
DispaSET Documentation

Release v2.3

Sylvain Quoilin

Feb 16, 2020

Contents

1	Downloading Dispa-SET	3
2	How to cite	5
3	Documentation	7
4	Main contributors:	9
5	Contents	11
6	Indices and tables	75
	Python Module Index	77
	Index	79

The Dispa-SET model is an open-source unit commitment and optimal dispatch model focused on the balancing and flexibility problems in European grids. Its pre and post-processing tools are written in Python and the main solver can be called via GAMS or via PYOMO. The selected Mixed-Integer Linear Programming (MILP) solver is CPLEX.

Dispa-SET is mainly developed within the Joint Research Centre of the EU Commission, in close collaboration with the University of Liège and the KU Leuven (Belgium).

CHAPTER 1

Downloading Dispa-SET

The public version of Dispa-SET can be downloaded in the *Releases* section or from its github repository (using the Clone or Download button on the right side of the screen): <https://github.com/energy-modelling-toolkit/Dispa-SET>

CHAPTER 2

How to cite

Depending on the version that was used, one of the following JRC technical reports can be selected to cite Dispa-SET:

- Kavvadias, K., Hidalgo Gonzalez, I., Zucker, A. and Quoilin, S., Integrated modelling of future EU power and heat systems: The Dispa-SET v2.2 open-source model, JRC Technical Report, EU Commission, 2018
- Quoilin, S., Hidalgo Gonzalez, I. and Zucker, A., Modelling Future EU Power Systems Under High Shares of Renewables: The Dispa-SET 2.1 open-source model, JRC Technical Report, EU Commission, 2017
- Hidalgo González, I., Quoilin, S. and Zucker, A., Dispa-SET 2.0: unit commitment and power dispatch model, Tech. rep., Publications Office of the European Union, 2014.

CHAPTER 3

Documentation

A pdf documentation of the model is available in the 2017 JRC technical report: [Modelling Future EU Power Systems Under High Shares of Renewables](#).



In addition, the latest model documentation can be obtained by running sphinx in the Docs folder of the project or by consulting the online documentation. This documentation corresponds to the latest available public version of Dispa-SET: <http://www.dispaset.eu/latest/index.html>

CHAPTER 4

Main contributors:

- [Sylvain Quoilin](#) (KU Leuven, Belgium))
- Konstantinos Kavvadias (Joint Research Centre, EU Commission)
- Matija Pavičević (KU Leuven, Belgium)

5.1 Overview

Organization Joint Research Centre, European Commission

Version 2.3

Git Revision v2.3

Date Feb 16, 2020

The Dispa-SET model is mainly developed within the “Joint Research Centre” of the European Commission and focuses on the balancing and flexibility problems in European grids¹.

It is written in GAMS an Python (Pyomo) and uses csv files for input data handling. The optimisation is defined as a Linear Programming (LP) or Mixed-Integer Linear Programming (MILP) problem, depending on the desired level of accuracy and complexity. Continuous variables include the individual unit dispatched power, the shedded load and the curtailed power generation. The binary variables are the commitment status of each unit. The main model features can be summarized as follows:

5.1.1 Features

- Minimum and maximum power for each unit
- Power plant ramping limits
- Reserves up and down
- Minimum up/down times
- Load Shedding
- Curtailment
- Pumped-hydro storage

¹ Quoilin, S., Hidalgo Gonzalez, I., & Zucker, A. (2017). Modelling Future EU Power Systems Under High Shares of Renewables: The Dispa-SET 2.1 open-source models. Publications Office of the European Union.

- Non-dispatchable units (e.g. wind turbines, run-of-river, etc.)
- Start-up, ramping and no-load costs
- Multi-nodes with capacity constraints on the lines (congestion)
- Constraints on the targets for renewables and/or CO2 emissions
- Yearly schedules for the outages (forced and planned) of each units
- CHP power plants and thermal storage
- Different clustering methods

The demand is assumed to be inelastic to the price signal. The MILP objective function is therefore the total generation cost over the optimization period.

5.1.2 Libraries used and requirements

- [Python 3.7](#)
- [pyomo](#) Optimization object library, interface to LP solver (e.g. CPLEX)
- [pandas](#) for input and result data handling
- [matplotlib](#) for plotting
- [GAMS_api](#) for the communication with GAMS

5.1.3 Dispa-SET in the scientific literature

In the past years, Dispa-SET has been used in various scientific works covering different geographical areas and with different focus points. The works for which scientific articles have been published are summarized hereunder:

- Contribution of hydropower for flexibility services in the European power system².
- Ongoing work aiming at coupling the JRC-EU-TIMES model with Dispa-SET by generating simplified variable RES flexibility constraints³⁴.
- Impact of Electric Vehicle deployment in The Netherlands⁵.
- Open-source model of the Balkans area, with some simulations involving high shares of renewables⁶⁷.
- Available technical flexibility to balance variable RES generation in Belgium⁸
- Comparison of clustering formulations⁹¹⁰

² Sánchez Pérez, A. (2017), Modelling Hydropower in detail to assess its contribution to flexibility services in the European power system. Master Thesis, University of Utrecht, Netherlands.

³ Quoilin, S., Nijs, W., Gonzalez, I. H., Zucker, A. and Thiel, C. (2015), Evaluation of simplified flexibility evaluation tools using a unit commitment model, In 12th International Conference on the European Energy Market (EEM), pp. 1-5.

⁴ Quoilin, S., Nijs, W. and Zucker, A. (2017), Evaluating flexibility and adequacy in future EU power systems: model coupling and long-term forecasting, In Proceedings of the 2017 ECOS Conference, San Diego.

⁵ Beltramo, A., Julea, A., Refa, N., Drossinos, Y., Thiel, C. and Quoilin, S. (2017), 'Using electric vehicles as flexible resource in power systems: A case study in the Netherlands, In 14th International Conference on the European Energy Market (EEM).

⁶ Pavičević, M., Tomić, I., Quoilin, S., Zucker, A. and Pukšec, T. and Krajačić, G. (2017), Applying the Dispa-SET model on the Western Balkans power systems, In Proceedings of the 2017 SDEWES Conference

⁷ Tomić, I., Pavičević, M., Quoilin, S., Zucker, A., Krajačić, G., Pukšec, T. and Duić, N. (2017), Applying the Dispa-SET model on the seven countries from the South East Europe, In 8th Energy Planning and Modeling of Energy Systems-Meeting, Belgrade

⁸ Quoilin, S., Gonzalez Vazquez, I., Zucker, A., and Thiel, C. (2014). Available technical flexibility for balancing variable renewable energy sources: case study in Belgium. Proceedings of the 9th Conference on Sustainable Development of Energy, Water and Environment Systems.

⁹ Pavičević, M., Quoilin, S. and Pukšec, T., (2018). Comparison of Different Power Plant Clustering Approaches for Modeling Future Power Systems, Proceedings of the 3rd SEE SDEWES Conference, Novi Sad.

¹⁰ Pavičević, M., Kavvadias, K. and Quoilin, S. (2018). Impact of model formulation on power system simulations - Example with the Dispa-SET Balkans model, EMP-E conference 2018: Modelling Clean Energy Pathways, Brussels.

5.1.4 Ongoing developments

The Dispa-SET project is relatively recent, and a number of improvements will be brought to the project in a close future:

- Grid constraints (DC power-flow)
- Stochastic scenarios
- Modelling of investment and capacity expansion
- Modeling of the ancillary markets

5.1.5 Licence

Dispa-SET is a free software licensed under the “European Union Public Licence” EUPL v1.2. It can be redistributed and/or modified under the terms of this license.

5.1.6 Main Developers

- [Sylvain Quoilin](#) (University of Liège, KU Leuven)
- Konstantinos Kavvadias (European Commission, Institute for Energy and Transport)
- Matija Pavičević (KU Leuven, Belgium)

5.1.7 References

5.2 Releases

Major stable releases:

- [Dispa-SET v2.3](#)
- [Dispa-SET v2.2](#)
- [Dispa-SET v2.1](#)
- [Dispa-SET v2.0](#)

5.2.1 Changelog

Version 2.3

- **Input Data:**
 - A complete EU dataset has been included to the repository for the year 2016.
 - More information: *[Dispa-SET for the EU28](#)*.
- **Reformulation of the reserve constraints:**
 - Secondary reserves are now covered by spinning units only.
 - Tertiary reserves can also be covered by quick start units.

- In total, three different reserve markets are now considered: Secondary up; Secondary down; and Tertiary up
- **Implementation of a new formulation (integer clustering) for power plant related constraints. This formulation divides the**
 - Standard formulation: low capacity or highly flexible units are merged
 - No clustering: all units are considered individually
 - LP clustering: all units are aggregated by technology and binary constraints are removed
 - Integer clustering: a representative unit is considered for each technology and multiplied N times.
- **Improved pre-processing:**
 - Improved log message during input data checks
 - New config files to test the different clustering methods
 - Added functions to perform parametric studies
 - Example scripts for Monte Carlo analyses using latin hypercube samplings
- **Improved post-processing:**
 - Netting interconnections in dispatch plots
 - New colour palette and polished dispatch plot
 - New fuels included
 - Improved representation of curtailment
- **External dependencies:**
 - Removed pre-compiled libraries for unix systems
 - Use of the low-level GAMS API (<https://github.com/kavvkon/gams-api>)
- **Python 3.7:**
 - Dispa-SET now runs exclusively on Python 3.7.
 - The compatibility with previous Python versions (2.7, 3.6) is not guaranteed anymore.
- **Miscellaneous:**
 - Unit tests on travis (<https://travis-ci.org/energy-modelling-toolkit/Dispa-SET>)
 - Bug fixes

Version 2.2

- Inclusion of CHP, power2heat and thermal storage (these new features can be tested by running the config file for Cyprus: 'ConfigFiles/ConfigCY.xlsx')
- Bug fixes
- Improved user interface

Version 2.1

- Major refactoring of the folder structure
- New data included in the database
- Inclusion of the LP formulation (in addition to the MILP)

Version 2.0

First public version of the Dispa-SET model.

5.3 Getting Started

This short tutorial describes the main steps to get a practical example of Dispa-SET running.

5.3.1 Prerequisites

Install Python 3.7, with full scientific stack. The [Anaconda](#) distribution is recommended since it comprises all the required packages. If Anaconda is not used, the following libraries and their dependencies should be installed manually:

- future >= 0.15
- click >= 3.3
- numpy>=1.10
- scipy>=0.15
- matplotlib>=1.5.1
- pandas>= 0.19
- xlrd >= 0.9
- pickle
- pyyaml
- pytest

Using Dispa-SET with GAMS:

Dispa-SET is primarily designed to run with GAMS and therefore requires GAMS to be installed with a valid user licence. Currently, only the 64-bit version of GAMS is supported in Dispa-SET!

The GAMS api for python has been pre-compiled in the “Externals” folder and is usable for Windows 64 bit systems. If the pre-compiled binaries are not available or could not be loaded, they must be installed manually using following command in the Anaconda prompt:

```
pip install gdxcc gamsxcc optcc
```

Alternatively, the gams python api can also be compiled from the source provided in the GAMS installation folder (e.g. “C:\GAMS\win64\24.3\apifiles\Python\api”):

```
python setup.py install
```

NB: For Windows users, the manual api compilation might require the installation of a C++ compiler for Python. This corresponds to the typical error message: “Unable to find vcvarsall.bat”. This can be solved by installing the freely available “Microsoft Visual C++ Compiler for Python”. In some cases the path to the compiler must be added to the PATH windows environment variable (e.g. C:\Program Files\Common Files\Microsoft Visual C++ for Python\9.0)

The api requires the path to the gams installation folder. The “get_gams_path()” function of dispa-set performs a system search to automatically detect this path. If it is not successful, the user is prompted for the proper installation path.

Using Dispa-SET with PYOMO:

Warning: The PYOMO version of Dispa-SET is currently outdated. The changes implemented in Dispa-SET version 2.2 and 2.3 will not work in the Pyomo solver. It is recommended using the GAMS solver to benefit from the latest improvements of the model.

- Install pyomo

```
pip install pyomo
```

- Install a solver and add it to the PATH environment variable (e.g. if cplex is installed, the “cplex” command should be callable from any command prompt).

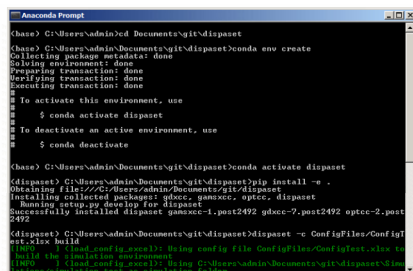
5.3.2 Step-by-step example of a Dispa-SET run

This section describes the pre-processing and the solving phases of a Dispa-SET run. Three equivalent methods are described in the next sections:

- Using the command line interface
- Using the Dispa-SET API
- Using GAMS

1. Using the command line interface

Dispa-SET can be run from the command line. To that aim, open a terminal window and change the directory to the Dispa-SET root folder.



```

Anaconda Prompt
C:\Users\admin\Documents>git\dispa-set
C:\Users\admin\Documents>git\dispa-set\conda env create
Collecting package metadata: done
Solving environment: done
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

To activate this environment, use
$ conda activate dispa-set
To deactivate an active environment, use
$ conda deactivate

C:\Users\admin\Documents>git\dispa-set\conda activate dispa-set
C:\Users\admin\Documents>git\dispa-set\pip install -e .
Obtaining file:///C:/Users/admin/Documents/git/dispa-set
Installing collected packages: gams, pyomo, optc, dispa-set
Running setup.py develop for dispa-set
Successfully installed dispa-set-1.post2492 gams-7.post2492 optc-2.post2492

C:\Users\admin\Documents>git\dispa-set\dispa-set -n ConfigFiles\Conf1\
out.cplex build
Building dispa-set. Using config file ConfigFiles\Conf1\out.cplex
to build the extension module.
Note: If you are using the Anaconda Prompt, you must run the command
'conda activate dispa-set' before running this command.

```

1.0. Install Dispa-SET and the required dependencies

Use the following commands in a terminal (Anaconda prompt in Windows):

```
conda env create # Automatically creates environment based on environment.yml
conda activate dispaset
pip install -e . # Install editable local version
```

The above commands create a dedicated environment so that your anconda configuration remains clean from the required dependencies installed. If preferred, the Gams libraries can also be installed without creating a dedicated environment. In that case, replace the above commands with these ones:

```
pip install gamsxcc gdxcc optcc
python setup.py install
```

To check that everything runs fine, you can build and run a test case by typing:

```
dispaset -c ConfigFiles/ConfigTest.xlsx build simulate
```

1.1. Check the configuration file

Dispa-SET runs are defined in dedicated excel configuration files stored in the “ConfigFiles” folder. The configuration file “ConfigTest.xlsx” is provided for testing purposes. It generates a 10-days optimisation using data relative a fictitious power system composed of two zones Z1 and Z2.

1.2. Pre-processing

From the command line, specify the configuration file to be used as an argument, the solver (Pyomo or GAMS) and the actions to be performed. Within the “Dispa-SET” folder, run:

```
dispaset -c ./ConfigFiles/ConfigTest.xlsx build
```

1.3. Check the simulation environment

The simulation environment folder is defined in the configuration file. In the test example it is set to “Simulations/simulation_test”. The simulation inputs are written in three different formats: excel (34 excel files), Python (Inputs.p) and GAMS (Inputs.gdx).

1.4. Run the optimisation

The simulation can be started directly from the main DispaSet python file after the pre-processing phase. From the “Dispa-SET” folder, run:

```
dispaset -c ./ConfigFiles/ConfigTest.xlsx simulate
```

This runs the optimisation, and stores the results in the same folder. Note that this can only work if the simulation has been pre-processed before (step 1.2). It is possible to combine the pre-processing and simulation step in one command:

```
dispaset -c ./ConfigFiles/ConfigTest.xlsx build simulate
```

The same action can be performed using the PYOMO solver. In that case, the “-g” argument must be changed into “-p”:

```
dispaset -p -c ./ConfigFiles/ConfigTest.xlsx build simulate
```

2. Using the Dispa-SET API.

The steps to run a model can be also performed directly in python, by importing the Dispa-SET library. An example file (“build_and_run.py”) is available in the “scripts/” folder.

To run the commands below, the Gams libraries are required. Install them using the following command in an Anaconda prompt:

```
pip install gamsxcc gdxcc optcc
```

After checking the configuration file “ConfigTest.xlsx” (in the “ConfigFiles” folder). Run the following python commands:

2.1 Import Dispa-SET:

```
import dispaset as ds
```

2.2 Load the configuration file:

```
config = ds.load_config_excel('ConfigFiles/ConfigTest.xlsx')
```

2.3 Build the simulation environment (Folder that contains the input data and the simulation files required for the solver):

```
SimData = ds.build_simulation(config)
```

2.4 Solve using GAMS:

```
r = ds.solve_GAMS(config['SimulationDirectory'], config['GAMS_folder'])
```

A more detailed description of the Dispa-SET functions is available in the API section.

3. Using GAMS

It is sometimes useful to run the dispa-SET directly in GAMS (e.g. for debugging purposes). In that case, the pre-processing must be run first (steps 1.2 or 2.1, 2.2 and 2.3) and the gams file generated in the simulation folder can be used to run the optimization.

Using the GAMS graphical user interface:

From the simulation folder (defined in the config file), the Dispa-SET model can be run following the instruction below:

1. Open the UCM.gpr project file in GAMS
2. From GAMS, open the UCM_h.gmx model file
3. Run the model in GAMS.

The result file is written in the.gdx format and stored in the Simulation folder, together with all input files.

Using the GAMS command line:

GAMS can also be run from the command line (this is the only option for the Linux version).

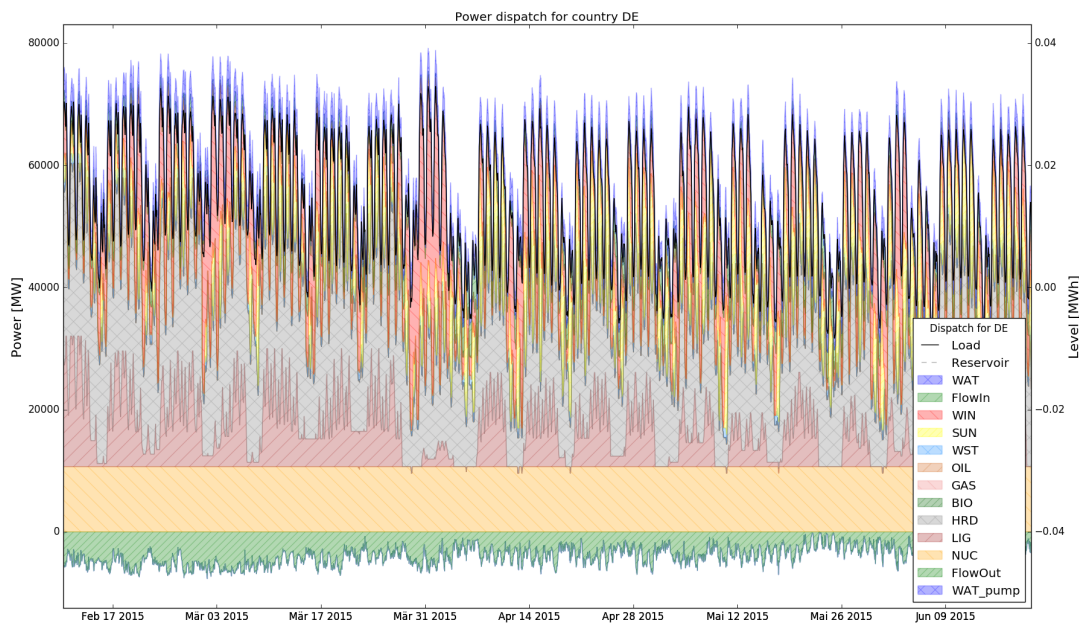
1. Make sure that the gams binary is in the system PATH
2. From the simulation environment folder, run:

```
gams UCM_h.gms
```

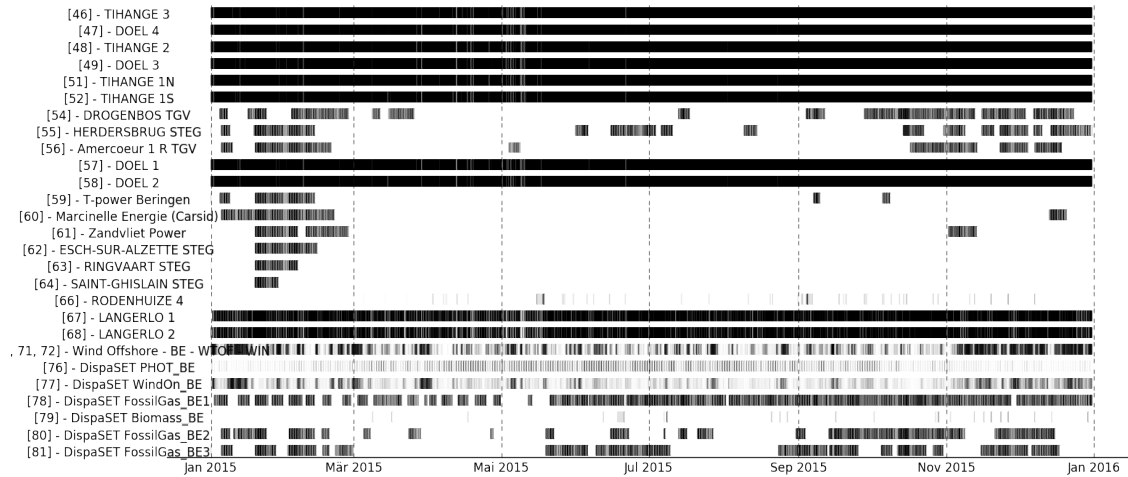
5.3.3 Postprocessing and result display

Various functions and tools are provided within the PostProcessing.py file to load, analyse and plot the simulation results. The use of these functions is illustrated into the “Read_results_notebook.ipynb” Notebook or in the “read_results.py” script, which can be run by changing the path to the simulation folder. The type of results provided by the post-processing is illustrated hereunder.

The power dispatch can be plotted for each simulated zone. In this plot, the units are aggregated by fuel type. The power consumed by storage units and the exportations are indicated as negative values.



It is also interesting to display the results at the unit level to gain deeper insights regarding the dispatch. In that case, a plot is generated, showing the commitment status of all units in a zone at each timestep. Both the dispatch plot and the commitment plot can be called using the CountryPlots function.



Some aggregated statistics on the simulations results can also be obtained, including the number of hours of congestion in each interconnection line, the yearly energy balances for each zone, the amount of lost load, etc.

Aggregated statistics for the considered area:

Total consumption: 1227.07310992 TWh

Peak load: 203182.461067 MW

Net importations: -42.20072928 TWh

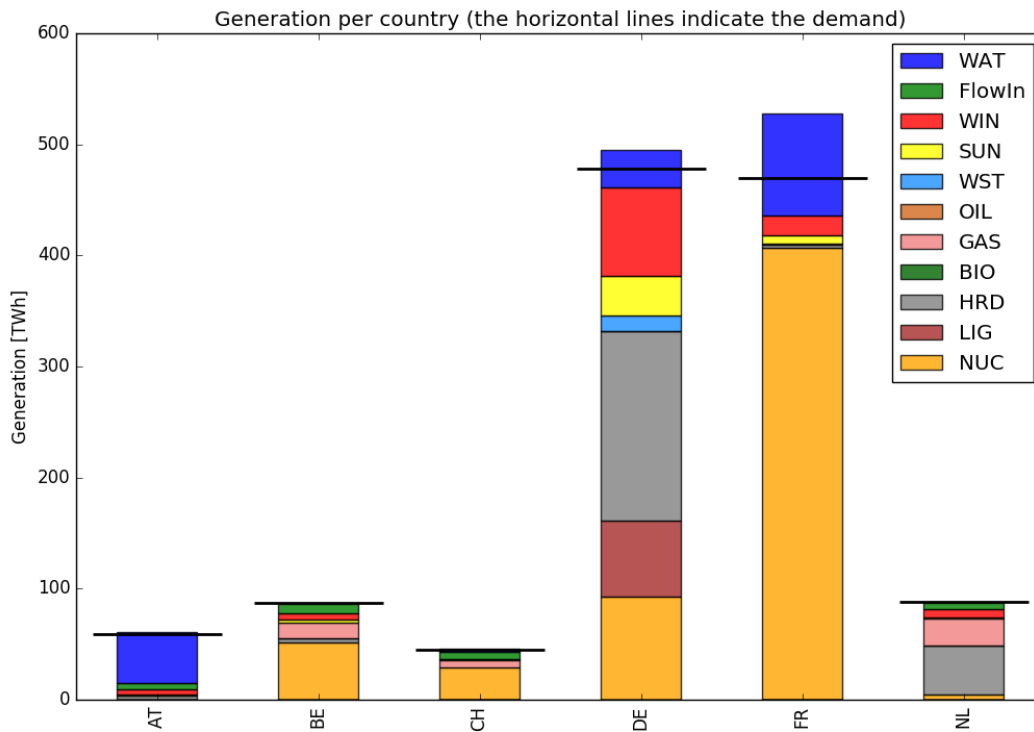
Country-Specific values (in TWh or in MW):

	Demand	PeakLoad	NetImports	LoadShedding	Curtailment
AT	59.375448	10144.000000	5.144132	NaN	NaN
BE	86.971154	13632.250000	8.911190	NaN	NaN
CH	44.694098	7794.262468	7.199527	NaN	NaN
DE	478.030824	76212.250000	-17.260122	NaN	NaN
FR	470.075612	90588.000000	-51.878128	NaN	NaN
NL	87.925973	16285.500000	5.682672	NaN	NaN

Number of hours of congestion on each line:

```
{'AT -> CH': 5917,
 'AT -> DE': 430,
 'BE -> FR': 62,
 'BE -> NL': 344,
 'CH -> AT': 720,
 'CH -> DE': 15,
 'CH -> FR': 56,
 'DE -> AT': 1522,
 'DE -> CH': 4378,
 'DE -> NL': 2803,
 'FR -> BE': 2689,
 'FR -> CH': 7665,
 'NL -> BE': 1403,
 'NL -> DE': 60}
```


The yearly energy balance per fuel or per technology is also useful to compare the energy mix in each zone. This can be plotted using the EnergyBarPlot function, with the following results:



5.4 Model Description

The model is expressed as a MILP or LP problem. Continuous variables include the individual unit dispatched power, the shedded load and the curtailed power generation. The binary variables are the commitment status of each unit. The main model features can be summarized as follows:

5.4.1 Variables

Sets

Name	Description
f	Fuel types
h	Hours
i	Time step in the current optimization horizon
l	Transmission lines between nodes
mk	{DA: Day-Ahead, 2U: Reserve up, 2D: Reserve Down}
n	Zones within each country (currently one zone, or node, per country)
p	Pollutants
t	Power generation technologies
tr	Renewable power generation technologies
u	Units
s(u)	Storage units (including hydro reservoirs)
chp(u)	CHP units

Parameters

Name	Units	Description
AvailabilityFactor(u,i)	%	Percentage of nominal capacity available
CHPPowerLossFactor(u)	%	Power loss when generating heat
CHPPowerToHeat(u)	%	Nominal power-to-heat factor
CHPMaxHeat(chp)	MW	Maximum heat capacity of chp plant
CHPType	n.a.	CHP Type
CommittedInitial(u)	n.a.	Initial commitment status
CostFixed(u)	EUR/h	Fixed costs
CostLoadShedding(n,h)	EUR/MWh	Shedding costs
CostRampDown(u)	EUR/MW	Ramp-down costs
CostRampUp(u)	EUR/MW	Ramp-up costs
CostShutDown(u)	EUR/u	Shut-down costs for one unit
CostStartUp(u)	EUR/u	Start-up costs for one unit
CostVariableH(u,i)	EUR/MWh	Variable costs
CostHeatSlack(chp,h)	EUR/MWh	Cost of supplying heat via other means
Curtailment(n)	n.a.	Curtailment {binary: 1 allowed}
Demand(mk,n,i)	MW	Hourly demand in each zone
Efficiency(u)	%	Power plant efficiency
EmissionMaximum(n,p)	EUR/tP	Emission limit per zone for pollutant p
EmissionRate(u,p)	tP/MW	Emission rate of pollutant p from unit u
Fuel(u,f)	n.a.	Fuel type used by unit u {binary: 1 u uses f}
HeatDemand(chp,h)	MWh/u	Heat demand profile for chp units
K_QuickStart(n)	n.a.	Part of the reserve that can be provided by offline quickstart units
LineNode(l,n)	n.a.	Line-zone incidence matrix {-1,+1}
LoadShedding(n,h)	MW	Load that may be shed per zone in 1 hour
Location(u,n)	n.a.	Location {binary: 1 u located in n}
Nunits(u)	n.a.	Number of units inside the cluster
OutageFactor(u,h)	%	Outage factor (100 % = full outage) per hour
PartLoadMin(u)	%	Percentage of minimum nominal capacity
PowerCapacity(u)	MW/u	Installed capacity
PowerInitial(u)	MW/u	Power output before initial period

Continued on next page

Table 5.1 – continued from previous page

Name	Units	Description
PowerMinStable(u)	MW/u	Minimum power for stable generation
PowerMustRun(u)	MW	Minimum power output
PriceTransmission(l,h)	EUR/MWh	Price of transmission between zones
QuickStartPower(u,h)	MW/h/u	Available max capacity for tertiary reserve
RampDownMaximum(u)	MW/h/u	Ramp down limit
RampShutDownMaximum(u)	MW/h/u	Shut-down ramp limit
RampStartUpMaximum(u)	MW/h/u	Start-up ramp limit
RampUpMaximum(u)	MW/h/u	Ramp up limit
Reserve(t)	n.a.	Reserve provider {binary}
StorageCapacity(s)	MWh/u	Storage capacity (reservoirs)
StorageChargingCapacity(s)	MW/u	Maximum charging capacity
StorageChargingEfficiency(s)	%	Charging efficiency
StorageDischargeEfficiency(s)	%	Discharge efficiency
StorageInflow(s,h)	MWh/u	Storage inflows
StorageInitial(s)	MWh	Storage level before initial period
StorageMinimum(s)	MWh/u	Minimum storage level
StorageOutflow(s,h)	MWh/u	Storage outflows (spills)
StorageProfile(u,h)	MWh	Storage long-term level profile
Technology(u,t)	n.a.	Technology type {binary: 1: u belongs to t}
TimeDownMinimum(u)	h	Minimum down time
TimeUpMinimum(u)	h	Minimum up time
VOLL()	EUR/MWh	Value of lost load

NB: When the parameter is expressed per unit (“/u”), its value must be provided for one single unit (even in the case of a clustered formulation).

Optimization Variables

Name	Units	Description
Committed(u,h)	n.a.	Unit committed at hour h {1,0}
CostStartUpH(u,h)	EUR	Cost of starting up
CostShutDownH(u,h)	EUR	Cost of shutting down
CostRampUpH(u,h)	EUR	Ramping cost
CostRampDownH(u,h)	EUR	Ramping cost
CurtailedPower(n,h)	MW	Curtailed power at node n
Flow(l,h)	MW	Flow through lines
Heat(chp,h)	MW	Heat output by chp plant
HeatSlack(chp,h)	MW	Heat satisfied by other sources
Power(u,h)	MW	Power output
PowerMaximum(u,h)	MW	Power output
PowerMinimum(u,h)	MW	Power output
Reserve_2U(u,h)	MW	Spinning reserve up
Reserve_2D(u,h)	MW	Spinning reserve down
Reserve_3U(u,h)	MW	Non spinning quick start reserve up
ShedLoad(n,h)	MW	Shed load
StorageInput(s,h)	MWh	Charging input for storage units
StorageLevel(s,h)	MWh	Storage level of charge
Spillage(s,h)	MWh	Spillage from water reservoirs
SystemCost(h)	EUR	Total system cost
LL_MaxPower(n,h)	MW	Deficit in terms of maximum power
LL_RampUp(u,h)	MW	Deficit in terms of ramping up for each plant
LL_RampDown(u,h)	MW	Deficit in terms of ramping down
LL_MinPower(n,h)	MW	Power exceeding the demand
LL_2U(n,h)	MW	Deficit in reserve up
LL_3U(n,h)	MW	Deficit in reserve up - non spinning
LL_2D(n,h)	MW	Deficit in reserve down

Integer Variables

Name	Units	Description
Committed(u,h)	n.a.	Number of unit committed at hour h {1 0} or integer
StartUp(u,h)	n.a.	Number of unit startups at hour h {1 0} or integer
ShutDown(u,h)	n.a.	Number of unit shutdowns at hour h {1 0} or integer

5.4.2 Optimisation model

The aim of this model is to represent with a high level of detail the short-term operation of large-scale power systems solving the so-called unit commitment problem. To that aim we consider that the system is managed by a central operator with full information on the technical and economic data of the generation units, the demands in each node, and the transmission network.

The unit commitment problem considered in this report is a simplified instance of the problem faced by the operator in charge of clearing the competitive bids of the participants into a wholesale day-ahead power market. In the present formulation the demand side is an aggregated input for each node, while the transmission network is modelled as a transport problem between the nodes (that is, the problem is network-constrained but the model does not include the calculation of the optimal power flows).

The unit commitment problem consists of two parts: i) scheduling the start-up, operation, and shut down of the available generation units, and ii) allocating (for each period of the simulation horizon of the model) the total power demand among the available generation units in such a way that the overall power system costs is minimized. The first part of the problem, the unit scheduling during several periods of time, requires the use of binary variables in order to represent the start-up and shut down decisions, as well as the consideration of constraints linking the commitment status of the units in different periods. The second part of the problem is the so-called economic dispatch problem, which determines the continuous output of each and every generation unit in the system. Therefore, given all the features of the problem mentioned above, it can be naturally formulated as a mixed-integer linear program (MILP).

Since our goal is to model a large European interconnected power system, we have implemented a so-called tight and compact formulation, in order to simultaneously reduce the region where the solver searches for the solution and increase the speed at which the solver carries out that search. Tightness refers to the distance between the relaxed and integer solutions of the MILP and therefore defines the search space to be explored by the solver, while compactness is related to the amount of data to be processed by the solver and thus determines the speed at which the solver searches for the optimum. Usually tightness is increased by adding new constraints, but that also increases the size of the problem (decreases compactness), so both goals contradict each other and a trade-off must be found.

Objective function

The goal of the unit commitment problem is to minimize the total power system costs (expressed in EUR in equation), which are defined as the sum of different cost items, namely: start-up and shut-down, fixed, variable, ramping, transmission-related and load shedding (voluntary and involuntary) costs.

$$\begin{aligned}
 \min \sum_{u,n,i} & \left[CostStartUp_{u,i} + CostShutDown_{u,i} + CostFixed_u \cdot Committed_{u,i} \right. \\
 & + CostVariable_{u,i} \cdot Power_{u,i} + CostRampUp_{u,i} + CostRampDown_{u,i} \\
 & + PriceTransmission_{i,l} \cdot Flow_{i,l} + (CostLoadShedding_{i,n} \cdot ShedLoad_{i,n}) \\
 & + \sum_{chp} CostHeatSlack_{chp,i} \cdot HeatSlack_{chp,i} \\
 & + \sum_{chp} CostVariable_{chp,i} \cdot CHPPowerLossFactor_{chp} \cdot Heat_{chp,i} \\
 & + VOLL_{Power} \cdot (LL_{MaxPower,i,n} + LL_{MinPower,i,n}) \\
 & + VOLL_{Reserve} \cdot (LL_{2U,i,n} + LL_{2D,i,n} + LL_{3U,i,n}) \\
 & \left. + VOLL_{Ramp} \cdot (LL_{RampUp,p,u,i} + LL_{RampDown,u,i}) \right]
 \end{aligned}$$

The costs can be broken down as:

- Fixed costs: depending on whether the unit is on or off.
- Variable costs: stemming from the power output of the units.
- Start-up costs: due to the start-up of a unit.
- Shut-down costs: due to the shut-down of a unit.
- Ramp-up: emerging from the ramping up of a unit.
- Ramp-down: emerging from the ramping down of a unit.
- Load shed: due to necessary load shedding.
- Transmission: depending of the flow transmitted through the lines.
- Loss of load: power exceeding the demand or not matching it, ramping and reserve.

The variable production costs (in EUR/MWh), are determined by fuel and emission prices corrected by the efficiency (which is considered to be constant for all levels of output in this version of the model) and the emission rate of the unit (equation):

$$CostVariable_{u,h} = Markup_{u,h} + \sum_{n,f} \left(\frac{Fuel_{u,f} \cdot FuelPrice_{n,f,h} \cdot Location_{u,n}}{Efficiency_u} \right) + \sum_p (EmissionRate_{u,p} \cdot PermitPrice_p)$$

The variable cost includes an additional mark-up parameter that can be used for calibration and validation purposes.

From version 2.3, Dispa-SET uses a 3 integers formulations of the up/down status of all units. According to this formulation, the number of start-ups and shut-downs is at each time step is computed by:

$$Committed_{u,i} - Committed_{u,i-1} = StartUp_{u,i} - ShutDown_{u,i}$$

The start-up and shut-down costs are positive variables, calculated from the number of startups/shutdowns at each time step:

$$CostStartUp_{u,i} = CostStartUp_u \cdot StartUp_{u,i}$$

$$CostShutDown_{u,i} = CostShutDown_u \cdot ShutDown_{u,i}$$

Ramping costs are defined as positive variables (i.e. negative costs are not allowed) and are computed with the following equations:

$$CostRampUp_{u,i} \geq CostRampUp_u \cdot (Power_{u,i} - Power_{u,i-1})$$

$$CostRampDown_{u,i} \geq CostRampDown_u \cdot (Power_{u,i-1} - Power_{u,i})$$

It should be noted that in case of start-up and shut-down, the ramping costs are added to the objective function. Using start-up, shut-down and ramping costs at the same time should therefore be performed with care.

In the current formulation, all other costs (fixed and variable costs, transmission costs, load shedding costs) are considered as exogenous parameters.

As regards load shedding, the model considers the possibility of voluntary load shedding resulting from contractual arrangements between generators and consumers. Additionally, in order to facilitate tracking and debugging of errors, the model also considers some variables representing the capacity the system is not able to provide when the minimum/maximum power, reserve, or ramping constraints are reached. These lost loads are a very expensive last resort of the system used when there is no other choice available. The different lost loads are assigned very high values (with respect to any other costs). This allows running the simulation without infeasibilities, thus helping to detect the origin of the loss of load. In a normal run of the model, without errors, all these variables are expected to be equal to zero.

Day-ahead energy balance

The main constraint to be met is the supply-demand balance, for each period and each zone, in the day-ahead market (equation). According to this restriction, the sum of all the power produced by all the units present in the node (including the power generated by the storage units), the power injected from neighbouring nodes, and the curtailed power from intermittent sources is equal to the load in that node, plus the power consumed for energy storage, minus

the load interrupted and the load shed.

$$\begin{aligned}
 & \sum_u (Power_{u,i} \cdot Location_{u,n}) \\
 & + \sum_l (Flow_{l,i} \cdot LineNode_{l,n}) \\
 & = Demand_{DA,n,h} + \sum_r (StorageInput_{s,h} \cdot Location_{s,n}) \\
 & \quad - ShedLoad_{n,i} \\
 & \quad - LL_{MaxPower_{n,i}} + LL_{MinPower_{n,i}}
 \end{aligned}$$

Reserve constraints

Besides the production/demand balance, the reserve requirements (upwards and downwards) in each node must be met as well. In Dispa-SET, three types of reserve requirements are taken into account:

- Upward secondary reserve (2U): reserve that can only be covered by spinning units
- Downward secondary reserve (2D): reserve that can only be covered by spinning units
- Upward tertiary reserve (3U): reserve that can be covered either by spinning units or by quick-start offline units

The secondary reserve capability of committed units is limited by the capacity margin between current and maximum power output:

$$\begin{aligned}
 & Reserve_{2U_{u,i}} \\
 & \leq PowerCapacity_u \cdot AvailabilityFactor_{u,i} \\
 & \quad \cdot (1 - OutageFactor_{u,i}) \cdot Committed_{u,i} \\
 & \quad - Power_{u,i}
 \end{aligned}$$

The same applies to the downwards secondary reserve capability, with an additional term to take into account the downward reserve capability of pumping storage units:

$$\begin{aligned}
 & Reserve_{2D_{u,i}} \\
 & \leq Power_{u,i} - PowerMustRun_{u,i} \cdot Committed_{u,i} \\
 & \quad + (StorageChargingCapacity_u \cdot Nunits_u - StorageInput_{u,i})
 \end{aligned}$$

The quick start (non-spinning) reserve capability is given by:

$$\begin{aligned}
 & Reserve_{3U_{u,i}} \\
 & \leq (Nunits_u - Committed_{u,i}) \cdot QuickStartPower_{u,i}
 \end{aligned}$$

The secondary reserve demand should be fulfilled at all times by all the plants allowed to participate in the reserve market:

$$\begin{aligned}
 & Demand_{2U,n,h} \\
 & \leq \sum_{u,t} (Reserve_{2U_{u,i}} \cdot Technology_{u,t} \cdot Reserve_t \cdot Location_{u,n}) \\
 & \quad + LL_{2U_{n,i}}
 \end{aligned}$$

The same equation applies to downward reserve requirements (2D).

The tertiary reserve can also be provided by non-spinning units. The inequality is thus transformed into:

$$\begin{aligned} & Demand_{3U,n,h} \\ & \leq \sum_{u,t} [(Reserve_{2U,u,i} + Reserve_{3U,u,i}) \\ & \quad \cdot Technology_{u,t} \cdot Reserve_t \cdot Location_{u,n}] \\ & \quad + LL_{3U,n,i} \end{aligned}$$

The reserve requirements are defined by the users. In case no input is provided a default formula is used to evaluate the needs for secondary reserves as a function of the maximum expected load for each day. The default formula is described by:

$$Demand_{2U,n,i} = \sqrt{10 \cdot \max_h (Demand_{DA,n,h}) + 150^2} - 150$$

Downward reserves are defined as 50% of the upward margin:

$$Demand_{2D,n,h} = 0.5 \cdot Demand_{2U,n,h}$$

Power output bounds

The minimum power output is determined by the must-run or stable generation level of the unit if it is committed:

$$\begin{aligned} & PowerMustRun_{u,i} \cdot Committed_{u,i} \\ & \leq Power_{u,i} \end{aligned}$$

In the particular case of CHP unit (extraction type or power-to-heat type), the minimum power is defined for a heat demand equal to zero. If the unit produces heat, the minimum power must be reduced according to the power loss factor and the previous equation is replaced by:

$$\begin{aligned} & PowerMustRun_{chp,i} \cdot Committed_{chp,i} \\ & - StorageInput_{chp,i} \cdot CHPPowerLossFactor_u \\ & \leq Power_{chp,i} \end{aligned}$$

The power output is limited by the available capacity, if the unit is committed:

$$\begin{aligned} & Power_{u,i} \\ & \leq PowerCapacity_u \cdot AvailabilityFactor_{u,i} \\ & \quad \cdot (1 - OutageFactor_{u,i}) \cdot Committed_{u,i} \end{aligned}$$

The availability factor is used for renewable technologies to set the maximum time-dependent generation level. It is set to one for the traditional power plants. The outage factor accounts for the share of unavailable power due to planned or unplanned outages.

Ramping Constraints

Each unit is characterized by a maximum ramp up and ramp down capability. This is translated into the following inequality for the case of ramping up:

$$\begin{aligned} & Power_{u,i} - Power_{u,i-1} \leq \\ & (Committed_{u,i} - StartUp_{u,i}) \cdot RampUpMaximum_u \\ & + StartUp_{u,i} \cdot RampStartUpMaximum_u \\ & - ShutDown_{u,i} \cdot PowerMustRun_{u,i} \\ & + LL_{RampUp_{u,i}} \end{aligned}$$

and for the case of ramping down:

$$\begin{aligned} & Power_{u,i-1} - Power_{u,i} \leq \\ & (Committed_{u,i} - ShutDown_{u,i}) \cdot RampDownMaximum_u \\ & + ShutDown_{u,i} \cdot RampShutDownMaximum_u \\ & - StartUp_{u,i} \cdot PowerMustRun_{u,i} \\ & + LL_{RampDown_{u,i}} \end{aligned}$$

Note that this formulation is valid for both the clustered formulation and the binary formulation. In the latter case (there is only one unit u), if the unit remains committed, the inequality simplifies into:

$$\begin{aligned} & Power_{u,i} - Power_{u,i-1} \leq \\ & RampUpMaximum_u + LL_{RampUp_{u,i}} \end{aligned}$$

If the unit has just been committed, the inequality becomes:

$$\begin{aligned} & Power_{u,i} - Power_{u,i-1} \leq \\ & RampStartUpMaximum_u + LL_{RampUp_{u,i}} \end{aligned}$$

And if the unit has just been stopped:

$$\begin{aligned} & Power_{u,i} - Power_{u,i-1} \leq \\ & -PowerMustRun_{u,i} + LL_{RampUp_{u,i}} \end{aligned}$$

Minimum up and down times

The operation of the generation units is also limited as well by the amount of time the unit has been running or stopped. In order to avoid excessive ageing of the generators, or because of their physical characteristics, once a unit is started up, it cannot be shut down immediately. Reciprocally, if the unit is shut down it may not be started immediately.

To model this in MILP, the number of startups/shutdowns in the last N hours must be limited, N being the minimum up or down time. For the minimum up time, the number of startups during this period cannot be higher than the number of currently committed units:

$$\sum_{ii=i-TimeUpMinimum_u}^i StartUp_{u,ii} \leq Committed_{u,i}$$

i.e. the currently committed units are not allowed to have performed multiple on/off cycles during the last TimeUp-Minimum periods. In case of a binary formulation ($Nunits=1$), if the unit is ON at time i , only one startup is allowed in the last TimeUpMinimum periods. If the unit is OFF at time i , no startup is allowed.

A similar inequality can be written for the minimum down time:

$$\sum_{ii=i-TimeDownMinimum_u}^i ShutDown_{u,ii} \leq Nunits_u - Committed_{u,i}$$

Storage-related constraints

Generation units with energy storage capabilities (mostly large hydro reservoirs and pumped hydro storage units) must meet additional restrictions related to the amount of energy stored. Storage units are considered to be subject to the same constraints as non-storage power plants. In addition to those constraints, storage-specific restrictions are added for the set of storage units (i.e. a subset of all units). These restrictions include the storage capacity, inflow, outflow,

charging, charging capacity, charge/discharge efficiencies, etc. Discharging is considered as the standard operation mode and is therefore linked to the Power variable, common to all units.

The first constraint imposes that the energy stored by a given unit is bounded by a minimum value:

$$StorageMinimum_s \leq StorageLevel_{s,i} \cdot Nunits_s$$

In the case of a storage unit, the availability factor applies to the charging/discharging power, but also to the storage capacity. The storage level is thus limited by:

$$StorageLevel_{s,i} \leq StorageCapacity_s \cdot AvailabilityFactor_{s,i} \cdot Nunits_s$$

The energy added to the storage unit is limited by the charging capacity. Charging is allowed only if the unit is not producing (discharging) at the same time (i.e. if Committed, corresponding to the `{textquotedbl}normal{textquotedbl}` mode, is equal to 0).

$$StorageInput_{s,i} \leq StorageChargingCapacity_s \cdot (Nunits_s - Committed_{s,i})$$

Discharge is limited by the level of charge of the storage unit:

$$\begin{aligned} \frac{Power_{i,s}}{StorageDischargeEfficiency_s} + StorageOutflow_{s,i} \cdot Nunits_s \\ + Spillage_{s,i} - StorageInflow_{s,i} \cdot Nunits_s \\ \leq StorageLevel_{s,i} \end{aligned}$$

It is worthwhile to note that StorageInflow and StorageOutflow must be multiplied by the number of units because they are defined for a single storage plant. On the contrary StorageLevel, Spillage and Power are defined for all units s .

Charge is limited by the level of charge of the storage unit:

$$\begin{aligned} StorageInput_{s,i} \cdot StorageChargingEfficiency_s \\ - StorageOutflow_{s,i} \cdot Nunits_s - Spillage_{s,i} \\ + StorageInflow_{s,i} \cdot Nunits_s \\ \leq StorageCapacity_s \cdot AvailabilityFactor_{s,i} \\ - StorageLevel_{s,i} \end{aligned}$$

Besides, the energy stored in a given period is given by the energy stored in the previous period, net of charges and discharges:

$$\begin{aligned} StorageLevel_{s,i-1} + StorageInflow_{s,i} \cdot Nunits_s \\ + StorageInput_{s,i} \cdot StorageChargingEfficiency_s \\ = StorageLevel_{s,i} + StorageOutflow_{s,i} \cdot Nunits_s \\ + \frac{Power_{s,i}}{StorageDischargeEfficiency_s} \end{aligned}$$

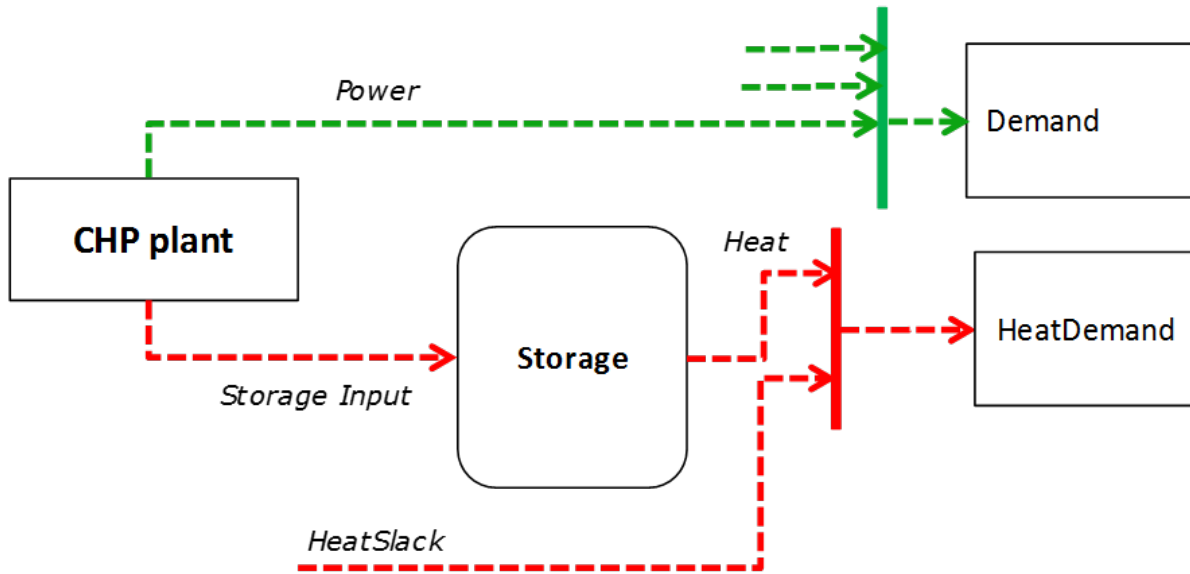
Some storage units are equipped with large reservoirs, whose capacity at full load might be longer than the optimisation horizon. Therefore, a minimum level constraint is required for the last hour of the optimisation, which otherwise would systematically tend to empty the reservoir as much as possible. An exogenous minimum profile is thus provided and the following constraint is applied:

$$\begin{aligned} StorageLevel_{s,N} \geq \min(StorageProfile_{s,N} \\ \cdot AvailabilityFactor_{s,N} \cdot StorageCapacity_s \cdot Nunits_s, \\ StorageLevel_{s,0} + (\sum_{i=1}^N InFlows_{s,i} - \sum_{i=1}^N OutFlows_{s,i}) \cdot Nunits_s) \end{aligned}$$

where N is the last period of the optimization horizon and $StorageProfile$ is a non-dimensional minimum storage level provided as an exogenous input. The minimum is taken to avoid unfeasibilities in case the provided inflows are not sufficient to comply with the imposed storage level at the end of the horizon.

Heat production constraints (CHP plants only)

In DispaSET Power plants can be indicated as CHP satisfying one heat demand. Heat Demand can be covered either by a CHP plant or by alternative heat supply options (Heat Slack).



The following two heat balance constraints are used for any CHP plant type.

$$Heat(chp, i) + HeatSlack(chp, i) = HeatDemand(chp, i)$$

$$StorageInput_{chp, i} \leq CHPMaxHeat_{chp} \cdot Nunits_{chp}$$

The constraints between heat and power production differ for each plant design and explained within the following subsections.

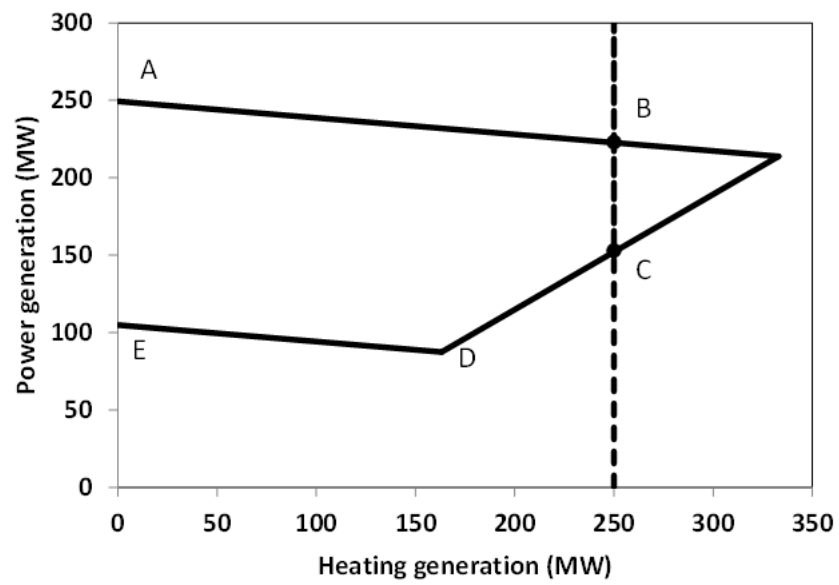
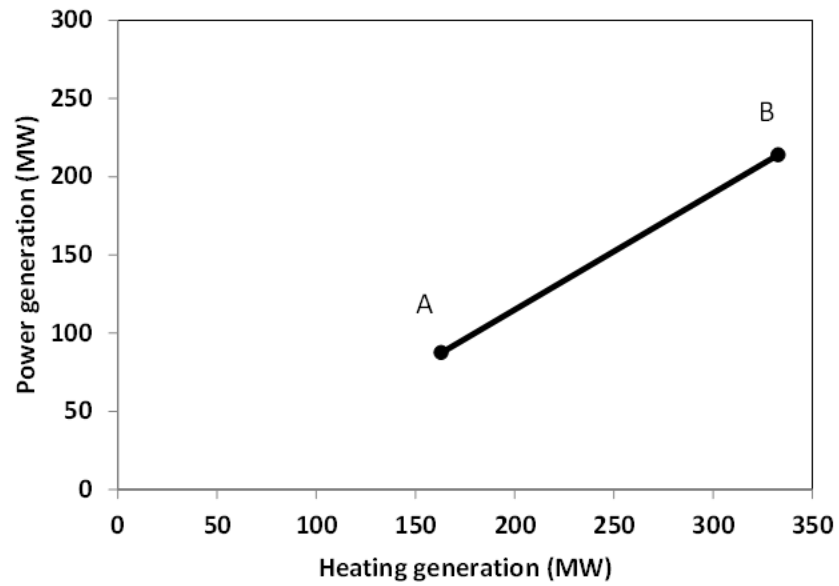
Steam plants with Backpressure turbine

This options includes steam-turbine based power plants with a backpressure turbine. The feasible operating region is between AB. The slope of the line is the heat to power ratio.

$$Power_{chp, i} = StorageInput_{chp, i} \cdot CHPPowerToHeat_{chp}$$

Steam plants with Extraction/condensing turbine

This options includes steam-turbine based power plants with an extraction/condensing turbine. The feasible operating region is within ABCDE. The vertical dotted line BC corresponds to the minimum condensation line (as defined by $CHPMaxHeat$). The slope of the DC line is the heat to power ratio and the slope of the AB line is the inverse of the power penalty ratio.



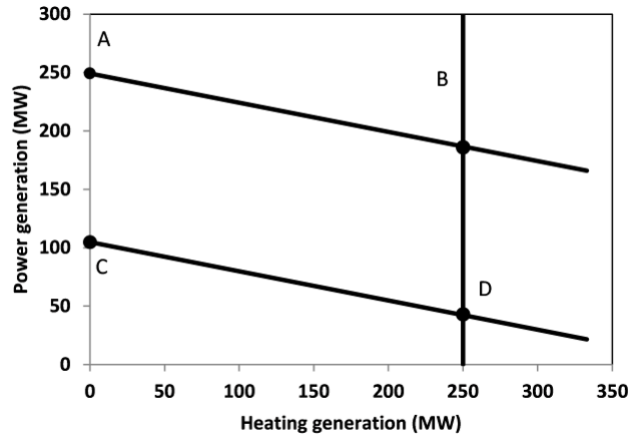
$$Power_{chp,i} \geq StorageInput_{chp,i} \cdot CHPPowerToHeat_{chp}$$

$$Power_{chp,i} \leq PowerCapacity_{chp} - StorageInput_{chp,i} \cdot CHPPowerLossFactor_{chp}$$

$$Power_{chp,i} \geq PowerMustRun_{chp,i} - StorageInput_{chp,i} * CHPPowerLossFactor_{chp}$$

Power plant coupled with any power to heat option

This option includes power plants coupled with resistance heater or heat pumps. The feasible operating region is between ABCD. The slope of the AB and CD line is the inverse of the COP or efficiency. The vertical dotted line corresponds to the heat pump (or resistance heater) thermal capacity (as defined by $CHPMaxHeat$)



$$Power_{chp,i} \leq PowerCapacity_{chp} - StorageInput_{chp,i} \cdot CHPPowerLossFactor_{chp}$$

$$Power_{chp,i} \geq PowerMustRun_{chp,i} - StorageInput_{chp,i} * CHPPowerLossFactor_{chp}$$

Heat Storage

Heat storage is modeled in a similar way as electric storage as follows:

Heat Storage balance:

$$+StorageLevel_{chp,i-1} + StorageInput_{chp,i} = StorageLevel_{chp,i} + Heat_{chp,i} + StorageSelfDischarge_{chp} \cdot StorageLevel_{chp,i}/24$$

Storage level must be above a minimum and below storage capacity:

$$StorageMinimum_{chp} \leq StorageLevel_{chp,i} \leq StorageCapacity_{chp} \cdot Nunits_{chp}$$

Emission limits

The operating schedule also needs to take into account any cap on the emissions (not only CO2) from the generation units existing in each node:

$$\sum_u (Power_{u,i} \cdot EmissionRate_{u,p} \cdot Location_{u,n}) \leq EmissionMaximum_{n,p}$$

It is important to note that the emission cap is applied to each optimisation horizon: if a rolling horizon of one day is adopted for the simulation, the cap will be applied to all days instead of the whole year.

Network-related constraints

The flow of power between nodes is limited by the capacities of the transmission lines:

$$\begin{aligned} FlowMinimum_{l,i} &\leq Flow_{l,i} \\ Flow_{l,i} &\leq FlowMaximum_{l,i} \end{aligned}$$

In this model a simple Net Transfer Capacity (NTC) between countries approach is followed. No DC power flow or Locational Marginal Pricing (LMP) model is implemented.

Curtailment

If curtailment of intermittent generation sources is allowed in one node, the amount of curtailed power is bounded by the output of the renewable (tr) units present in that node:

$$CurtailedPower_{n,i} \leq \sum_{u,tr} (Power_{u,i} \cdot Technology_{u,tr} \cdot Location_{u,n}) \cdot Curtailment_n$$

Load shedding

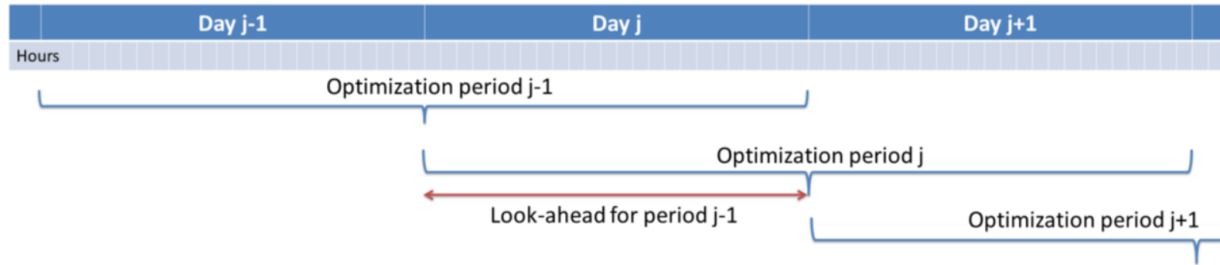
If load shedding is allowed in a node, the amount of shed load is limited by the shedding capacity contracted on that particular node (e.g. through interruptible industrial contracts)

$$ShedLoad_{n,i} \leq LoadShedding_n$$

5.4.3 Rolling Horizon

The mathematical problem described in the previous sections could in principle be solved for a whole year split into time steps of one hour, but with all likelihood the problem would become extremely demanding in computational terms when attempting to solve the model with a realistically sized dataset. Therefore, the problem is split into smaller optimization problems that are run recursively throughout the year.

The following figure shows an example of such approach, in which the optimization horizon is one day, with a look-ahead (or overlap) period of one day. The initial values of the optimization for day j are the final values of the optimization of the previous day. The look-ahead period is modelled to avoid issues related to the end of the optimization period such as emptying the hydro reservoirs, or starting low-cost but non-flexible power plants. In this case, the optimization is performed over 48 hours, but only the first 24 hours are conserved.



Although the previous example corresponds to an optimization horizon and an overlap of one day, these two values can be adjusted by the user in the Dispa-SET configuration file. As a rule of thumb, the optimization horizon plus the overlap period should be at least twice the maximum duration of the time-dependent constraints (e.g. the minimum up and down times). In terms of computational efficiency, small power systems can be simulated with longer optimization horizons, while larger systems should reduce this horizon, the minimum being one day.

5.4.4 References

5.5 Model Formulations and clustering

Because of the constraints linked to computational efficiency and to data availability, it is not necessarily desirable to accurately model each individual unit in the power system. For that reason, Dispa-SET can operate under different modelling hypotheses and levels of complexity. In terms of formulation of the optimization problem, these include for example: - A linear programming formulation, in which all units are clustered by technology - An integer formulation in which a typical unit is considered for each technology and multiplied N times. The formulation allows taking into account constraints such as minimum up/down times, minimum load, etc. - A binary formulation in which each power plant in the system is considered individually

The section describes the various clustering options and modeling formulations available in Dispa-SET. It is worthwhile to note that each clustering method and/or modelling formulation can be applied to the same reference dataset. This allows comparing the various methods in terms of computational efficiency, but also in terms of accuracy.

Four main formulations are currently available:

5.5.1 No clustering

In this case, the pre-processing tool does not modify the power plant input data. The user is allowed to cluster some power plants himself by defining the `Nunits` input variable in the power plants input csv file. Let us consider the following (incomplete) inputs as an example:

Unit	PowerCapacity	Nunits	Zone	Technology	Fuel
Maasvlakte	500	1	Z2	STUR	HRD
Diemen	430	1	Z2	COMC	GAS
CCGTs	400	6	Z2	COMC	GAS
Borssele	408	1	Z2	STUR	HRD
OCGT1	25	1	Z2	GTUR	GAS
TIHANGE 3	1000	1	Z1	STUR	NUC
DROGENBOS TGV	465	1	Z1	COMC	GAS
SISTERON	214	2	Z1	HDAM	WAT
SIERREUX	20	1	Z1	GTUR	GAS
ANGLEUR	30	1	Z1	GTUR	GAS
WindOn_Z1	200	1	Z1	WTON	WIN

In this example, there are a number of different unit types and two zones. Some power plants have $N_{units}=1$, which implies that they will be considered individually in the optimization. Other power plants (CCGTs and SISTERON) are multiplied 6 and 2 times, respectively. This implies that the total capacity of CCGTs and SISTERON units is 2400 MW and 228 MW, respectively. Note that all unit characteristics in the input data (not appearing in the above table) should be defined for a single unit!

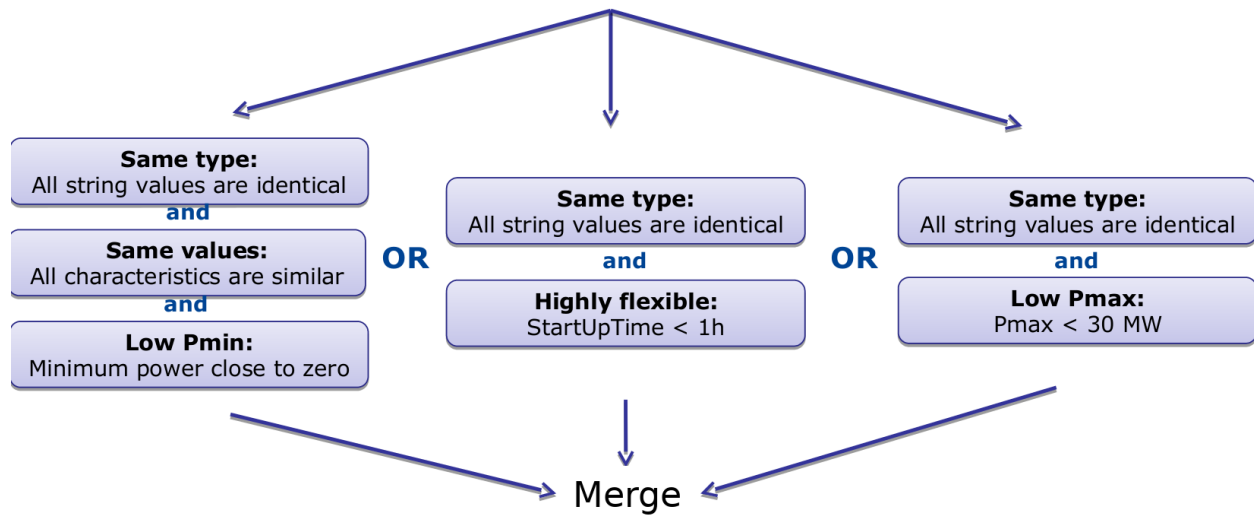
These two units are assigned an integer variable instead of a binary variable in the optimization. The solver successively starts 1, 2, 3, 4, etc. units with the exact same characteristics. In this approach, start-up costs, minimum up and down times, minimum part-load are considered, but with a significantly improved computational efficiency. The loss in accuracy resides in the hypothesis of identical characteristics for all units in a clustered group. This is however acceptable if not data is available at the individual power plant level, or if the complexity of the modeled system does not justify such a high disaggregation level.

5.5.2 Standard formulation

For computational efficiency reasons, it is useful to merge some of the original units into larger units. This reduces the number of continuous and binary variables and can, in some conditions, be performed without significant loss of simulation accuracy.

In the standard formulation (formerly call MILP formulation), the units that are either very small or very flexible are merged into larger units. Some of these units (e.g. the turbojets) indeed present a low capacity or a high flexibility: their output power does not exceed a few MW and/or they can reach full power in less than 15 minutes (i.e. less than the simulation time step). For these units, a unit commitment model with a time step of 1 hour is unnecessary and computationally inefficient. They are therefore merged into one single, highly flexible unit with averaged characteristics.

The condition for the merging of two units is a combination of subconditions regarding their type, maximum power, flexibility and technical similarities. They are summarized in the figure below (NB: the thresholds are for indicative purpose only, they can be user-defined).



When two units are merged, the minimum and maximum capacities of new aggregated units (indicated by the star) are given by:

$$P_{min}^* = \min(P_{j,min})$$

$$P_{max}^* = \sum_j (P_{j,min})$$

The last equation is also applied for the storage capacity or for the storage charging power.

The unit marginal (or variable cost) is given by:

$$Cost_{Variable}^* = \frac{\sum_j (P_{j,max} \cdot Cost_{Variable,j})}{P_{max}^*}$$

The start-up/shut-down costs are transformed into ramping costs (example with ramp-up):

$$Cost_{RampUp}^* = \frac{\sum_j (P_{j,max} \cdot Cost_{RampUp,j})}{P_{max}^*} + \frac{\sum_j (Cost_{StartUp,j})}{P_{max}^*}$$

Other characteristics, such as the plant efficiency, the minimum up/down times or the CO2 emissions are computed as a weighted averaged:

$$Efficiency^* = \frac{\sum_j (P_{j,max} \cdot Efficiency_j)}{P_{max}^*}$$

It should be noted that only very similar units are merged (i.e. their quantitative characteristics should be similar), which avoids errors due to excessive aggregation.

In the example provided in the above table the following would occur:

- SIERREUR and ANGLEUR would be merged because they are small and highly flexible.
- OCGT1 cannot be merged with SIERREUX and ANGLEUR since they don't belong to the same zone.
- Maasvlakte and Borssele are not merged, although they have the same technology, fuel and zone. This is because their size is significant and their flexibility is low.
- Diemen and CCGTS are merged only if their flexibility is high (i.e. they can start/stop or ramp to full load in less than one hour).

5.5.3 Integer clustering

In this formulation, all units of a similar technology, fuel and zone are clustered: a typical unit is defined by averaging the characteristics of all units belonging to the cluster. The total number of units is conserved, allowing a proper representation of constraints such as start-up costs, minimum up/down times or minimum stable load values. In the example provided above, the integer clustering would results into the following unit list:

Unit	PowerCapacity	Nunits	Zone	Technology	Fuel
Z2_STUR_HRD	454	2	Z2	STUR	HRD
Z2_COMC_GAS	404	7	Z2	COMC	GAS
OCGT1	25	1	Z2	GTUR	GAS
TIHANGE 3	1000	1	Z1	STUR	NUC
DROGENBOS TGV	465	1	Z1	COMC	GAS
SISTERON	214	2	Z1	HDAM	WAT
Z1_GTUR_GAZ	25	2	Z1	GTUR	GAS
WindOn_Z1	200	1	Z1	WTON	WIN

where the total capacity and number of units for each technology/fuel is conserved. More details regarding the formulation and the implementation of the integer clustering are available in¹

¹ Incorporating Operational Flexibility Into Electric Generation Planning: Impacts and Methods for System Design and Policy Analysis, Palmintier, B.S. (2012). Ph.D. Thesis, Engineering Systems Division, MIT

5.5.4 LP clustering

Dispa-SET provides the possibility to generate the optimisation model as an LP problem (i.e. without the binary variables). In that case, the following constraints are removed since they can only be expressed in an MILP formulation:

- Minimum up and down times
- Start-up costs
- Minimum stable load

Since the start-up of individual units is not considered anymore, it is not useful to disaggregate them in the optimisation. All units of a similar technology, fuel and zone are merged into a single unit using the equations proposed in the previous sections.

5.6 Implementation and interface

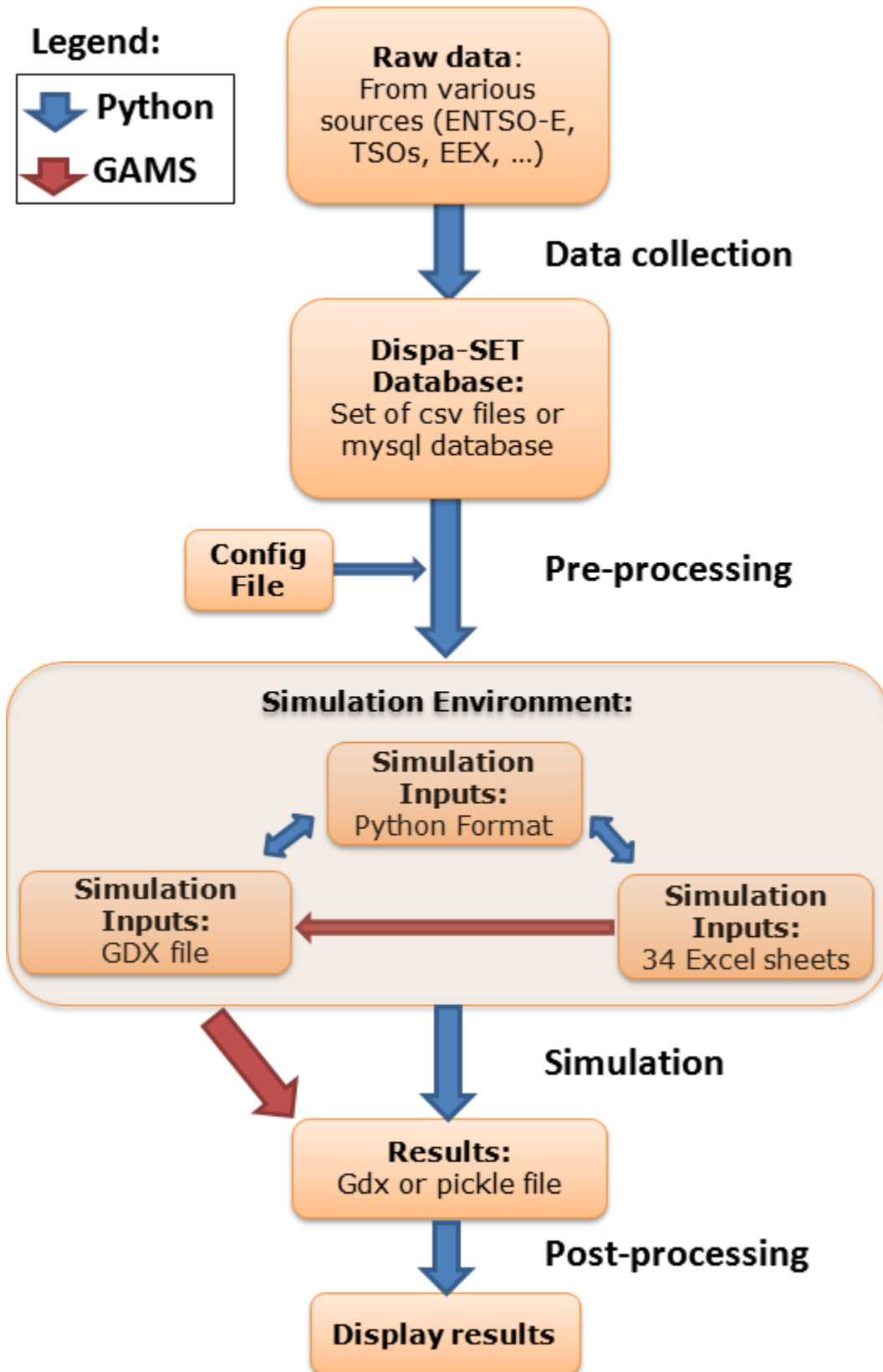
The typical step-by-step procedure to parametrize and run a DispaSET simulation is the following:

1. Fill the Dispa-SET database with properly formatted data (time series, power plant data, etc.)
2. Configure the simulation parameters (rolling horizon, data slicing) in the configuration file.
3. Generate the simulation environment which comprises the inputs of the optimisation
4. Open the GAMS simulation files (project: UCM.gpr and model: UCM_h.gms) and run the model.
5. Read and display the simulation results.

This section provides a detailed description of these steps and the corresponding data entities.

5.6.1 Resolution Flow Chart

The whole resolution process for a dispa-SET run is defined from the processing and formatting of the raw data to the generation of aggregated result plots and statistics. A flow chart of the consecutive data entities and processing steps is provided hereunder.



Each box in the flow chart corresponds to one data entity. The links between these data entities correspond to script written in Python or in GAMS. The different steps perform various tasks, which can be summarized by:

1. Data collection:

- Read csv sheets, assemble data
- Convert to the right format (timestep, units, etc).
- Define proper time index (duplicates not allowed)
- Connect to database
- Check if data present & write data
- Write metadata

2. Pre-processing:

- Read the config file
- Slice the data to the required time range
- Deal with missing data
- Check data for consistency (min up/down times, startup times, etc.)
- Calculate variable cost for each unit
- Cluster units
- Define scenario according to user inputs (curtailment, participation to reserve, amount of VRE, amount of storage, ...)
- Define initial state (basic merit-order dispatch)
- Write the simulation environment to a user-defined folder

3. Simulation environment and interoperability:

- **Self-consistent folder with all required files to run the simulation:**
 - Excel files
 - GDX file
 - Input files in pickle format
 - Gams model files
- Python scripts to translate the data between one format to the other.
- Possibility to modify the inputs manually and re-generate a GDX file from the excel files

4. Simulation:

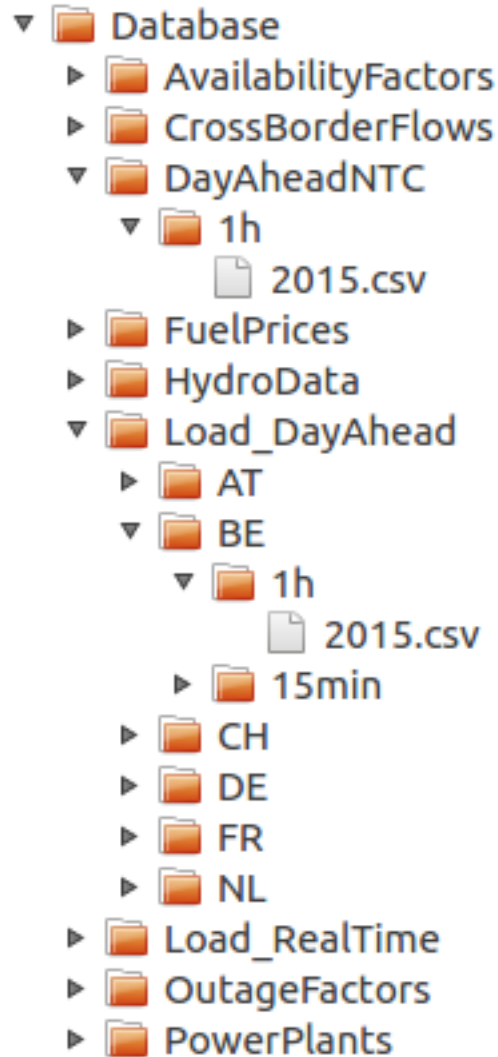
- The GAMS simulation file is run from the simulation environment folder
- Alternatively the model is run with the PYOMO solver
- All results and inputs are saved within the simulation environment

5. Post-processing:

- Reads the simulation results saved in the simulation environment
- Aggregates the power generation and storage curves
- Computes of yearly statistics
- Generates plots

5.6.2 Dispa-SET database

Although two versions of the database are available (mysql and csv), the public version of Dispa-SET only comes with the latter. The Dispa-SET input data is stored as csv file in directory structure. A link to the required data is then provided by the user in the configuration file.



The above figure shows a partially unfolded view of the database structure. In that example, data is provided for the day-ahead net transfer capacities for all lines in the EU, for the year 2015 and with a 1h time resolution. Time series are also provided for the day-ahead load forecast for Belgium in 2015 with 1h time resolution.

5.6.3 Configuration File

The excel config file is read at the beginning of the pre-processing phase. It provides general inputs for the simulation as well as links to the relevant data files in the database.

5.6.4 Simulation environment

This section describes the different simulation files, templates and scripts required to run the DispaSET model. For each simulation, these files are included into a single directory corresponding to a self-sufficient simulation environment.

A more comprehensive description of the files included in the simulation environment directory is provided hereunder.

UCM_h.gms and UCM.gpr

UCM_h.gms is the main GAMS model described in Chapter 1. A copy of this file is included in each simulation environment, allowing keeping track of the exact version of the model used for the simulation. The model must be run in GAMS and requires a proper input file (Inputs.gdx).

Requires:	Inputs.gdx	Input file for the simulation.
Generates:	Results.gdx	Simulation results in.gdx format
.	Results.xlsx	Simulation results in.xlsx format.

UCM.gpr is the GAMS project file which should be opened before UCM_h.gms.

make_gdx.gms

GAMS file that reads the different template excel files and generates the Inputs.gdx file. This file should be opened in GAMS.

Requires:	InputDispa-SET – xxx.xlsx	DispaSET template files
Generates:	Inputs.gdx	Input file for the simulation

makeGDX.bat

Batch script that generates the input file from the template without requiring opening GAMS. The first time it is executed, the path of the GAMS folder must be provided.

Requires:	InputDispa-SET – xxx.xlsx	DispaSET template files
.	make_gdx.gms	GAMS file to generate Inputs.gdx
Generates:	Inputs.gdx	Input file for the simulation

writeresults.gms

GAMS file to generate the excel Results.xlsx file from the Results.gdx generated by GAMS (in case the write_excel function was deactivated in GAMS).

Requires:	Results.gdx	Simulation results in.gdx format
Generates:	Results.xlsx	Simulation results in.xlsx format

Inputs.gdx

All the inputs of the model must be stored in the Inputs.gdx file since it is the only file read by the main GAMS model. This file is generated from the DispaSET template.

Requires:	InputDispa-SET – xxx.xlsx	DispaSET template files
Generates:		

InputDispa-SET - [ParameterName].xlsx

Series of 42 excel files, each corresponding to a parameter of the DispaSET model (see Chapter 1). The files must be formatted according to section 2.2.

InputDispa-SET - Sets.xlsx

Single excel file that contains all the sets used in the model in a column format.

InputDispa-SET - Config.xlsx

Single excel file that contains simulation metadata in the form of a Table. This metadata allows setting the rolling horizon parameter and slicing the input data to simulate a subset only.

Table 5.2: Config

FirstDay	2012	10	1	First day of the simulation in the template data
LastDay	2013	9	30	Last day of the simulation in the template data
RollingHorizon Length	0	0	3	Length of the rolling horizons
RollingHorizon LookAhead	0	0	1	Overlap period of the rolling horizon

Structure of the Excel template

The name of the input files are “Input Dispa-SET – [Parameter name].xlsx”. These files contain the data to be read by the model, after conversion into a GDX file.

The structure of all input files follows the following rules:

1. There is one file per model parameter
2. Each file contains only one sheet
3. The first row is left blank for non-time series data (i.e. data starts at A2)
4. **For time series data, the rows are organized as follows:**
 - (a) The first row is left blank
 - (b) Rows 2 to 5 contains the year, month, day and hour of each data
 - (c) Row 6 contains the time index of the data, which will be used in DispaSET
 - (d) The data therefore starts at A6
5. If one of the input sets of the data is u (the unit name), it is always defined as the first column of the data (column A)

6. If one of the input sets of the data is *h* (the time index), it is always defined as the only horizontal input in row 6. In the case of the file “Input Dispa-SET – Sets.xlsx”, all the required sets are written in columns with the set name in row 2.

5.6.5 Post-processing

Post-processing is implemented in the form of a series of functions to read the simulation inputs and results, to plot them, and to derive statistics.

The following values are computed:

- The total energy generated by each fuel, in each country.
- The total energy curtailed
- the total load shedding
- The overall country balance of the interconnection flows
- The total hours of congestion in each interconnection line
- The total amount of lost load, indicating (if not null) that the unit commitment problem was unfeasible at some hours
- The number of start-ups of power plants for each fuel

The following plots can be generated:

- A dispatch plot (by fuel type) for each country
- A commitment status (ON/OFF) plot for all the unit in a given country
- The level (or state of charge) of all the storage units in a given country
- The overall power generation by fuel type for all countries (bar plot)

An example usage of these functions is provided in the “Read_Results.ipynb” notebook.

5.7 Input Data

In this section, “Input Data” refers to the data stored in the Dispa-SET database. The format of this data is pre-defined and imposed, in such a way that it can be read by the pre-processing tool.

Two important preliminary comments should be formulated:

- All the time series should be registered with their timestamps (e.g. ‘2013-02-20 02:00:00’) relative to the UTC timezone.
- Although the optimisation model is designed to run with any technology or fuel name, the pre-processing and the post-processing tools of Dispa-SET use some hard-coded values. The Dispa-SET database should also comply with this convention (described in the next sections). Any non-recognized technology or fuel will be discarded in the pre-processing.

5.7.1 Technologies

The Dispa-SET input distinguishes between the technologies defined in the table below. The VRES column indicates the variable renewable technologies (set “tr” in the optimisation) and the Storage column indicates the technologies which can accumulate energy.

Table 5.3: Dispa-SET technologies

Technology	Description	VRES	Storage
COMC	Combined cycle	N	N
GTUR	Gas turbine	N	N
HDAM	Conventional hydro dam	N	Y
HROR	Hydro run-of-river	Y	N
HPHS	Pumped hydro storage	N	Y
ICEN	Internal combustion engine	N	N
PHOT	Solar photovoltaic	Y	N
STUR	Steam turbine	N	N
WTOF	Offshore wind turbine	Y	N
WTON	Onshore wind turbine	Y	N
CAES	Compressed air energy storage	N	Y
BATS	Stationary batteries	N	Y
BEVS	Battery-powered electric vehicles	N	Y
THMS	Thermal storage	N	Y
P2GS	Power-to-gas storage	N	Y

5.7.2 Fuels

Dispa-SET only considers a limited number of different fuel types. They are summarised in the following table, together with some examples.

Table 5.4: Dispa-SET fuels

Fuel	Examples
BIO	Bagasse, Biodiesel, Gas From Biomass, Gasification, Biomass, Briquettes, Cattle Residues, Rice Hulls Or Padi Husk, Straw, Wood Gas (From Wood Gasification), Wood Waste Liquids Excl Blk Liq (Incl Red Liquor, Sludge, Wood, Spent Sulfite Liquor And Oth Liquids, Wood And Wood Waste
GAS	Blast Furnace Gas, Boiler Natural Gas, Butane, Coal Bed Methane, Coke Oven Gas, Flare Gas, Gas (Generic), Methane, Mine Gas, Natural Gas, Propane, Refinery Gas, Sour Gas, Synthetic Natural Gas, Top Gas, Voc Gas & Vapor, Waste Gas, Wellhead Gas
GEO	Geothermal steam
HRD	Anthracite, Other Anthracite, Bituminous Coal, Coker By-Product, Coal Gas (From Coal Gasification), Coke, Coal (Generic), Coal-Oil Mixture, Other Coal, Coal And Pet Coke Mi, Coal Tar Oil, Anthracite Coal Waste, Coal-Water Mixture, Gob, Hard Coal / Anthracite, Imported Coal, Other Solids, Soft Coal, Anthracite Silt, Steam Coal, Subbituminous, Pelletized Synthetic Fuel From Coal, Bituminous Coal Waste)
HYD	Hydrogen
LIG	Lignite black, Lignite brown, lignite
NUC	U, Pu
OIL	Crude Oil, Distillate Oil, Diesel Fuel, No. 1 Fuel Oil, No. 2 Fuel Oil, No. 3 Fuel Oil, No. 4 Fuel Oil, No. 5 Fuel Oil, No. 6 Fuel Oil, Furnace Fuel, Gas Oil, Gasoline, Heavy Oil Mixture, Jet Fuel, Kerosene, Light Fuel Oil, Liquefied Propane Gas, Methanol, Naphtha, ,Gas From Fuel Oil Gasification, Fuel Oil, Other Liquid, Orimulsion, Petroleum Coke, Petroleum Coke Synthetic Gas, Black Liquor, Residual Oils, Re-Refined Motor Oil, Oil Shale, Tar, Topped Crude Oil, Waste Oil
PEA	Peat Moss
SUN	Solar energy
WAT	Hydro energy
WIN	Wind energy
WST	Digester Gas (Sewage Sludge Gas), Gas From Refuse Gasification, Hazardous Waste, Industrial Waste, Landfill Gas, Poultry Litter, Manure, Medical Waste, Refused Derived Fuel, Refuse, Waste Paper And Waste Plastic, Refinery Waste, Tires, Agricultural Waste, Waste Coal, Waste Water Sludge, Waste

Different fuels may be used to power a given technology, e.g. steam turbines may be fired with almost any fuel type. In Dispa-SET, each unit must be defined with the pair of values (technology,fuel). The next tables is derived from a commercial power plant database and indicates the number of occurrences of each combination. It appears clearly that, even through some combinations are irrelevant, both characteristics are needed to define a power plant type.

f/t	COMC	GTUR	HDAM	HPHS	HROR	ICEN	PHOT	STUR	WTOF	WTON	Total
BIO		2				10		79			91
GAS	485	188				28		97			798
GEO								10			10
HRD	4							389			393
HYD		1						1			2
LIG								249			249
NUC								138			138
OIL	7	94				27		146			274
PEA								17			17
SUN							20	7			27
UNK		2				1		1			4
WAT			33	23	21			1			78
WIN									9	27	36
WST		3				7		46			56
Total	496	290	33	23	21	73	20	1181	9	27	2173

5.7.3 Unit-specific or technology-specific inputs

Some parameters, such as the availability factor, the outage factor or the inflows may be defined at the unit level or at the technology level. For that reason, the pre-processing tool first lookups the unit name in the database to assign it a value, and then lookups the technology if no unit-specific information has been found.

5.7.4 Demand

Electricity demand is given per zone/country and the first row of each column should reflect that name.

Heat demand timeseries is needed where CHP plants are used. In the current formulation, each CHP plant is covering a heat load. In other words, one power plant is connected to a single district heating network. Therefore, in the heat demand input file, the first column has to be a time index and the following columns the heat demand in MW. The first row should contain the exact name of the power plant that will cover this demand.

5.7.5 Countries

Although the nodes names can be freely user-defined in the database, for the Dispa-SET EU model, the ISO 3166-1 standard has been adopted to describe each country at the NUTS1 level (except for Greece and the United Kingdom, for which the abbreviations EL and UK are used according to [EU Interinstitutional style guide](#)). The list of countries is defined as:

Code	Country
AT	Austria
BE	Belgium
BG	Bulgaria

Continued on next page

Table 5.5 – continued from previous page

Code	Country
CH	Switzerland
CY	Cyprus
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
EL	Greece
ES	Spain
FI	Finland
FR	France
HR	Croatia
HU	Hungary
IE	Ireland
IT	Italy
LT	Lithuania
LU	Luxembourg
LV	Latvia
MT	Malta
NL	Netherlands
NO	Norway
PL	Poland
PT	Portugal
RO	Romania
SE	Sweden
SI	Slovenia
SK	Slovakia
UK	United Kingdom

5.7.6 Power plant data

The power plant database may contain as many fields as desired, e.g. to ensure that the input data can be traced back, or to provide the id of this plant in another database. However, some fields are required by Dispa-SET and must therefore be defined in the database.

Common fields

The following fields must be defined for all units:

Table 5.6: Common fields for all units

Description	Field name	Units
Unit name	Unit	
Commissioning year	Year	
Technology	Technology	
Primary fuel	Fuel	
Zone	Zone	
Capacity	PowerCapacity	MW
Efficiency	Efficiency	%
Efficiency at minimum load	MinEfficiency	%
CO2 intensity	CO2Intensity	TCO2/MWh
Minimum load	PartLoadMin	%
Ramp up rate	RampUpRate	%/min
Ramp down rate	RampDownRate	%/min)
Start-up time	StartUPTime	h
Minimum up time	MinUpTime	h
Minimum down time	MinDownTime	h
No load cost	NoLoadCost	EUR/h
Start-up cost	StartUpCost	EUR
Ramping cost	RampingCost	EUR/MW
Presence of CHP	CHP	y/n

NB: the fields indicated with % as unit must be entered in a non-dimensional way (i.e. 90% should be written 0.9).

Storage units

Some parameters must only be defined for the units equipped with storage. They can be left blank for all other units.

Table 5.7: Specific fields for storage units

Description	Field name	Units
Storage capacity	STOCapacity	MWh
Self-discharge rate	STOSelfDischarge	%/h
Maximum charging power	STOMaxChargingPower	MW
Charging efficiency	STOChargingEfficiency	%

In the case of a storage unit, the discharge efficiency should be assigned to the common field “Efficiency”. Similarly, the common field “PowerCapacity” is the nominal power in discharge mode.

CHP units

Some parameters must only be defined for the units equipped with CHP. They can be left blank for all other units.

Table 5.8: Specific fields for CHP units

Description	Field name	Units
CHP Type	CHPType	extraction/back-pressure/p2h
Power-to-heat ratio	CHPPowerToHeat	•
Power Loss factor	CHPPowerLossFactor	•
Maximum heat production	CHPMaxHeat	MW(th)
Capacity of heat Storage	STOCapacity	MWh(th)
% of storage heat losses per timestep	STOSelfDischarge	%

In the current version of DispaSet three type of combined heat and power units are supported:

- Extraction/condensing units
- Backpressure units
- Power to heat

For each of the above configurations the following fields must be filled:

Table 5.9: Mandatory fields per type of CHP unit (X: mandatory, o:optional)

Description	Extraction	Backpressure	Power to heat
CHPType	X	X	X
CHPPowerToHeat	X	X	
CHPPowerLossFactor	X		X
CHPMaxHeat	o	o	X
STOCapacity	o	o	o
STOSelfDischarge	o	o	o

There are numerous data checking routines to ensure that all data provided is consistent.

Warning: For extraction/condensing CHP plants, the power plant capacity (*PowerCapacity*) must correspond to the nameplate capacity in the maximum heat and power mode. Internal DispaSet calculations will use the equivalent stand-alone plants capacity based on the parameters provided.

5.7.7 Renewable generation

Variable renewable generation is defined as power generation from renewable source that cannot be stored: it is either fed to the grid or curtailed. The technologies falling under this definition are the ones described in the subset “tr” in the model definition.

The time-dependent generation of for these technologies must be provided as an exogenous time series in the form of an “availability factor”. The latter is defined as the proportion of the nominal power capacity that can be generated at each hour.

In the database, the time series are provided as column vectors with the technology name as header. After the pre-processing, an availability factor is attributed to each unit according to their technology. Non-renewable technologies are assigned an availability factor of 1.

5.7.8 Storage and hydro data

Storage units are an extension of the regular units, including additional constraints and parameters. In the power plant table, four additional parameters are required: storage capacity (in MWh), self-discharge (in %/h), discharge power (in MW) and discharge efficiency (in %).

Some other parameters must be introduced in the form of time series in the “HydroData” section of the Dispa-SET database. There are described hereunder.

It should be noted that the nomenclature adopted for the modeling of storage units refers to the characteristics of hydro units with water reservoirs. However, these parameters (e.g. inflows, level) can easily be transposed to the case of alternative storage units such as batteries or CAES.

Inflows

The Inflows are defined as the contribution of exogenous sources to the level (or state of charge) or the reservoir. They are expressed in MWh of potential energy. If the inflows are provided as m^3/h , they must be converted.

The input to dispa-set is defined as “ScaledInflows”. It is the normalized values of the inflow with respect to the nominal power of the storage unit (in discharge mode). As an example, if the inflow value at a certain time is 100MWh/h and if the turbinning capacity of the hydro plant is 200 MW, the scaled inflow value must be defined as 0.5.

Scaled inflows should be provided in the form of time series with the unit name or the technology as columns header.

Storage level

Because emptying the storage has a zero marginal cost, a non-constrained optimization tends to leave the storage completely empty at the end of the optimisation horizon. For that reason, a minimum storage level is imposed at the last hour of each horizon. In Dispa-SET, a typical optimisation horizon is a few days. The model is therefore not capable of optimising the storage level e.g. for seasonal variations. The minimum storage level at the last hour is therefore an exogenous input. It can be selected from a historical level or obtained from a long-term hydro scheduling optimization.

The level input in the Dispa-SET database is normalized with respect to the storage capacity: its minimum value is zero and its maximum is one.

Variable capacity storage

In special cases, it might be necessary to simulate a storage unit whose capacity varies in time. A typical example is the simulation of the storage capacity provided by electric vehicles: depending on the time of the day, the connected battery capacity varies.

This special case can be simulated using the “AvailabilityFactor” input. In the case of a storage unit, reduces the available capacity by a factor varying from 0 to 1.

5.7.9 Power plant outages

In the current version, Dispa-SET does not distinguish planned outages from unplanned outages. They are characterized for each unit by the “OutageFactor” parameter. This parameter varies from 0 (no outage) to 1 (full outage). The available unit power is thus given by its nominal capacity multiplied by $(1 - \text{OutageFactor})$.

The outages are provided in the dedicated section of the Database for each unit. They consist of a time series with the unit name as columns header.

5.7.10 Interconnections

Two case should be distinguished when considering interconnections:

- Interconnections occurring between the simulated zones
- Interconnections occurring between the simulated zones and the Rest of the World (RoW)

These two cases are addresses by two different datasets described here under.

Net transfer capacities

Dispa-SET indogenously models the internal exchanges between countries (or zones) using a commercial net transfer caapcity (NTC). It does not consider (yet) DC power flows or more complex grid simulations.

Since the NTC values might vary in time, they must be supplied as time series, whose header include the origin country, the string ' -> ' and the destination country. As an example, the NTC from belgium to france must be provided with the header 'BE -> FR'.

Because NTCs are not necessarily symetrical, they must be provided in both directions (i.e. 'BE -> FR' and 'FR -> BE'. Non-provided NTCs are considered to be zero (i.e. no interconnection).

Historical physical flows

In Dispa-SET, the flows between internal zones and the rest of the world cannot be modeled endogenously. They must be provided as exogenous inputs. These inputs are referred to as "Historical physical flows", although they can also be user-defined.

In the input table of historical flows, the headers are similar to those of the NTCs (ie. 'XX -> YY'). All flows occurring an internal zone of the simulation and outside zones are considered as external flows and summed up. As an example, the historical flows 'FR -> XX', 'FR -> YY' and 'FR -> ZZ' will be aggregated in to a single interconnection flow 'FR -> RoW' if XX, YY and ZZ are not simulated zones.

These aggregated historical flows are then imposed to the solver as exogenous inputs.

In Dispa-SET, the flows are defined as positive variables. For each zone, there will thus be a maximum of two vectors defining its exchanges with the rest of the world (e.g. 'FR -> RoW' and 'RoW -> FR').

As for the NTCs, undefined historical flows are considered to be zero, i.e. not provided any historical flows is equivalent to consider the system as islanded.

5.7.11 Fuel Prices

Fuel prices vary both geographically and in time. They must therefore be provided as a time series for each simulated zone. One table is provided per fuel type, with as column header the zone to which it applies. If no header is provided, the fuel price is applied to all the simulated zones.

5.8 Case Studies

In the past years, Dispa-SET has been used in various scientific works covering different geographical areas and with different focus points. It is originally designed to simulate EU countries (one node per country) but has also been applied to other regions such as Western Africa, Bolivia or the Balkans. The model includes the constraints linked to each generation unit (min/max power, ramping rates, efficiencies, storage capacities, etc.), to the interconnections

and to the power and thermal demands. It uses high-resolution time series for the demands, renewable generation and outages in each simulated country.

The case studies for which scientific analysis has been carried out are summarized hereunder:

5.8.1 Dispa-SET for the EU28

Description

Dispa-SET is provided with a ready-to-use dataset of the EU28 (+Norway +Switzerland) power system. A detailed description of the model and of the selected input data is available in¹ and².

The power plants in the model are represented as a cluster of units powered by the same fuel type and technology. They can be modelled together with a large number of RES units with separate hourly distribution curves.

Features

The model is expressed as an optimization problem. Continuous variables include the individual unit dispatched power, the shedded load and the curtailed power generation. The binary variables are the commitment status of each unit. The main model features can be summarized as follows:

- Minimum and maximum power for each unit
- Power plant ramping limits
- Reserves up and down
- Minimum up/down times
- Load Shedding
- Curtailment
- Pumped-hydro storage
- Non-dispatchable units (e.g. wind turbines, run-of-river, etc.)
- Start-up, ramping and no-load costs
- Multi-nodes with capacity constraints on the lines (congestion)
- Constraints on the targets for renewables and/or CO2 emissions
- Yearly schedules for the outages (forced and planned) of each units
- CHP power plants and thermal storage
- Integer clustering: a representative unit is considered for each technology and multiplied N times.

The demand is assumed to be inelastic to the price signal. The MILP objective function is therefore the total generation cost over the optimization period.

¹ Kavvadias, K., Hidalgo Gonzalez, I., Zucker, A., & Quoilin, S. (2018). Integrated modelling of future EU power and heat systems - The Dispa-SET v2.2 open-source model (EUR 29085 EN). Luxembourg: European Commission.

² Matija Pavičević, Wouter Nijs, Konstantinos Kavvadias, Sylvain Quoilin, Modelling flexible power demand and supply in the EU power system: soft-linking between JRC-EU-TIMES and the open-source Dispa-SET model, ECOS 2019

Run the EU model

A specific config file is provided with the standard Dispa-SET installation (starting from v2.3). After installing Dispa-SET and checking that everything is fine, you can run the EU model in different ways:

From the command line (if Dispa-SET was properly installed):

```
dispaset -c ConfigFiles/ConfigEU.xlsx build simulate
```

Using the Dispa-SET Api and the provided example script:

```
scriptsbuild_and_run_EU_model.py
```

Documentation

The general documentation of the Dispa-SET model and the stable releases are available on the main Dispa-SET website: <http://www.dispaset.eu>

Licence

Dispa-SET is a free software licensed under the “European Union Public Licence” EUPL v1.2. It can be redistributed and/or modified under the terms of this license.

Important results

Main developers

- Sylvain Quoilin (University of Liège, KU Leuven)
- Konstantinos Kavvadias (European Commission, Institute for Energy and Transport)
- Matija Pavičević (KU Leuven)

References

More details regarding the model and its implementation are available in the following publications

5.8.2 Dispa-SET for the Balkans region

Description

This is input data of the Dispa-SET model, applied to the Balkans Region

Countries included in different scenarios are show in the table¹²³⁴ :

¹ Pavičević, M., Kavvadias, K. & Quoilin, S. (2018). Impact of model formulation on power system simulations - Example with the Dispa-SET Balkans model, EMP-E conference 2018: Modelling Clean Energy Pathways, Brussels.

² Pavičević, M., Quoilin, S. & Pukšec, T., (2018). Comparison of Different Power Plant Clustering Approaches for Modeling Future Power Systems, Proceedings of the 3rd SEE SDEWES Conference, Novi Sad.

³ Tomić, I., Pavičević, M., Quoilin, S., Zucker, A., Pukšec, T., Krajačić, G. & Duić, N., (2017). Applying the Dispa-SET model on the seven countries from the South East Europe. 8th Energy Planning and Modeling of Energy Systems-Meeting, Belgrade. <https://bib.irb.hr/prikazi-rad?rad=901595>

⁴ Pavičević, M., Tomić, I., Quoilin, S., Zucker, A., Pukšec, T., Krajačić, G. & Duić, N., (2017). Applying the Dispa-SET model on the Western Balkans power systems. Proceedings of the 2017 12th SDEWES Conference, Dubrovnik. <http://hdl.handle.net/2268/215095>

Countries	2010	2015	2020	2030
Albania	[O]	[O]	[O]	[O]
Bosnia and Herzegovina	[O]	[O]	[O]	[O]
Croatia	[X]	[O]	[O]	[O]
Kosovo	[O]	[O]	[O]	[O]
Macedonia	[O]	[O]	[O]	[O]
Montenegro	[O]	[O]	[O]	[O]
Serbia	[O]	[O]	[O]	[O]
Slovenia	[X]	[O]	[O]	[O]

The model has the ability to describe every single unit, or a cluster of units powered by the same fuel type and technology, with a high level of detail can be modelled together with a large number of RES units with separate hourly distribution curves.

Features

The model is expressed as an optimization problem. Continuous variables include the individual unit dispatched power, the shedded load and the curtailed power generation. The binary variables are the commitment status of each unit. The main model features can be summarized as follows:

- Minimum and maximum power for each unit
- Power plant ramping limits
- Reserves up and down
- Minimum up/down times
- Load Shedding
- Curtailment
- Pumped-hydro storage
- Non-dispatchable units (e.g. wind turbines, run-of-river, etc.)
- Start-up, ramping and no-load costs
- Multi-nodes with capacity constraints on the lines (congestion)
- Constraints on the targets for renewables and/or CO2 emissions
- Yearly schedules for the outages (forced and planned) of each units
- CHP power plants and thermal storage

The demand is assumed to be inelastic to the price signal. The MILP objective function is therefore the total generation cost over the optimization period.

Quick start

If you want to download the latest version from github for use or development purposes, make sure that you have git and the [anaconda distribution](<https://www.continuum.io/downloads>) installed and type the following:

```
git clone https://github.com/energy-modelling-toolkit/Dispa-SET.git
cd Dispa-SET
conda env create # Automatically creates environment based on environment.yml
source activate dispaset # in Windows: activate dispaset
pip install -e . # Install editable local version
```

The above commands create a dedicated environment so that your anconda configuration remains clean from the required dependencies installed.

To check that everything runs fine, you can build and run a test case by typing:

```
dispaset -c ConfigFiles/ConfigTest.xlsx build simulate
```

Make sure that the path is changed to local Dispa-SET folder in following scripts (the procedure is provided in the scripts):

```
build_and_run.py
read_results.py
```

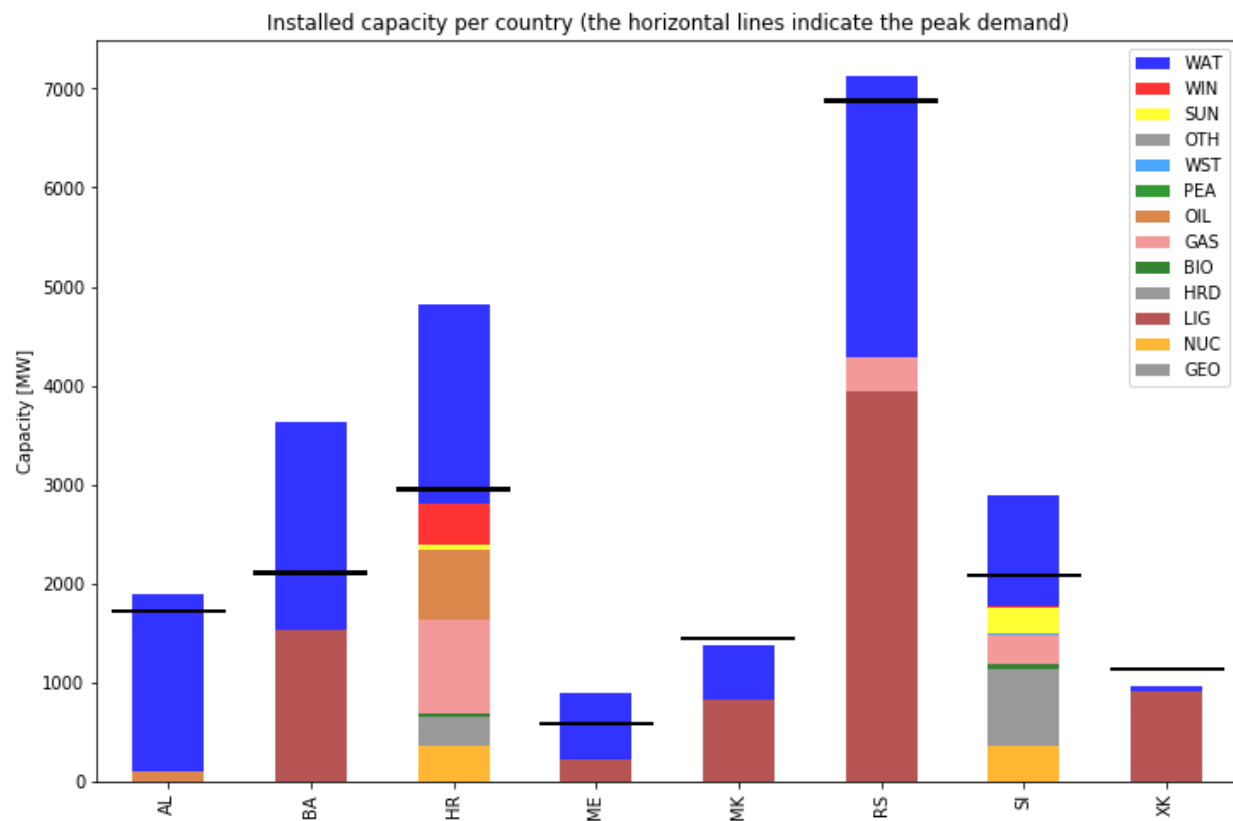
Documentation

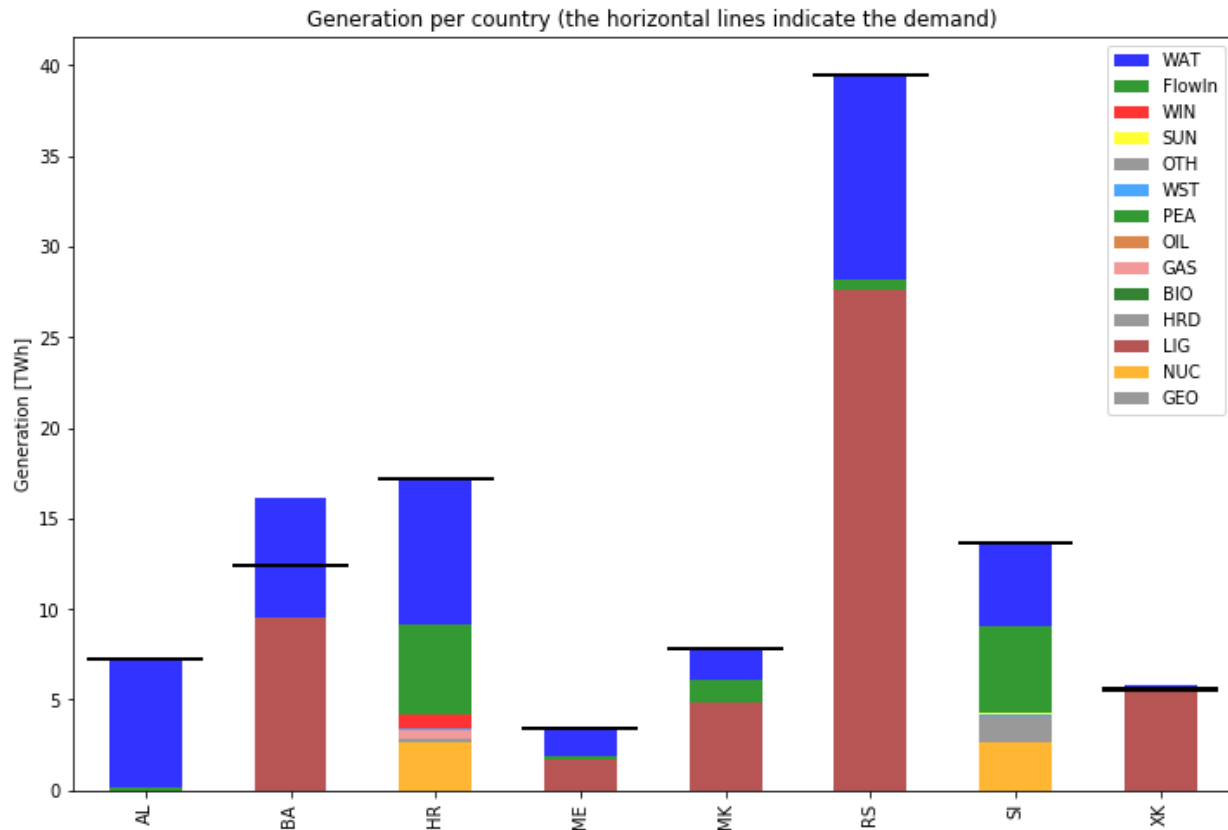
The general documentation of the Dispa-SET model and the stable releases are available on the main Dispa-SET website: <http://www.dispaset.eu>

Licence

Dispa-SET is a free software licensed under the “European Union Public Licence” EUPL v1.2. It can be redistributed and/or modified under the terms of this license.

Important results





Main developers

- Matija Pavičević (KU Leuven) - gathered and analysed the data, performed the computations, analysed and verified the results
- Sylvain Quoilin (University of Liège, KU Leuven) - designed the model and the computational framework, verified the results
- Andreas Zucker (Joint Research Centre, European Commission) - supervised the whole process

References

More details regarding the model and its implementation are available in the following publications

Other contributors

- Ivan Tomić (University of Zagreb) - gathered and analysed the initial data
- Tomislav Pukšec (University of Zagreb) - analysed the initial results
- Goran Krajačić (University of Zagreb) - supervised initial project
- Neven Duić (University of Zagreb) - supervised initial project

5.8.3 Dispa-SET for the Belarus

Description

Planning the future: Integrating renewable energy sources in the Belarusian power system

Background

The Belarusian energy sector is mainly running on fossil fuels. Approximately two third of the country's energy production is covered by natural gas, which is mainly imported from Russia. Therefore, increasing the share of renewables in the energy balance has become one of the priority areas of the economic policy of the Belarusian government. In this regard, the objective of this work is the development of smart power and heating systems that can handle increased shares of renewable energy in the Belarusian system. This research focuses on several aspects such as balancing issues, flexibility requirements and congestion management in Belarusian grid.

Methods and Features

The Belarusian energy system has been modeled in Dispa-SET, an open-source unit commitment and optimal dispatch model focused on the balancing and flexibility problems in European grids. A reference and several future scenarios with high share of renewable energy sources were created. This case study analysis provides insights on the ability to integrate as much renewables as possible into the system and to check its impact on the price of running the system.

The model is expressed as an optimization problem. Continuous variables include the individual unit dispatched power, the shedded load and the curtailed power generation. The binary variables are the commitment status of each unit. The main model features can be summarized as follows:

- Minimum and maximum power for each unit
- Power plant ramping limits
- Reserves up and down
- Minimum up/down times
- Load Shedding
- Curtailment
- Pumped-hydro storage
- Non-dispatchable units (e.g. wind turbines, run-of-river, etc.)
- Start-up, ramping and no-load costs
- Multi-nodes with capacity constraints on the lines (congestion)
- Constraints on the targets for renewables and/or CO2 emissions
- Yearly schedules for the outages (forced and planned) of each units
- CHP power plants and thermal storage

The demand is assumed to be inelastic to the price signal. The MILP objective function is therefore the total generation cost over the optimization period.

Quick start

If you want to download the latest version from github for use or development purposes, make sure that you have git and the [anaconda distribution](<https://www.continuum.io/downloads>) installed and type the following:

```
git clone https://github.com/energy-modelling-toolkit/Dispa-SET.git
cd Dispa-SET
conda env create # Automatically creates environment based on environment.yml
source activate dispaset # in Windows: activate dispaset
pip install -e . # Install editable local version
```

The above commands create a dedicated environment so that your anconda configuration remains clean from the required dependencies installed.

To check that everything runs fine, you can build and run a test case by typing:

```
dispaset -c ConfigFiles/ConfigTest.xlsx build simulate
```

Make sure that the path is changed to local Dispa-SET folder in following scripts (the procedure is provided in the scripts):

```
build_and_run.py
read_results.py
```

Documentation

The general documentation of the Dispa-SET model and the stable releases are available on the main Dispa-SET website: <http://www.dispaset.eu>

Licence

Dispa-SET is a free software licensed under the “European Union Public Licence” EUPL v1.2. It can be redistributed and/or modified under the terms of this license.

Results

In this model the potential for integration of renewables in the power system of Belarus was examined. The results of this study indicate that the current grid infrastructure can utilize up to 30% of the energy generated by renewables without causing any balancing and stability issues while applying heat pumps, thermal storage and bio-waste-based technologies.

Conclusions

The analysis performed in this work has demonstrated that the utilization of renewables could greatly reduce the use of fossil fuels and hence reduce the annual CO₂ emissions by about 5 million tons in the Belarusian energy sector. This way the high dependence on external energy markets should decrease. The designed scenarios should help to realize a Belarusian Energy and Environmental Policy where the share of renewables should reach 30%.

Highlights

Main developers

- Matija Pavičević (KU Leuven) - gathered and analysed the data, performed the computations, analysed and verified the results
- Darya Muslina (Belarusian National Technical University) - gathered and analysed the data

- Yuliya Stanetskaya (Belarusian National Technical University) - gathered and analysed the data

References

More details regarding the model and its implementation are available in the following publications

5.8.4 External links

- [EU28](#) DispaSET applied to the EU28 member states
- [Balkans](#) Western Balkans and neighbouring countries
- [Belarus](#) Planning the future: Integrating renewable energy sources in the Belarusian power system
- [Belgium](#) Coupling a power system model to a building model to evaluate the flexibility potential of DSM at country level
- [Bolivia](#) Techno-economic assessment of hing renewable energy source penetration in the Bolivian interconnected electric system
- [Central Europe](#) Evaluating flexibility and adequacy in future EU power systems: model coupling and long-term forecasting
- [Netherlands](#) Evaluating the impact of EV charging demand on the Dutch energy system

5.9 Developers' section

5.9.1 Folders organization

- GAMS code and scripts are included within the “GAMS-files” folder. The code should only be modified in that folder!
- Python code and scripts are included within the “dispaset” folder.
- DispaSET configuration files are included within the “ConfigFiles” folder (one file per simulation).
- Input files for each country are stored in the “Database” folder
- Simulation directories can be written into the “Simulations” folder, which is not tracked by git

A sample Simulation Environment Folder is available at `SimulationReferenceTestCase`. The files in this folders are generated automatically by the pre-processing scripts. They should not be modified. The folder should be replaced and updated at every major Dispa-SET release.

By default, the pre-processing scripts are set to generate the simulation environment within the “Simulations” folder. This folder is excluded in `.gitignore` and can therefore be used for testing purposes.

5.9.2 Math equations in the Docs

- To use online mathjax (default), there is nothing to do but displaying the equation requires an internet connection. In linux, the Makefile call to build with mathjax is written:

```
$(SPHINXBUILD) -b html $(ALLSPHINXOPTS) $(BUILDDIR)/html
```

```
In conf.py, the only math equation should be: 'sphinx.ext.mathjax'
```

- To use pngmath (for Linux):

```
sudo apt-get install dvipng

In conf.py, add 'sphinx.ext.pngmath' in the extensions

in Makefile: $(SPHINXBUILD) -D pngmath_latex=latex -b html $(ALLSPHINXOPTS)
↪ $(BUILDDIR)/html
```

- To use mathjax in offline mode, download the latest release from the mathjax github, copy it to Docs/_static/, and include it in the conf.py:

```
mathjax_path = "MathJax/MathJax.js"
```

5.9.3 Using Autodoc

The “API” section of the Docs uses the sphinx autodoc extension to scan the source code of Dispa-SET and display the relevant functions together with their description, parameters and outputs. In the Sphinx “conf.py”, the path to the source file must be added:

```
sys.path.insert(0, os.path.abspath('../dispaset'))
```

If the API documentation is generated with sphinx-apidoc, from the Docs folder, use:

```
sphinx-apidoc -o . ../dispaset/
```

Add a link to “DispaSET” in the table of content of index.rst and include the root folder in conf.py:

```
sys.path.insert(0, os.path.abspath('../'))
```

5.9.4 Clone git repository to svn

- Install git svn

```
sudo apt-get install git-svn
```

- Create svn branch

```
git branch svn
```

- Connect to svn repository

```
git svn init -s --prefix=svn/ --username <user> https://joinup.ec.europa.eu/svn/
↪ dispaset
```

- Checkout svn banch

```
git checkout svn
```

- Fetch remote content

```
git svn fetch
```

- Reset repository


```
git reset --hard remotes/svn/trunk
```

- Merge master into svn

```
git merge master
```

- Commit to the remote repository

```
git svn dcommit
```

The folder can finally optionally be deleted to avoid any confusion.

5.9.5 Issue with the compilation of the GAMS API

First, check that the installed version of GAMS is the 64 bit. 32 bit versions tend to generated compatibility issues.

When the pre-compiled libraries do not work, they must be re-compiled from the GAMS apifile folder. In Windows, this generally raises the issue of the missing vcvarsall.bat file. If the issue persists after installing the Microsoft C++ compiler for Python 2.7, try the following:

1. Enter MSVC for Python command prompt
2. SET DISTUTILS_USE_SDK=1
3. SET MSSdk=1
4. python.exe gdxsetup.py install

5.9.6 Public version of Dispa-SET

Because some input files are subject to intellectual property and copyrights, some folders available on the private repository cannot be uploaded to the public GitHub repository. The script DispaSync.sh in the root folder has been written to synchronize the subset of publicly available folders and files with an external folder. This folder can then be committed and pushed to the public repository.

By default, all file and folders are synchronized. In order to add a private path (to a file or to a folder), edit the DispaSync.sh file and add an entry to the “--exclude” argument.

The “rsync” software is required for the synchronization and must be installed on the local machine. The script can be run in any UNIX terminal (i.e. it cannot be run in Windows).

5.10 dispaset package

5.10.1 Subpackages

dispaset.preprocessing package

Submodules

dispaset.preprocessing.data_check module

This files gathers different functions used in the DispaSET to check the input data

```
__author__ = 'Sylvain Quoilin (sylvain.quoilin@ec.europa.eu)'
```

`dispaaset.preprocessing.data_check.check_AvailabilityFactors (plants, AF)`
Function that checks the validity of the provided availability factors and warns if a default value of 100% is used.

`dispaaset.preprocessing.data_check.check_MinMaxFlows (df_min, df_max)`
Function that checks that there is no incompatibility between the minimum and maximum flows

`dispaaset.preprocessing.data_check.check_chp (config, plants)`
Function that checks the CHP plant characteristics

`dispaaset.preprocessing.data_check.check_clustering (plants, plants_merged)`
Function that checks that the installed capacities are still equal after the clustering process

Parameters

- **plants** – Non-clustered list of units
- **plants_merged** – clustered list of units

`dispaaset.preprocessing.data_check.check_df (df, StartDate=None, StopDate=None, name="")`
Function that check the time series provided as inputs

`dispaaset.preprocessing.data_check.check_heat_demand (plants, data)`
Function that checks the validity of the heat demand profiles

Parameters plants – List of CHP plants

`dispaaset.preprocessing.data_check.check_simulation_environment (SimulationPath, store_type='pickle', firstline=7)`
Function to test the validity of dispaaset inputs :param SimulationPath: Path to the simulation folder :param store_type: choose between: “list”, “excel”, “pickle” :param firstline: Number of the first line in the data (only if type==’excel’)

`dispaaset.preprocessing.data_check.check_sto (config, plants, raw_data=True)`
Function that checks the storage plant characteristics

`dispaaset.preprocessing.data_check.check_units (config, plants)`
Function that checks the power plant characteristics

`dispaaset.preprocessing.data_check.isStorage (tech)`
Function that returns true the technology is a storage technology

`dispaaset.preprocessing.data_check.isVRE (tech)`
Function that returns true the technology is a variable renewable energy technology

dispaaset.preprocessing.data_handler module

`dispaaset.preprocessing.data_handler.NodeBasedTable (path, idx, countries, tablename="", default=None)`
This function loads the tabular data stored in csv files relative to each zone (a.k.a node, country) of the simulation.

Parameters

- **path** – Path to the data to be loaded
- **idx** – Pandas datetime index to be used for the output
- **countries** – List with the country codes to be considered
- **fallback** – List with the order of data source.

- **tablename** – String with the name of the table being processed
- **default** – Default value to be applied if no data is found

Returns Dataframe with the time series for each unit

```
dispaset.preprocessing.data_handler.UnitBasedTable(plants, path, idx, countries, fallbacks=['Unit'], tablename="", default=None, RestrictWarning=None)
```

This function loads the tabular data stored in csv files and assigns the proper values to each unit of the plants dataframe. If the unit-specific value is not found in the data, the script can fallback on more generic data (e.g. fuel-based, technology-based, zone-based) or to the default value. The order in which the data should be loaded is specified in the fallback list. For example, ['Unit','Technology'] means that the script will first try to find a perfect match for the unit name in the data table. If not found, a column with the unit technology as header is search. If not found, the default value is assigned.

Parameters

- **plants** – Dataframe with the units for which data is required
- **path** – Path to the data to be loaded
- **idx** – Pandas datetime index to be used for the output
- **countries** – List with the country codes to be considered
- **fallback** – List with the order of data source.
- **tablename** – String with the name of the table being processed
- **default** – Default value to be applied if no data is found
- **RestrictWarning** – Only display the warnings if the unit belongs to the list of technologies provided in this parameter

Returns Dataframe with the time series for each unit

```
dispaset.preprocessing.data_handler.define_parameter(sets_in, sets, value=0)
```

Function to define a DispaSET parameter and fill it with a constant value

Parameters

- **sets_in** – List with the labels of the sets corresponding to the parameter
- **sets** – dictionary containing the definition of all the sets (must comprise those referenced in sets_in)
- **value** – Default value to attribute to the parameter

```
dispaset.preprocessing.data_handler.export_yaml_config(ExcelFile, YAMLFile)
```

Function that loads the DispaSET excel config file and dumps it as a yaml file.

Parameters

- **ExcelFile** – Path to the Excel config file
- **YAMLFile** – Path to the YAML config file to be written

```
dispaset.preprocessing.data_handler.invert_dic_df(dic, tablename="")
```

Function that takes as input a dictionary of dataframes, and inverts the key of the dictionary with the columns headers of the dataframes

Parameters

- **dic** – dictionary of dataframes, with the same columns headers and the same index

- **tablename** – string with the name of the table being processed (for the error msg)

Returns dictionary of dataframes, with swapped headers

`dispaaset.preprocessing.data_handler.load_config_excel (ConfigFile, AbsPath=True)`

Function that loads the DispaSET excel config file and returns a dictionary with the values

Parameters

- **ConfigFile** – String with (relative) path to the DispaSET excel configuration file
- **AbsPath** – If true, relative paths are automatically changed into absolute paths (recommended)

`dispaaset.preprocessing.data_handler.load_config_yaml (filename, AbsPath=True)`

Loads YAML file to dictionary

`dispaaset.preprocessing.data_handler.load_csv (filename, TempPath='.pickle', header=0, skiprows=None, skipfooter=0, index_col=None, parse_dates=False)`

Function that loads acsv sheet into a dataframe and saves a temporary pickle version of it. If the pickle is newer than the sheet, do no load the sheet again.

Parameters

- **filename** – path to csv file
- **TempPath** – path to store the temporary data files

`dispaaset.preprocessing.data_handler.merge_series (plants, data, mapping, method='WeightedAverage', tablename="")`

Function that merges the times series corresponding to the merged units (e.g. outages, inflows, etc.)

Parameters

- **plants** – Pandas dataframe with the information relative to the original units
- **data** – Pandas dataframe with the time series and the original unit names as column header
- **mapping** – Mapping between the merged units and the original units. Output of the clustering function
- **method** – Select the merging method ('WeightedAverage'/'Sum')
- **tablename** – Name of the table being processed (e.g. 'Outages'), used in the warnings

Return merged Pandas dataframe with the merged time series when necessary

`dispaaset.preprocessing.data_handler.write_to_excel (xls_out, list_vars)`

Function that reads all the variables (in list_vars) and inserts them one by one to excel

Parameters

- **xls_out** – The path of the folder where the excel files are to be written
- **list_vars** – List containing the dispaaset variables

Returns Binary variable (True)

dispaaset.preprocessing.preprocessing module

This is the main file of the DispaSET pre-processing tool. It comprises a single function that generated the DispaSET simulation environment.

@author: S. Quoilin

```
dispaset.preprocessing.preprocessing.adjust_capacity(inputs, tech_fuel, scaling=1, value=None, singleunit=False, write_gdx=False, dest_path="")
```

Function used to modify the installed capacities in the Dispa-SET generated input data. The function updates the Inputs.p file in the simulation directory at each call.

Parameters

- **inputs** – Input data dictionary OR path to the simulation directory containing Inputs.p
- **tech_fuel** – tuple with the technology and fuel type for which the capacity should be modified
- **scaling** – Scaling factor to be applied to the installed capacity
- **value** – Absolute value of the desired capacity (! Applied only if scaling != 1 !)
- **singleunit** – Set to true if the technology should remain lumped in a single unit
- **write_gdx** – boolean defining if Inputs.gdx should be also overwritten with the new data
- **dest_path** – Simulation environment path to write the new input data. If unspecified, no data is written!

Returns New SimData dictionary

```
dispaset.preprocessing.preprocessing.adjust_storage(inputs, tech_fuel, scaling=1, value=None, write_gdx=False, dest_path="")
```

Function used to modify the storage capacities in the Dispa-SET generated input data. The function updates the Inputs.p file in the simulation directory at each call.

Parameters

- **inputs** – Input data dictionary OR path to the simulation directory containing Inputs.p
- **tech_fuel** – tuple with the technology and fuel type for which the capacity should be modified
- **scaling** – Scaling factor to be applied to the installed capacity
- **value** – Absolute value of the desired capacity (! Applied only if scaling != 1 !)
- **write_gdx** – boolean defining if Inputs.gdx should be also overwritten with the new data
- **dest_path** – Simulation environment path to write the new input data. If unspecified, no data is written!

Returns New SimData dictionary

```
dispaset.preprocessing.preprocessing.build_simulation(config)
```

This function reads the DispaSET config, loads the specified data, processes it when needed, and formats it in the proper DispaSET format. The output of the function is a directory with all inputs and simulation files required to run a DispaSET simulation.

Parameters

- **config** – Dictionary with all the configuration fields loaded from the excel file. Output of the 'LoadConfig' function.
- **plot_load** – Boolean used to display a plot of the demand curves in the different zones

```
dispaaset.preprocessing.preprocessing.get_git_revision_tag()
```

Get version of DispaSET used for this run. tag + commit hash

dispaaset.preprocessing.utils module

This file gathers different functions used in the DispaSET pre-processing tools

@author: Sylvain Quoilin (sylvain.quoilin@ec.europa.eu)

```
dispaaset.preprocessing.utils.clustering(plants, method='Standard', Nslices=20, PartLoad-  
Max=0.1, Pmax=30)
```

Merge excessively disaggregated power Units.

Parameters

- **plants** – Pandas dataframe with each power plant and their characteristics (following the DispaSET format)
- **method** – Select clustering method ('Standard'/'LP'/None)
- **Nslices** – Number of slices used to fingerprint each power plant characteristics. slices in the power plant data to categorize them (fewer slices involves that the plants will be aggregated more easily)
- **PartLoadMax** – Maximum part-load capability for the unit to be clustered
- **Pmax** – Maximum power for the unit to be clustered

Returns A list with the merged plants and the mapping between the original and merged units

```
dispaaset.preprocessing.utils.incidence_matrix(sets, set_used, parameters, param_used)
```

This function generates the incidence matrix of the lines within the nodes A particular case is considered for the node "Rest Of the World", which is not explicitly defined in DispaSET

```
dispaaset.preprocessing.utils.interconnections(Simulation_list, NTC_inter, Histori-  
cal_flows)
```

Function that checks for the possible interconnections of the countries included in the simulation. If the interconnections occurs between two of the countries defined by the user to perform the simulation with, it extracts the NTC between those two countries. If the interconnection occurs between one of the countries selected by the user and one country outside the simulation, it extracts the physical flows; it does so for each pair (country inside-country outside) and sums them together creating the interconnection of this country with the RoW.

Parameters

- **Simulation_list** – List of simulated countries
- **NTC** – Day-ahead net transfer capacities (pd dataframe)
- **Historical_flows** – Historical flows (pd dataframe)

Module contents

dispaaset.postprocessing package

Submodules

dispaaset.postprocessing.postprocessing module

Set of functions useful to analyse to DispaSET output data.

@author: Sylvain Quoilin, JRC

`dispaSet.postprocessing.postprocessing.CostExPost (inputs, results)`

Ex post computation of the operational costs with plotting. This allows breaking down the cost into its different components and check that it matches with the objective function from the optimization.

The cost objective function is the following:

$$\begin{aligned} \text{SystemCost}(i) = & \text{E} = \text{sum}(u, \text{CostFixed}(u) * \text{Committed}(u, i)) \\ & + \text{sum}(u, \text{CostStartUpH}(u, i) + \text{CostShutDownH}(u, i)) + \text{sum}(u, \text{CostRampUpH}(u, i) + \text{CostRampDownH}(u, i)) \\ & + \text{sum}(u, \text{CostVariable}(u, i) * \text{Power}(u, i)) + \text{sum}(l, \text{PriceTransmission}(l, i) * \text{Flow}(l, i)) \\ & + \text{sum}(n, \text{CostLoadShedding}(n, i) * \text{ShedLoad}(n, i)) + \text{sum}(chp, \text{CostHeatSlack}(chp, i) * \text{HeatSlack}(chp, i)) \\ & + \text{sum}(chp, \text{CostVariable}(chp, i) * \text{CHPPowerLossFactor}(chp) * \text{Heat}(chp, i)) \\ & + \text{Config}(\text{"ValueOfLostLoad"}, \text{"val"}) * (\text{sum}(n, \text{LL_MaxPower}(n, i) + \text{LL_MinPower}(n, i))) + 0.8 * \text{Config}(\text{"ValueOfLostLoad"}, \text{"val"}) * (\text{sum}(n, \text{LL_2U}(n, i) + \text{LL_2D}(n, i) + \text{LL_3U}(n, i))) \\ & + 0.7 * \text{Config}(\text{"ValueOfLostLoad"}, \text{"val"}) * \text{sum}(u, \text{LL_RampUp}(u, i) + \text{LL_RampDown}(u, i)) + \text{Config}(\text{"CostOfSpillage"}, \text{"val"}) * \text{sum}(s, \text{spillage}(s, i)); \end{aligned}$$

Returns tuple with the cost components and their cumulative sums in two dataframes.

`dispaSet.postprocessing.postprocessing.GAMSstatus (statustype, num)`

Function that returns the model status or the solve status from gams

Parameters

- **statustype** – String with the type of status to retrieve (“solver” or “model”)
- **num** – Indicated termination condition (Integer)

Returns String with the status

`dispaSet.postprocessing.postprocessing.aggregate_by_fuel (PowerOutput, Inputs, SpecifyFuels=None)`

This function sorts the power generation curves of the different units by technology

Parameters

- **PowerOutput** – Dataframe of power generation with units as columns and time as index
- **Inputs** – DispaSet inputs version 2.1.1
- **SpecifyFuels** – If not all fuels should be considered, list containing the relevant ones

Returns **PowerByFuel** Dataframe with power generation by fuel

`dispaSet.postprocessing.postprocessing.ds_to_df (inputs)`

Function that converts the dispaSet data format into a dictionary of dataframes

Parameters **inputs** – input file

Returns dictionary of dataframes

`dispaSet.postprocessing.postprocessing.filter_by_country (PowerOutput, inputs, c)`

This function filters the dispaSet Output Power dataframe by country

Parameters

- **PowerOutput** – Dataframe of power generation with units as columns and time as index
- **Inputs** – DispaSet inputs version 2.1.1
- **c** – Selected country (e.g. ‘BE’)

Returns **Power** Dataframe with power generation by country

`dispaSet.postprocessing.postprocessing.get_imports (flows, c)`

Function that computes the balance of the imports/exports of a given zone

Parameters

- **flows** – Pandas dataframe with the timeseries of the exchanges
- **c** – Country (zone) to consider

Returns **NetImports** Scalar with the net balance over the whole time period

`dispaset.postprocessing.postprocessing.get_indicators_powerplant` (*inputs*, *re-*
sults)

Function that analyses the dispa-set results at the power plant level Computes the number of startups, the capacity factor, etc

Parameters

- **inputs** – DispaSET inputs
- **results** – DispaSET results

Returns out Dataframe with the main power plants characteristics and the computed indicators

`dispaset.postprocessing.postprocessing.get_load_data` (*inputs*, *c*)

Get the load curve, the residual load curve, and the net residual load curve of a specific country

Parameters

- **inputs** – DispaSET inputs (output of the `get_sim_results` function)
- **c** – Country to consider (e.g. 'BE')

Return out Dataframe with the following columns: Load: Load curve of the specified country
ResidualLoad: Load minus the production of variable renewable sources NetResidualLoad:
Residual netted from the interconnections with neighbouring countries

`dispaset.postprocessing.postprocessing.get_plot_data` (*inputs*, *results*, *c*)

Function that reads the results dataframe of a DispaSET simulation and extract the dispatch data specific to one country

Parameters

- **results** – Pandas dataframe with the results (output of the `GdxToDataframe` function)
- **c** – Country to be considered (e.g. 'BE')

Returns plotdata Dataframe with the dispatch data storage and outflows are negative

`dispaset.postprocessing.postprocessing.get_result_analysis` (*inputs*, *results*)

Reads the DispaSET results and provides useful general information to stdout

Parameters

- **inputs** – DispaSET inputs
- **results** – DispaSET results

`dispaset.postprocessing.postprocessing.get_sim_results` (*path*='.', *gams_dir*=None,
cache=False,
temp_path='.pickle')

This function reads the simulation environment folder once it has been solved and loads the input variables together with the results.

Parameters

- **path** – Relative path to the simulation environment folder (current path by default)
- **cache** – If true, caches the simulation results in a pickle file for faster loading the next time
- **temp_path** – Temporary path to store the cache file

Returns inputs,results Two dictionaries with all the input and outputs

`dispaaset.postprocessing.postprocessing.get_units_operation_cost` (*inputs, re-*
sults)

Function that computes the operation cost for each power unit at each instant of time from the DispaSET results
Operation cost includes: CostFixed + CostStartUp + CostShutDown + CostRampUp + CostRampDown + CostVariable

Parameters

- **inputs** – DispaSET inputs
- **results** – DispaSET results

Returns out Dataframe with the the power units in columns and the operatopn cost at each instant in rows

Main Author: @AbdullahAlawad

`dispaaset.postprocessing.postprocessing.plot_country` (*inputs, results, c="", rng=None,*
rug_plot=True)

Generates plots from the dispa-SET results for one specific country

Parameters

- **inputs** – DispaSET inputs
- **results** – DispaSET results
- **c** – Considered country (e.g. 'BE')

`dispaaset.postprocessing.postprocessing.plot_country_capacities` (*inputs,*
plot=True)

Plots the installed capacity for each country, disaggregated by fuel type

Parameters **inputs** – Dictionnary with the inputs of the model (output of the function GetResults)

`dispaaset.postprocessing.postprocessing.plot_dispatch` (*demand, plotdata, level=None,*
curtailment=None, rng=None,
alpha=None, figsize=(13, 6))

Function that plots the dispatch data and the reservoir level as a cumulative sum

Parameters

- **demand** – Pandas Series with the demand curve
- **plotdata** – Pandas Dataframe with the data to be plotted. Negative columns should be at the beginning. Output of the function GetPlotData
- **level** – Optional pandas series with an aggregated reservoir level for the considered zone.
- **curtailment** – Optional pandas series with the value of the curtailment
- **rng** – Indexes of the values to be plotted. If undefined, the first week is plotted

`dispaaset.postprocessing.postprocessing.plot_energy_country_fuel` (*inputs, results,*
PPindicators)

Plots the generation for each country, disaggregated by fuel type

Parameters

- **results** – Dictionnary with the outputs of the model (output of the function GetResults)
- **PPindicators** – Por powerplant statistics (output of the function get_indicators_powerplant)

```
dispaset.postprocessing.postprocessing.plot_rug(df_series, on_off=False,
                                              cmap='Greys', fig_title="", nor-
                                              malized=False)
```

Create multiaxis rug plot from pandas Dataframe

Arguments: `df_series` (pd.DataFrame): 2D pandas with timed index `on_off` (bool): if True all points that are above 0 will be plotted as one color. If False all values will be colored based on their value. `cmap` (str): palette name (from colorbrewer, matplotlib etc.) `fig_title` (str): Figure title normalized (bool): if True, all series colormaps will be normalized based on the maximum value of the dataframe

Returns: plot

Function copied from `enlpy` v0.1 www.github.com/kavvkon/enlpy. Install with *pip install enlpy* for latest version.

```
dispaset.postprocessing.postprocessing.storage_levels(inputs, results)
```

Reads the DispaSET results and provides the difference between the minimum storage profile and the computed storage profile

Parameters

- **inputs** – DispaSET inputs
- **results** – DispaSET results

Module contents

dispaset.pyomo package

Submodules

dispaset.pyomo.model module

This worksheet contains the two main functions to solve the DispaSET optimization problem using PYOMO.

Functions: - `DispaSolve`: Implements the rolling horizon - `DispOptim`: Performs the optimization for each horizon

@author: 'Sylvain Quoilin'

```
dispaset.pyomo.model.DispOptim(sets, parameters, LPFormulation=False)
```

This is the main optimization function of DispaSet. Two operation are performed: 1. Translation of the dispaset data format into the pyomo format 2. Definition of the Pyomo optimization model as a ConcreteModel

Parameters

- **sets** – Dictionary containing the sets (defined as a list of strings or integers)
- **parameters** – Dictionary containing the parameters of the optimization problem (in the DispaSET 2.1.1 format)

Returns The Pyomo optimization model instance

```
dispaset.pyomo.model.DispaSolve(sets, parameters, LPFormulation=False, path_cplex="")
```

The `DispaSolve` function defines the rolling horizon optimization and saves each result variable in a pandas dataframe The definition of the rolling horizon must be included into the DispaSET Config parameter'

Parameters

- **sets** – Dictionary containing the sets (defined as a list of strings or integers)

- **parameters** – Dictionary containing the parameters of the optimization problem (in the DispaSET 2.1.1 format)

Returns Dictionary of pandas dataframes with the optimization variables

`dispaset.pyomo.model.after_solver(results)`

`dispaset.pyomo.model.run_solver(instance, solver='cplex', solver_manager='serial', options_string="", path_cplex=")`

`dispaset.pyomo.model.try_solvers(preferred_solver, path_cplex=")`

Initialize solver engine, trying different solvers. Faster solvers are tried first

dispaset.pyomo.utils module

`dispaset.pyomo.utils.get_set_members(instance, sets)`

Get set members relative to a list of sets

Parameters

- **instance** – Pyomo Instance
- **sets** – List of strings with the set names

Returns A list with the set members

`dispaset.pyomo.utils.get_sets(instance, varname)`

Get sets that belong to a pyomo Variable or Param

Parameters

- **instance** – Pyomo Instance
- **varname** – Name of the Pyomo Variable (string)

Returns A list with the sets that belong to this Param

`dispaset.pyomo.utils.pyomo_format(sets, param)`

Function that flattens the multidimensional dispaset input data into the pyomo format: a dictionary with a tuple and the parameter value. The tuple contains the strings of the corresponding set values

`dispaset.pyomo.utils.pyomo_to_pandas(instance, varname)`

Function converting a pyomo variable or parameter into a pandas dataframe. The variable must have one or two dimensions and the sets must be provided as a list of lists

Parameters

- **instance** – Pyomo Instance
- **varname** – Name of the Pyomo Variable (string)

Module contents

dispaset.misc package

Submodules

dispaset.misc.colorstreamhandler module

`dispaset.misc.colorstreamhandler.ColorStreamHandler`
alias of `_AnsiColorStreamHandler`

dispaset.misc.gdx_handler module

Collection of functions to write Dispa-SET input data to a.gdx file and/or to a simulation directory with one excel file per parameter.

Example: read.gdx file:

```
data = GdxToList(gams_dir, 'Results.gdx', varname='all', verbose=True)
```

write it to a dictionary of dataframes:

```
dataframes = GdxToDataframe(data, fixindex=True, verbose=True)
```

@author: Sylvain Quoilin (sylvain.quoilin@ec.europa.eu)

`dispaset.misc.gdx_handler.gdx_to_dataframe(data, fixindex=False, verbose=False)`

This function structures the raw data extracted from a.gdx file (using the function `GdxToList`) and outputs it as a dictionary of pandas dataframes (or series)

Parameters

- **data** – Dictionary with all the collected values (within lists), from `GdxToList` function
- **fixindex** – This flag allows converting string index into integers and sort the data

Returns dictionary of dataframes

`dispaset.misc.gdx_handler.gdx_to_list(gams_dir, filename, varname='all', verbose=False)`

original This function loads the.gdx with the results of the simulation All results are stored in an unordered list

Parameters

- **gams_dir** – Gams working directory
- **filename** – Path to the.gdx file to be read
- **varname** – In case online one variable is needed, specify it name (otherwise specify 'all')

Returns Dictionary with all the collected values (within lists)

`dispaset.misc.gdx_handler.get_gams_path(gams_dir=None)`

Function that attempts to search for the GAMS installation path (required to write the GDX or run gams)

It returns the path if it has been found, or an empty string otherwise. If a `gams_dir` argument is passed it tries to validate before searching

Currently works for Windows, Linux and OSX. More searching rules and patterns should be added in the future

`dispaset.misc.gdx_handler.get_gdx(gams_dir, resultfile)`

Short wrapper of the two.gdx reading functions (`GdxToDataframe` and `GdxToList`)

Parameters

- **gams_dir** – Gams working directory
- **resultfile** – Path to the.gdx file to be read

Returns dictionary of dataframes

```
dispaset.misc.gdx_handler.import_local_lib(lib)
```

Try to import the GAMS api and.gdxcc to write.gdx files

```
dispaset.misc.gdx_handler.package_exists(package)
```

```
dispaset.misc.gdx_handler.write_variables(gams_dir,.gdx_out, list_vars)
```

This function performs the following: * Use the.gdxcc library to create a.gdxHandle instance * Check that the gams path is well defined * Call the 'insert_symbols' function to write all sets and parameters to.gdxHandle

Parameters

- **gams_dir** – (Relative) path to the gams directory
- **.gdx_out** – (Relative) path to the.gdx file to be written
- **list_vars** – List with the sets and parameters to be written

dispaset.misc.str_handler module

```
dispaset.misc.str_handler.clean_strings(x, exclude_digits=False, exclude_punctuation=False)
```

Function to convert strange unicode and remove characters punctuation

Parameters **x** – any string or list of strings

Usage:

```
df['DisplayName'].apply(clean_strings)
```

```
dispaset.misc.str_handler.force_str(x)
```

Used to get a str object both in python 2 and 3 although they represent different objects (byte vs unicode) It is small hack for py2->3 compatibility of gams APIs which require a str object

```
dispaset.misc.str_handler.shrink_to_64(x, N=64)
```

Function that reduces the length of the keys to be written to 64 (max admissible length for GAMS)

Parameters

- **x** – String or list of strings
- **N** – Integer with the maximum string length (if different from 64)

Returns Shrunk string or list of strings

Module contents

5.10.2 Submodules

5.10.3 dispaset.solve module

This worksheet contains the two main functions to solve the DispaSET optimization problem using PYOMO or GAMS.

Solve with GAMS and the high level API

The high level interface is recommended for Linux users because it solves the “whitespace in the simulation folder” issue.

Installation: To install the high-level API in Python 2.x:

```
cd gams24.4_linux_x64_64_sfx/apifiles/Python/api
python gamssetup.py install
```

To install the high-level API in Python 3.x:

```
cd gams24.6_linux_x64_64_sfx/apifiles/Python/api_34
python setup.py install
```

Solve with GAMS and the low level APIs

Use lower level apis to run GAMS. BAsed on GAMS xpexample2.py

The advantage of the low level API is that it can easily be installed from pip:

```
pip install gdxcc
pip install gamsxcc
pip install optcc
```

Solve with PYOMO

The Pyomo version of Dispa-SET is currently not up-to-date. Use at your own risk.

`dispaset.solve.is_sim_folder_ok(sim_folder)`

Function that checks if the provided path is a valid Dispa-SET simulation folder. The following files are required:

- Inputs.gdx
- UCM_h.gms

Parameters `sim_folder` – path (absolute or relative) to the simulation folder

`dispaset.solve.solve_GAMS(sim_folder, gams_folder=None, output_lst=False)`

Function used to run the optimization using the GAMS engine.

Parameters

- **sim_folder** – path to a valid Dispa-SET simulation folder
- **gams_folder** – path to the gams folder. If not provided, the script will try to find it automatically
- **work_dir** – path to the working directory (does not need to be provided)
- **output_lst** – Set to True to conserve a copy of the GAMS lst file in the simulation folder

`dispaset.solve.solve_pyomo(sim_folder)`

Function used to run the optimization using the PYOMO engine.

Parameters `sim_folder` – path to a valid Dispa-SET simulation folder

5.10.4 Module contents

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

d

- `dispaset`, [74](#)
- `dispaset.misc`, [73](#)
- `dispaset.misc.colorstreamhandler`, [72](#)
- `dispaset.misc.gdx_handler`, [72](#)
- `dispaset.misc.str_handler`, [73](#)
- `dispaset.postprocessing`, [70](#)
- `dispaset.postprocessing.postprocessing`,
[66](#)
- `dispaset.preprocessing`, [66](#)
- `dispaset.preprocessing.data_check`, [61](#)
- `dispaset.preprocessing.data_handler`, [62](#)
- `dispaset.preprocessing.preprocessing`,
[64](#)
- `dispaset.preprocessing.utils`, [66](#)
- `dispaset.pyomo`, [71](#)
- `dispaset.pyomo.model`, [70](#)
- `dispaset.pyomo.utils`, [71](#)
- `dispaset.solve`, [73](#)

A

adjust_capacity() (in module
paset.preprocessing.preprocessing), 65
adjust_storage() (in module
paset.preprocessing.preprocessing), 65
after_solver() (in module dispaset.pyomo.model), 71
aggregate_by_fuel() (in module
paset.postprocessing.postprocessing), 67

B

build_simulation() (in module
paset.preprocessing.preprocessing), 65

C

check_AvailabilityFactors() (in module
paset.preprocessing.data_check), 61
check_chp() (in module
paset.preprocessing.data_check), 62
check_clustering() (in module
paset.preprocessing.data_check), 62
check_df() (in module
paset.preprocessing.data_check), 62
check_heat_demand() (in module
paset.preprocessing.data_check), 62
check_MinMaxFlows() (in module
paset.preprocessing.data_check), 62
check_simulation_environment() (in module
paset.preprocessing.data_check), 62
check_sto() (in module
paset.preprocessing.data_check), 62
check_units() (in module
paset.preprocessing.data_check), 62
clean_strings() (in module dispaset.misc.str_handler), 73
clustering() (in module dispaset.preprocessing.utils), 66
ColorStreamHandler (in module
paset.misc.colorstreamhandler), 72
CostExPost() (in module
paset.postprocessing.postprocessing), 67

D

define_parameter() (in module
paset.preprocessing.data_handler), 63
dispaset (module), 74
dispaset.misc (module), 73
dispaset.misc.colorstreamhandler (module), 72
dispaset.misc.gdx_handler (module), 72
dispaset.misc.str_handler (module), 73
dispaset.postprocessing (module), 70
dispaset.postprocessing.postprocessing (module), 66
dispaset.preprocessing (module), 66
dispaset.preprocessing.data_check (module), 61
dispaset.preprocessing.data_handler (module), 62
dispaset.preprocessing.preprocessing (module), 64
dispaset.preprocessing.utils (module), 66
dispaset.pyomo (module), 71
dispaset.pyomo.model (module), 70
dispaset.pyomo.utils (module), 71
dispaset.solve (module), 73
DispaSolve() (in module dispaset.pyomo.model), 70
DispOptim() (in module dispaset.pyomo.model), 70
ds_to_df() (in module
paset.postprocessing.postprocessing), 67

E

export_yaml_config() (in module
paset.preprocessing.data_handler), 63

F

filter_by_country() (in module
paset.postprocessing.postprocessing), 67
force_str() (in module dispaset.misc.str_handler), 73

G

GAMSstatus() (in module
paset.postprocessing.postprocessing), 67
gdx_to_dataframe() (in module
paset.misc.gdx_handler), 72
gdx_to_list() (in module dispaset.misc.gdx_handler), 72

[get_gams_path\(\)](#) (in module `dispaset.misc.gdx_handler`), [72](#)
[get_gdx\(\)](#) (in module `dispaset.misc.gdx_handler`), [72](#)
[get_git_revision_tag\(\)](#) (in module `dis-
paset.preprocessing.preprocessing`), [65](#)
[get_imports\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [67](#)
[get_indicators_powerplant\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [68](#)
[get_load_data\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [68](#)
[get_plot_data\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [68](#)
[get_result_analysis\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [68](#)
[get_set_members\(\)](#) (in module `dispaset.pyomo.utils`), [71](#)
[get_sets\(\)](#) (in module `dispaset.pyomo.utils`), [71](#)
[get_sim_results\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [68](#)
[get_units_operation_cost\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [69](#)

I

[import_local_lib\(\)](#) (in module `dis-
paset.misc.gdx_handler`), [72](#)
[incidence_matrix\(\)](#) (in module `dis-
paset.preprocessing.utils`), [66](#)
[interconnections\(\)](#) (in module `dis-
paset.preprocessing.utils`), [66](#)
[invert_dic_df\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [63](#)
[is_sim_folder_ok\(\)](#) (in module `dispaset.solve`), [74](#)
[isStorage\(\)](#) (in module `dis-
paset.preprocessing.data_check`), [62](#)
[isVRE\(\)](#) (in module `dispaset.preprocessing.data_check`), [62](#)

L

[load_config_excel\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [64](#)
[load_config_yaml\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [64](#)
[load_csv\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [64](#)

M

[merge_series\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [64](#)

N

[NodeBasedTable\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [62](#)

P

[package_exists\(\)](#) (in module `dispaset.misc.gdx_handler`), [73](#)
[plot_country\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [69](#)
[plot_country_capacities\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [69](#)
[plot_dispatch\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [69](#)
[plot_energy_country_fuel\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [69](#)
[plot_rug\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [69](#)
[pyomo_format\(\)](#) (in module `dispaset.pyomo.utils`), [71](#)
[pyomo_to_pandas\(\)](#) (in module `dispaset.pyomo.utils`), [71](#)

R

[run_solver\(\)](#) (in module `dispaset.pyomo.model`), [71](#)

S

[shrink_to_64\(\)](#) (in module `dispaset.misc.str_handler`), [73](#)
[solve_GAMS\(\)](#) (in module `dispaset.solve`), [74](#)
[solve_pyomo\(\)](#) (in module `dispaset.solve`), [74](#)
[storage_levels\(\)](#) (in module `dis-
paset.postprocessing.postprocessing`), [70](#)

T

[try_solvers\(\)](#) (in module `dispaset.pyomo.model`), [71](#)

U

[UnitBasedTable\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [63](#)

W

[write_to_excel\(\)](#) (in module `dis-
paset.preprocessing.data_handler`), [64](#)
[write_variables\(\)](#) (in module `dispaset.misc.gdx_handler`), [73](#)