University of Liège - School of Engineering and Computer Science

MASTER'S THESIS

Improving the simulation of variable renewable energy in integrated assessment models

Master's thesis completed in order to obtain the degree of Master of Science in Computer Science Engineering by Straet François

Supervisors:
Prof. S. Quoilin

Academic year 2022-2023

Keywords

CF	Capacity factor
DLL	Dynamically loaded library
LP	Linear programming
LHS	Latin hypercube sampling
MILP	Mixed integer linear programming
MTS	Mid-term scheduling
P2H	Power
RES	Renewable energy sources
VRES	Variable renewable energy sources

Acknowledgements

I would like to thank professor Quoilin first, for his mentoring during this thesis and his advices. Then Jade, for her collaboration on the linkings between the created model and MEDEAS.

Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11 and by the Walloon Region. I am also grateful to the cluster administrator, M. Collignon, for his patience and valuable advices.

Thanks should also go to my mother, and everyone to which I talked about my thesis, what sometimes led me to get a new perspective on the work as I had to explain it in a new, concise and understandable way.

Abstract

TODO

Todo-list

1. Check that all Keywords are actually used

Contents

Warnings

I have 0 warnings:

– `Warning` No.1

– `Warning` No.0

1 Introduction

Our societies run on energy. Energy is required to power pretty much whatever we do and need. Fossil fuels provided us with relatively easy to access, store and use energy for decades, bringing a solution to a problem.

Now, the awareness of the climate change, mostly caused by the release of large amounts of greenhouse gases, among them carbon dioxyde, produced by the combustion of these fossil fuels, is a game-changer. In order to acheive set targets in terms of climate evolution, bounds on carbon emissions have been defined, and therefore limiting the prevously endless source of energy that were fossil fuels in the very long run.

The solutions to compensate for the lacking energy generation, that from now on should not originate from fossil resources, are renewable energy sources. These refer to every energy generation technique originating from a renewable source, such as the sun, the wind and the rivers. However, the creation of these units may not be completely renewable in itself, for example, the photovoltaic cells necessary for the exploitation of the incoming solar energy are pretty difficult to recycle, making them rely on specific materials that are not obtainable renewably. Still, their use on a complete lifetime, and increasing capabilities in recycling, pays off the invesment made at their construction.

In this context, a significant increase in the energy produced from such energy sources is to be expected, in particular, from:

- the sun, through photovoltaic panels,
- the wind, through on-shore and off-shore wind turbines, and
- rivers, through hydroelectric dams.

The third one, due to its dependency on the geographic context, will however not expand foerever, as there are not illimited spots to build such dams.

One thus falls back to photovoltaic and wind energy, but both have a major, trivial drawback: they rely on the sun and the wind, respectively. And this becomes a huge deal, because the amount of energy that can be generated by exploiting these is variable, hence their designation as variable renewable energy sources, or VRES.

It is to be mentioned that these variabilities comprise some predictability, for example, there is on average more photovoltaic production potential during summer. On a daily scale as well, with the day night cycle. Weather forecasts can also be taken advantage of in order to predict wind turbines' production.

The larger variability of the power output of VRES causes a problem, because modern electrical system is dictated by consumers' demand. And to do so, electricity generators are dispatched in real-time, so that the production always matches the demand on the network. This technique works because the concerned power plants can be started and shut down on demand, almost at any time. With VRES, this assumption drops, as one cannot start an extra wind turbine if there is no wind, nor use a photovoltaic panel during the night.

1.1 Flexibility assessment in energy system models

As explained before, higher shares of VRES in the electricity production mix create the additional challenge, that is the handling of the partlially predictable variability of wind and sun energy.

This handling requires a larger flexibility of the electrical system, that is, a better ability to adapt to changes in the demand and supply, either expected or not [25].

To improve the flexibility of an existing power system, some mechanisms already exists.

- Use of the regular dispatchable energy production plants to mitigate the energy not produced from VRES. Depending on the plant characteristics, it may be difficult to address short term drops in production, as there is some start up time required [1].
- Large interconnected electricity networks, that are able to smooth the VRES power output. There may be not enough sun in some region, creating a deficit, while there is too much in the neighboring country. By connecting them, the overproduction will compensate the unerproduction of the other [13].
- Energy storage facilities. Of course, storing the produced energy for later use is an easy way to account for the intermittency of the production. Typically, storing solar energy during day time to be used in the night. These technologies include pumped hydro-storage, batteries, compressed air. While pumped hydro-storage units are the most common, their very limited expansion options make them unlikely to grow in the future [27].
- Acting on the demand, in the extent that it can change its shape by promoting policies to the end users. Such policies focus on flattening the daily demand curve, facilitating the energy production dispatch. Typically, asking to delay greedy devices like dishwashers until night, where the overall demand is lower. But it may concern other domains, such as electric vehicles charge, heating and cooling etc [27].

The two main consequences of insufficiently flexible energy systems are curtailment, when there is too much energy produced, and load shedding, when there is not enough electricity to satisfy the demand. In case of load shedding, parts of the grid may be entirely shut down.

1.1.1 Literature review

Similar work has been done by Parrado-Hernando et al. in [22], that aims at "capturing features of hourly-resolution energy models". The methodology used to acquire the data presents two downsides that are adressed in this work.

First, the inputs of the energy model are handled as discrete variable, and simulation have been run using all possible combination for these values. This can be seen in some of their figures, where the data points seem to follow some lines. While it does not invalidate the process, this creates a bias in the repartition of the data. Second, linear approximation are used to fit the data obtained. Admittedly, this is identified as a limitation of the work.

To palliate these, the input space will be tackled as a continuous domain for the design of experiment, and other machine learning technique will be considered as to candidates for the creation of the surrogate model.

1.2 Short-term dispatch models

There exists tools built in order to assess the behaviour of large electrical systems, that are subject to higher share of VRES. These tools typically aims at predicting the electricity flows, dispatching available power plants in order to match the production to the demand. And from there on, some higher level metrics can be computed, and in particular, we will be interested by:

– the curtailment, that is, the energy produced in excess while the electricty demand was already met, that end up wasted, and

– the lost load, that is, the energy that could not be produced, hence some demand could not be served.

Among these tools, the Dispa-SET model will be considered in this work. Dispa-SET is open-source, and focused on balancing problem in the European grid specifically.

This model is formulated in linear programming, that is, a set of linear constraints are defined and an objective function is given. The solver inputs both of these and computes the parameters values that maximize the objective function while matching the constraints.

1.3 Integrated assessment models

On another level, integrated assessment models (IAM) aim at estimating the evolution of large, intricated systems involving a lot of different interconnected areas and actors. These are most often multidisciplinary and require a lot of modelling choices.

In particular, some IAM attempt to model the evolution of the whole society, from a socio-economical perspective, including environmental aspects and energy concerns.

In this subset of IAMs, the MEDEAS model is chosen for this work, being open-source as well and providing a EU-specific model.

MEDEAS is expressed in terms of systems dynamics, that is, the evolution of the state of the simulation is computed as a function of its current state. And this comes down to solving a set of differential equations.

1.4 Model linking

Due to computational constrain, IAMs often have a pretty low level of temporal, or spacial accuracy [23]. This is not the case for dispatch models, that carry out more extensive simulations. Therefore, it is meaningful to link two of these models together, as the IAM would benefit from the better accuracy of the energy models.

An high-level illustration of the positionw of MEDEAS and Dispa-SET on the timescale is provided on Figure 1.1 [12].

1.4.1 Linking types

There are several strategies that may be used in order to link two models together [23].

– Soft linking: the models communicate between each other. This communication may be uni-directional or bi-directional. Both of the models are run iteratively, thus keeping their separate efficiencies in the same order of magnitude. However, the iteration lead to low overall speed, and convergence is not guaranteed.

– Hard linking: the models are combined into a single, unified mathematical formulation. This newly created model can then be solved all at once. This approach is burdened by higher computational costs and lower chance of feasibility.

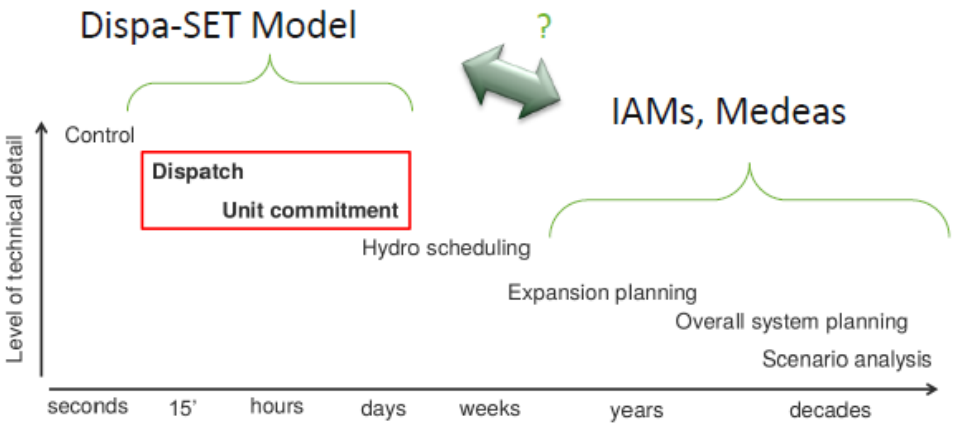


Figure 1.1: An illustration of where Dispa-SET and MEDEAS operate on the timescale

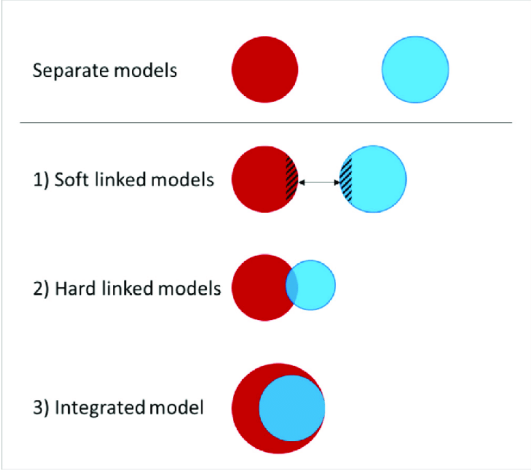


Figure 1.2: Illustration of the different linking methods [11].

These linking types are illustrated on Figure 1.2.

One may also add model integration, consisting in completely embedding a model into the other. But this is not tractable in this setting, and would also requires compatible model formulations, as explained below.

In this case, hard linking is not possible due to the different formulations of Dispa-SET, in linear programming, and MEDEAS, in differential equations.

We also dismiss the soft linking strategy because of its slowness, and absence of convergence guarantee.

1.4.2 Surrogate models

The linking technique explored in this work is the surrogate model. In this case, an approximation of the Dispa-SET dispatch model will be integrated in the MEDEAS IAM.

The idea behind surrogate models is simple. First, a fast, easy-to use approximation of the model is made, then it its completely integrated in the other model. In this case, the relevant outputs of the Dispa-SET model will be approximated from relevant inputs regarding MEDEAS,

then the approximator will be integrated in the model.

The soft-linking approach has already been explored [2], [7], and the hard-linking remains the hardest to investigate, notably due to incompatible formulations. Surrogate models are still unexplored and are of great interest for this use case.

As the combination of the model will use an approximation, this appriximation being fast will not burden the computations of the IAM, hence keeping it efficient.

1.5 This work

1.5.1 Objective

This master's thesis is dedicated to the integration of the flexibility constraints, the Dispa-SET surrogate model, into the MEDEAS model, being less precise on the matter. This includes the creation of a proper dataset, the definition, training and integration of the surrogate model into MEDEAS.

1.5.2 Contributions

This works follows what was started by another student, Carla Vidal, that went until the surrogate model training, included. Given that improvements where implemented in Dispa-SET since then, the runs had to be re-done. However, there were no easy to use scripts to set up the simulation files etc., so that is has been chosen to write new ones.

Furthermore, the present thesis also considers other machine learning algorithms, although the same choice of neural networks is made, this time based on better performance compared to the other options.

Another student, Jade Paris, was in charge of the linking of the surrogate model, given as a functions of some inputs variables into the MEDEAS model, and its actual use and runs of MEDEAS with the surrogate model integrated.

This work constisted in:

- The writing of easy-to-use scripts to run Dispa-SET on those experiments
- The efinition and implementation of an adequate machine learning model (neural net), and training
- The integration the model in MEDEAS, by writing a C++ external function library for Vensim, or by inserting it into the PySD model of MEDEAS directly in python
- Runs and analysis of the improved MEDEAS model

All the produced work, and necessary data is available on the online github repository at the following address: https://github.com/Rayerdyne/master-thesis.

1.5.3 Outline

This document is organized into seven main sections. The second section presents an in-depth description of the Dispa-SET model, along with the tools employed within its framework. Following this, the third section provides a detailed account of the MEDEAS model, outlining its underlying principles and the broader framework in which it operates.

To facilitate the successful integration of models, the fourth section outlines the process of data generation. This encompasses a complete overview of the methodologies employed to produce the necessary training data, the design of experiments and the execution of Dispa-SET runs.

Subsequently, the fifth section characterizes the surrogate model, offering a comprehensive description of its design. The section further expounds on the training and validation processes undertaken to ensure the surrogate model aligns accurately with the main models' outcomes.

The sixth section then shifts the focus to the crucial process of integrating the surrogate model into MEDEAS. A step-by-step description is provided, elucidating how the surrogate model becomes an integral component of the broader MEDEAS framework, and how it interacts with the existing models.

Finally, the document concludes in the seventh section, offering a comprehensive summary of the resulting model and outcomes. This concluding section also identifies any limitations and outlining future areas of research for further enhancements.

2 The Dispa-SET model

2.1 Overview

The Dispa-SET model [12] is described in [26] as "an open-source unit commitment and optimal dispatch model focused on the balancing and flexibility problems in European grids".

More precisely, it is focused on simulating large scale power systems, with emphasis on high shares of VRES. It follows that it is used as tool for the analysis of the impacts of VRES on the power systems, thank to its ability to take into account several technical constraints of the power system.

A schematic of the Dispa-SET architecture is given in Figure 2.1.

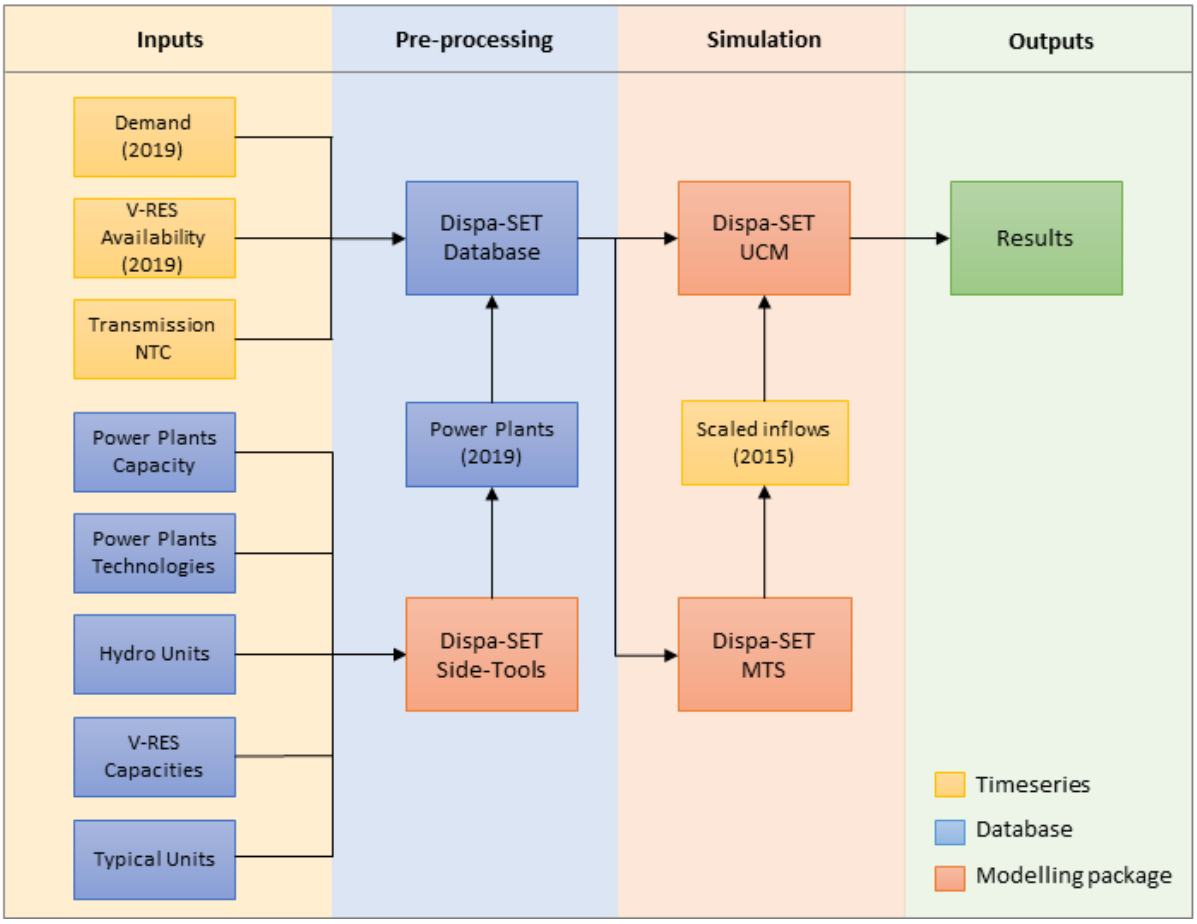


Figure 2.1: Block diagram of Dispa-SET architecture

Dispa-SET has several option regarding the formulation of the problem:

– Linear programming (LP) optimization problem

– Mixed Integer Linear Programming (MILP) optimization problem.

Its interface is written in the Python programming language, and calls GAMS [6] as the main solver engine.

2.2 Objective function

The Dispa-SET model aims at minimizing the overall operating costs of the grid, that is, its objective function. These costs typically include transportation, power and heating costs required to split efficiently the demand between the available generation units.

The total system costs is splitted as follows:

- *Fixed cost*: fixed amount, charged if the unit is on.
- *Variable costs*: amount that is a function of the power output the units are operating at.
- *Start-up* and *Shutdown costs*: amount charged on start and on shutdown of a unit.
- *Ramp-up* and *Ramp-down costs*: costs due to the increase or decrease in power output of a unit.
- *Shed load costs*: costs due to necessary load sheddings.
- *Loss of load costs*: due to generated either exceeding the demand, or not matching it .
- *Transmission costs*: due to the use and wear of the transmission network.
- *Spillage costs*: due to spillage in storage units.

Adding these, Equation 2.1 is obtained, where u refers to the index on each units, and i is the time index.

2.3 Supply and demand balance

At all time and in each zone, the fundamental constraint that has to be met is the supply-demand balance in terms of energy production (supply) and consumption (demand), in the day-ahead market.

The supply sources are:

- The power outputs from each units
- The power outputs from storage units discharging
- The (eventual) net income from importation from neighbouring zones
- The (eventual) shed load

Whereas the demand originates from:

- The load in that zone
- The (eventual) net exportations to neighbouring zones
- The power consumed by charging storage units
- The power consumed by P2H units

The Equation 2.2 expresses this target equality.

$$
\begin{aligned}
Min_{TotalSystemCost} = \\
&\sum_{u,i}(CostStartUp_{i,u} + CostShutDown_{i,u}) + \\
&\sum_{u,i}(CostRampUp_{i,u} + CostRampDown_{i,u}) + \\
&\sum_{u,i} CostFixed_u \cdot Comitted_{i,u} \cdot TimeStep + \\
&\sum_{u,i} CostVariable_{i,u} \cdot Power_{i,u} \cdot TimeStep + \\
&\sum_{hu,i} CostVariable_{i,u} \cdot Heat_{i,u} \cdot TimeStep + \\
&\sum_{l,i} PriceTransmission_{i,l} \cdot Flow_{i,l} \cdot TimeStep + \\
&\sum_{n,i} CostLoadShedding_{i,n} \cdot ShedLoad_{i,n} \cdot TimeStep + \\
&\sum_{n\_th,i} CostHeatSlack_{n\_th,i} \cdot HeatSlack_{n\_th,i} \cdot TimeStep + \\
&\sum_{n\_h2,i} CostH2Slack_{n\_h2,i} \cdot H2Slack_{n\_h2,i} \cdot TimeStep + \\
&\sum_{chp,i} CostVariable_{chp,i} \cdot CHPPowerLossFactor_{chp,i} \cdot Heat_{chp,i} \cdot TimeStep + \\
&\sum_{i,n} VOLL_{Power}(LL_{MaxPower,i,n} + LL_{MinPower,i,n}) \cdot TimeStep + \\
&\sum_{i,n} 0.8 \cdot VOLL_{reserve}(LL_{2U,i,n} + LL_{2D,i,n} + LL_{3D,i,n}) \cdot TimeStep + \\
&\sum_{u,i} 0.7 \cdot VOLL_{Ramp}(LL_{RampUP,u,i} + LL_{RampDown,u,i}) \cdot TimeStep + \\
&\sum_{s,i} CostOfSpillage \cdot Spillage_{s,i}
\end{aligned}
\tag{2.1}
$$

Equation 2.1: Objective function of the Dispa-SET model

$$\sum_{u}(Power_{u,i} \cdot Location_{u,n}) + \sum_{l}(Flow_{u,i} \cdot LineNode_{l,n})$$
$$= Demand_{DA,n,i} + Demand_{Flex,n,i} + \sum_{s}(StorageInput_{s,i} \cdot Location_{s,n}) +$$
$$\sum_{p2h}(PowerConsumption_{p2h,i} \cdot Location_{p2h,i}) - ShedLoad_{n,i} - LL_{MaxPower_{n,i}} + LL_{MinPower_{n,i}} \tag{2.2}$$

Equation 2.2: Supply-demand balance in Dispa-SET

2.4 Rolling horizon

The perfect solution would be solving the whole system, for every time step in the complete duration of the simulation in one go. But this would create a system too large to be reasonably solved.

To manage this, the simulation is split into smaller, manageable parts. The simulation is made for a smaller time frame, called the optimization period, over which the simulation can be made easily.

The start of optimization period j overlaps the optimization period $j-1$, to that the simulation j is the correct prolongation of the same setting fixed by the simulations up to $j-1$. The period that overlaps is called the look-ahead, in which the values of the parameters for period $j-1$ is determined during simulation $j-1$, and are used as fixed context for simulation j. A depiction of rolling horizon is given on Figure 2.2.

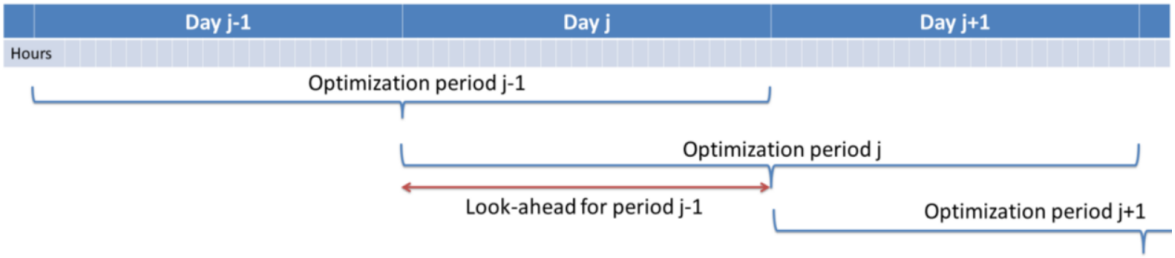


Figure 2.2: Depiction of the rolling horizon mechanism

2.5 Mid-term scheduling

Without further constraints, the optimization will most often leave all the storage facilities empty at the end of the simulation horizon (typically a few days). This is a consequence of the variable operational cost of discharging these storage units being smaller than the cost of running another unit, thus charging extra fixed and variable costs.

In order to tackle this issue, the Dispa-SET model has to be run in Mid-Term Scheduling (MTS) mode. In this mode, the initial and final levels of the storage units (in particular, pumped hydro storage units) are given as exogenous input to the model. These levels are enforced with additional constraints.

In this work, the MTS mode is enabled and the exogenous inputs, for both initial and final levels, are set to half of the storage capacity.

2.6 Dispa-SET components and representation

Dispa-SET provides us with several predefined configurations, each of these defining the zones and their units of interest, and linking to the relevant data (e.g. times series provided as `csv` files).

In this work, the european setting is used.

2.6.1 Zones

Most of the EU contries are represented, for completeness they are reported on Table 2.1.

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

Table 2.1: Countries present in Dispa-SET EU, and their ISO Alpha 2 country codes. These are all the EU contry except for Cyprus and Malta and Luxembourg, plus Norway, Switzerland and the UK.

2.6.2 Technologies

Table 2.2 lists all the technologies taken into account by Dispa-SET, alongside with their main properties:

- VRES: does the technology belongs to VRES?
- Storage: can it store energy?
- Flexibility: ease of control of the unit's power output.

Due to the intermittency of their resources, and because one cannot dispatch them, VRES are considered inflexible.

Although, hydroelectric units disposing of a reservoir are have some room for flexibility, given their ability to manage their storage level.

Identifier	Description	VRES	Storage	Flexibility
COMC	Combined cycle	No	No	High
GTUR	Gas turbine	No	No	High
ICEN	Internal combustion engine	No	No	High
STUR	Steam turbine	No	No	Medium
HDAM	Conventional hydro dam	No	Yes	Medium
HROR	Hydro run-of-river	Yes	No	Low
HPHS	Pumped hydro storage	No	Yes	Medium
WTOF	Offshore wind turbine	Yes	No	Low
WTON	Onshore wind turbine	Yes	No	Low
PHOT	Solar photovoltaic	Yes	No	Low
BATS	Stationary batteries	No	Yes	High

Table 2.2: Technologies present in Dispa-SET

Steam turbines, because of their dependency on the fuel used, e.g. nuclear energy would be less flexible than natural gas.

Heating and combined heat and power units are not covered, as only the electricity is of interest in this scope.

2.6.3 Fuels

Table 2.3 summarizes the fuel types in Dispa-SET.

It is important to highlight that technologies may not always be powered by the same fuel, for instance, the steam turbines can use most of them.

Each unit must specify its technology and fuel. Depending on the optimization problem formulation, units featuring the same (technology-fuel) pair will be grouped together and thereafter be treated as one single unit.

Fuel	Description
BIO	Biofuels
GAS	Gas
HRD	Coal
LIG	Lignite
NUC	Nuclear energy
OIL	Petroleum
PEA	Peat Moss
GEO	Geothermal steam
SUN	Solar energy
WAT	Hydro energy
WIN	Wind energy
WST	Energy from waste
OTH	Other fuels and energy carriers

Table 2.3: Fuel types in Dispa-SET

A major consideration for the optimization problem is the fuel prices, that are summarized in Table 2.4.

A key feature is the relationship between the price of coal and the price of gas: depending on which one is the cheaper, the optimal behaviour change dramatically. Obviously, the cheapest one will always be preferred over the other when choice arise.

	Price
Nuclear	3
Black coal	20
Gas	45
Fuel-Oil	65
Biomass	10.08
Lignite	7.23
Peat	9.36

Table 2.4: Fuel prices considered, in €/MWh

2.6.4 Other prices

Some other price values are relevant, such as the price of the load shedding per MWh. These are presented in Table 2.5.

What	Price
CO2	25
Unserved Heat	84.21
Load Shedding Cost	1000
Transmission	0
Unserved H2	75
Curtailment Cost	20

Table 2.5: Other relevant prices, in €/MWh

2.6.5 Power plants

As a dispatch model, Dispa-SET oviously has to model the units it dispatches, namley the power plants that are present in each of the modelled zones.

For performance reasons, some of the units initially described are merged into clustered units at the pre-processing step. Thus, the amount of variable in the simulation is reduced, while the accuracy is not significantly impacted [12].

Dispa-SET disposes of utilities to do so, but also needs to craft the new, aggregated units properties table. These are defined by the set of feilds that are shown on Table 2.6.

For the storage units, one needs some more parameters, given on Table 2.7. For the other unit types, these feilds will be left empty.

Their discharge efficiency will be assigned to the common Efficiency field, and the PowerCapacity will be assigned the power output on discharge.

For batteries units, the RampUpRate and RampDownRate fields are set to 1, while the others but efficiency are set to 0. In previous work in this context, the number of hours a unit can run at maximum output capacity is fixed as 4 hours, thus implicitly fixing a storage capacity given a power output.

Field	Description	Type
Unit	Unit name	string
PowerCapacity	Maximum power output	value in MW
Nunits	Number of initial units clustered	integer
Zones	The unit's zone	string
Fuel	The fuel used	string
Efficiency	The unit's efficiency	real in [0,1]
MinEfficiency	Efficiency at minimum load	real in [0,1]
MinUpTime	Minimum up time	value in hours
MinDownTime	Minimum down time	value in hours
RampUpRate	Ramp up rate	value in minute^{-1}
RampDownRate	Ramp down rate	value in minute^{-1}
RampingCost	Cost of ramping up or down	value in €/hour
StartUpCost_pu	Start up cost per clustered unit	value in €
NoLoadCost_pu	Cost of having no load on a unit	value in €/hour
PartLoadMin	Ratio of the minimum nominal capacity	real in [0,1]
StartUpTime	Time to start up the plant	value in hour
CO2Intensity	Amount of CO_2 emitted per MW	value in €/MW

Table 2.6: The table fields used to describe a optionnally aggregated power plant unit

Field	Description	Type
STOCapacity	The total energy storage capacity	value in MWh
STOSelfDischarge	The discharge rate (w.r.t. to the total)	value in day^{-1}
STOMaxChargingPower	Maximum energy inflow	value in MW
STOChargingEfficiency	The unit's charging efficiency	real in [0,1]

Table 2.7: Fields describing the storage capabilities of the units

This choice is arbitrary and leads to a simplification of the reality, where one could find huge differences in this ratio. To remove this, an option is added in Dispa-SET's adjusting function, to be able to filter the adjustments by range, making it now able to discriminate the units based on the storage capacity over maximum output power ratio, enabling its use to adjust storage units with different "longevity" separately.

However in reality, most of the difference comes from the pumped hydro storage, that can typically output their maximum power for a longer time than the other storage technologies, such as batteries.

At the end, this differenciation is not done, as it would also require the inputs of the surrogate model to be changed, to take into account the share of "high-longevity" storage units with respect to the "low-lengevity" ones.

2.6.6 Notes on the other inputs

- The electricity demand is a time series from year 2019, per zone. It is assumed to be independent of the price.

- The net transfer capacities (NTC) between the different zones are given as inputs as hourly times series over a year. Then the maximum is picked and it is assumed that it remains

constant over the year.

– The availability factors (AF) for renewable energy sources, defined as the ratio of the nominal power that is possible to output hourly. It is given as an hourly time series (adimentional).

  This variable energy generation is either curtailed or sent to the grid.

  Non-renewable technologies have their AF set to 1.

2.7 Problem formulations

Dispa-SET features several, different formulations, that have a significant impact on the realism and accuracy of the output.

2.7.1 Linear programming

In the LP formulation, every variable is considered to be continuous, and can take values down to zero.

In particular, this constraint applies to the power output of each unit, that is, the ratio of the actual power the unit is operating over its maximum capacity. Therefore, as a unit cannot produce more than its maximum capacity, this value must lie in the $[0, 1]$ interval at every hour of the simulation.

This constrain causes a significant issue, in the extent that each unit cannot specify a minimal level of operation, e.g. 0.5. This leads to unrealistic situations being considered valid, for example a nuclear unit operating at 10% of its maximal power.

2.7.2 Binary formulation

To mitigate the issue encountered with the LP formulation, it is extended by adding a boolean variable to each unit, which determines wether or not the corresponding unit is started or not. That way, the minimum operating value can be enforced.

This strategy, however acheiving the better accuracy targetted, is unfortunately often intractable for realistic settings, because the large number of binary variables, yields an exploding number of 2^N options to be examinated.

2.7.3 Mixed integer linear programming

The MILP formulation is created to leverage the computational cost of the simulation while keeping a good level of accuracy. The main idea is to group the units that share similar properties together, and only keep track of the number of units in each group that are currently running.

This permits the number of options to explore reasonnable, without hurting the precision of the output.

In this work, the MILP formulation is chosen, because of its better efficiency to computational cost trade-off.

3 The MEDEAS model

3.1 Overview

This sections aims at describing the main components of the MEDEAS IAM and its components, with a focus on the parts that are the most relevant in this context.

The project, named "Modelling the Energy Development under Environmental And Socioeconomic constraints", aims at creating a modern computational model to predict the future of the energy systems in Europe, while integrating a wide range of physical and social constraints.

First, the IAMs are depicted, then a high-level description of the MEDEAS model is provided. The third subsection will explain more in depth in a key component of MEDEAS, the energy return on investment. In the fourth subsection, the modelling of the RES in MEDEAS is covered. Finally, the principal scenarios in MEDEAS are presented in the fifth subsection.

3.2 The MEDEAS integrated assessment models

MEDEAS is an open-source integrated assessment model (IAM), built to "guide the transition to a low carbon European socio-economy" [17].

Integrated assessment models are used to make general purpose analysis, combining aspects from different disciplines, such as economy, environment and energy, land use etc. These kinds of models, once properly defined from a mathematical point of view, can then be simulated by computer, for their result to be analysed.

There is a large variety of IAMs, because there are a lot of ways to model the complex interactions, and the large amount of uncertainties between the different components of a model. Hence, there exist a lot of different approaches to the creation of an IAM.

As a side note, the model being open-source is probably one strength as an IAM, meaning that any expert in one domain may be able to contribute to the project.

As previously quoted from their website, the MEDEAS model has been built with the purpose of guiding decarbonation in Europe. It has been designed to compensate for the flaws of other available IAMs, in order to inform policy makers towards a transition to more carbon-independent, sustainable energy.

MEDEAS is built using the Vensim software, and may be used from the Python programming language through the `pySD` package.

3.3 Model overview

MEDEAS funds come from the EU for its Horizon 2020 program, under the "Modelling and analysing the snergy system, its transformation and impacts (social, environmental and economic aspects of the energy system)".

And to do so it models the long-term implications of the decisions made by the society. As one cannot predict them, several hypothesis are needed to fix the choices that will be made. Such a set of hypothesis on the evolution of the long-term policy of the society as a whole is called a scenario.

MEDEAS typically sets the simulation horizon between 1995 and 2060, while for longer term analysis it may be raised up to 2100. It also features different settings:

– MEDEAS-W, the global one,

– MEDEAS-EU, targeting the European Union,

– MEDEAS-AU and MEDEAS-BG, targetting Austria and Bulgaria respectively.

The MEDEAS IAM is organized into seven modules, that are economy, energy, energy infrastructures, materials, land use, climate change and socio-environmental impacts indicators. The general structure is illustrated on Figure 3.1.

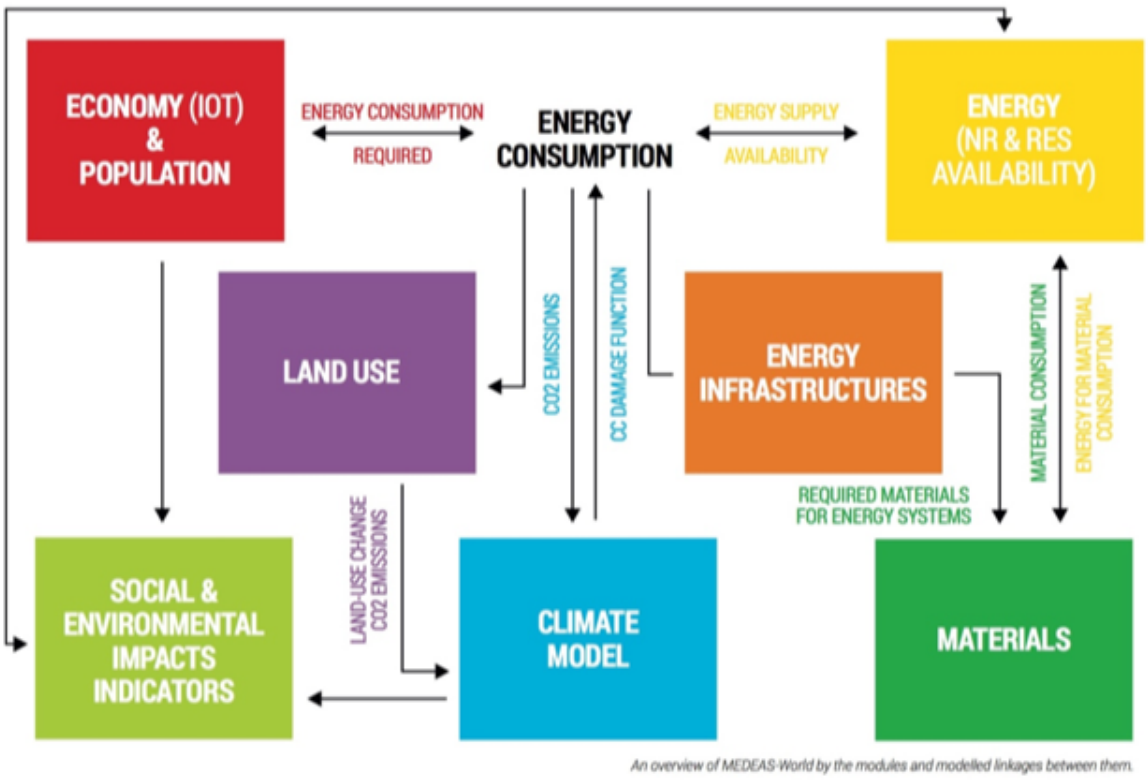


Figure 3.1: The MEDEAS IAM modules

Quite evidently, in this work, the MEDEAS-EU version will be considered, as this is the chosen setting for Dispa-SET, and with an obvious focus on the energy module.

3.3.1 System dynamics

It is formulated using the system dynamics toolset—what Vensim is built for—which is to help the aggregation of the knowledge from different experts from different domains and backgrounds. This will also enable easy modelling of feedback between the different components.

System dynamics is a modeling language, made to obtain understandings of complex and dynamic systems. It acheives using nodes and arrows, corresponding to values and relations between those respectively. The former can be either a stock, a flow or a constant, while the latter makes the link through equations. It is therefore possible to create feedback loops and complex interconnected networks.

System dynamics have been popularized by the *world3* model, used to predict the consequences of long-term policies applied by the society on the planet in the Limits to Growth report [16].

Needless to say, this approach is far from the linear programming paradigm, that has a too different approach for both to be combined.

3.4 Energy return on investment

When having energy consideration in the long term, the energy return on investment (EROI) becomes a key indicator.

It is defined as the ratio of the exploitable energy obtained from some energy resource to the amount of exploitable energy used to get that energy resource [30].

The EROI is of great importance for the assessment of the energy sources efficiency, providing a measure of how efficient this energy source is to make use of. On most cases, the EROI of RES is lower, meaning a lower energy gain, than fossil fuels'.

It is not to be confused with the net energy gain, that is the difference between the exploitable energy obtained and the exploitable energy invested. The net energy is the amount of energy that has been made exploitable, thus now availible for public consumption. Obviously, its value should be larger than zero for the system to be profitable.

Equations 3.1 and 3.2 summarize their definitions and the relationship between the two.

$$EROI = \frac{Energy_{returned}}{Energy_{invested}} \tag{3.1}$$

$$NetEnergy = Energy_{returned} - Energy_{invested} = Energy_{returned}\Big(1 - \frac{1}{EROI}\Big) \tag{3.2}$$

However, this also requires to set a definition on what exactly is the energy invested on the aquisition of an other resource. The MEDEAS model thus provides several EROI values relating to different approaches in this matter [5].

- Standard EROI, that "includes the direct (i.e. on site) and indirect (i.e. offsite energy needed to make the products used on site) energy requirements to get the energy (e.g. build, operate and maintain a power plant)" [5].

- Point of use EROI, that includes the energy cost of obtaining and transporting the fuel to the actual location where it will be used by society.

- Extended EROI, that " considers the energy required to get, deliver and use a unit of energy, i.e. the energy required to produce the machinery and devices used to build, operate and maintain a power plant or a transportation facility (tank truck, pipeline, etc.) as well as the energy required for exploration, investment, communication, labour, etc. in the energy system" [5].

In this context, we will focus on the standard EROI.

In MEDEAS, the EROI is dynamically computed, meaning it is an endogenous variable, as the ratio of the exploitable energy delivered to consumers, over the sum of the total energy costs required for operating the plant and the energy costs of handling the variability of the power output and the costs of operating the energy transportation network. The total costs for an operating plant include the building, maintenance and disposal costs.

On Figure 3.2, a depiction of a high-level energy flow is given. The exploitable energy delivered to consumers is the flow labelled (1), and the operating costs of the plants (2), whereas the costs accounting for the handling of variabilities is labelled (3) and for energy transportation (4). Equation 3.3, 3.4 and 3.5 below shows the MEDEAS' computation of the EROI based on these.

$$EROI_{st} = \frac{(1)}{(3) + (4)} \tag{3.3}$$

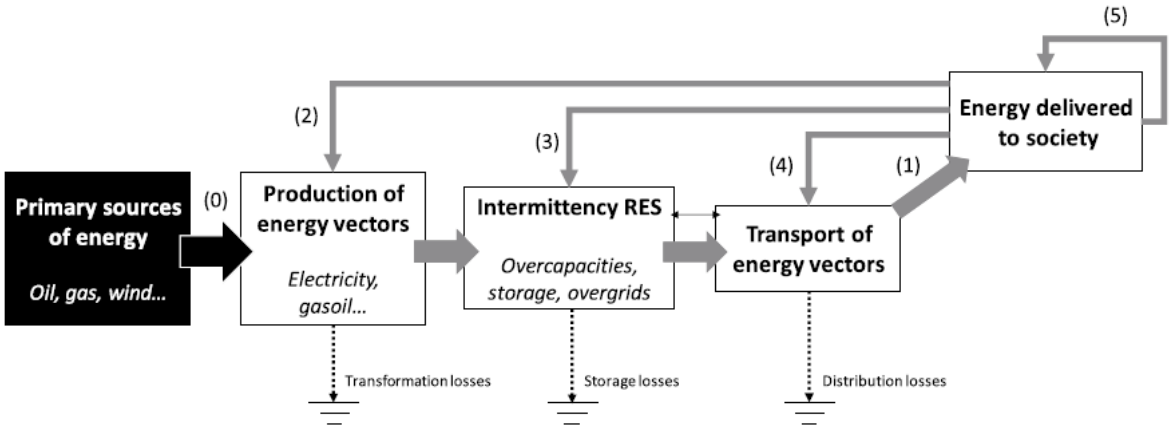


Figure 3.2: Representation of society's principal energy flows[3]

$$EROI_{pou} = \frac{(1)}{(2)+(3)+(4)} \tag{3.4}$$

$$EROI_{ext} = \frac{(1)}{(2)+(3)+(4)+(5)} \tag{3.5}$$

Furthermore, MEDEAS [3]:

- Assumes the EROI of non renewable energy sources to be constant over time,
- Dynamically estimates the EROI of RES producing electricity,
- Allocates technologies based on their EROI as a performance measure, meaning that higher EROI RES will be preferred,
- Computes overcapacities as a result of an increasing share of VRES endogenously,
- Takes additional losses into account for the use of storage units.

3.5 Modelling of RES

The impacts of the variability of electricity production technologies are tackled in the MEDEAS framework. Indeed, not as extensively as they are in Dispa-SET simulation, what is the whole point of the present work.

3.5.1 Limitation

First, it is recalled that the time step used by MEDEAS is quite significant. Is recommended value is 0.03125, expressed as a fraction of a month. This amounts to approximately a day: $0.03125 \times \frac{365.25}{12} = 0.951$.

This value is obviously too high to model extensively daily variations of the power output of solar photovoltaic units, for example.

In the following, the mechanisms implemented by MEDEAS in order to incorporate the RES variability are described.

3.5.2 Grid extension

MEDEAS estimates, per MW of VRES, the additional electricity grid extensions required in order to incorporate those in the existing network. The materials needed for these expansions are also computed, what ultimately affects the EROI.

3.5.3 Storage units

In MEDEAS, the first storage technology in use is the pumped hydro-storage (PHS). One will notice that it contrasts with the Dispa-SET setting where batteries are used as the main storage technology to act on, mostly by creating new plants. As already discussed before, that is to account for the fact that the european region is almost saturated in terms of PHS, as one could not find suitable location for new units to be built.

An estimation of the storage needs as a function of the VRES share is depicted on Figure 3.3 [19].

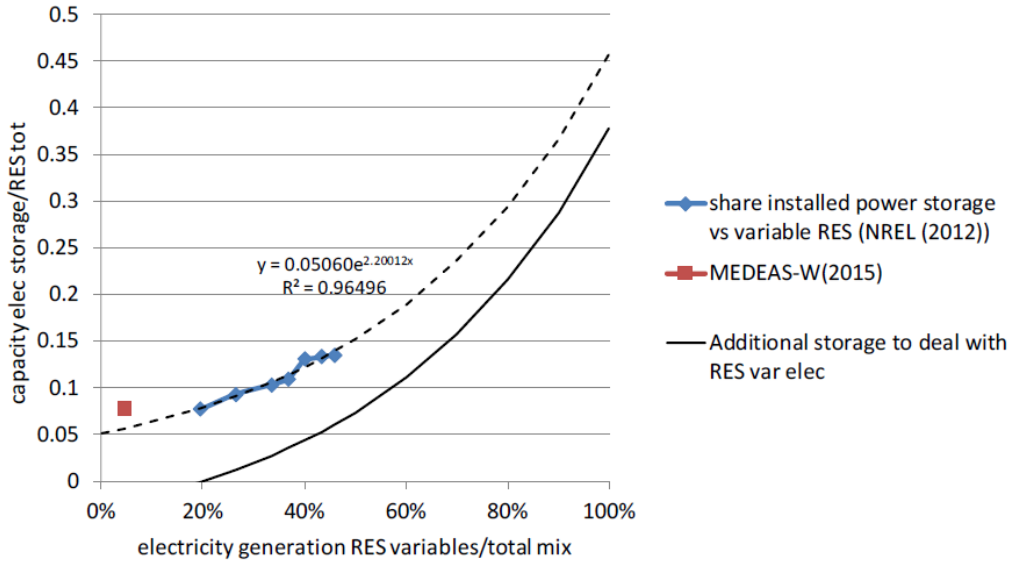


Figure 3.3: Energy storage capacity required as a function of the VRES share according to [19].

3.5.4 Dispatchable RES pants

Here will be describe the evolution of the capacity factor (CF) as the RES share among the electricitym mix changes. The capacity factor is the ratio of the electricity produced by a unit over a period of time Δt over the maximum amount of energy that could have been produced. This is represented on Equation 3.6.

$$CF = \frac{Energy_{produced}}{\Delta t \times PowerCapacity} \tag{3.6}$$

The other meaningfull metric in this context is the overcapacity, that is, the ratio of the energy that could have been produced, over the energy actually produced.

$$overcapacity = \frac{\Delta t \times PowerCapacity - Energy_{produced}}{Energy_{produced}} \tag{3.7}$$

An estimate of the overcapacity of dipachable RES is provided in [19], the resulting capacity factor evolution is depicted as a function of the VRES share on Figure 3.4. It can be observed that capacity factor decreases quadratically in the VRES share in the electricity mix.

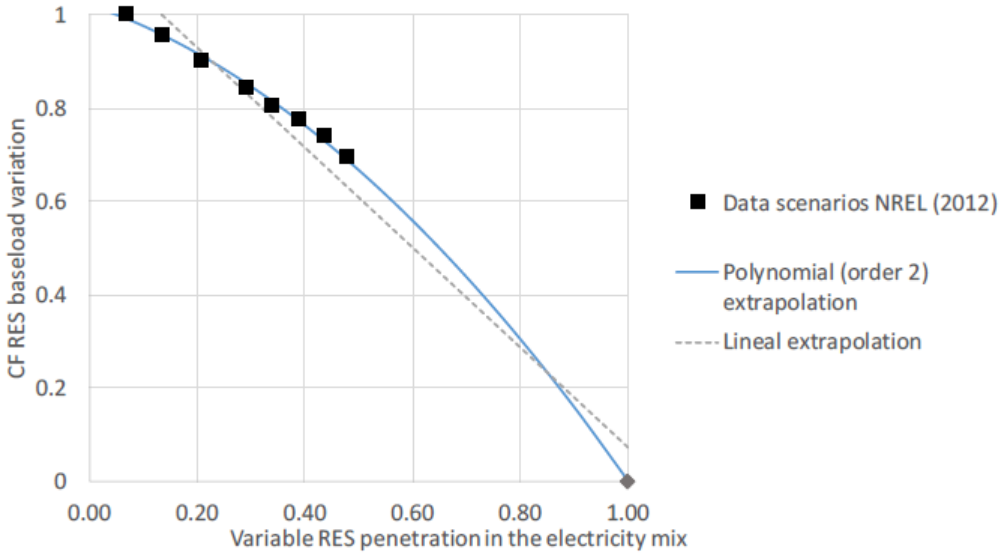


Figure 3.4: Capacity factor of RES evolution depending on the VRES share [19]

3.5.5 VRES plants

MEDEAS based its estimates of the VRES induced overcapacities on [8]. These two main impacts of the VRES share are taken into account:

– The exponential growth of VRES overcapacities and

– The decrease of the VRES capacity factor.

The two estimate functions used in MEDEAS [3] from [8] are presented on Figure 3.5.

The capacity factor is evaluated as a function of the overcapacity, following Equation 3.8, that can actually be derived from Equations 3.6 and 3.7.

$$CF = \frac{1}{1 + overcapacity} \tag{3.8}$$

3.6 Scenarios

As no model can predict with certainty, neither with significant confidence, the decision that will be made and will rule the society for the future, several sets of hypothesis, called scenarios, are made.

When running the MEDEAS model, these scenarios are run in parallel, as in Vensim these are present as possible values for the *scenario* subscript. These possible value have to be chosen in the dedicated panel in Vensim.

By default, the three following scenarios are available, but it is possible to run user-defined scenarios [17].

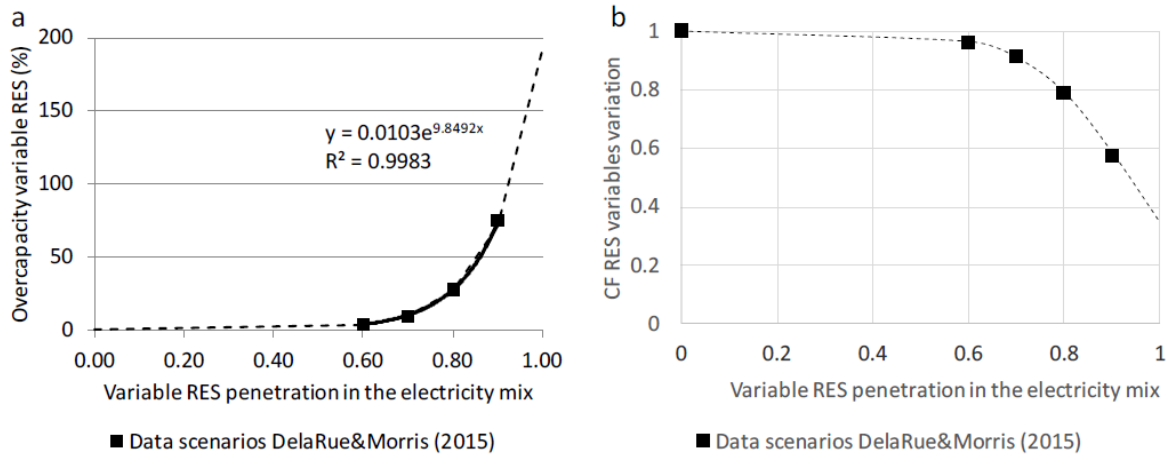


Figure 3.5: (a) Overcapacity estimate and (b) CF reduction of VRES plants on [3]

- **BAU** (Business As Usual) corresponds to no particular effort being implemented, the transition continues as it is.
- **OT** (Optimal Transition) where all the resources available are allocated to the best renewable transition possible, that has become a social priority. The only constraints for faster transition are physical limitations.
- **MLT** (Mid-Level Transition) is a mix of the previous ones, some effort are made but not all. Actions towards renewable transition are delayed.

A depiction of these scenarios in terms of emissions over time is given on Figure 3.6.

These scenarios have been defined somewhat arbitrarily, but the custom scenario capability opens the door to anyone willing to use a more precise, more accurate scenario, given for example additional information brought by major events in the future.

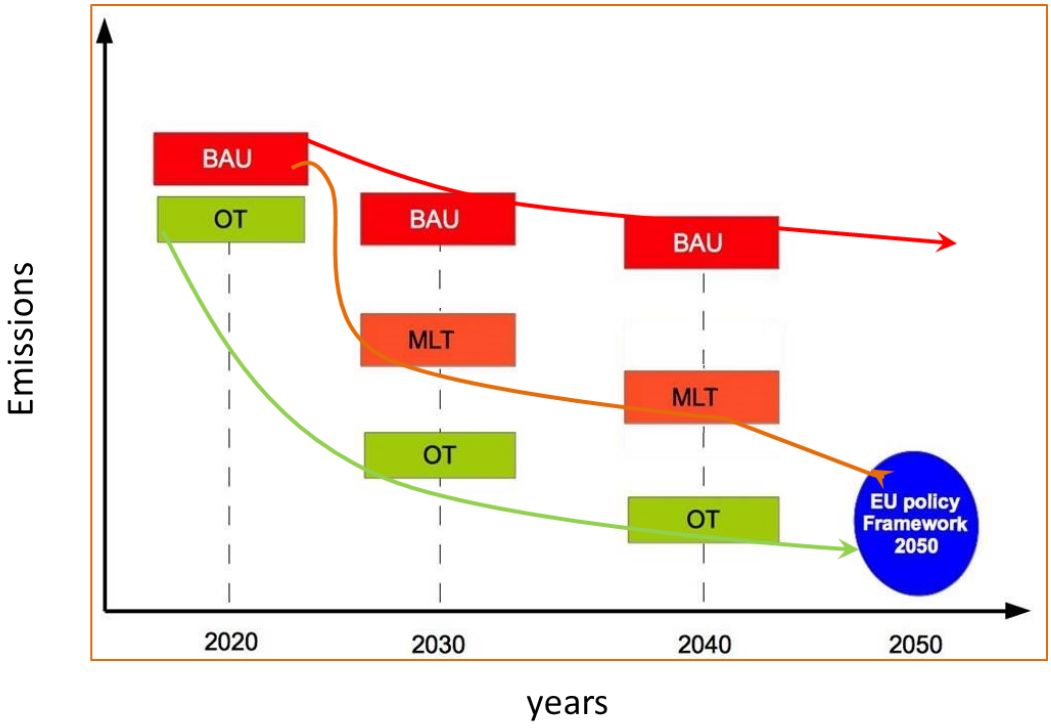


Figure 3.6: Qualitative illustration of the BAU, MLT and OT scenarios for the greenhouse gas emissions in Europe

4 Dispa-SET simulation and database generation

4.1 Overview

This section aims at describing the process that lead to the creation of the dataset, required in order to train the neural network model.

First, an input space is defined, on which will properly select points to form the dataset features. Then, computationally expensive simulation will be run on these points, and meaningful data will be extracted from the simulation results to obtain the desired output features to be predicted by the model.

This dataset will afterwards be used to train the surrogate model on [33].

4.2 Preparatory work

As stated earlier, this setting only considers the european power system in Dispa-SET, then to create and validate our surrogate model. Each simulation is run over a period of 2019.

4.2.1 Unit groupings

In this context, the precise technology and fuel types of each plant is not relevant, as they won't influence the input features of our dataset. Hence, the units are grouped into five categories: flexible units, slow units, storage units, PV units and wind units.

IRENA [25] describes flexible units as "units that can ramp up and down quickly, have a low minimum operating level and fast start-up and shutdown times", what criterion will be used to separate regular units into the slow and flexible units. This criterion is presented in Table 4.1.

Units	Fuel	Condition
$Flex_{units}$	GAS, HRD, OIL, BIO, LIG, PEA, NUC, GEO	$PartLoadMin < 0.5$ and $TimeUpMin < 5$ and $RampUpRate > 0.01$
$Slow_{units}$		$PartLoadMin \geq 0.5$ or $TimeUpMin \geq 5$ or $RampUpRate \leq 0.01$

Table 4.1: Flexible and slow units classification criterion

Refer to Tables 2.2 and 2.3 for their names.

One also has to consider the limit to the number of hydroelectric units that are possible to build given a geographical area. Given that EU is already almost at saturation, stationary batteries are considered, among other energy storage technology (e.g. compressed air, electric vehicles' battery grid).

As for the other groups, they can be simply described with technology-fuel pairs as follows:

- $Storage_{units}$ with (OTH, BATS)
- PV_{units} with (SUN, PHOT)
- $Wind_{units}$ with either (WIN, WTON) or (WIN, WTOF), the latter not being considered in this work

4.2.2 Parameters estimates

The availability factors of PHOT and WTON are also required, as well as the peak load. These values are given as time-series inputs to the simulations, for the year 2019. To give an order of magnitude, their averages are given in Table 4.2.

Variable	Value	Units
AF_{PV}	0.1381	[·]
AF_{WTON}	0.2007	[·]
AF_{WTOF}	0.1050	[·]
$PeakLoad$	440929	MW

Table 4.2: Average values for availability factor over the whole timeseries and all zones, and maximum total demand over the timeserie, i.e. sum of the demand in each country

4.3 Design space

4.3.1 Shape

The first necessary step in order to select our data points for our dataset, is to define the space in which we will sample them. In our case, this space will be the product of 6 ranges, that is a 6 dimensional hypercube.

One may argue that some areas of this hypercube, typically around the vertices, will be extremely unlikely to happen in a real setting. More precisely, as this cube will be the input space of the surrogate model that will be connected to another model, it may be suitable to prune the areas of the cube that will never be reached. Indeed, if we know that some areas will never be queried, there is no use covering them.

Furthermore, assuming we would obtain the exact space of possible queries, this space is not likely to be close to some common shape (hypercube, hyperball or combination). Given that most of the design of experiments techniques assume these kinds of space, a mapping would be needed to benefit from the better sampling strategies. Such a mapping would be pretty complex to develop.

More importantly, the cost of being more general than strictly required is small, mainly consisting of a slightly larger surrogate model (in this specific case, a larger neural network), and a larger dataset.

For these reasons, an hypercube will be used.

4.3.2 Input variables

The six adimensional variables, corresponding to a dimension, are described below, with their given range.

The notation $PowerCap_x$ refers to the maximum power output of all the units in x. See Table 4.2 for the values of the AF and peak load value.

1. **CapacityRatio [·]**

   Ratio of the maximum production over the maximum demand.

$$CapacityRatio = \frac{PowerCap_{flexunits} + PowerCap_{slowunits} + PowerCap_{storageunits}}{PeakLoad} \tag{4.1}$$

2. **ShareFlexibility [·]**

   Share of the units that are flexible.

$$Share_{flex} = \frac{PowerCap_{flexunits}}{PowerCap_{flexunits} + PowerCap_{slowunits}} \tag{4.2}$$

3. **ShareStorage [·]**

   Ratio of the maximum power output of all storage units over the maximum demand.

$$Share_{storage} = \frac{PowerCap_{storageunits}}{PeakLoad} \tag{4.3}$$

4. **ShareWind [·]**

   Ratio of the maximum power output of all wind units over the maximum demand.

$$Share_{wind} = \frac{PowerCap_{windunits}}{PeakLoad} \cdot AF_{WTON} \tag{4.4}$$

5. **SharePV [·]**

   Ratio of the maximum power output of all PV units over the maximum demand.

$$Share_{PV} = \frac{PowerCap_{PVunits}}{PeakLoad} \cdot AF_{PV} \tag{4.5}$$

6. **rNTC [·]**

   Net transfer capacity ratio. This variable is a measure of the grid effect on the network, as the zones are able to transmit power between them.

   The data we are provided contains hourly logs of the power transmitted between each pair of zones. The following describes how to compute the rNTC value given these.

   First, we compute the average net transfer capacity (NTC) for each zone z to any other zone x over each of the N_h hours in the input data, via Equation 4.6.

$$NTC_{z \to x} = \frac{1}{N_h} \sum_h NTC_{z \to x,h} \tag{4.6}$$

   Then Equation 4.7 is used to compute the zonal NTC, that is the ratio of the sum of all NTCs from this zone to any other zones, over the peak load for that zone.

$$NTC_z = \frac{\sum_x NTC_{z \to x}}{PeakLoad_z} \tag{4.7}$$

   A zonal NTC of 1 for zone z thus means that z would be able, at any time, to fulfill the integrity of its demand by importing electricity from connected zones.

   The final rNTC value is a weighed sum of the zonal NTCs. The weight for a zone z is computed as the ratio of its peak load over the sum of each peak loads. This is expressed by Equation 4.8.

$$rNTC = \sum_z \frac{PeakLoad_z}{\sum_x PeakLoad_x} NTC_z \tag{4.8}$$

4.3.3 Output variable

The target outputs of the systems are the curtailment and the shedding. In order to make these values more scalable, we normalize them, by the maximum RES generation that could be produced (that is, the sum of the availability factors multiplied by the power capacity for each units), and the total demand respectively.

The following convention is used: A_f refers to the yourly availability factor, while C_f denotes the yearly capacity factor. We have: $C_f = \frac{\sum_{timesteps} A_f}{N_{timesteps}}$.

1. **Curtailment**

   Percentage of the curtailment to the maximum RES generation from all units:

   $$Curtailment = 100 \times \frac{EnergyCurtailed}{8760 \sum_{units} C_f * PowerCap_u} \tag{4.9}$$

   Where $EnergyCurtailed$ represents the total amount of energy from VRES in MWh for all zones considered, C_f is the per unit annual capacity factor, and $PowerCap$ is the installed capacity for each unit in MW.

2. **LoadShedding**

   Percentage of the shedding to the total demand, that is, the part of the demand that has not been satisfied:

   $$LoadShedding = 100 \times \frac{\sum ShedLoad}{\sum Demand} \tag{4.10}$$

4.3.4 Reference values and ranges

Given the 2019 input data, a reference simulation is run and one obtains the values presented in table 4.3.

Variable	Value	Lower bound	Upper bound
CapacityRatio	1.658	0.5	1.8
ShareFlexibility	0.418	0.01	0.99
ShareStorage	0.497	0	0.5
ShareWind	0.106	0	0.5
SharePV	0.035	0	0.5
rNTC	0.282	0	0.7

Table 4.3: Values of the different variable for the reference simulation in 2019.

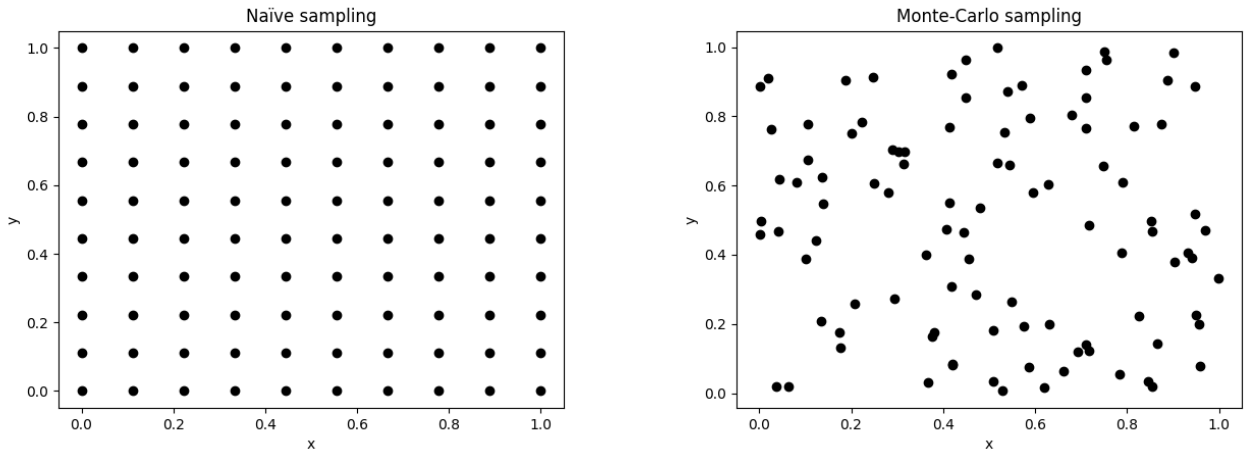
4.4 Design of experiments

A strategy to choose sampling points from some design space is called a design of experiment (DoE). It aims at producing a set of samples that represent as best as possible the entire design space, a property that is required to obtain a well balanced dataset.

The main methods to acheive such sampling are [18] illustrated in the following.

1. The "naïve" sampling: take samples at regular intervals on the design space. Note that one may not choose the same intervals for different dimensions, and we have to set the strategy around the boundaries. It is depicted on Figure 4.1a.
2. The Monte-Carlo sampling: pick samples at random all over the design space. It is depicted on Figure 4.1b.
3. The Latin-hypercube sampling [31], maximizing a criterion that is either [9]:
   (a) centering samples in sampling intervals
   (b) maximizing the minimum distance between two samples
   (c) maximizing the minimum distance between two samples, but place sample in a random location in its interval
   (d) minimizing the maximum correlation between two samples

   These are depicted on Figures 4.2a, 4.2b, 4.2c and 4.2d.



(a) Naïve strategy

(b) Monte-Carlo strategy

Figure 4.1: Basic sampling strategies

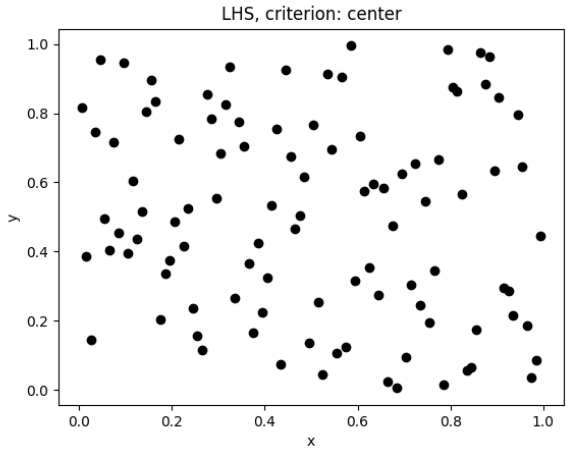
From these 2-dimensional illustration it is clear that the latin hypercube sampling performs best, the naive sampling featuring too much regularities that is not wanted, as they may introduce some bias, and Monte-Carlo sampling tends to make more clusters of samples, that would be inefficient (indeed, making two times the same simulation is useless).

The number of points will be chosen quite arbitrarily to 2000. This leads to an average of $\sqrt[6]{2000} = 3.550$ points per dimension, if the points had been placed on a 6-dimensional grid in the input space.

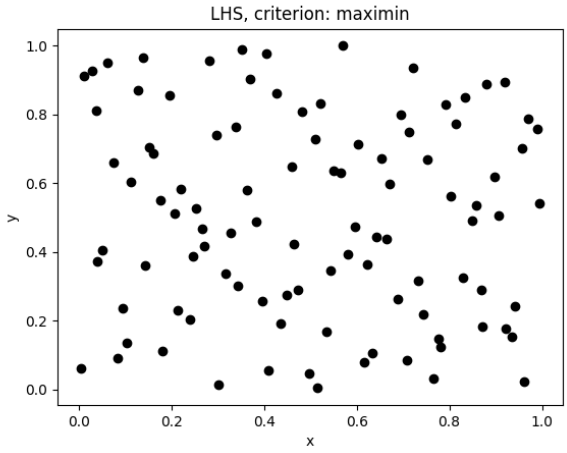
4.5 Generation of the dataset

With the input samples now obtained, the next step is now to compute the simulations on each of these points.

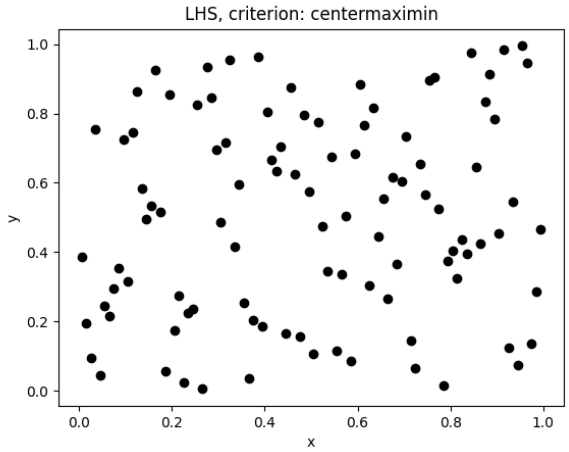
But this task is not trivial since the data that corresponds to these exact configuration is not available. It is thus needed to craft some new simulation settings given a reference, that is the year 2019.



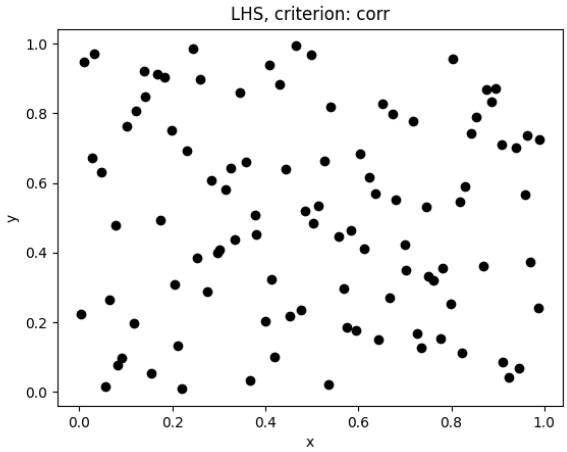
(a) LHS with center criterion



(b) LHS with max min distance criterion



(c) LHS with max min distance and center criterion



(d) LHS with correlation criterion

Figure 4.2: Latin hypercube sampling strategies with every criterion

4.5.1 Adjusting functions

To do so, Dispa-SET disposes of utility functions that do just that, adjusting the capacities, i.e., the power ouputs of the units, in function of some parameters.

These adjusting functions were specifically developped for this topic by Vidal in her master thesis [29].

– `adjust_flexibility` modifies installed capacities to reach the desired $Share_{flex}$.

To do so, it first computes the target capacity, by multiplying the total capacity by the desired $Share_{flex}$. It then add or subtracts the missing or exceeding flexible unit power capacity to each zone, weighting by their total capacity. Equation 4.11 provides a good approximation of the update, but the full algorithm is presented in Algorithm 1.

$$PowerCap_{z,new} = PowerCap_{z,old} + \frac{PowerCap_{z,old}}{\sum_x PowerCap_{x,old}}(target - actual) \tag{4.11}$$

Algorithm 1 Adjust_flexibility algorithm

if $\delta > 0$ **then**:
  $remain \leftarrow \delta$
  for $z \in zones$ **do**
    $weight \leftarrow \frac{total_z}{current\_total\_cap - cum\_sum_z + total_z}$
    $added\_cap \leftarrow min(weight \cdot remain, total_z - flex_z)$
    $new\_flex\_cap_z \leftarrow flex_z + added\_cap$
    $new\_slow\_cap_z \leftarrow slow_z - added\_cap$
  end for
else if $\delta < 0$ **then**:
  $remain \leftarrow -\delta$
  for $z \in zones$ **do**
    $weight \leftarrow \frac{total_z}{current\_total\_cap - cum\_sum_z + total_z}$
    $removed\_cap \leftarrow min(weight \cdot remain, flex_z)$
    $new\_flex\_cap_z \leftarrow flex_z - removed\_cap$
    $new\_slow\_cap_z \leftarrow slow_z + removed\_cap$
  end for
else
  $new\_flex\_cap_z \leftarrow flex_z$
  $new\_slow\_cap_z \leftarrow slow_z$
end if

– `adjust_capacity` applies a linear scaling to the power output of some given set of units, in particular, it will be called multiples times to adjust the storage, PV and wind capacities.

Scaling factors applied are summarized in Table 4.4.

Units	Scaling factor
$Storage_{units}$	$Share_{storage}$
$Wind_{units}$	$\frac{CapacityRatio \cdot Share_{wind}}{AF_{wton}}$
PV_{units}	$\frac{CapacityRatio \cdot Share_{PV}}{AF_{PV}}$

Table 4.4: Scaling factors applied to different units

– `adjust_rntc` applies a linear scaling to each zonal NTC time series.

4.5.2 Extracted outputs

As every simulation runs outputs a lot more than only the curtailment and lost load values, some other outputs variables will be extracted from the simulations, while not directly useful for the time being.

Of course, these may reveal themselves useful for future work.

All the outputs extracted from the simulations are displayed in Table 4.5.

Distinction should be made between the *Shedding* and the *LostLoad*. The former is a consequence of voluntary action, following set rules between consumers and producers. The latter is a consequence of additional variables added to Dispa-SET to reduce the infeasibilities, assigning a

Parameter	Unit	Parameter	Unit
Cost	€/MWh	Shedding	TWh
Congestion	h	LostLoad	TWh
PeakLoad	MW	CF gas	[·]
MaxCurtailment	MW	CF nuc	[·]
MaxLoadShedding	MW	CF wat	[·]
Demand	TWh	CF win	[·]
NetImports	TWh	CF sun	[·]
Curtailment	TWh		

Table 4.5: Values extracted from each simulations

high cost to the capacity the system is not able to generate due to maximum capacity and ramping constraint being reached.

4.5.3 Dataset creation

With the help of the adjusting functions, creating a whole dataset now comes down to generate samples from LHS as previously discussed, adjusting the reference simulation setting to each sample, run the simulation and extract the desired features from the outputs.

As this whole process will be completely automated, it is pretty easy to obtain a second dataset, based on a different LHS. In particular, a smaller dataset will be of interest for the validation and testing stages of the surrogate model, as will be elaborated in subsection 5.3.1.

4.6 Implementation

One simple, yet important notice: running all of these simulation is not feasible on a basic hardware. A single MILP simulation typically takes around two hours to complete on the cluster, as the latin hypercube sampling targetted 2000 simulations, yielding to 4000 hours or 166.6 days if the simulation are run sequentially. Moreover, the testing phase will also require a lot of runs. This arises the need for the cluster use, and thus of submitting these as jobs on the cluster.

As the NIC5 cluster provided by CÉCI is obviously shared, one needs to manage the submitted jobs appropriately. In our case, we simply have the same program to be run a bunch of times, that are jobs independent of each other.

4.6.1 Steps

For a complete experiment to be completed, these steps have to be followed:

1. Generating the reference simulation, to extract the data that will be manipulated by the adjusting function
2. For each sample, do:
   (a) Call the adjusting function and create the simulation directory, with all the simulation input data
   (b) Call GAMS in this simulation directory
   (c) Fetch GAMS outputs in this directory

4.6.2 Scripts and code

All the files for this section lie in the `data-generation` folder.

The steps presented above almost map to a script or function written. The flow of the dataset generation is as described here.

1. The SLURM script `main.sh` is submitted on the cluster. It fetches relevant data in the `config.py` script.
2. It submits the generation of the reference simulation as another job on the cluster and waits for its completion.
3. It calls `sampling.py` with argument `-sample-only`, that will create the `samples.csv` file containing all the samples.
4. It prepares the file `dataset.csv` by writing its header line.
5. It finally runs the bash script `launch-job-series.sh`, with serie index 0, that will submit some fixed number of sample jobs, as an array, through the SLURM script `launch-simulation-jobs.sh`.
6. Each sample job runs:
   (a) The simulation directory is prepared by calling the `sampling.py` script with argument `-prepare-one` and the index of this simulation (it reads the corresponding line in `samples.csv`).
   (b) GAMS is called on the simulation directory.
   (c) The simulation results are read with `read_results.py -single`, then outputted to `dataset.csv`.
   (d) The simulation directory is cleaned.

This list is not exhaustive, see the Annex for a complete version.

4.6.3 Technical aspects

During the creation of this set of scripts, some technical details require specific attention:

– The GAMS solver has options to set the number of threads to do the computations, that number of threads must not exceed the number of CPUs allocated by SLURM for that job, otherwise it will disturb the whole node on which it is running, including unrelated jobs.

  The thread setting in the `UCM_h.gms` file generated by Dispa-SET is set to 16 by default, this line may be removed if one wants to set it manually through the command line, as the file input will take precedence over the command-line value set.

– The total amount of each prepared simulation directory is too large to fit on the allocated disk memory on the `$HOME` partition on the cluster. Moreover, for efficiency it is better to use the `$GLOBALSCRATCH` file system.

– The Dispa-SET ajdusting function do not write adjusted data to a directory if this directory already exists before the function is called. That is, the directory containing the simulation files should not be created beforehand.

– Due to SLURM limiting the maximum number of jobs a user can have in the submit queue, queuing all the simulation jobs at once is not possible. Moreover, the number of simulation appeared to cause issues with the SLURM maximum array size.

  This is the reason why `launch-job-series.sh` exists. The script takes as an argument the serie index i, then starts a $[0:399]$ array of `launch-simulation-jobs.sh` jobs, with i as an argument. As the latter script has also access to its array index via SLURM environment variables, it can compute the intended simulation index, that will cover the $[400i:400i+399]$ range. The next series is queued with the job array as a dependency, so that it is not launched before the current series terminated.

4.6.4 Unsuccessful simulations

During the execution of the simulation, it appeared that for some of them GAMS was completely stuck at some point, hence wasting resources and preventing the other simulations to start by keeping available CPUs busy.

A timeout on the GAMS simulation has therefore been set, after looking at the typical duration of a "regular" simulation. The simulation that were timed out had thus the corresponding error message in their output, that can be checked when reading the results.

Hence, we can label the samples whose simulation failed with an additional error field in the dataset, and compare them to the other. Typically, around 10% of the simulation fail, in this case 218 over 2400 total.

Loading this dataset and separating the successful simulations from failing ones, and extracting meaningful metrics from both, it appeared that the failing simulations featured an average share flexibility significantly higher than the other, while the other input parameters seemed around the same. The share of flexible generation is taken in the $[0.01, 0.99]$ interval, and the average of failing simulations is around 0.49 higher than the average of the successful ones.

To further validate our observation, the quantiles of the distribution of the share of flexible generation in the stalling settings are explored, and it appears that they lie really high, the first one (25%) is around 0.9, the median lies near 0.95 and the third one (75%) is around 0.96.

More decisively, the maximum value of a successful simulation is 0.9011, while the minimum value of a stalling simulation is 0.9015, that is, slightly more than the max of valid simulations. Thus, there exists a net boundary between the simulation that run successfully and the other.

The distribution of $ShareFlex$ values in both part of the dataset is depicted as boxplots on Figure 4.3

As a consequence, the validity region for the surrogate model has to be updated, as there is no way of producing an estimate of the outputs on a region where no sample was simulated correctly.

This issue interpreted as the simulations having too high flexibility are harder to solve than the other, as the system has a lot of options to choose from when confronted to a change in demand.

It is noteworthy that in about 30% of the simulation, a division by zero error was also stated. Since this happens in the preprocessing, it has no influence on the results and this error is discarded.

It would also be useful to find in which equation this happens for debugging purposes.

4.6.5 Dataset fields

For completeness, all the fields in the created dataset in `dataset.csv` are shown on Table 4.6.

Only two of these, *LoadShedding* and *Curtailment* are actually used to train the surrogate model, and six as the model features. As the marginal cost of including all the other outputs is

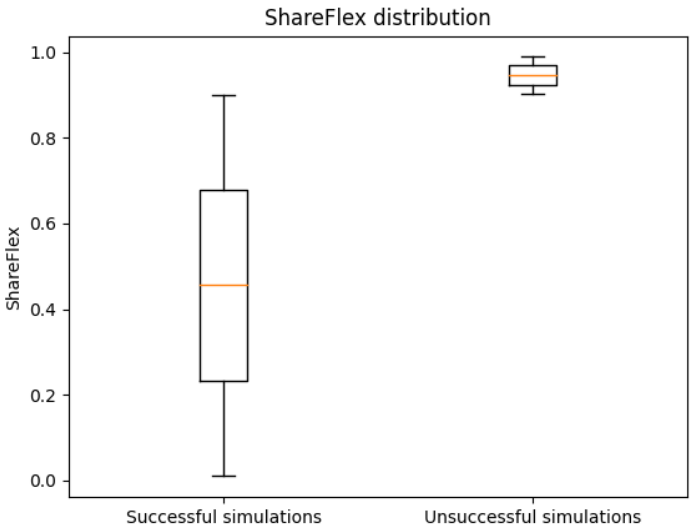


Figure 4.3: Distributions of the *ShareFlex* parameter value in the unsuccessful simulations against successful simulation. The distributions of the other parameters are similar.

null, they are saved as well in case someone would find a use for it in the future.

[1]The MaxLoadSheddingShare is taken as a fraction of the demand at the time.

Parameter	Unit	Parameter	Unit
Cost	€/MWh	Curtailment	%
Congestion	h	MaxLoadSheddingShare[1]	%
PeakLoad	MW	CF gas	[·]
MaxCurtailment	MW	CF nuc	[·]
MaxLoadShedding	MW	CF wat	[·]
Demand	TWh	CF win	[·]
NetImports	TWh	CF sun	[·]
Curtailment	TWh	Capacity ratio	[·]
Shedding	TWh	Share flex	[·]
LostLoad	TWh	Share sto	[·]
MaxRESGeneration	MW	Share Wind	[·]
TotalGeneration	TWh	Share PV	[·]
ShareRESGeneration	%	rNTC	[·]
LoadShedding	[·]		

Table 4.6: Dataset fields. "%" as a unit means a unitless ratio multiplied by 100. Elements displayed in green correspond to the six features, and elements in blue the two target outputs.

5 The surrogate model

5.1 Overview

This section presents the training process that lead to the description and definition of the target surrogate model, given a ready to use dataset, and the desired characteristics.

This is a straightforward example of a regression problems, that is to predict some output features from the input features, given a number of training example, that is the dataset. This is a typical machine learning problem, and is be tackled as such.

First, different methods' performances are measured and the best one is chosen. Then, a model is trained and evaluated, that is the core surrogate model.

5.2 Machine learning methods

In the following, some of the most common machine learning algorithms for regression problems [10] are assessed.

5.2.1 K nearest neighbors

In this setting, the K-nearest neighbors (k-NN) methods would be one of the easiest to implement, using for example the scikit-learn [24] module in python. Indeed, these simply require a single parameter value k and output the average value of the output features of the k nearest neighbors, computing a distance on the input features.

This will come with the drawback of not being able to learn quick variations in the outputs, as the averaging over the nearest sample points will filter out high frequency variations.

Results obtained with k-NN for several values of k are shown on Table 5.1.

k	Validation error
4	0.05082837
5	0.051765025
6	0.047083396
7	0.046183977
8	0.049501333
9	0.049218103
10	0.04949624
11	0.05036374
12	0.050881956
13	0.05243453
14	0.052941263
15	0.05362743
16	0.05440363

Table 5.1: Results obtained with k-NN

5.2.2 Decision trees

Decision trees, and extremely randomized trees [21] another category of easy to implement[2] machine learning methods, as they also come with a limited, fixed number design parameters.

But these also come with the drawback of having a piecewise constant output, and as stated above, it is required to be able to represent significantly fast variations in the output. Although it is possible to mimic with many successive steps, being peicewise constant will now introduce non linearities in the outputs, that is, unwanted steps in the prediction.

Results obtained using decision trees are shown on Table 5.2.

Number of trees	Validation error
25	0.02385
35	0.02347
45	0.02275
55	0.02342
65	0.02361
75	0.02272
85	0.02360
95	0.02255
105	0.02294
115	0.02379
125	0.02263
135	0.02232
145	0.02317

Table 5.2: Results obtained with random forests

These performances are better than the nearest neighbors method, but do not perform as good as the following techniques.

5.2.3 XGBoost

XGBoost is one of the most popular machine learning technique nowadays [4]. XGBoost in itself is actually an efficient implementation of a machine learning method, tree boosting.

Boosting aims at building a good predictive model out of so-called weak learners, in this case, trees. Each learner is trained on the error of the output of the previous model, so that the previous output plus the newly trained learner output is a better prediction of the target output. Moreover, the samples are weighted in favor of the ones that were badly predicted by the previous model, in order to make sure we correct past mistakes.

In tree boosting, the main hyper-parameters are:

- the number n of weak learners stacked
- the *learning rate*, a constant factor applied to the target difference (target output minus previous model output)
- the maximum depth of the weak learners

[2]Again, using scikit-learn

Resulting performance for some values of the hyper-parameters are reported on Table 5.3.

XGBoost's results are really good, down to 0.01 mean squared error on the validation set. This is not surprising given that this technique is quite popular in the literature. Actually, for the first testing stage where the dataset created in this work was not available, the dataset from C. Vidal's work has been used as a first test. In this setting, the results obtained with XGBoost outperformed, even though by only a little, the best neural network architecture.

5.2.4 Kernel-based methods

One may consider to apply a kernel method with one of the other method mentioned above. While this may indeed improve the quality of the resulting model, and also solve the issue that was representing fast changes in the output, there is a main, almost trivial issue with them. Obviously, one need a kernel, but what kernel?

As there are no ready-to-use kernel for this specific case, and that finding such a kernel would be a hard task, kernel methods are abandonned.

5.2.5 Artificial neural network

Artificial neural networks (ANN) are used in this work to build our surrogate model. These kind of methods provide a large flexibility, due to their entirely customizable architecture, as well as a large learning capacity. This makes them able to modelize with good accuracy some complex, non-linear functions.

In most recent applications of these ANNs, a lot of different strategies are used to process the data efficiently. For example, convolutional layers convey a lot of meaning in the context of image processing, or transformers are well suited to process sequences [14].

In this case, the inputs boils down to the 6 variables values, listed in Table 4.3. There are no patterns in this data, because even if their actual values were correlated in some way, the simulations dataset we have as an input at this stages has its input features drawn from a latin hypercube sampling, meaning they have a fixed, very low correlation. This correlation originates from the fact that the sampling aims at optimizing the design space coverage, not from a meaningful, exploitable source.

Thus, a simple multi-layer perceptron (MLP) architecture is chosen, and the next requirement is to describe the characteristics of that MLP, that are:

- The number of layers
- The numbers of neurons in each layers
- The activation function at each layer

Some experiments are run to provide some baselines for getting a first approximation of what performs well. These are presented on Table 5.4.

These result outperform every other method. The fact that different but close architecture show similar performance brings some confidence about the reproducibility: the random initialization of the weights in the network could have led to an exceptionally good model. Moreover, several runs of the same architecture yield similar ouputs, validating this intuition.

It is worth mentioning that some larger network architecture, *e.g.* $(300,'relu',0)$, $(300,'relu',0)$, $(200,'relu',0)$ acheived really good results, even slightly better than the best one. However these

were not implementing dropout, so that they are high changes of being subject to overfitting. If dropout is added, the observed performance decrease, confirming the overfitting hypothesis.

This is why the relative smallness of the network is a strength, as it will also act against overfitting, as having fewer parameters leaves less room for learning noise.

5.2.6 Selection of the machine learning technique and parametrization

In the end, neural networks are opted for. They benefit from the best performance in terms of mean squared error on the validation test, but neural networks MLPs also present the following advantages:

- They are relatively lightweight, in comparison to the K-nearest neighbors methods, that needs to store the entire dataset, and the randomized trees, that need to store its trees structures. Neural networks only needs their weights that are fairly small with this few input variables.

- Grasp non-linear behaviour with accuracy

Preliminary experiments basically consist in the test undertaken to assess the performance of different architectures (reported on Table 5.4). While the firsts steps of testing used another dataset that available at the time, ultimately the generated dataset was the only one in use for this assessments.

The architecture is chosen to be the best of the baselines, and is provided in Table 5.5. While hyper-parameter tuning may have produced another result, the latter could have been subject to overfitting, as described above.

The final parameterization of the model is therefore the setting of all its weights, that lie in layers 1, 4 and 7. The total count of parameters adds up: $1080 + 18000 + 200 = 19280$.

5.3 Machine learning aspects

5.3.1 Validation and testing

In machine learning, validation and testing are crucial steps in order to ensure the performance of a model. To implement these, one has to separate the data into three sets.

- The validation set is used to tune the model hyperparameters on. Due to its influence on the model, it is thus susceptible to bias the model.

- The test set is used to evaluate the model's performance, on unseen data.

- The training set is used to train the model.

In most cases, one will simply split available data into three subsets. However, in this setting, the training, input data comes from a latin hypercube, and can be generated. Taking this into account, one may then consider the use of different LHS.

For the training set, joining different sets drawn with LHS is not expected to improve performance. As the LHS are independent of each other, there is a great chance to draw samples that are really close to each other, or worse, equal, what is exactly what LHS aims at avoiding.

On the other hand, using a set drawn from a different LHS (than the one used for the training set) for testing and validation is interesting. This will provide data that spans the whole input

space, but not yet exactly the same as the data used for training. Thanks to the input space coverage, the evaluation made at testing will be more extensive, and thanks to being independent of the training data, this evaluation should remain unbiased.

On the other hand, the validation set being sampled from a LHS now providing a guarantee that it spans the whole space could make the work of the learning agent easier, giving it access to each of the LHS sample from the training set.

However promising, this approach will not be done in this work, to make sure the unbiased property of the trained model persists, and that the term "validation error" keeps the same meaning as everywhere else.

5.3.2 Overfitting

An inherent problem to machine learning methods is the overfitting, or its opposite, underfitting. These terms refer to the cases where the training is respectively too specific to the training data, and not enough specific.

Overfitting is thus a symptom of too much learning, leading to learning some noise or some particularities of the dataset, while underfitting means that there is not enough learning, so that the model is not able to represent all the cases, even the one that are well represented in the available data.

The main tool used to prevent it in a neural network is the dropout. During the training phases, each neuron on a layer will have some probability p, typically between 0.3 and 0.5, of being set to 0, independently of its value. This methods ensures that the network will not be excessively relying on some neurons in its output. During testing oviously, the dropout is removed and all neurons are functional.

5.3.3 Underfitting

On the other hand, underfitting means there is not enough learning happening. For example, a neural network with only 2 neurons corresponding to the ouputs could not learn any non-linear functions, with some tweaks due to the eventual activation function.

In this setting, although underfitting is still a shortfall, it is not regarded as bad as the overfitting. The latter introduces errors that come from noise that is completely random, inevitably introducing imprecisions. But we keep in mind that the target to be approximated, that are outputs from the Dispa-SET model, are themselves an approximation, though accurate, of the reality. So if these are subject to some bias or imperfections, the best model would learn them as well.

Because an underfitting model of the dataset created using Dispa-SET would still be a decent estimate of the reality, what is actually the main goal here, underfitting is preferable to overfitting.

These will have to be assessed during training to ensure the validity of the model.

5.3.4 Bias

An other source of imprecision in machine learning is the bias. This relates to the fact that there exist some noise in the data, that cannot be filtered out, or imprecisions in the assumptions, that inevitably conducts to noise in the output.

However, in this setting, there is very little one could do to reduce its significance. First, the data points have been drawn from a latin-hypercube sampling strategy, that precisely aims at spreading the samples equitably all over the input space. Then, the output features were computed from a Dispa-SET run on this sample.

Thus, the main source of bias one may have an influence in is the bias originating from incorrectnesses during the model training, due to a poor model design.

The other plausible source of bias is the simulation made in Dispa-SET. Of course, Dispa-SET is also itself a model, thus relying on some assumptions and subject to its own modelling of the reality. And as such, it may introduce a bias in its computations, that will necessarily be learned by the surrogate model. But there is no way to assess this bias, and obviouly Dispa-SET itself focuses on making that bias as negligible as possible.

This consideration is of interest, as Dispa-SET has multiple formulations, namely LP and MILP, that then have different bias with respect to reality.

5.4 Training

5.4.1 Implementation

All the files for this section lie in the `nn` folder.

The implementation of the training process comprises the following files:

- `config.py`, that holds all the high-level specifications of the training, such as the names of the outputs, the train-test-validation set ratios, the number of epochs etc.
- `model.py`, that contains the function building the model, thus the definition of the neural network's architecture.
- `baselines.py`, that contains code to train models with pre-defined architectures, in order to quickly and easyly have an overview of the order of magnitude involved.
- `train.py`, that contains the code for the hyperparameter tuning and model training.
- `view.py`, that contains the utilities to visualize the results.
- `tests.py`, that contains the testing of the other machine learning methods (k-NN, trees, XGBoost)
- `data`, `logs`, `models` directories, that contain the datasets, the runs' logs, and the trained models respectively.

5.4.2 Results

The chosen architecture is a two-layer network:

1. 180 neurons, ReLU activation, 0.4 chance of dropout
2. 100 neurons, hyperbolic tangent activation, 0.4 chance of dropout

The mean squared error over the training epochs is shown on Figure 5.1.

Overfitting is not likely, because as can be seen on Figure 5.1, the validation error never increases significantly after a given amount of learning epochs, while the training error is still decreasing. An example of such obvious overfitting is shown on Figure 5.2.

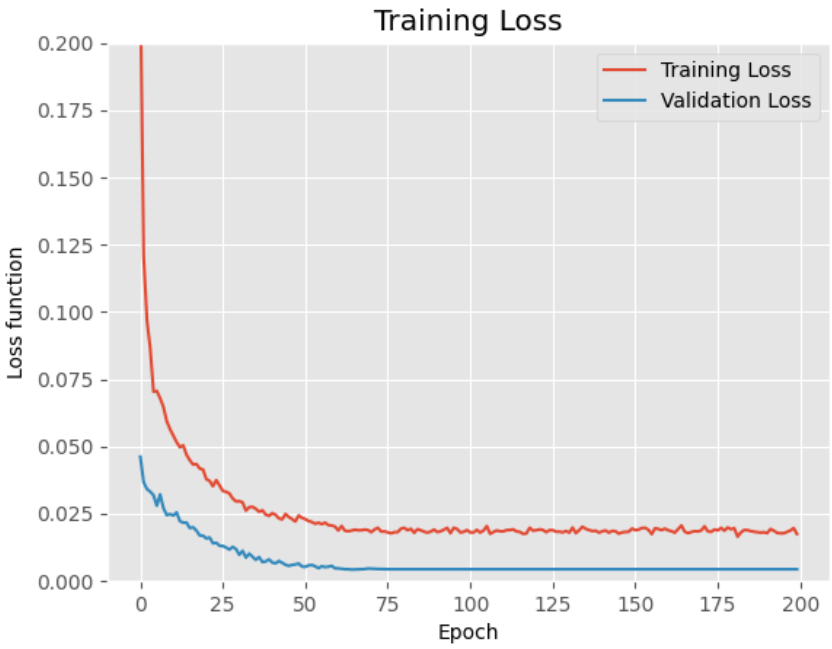


Figure 5.1: Mean squared error on the validation set

5.4.3 Observations

To further confirm the absence of overfitting in this result, the `view.py` file is used to browse across the multi-dimensional function, with surface plots.

These plots draw one of the outputs against two of the inputs, keeping the four other inputs constant. These constant values are summarized in Table 5.6. Attention should be paid to the scale of the plots, as these do not start at 0.

Several surfaces are depicted on Figure 5.3. The following observation are made from the latter illustrations.

- Figure 5.3a follows the natural intuition, as higher shares of storage units and flexible units both contribute to the reduction of the curtailment.
- Figure 5.3b points out the crucial impact of the share of flexible units on the load shedding for small values. Naturally, when peaks in demand appear, if no unit is susceptible to be started, there is no other way than to cut the exceeding demand out.
- Figure 5.3c also confirms the intuition, as it outlines the increase in curtailment with an increase of either share solar or share PV.
- Figure 5.3d is the most intriguing one. First, it is noticed that the scale is the lower here. The most surprising part is that it features local minima, and that when the share PV is close to 0, the load shedding as a function of the share of wind energy is U-shaped. This phenomenon is hard to explain theoretically, the most likely cause remains a weak learning of the model.

  Moreover, the shape of the surface changes rapidly when changing the other constant values (therefore taking other values than disclosed in Table 5.6), validating the hypothesis of a weak learning.

Figure 5.2: An (unrelated) example of clear overfitting, the validation loss increasing after some time

In comparison, the other surfaces do not move that significantly with similar change, coming down to a slight upwards or downwards shift, depending on the direction of the change, occasionnaly with a light amplitude change.

– Figure 5.3e follows the intuition as well, but also highlights the stronger impact of the capacity ratio on the curtailment compared to the rNTC. This is not surprising, as the share of electricity import is not that huge. However, this may change dramatically depending on the country considered.

– Figure 5.3f is interesting, as it shows that the load shedding grows when both the rNTC and capacity ratio lower. As low rNTC means little importation possible, and low capacity ratio not much margin to fulfill the demand, this actually makes sense.

lr	d	n	err	lr	d	n	err
3	0.1	8	0.2446	4	0.4	8	0.0215
3	0.1	9	0.2141	4	0.4	9	0.0198
3	0.1	10	0.1886	4	0.4	10	0.0181
3	0.1	11	0.1672	4	0.4	11	0.0170
3	0.1	12	0.1484	4	0.4	12	0.0163
3	0.1	13	0.1307	4	0.4	13	0.0156
3	0.2	8	0.0865	5	0.1	8	0.1984
3	0.2	9	0.0707	5	0.1	9	0.1681
3	0.2	10	0.0594	5	0.1	10	0.1425
3	0.2	11	0.0509	5	0.1	11	0.1212
3	0.2	12	0.0434	5	0.1	12	0.1035
3	0.2	13	0.0388	5	0.1	13	0.0889
3	0.3	8	0.0430	5	0.2	8	0.0530
3	0.3	9	0.0373	5	0.2	9	0.0414
3	0.3	10	0.0315	5	0.2	10	0.0330
3	0.3	11	0.0276	5	0.2	11	0.0272
3	0.3	12	0.0249	5	0.2	12	0.0228
3	0.3	13	0.0229	5	0.2	13	0.0199
3	0.4	8	0.0313	5	0.3	8	0.0221
3	0.4	9	0.0279	5	0.3	9	0.0185
3	0.4	10	0.0254	5	0.3	10	0.0157
3	0.4	11	0.0234	5	0.3	11	0.0140
3	0.4	12	0.0221	5	0.3	12	0.0125
3	0.4	13	0.0205	5	0.3	13	0.0114
4	0.1	8	0.2133	5	0.4	8	0.0155
4	0.1	9	0.1822	5	0.4	9	0.0138
4	0.1	10	0.1564	5	0.4	10	0.0128
4	0.1	11	0.1359	5	0.4	11	0.0119
4	0.1	12	0.1178	5	0.4	12	0.0113
4	0.1	13	0.1030	5	0.4	13	0.0109
4	0.2	8	0.0626	6	0.1	8	0.1938
4	0.2	9	0.0498	6	0.1	9	0.1620
4	0.2	10	0.0405	6	0.1	10	0.1368
4	0.2	11	0.0344	6	0.1	11	0.1163
4	0.2	12	0.0292	6	0.1	12	0.0990
4	0.2	13	0.0253	6	0.1	13	0.0848
4	0.3	8	0.0281	6	0.2	8	0.0494
4	0.3	9	0.0239	6	0.2	9	0.0379
4	0.3	10	0.0207	6	0.2	10	0.0302
4	0.3	11	0.0183	6	0.2	11	0.0248
4	0.3	12	0.0161	6	0.2	12	0.0211
4	0.3	13	0.0146	6	0.2	13	0.0181

Table 5.3: Results for some values of the XGBoost parameters. The learning rate is labelled "lr", the maximum depth "d" and the number of estimators n, while the validation error is denoted by "err".

Architecture	Validation error
(180, 'relu', 0.4), (100, 'tanh', 0.4)	0.00484
(180, 'relu', 0.4), (100, 'tanh', 0.4)	0.00499
(180, 'relu', 0.4), (100, 'tanh', 0.4)	0.00480
(70, 'relu', 0.5), (70, 'relu', 0.5)	0.04538
(100, 'relu', 0.4), (100, 'relu', 0.4)	0.02506
(100, 'relu', 0.5), (100, 'relu', 0.5)	0.04111
(100, 'relu', 0.6), (100, 'relu', 0.6)	0.04481
(100, 'relu', 0.7), (100, 'relu', 0.7)	0.10410
(80, 'relu', 0.7), (80, 'relu', 0.7)	0.06514
(150, 'relu', 0.6), (100, 'relu', 0.6)	0.02693
(250, 'relu', 0.4), (125, 'relu', 0.4)	0.01295
(200, 'relu', 0.5), (125, 'relu', 0.5)	0.01734
(200, 'relu', 0.5), (125, 'tanh', 0.5)	0.00532
(200, 'relu', 0.4), (125, 'tanh', 0.4)	0.00555
(200, 'relu', 0.4), (100, 'tanh', 0.4)	0.00581
(150, 'relu', 0.45), (100, 'tanh', 0.45)	0.00585
(150, 'relu', 0.5), (100, 'tanh', 0.5)	0.00603
(150, 'relu', 0.4), (80, 'tanh', 0.4)	0.00597
(220, 'relu', 0.5), (125, 'relu', 0.5)	0.01086
(250, 'relu', 0.5), (125, 'relu', 0.5)	0.02421
(250, 'tanh', 0.4), (125, 'tanh', 0.4)	0.04193
(200, 'relu', 0.5), (150, 'relu', 0.5), (100, 'relu', 0.4)	0.07372
(120, 'relu', 0.4), (120, 'relu', 0.4), (120, 'relu', 0.4) (120, 'relu', 0.4), (80, 'relu', 0.4)	0.12163
(50, 'relu', 0.3), (50, 'relu', 0.3), (50, 'relu', 0.3), (50, 'relu', 0.3)	0.08267
(50, 'relu', 0.4), (50, 'relu', 0.4), (50, 'relu', 0.4), (50, 'relu', 0.4)	0.14292
(50, 'relu', 0.2), (50, 'relu', 0.2), (50, 'relu', 0.2)	0.02622
(50, 'relu', 0.2), (50, 'relu', 0.2), (50, 'relu', 0.2), (50, 'relu', 0.2)	0.05221
(50, 'relu', 0.3), (50, 'relu', 0.3), (50, 'relu', 0.3)	0.05723
(50, 'relu', 0.4), (50, 'relu', 0.4), (50, 'relu', 0.4)	0.08706
(150, 'relu', 0.5), (150, 'relu', 0.5), (150, 'relu', 0.5)	0.07769
(150, 'relu', 0.5), (100, 'relu', 0.5), (100, 'relu', 0.5)	0.08809
(150, 'relu', 0.5), (150, 'relu', 0.5), (100, 'relu', 0.5)	0.08098
(200, 'relu', 0.5), (200, 'relu', 0.5), (150, 'relu', 0.5)	0.07682
(190, 'relu', 0.5), (190, 'relu', 0.5), (140, 'relu', 0.5)	0.06489
(200, 'relu', 0.5), (200, 'relu', 0.5), (150, 'relu', 0.5), (30, 'relu', 0.3)	0.08749
(200, 'relu', 0.5), (200, 'relu', 0.5), (150, 'relu', 0.5), (50, 'relu', 0.3)	0.09094
(200, 'relu', 0.5), (200, 'relu', 0.5), (200, 'relu', 0.5)	0.07010
(150, 'relu', 0.5), (150, 'relu', 0.5), (150, 'relu', 0.5), (150, 'relu', 0.5)	0.13490

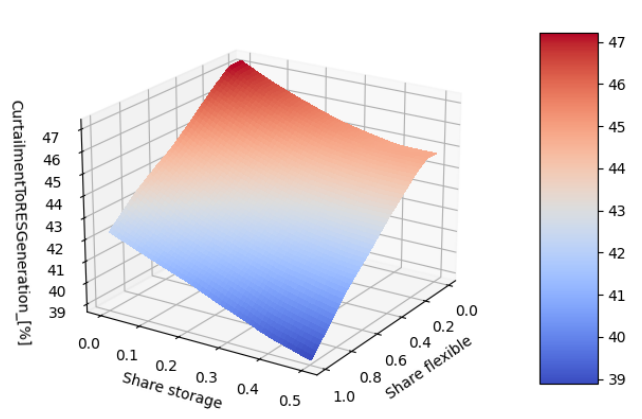
Table 5.4: Results for some architecures of neural networks. Architectures are formatted as a list of layers, which are written as (n, a, p) tuples, where n is the number of neurons, p the dropout value and a the activation function

Layer index	Layer type	Weights count
1	Fully connected linear layer, 6 inputs to 180 outputs	$6 \times 180 = 1080$
2	ReLU activation function	0
3	Dropout layer, with $p = 0.4$	0
4	Fully connected linear layer, 180 inputs to 100 inputs	$180 \times 100 = 18000$
5	tanh activation function	0
6	Dropout layer, with $p = 0.4$	0
7	Fully connected linear layer, 100 inputs to 2 outputs	100×2

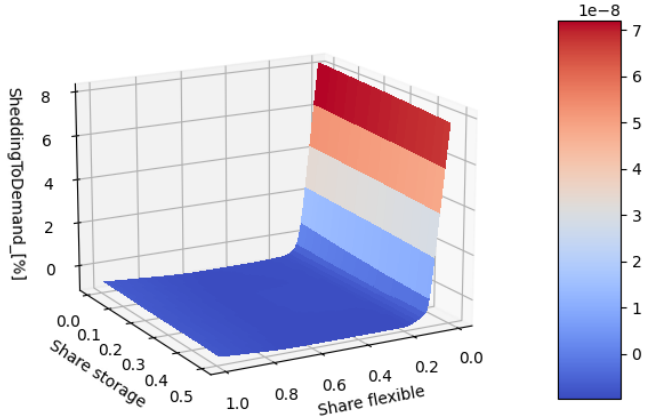
Table 5.5: Description of the layers of in the selected model's architecture

Input name	Value
Capacity ratio	1.15
Share flexibility	0.5
Share storage	0.25
Share wind	0.25
Share PV	0.25
rNTC	0.35

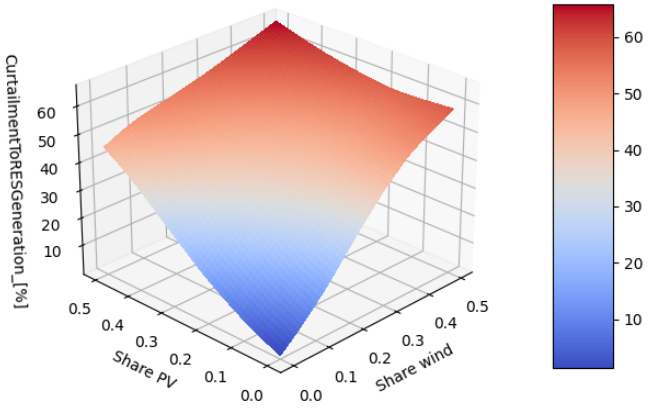
Table 5.6: Default values for constant inputs. These are the middle of their base interval, see Table 4.3



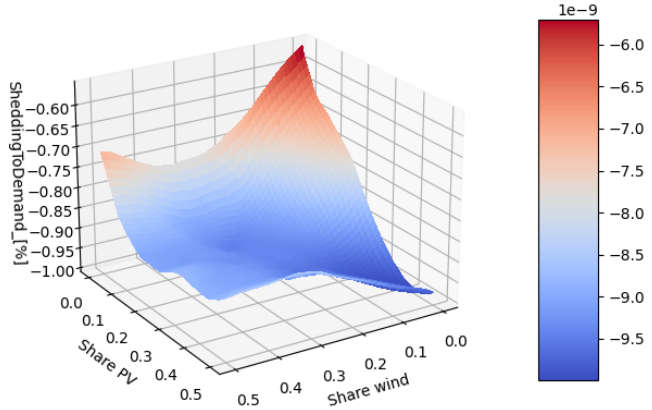
(a) Curtailment against share flexibility and share storage



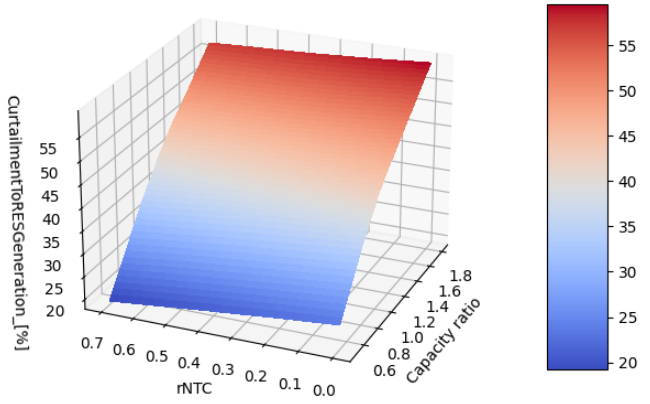
(b) Load shedding against share flexibility and share storage



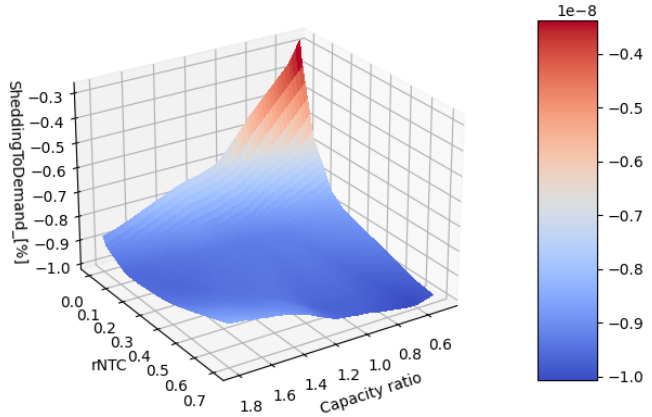
(c) Curtailment against share wind and share PV



(d) Load shedding against share wind and share PV



(e) Curtailment against capacity ratio and rNTC



(f) Load shedding against capacity ratio and rNTC

Figure 5.3: Different views of the multi-dimensional function, with four fixed inputs and two varying inputs.

6 Integration

6.1 Overview

This section describes the process of integrating the trained surrogate model into the MEDEAS model.

The MEDEAS model is built with the Vensim software, and is available to the public as a Vensim model file. However, the python library `pysd` [15], focused on system dynamics simulations, is able to parse such files and run the simulations. The reasons why this will not be taken advantage of is discussed in the first subsection.

The task of integrating the surrogate model into MEDEAS is split in two smaller steps:

1. **Vensim integration**, that is to make the surrogate model, available as a Tensorflow model, callable from within a Vensim model
2. **Variable linking**, Link the input and output features of the surrogate model to the actual variables used in the MEDEAS model

6.2 Vensim integration

6.2.1 The Vensim software

Vensim is a system dynamics simulation software, developped by Ventana Systems, Inc. It primarily solves the system of differential equation represented by the user-defined model, and is mainly used, according to its description, "for developing, analyzing, and packaging dynamic feedback models" [28].

Its most common application areas include [32]:

- Transportation and energy,
- Project management,
- Environment.

Vensim also comes in different distribution, such as Vensim PLE that is the free, personal learning edition. In this work, Vensim DSS is used, with an academic license.

6.2.2 Vensim models

Vensim provides a variety of tools to describe models, but at the end every model is an interconnection of variables, the math hiding in the connections between these.

Vensim provides the following types of variables:

- **Auxiliary variables**, that are regular variable that have no memory, that is, are independent from their value at the previous time step and are computed from every type of variable.

  For example, a *temperature* variable that is computed from some *sunshine* and *latitude*, that is used to compute the birth rate of *rabbits* and $foxes$.

- **Constant variables**, that hold one value.

  For example, a mathematical constant like π.

– **Data variables**, or exogenous variables, whose value evolve over time but is not dependent of the model.

  For example, typical *sunshine* data over a year.

– **Stock variables**, that change only over time as a function of the incoming rates, i.e., they integrates the rates.

  For example, the population of some species at a given time.

– **Rate variables**, or flows, that directly impact the Stock variables.

  For example, the birth or death rate of some population at a given time.

The connections between the variables are virtually done by arrows. The only practical use of arrows is to make the variable at the origin appear in the selection of variables in the variable equation screen for the variable pointed by the arrow. But obviously they are of great utility in terms of visualization of the model.

An example of a Vensim model is depicted on Figure 6.1.

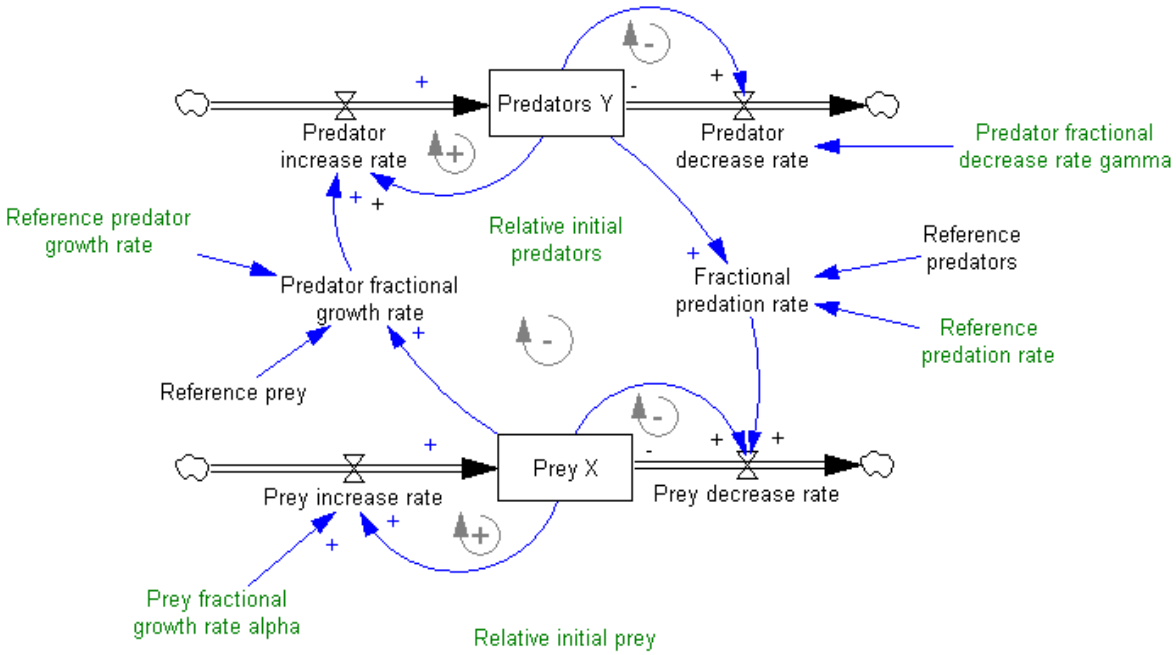


Figure 6.1: A model in Vensim: Lotka-Volterra predator-prey model.

6.2.3 Vensim external functions

This specific feature of the Vensim software is evidently of great interest for this work. It enables the user to provide and use any arbitrary function in Vensim models and simulations.

To do so, the user needs to provide a dynamically linked library (DLL), packaged as a `dll` file, then provide its path in Vensim. These files are Windows specific (as Linux uses `so` files and macOS `dylib`), hence they have to be handled as such.

Dynamically linked libraries are typically compiled from the C or C++ programming language. In this work, C++ has been chosen, as the library to load and call Tensorflow models that will be used is written in that language.

To be able to integrate the user-defined functions in its simulation environment, the user's library is expected to provide some functions, that are part of some interface that the Vensim software knows and can communicate with. This interface comprises of a set of functions that is described on Table 6.1.

Function name	Description
`version_info`	Provide information about the Vensim version this library has been built for
`set_gv`	Utility to set the global variable that depend on the Vensim simulation environment
`user_definition`	Provide a way to get all the necessary information about each user-defined function, mostly their names, number of arguments and a identifier code
`simulation_setup`	This function is called by Vensim on simulation startup, allowing the library to do some preparative work if needed, such as allocating memory
`simulation_shutdown`	Same as `simulation_setup`, but on simulation shutdown
`vensim_external`	This function is expected to, given an array of input arguments, their number and a function code, call the function associated to that code and write its return value into the first input argument.

Table 6.1: Description of the mandatory functions that a Vensim user library has to provide

Luckily, Vensim DSS ships with an example of such a library. As this file is not publicly disclosed, caution should be paid to keeping the library code private—actually, the external function capability is only available in Vensim DSS.

In the implementation of the library, this file was copied then adapted, as suggested in Vensim's documentation.

6.2.4 Vensim subscripts

This feature of Vensim is advertised as one of its most powerful tools. These allow to subscript variables appearing in the equations and perform operations across subscripted values if needed.

Importantly, MEDEAS use subscripts to run several simulations in parallel called scenarios. Therefore, one needs to edit the subscript properties in order to manage the scenario that will be run and displayed.

6.2.5 Calling a Tensorflow model

It is thus needed to call our model, presented to as a Tensorflow model, from the C or C++ language.

In order to do so, one basically needs two things:

- the said model, saved in a directory from the Tensorflow python API.
- the Tensorflow library, that is another DLL, to perform the actual computations from C/C++.

The complicated part being the linking between the two. In order to do so, the Cppflow tool is used. It is an "in-between" layer from the C++ side and the Tensorflow model side.

This tool is not available in C, this is the main reason why the library is written in C++.

The main purpose of Cppflow is precisely to run Tensorflow models from C++, and to acheive this it provides easy to use functions to load such a model, and to call it with some given intput data.

To carry out all these operation in a library, it is thus required to link properly the library code to the two tools it uses, that are the Tensorflow DLL and Cppflow. Hence, using the GNU `make` facility to accomodate the compilation, a few `Makefile`s were written[3].

On this, one thing is to be remembered: some code may compile and link successfully, but one also has to link properly the path to the DLL you linked to, that is, not only to the compiler.

6.3 The pysd option

As previouly introced, the `pysd` python library can read and run Vensim model files. Of course, as the surrogate model is primarily defined in that language, one would deduce that its integration would be easier that way.

Doing so, while not explored in this work, is actually expected to be simple. Using `pysd` loading functionality, a python module representing the simulation is obtained. This module is then loaded with `pysd` for the simulation to be run. The linking can thus be done by editing the module file directly, and inserting the call to the model at the appropriate place.

This approach is nevertheless not chosen. While it has been thought about late in the work, after the current Vensim integration had been implemented, the main reason is the convenience of the resulting model. If the model combination was made available sa python module, it would be much harder for external people to edit the original MEDEAS model from Vensim, then run the modified version while keeping the surrogate model integrated. The editor would have to manually re-insert the surrogate model into the python module, that also have to be recreated.

Keeping the surrogate model available as a Vensim external function is therefore a significant benefit for the further improvements of the model, as it enables edits to be made to MEDEAS independently of the surrogate model.

Another update has to be mentionned here. In facts, J. Paris had trouble making the MEDEAS model run successfully with the integrated surrogate model had contact with the MEDEAS developping team, that advised to wait for the release of a update of the MEDEAS port to python, making use of `pysd` and expected to be more stable and convenient. The main issue was that she were to leave before the anticipated release date, such that it was not acheivable.

However, this newer port to python with `pysd` may at the end make the integration of the model directly in python more portable and convenient, hence obtaining the same advantage of the Vensim external function. But this function was created and finished before learning the future availability of the new python port.

[3] At some point creating a program calling a Tensorflow model was managed, but this did not worked from the DLL. Therefore, a workaround was developed using a worker process (a seperate program) and Windows' tools for interprocess communication. But it turned out that the reason why the call could not be made to Tensorflow originated from an error in the linker arguments when compiling the library. With this fixed, it made the worker process workaround useless.

6.4 Variable linking

In its inner workings, the MEDEAS model does not directly uses all the variables that appear in our surrogate model. What arises the need for some links to be drawn between the two.

Hopefully, the desired values are often closely related to the other variables that are already present in the MEDEAS energy module, so these connection are expected to be as simple as linear rescalings or combinations, perhaps some more for the rNTC input.

The input variables, that are summarized on Table 4.3, do not map directly to already existing variables in the MEDEAS model. Thus, some mappings have to be made between the inputs and outputs of the surrogate model, the Dispa-SET side, and the MEDEAS side.

This work has been done with the help of Jade Paris, a student making her master's stage thesis on this specific topic as well.

6.4.1 Variables available in MEDEAS

The first step in this process is to list relevant variable appearing in MEDEAS, that will have to be exploited in order to do the linking.

These variable are listed on Table 6.2.

MEDEAS variable name	Description	Unit	Notation
FE elec generation from solar PV TWh	Total yearly production from solar photovoltaic units	TWh	$Generation_{PV}$
Total FE Elec demand TWh	Yearly total electricity demand	TWh	$Demand_{tot}$
FE Elec generation from offshore wind TWh	Total yearly production from offshore wind turbines	TWh	$Generation_{wind-offshore}$
FE Elec generation from onshore wind TWh	Total yearly production from onshore wind turbines	TWh	$Generation_{wind-onshore}$
Total capacity elec storage TW	Total power output of storage units	TW	$Storage_{tot}$
Total FE Elec genetaion TWh EU	Total yearly electricity production from all units	TWh	$Generation_{tot}$
new capacity installed growth rate RES elec	Yearly growth rate of the electric network capacity	[·]	$Growth_{capacity}$

Table 6.2: Relevant variable in MEDEAS

Unfortunately, some values are not present at all in MEDEAS, so that they can't be deduced from the model as is. But as these are mandatory to run the model, a value has to be provided imperatively. In this case, a constant value will be set.

As a consequence, additional parameters of the MEDEAS have been introduced, and will from now on be handled adequately. That is, as these may have an influence on the results, this influence will have to be assessed.

6.4.2 Linkings

The relations drawn from MEDEAS' variables to the surrogate model's are shown on Table 6.3.

Input variable (from surrogate model)	Linking equation
$share_{PV}$	$share_{PV} = \frac{Generation_{PV}}{Demand_{tot}}$
$share_{wind}$	$share_{wind} = \frac{Generation_{wind-onshore} + Generation_{wind-offshore}}{Demand_{tot}}$
$share_{flex}$	$share_{flex} = 40\%$
$share_{storage}$	$share_{storage} = \frac{Storage_{tot}}{Demand_{tot}} \times 365 \times 24$
$Capacity_{ratio}$	$Capacity_{ratio} = \frac{Generation_{tot}}{Demand_{tot}}$
$rNTC$	$rNTC = Growth_{capacity}$

Table 6.3: Variable linking equations. The left-hand side are Dispa-SET variable, and right-hand side MEDEAS variables.

These linkings have then to be drawn inside of the Vensim model, or directly in the computation if we use the python version of MEDEAS.

7 Analysis

7.1 Overview

This sections targets at describing the results observed when running the MEDEAS model integrating the created surrogate model.

This comparison will obviously have to be made over the different scenarios that are defined by default by MEDEAS, that differ for the most part by the evolution of the electricity production mix. These scenarios were detailed in Section 3.6.

7.2

References

[1] O.M. Babatunde, J.L. Munda, and Y. Hamam. "Power system flexibility: A review". In: *Energy Reports* 6 (2020). The 6th International Conference on Power and Energy Systems Engineering, pp. 101–106. ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2019.11.048. URL: https://www.sciencedirect.com/science/article/pii/S2352484719309242.

[2] Maarten Brinkerink et al. "Assessing global climate change mitigation scenarios from a power system perspective using a novel multi-model framework". In: *Environmental Modelling and Software* 150 (2022), p. 105336. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2022.105336. URL: https://www.sciencedirect.com/science/article/pii/S1364815222000421.

[3] Iñigo Capellán-Pérez, Carlos de Castro, and Luis Javier Miguel González. "Dynamic Energy Return on Energy Investment (EROI) and material requirements in scenarios of global transition to renewable energies". In: *Energy Strategy Reviews* 26 (2019), p. 100399. ISSN: 2211-467X. DOI: https://doi.org/10.1016/j.esr.2019.100399. URL: https://www.sciencedirect.com/science/article/pii/S2211467X19300926.

[4] Tianqi Chen and Carlos Guestrin. "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145%2F2939672.2939785.

[5] MEDEAS consortium. *Deliverable 6.2. Costs of the transition*. 2018. URL: https://medeas.eu/system/files/documentation/files/D6.2%28D20%29_Transition%20Costs.pdf.

[6] GAMS Development Corporation. *General Algebraic Modeling System (GAMS) Release 24.5.6*. 2015. URL: https://www.gams.com/.

[7] J.P. Deane et al. "Soft-linking of a power systems model to an energy systems model". In: *Energy* 42.1 (2012). 8th World Energy System Conference, WESC 2010, pp. 303–312. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2012.03.052. URL: https://www.sciencedirect.com/science/article/pii/S0360544212002551.

[8] Erik Delarue and Jennifer Morris. *Renewables Intermittency: Operational Limits and Implications for Long-Term Energy System Models*. MIT Joint Program on the Science and Policy of Global Change, 2015. URL: http://hdl.handle.net/1721.1/95762.

[9] pyDOE documentation. *Randomized designs*. Accessed in May 2023. URL: https://pythonhosted.org/pyDOE/randomized.html#latin-hypercube.

[10] P. Geurts and L. Wehenkel. "Introduction to Machine Learning". ELEN062-1 class. 2021.

[11] Per Helgesen et al. "Using a hybrid hard-linked model to analyze reduced climate gas emissions from transport". In: *Energy* 156 (May 2018). DOI: 10.1016/j.energy.2018.05.005.

[12] K. Kavvadias et al. "Integrated modelling of future EU power and heat systems: The DispaSET v2.2 open-source model". In: *JRC Technical Report, EU Commission* (2018).

[13] Wei Lin, Zhifang Yang, and Juan Yu. "Flexibility of interconnected power system operation: Analysis, evaluation and prospection". In: *Energy Conversion and Economics* 1.3 (2020), pp. 141–150. DOI: https://doi.org/10.1049/enc2.12013. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/enc2.12013. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/enc2.12013.

[14] G. Louppe. "Deep learning". INFO 8010 class. 2022.

[15] Eneko Martin-Martinez et al. "PySD: System Dynamics Modeling in Python". In: *Journal of Open Source Software* 7.78 (2022), p. 4329. DOI: 10.21105/joss.04329. URL: https://doi.org/10.21105/joss.04329.

[16] Donella H. Meadows, Jorgen Randers, and Dennis L. Meadows. "The Limits to Growth (1972)". In: *Documents of Global Change*. Ed. by Libby Robin, Sverker Sörlin, and Paul Warde. New Haven: Yale University Press, 2013, pp. 101–116. ISBN: 9780300188479. DOI: doi:10.12987/9780300188479-012. URL: https://doi.org/10.12987/9780300188479-012.

[17] medeas.eu. *MEDEAS home page*. Accessed in May 2023. URL: https://medeas.eu.

[18] mit.edu. *Optimal Latin Hypercube Technique*. Accessed in May 2023. URL: https://abaqus-docs.mit.edu/2017/English/IhrComponentMap/ihr-c-Reference-OptimalLatin.htm.

[19] NREL. *Renewable Electricity Futures Study (Entire Report)*. Golden, CO, USA: National Renewable Energy Laboratory, 2012.

[20] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[21] D. Ernst P. Geurts and L. Wehenkel. "Extremely randomized trees". In: *Machine learning* (2006). URL: https://doi.org/10.1007/s10994-006-6226-1.

[22] Gonzalo Parrado-Hernando et al. "Capturing features of hourly-resolution energy models through statistical annual indicators". In: *Renewable Energy* 197 (2022), pp. 1192–1223. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2022.07.040. URL: https://www.sciencedirect.com/science/article/pii/S0960148122010357.

[23] Matija Pavicevic et al. "Bi-directional soft-linking between a whole energy system model and a power systems model". English. In: *2022 IEEE PES/IAS PowerAfrica, PowerAfrica 2022*. Kigali, Rwanda: Institute of Electrical and Electronics Engineers Inc., 2022. DOI: 10.1109/PowerAfrica53997.2022.9905392. URL: http://xplorestaging.ieee.org/ielx7/9904951/9905072/09905392.pdf?arnumber=9905392.

[24] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[25] *Power System Flexibility for the Energy Transition, Part 1: Overview for policy makers*. Abu Dhabi: International Renewable Energy Agency, 2018. URL: https://www.irena.org/publications/2018/Nov/Power-system-flexibility-for-the-energy-transition.

[26] Sylvain Quoilin, Ignacio Hidalgo Gonzalez, and Andreas Zucker. *Modelling Future EU Power Systems Under High Shares of Renewables: The Dispa-SET 2.1 open-source model*. English. Publications Office of the European Union, 2017. ISBN: 978-92-79-65265-3. DOI: 10.2760/25400. URL: https://ec.europa.eu/jrc/en/publication/eur-scientific-and-technical-research-reports/modelling-future-eu-power-systems-under-high-shares-renewables-dispa-set-21-open-source.

[27] Umar Taiwo Salman et al. "A Review of Improvements in Power System Flexibility: Implementation, Operation and Economics". In: *Electronics* 11.4 (2022). ISSN: 2079-9292. URL: https://www.mdpi.com/2079-9292/11/4/581.

[28] Inc. Ventana Systems. *Vensim software*. Accessed in May 2023. URL: https://vensim.com/vensim-software/.

[29] Carla Vidal Montesinos. *Integrating short-term dispatch constraints in a system dynamics energy planning model*. 2022. URL: http://hdl.handle.net/2268.2/16566.

[30] Wikipedia. *Energy return on investment*. Accessed in May 2023. URL: https://en.wikipedia.org/wiki/Energy_return_on_investment.

[31] Wikipedia. *Latin hypercube sampling*. Accessed in May 2023. URL: https://en.wikipedia.org/wiki/Latin_hypercube_sampling.

[32] Wikipedia. *Vensim*. Accessed in May 2023. URL: https://en.wikipedia.org/wiki/Vensim.

[33] B.A. Williams and S. Cremaschi. "Surrogate Model Selection for Design Space Approximation And Surrogatebased Optimization". In: *Proceedings of the 9th International Conference on Foundations of Computer-Aided Process Design*. Ed. by Salvador Garcia Muñoz, Carl D. Laird, and Matthew J. Realff. Vol. 47. Computer Aided Chemical Engineering. Elsevier, 2019, pp. 353–358. DOI: https://doi.org/10.1016/B978-0-12-818597-1.50056-4. URL: https://www.sciencedirect.com/science/article/pii/B9780128185971500564.

Annex A: scripts and code

This annex documents briefly the roles of each scripts and code files.

Data generation

See Table 8.1.

File name	Description
`config.py`	Holds high level specification of the dataset to be created, such as the number of samples, the LHS strategy, and output files names.
`read_results.py`	Fetches the results from simulation directories, either one by one or all at once.
`reference.py`	Runs the reference simulation and serializes the results in a json file.
`sampling.py`	`-sample-only`: only run the LHS and store the samples in a CSV file. `-prepare-one idx`: prepares the simulation directory for one sample given its index. With no arguments, runs LHS and prepare all the simulation directories.
`utils_francois.py`	Stores the modified version of the `adjust_capacity` function of DispaSET.
`main.sh`	Starts all the scripts in the right order in order to produce a dataset. Runs the reference simulation, the sampling, prepends the header to the dataset file (as CSV), and starts the first series.
`launch-job-series.sh`	Submits a series of simulation jobs, and a job that will submit the following series with the current one as a dependency. If the series index given as argument is too high, exits.
`launch-reference-job.sh`	Submits a job that runs the reference simulation.
`launch-simulation-jobs.sh`	Submits the jobs required to run a simulation from a series. It takes the series index as an argument and the number in that series from SLURM environment variables. Uses `sampling.py -prepare-one`, GAMS and `read_results.py` successively.
`gams-simulation.sh`	Submits a job running the GAMS simulation of an already prepared simulation, for testing purposes.
`read-one.sh`	Submits a job fetcing the results of an arleady ran GAMS simulation, for testing purposes.
`get-longest-simulation.sh`	Bash script that calls the `seff` utility on each of the simulation ran to extract the longest ones. This has mainly be used once to set the simulation timeout that prevent stalling simulation to waste resources.

Table 8.1: Description of the data-generation files

Neural network

See Table 8.2.

File name	Description
`config.py`	Holds the configuration of the network to be trained, name, data to use, inputs and outputs.
`model.py`	Holds the description of the model to be trained, via the `build_model` function.
`baselines.py`	Builds and trains different, predefined neural network architectures, and stores their performances. This eases the process of looking for a good architecture for the ANN, to guide the bounds in the hyper-parameter tuning. With `sort -k2 -t '>' -i logsbaselines-results.txt` one easily sorts the results by increasing order.
`train.py`	Executes the tuner search for the best model and training of that best model.
`view.py`	Holds different utilities to view the results of some model and its performances. Use `view.py -surface <in1> <in2> <out>` to create a 3D surface of the out-th output depending on the in1 and in2-th inputs. The other inputs are constant and parameterizable with sliders.

Table 8.2: Description of the files for the neural network part.

Integration

See Table 8.3.

File name	Description
`external.h`	Header file for `external.cpp`.
`external.cpp`	Main source file for the library.
`main.cpp`	Code for running a test program.
`Makefile`	GNU make file for automating compilation.
`tensorflow.dll`	Tensorflow library file for Windows, can be downloaded from here.

Table 8.3: Description of the integration files.