NATIONAL LABORATORY OF THE ROCKIES

Contents

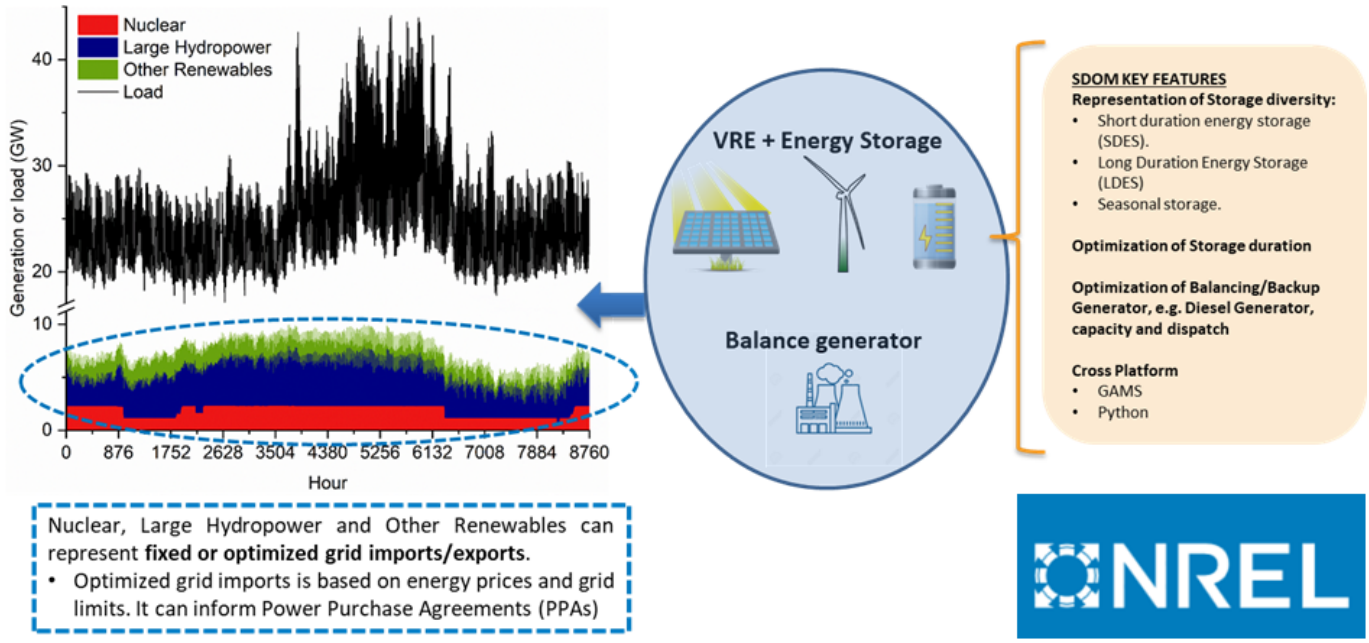
1. What is SDOM?

SDOM (Storage Deployment Optimization Model) is an open-source, high-resolution grid capacity-expansion framework developed by NLR. It's purpose-built to optimize the storage portfolio considering diverse storage technologies, leveraging hourly temporal resolution and granular spatial representation of Variable Renewable Energy (VRE) sources such as solar and wind.

SDOM is particularly well-suited for figure out the required capacity to meet a carbon-free generation mix target by:

- 📅 Evaluating long-duration and seasonal storage technologies
- 🌦️ Analyzing complementarity and synergies among diverse VRE resources and load profile
- 🗺️ Assessing curtailment and operational strategies under various grid scenarios

An illustrative figure below shows the flow from inputs to optimization results, enabling exploration of storage needs under varying renewable integration levels.



2. Getting started with SDOM

2.1 System Setup and Prerequisites

- a. You'll need to install python
  - After the installation make sure the python enviroment variable is set.
- b. Also, You'll need an IDE (Integrated Development Environment), we recommend to install MS VS code
- c. We also recommend to install extensions such as:
  - edit CSV: To edit and interact with input csv files for SDOM directly in vs code.
  - vscode-pdf: to read and see pdf files directly in vscode.

2.2 Install SDOM

It is recommended to load the packages in a virtual enviroment.

We recommend to use uv, a Python manager for virtual environments and packages.

- a. Create a new virtual environment named `.venv`:

```
uv venv .venv
```

  This command creates a Python virtual environment in the `.venv` directory.

- b. Activate your virtual environment and install the SDOM package:

```
uv pip install sdom
```

- c. Install the python module according to your solver. We'll use here HiGHS open-source solver

```
uv pip install highspy
```

- d. Install the Logging package to be able to see sdom info, warning and error messages and log those:

```
uv pip install logging
```

- e. Verify your environment by listing installed packages:

```
uv pip list
```

You should see output similar to:

```
Package         Version
--------------- -----------
highspy         1.11.0
logging         0.4.9.6
numpy           2.3.3
pandas          2.3.3
ply             3.11
pyomo           6.9.4
pytz            2025.2
sdom            0.0.7
six             1.17.0
tzdata          2025.2
```

3. Next Section - SDOM Inputs

Now, continue to the next section, SDOM Inputs

Contents

1. Input Files Folder

All the csv files you'll use in an SDOM optimization runs should be in one single folder. In this case, sample files are located at the route: "sample_data\br_test_daily_b".

This folder will be specified as a string containing the path in sdom when you do:

```
data_dir = "sample_data\\br_test_daily_b\\"
data = load_data( data_dir )
```

In the next section each file will be listed and the data it is supossed to be in each field will be described.

> ⚠ **Attention:**
>
> - Make sure all required CSV files are present in the specified folder before starting the simulation.

> - Please keep the root names of each file. For instance, in the sample files you can change "2025" for whatever you prefer, but keeping the root name. For example, for "CapSolar_2025.csv" file you need to keep the root name as "CapSolar_".
> - Please do not change the column names of each csv files.

2. CSV input files

In this section the SDOM input csv files will be grouped by technology, so all the data required for each technology is in a single subsection. The description of the required data for each field will be described.

2.1. formulations.csv

This file the user selects the modeling approach (formulation) for each major system component in the SDOM optimization model. Each row assigns a formulation to a component, determining how SDOM will represent its behavior and constraints during optimization.

Important Notes:

- Only one formulation should be assigned per component.
- The chosen formulation directly affects how SDOM optimizes and simulates each component.
- Refer to the tables below for valid formulations for each component.

CSV file columns:

Field/Column	Description	Expected type
Component	String with the component name you are going to set the formulation (Model).	string
Formulation	Name of the formulation (model) you want to use. See more below.	String
Description (Optional)	Just a description/guidelines that developers let for users.	String

The following table, shows the valid formulations for each component:

Component	Available formulations
Hydro	"MonthlyBudgetFormulation" (Budget of 730h), "DailyBudgetFormulation" (Budget of 24h), "RunOfRiverFormulation" (Generation according with input time series and it is not optimized).
Thermal	"NoRampsDispatchFormulation".
Imports	"CapacityPriceNetLoadFormulation", "NotModel" (When you want to ignore Imports).
Exports	"CapacityPriceNetLoadFormulation", "NotModel" (When you want to ignore Exports).

2.2. Variable Renewable Energies (VRE)

> ⚠ **Attention:**
>
> - Make sure that each "sc_gid" defined in "CapSolar.csv" and "CapWind.csv" has its correspondend capacity factor hourly profile in the "CFSolar.csv" or "CFWind.csv" files .

2.2.1 CapSolar.csv/CapWind.csv

This file lists all candidate sites for solar PV and wind energy deployment. For each site, it specifies the maximum allowed installed capacity, geographic coordinates, capital expenditure (CAPEX), fixed operation and maintenance (FOM) costs, and transmission interconnection costs. These parameters are used by SDOM to evaluate investment options and optimize resource allocation across the available sites.

CSV file columns:

Field/Column	Description	Expected type
sc_gid	Unique identifier for each PV/Wind site or resource that will be represented by a single profile.	string
capacity	Upper bound for the allowed installed capacity at the site (MW).	float
latitude	Latitude coordinate of the site (optional for future fetching of VRE files).	float
longitude	Longitude coordinate of the site (optional for future fetching of VRE profiles).	float
trans_cap_cost	Transmission Capital expediture costs associated with transmission in USD/kW	float
CAPEX_M	Capital expenditure in USD/kW.	float
FOM_M	Fixed operation and maintenance cost in USD/kW.	float

2.2.2 CFSolar.csv/CFWind.csv

This file defines the hourly solar/wind capacity factors for each one of the potential sites defined in 2.1.1. This information can be obtained using, for instance NLR SAM simulations or reV.

CSV file columns:

Field/Column	Description	Expected type
Hour	Number of the hour of the year, from 1 to 8760 (You can choose the number of hours you prefer)*.	Int
Col for each id	The estimated capacity factor at each hour of the year for each site in MWh/installed MW.	float

> *⚠ **Attention:**

> - Ensure that your input files contain a number of hours equal to or greater than the `n_hours` parameter specified when calling the `initialize_model()` function:

```
model = initialize_model(
    data,
    n_hours=n_steps,
    with_resilience_constraints=with_resilience_constraints
)
```

2.3. Data_BalancingUnits.csv

This file contains essential data for thermal generation plants or aggregated units that participate in system balancing. Each row represents a plant or group of plants, specifying their technical and economic parameters required for SDOM optimization.

Key considerations:

- Each `Plant_id` must be unique and consistently referenced across other input files.
- Capacity values (`MinCapacity`, `MaxCapacity`) set the bounds for possible investments (installed capacity).
- To represent already existent generation fleet it is recomended add a new row where `MinCapacity` = `MaxCapacity` and CAPEX = 0

CSV file columns:

Field/Column	Description	Expected type
Plant_id	Unique identifier for each thermal generation plant or aggregation of plants.	string
MinCapacity	Minimum allowed installed capacity at the plant (MW).	float
MaxCapacity	Maximum allowed installed capacity at the plant (MW).	float
Lifetime	Expected operational lifetime of the plant (years).	int
Capex	Capital expenditure in USD/kW.	float
HeatRate	Heat rate of the plant (fuel energy input per unit electricity output, typically in MMBtu/MWh).	float
FuelCost	Fuel cost in USD/MMBtu.	float
VOM	Variable operation and maintenance cost in USD/MWh.	float
FOM	Fixed operation and maintenance cost in USD/kW.	float

2.4. Hydro power

2.4.1 lahy_hourly.csv

This file provides the hourly generation time-series for hydropower plants. The way SDOM utilizes this data depends on the selected hydro formulation (see section 2.1 formulations.csv):

- **RunOfRiverFormulation:**
  Hydropower generation is directly set to the values specified in the time-series. No optimization is performed; the model simply follows the provided hourly profile.
- **Budget Formulations (MonthlyBudgetFormulation or DailyBudgetFormulation):**
  SDOM aggregates the time-series data into energy budgets over consecutive periods—24 hours for daily budgets and 730 hours for monthly budgets. These budgets define the total energy available for dispatch within each period, allowing SDOM to optimize the allocation of hydropower generation while respecting the specified limits. These budgets usually are outputs from long- and medium-term hydropower planning tools often based on Stochastic dual dynamic programming (SDDP). An example of open-source tool for this is SimSEE

In summary, this file either serves as a fixed generation profile or as the basis for energy budgets, depending on the chosen hydro formulation.

CSV file columns:

Field/Column	Description	Expected type
*Hour	Number of the hour of the year, from 1 to 8760 (You can teh number of hours you prefer).	Int
LargeHydro	The estimated hydropower generation at each hour of the year in MWh or average values to make the budget.	float

2.4.2 lahy_min_hourly.csv/lahy_max_hourly.csv (optional)

These files determine the lower and upper bounds for hydropower dispatch when Budget Formulations are used.

CSV file columns:

Field/Column	Description	Expected type
*Hour	Number of the hour of the year, from 1 to 8760 (You can teh number of hours you prefer).	Int
LargeHydro	The estimated hydropower generation bounds at each hour of the year.	float

2.5. Nucl_hourly.csv/otre_hourly.csv

This file provides the hourly generation time-series for nuclear power plants and other renewable plants respectively.

CSV file columns:

Field/Column	Description	Expected type
*Hour	Number of the hour of the year, from 1 to 8760 (You can teh number of hours you prefer).	Int
Nuclear/OtherRenewables	The estimated generation at each hour of the year in MWh from Nuclear of Other Renewables (Such as Biomass, for instance).	float

2.6. Load_hourly.csv

This file provides the system hourly electricity demand time-series.

CSV file columns:

Field/Column	Description	Expected type
*Hour	Number of the hour of the year, from 1 to 8760 (You can teh number of hours you prefer).	Int
Load	The estimated system electricity demand at each hour of the year in MWh.	float

2.7 StorageData.csv

This CSV input file provides key technical and economic parameters for diverse energy storage technologies (Some examples could be: Li-Ion (Lithium-Ion), CAES (Compressed Air Energy Storage), PHS (Pumped Hydro Storage), and H2 (Hydrogen Storage)). Each column represents a technology, and each row specifies a parameter:

Field	Description	Expected type
P_Capex	Power-related capital expenditure (USD/kW)	float
E_Capex	Energy-related capital expenditure (USD/kWh)	float
Eff	Round-trip efficiency (fraction)	float
Min_Duration	Minimum storage duration (hours)	float/int
Max_Duration	Maximum storage duration (hours)	float/int
Max_P	Maximum power capacity (kW)	float/int
MaxCycles	Maximum number of charge/discharge cycles	int
Coupled	Indicates if input and output power are coupled (1 = coupled - enforces input Power = output Power)	int (0 or 1)
FOM	Fixed operation and maintenance cost (USD/kW/year)	float
VOM	Variable operation and maintenance cost (USD/kWh)	float

Field	Description	Expected type
Lifetime	Expected system lifetime (years)	int
CostRatio	Ratio of cost allocation between input and output power. If Input Power Capex = Output Power Capex, then CostRatio = 0.5	float

Key considerations:

- If you dont have energy CAPEX for the technology, and you only have power CAPEX for a particular duration, enforce Min_Duration==Max_Duration.
- If the power capex is equally divided for the input power capacity and output power capacity, set CostRatio = 0.5.
- if the storage technology does not have an specification for MaxCycles, use a large value.

Cost Data Sources Some sources to get cost data for storage technologies are:

- NLR Annual Technology Baseline (ATB)
- PNNL "Energy Storage Cost and Performance Database v2024"

2.8. scalars.csv

This CSV file contains key parameters and their corresponding values for an energy system model. Each row specifies a parameter name and its value, which are used to configure various aspects of the model, such as technology lifetimes, financial rates, and generation mix targets.

Parameter	Description	Expected Type
LifeTimeVRE	Operational lifetime in years of Variable Renewable Energy sources (To calculate CRF)	int
GenMix_Target	Target value for generation mix (e.g., share of renewables). Between 0 and 1.	float
AlphaNuclear	Activation/Deactivation (1/0) for nuclear energy	int (0 or 1)
AlphaLargHy	Activation/Deactivation (1/0) for large hydro energy	int (0 or 1)
AlphaOtheRe	Activation/Deactivation (1/0) for other renewable energy sources	int (0 or 1)
r	Discount rate or interest rate used in financial calculations (Example: r=0.06)	float
EUE_max	Maximum allowed Expected Unserved Energy (EUE) - used when resiliency constraints = true	float

Note: Adjust parameter values as needed to reflect the specific scenario or assumptions for your energy system analysis.

2.9. Imports/Exports (Optional files)

These files are only required when Imports and Exports formulations are set different to "NotModel" in "formulations.csv" (see section 2.1 formulations.csv).

2.9.1. Import_Cap.csv/Export_Cap.csv

These files contain

CSV file columns:

Field/Column	Description	Expected type
*Hour	Number of the hour of the year, from 1 to 8760 (You can teh number of hours you prefer).	Int
Imports/Exports	The hourly maximum capacity to import/export in MW.	float

2.9.2. import_prices.csv/export_prices.csv

CSV file columns:

Field/Column	Description	Expected type
*Hour	Number of the hour of the year, from 1 to 8760 (You can teh number of hours you prefer).	Int
Imports_price/Exports_price	The estimated hourly electricity price for importing/exporting energy in USD/MWh.	float

3. Next Section - Getting Started: Running SDOM with a Python Script

Now, continue to the next section, [Getting Started: Running SDOM with a Python Script](training_material\training_sesion_1\1_3_Running_SDOM and outputs.md)

Contents

1. Getting Started: Running SDOM with a Python Script

This section explains the steps involved in running the SDOM model using Python, with code snippets for each step:

1. **Import Required Modules**
   Import SDOM functions, utilities for logging and the solver interface module:

```
import sdom
from sdom import run_solver, initialize_model, configure_logging,
get_default_solver_config_dict
from sdom import load_data, export_results

import logging
import highspy
```

2. **Configure Logging**
   Set up logging to display informational messages:

```
configure_logging(level=logging.INFO)
```

3. **Set Simulation Parameters**
   Define simulation steps, resilience constraints, and case name:

```
n_steps = 730*1  # Number of steps in the simulation (in hours)
with_resilience_constraints = False
case = "br_test_daily_b"
```

> ⚠ **Attention:**
>
> - Make sure all required CSV files are present in the specified folder before starting the simulation. See the required input files here
> - Please keep the root names of each file. For instance, in the sample files you can change "2025" for whatever you prefer, but keeping the root name. For example, for "CapSolar_2025.csv" file you need to keep the root name as "CapSolar".
> - Please do not change the column names of each csv files.

4. **Load Input Data**
   Load input data from the specified directory and define output folder:

```
data_dir = "sample_data\\br_test_daily_b\\"
data = load_data(data_dir)
output_dir = f'./sample_results_{case}/'
```

5. **Initialize the Model**
   Initialize the SDOM model with data and parameters:

```
model = initialize_model(
    data,
    n_hours=n_steps,
    with_resilience_constraints=with_resilience_constraints
)
```

6. **Configure Solver**
   Get default solver configuration for HiGHS open-source solver:

```
solver_dict = get_default_solver_config_dict(
    solver_name="highs",
    executable_path=""
)
```

7. **Run the Solver**
   Execute the solver to find an optimal solution:

```
best_result = run_solver(model, solver_dict)
```

8. **Export Results**
   Export results if a solution is found, otherwise print a message:

```
if best_result:
    export_results(model, case, output_dir=output_dir)
```

```
else:
    print(f"Solver did not find an optimal solution for given data and with
resilience constraints = {with_resilience_constraints}, skipping result
export.")
```

2. Outputs

In the path specified by "output_dir", sdom will writhe the following output csv files:

File name	Description
OutputGeneration_CASENAME.csv	Hourly generation results aggregated by technology, curtailment, imports/exports and Load.
OutputStorage_CASENAME.csv	Hourly storage operation results (charging/discharging and SOC).
OutputSummary_CASENAME.csv	Summary of key simulation results and statistics.
OutputThermalGeneration_CASENAME.csv	Hourly results for thermal generation plants.

3. Next Section - Debugging and Exploring SDOM Pyomo Model

Now, continue to the next section, Debugging and Exploring SDOM Pyomo Model

Contents

1. Debugging and Exploring SDOM Pyomo Model.

1.1. Pyomo Model Instance

This section introduces how the user can explore the pyomo model created by sdom and in that way debug the model in a detailed way by accessing all the variables, sets, parameters, expressions, constraints, etc.

SDOM uses pyomo to write the optimization model.

When you run

```
```python
model = initialize_model(
    data,
    n_hours=n_steps,
    with_resilience_constraints=with_resilience_constraints
)
```
```

the function `initialize_model()` returns the pyomo instance of the optimization model, which in this case is stores in the variable `model`.

1.2. Pyomo Blocks

SDOM leverages on pyomo blocks to separate in different blocks the variables, parameters, expressions, constraints, etc of diferent model components. In this way, in that pyomo instance, SDOM creates the following blocks:

- Core optimization Blocks (Blocks that include variables, sets and parameters)
  - thermal
  - pv
  - wind
  - hydro
  - storage
- Blocks containing only parameters (Do not include any decision variables)
  - demand
  - nuclear
  - other_renewables
- Optional blocks (These are created depending on the configuration provided by the user)
  - imports
  - exports
  - resiliency

1.3. Discovering model names for parameters, expressions, constraints, etc

You can access to parameters, expressions, constraints, etc by doing: model_instance.block_name. For instance:

```
```python
model.thermal
```
```

when you open this in a debug sesion, you'll observe:

```
```python
<pyomo.core.base.block.ScalarBlock object at 0x0000021A4C424280>
special variables
function variables
class variables
CAPEX_M = <pyomo.core.base.param.IndexedParam object at 0x0000021A4C4A34D0>
FCR = <pyomo.core.base.param.IndexedParam object at 0x0000021A4C4A3C90>
FOM_M = <pyomo.core.base.param.IndexedParam object at 0x0000021A4C4A36D0>
VOM_M = <pyomo.core.base.param.IndexedParam object at 0x0000021A4B3BD790>
active = True
capacity_generation_constraint = <pyomo.core.base.constraint.IndexedConstraint
object at 0x0000021A4D3AA610>
capex_cost_expr = <pyomo.core.base.expression.ScalarExpression object at
0x0000021A4C4D3BB0>
data = <pyomo.core.base.param.IndexedParam object at 0x0000021A4C4A2C90>
doc = None
fixed_om_cost_expr = <pyomo.core.base.expression.ScalarExpression object at
0x0000021A4C4D3B10>
fuel_price = <pyomo.core.base.param.IndexedParam object at 0x0000021A4C4AC610>
generation = <pyomo.core.base.var.IndexedVar object at 0x0000021A4C5F3190>
```
```

```
heat_rate = <pyomo.core.base.param.IndexedParam object at 0x0000021A4C4A3290>
...
```

1.4. Exploring a parameter - model.thermal.heat_rate

Using `pprint()` function, you'll be able to explore model objects in a simple way. If you run:

```python
model.thermal.heat_rate.pprint()
```

You can obtain:

```python
heat_rate : Size=13, Index=thermal.plants_set, Domain=Any, Default=None,
Mutable=False
Key  : Value
106c :   1.0
147c :   1.0
162c :   1.0
163c :   1.0
167c :   1.0
170c :   1.0
221c :   1.0
223c :   1.0
224c :   1.0
235c :   1.0
241c :   1.0
 83c :   1.0
 98c :   1.0
```

1.5. Exploring a Constraint - model.thermal.capacity_generation_constraint

Run

```python
model.thermal.capacity_generation_constraint.pprint()
```

an in this way, you can obtain the constraint that limits the output generation of each thermal power plant (Variable) for each time-step to the installed capacity (Variable):

```python
capacity_generation_constraint : Size=9672, Index=h*thermal.plants_set,
Active=True
Key           : Lower : Body
: Upper : Active
  (1, '106c') :  -Inf :   thermal.generation[1,106c] -
thermal.plant_installed_capacity[106c] :   0.0 :   True
  (1, '147c') :  -Inf :   thermal.generation[1,147c] -
thermal.plant_installed_capacity[147c] :   0.0 :   True
  (1, '162c') :  -Inf :   thermal.generation[1,162c] -
thermal.plant_installed_capacity[162c] :   0.0 :   True
  (1, '163c') :  -Inf :   thermal.generation[1,163c] -
thermal.plant_installed_capacity[163c] :   0.0 :   True
  (1, '167c') :  -Inf :   thermal.generation[1,167c] -
thermal.plant_installed_capacity[167c] :   0.0 :   True
  (1, '170c') :  -Inf :   thermal.generation[1,170c] -
thermal.plant_installed_capacity[170c] :   0.0 :   True
  (1, '221c') :  -Inf :   thermal.generation[1,221c] -
thermal.plant_installed_capacity[221c] :   0.0 :   True
  (1, '223c') :  -Inf :   thermal.generation[1,223c] -
thermal.plant_installed_capacity[223c] :   0.0 :   True
  (1, '224c') :  -Inf :   thermal.generation[1,224c] -
thermal.plant_installed_capacity[224c] :   0.0 :   True
  (1, '235c') :  -Inf :   thermal.generation[1,235c] -
thermal.plant_installed_capacity[235c] :   0.0 :   True
  (1, '241c') :  -Inf :   thermal.generation[1,241c] -
thermal.plant_installed_capacity[241c] :   0.0 :   True
   (1, '83c') :  -Inf :     thermal.generation[1,83c] -
thermal.plant_installed_capacity[83c] :   0.0 :   True
   (1, '98c') :  -Inf :     thermal.generation[1,98c] -
thermal.plant_installed_capacity[98c] :   0.0 :   True
  (2, '106c') :  -Inf :   thermal.generation[2,106c] -
thermal.plant_installed_capacity[106c] :   0.0 :   True
  ...
```

Notice that the index of each constraint and variable in thos case is given by `(time_step, plant_name)`.