:00:00	200
2018-01-01 02:00:00	200

Zonal_Demand_Real

This sheet contains the real load schedule for each region represented in the network configuration. The sheet contains real hour by hour demand data which is calculated using values in the Province_Total_Real and dr_potential sheets. The units for the cell values are in MWh. The name of each region must correspond to the name that is represented in the model inputs spreadsheet in the demand centres sheet.

date	AB.a	BC.a
2018-01-01 00:00:00	97	97
2018-01-01 01:00:00	97	97
2018-01-01 02:00:00	97	97

Province_Total_Real

This sheet contains the real aggregated load schedule for the entire network configuration. The sheet contains real hour by hour demand data **which is input by the user before simulation**. The units for the cell values are in MWh.

date	demand
2018-01-01 00:00:00	194
2018-01-01 01:00:00	194
2018-01-01 02:00:00	194

dr_potential

This sheet contains data pertinent for characterizing individual loads and their dr potential. The bottom table is used to aggregate individual city-level or municipal-level loads into regional loads. The "frac pop" parameter represents the fraction of the population of the region that the individual load feeds. The latitude and longitude values are not used directly by the modelling framework, but they are sometimes useful for organizing and/or visualizing the loads. The top table is used to breakdown the demand fraction of each region in the network configuration. The demand fraction values will be used in the zonal demand sheets to calculate the load schedule for each region. **The user must enter this information and calculate values for the two tables accordingly as they are scenario-specific.** The information entered in here should correspond to the same information that is entered into the demand centres sheet in the modeled attributes spreadsheet.

	AB	BC

demand fraction	0.5	0.5

city	frac pop	lat	long	bus	dr_potential	region
Load_1	1	50	50	AB	1	AB
Load_2	1	50	50	BC	1	BC

2.3.2.9 model inputs - XX - Excel CSV

This file includes data of power system components. Note that the "XX" in the beginning of the file name stands for the province, such as "BC" for British Columbia. The file has a number of sheets which include:

site independent:
Includes general characteristics of network components.

modeled attributes:
The modeled attributes sheet contains information on operational parameters and costs for different generation technologies. Modeled attributes are a great place to start when constraint relaxation is required: ramp rates and min up/down times. This sheet is also important for setting up characteristics for dummy generators which are useful for modelling supply/demand imbalances. To designate a generation type as a dummy generator, the user should change the start-up cost to an extremely high value such as $10,000/MWcap.

Column	Data Type	Example	Description
Non-vre technology assumptions	String	coal	identifies the type of generator
start-up cost	float	54	cost to start up the generator for commitment ($) [$ / MWcap]
shut-down cost	int	0	cost to shut down the generator ($) [$ / MWcap]
min up time	int	12	minimum time for which a unit must be on (status =1) after it is committed [hours]
min down time	int	1	minimum time for which a unit must be off (status = 0) after it is de-commited [hours]
ramp rate max	float	.25	positive change in real power over 1 hour (MW/hr) for a spinning unit (with status = 1)
ramp rate min	float	-.25	negative change in real power over 1 hour (MW/hr) for a spinning unit (with status =1)
start up ramp limit	float	no values present	maximum positive change in real power over the first hour after startup (MW/hr)

shut down ramp limit	float	no values present	maximum negative change in real power over the last hour before shutdown (MW/hr)

vre plants:

Includes data for wind and solar generation units. This sheet is used to tabulate details for all VRE plants represented in the network configuration. Usage in main_model.py function read_vre_plants() also contains most information for vre generators that ends up in all_plants df and eventually generators.csv

Column	Data Type	Example	Description
Plant ID	String	Wind	Identifies the type of VRE, first letter is always capitalized
name	String	Wind_1	Unique identifier of VRE plant
kind	String	Wind_onshore	more specific description of VRE type
latitude	Float	44.2044	geographic latitude of plant [degrees]
longitude	Float	44.2044	geographic longitude of plant [degrees]
[MW]	Float	102.4	maximum real power of generator [MW]
bus	string	Bruce A TS	bus station the generator is connected to
technology type	String	V_112-3.0	unit model representing the generator
latitude_MERRA	Int	268	Corrected MERRA latitude =(latitude+90)*2; used to locate schedule folder for vre plant
longitude_MERRA	Int	148	Corrected MERRA longitude =(longitude+180)*(3/2); used to locate schedule folder for vre plant

non-vre plants:

This sheet is similar to the one for VRE plants. Includes data of Non-VRE generation units (e.g. nuclear, gas, hydro, etc). Non-VRE plants are simplified as each plant is dispatchable and does not require its own schedule to be made. However, each hydro generator does have a schedule of availability that is prespecified, which can be found in the Hydro_Data inputs folder in user inputs.

Column	Data Type	Example	Description
Plant ID	String	nuclear_ON.a	unique identifier general form is <kind>_##
name	String	nuclear_ON.a	unique identifier same as Plant ID
kind	String	nuclear	type of generator
cost curve equation	string	12.P	costcurveequation: text describing a polynomial cost curve ($/MWh) read by bidding.parse_polynomial and returns a list; 12.P -> [0,12.0] constant_term = 0 polynomial = [0, 12.0]
pmin		75	minimum real power of generator [MW]
[MW]	Float	100	maximum real power of the generator [MW]
bus	string	AB.a	bus station the generator is connected to

initial power	String	80	Initial power of the storage unit [MW]. (used to assess whether the unit will have a cold start)
initial status	Int	1	specifies whether a generator is on or off at start of simulation period (0 = off; 1 = on)
hours in status	Int	268	stores the number of hours that a generator has been in the current status (used for min up/down time constraints)

storage:
Includes input data for storage units (e.g. hydro pump storage, electric storage, etc).

Column	Data Type	Example	Description
plant ID		LB_PE.a	unique identifier general form is <kind>_##
name		LB_PE.a	unique identifier same as Plant ID
kind		batteries	Type of storage unit (currently supports pumped storage hydro and lithium ion battery)
[MW]	String	30	Maximum charge and discharge rate [MW]
storagecapacitymax	float	120	Maximum capacity of storage unit [MWh]
pumprampmax	int	1	Maximum ramp rate of the storage unit (set to a value of 1)
bus	String	AB.a	The connected bus of the storage unit
initial power	float		Initial power [MW]
initial status	int		Status of the storage unit at the beginning of simulation (0 = off; 1 = on)
hours in status	int		stores the number of hours that a storage unit has been in the current status (used for min up/down time constraints)
storage content	float	60	Initial storage content [MWh]

EV_aggregator:
Includes data for electric vehicles.

demand response:
Includes parameters for simulating the demand response program. This sheet contains values that are used in the UC problem step to determine the system-wide demand response constraints. To inactivate demand response in your simulation, leave both the over-production and under-production values at 100%.

Column	Data Type	Example	Description
index	String	maximum hourly dr limit (% of schedule)	holds constraint names
1	varies	130%, 1, over-production	generally holds max values for constraints
2	varies	130%, 1, under-production	generally holds min values for constraints

demand centres:
Includes load data (e.g. load location on power system, demand response potential of load, etc.). This model input sheet is tightly coupled with the Demand_Real_Forecasted.xlsx workbook. This sheet stores load information and is read into silver as two initial dataframes in setup_call_weekly.py: “regions_df” which

contains the entire excel sheet with an additional index and “loads” which only contains city as “name”, bus, and dr_potential, these are then passed to call_mp. For more context of usage see loads_opf_csv() in main_model.py

Column	Data Type	Example	Description
city	String	Load_1	specific load within region/city read into loads df as “name”
frac pop	float	range(0,1)	proportion of demand of the associated region represented by the load (expressed as a percentage 0%->100%)
bus	string	AB.a	bus station that will supply electricity to the associated region to satisfy the load
dr_potential	%	0.00%	Proportion of load that can be shifted for demand response (expressed as a percentage 0%->100%)
region	string	AB.a	specifies area of demand, the region names must match the demand forecast sheets names exactly

existing transmission:
This sheet includes data of transmission system characteristics in the network configuration (e.g. line reactance, line maximum capacity, substations, etc). There are really two constraints on lines, maximum capacity and the reactance, which acts as an indirect constraint on the line flow.

Column	Data Type	Example	Description
name	String	Line_1	Name of the transmission line (unique)
from bus		AB.a	Starting bus connecting the transmission line
to bus		BC.a	Ending bus connecting the transmission line
Voltage		345	Voltage level of the transmission line [kV]
length		142	Length of the transmission line in [km]
pmax			Maximum power capacity of the line [MW] (leave blank for no limit - although the reactance will serve as another constraint on the line capacity)
reactance		0.01	Reactance value of the transmission line, which is used as an indirect constraint on the line flow

2.3.3 <config_variables.py>

This python file stores some of the parameters that are entered by the user in the configVariables.ini file, including the directory of Excel files and directory of results folder among others. In addition, some parameters such as hours of commitment and year of operation (which are defined according to available data in Excel file) are stored in this Python file.

2.4 Creation of UC and OPF problems

To model the network and other components, and also create OPF and UC problems, Minpower package3 is used in SILVER. Minpower is an open-source and free package written in a series of Python files [2]. It

is used to create usual power system problems such as economic dispatch(ED), OPF and UC. As Minpower is open-source, it gives the opportunity to users to change and develop aforementioned problems according to their preferences, for example, a user can add additional constraints for components, change objective function or even add new components to the models. In what follows, the application of each Minpower package module is explained.

2.4.1 <generators.py>

In this module, conventional generators, electric vehicles (EVs) and electric storages are modeled. To do this, initial values, constraints and operation cost functions related to each component are created using different functions in the module. As mentioned earlier, users can change and develop components for the model, for instance, add new constraints, etc.

2.4.2 <powersystems.py>

Power system components such as load, line and bus are modeled in this module. To do this, variables, constraints and the objective function related to each component are created using different functions.

2.4.3 <optimization.py>

This module is an optimization command library for Minpower and acts as an interface between Minpower and the optimization solver. In fact, this file can send the optimization problem off from Minpower to the solver.

2.4.4 <schedule.py>

In this module, there are functions that provide time and schedule for related modules.

2.4.5 <stochastic.py>

This module creates a scenario tree for uncertain parameters in order to make a stochastic programming model. Considering stochastic programming, uncertain behaviors of parameters are modeled with several scenarios in which each of these scenarios has its own occurrence probability. In the SILVER model, stochastic programming is applied to cover uncertain behaviors of wind and PV units and loads.

2.4.6 <standalone.py>

Generally, this module acts as memory in Minpower. In fact, data are saved by this module to be used in continuation of the program. standealone.py is a module to workaround a memory leak in coopr by saving the results of each day of a rolling unit commitment to disk (in HDF format) and reloading them to run the next day as a memory independent sub-process.

2.4.7 <silver_variable.py>

This module imports the parameters (folder_opf, path, folder_uc, path_model_inputs, DR_shedding, etc) that are defined in config.py by the user. To use these parameters in other modules, silver_variable.py is called.

2.4.8 <solve.py>

This module contains the top-level commands for creating problems and solving them. Using the create_problem function in the module, the user can create ED, OPF and UC problems. Created problems consist of variables, constraints and an objective function. The aforementioned problems can be solved using the solve_problem function.

2.4.9 <bidding.py>

A bid is modeled by a polynomial function or a set of piecewise points in this module. The bid is used to define a generator's operation cost in an operation period.

2.4.10 <commonscripts.py>

This module consists of applicable and practical functions that are used in the framework which makes it easier to work with. For instance, convert_str2num function receives a string as input and returns a number as output. Another example is the Correct_status function, this function gets the status of components as input and changes their values to 1 if the status is more than 0.99 and 0 if status is less than 0.01. This function is practical because sometimes solvers do not give an absolute number (0 and 1) for the status of components and it may result in some problems in the execution program.

2.4.11 <config.py>

This module provides default parameter values and configuration for other modules. To do this, user_config is deemed as global. It consists of several default and user-defined parameters. For example, user_config.breakpoints indicates number of breakpoints which is a key parameter in creation of the bidding piecewise function. Another example is user_config.solver, it illustrates the solver that the user considers to solve the problem.

2.4.12 <get_data.py>

This module acts as an interface between Minpower and input data (Excel files). Using the key function read_csv(an internal function in Python), data are read from Excel files and entered into the Python environment. Next, these data are parsed into related classes.

2.4.13 <results.py>

This module provides outputs to display results of power system optimization problems. The user can apply this module to plot the results of problems. The module takes advantage of Matplotlib and network functions for visualization.

2.5 Intermediary folder contents

Several intermediary files are created and deleted in the process of building and solving the linear program models for the price setting OPF, UC, and real-time OPF problems. Several of these are described below.

2.5.1 Bus-name.csv

This file contains a list of bus names represented in the network configuration.

2.5.2 generators.csv

This file is created from all_plants df, written to intermediary folders and then read by get_data.py.

Column	Data Type	Example	Description
name	String	nuclear_1	unique identifier same as Plant ID
kind	String	nuclear	type of generator
cost curve equation		12.P	polynomial cost curve ($/MWh)
pmax		3545	max real power
pmin	Float	0	minimum real power
storagecapacitymax	Float		Maximum capacity of storage unit [MWh]
pumprampmax	Float		Maximum ramp rate of the storage unit (set to a value of 1)
schedule filename	String		filename associated with the generation schedule
shedding_allowed	Int		dr parameter
start up cost	Int	193820	only used in UC -cost to commit ($) cost to start up the generator for commitment ($) [$ / MWcap]
shut down cost	Float		only used in UC cost to shut down the generator ($) [$ / MWcap]
min up time	Int	168	only used in UC • min time after commitment in on state (hours) minimum time for which a unit must be on (status =1) after it is committed [hours]
min down time	Int	24	only used in UC • min time after commitment in off status (hours) minimum time for which a unit must be off (status = 0) after it is de-commited [hours]
ramp rate max	Float	35.24	only used in UC • max. positive change in real power over 1hr (MW/hr) positive change in real power over 1 hour (MW/hr) for a spinning unit (with status = 1)
ramp rate min	Float	-34.23	only used in UC

			• max. negative change in real power over 1hr (MW/hr) negative change in real power over 1 hour (MW/hr) for a spinning unit (with status =1)
start up ramp limit			only used in UC -max. positive change in real power over the first hour after startup (MW/hr) maximum positive change in real power over the first hour after startup (MW/hr)
shut down ramp limit			only used in UC -max. negative change in real power over the last hour before shutdown (MW/hr) maximum negative change in real power over the last hour before shutdown (MW/hr)
mustrun	boolean		flag that forces commitment to be on

2.5.3 lines.csv

This file is created from the "existing transmission" sheet in model – inputs used for price setting, realtime and UC if UC_networked == True. This file is read by the minpower module as file_lines = lines.csv then stored as a df called lines_data and finally stored as lines from build_class_list().

used to Create the admittance matrix (B),
with elements = total admittance of line from bus i to j.
used in calculating the power balance for OPF problems.

Column	Data Type	Example	Description
name	index	Line_1	Line_1 – Line_XX
from bus	string	Evergreen SS	
to bus		Longwood TS	
pmax	Float	blank	if blank default value is 9999
reactance	Float	blank	if blank default value is .05 as defined in Line class initialization

2.5.4 loads.csv

This file is created from the "demand centres" sheet in model – inputs used for price setting, realtime and UC if UC_networked == True. This file is read by the minpower module as file_loads = loads.csv then stored as a df called loads_data and finally stored as loads from build_class_list().

used to Create the admittance matrix (B),
with elements = total admittance of line from bus i to j.
Used in calculating the power balance for OPF problems.

Column	Data Type	Example	Description
name	index	Load_1	Load_1 – Load_XX
bus	string	Evergreen SS	Bus that load is connected to
power		33.99	Power consumption of the load [MW]
dr_potential		0.001455	Used to add dr constraints

2.5.5 powerflow-dr.csv

This file contains demand response values by bus, it is updated every hour during price-setting OPF and real-time OPF. In price-setting OPF the values represent the DR potential. In real-time OPF the values represent the DR values that were chosen in the UC problem.
Seems broken and extra empty lines between each entry

Column	Data Type	Example	Description
bus	string	bus names	bus names repeated with different dr_2 values
dr_2	string	Evergreen SS	all 0s for price setting opf not read for UC stage calculated and written after each iteration

2.5.6 powerflow-generators.csv

This file contains the status and powerflow of each generator in the network configuration, it is updated every hour during price-setting OPF and real-time OPF.

Column	Data Type	Example	Description
generator	string	Wind_1	bus names repeated
u	string	1	U stands for status. 1 for all entries except import export which is TRUE
p	float	148.235	P for power. positive floats including 0

2.5.7 powerflow-lines.csv

This file contains the powerflow of each transmission line in the network configuration, it is updated every hour during price-setting OPF and real-time OPF.

Column	Data Type	Example	Description
from	string	Ashfield SS	bus names repeated
to	string	Longwood TS	1 for all entries except import export which is TRUE
power	Float	148.235	positive floats including 0
congestion		0	0 for all
shadow price			blank

2.5.8 vre_schedule_Solar_1.csv

Stores generation schedule for vre plants. Value of filename is created by string concat in append_vre(). In the UC problem the schedule represents the forecasted generation profile.

```
'vre_schedule_' + vre_plants.loc['name'].iloc[plant] + '.csv'
```

written to file for UC run in main_model.create_uc_temp_vre_csv()

- source of schedule is vre_timeseries_fc source of this input is the vre folder is Available_VRE_

Column	Data Type	Example	Description
date	index		Date and hour
cf	float	0.00345634	Capacity factor of unit (percentage)

2.5.9 storage_cost_opf.csv

This file contains the cost of storage for each storage unit in the real-time OPF problem.

Column	Data Type	Example	Description
	t index format	t00-txx	identifies which hour data is for range from 0 to hours_commitment – 1
0 -> # of storage units	float	3.3	

2.5.10 hydro_daily.csv

This file contains the daily generation values for each hydro_daily generator.

	hydro_daily_AB1
day_1	232.8
day_2	232.8
…	…

2.5.11 hydro_hourly.csv

This file contains the hourly generation values for each hydro_hourly generator in units of MWh.

	hydro_hourly_BC1
t00	2.152
t01	2.152
t02	2.152
t03	2.152
…	…

2.5.12 hydro_monthly.csv

This file contains the monthly available generation values for each hydro_monthly generator in units of MWh.

	hydro_monthly_BC1
month_1	40027.2

2.5.13 commitment-dr.csv

This file contains dr commitment values for the UC problem.

time	dr
2018-01-01 0:00	0
2018-01-01 1:00	10.5765
2018-01-01 2:00	6.7033
2018-01-01 3:00	-11.2344

2.5.14 commitment-linesflow.csv

This file contains the powerflow values for each transmission line for the UC problem for a case where UC_networked ==True, in units of MW.

from	to	t00	t01	t02	t03	…
						…
AB.a	BC.a	11.8	-41.42	-46.29	-54	…

2.5.15 commitment-power.csv

This file contains the generation values for each generator unit in the network configuration for the UC problem in units of MWh.

time	g0	g1	g2	g3	g4	g5
2018-01-01 0:00	79	4	0	32.8	2.152	66.6978
2018-01-01 1:00	78	3.5	0	0	2.152	100

2.5.16 commitment-status.csv

This file contains the generation status values for each generator unit in the network configuration for the UC problem.

time	g0	g2	g3	g8	g1
2018-01-01 12:00:00 AM	1	1	1	0	1
2018-01-01 1:00:00 AM	1	1	1	1	1

2.5.17 initial.csv

This file contains characteristics for generators that are initialized with a status of 1. The power value is in units of MW.

	name	kind	power	status	hours in status	storage content
0	nuclear_ON.a	nuclear	75	1	48	0

2.5.18 LMP.csv

This file contains the locational marginal price of electricity for regions connected to each bus in units of $/MWh.

bus_name	LMP
AB.a	288
BC.a	288

2.5.19 load_schedule_real.csv

This file contains the real load schedule for each load represented in the network configuration, in units of MWh.

	Load_1	Load_2
t00	100	100
t01	100	100
t02	100	100
…	…	…

2.5.20 all_costs.csv

This file contains all costs for each generator in the network configuration in units of $. The first row represents total generation cost, the second represents total fuel cost, and the third represents the incremental cost of the generator.

time	g0	g2	g3	g8	g1	g4	g5	g6	g7	g9
2022-10-14	900	0	0	0	0	13.2	168.597	3.10E-06	8.10E-07	0
2022-10-14	900	0	0	0	0	13.2	168.597	3.10E-06	8.10E-07	0
2022-10-14	12	350	3.3	20	100	3.3	3.3	1.00E-06	1.00E-06	20

2.5.21 totalcost_generation.csv

This file contains the total cost of generation for each generator in units of $.

time	g0	g2	g3	g8	g1	g4	g5	g6	g7	g9
2022-10-14	900	0	0	0	0	13.2	168.597	3.10E-06	8.10E-07	0

2.6 Problem solving

In this section, general descriptions are provided for the modules that are used to solve the OPF and UC problems.

2.6.1 <scenario_analysis_module.py>

In this module, four metrics are calculated in the first step of the SILVER model for different system design scenarios. The metrics are calculated by the Scenario_metrics function. These metrics measure the acceptability of system design scenarios. These four metrics have been defined in [1].

2.6.2 <vre_module.py>

Generally, this module provides renewable output power prediction. In this module, predicted wind speed and solar irradiance are entered as input values. Then, using a conversion function of the wind turbine and photovoltaic panel, their output power is calculated.

2.6.3 <main_model.py>

As mentioned earlier, four metrics in the first step of SILVER are calculated in the scenario_analysis_module module. The second, third and fourth steps are implemented in the main_model module. In the second step, an OPF problem is solved in order to determine marginal operation cost. To do this, all of the necessary data are collected in the vre_model_system_cost_opf folder. Then, using solve_problem (folder_price_opf) function, OPF problem is solved to determine operation marginal price. In the OPF problem, only conventional generation resources that are price maker players in the network are considered. The marginal price is applied in the third step (UC problem) as a reference price for operation of price taker components such as EVs and storage. Following the determination of the marginal price in the second step, the UC problem is solved in the third step to find the optimal operation of the network with the aim of cost minimization for the forecasted value of the load and VRE output. All of the necessary data for the UC problem are saved in the vre_model_outputs_uc folder. Then, the solve_problem (folder_uc) function is implemented to solve the UC problem. Because of the intermittent and unpredictable behavior of VREs and load uncertainty, it is possible that forecasted power output of VRE and predicted load are not equal to their real values. So, in the fourth step of SILVER, to compensate for prediction violation, an OPF problem is solved considering real output power of VRE and real load. Necessary data for this OPF problem are collected in the vre_model_outputs_opf folder. Then, the solve_problem (folder_opf) function is implemented to solve the OPF problem. All the mentioned steps are implemented in the main_model module, respectively.

2.6.4 <SILVER_VER_18.py>

To start the run of SILVER in Python, the SILVER_VER_18.py module must be run, all of the necessary modules to create and solve SILVER are called in this module one by one.

2.6.5 <run_uc_standalone.py>

This module has been provided to run the UC or OPF problems in the SILVER model separately. To run a price-setting OPF problem, the directory is set to folder_price_opf which indicates the directory of the vre_model_system_cost_opf folder. To run a UC problem, the directory is set to folder_uc which is the directory of the vre_model_outputs_uc folder. To run a RT_OPF problem, the directory is set to folder_opf which represents the directory of the vre_model_outputs_opf folder.

2.7 Modeling descriptions

Methods used to model various components and activities in the SILVER framework are described below. Descriptions are provided for demand response, energy storage, hydroelectricity, and calculating VRE output

2.7.1 Demand response

In the current version of SILVER, the load shifting program is modeled as a DR program. In this program, according to the price of electricity, loads can shift their electricity demand from high-price periods to low-price periods to reduce network operation cost.

To make DR active for a run, the user can set the value of the DR_shedding parameter (in config_variables.py) to True. The user can also set parameters related to DR in the Demand Response sheet in the Model Inputs excel file. The parameters are: maximum hourly dr limit (% of schedule); absolute hourly dr limit (MW); maximum daily dr (MW); ramping limitations (MW); and minimum up/down times (hours).

2.7.2 Storage

In the SILVER framework, storage is defined as a component that can charge and discharge electricity. It can be modeled after different technologies such as pumped hydro storage (PHS), battery energy storage, etc. To include a storage unit in a case study, the user must populate the parameters of the storage unit in the Storage sheet in the Model Inputs excel file.

2.7.3 Hydroelectricity

Hydroelectric generators are categorized into three sorts including hydro_hourly (run of river), hydro_daily and hydro_monthly. The difference between these three sorts of generators is in the duration that their reservoirs can store water (energy). For an hourly hydro generator, water behind the dam can be used on an hourly basis. Similarly, for daily and monthly hydro generators, water behind the dam can be used on a daily and monthly basis, respectively. To model hydro generation, one of the most important factors is the availability of energy in the reservoir during operation time. In the SILVER framework, availability of energy (in MW) of hydro units is populated in excel files (hydro_hourly, hydro_daily, hydro_monthly) in the Hydro_Data folder.

Additionally, the Hydro model in the SILVER framework includes all of the important operation constraints such as min/max output power, min up/down time, max ramp up/down rate, etc.

2.7.4 Calculating variable renewable energy output

The output power of VREs is calculated using the GRETA modeling tool [5]. GRETA calculates output power in a more accurate way based on data of wind speed, solar irradiance, temperature, technology, etc. The output power calculation is performed based on functions available in vre_module.py.

2.8 Model results

The SILVER framework saves simulation results in a series of Microsoft Excel files located in a folder within the SILVER_Data folder. There are files in the model results folder that contain hourly data on available VRE generation, energy storage, first OPF results, line flow, real-time OPF costs, real-time OPF results, and UC results.

2.8.1 Available VRE generation

Files that contain hourly available VRE generation values have names that begin with "Available_VRE_generation." These files contain hourly generation values for all VRE generators that are simulated in the framework. The units of the generation values are in MWh.

2.8.2 Energy storage - available energy

Files that contain hourly energy storage values have names that begin with "Energy_Storage." These files contain hourly storage values for all energy storage units that are simulated in the framework. The storage values represent the total energy contained and are in units of MWh.

2.8.3 First OPF results

Files that contain values for the first OPF problem have names that begin with "First_OPF_Results." These files contain hourly generation values for each generator simulated in the framework in units of MWh. Additionally, these files contain hourly load values for each demand node in the model in units of MWh. Finally, these files hold hourly power flow values for all transmission lines simulated in the model in units of MWh.

2.8.4 Line flow - UC

Files that contain power flow values for transmission lines in the simulation framework have names that begin with "Line_Flow." These files hold hourly power flow values for all transmission lines simulated in the model in units of MWh.

2.8.5 Real-time OPF costs

Files that contain generation cost values for the real-time OPF problem have names that begin with "OPF_Costs." These files hold hourly generation cost values for each generator simulated in the framework in units of dollars.

2.8.6 Real-time OPF results

Files that contain values for the real-time OPF problem have names that begin with "OPF_Results." These files contain hourly generation values for each generator simulated in the framework in units of MWh. Additionally, these files contain hourly load values for each demand node in the model in units of MWh. Finally, these files hold hourly power flow values for all transmission lines simulated in the model in units of MWh.

2.8.7 UC results

Files that contain generation values for the day-ahead UC problem have names that begin with "UC_Results." These files hold hourly UC values for each generator simulated in the framework in units of MWh.

2.9 Test cases for SILVER simulation

A series of test cases have been performed to give users a chance to validate their SILVER framework installation. The master SILVER framework has been preloaded with test case results for the Silver Simple (SS) scenario.

3. References

[1] M. McPherson and B. Karney, "A scenario based approach to designing electricity grids with high variable renewable energy penetrations in Ontario, Canada: Development and application of the SILVER model," Energy, vol. 138, pp. 185-196, 2017/11/01/ 2017.

[2] A. Greenhall, R. Christie and J. Watson, "Minpower: A power systems optimization toolkit," 2012 IEEE Power and Energy Society General Meeting, 2012, pp. 1-6, doi: 10.1109/PESGM.2012.6344667.

[3] McPherson, Madeleine, and Samiha Tahseen. "Deploying storage assets to facilitate variable renewable energy integration: The impacts of grid flexibility, renewable penetration, and market structure." *Energy* 145 (2018): 856-870.

[4] McPherson, Madeleine, et al. "Planning for variable renewable energy and electric vehicle integration under varying degrees of decentralization: A case study in Lusaka, Zambia." *Energy* 151 (2018): 332-346.

[5] McPherson, Madeleine, et al. "An open-access web-based tool to access global, hourly wind and solar PV generation time-series derived from the MERRA reanalysis dataset." *Energies* 10.7 (2017): 1007.