POLITECNICO
DI TORINO

Technische Universität München

TESI DI LAUREA MAGISTRALE
IN INGEGNERIA ELETTRICA

SMART GRIDS' SIMULATION

Interface of an electrical distribution network simulator
for smart grid simulators.

Energy Department
Prof. GIANFRANCO CHICCO
Electrical Engineering
Politecnico di Torino

candidate: HAMIDREZA MIRTAHERI

Department of Computer Science
Prof. HANS-ARNO JACOBSEN
Middleware Systems
Technische Universität München

supervisor: Victor del Razo

October, 2017

Thanks to all my professors
in Politecnico di Torino

To my parents...

Acknowledgements

This thesis has been carried out in the department of Computer Science, Technische Universität München, in the chair of professor Hans-Arno Jacobsen, Energy-Informatics group, under supervision of doctoral researcher Victor del Razo Sarmina, to obtain master of science degree in Electrical Engineering, held by Energy department of Politecnico di Torino. The agreement on exchange, topic and context is issued by professor Gianfranco Chicco.

Torino, 12 March 2016 Hamidreza Mirtaheri.

Abstract

The increasing share of renewable energy sources and distributed energy generation (DER) in the energy mix brings a number of challenges to the power system, particularly to the distribution network (DN). New control methods and algorithms are being studied to address these challenges, but the effects of such algorithms on actual DNs are often difficult to validate. Co-simulation, considering the advanced state of existing DN simulation tools, is a potential solution but usually requires some degree of expertise. Yet state-of-the-art simulation approaches need to be adapted for visual analyses, in presence of numerous players in the either physical and cyber layer.
In this work, first we introduce a framework for implementing control algorithms for DNs assisted by co-simulation. This framework uses the commercial simulator PowerFactory but does not require a high tool-specific expertise and can be integrated into other simulation tools. Then we apply this framework to an electric vehicle charging use case and show the results and benefits. Furthermore we extend an existing charging control algorithm to support local voltage control. And finally we provide a visualization extension for the most in use analysis tool; Matlab. This part is dedicated to develop visualization and heat map analyses for Matlab-based simulations.

Key words: Co-simulation, Smart Grids, Distribution Network, Visualization, PowerFactory, Python, Matlab, Smart Vehicle Charging.

Contents

List of Figures

List of Tables

Introduction

The increasing share of renewable energy sources in the energy mix brings a number of challenges to the power system, particularly to the distribution network (DN). First, given the random nature of these renewable sources, demand management and storage become increasingly important. Second, as distributed energy resources (DER) become more common and distribution level becomes active, medium and low voltage (MV, LV) networks face new challenges which were not considered in their original design.

Researchers and engineers have been working in new control methods to address these challenges, but the effects of such algorithms on actual DNs are often difficult to validate.

In the traditional paradigm where a super grid might be modeled through limited number of very large scale generations and load side also with a regular and somehow predictable in an aggregated frame, these dumb components provide good and acceptable results with required accuracy. Now as the inevitable transition towards smart grid highlights uncertainties especially in generation side, the methods to model a grid are becoming completely different.

The contributions of this thesis can be summarized as follows: introduction of an inter-domain and interactive co-simulation framework, in which the most important challenges of power system simulation have been addressed.

Then a demonstration of the proposed framework for Vehicle- Originating-Signals (VOS) electric vehicle (EV) charging control, and the extension of an existing VOS algorithm for local voltage control, are presented.

The result of different study cases and tests done using the introduced comprehensive co-simulation approach are also shown.

The rest of the work has been dedicated to the visualization issue for the most in-use power system simulation tool; Matlab.

1 A Framework for Control and Co-Simulation in Distribution Network

This chapter presents an overview of simulation tools, pros and cons, and also taking a look at some already tried co-simulation methods. The last part of the chapter indicates why and how to create a new co-simulation framework.

1.1 Background; Power system simulation tools

The complexities on Smart Grid studies are growing up as more agents are being integrated in the energy chain. Important challenges refer to modeling a big number of constitutive elements in power system instead of considering them as dumb components, integrating Information and Communication Technologies (ICT) and involving power market models simultaneously. As a result, smart grid simulation is an evolutionary move towards modeling and simulation of either physical layer, decision making layer (function of economic signals) and cyber layer all together. Additionally, Smart Grid simulation is increasingly requiring a lower time granularity and the integration of external events during runtime, resulting in an extremely high computational burden.

Some studies try to address this by coordinating the operation of a decision-making tool, either artificial-intelligence or multi-agent-based, and a power analysis tool [1].

Smart Grid's analysis generally consists of steady-state, dynamic (electromechanical), and transient (electromagnetic) studies among other power system aspects. However, state-of-the- art co-simulation approaches are mostly domain-specific and unable to model a fully detailed network. In [2] a Matlab toolbox for GridLAB-D that supports modeling, simulation and grid impact analysis is introduced. Access to Matlab toolboxes and a debugging feature prove useful for power-flow-based analysis feature. This co-simulation approach has been also used in [3] where GridLAB-D simulates

the distribution networks while MATLAB deals with the control schemes. Such frameworks, use scripts to describe the physical network, and are constrained to a specific domain excluding, for example, short circuit analysis. Therefore those functions must be defined completely by the user, which is not an easy task, especially at the distribution level and in the presence of DER. Reference [4] presents a co-simulation of distribution network with the simulators GridLAB-D and EnergyPlus [5], via Functional Mockup Interface (FMI). Here, the effects of weather and environmental factors are reflected into energy consumption and the DN state.

Trends for co-simulation frameworks based on open source tools have been considered by [6] with a combination of WindMil, GridLAB-D and OpenDSS. Substation active power, currents and voltage at the furthest bus, obtained from each tool behave consistently. However, a slightly different behavior of voltage drop along feeders has been found in some cases [6]. Authors in [7] present PowerWorld automation with Matlab, via PowerWorld's Add-On, Simulator Automation Server (SimAuto). PowerWorld is known as an "essentially" high voltage simulator, nevertheless it could also be suitable for medium voltage (MV) analysis, integrating an Electromagnetic Transients (EMT) module if required.

In [8] a combination of flexible Smart Grid co-simulation framework Mosaik [9] with an all-round power system analysis methods software DIgSILENT PowerFactory [10] is introduced. It is made to compare and thus validate the output of open source simulator PYPOWER power flow with the output of the "trusted" simulator PowerFactory. The proven advantages of the PowerFactory software are its overall functional integration and its applicability to the detailed modelling from power elements to environmental and weather conditions.

In [11] a transient analysis is performed using Power- Factory's integrated MATLAB/Simulink interface mechanism. In [12] comparisons for different simulation coupling for PowerFactory are provided, referring to version 14.1 of PowerFactory; some of the issues have been dealt with in later releases.

Co-simulation approaches also include coordinating power and communication network simulators, where there have been several successful attempts to model power-communication networks for smart grids [13-17]. Although existing tools address a number of co-simulation requirements, there is need of a simulation framework that goes beyond single domains e.g., be able to analyze steady-state and transient models in the same framework.

Comprehensive studies on simulation tools suggest a wide range of issues associated with the distribution network level. A major issue [18] is that, in order to support the

study of the same system for different domains, either many different software tools need to be used, or an all-in-one analysis package or a suite of programs that could operate on the same database must be made available.

On the other hand, since that effective power system operation requires power system engineers and operators to analyze vast amounts of information in systems containing hundreds of buses, a key challenge is to present data such that the user can intuitively and quickly assess the state of the system [19].

Simulation visualization is key to system analysis, highlighting important patterns to facilitate decision making [20]. Animated simulation visualization is also valuable [21].

1.2 DIgSILENT PowerFactory

As is mentioned above in order to create an efficient and comprehensive structure for a co-simulator, an all-in-round analysis software is the indispensable foundation.

1.2.1 Software

The proven advantages of the PowerFactory software are its overall functional integration, its applicability to the modeling of generation, transmission, distribution and industrial grids, and the analysis of these grids' interactions.

DIgSILENT PowerFactory is the most economical solution, as data handling, modeling capabilities and overall functionality replace a set of other software systems, thereby minimizing project execution costs and training requirements. The all-in-one PowerFactory solution promotes highly-optimized workflow. DIgSILENT PowerFactory is easy to use and caters for all standard power system analysis needs, including high-end applications in new technologies such as wind power and distributed generation and the handling of very large power systems. In addition to the stand-alone solution, the PowerFactory engine can be smoothly integrated into GIS, DMS and EMS supporting open system standards.

Some PowerFactory highlights are:

- Extensive and flexible modelling capabilities with rich suite of power equipment models and libraries,
- Supports all network representations and phase technologies, i.e. any kind of

radial or meshed, 1, 2, 3 and 4-wire (combined) AC and DC networks,

- Powerful network diagrams and graphic/visualization features,
- Single- and multi-user environment with full support of team working, user accounting, profiles and flexible customization,
- Unique data management concept including project versioning and archiving mechanisms, master/derived concepts with compare and merge tools,
- Unlimited opportunitites in process optimisation based on integrated scripting functionality,
- Rich interfacing and system integration options (e.g. GIS, SCADA, EMS).

1.2.2 Functional Integration and Applications

PowerFactory is highly adapted for Smart Grid modeling:

- Includes virtual power plants and distributed generation such as PV-panels, micro turbines, battery storage, CHP, etc,
- Once implemented as a single software solution (while modeling physical layer), allows for fast 'walk around' through the database and execution environment,
- No need to reload modules and update, transfer and convert data and results between different program applications,
- Vertically integrated power equipment model concept allowing models to be shared by all analysis functions,
- Support of transmission-, distribution- and industrial system design and simulation,
- Modelling and simulation of railway systems,
- Simulation of any kind of wind turbines and wind parks.

1.2.3 Application Programming Interface (API)

PowerFactory API offers third party applications the possibility to embed PowerFactory's functionality into their own program. Technically, the interface is realized in C++ and provided as a DLL that can dynamically be linked to any external application.

1.2.3.1 Logical Structure

The PowerFactory API is a logical layer of the PowerFactory application that encapsulates the internal data structures and makes them available to external applications. Its purpose is to give a consistent interface being as close as on the PowerFactory data model. The API takes care about internal memory management and data persistency. It does not allow any external functions to access directly PowerFactory objects as illustrated in Fig. 1.1.

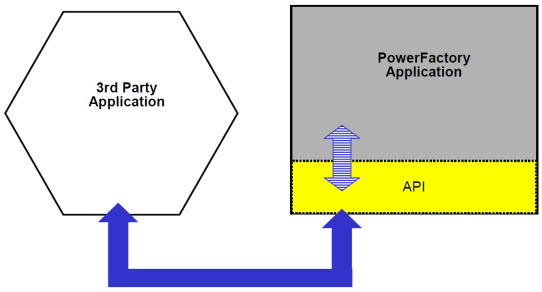


Figure 1.1 – Third party application of PowerFactory

1.2.3.2 Physical Structure

Physically, the interface consists of the following files;
digapi.dll: dynamic library that is part of the PowerFactory application. This library holds the implementation of the interface layer. For static linking, a **digapi.lib** can be provided. **api.hpp**: interface definition as a C++ header file. **apivalue.hpp / apivalue.lib**: header and static library for the value transfer object that is used by the **API.l**, as is depicted in figure 1.2.

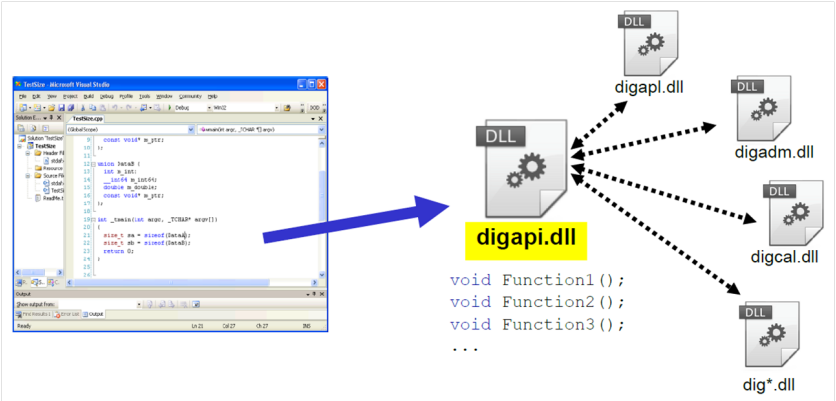


Figure 1.2 – Api's physical structure

1.2.3.3 Python API

PowerFactory provides its interface with programming languages such as C++, Python and Matlab.

Among those Python API offers a complete access to all functions and classes, in all domains.

Table 1.1 – Available domains for APIs

	Steady-state	EMT	EMC
C++	✓	✓	✓
Python	✓	✓	✓
Matlab	✓	X	X

1.3 Jupyter Notebook

The Jupyter Notebook is a web application that allows the user to create and share documents that contain *live* code, equations, visualizations and explanatory text.

Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.

IPython provides extensions to the Python programming language that make working interactively convenient and efficient. These extensions are implemented in the IPython Kernel and are available in all of the IPython Frontends (Notebook, Terminal, Console and Qt Console) when running this kernel.

One of most interesting particularity of Jupyter Notebook is its support of HTML, CSS and JavaScript. Then we will use HTML to define the content different pages, CSS to specify the layout of notebook and JavaScript to program the behavior of pages using the Widgets also.

High interactivity takes to the conclusion that Jupyter Notebook seems to be an adequate User Interface for a co-simulator.

1.3.1 Parallel Computing

Jupyter Notebook allows parallel computing in different Notebook at the same time, and also synchronization between them, providing an interesting solution for either

parallel computing and interface modeling.

To launch a program or script there is no need to relaunch and restart the entire of script, is enough to run a specific (needed) cell.

1.3.2 Widgets

Another superiority of this Notebook is its interactivity, which is also reinforced thanks to JavaScript widgets.

1.3.2.1 OpenStreetMap

As an example of the widgets, OpenStreetMap might be mentioned. With all its utility and features from which some will be mentioned in this work.

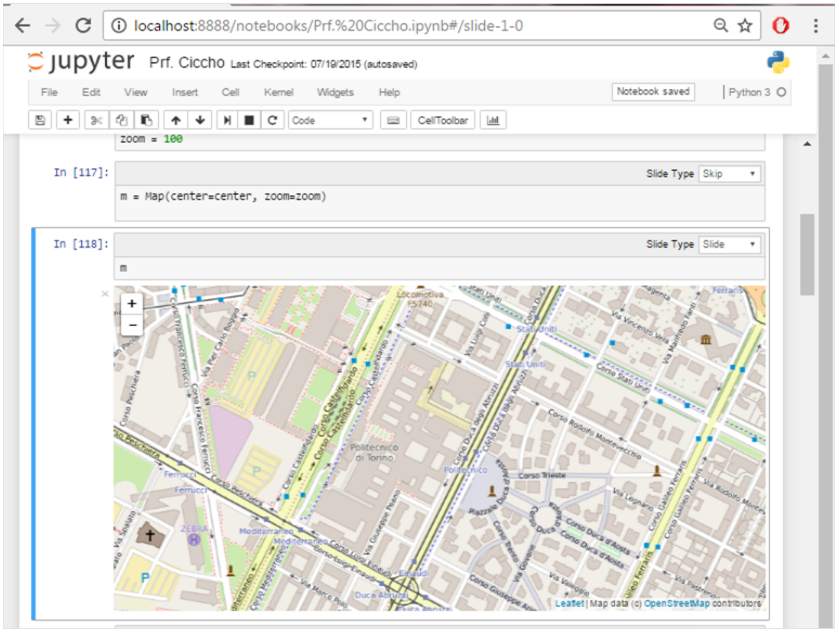


Figure 1.3 – OpenStreetMap, geographic features.

Adding shapes and signs is one of characteristic of OpenStreetMap. Figure 3.1 shows a demonstration of OpenStreetMap in Jupyter Notebook.

1.4 Black-Box Co-Simulation

The main focus of this co-simulator will be on;

- *Functionality*
- *Interactivity*

1.4.1 General Overview

The framework considered here for implementing control algorithms in a distribution network is based on the commercial simulator DIgSILENT PowerFactory. To gain access to PowerFactory's functionality, its Application Programming Interface (API) is used focusing on its Python API to gain interactivity. For the user side, given the mentioned conclusion, Jupyter Notebook is used as a very high interactive user interface.

Then a modular structure is implemented in a Black-Box mode, that enables the user to synchronize and get use of several simulator, even simultaneously.

The proposed framework allows co-simulation with a distribution network simulator without the need of expert knowledge in the tool. The user is just expected to know how to define grid or study case and to run the simulation, in a simple way, no matter which simulation tool is being used for modeling and how it is going to be interfaced. Figure 1.4 shows the general scheme of the framework at high level.

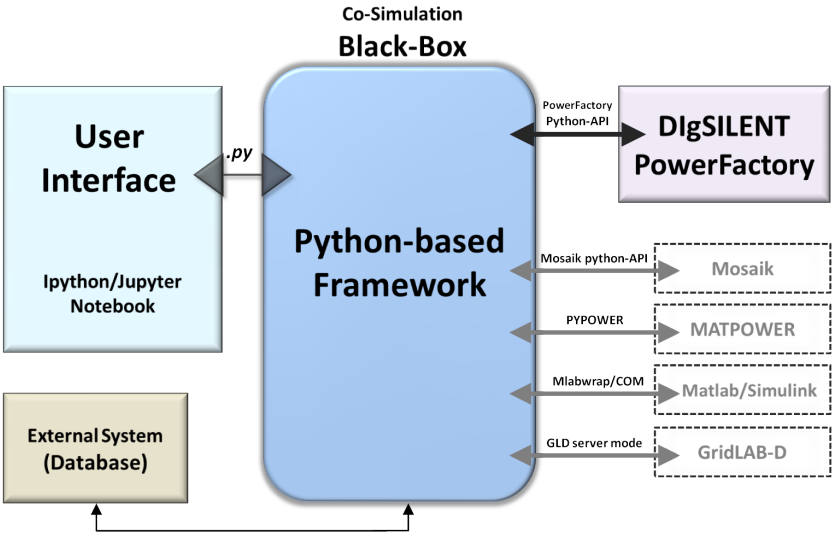


Figure 1.4 – Python-based power system analysis integration for control and co-simulation

In the proposed framework, it is possible to launch a wide range of analyses and calculations. Being based on PowerFactory, steady-state, quasi-steady-state, dynamic

and transient (EMT) analyses are supported for power flow, short circuit, stability, DC system analysis, optimal power flow, contingency, reliability, harmonics and power quality, power market modeling and analysis simulation. The user could also take advantage of numerous available standard component models and characteristics. Since the proposed framework is conceived as a middleware element, one could potentially integrate additional tools like Mosaik network analysis through its Python API, MATPOWER via its Python-port PYPOWER [22], Matlab/ Simulink in various manners, either through Python libraries, e.g., mlabwrap or MATLAB Engine for Python [23], and GridLAB-D, in server mode.

1.4.2 Functionality

From the functionality perspective, the framework is based on Python and can be accessed via an Ipython/Jupyter Notebook. This allows for efficient prototyping and interactive programming. Designers and programmers also benefit from extensive access to Python libraries and packages, and the ability to interact with large range of programming languages.

1.4.3 Architecture

From the architecture perspective, the proposed framework benefits from modularity. That is, an interface module has been implemented for interacting with the simulator, making it possible, in the future, to design new interface modules for other simulation tools. This interface module enables the algorithm to access all details of individual building blocks and benefits from the extensive functionality of state-of-the-art simulators.

Figure 1.5 demonstrates the architecture of the proposed framework.

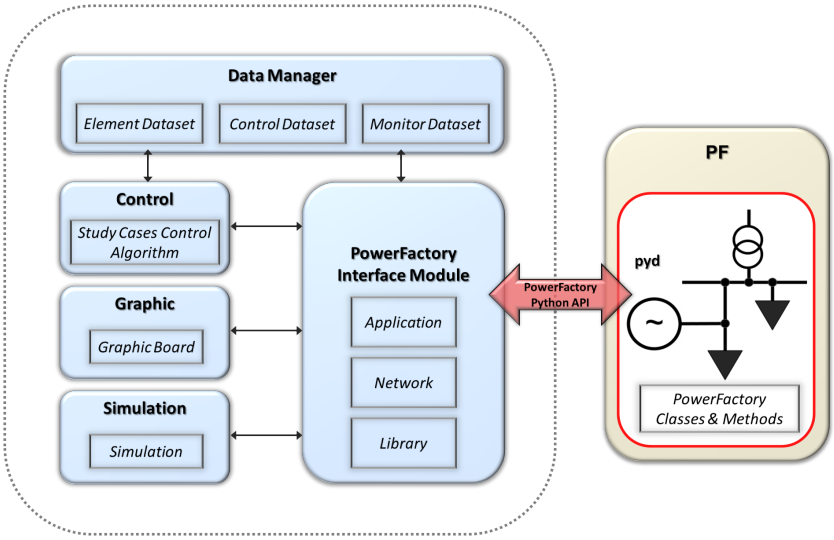


Figure 1.5 – Modular Architecture of Co-simulation Framework

1.4.3.1 PowerFactory Management

This module is designed to *import*, get access and use of PowerFactory objects. All bilateral read/write and command execution are being handled through provided classes. To get start with in every simulation imports PowerFactory module, that in its turn interfaces with the PowerFactory API (Application Programming Interface).

To allow Python to have access to PowerFactory , “powerfactory” module must be imported. This solution enables a Python script to have access to a comprehensive range of data available in PowerFactory such as:

- All objects
- All attributes (element data, type data, results)
- All commands (load flow calculation, etc)
- Most special built-in functions (DPL functions)

The PowerFactory objects such as calculation functions and elements are accessible from this module.

Using instance of this class makes it possible to access database tree, and all files and projects.

```
In [1]:
import power_factory_management.application as pf_app
pf_application = pf_app.PowerFactoryApplication()
pf_application.open_pf()
pf_application.show_database_tree()
pf_application.load_project('SmartEV(3)')
pf_application.get_study_cases()

out [1]:
[<powerfactory.DataObject <l1>\HAMID REZA.IntUser\
  SmartEV(3).IntPrj\Study Cases.IntPrjfolder\Study
  Case.IntCase</l1>>]
```

Figure 1.6 demonstrate the hierarchical database of PowerFactory which is completely accessible from the user-interface.

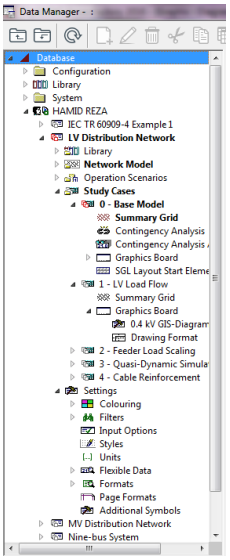


Figure 1.6 – Database tree of PowerFactory

Library; This class embeds the *templates* of elements and objects including time-characteristics into the local library, as soon as a project in activated.

Application; through this class the PowerFactory becomes accessible. It is the *handle class* of the black-box.

Network; acts as network constructor, this class forms a database of all constructive object of a project in a particular framework, adapted for the black-box structure.

```
In [1]:
load_low_voltage_data_manager.build_data_set(
  the_complete_network.load_low_voltage_elements)
solar_pv_data_manager.build_data_set(the_complete_network.
  solar_pv_elements)
trafo_station_data_manager.build_data_set(
  the_complete_network.trafo_station_elements)
substation_data_manager.build_data_set(
  the_complete_network.substation_elements)
node_data_manager.build_data_set(the_complete_network.
  node_elements)
```

1.4.3.2 Data Manager

Control and registration of all input-output data is assigned to this module, which includes three classes; Element Dataset, Monitor Dataset and Control Dataset.

Element Dataset, in fact, Element Dataset is a dictionary with key ElementName and a dictionary consisting of ElementType, Control, Monitor, ControlVariables, MonitorVariables.

Monitor Dataset, monitors the results, in a desired framework depending on the elements of interest assigned by the user.

```
In [1]:
fill_in_monitors(manager, monitor, network, the_sim.
  simulation_time_in_UTC_tz)
```

Control Dataset, every action carried out by the controller module will be recorded with the time and exact specification of that action.

1.4.3.3 Simulation Management

The simulation module makes it possible to pause a simulation-run, check results and modify parameters if is needed, and resume it again. This attribute helps engineers, especially in smart grid time-consuming simulations. Also this module sets datetime for start, stop time as datetime object with valid timezone. Time converter to UTC is included in the simulation module. Simulation granularity is in seconds.

Simulation, every instance of this module holds all necessary attributes for an independent and parallel simulation.

```
In [1]:
SIMULATION_START = dt.datetime(2012,6,5, 12, 0, 0, tzinfo=
  tz.utc)
SIMULATION_END = dt.datetime(2012,6,6, 12, 0, 0, tzinfo=tz
  .utc)
STEP_SIZE = 180
SECONDARY_STEP_SIZE = 60
start = time.time()
the_sim = sim.Simulation()
inner_sim = sim.Simulation()
the_sim.set_simulation_parameters(pf_application,
  SIMULATION_START,SIMULATION_END,STEP_SIZE)
#Initialize simulation:
the_sim.initialize_simulation(pf_application)
```

1.4.3.4 Controllers

This module hosts the study cases' classes.

EV Smart Charging, For the main use case of the co-simulator which is electric vehicle charging, three class **Grid**, **ElectricVehicles** and **SmartControl** are included. These classes use *inheritance* concept in such a way the parent is **Grid** class and both **ElectricVehicles** and **SmartControl** inherit from it. Figure 1.7 demonstrates the inheritance scheme of the smart EV charging controller, located in control module.

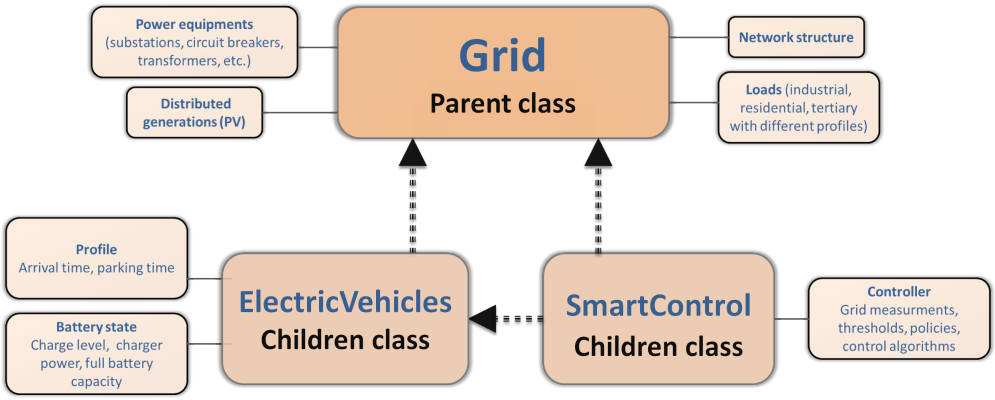


Figure 1.7 – Inheritance scheme in control module

1.4.3.5 Graphic

Graphic module that makes use and controls PowerFactory's graphic objects has been included to provide result visualization. As is mentioned above this part of work was one of the main focuses, interactivity.

After every simulation run, the user can extract quick-overview videos that show the results visually, where the thickness of the lines represent loading and its color, distance from its thermal limits. How much the loads and generations are being varied over the simulation time, is shown with the color of the nodes and its diameter.

Those mentioned animations are available in different windowing versions;

- single line diagram
- geographical map
- single elements of interest
- background image

Furthermore those informations are available in single frame images as well. More result information is also available by numerical values in the images and videos.

It is notable that these features are additional to the bunch of results being obtained by every simulation tool.

Graphic Board, is the interact with the graphic board subfolder in the PowerFactory project. A quick-overview video or single frame images are being provided in different

windows, that depends on which window is selected by the user.

Figure 1.8 shows a single frame result of a generic study case in single line diagram window.

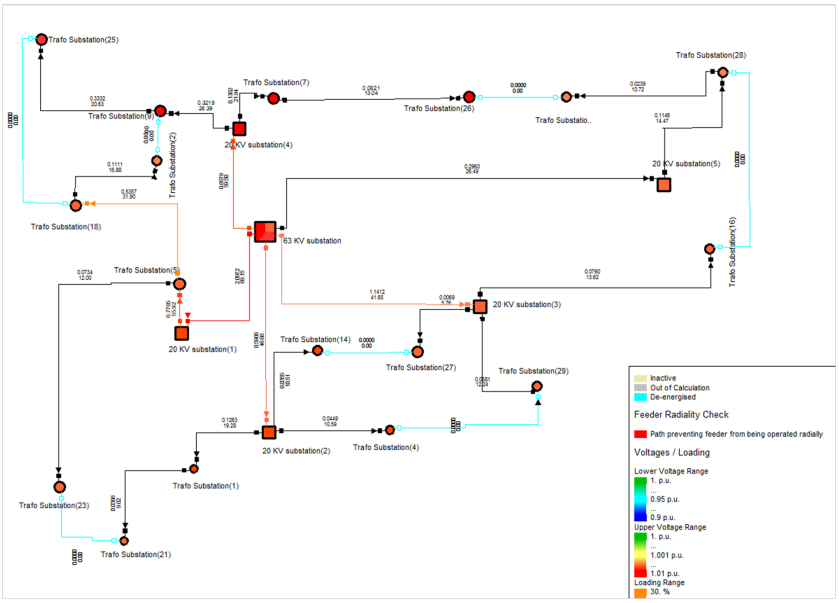


Figure 1.8 – Windowing mode

1.4.4 Ease of use

One can create the network and its detailed constitutive elements either via PowerFactory windowing-mode, or through our user-interface (UI) based on Ipython/Jupyter-Notebook. In the latter, study-cases, grids, elements and connections can be created with a simplified algorithm by the Network class. Templates and standard models are also embedded through the project's operational library.

The interactive and efficient use-interface is much suitable than every GUI. Figure 1.9 shows the user-interface for a generic case.

The study results are available in various formats such as csv, jpg, pdf and mp4, and are visible also in the user-interface itself.

In order to have a general overview, by the end of each simulation, quick animated mp4 videos files are produced. Those animations could be made as detailed as needed e.g. feeders, substations and transformer, even for every single element during the run-time, from geographical view or heat map.

Direction arrows show the active power flow direction, and power line thickness

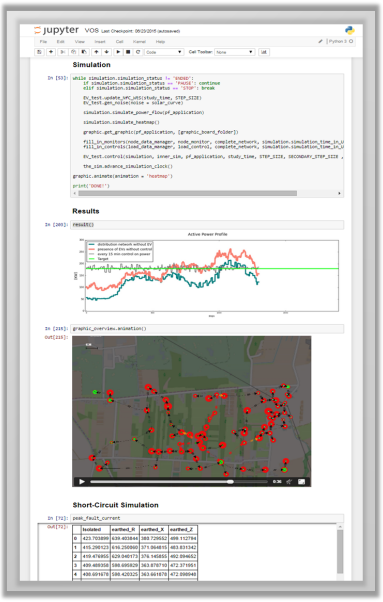


Figure 1.9 – User Interface

represents loading level with respect to the nominal characteristics. Through the Graphic module all graphic objects and their attributes are accessible, for instance setting color thresholds in the heat map.

Electrical engineers benefit from both user-friendly interface and outputs, provided very intuitively.

1.4.5 Features

This comprehensive co-simulation approach is capable to handle some important issues that could not be found altogether in a single framework. The most relevant features are:

- Support for pausing, modifying inputs and parameters, then resuming simulation-run from the same point, without the need to reload/restart.
- Support for wide range of analysis fields, e.g. power flow and short circuit, balanced and unbalanced.
- Support for exchanging input data, output data and parameters with an external system, e.g. a database controlled by the main simulator.
- Support for extracting the general state and the state of each element, at any point during simulation.

- Support for restoring a given state within a defined period of time in the past.
- Support for graphical visualization of inputs (network and elements) and outputs (results) with animations.

2 Electric Vehicle Charging with Vehicle-Originating-Signal

This chapter deals with the impact of electric vehicles presence within distribution grid.

2.1 Electric and Hybrid Vehicle within Electrical Grid

Mass deployment of Electric Vehicles will have significant impact on power networks, for two main reasons:

- Electric car charger devices are high electricity consumer,
  in scale of a normal low-voltage consumer. Current electric vehicles have chargers rated at from about 3 kilowatts to 20 kilowatts, although charger demand capacity is independent of the size of the car battery pack. Figure 2.1 shows typical home appliances power consumption versus electric car charger.

  Plug-in hybrid electric passenger vehicles have battery sizes that range from 7 kWh to 17 kWh, and battery-only electric vehicles have battery sizes that range from about 20 kWh to 50 kWh, or greater. Charger size determines how fast the battery draws energy from the grid; battery size relates to the amount of energy stored.

- Car charging is a behavioral issue,
  charging behaviors differ depending on whether the customer is charging at home or at a public station, and what type of vehicle they have, either an all-electric vehicle or a plug-in hybrid. Anyway charging time for the car holders coincide with the hours of pick-load, especially in public charging stations. Figure 2.2 shows a typical charging time occurrence.

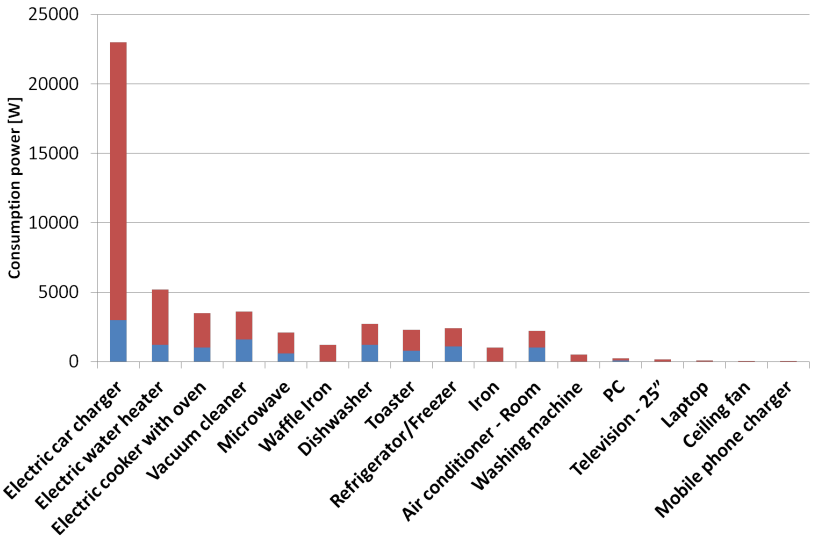


Figure 2.1 – Electrical Appliance Typical Energy Consumption Table

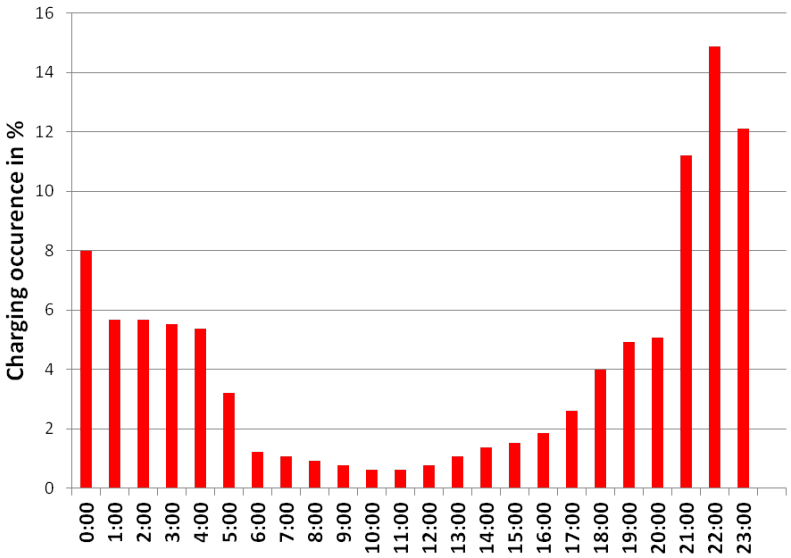


Figure 2.2 – Charging behavior

On the other hand, the electric car market data show constant increase in number of sold EVs in the recent years. The growth in Europe market according to **EV-Volumes** is shown in the Figure 3.1.

As the numbers of electric vehicles on the road increases, grid utilities may need to upgrade the existing infrastructure or build new capacity to handle growth in electricity demand.

In rise of the electric car, where plugging in an electric vehicle is, in some cases, the equivalent of adding three houses to the grid. That has utilities scrambling to upgrade the grid to avoid power outages and also taking new measures to save the existing grids within its operational limits. In reality it's not feasible to upgrade and rebuild new power equipment and infrastructure to meet this rapid growth, for either economic and technical limits.

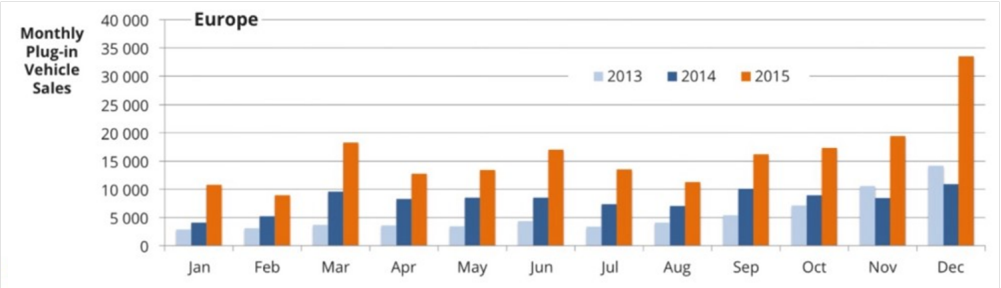


Figure 2.3 – Electric vehicle sales growth

2.2 Vehicle-Originating-Signal (VOS)

There have been several efforts and researches in this field going on in the recent years, like Vehicle-Originating-Signal (VOS). Its objective is to simplify the energy balance through reducing the difference between a target load profile and the power consumed by a section of consumers, for instance a neighborhood in being supplied by a section of a distribution system. That said, this aims as if an external energy resource was in chain, trough compensating for the variations using existing power supplies. This external energy resource is the result of a unit commitment process not based on re-dispatching instead to match expected demand and generator capabilities. Matching to a known planned supply permits the operators and utilities to use the available resources in a more efficient way since the need for reserve energy is lower. For the internal resources, it takes into account a fleet of electric cars and photovoltaics panels in the area of scenario. The electric vehicles are being looked like flexible loads, if those are just energy consumers as storage, or if those are available to supplying energy back to the grid in a scheme so-called as vehicle-to-grid (V2G).

That work comprehensively presents a smart charging control approach for EV real-time charging control, and takes into account computational burden, communications, aggregation, and also end user needs. The Vehicle Originating Signal method could be generalized for various loads and goals. It is also demonstrated how it can be exploited to cut down the variability nature of demand and distributed generation (DG) through smart charge and discharge plug-in electric vehicles in real time. The

mentioned study introduces an extension to the Vehicle Originating Signal method for reducing the communication overflow, that can reduce the messages' numbers by around 70%.

The objective of this algorithm is to manage the electric vehicles in such a way that the aggregated active power consumption, including constant demand and photovoltaics generation, closely follows a specific objective profile. The Vehicle Originating Signal method enables electric plug-in vehicles within area of control to compute signals indicating the need for energy and willingness to supply power back.

2.2.1 Power-Matching EV Charging Control

For a specific area of the distribution network (substation) and the loads and distributed generation connected to it, the objective of the aggregator is to control the charge/discharge actions of plug-in electric vehicles in a way that the aggregated load profile meets a target load profile $P_O(k)$ as close as possible. The aggregated power profile $P_{agg}(k)$ is therefore being defined as the difference between the demand and the local (photovoltaics) generation within the area of interest, that means the total power demanded from the grid at time step k.

$$P_{agg}(k) = P_D(k) - P_S(k) + \sum_{i=1}^{N} P_{EV}^{\mathrm{i}}(k) \tag{2.1}$$

In equation 2.1 for a specific time-step k, $P_D(k)$ denotes the inflexible demand, $P_S(k)$ represents the produced photovoltaics power, and $P_{EV}^{\mathrm{i}}(k)$ is the active power absorbed by the ith EV. P_{EV}^{i} can also be negative, indicating that the electric car is delivering power back to the grid (V2G).

Complying a specific target power profile is a way to avoid re-dispatching generators to meet the demand of an area of a distribution network or even an alternative to splitting the network due to congestion.

By matching demand with a specific $P_O(k)$, which has been designated to meet the constraints, it will be possible that operators, utilities to exploit their available capacity in a more efficient way and to decrease or some how eliminate the unnecessary operational related activities.

The objective is;

$$\underset{k\in T}{\text{Min}} \left| P_O(k) - P_{agg}(k) \right| \tag{2.2}$$

2.2.2 VOS Approach

In the Vehicle Originating Signal method, an integrator directly controls a fleet of electric plug-in cars At every time-step, each plugged electric car computes a need-for-charge (*NfC*) signal and a willingness-to-supply (*WtS*) signal, and the transmit them to the integrator. The integrator the harvest those signals and gives back to the cars a charge instructions to. The *NfC* and *WtS* signals exist both, it means that, at a specific time, a car might be willing to either charge or discharge with a specific *NfC* or *WtS*. The integrator at this point makes decision to instruct one of the options regarding on availability of the resources and also the values of other electric cars' *NfC*/*WtS*signal.

The Vehicle Originating Signal method has two obvious benefits;

1. The *NfC* and *WtS* signals are computed by every single EV, that results a very elevated distribution of computation, that in its turn yields elevated conservation of the electric cars holder's data security.

2. The exchange of information is dropped to two scalars per time-step, this says low communication traffic, especially in case if compared with distributed iterative methods. On the other hand, since the *NfC* and *WtS* are already evaluated numbers, the value of messages transmitted could be dropped significantly.

2.2.2.1 EV Need for Charge Signal (*NfC*)

The *NfC* is calculated as function of the needed connection time k^{i}_{av} to charge up to wanted level and the necessary time k^{i}_{req} to reach the wanted state-of-charge *(SOC)*, a percentage value. A threshold process ensures that the electric car limits are completely satisfied. These thresholds are in fact scalar unitless values.

For each electric car i and time-step k, based on its target charge level SOC^{i}_{tar}, battery capacity E^{i}_{max}, existing energy in the battery pack E^{i}_{bat}, permitted charge rate constraint per time-step E^{i}_{rate}, and next departure-time k^{i}_{dep}, it instructs the needed number of time-steps k^{i}_{req} and k^{i}_{av},

$$k^i_{req} = \frac{SOC^i_{tar} E^i_{max} - E^i_{bat}}{E^i_{rate}} \tag{2.3}$$

and available steps to charge the car,

$$k^i_{av} = k^i_{dep} - k \tag{2.4}$$

where k^{i}_{req} can take non-integer values. The departure-time k^{i}_{dep} can be either a time which has been set by the car holder or even an foregone leaving charge time that comes from a statistical analysis. Also, E^{i}_{max} is given to car i; so, various battery pack size to consider for various models or degradation of the battery over time is available as well.

The *NfC* takes into account the limits $C_{QoS} > C_{tar} > C_{full}$, where C_{QoS} represents that the car *must* charge to reach SOC^{i}_{tar}, C_{tar} shows that the car has already get to SOC^{i}_{tar}, and C_{full} shows that the car's battery pack is completely charged. The *NfC* is introduced as:

$$\text{NfC}^i(k) = \begin{cases} C_{QoS}, & \text{if } C_{QoS} \leq \text{NfC}^i_{temp} \\ \text{NfC}^i_{temp}, & \text{if } C_{tar} \leq \text{NfC}^i_{temp} < C_{QoS} \\ C_{tar}, & \text{if } \text{NfC}^i_{temp} < C_{tar} \\ C_{full}, & \text{if } E^i_{bat} = E^i_{max} \end{cases} \tag{2.5}$$

where

$$\text{NfC}^i_{temp} = C_{QoS} \frac{k^i_{req}}{k^i_{av}} \tag{2.6}$$

2.2.2.2 EV Willingness to Supply Signal (WtS)

To calculate the *WtS*, let us consider the limits C_{max} showing the maximum willingness, and C_{noS} showing that the electric car holder is not intended to supply back at all, where $C_{maxS} \gg C_{noS}$. For a given car i and time step k, the *WtS* is a function of the available-time k^{i}_{av}, the leaving time k^{i}_{dep}, and the actual SOC^{i}. To set the *WtS* within

its limits, it is rescaled by C_{maxS} and then shifted by C_{noS}, that results in

$$\mathrm{WtS}^i(k) = \begin{cases} C_{noS}, & \text{if } E^i_{bat} \leq E^i_{min} \text{or } k^i_{av} \leq k^i_{req} \\ C_{maxS} SOC^i \frac{k^i_{av}}{k^i_{dep}} + C_{noS}, & \text{otherwise} \end{cases} \tag{2.7}$$

2.2.2.3 Integrator

At every time-step, the integrator collects the *NfC* and *WtS* signals from the electric cars, determine a charging scenario and assigns the cars to being charged regarding to that. The progress is as follows:

at each step-time (in this case every 15 minutes), P_{agg} is updated as a net active power in the simulation's unique node

$$\mathrm{P}_{agg}(k) = P_D(k) - P_S(k) \tag{2.8}$$

then, the aggregator sorts down cars by *NfC*, as long as $P_{agg}(k) < P^{max}_{agg}$ and NfC = C_{QoS}, the cars with NfC = C_{QoS} will be started to charging.

2.2.2.4 Excessive Power Available with Respect to the Target

If excessive power is available respect to the target, i.e. $P_{agg}(k) \leq P_O(k)$, cars with $C_{full} <$ NfC $< C_{QoS}$ will be selected and charged. At this point Vehicle-Originating-Signals suggests continuous and discrete charging modes:

continuous case Allocates available power to cars proportional to their *NfC*:

$$P^i_{EV} \leftarrow \min\left(P^{max}_{EV}, \frac{\mathrm{NfC}^i}{\sum^S \mathrm{NfC}^i}\left[P_O(k) - P_{agg}(k)\right]\right) \tag{2.9}$$

and again the $P_{agg}(k)$ will be updated:

$$¶_{agg}(k) = P_{agg}(k) + \sum^S P^i_{EV}(k) \tag{2.10}$$

integer case allocates available power to cars with highest *NfC*:

$$P_{EV}^{i} \leftarrow P_{EV}^{max} \tag{2.11}$$

$$P_{agg}(k) = P_{agg}(k) + P_{EV}^{max} \tag{2.12}$$

2.2.2.5 Power Available Lower Than Target

in case that $P_O(k) < P_{agg}(k)$, the same as having shortage of active power integrator sorts remaining cars by *WtS* in descending order:

$$S \leftarrow \text{EVs with} WtS > C_{noS} \text{ and discharge EVs} \tag{2.13}$$

continuous case Allocates required power to cars proportional to *WtS*:

$$P_{EV}^{i} \leftarrow \min\left(P_{EV}^{max}, \frac{\text{NfC}^{i}}{\sum^{S}\text{NfC}^{i}}\left[P_O(k) - P_{agg}(k)\right]\right) \tag{2.14}$$

then

$$P_{agg}(k) = P_{agg}(k) + \sum^{S} P_{EV}^{i}(k) \tag{2.15}$$

Integer case Allocates the power to cars with greater *WtS*

$$P_{EV}^{i} \leftarrow -P_{EV}^{max} \tag{2.16}$$

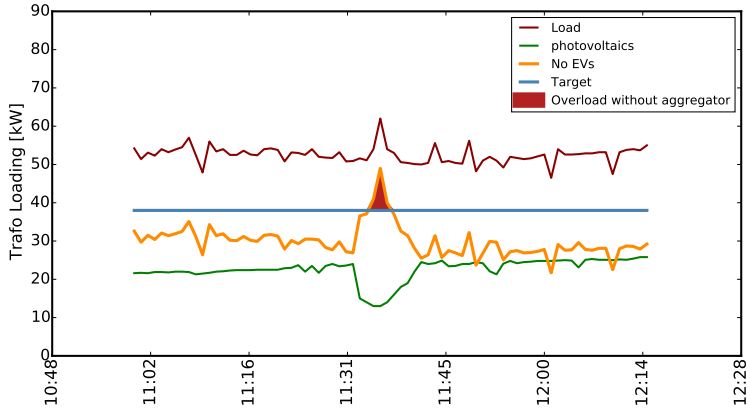
$$P_{agg}(k) \leftarrow P_{agg}(k) - P_{EV}^{max} \tag{2.17}$$

To fulfill the progress in real-time, cars and integrator must finish the progress in less time than the duration of one time-step k that is 15 minutes and leave an idle. Every car needs to calculate two scalar values and implement the charging discharging action. however given that this is done by each EV separately, its impact in the overall solving time is insignificant. The integrator, implements up to two sort operations and progresses every car's received signal. Thus, the solving time depends on the number of electric cars fleet and its upper limit is proportional to the number of cars N as well.

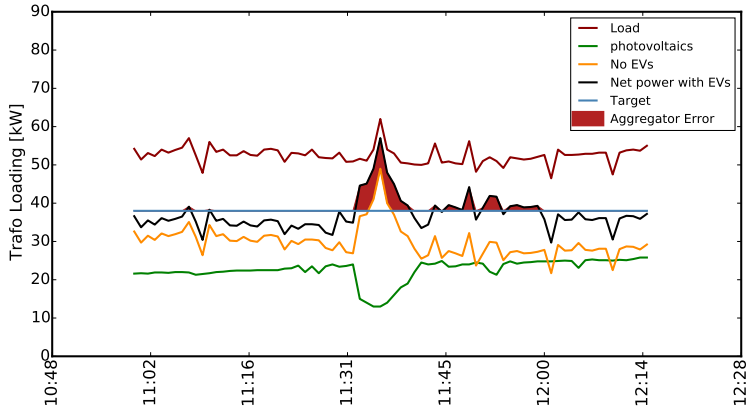
2.2.2.6 Drawbacks

- This method however assumes that the internal generation and loads can be aggregated into a single node, because it concentrates on a limited section of a distribution grid, so it is not valid in case one wants to model the EVs in a vast area in which the distances are important, and the grid parameters have a meaningful role. As a result the effect of such method on physical network e.g. power losses and lines loading, and economic analysis e.g. defining *local marginal prices* are impossible to evaluate.

- Given these scenarios are being modeled in a distribution grid, where variation in demand and distributed generation have tangible effects on the measurement metrics, 15 minutes step-size seems to be insecure. For this reason there is a lack of instantaneous unpredictable changes in load and solar generation.



(a) Operation with no EV



(b) Aggregator handling in presence of EVs

Figure 2.4 – Load profile aggravation caused by wrong decision made by aggregator.

2.2.3 VOS with additional local control

VOS models in a real-time smart grid simulation, likely fails. Also is based on aggregated load and generator in a unique node. All components are assumed to be connected in one single node omitting all other constructive elements, e.g., transformers and power lines, that should be taken into account, in a real grid analysis. The original work does not provide any indication on the effect of such an algorithm on voltage deviation, branch elements loading and power losses.

To deal with this issue first we add disturbances both in load side and local generation, as it is in the randomly nature of it. In addition we redefined the models and simulation, closer to the reality, modeling the grid in with several substation, transformers and other power system equipments such as circuit breakers, power lines and etc. We use numerous loads with different profile categories.

To do this simulation we get use of introduced co-simulation and model physical network in PowerFactory environment.

2.2.3.1 Load Profiles

As already mentioned, this work presents different standard load profiles, each one contains seasonal consumption behaviors, also over the week-days.

For this study case

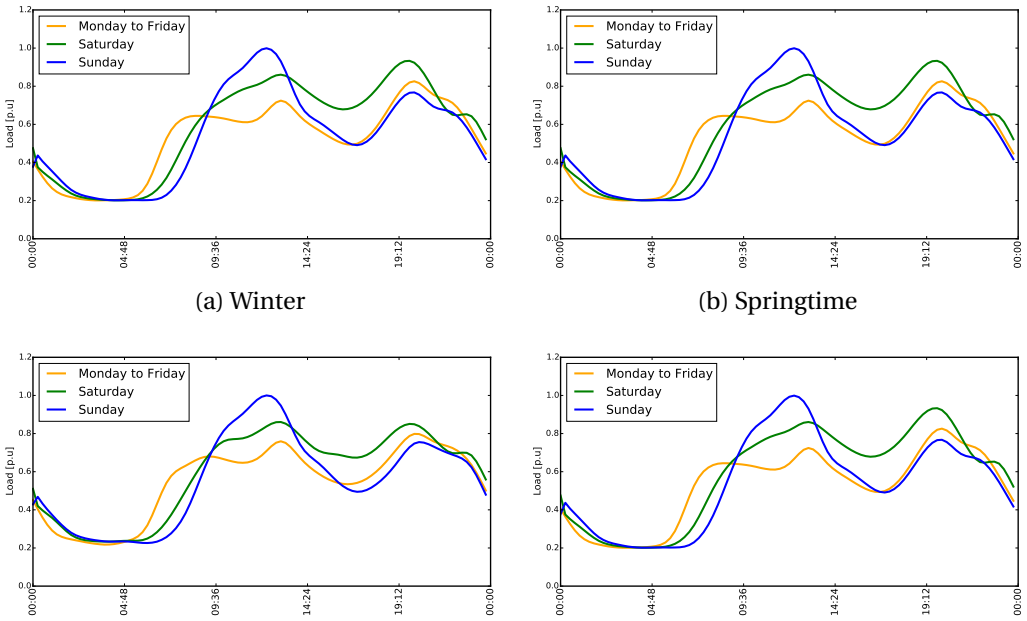
1. **Householder** → Fig. 2.5b

2. **Commercial**

   - **General** → Fig. 2.6b
   - **Evening** → Fig. 2.7b
   - **8 to 18** → Fig. 2.8b
   - **continuous**
     - Type A → Fig. 2.9b
     - Type B → Fig. 2.10b
   - **Backery** → Fig. 2.11b
   - **Weekend operation** → Fig. 2.12b

3. **Agriculture**

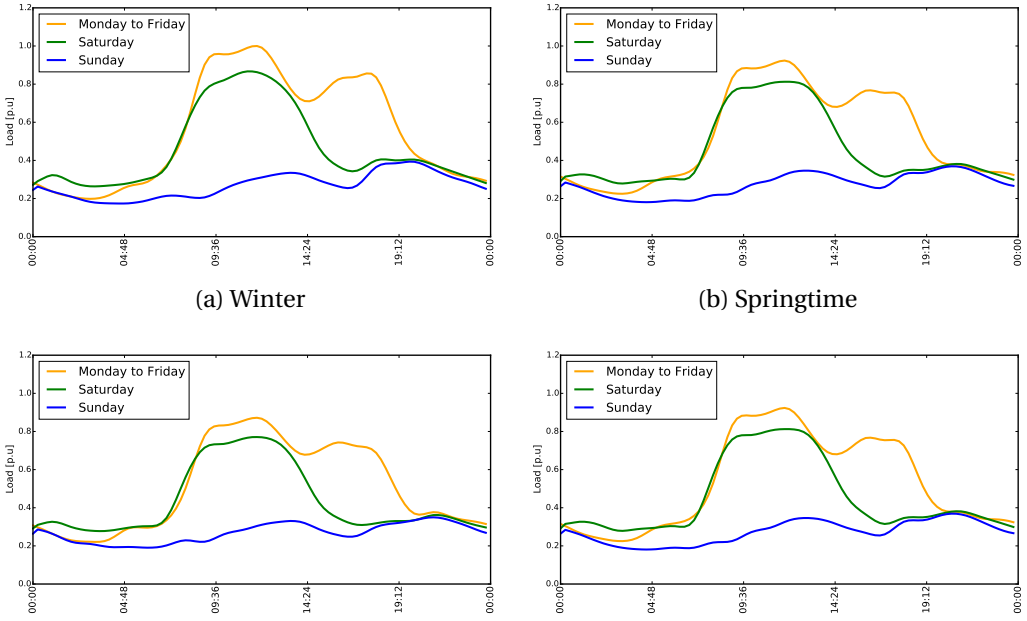
   - **General** → Fig. 2.13b
   - **Dairy farming** → Fig. 2.14b
   - **Other** → Fig. 2.15b



(a) Winter (b) Springtime

(c) Summer (d) Fall

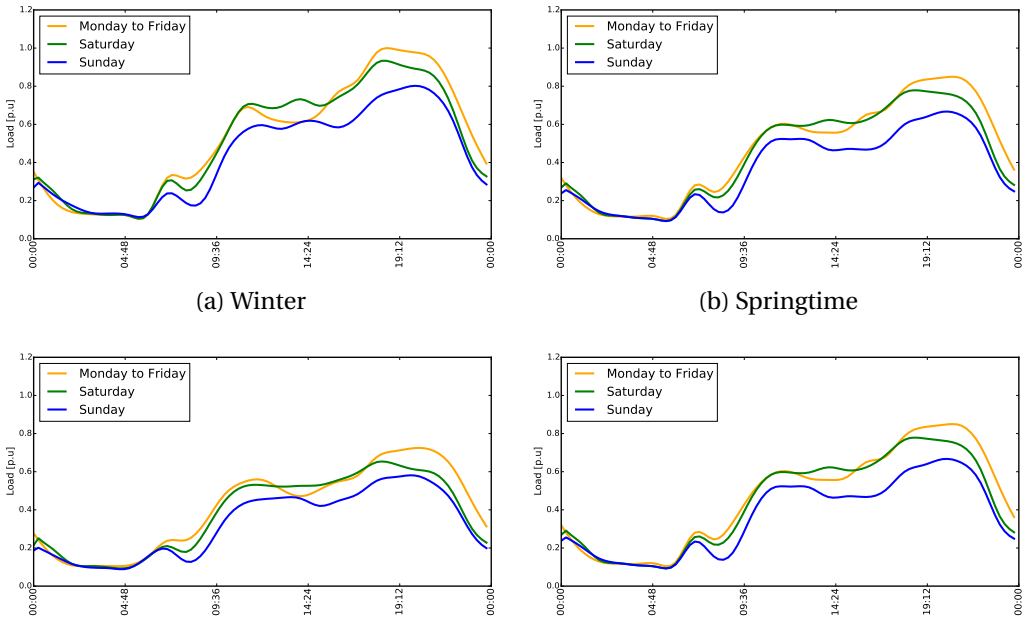
Figure 2.5 – Householder standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

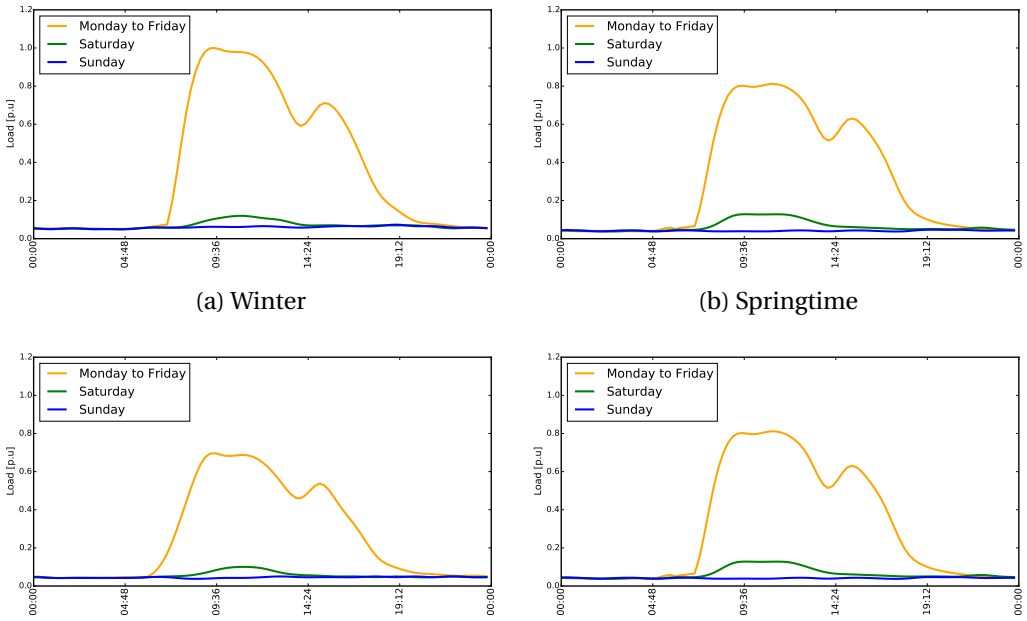
Figure 2.6 – Commercial general standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

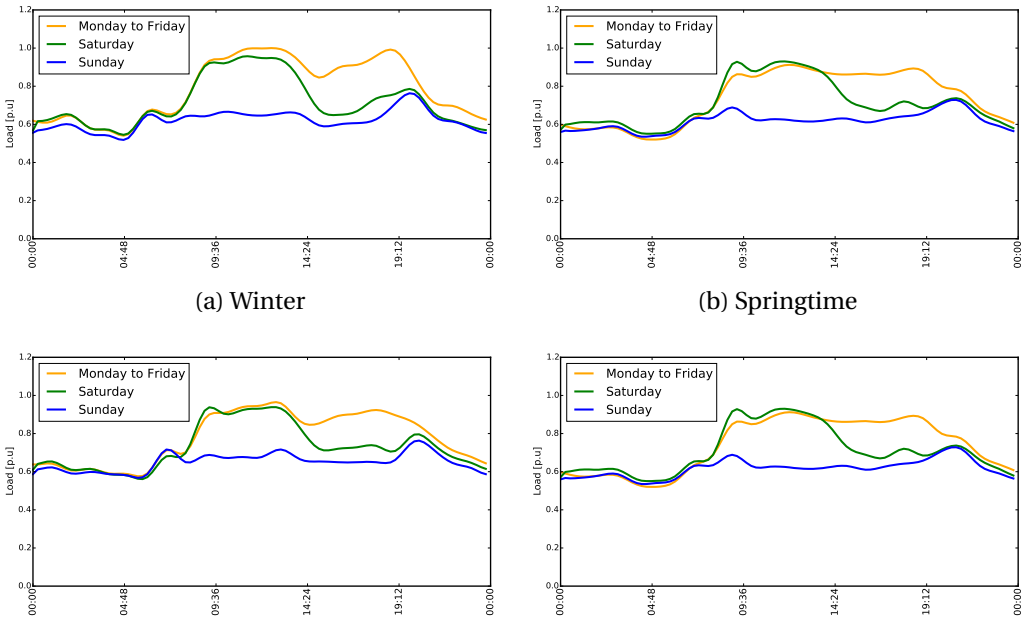
Figure 2.7 – Commercial evening standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

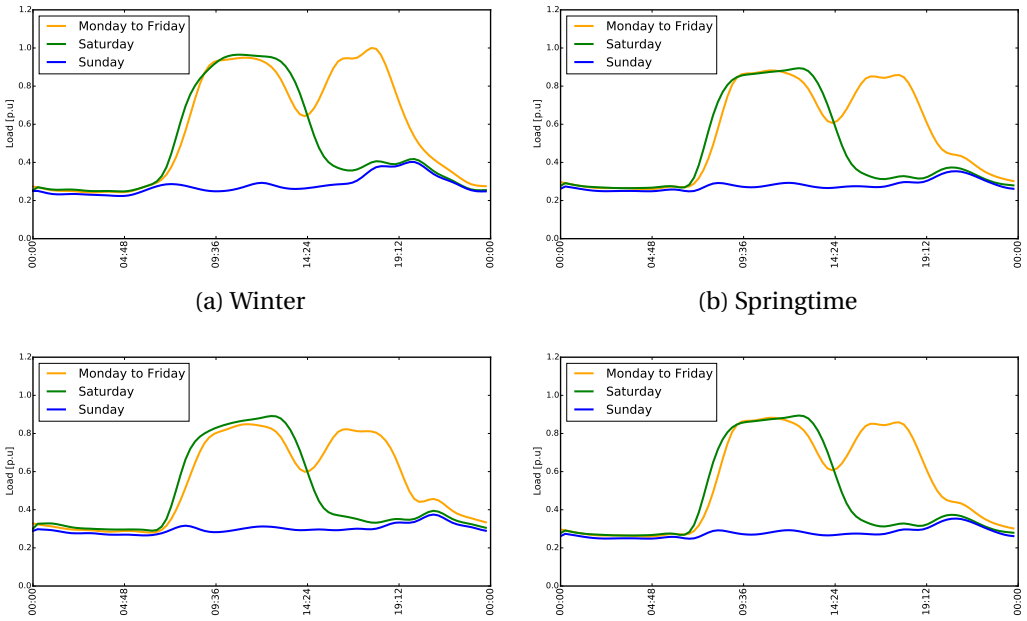
Figure 2.8 – Commercial 8 to 18 standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

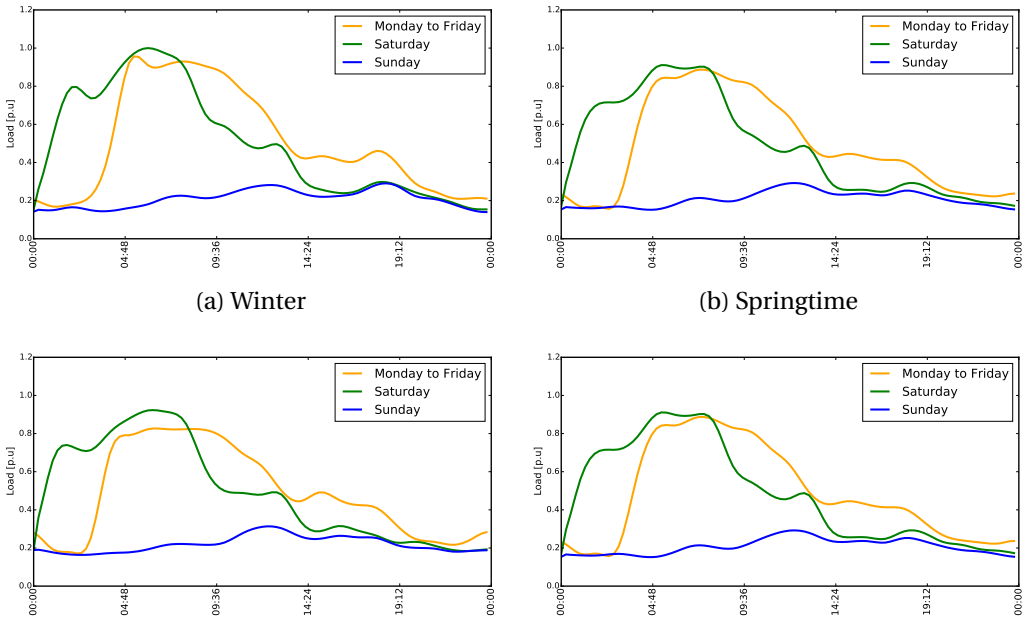
Figure 2.9 – Commercial continuous standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

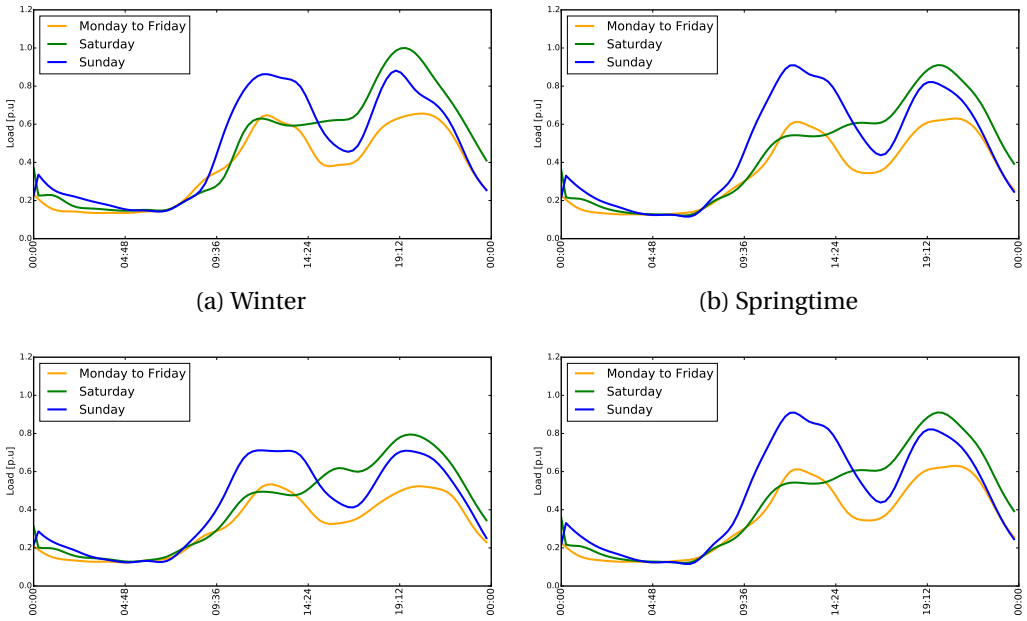
Figure 2.10 – Weekend commercial standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

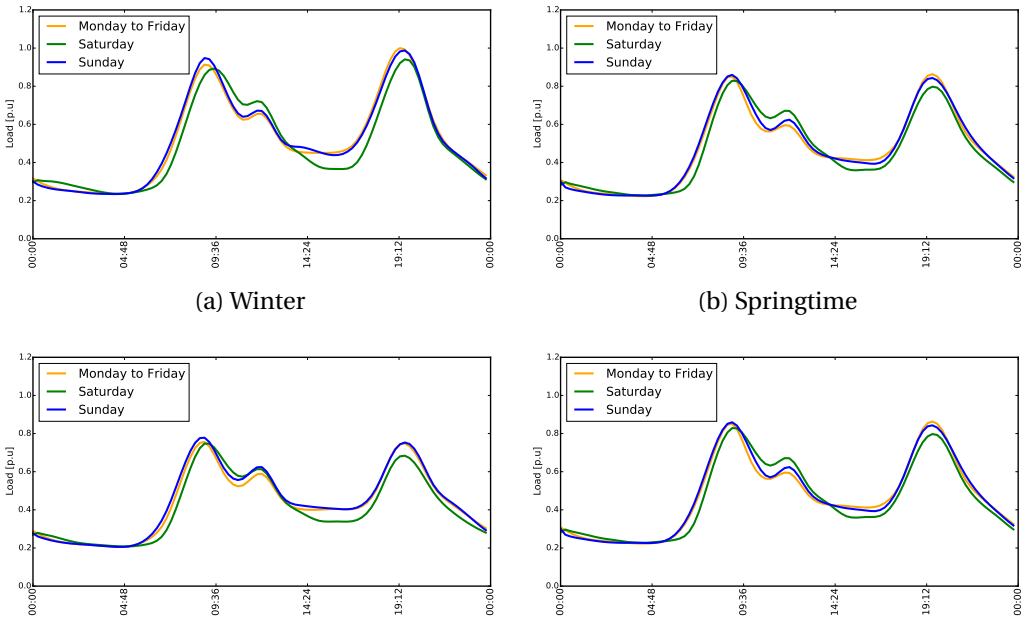
Figure 2.11 – Backery standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

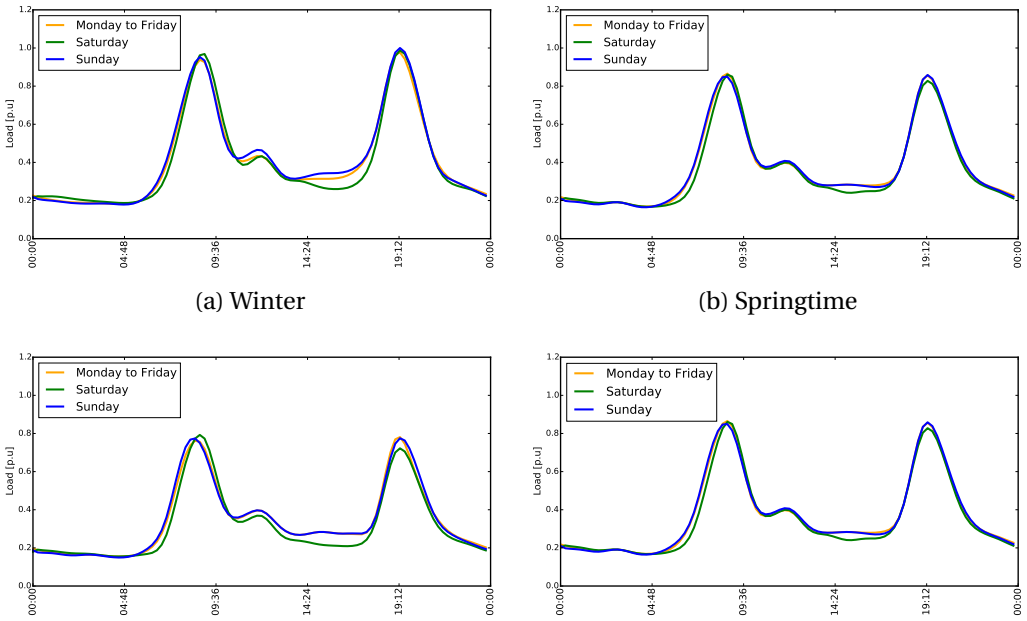
Figure 2.12 – Weekend operation standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

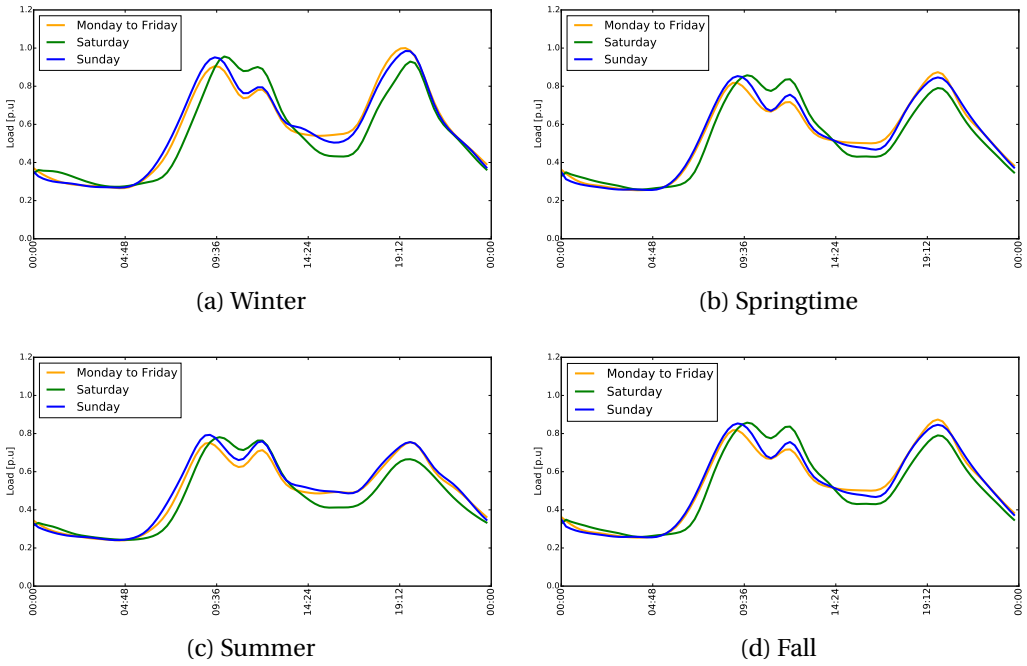
Figure 2.13 – General agriculture standard load profile in different seasons.



(a) Winter (b) Springtime

(c) Summer (d) Fall

Figure 2.14 – Dairy farm standard load profile in different seasons.



(a) Winter

(b) Springtime

(c) Summer

(d) Fall

Figure 2.15 – Other types of farms standard load profile in different seasons.

For our study case 2034 load with the mentioned profiles have been selected from the *template library* of the co-simulator.

2.2.3.2 Photovoltaics

The summary of photovoltaics used in this simulation are given in Table 2.1.

Table 2.1 – Photovoltaics parameters used in simulation.

Panel per inverter	**Apparent power [kVA]**	**Power factor**	**Tilt angle [deg]**	
Distribution	1 ÷ 15	63, 20	20, 0.4	0.75 ÷ 1.2

In the *controller class* of simulation, irradiation data regarding time and location of analysis is calculated. Figure 2.16 shows solar irradiation of Munich in full sunshine, which is maximum on July 21th and minimum on January 5th. However these curves are subjected to the cloud presence.

```
In [1]:
controller.grid_sim.irradiation.calculation(
  SIMULATION_START.year, SIMULATION_START.month,
  SIMULATION_START.day, longitude = 48.1351, Latitude
  = 11.5820)
controller.grid_sim.irradiation.show()
```

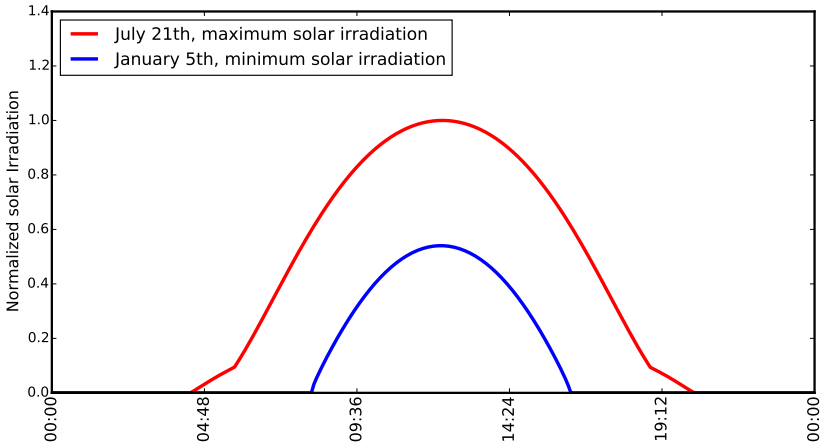


Figure 2.16 – Munich solar irradiation, maximum and minimum with no cloud

The real sunny hours in Munich considering presence of the clouds according to **Weather/Germany** are embedded to the controller class.

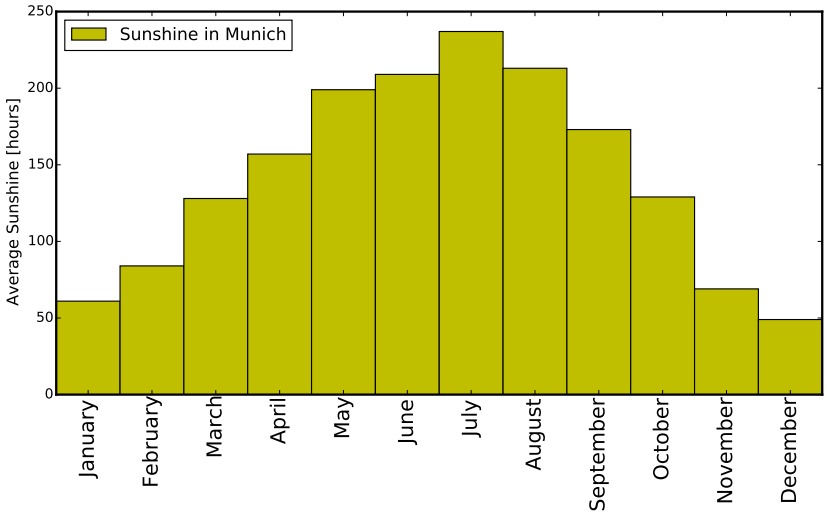


Figure 2.17 – Munich real solar irradiation, obtained by data accumulated of more than 40 years

Let us define a *noise* function in the class of *control*, that makes a comparison between the perfect irradiation as an example is shown in Figure 2.16 for specific time, for the simulation period and statistical experienced irradiation depicted in Fig. 2.17, and defines a probability distributed model which on its turn shows the cloud presence probability as a power generation noise. This random noise then is being added to the photovoltaics.

Consequently the solar generation noise is function of season as it is in reality.

In fact it is just enough to insert solar noise command:

```
In[1]:
controller.solar_noise(level = default)
```

A generic solar generation in presence of noise (clouds) is shown in Figure 2.18.

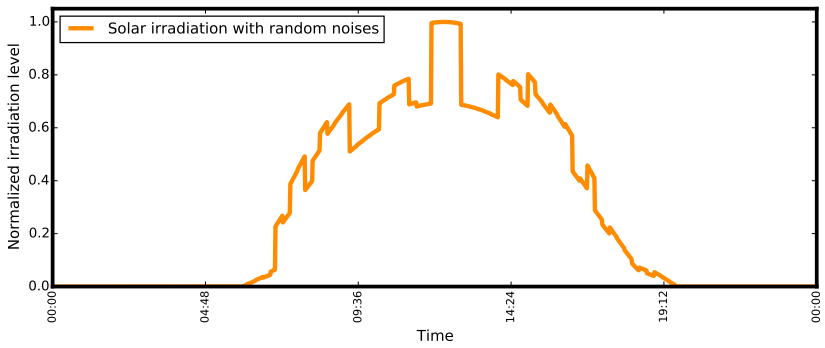


Figure 2.18 – An example of solar irradiation with noise

```
def get_irradiation(self, PV_RES_object, study_time,
 epoch_start, epoch_stop, step_size, noise):
    actual_irradiation = {}
    random_PV = PV_RES_object
    irradiation_list = []
    time_list = []
    while epoch_start < epoch_stop:
        PV_RES_object.inveff = noise.curval
        study_time.SetTimeUTC(epoch_start)
        irradiation = random_PV.pgini_a
        actual_irradiation[epoch_start] = irradiation
        irradiation_list.append(irradiation)
        time_list.append(dt.datetime.fromtimestamp(
          epoch_start).strftime('%H:%M'))
```

```
13          epoch_start += step_size
14      datetime_format_time = dt.datetime.fromtimestamp(
          epoch_start).strftime('%Y-%m-%d')
15      d = pd.DataFrame(irradiation_list, columns=['Level'])
16      d['Time'] = pd.Series(time_list)
17      ds = d.plot(x='Time', y='Level', title='Solar
          Irradiation Level')
18      ds.set_ylabel('Solar Generation on '+
          datetime_format_time)
19      return irradiation_list
20
21  def power_prediction(self,pf_app, substat, study_time,
      solar_curve, epoch_start=1338940800, epoch_stop
      =1339027200, step_size=60):
22      generator = pf_app.app.GetCalcRelevantObjects('*.
          ElmPvsys')[0]
23      demand = substat.GetContents('inflexible_demand m.
          ElmLodlv')[0][0]
24      inflexible_load = demand.GetContents('Partial LV-Load
          ph0.ElmLodlvp')[0][0]
25      noise = demand.GetContents('noise.ElmLodlvp')[0][0]
26      solar_predict = []
27      inflexible_predict =[]
28      inflexible_constant = []
29      time_list = []
30      generator.inveff = np.mean(solar_curve.vector)
31      while epoch_start < epoch_stop:
32          study_time.SetTimeUTC(epoch_start)
33          solar_predict.append(generator.pgini_a)
34          inflexible_constant.append(inflexible_load.plini_a
              )
35          inflexible_predict.append(inflexible_load.plini_a
              - generator.pgini_a)
36          time_list.append(dt.datetime.fromtimestamp(
              epoch_start).strftime('%H:%M'))
37          epoch_start += step_size
38      noise = self.noise_mean_value(noise, study_time)
39      noise_mean = np.mean(noise)
40      prediction = [i + noise_mean for i in
```

```
      inflexible_predict]
41    my_dict = dict(solar_predict = solar_predict,
42    inflexible_constant=inflexible_constant,
43    prediction_plus_noise_mean=prediction)
44    d = pd.DataFrame(my_dict)
45    d['Time'] = pd.Series(time_list)
46    ds = pd.DataFrame(noise, columns=['Level'])
47    ds.plot( y='Level', title='Noise')
48    d.plot(x='Time', title='Power Prediction')
49    return dict(prediction=prediction,noise=noise,
       solar_predict=solar_predict,inflexible_part=
       inflexible_predict)
50
51 def noise_mean_value(self, noise, study_time, epoch_start
   =1338940800, epoch_stop=1339027200, step_size=60):
52    time_list = []
53    noise_value =[]
54    study_time.SetTimeUTC(epoch_start)
55    while epoch_start < epoch_stop:
56        noise_value.append(noise.plini_a)
57        study_time.SetTimeUTC(epoch_start)
58        time_list.append(dt.datetime.fromtimestamp(
           epoch_start).strftime('%H:%M'))
59        epoch_start += step_size
60    return noise_value
```

2.2.3.3 Lines

Power lines used in this simulation are as follows,

Type	Rated current [kA]	Length [km]	R (+seq.)[ohm]	R (+seq.) [ohm]
NA2XSEY 3*240	041	0.07 ÷ 11	0.017 ÷ 0.348	0.004 ÷ 0.3

2.2.3.4 Transformers

Parameters of the distribution transformer are reported below,

Table 2.2 – Transformer

Type	Rated power [MVA]	HV-side [kV]	LV-side [kV]	Cupper losses [kW]
Distribution	1 ÷ 15	63, 20	20, 0.4	0.75 ÷ 1.2

In presence of load and local generation variations (noises) there is a need for an additional control, assigned to the vehicles. The single noise is redefined to several feeders and transformers, each one with a specific *target* regarding the nominal power of the transformer P_{nom} and the number of EVs introduced for the area of supply.

For the introduction of the local voltage control, there is a difference between charging instruction and target action. Being affected by random noises, a powerful noise combination of $N_L(k)$ and $N_S(k)$) at step k may lead the aggregator to make a wrong decision, consequently worsening indicators for the next 15 minutes of simulation. Thus we use a time consumption pattern $P_P(k)$ in every step, and define an error threshold ΔP to detect strong noises. Each EV in step k receives a target action indicating whether it must be charged, discharged or stay stand-by.

The control process becomes:

Algorithm: aggregator + local control

$\boldsymbol{N_L(k)}$ and $\boldsymbol{N_S(k)}$, the noise added to the load profile and solar power generation will be recorded and repeated for all 4 scenarios

$P_D(k) \leftarrow P_D(k) + N_L(k)$

$P_S(k) \leftarrow P_S(k) + N_S(k)$

$\boldsymbol{P_{agg}(k)}$ becomes aggregated load profile containing random noises, close to the reality.

$P_{agg}(k) \leftarrow P_D(k) - P_S(k)$

a comparison with the $\boldsymbol{P_p(k)}$ is made to prevent excessive errors more than $\boldsymbol{\Delta}_P$.

If $\left|P_{agg}(k) - P_P(k)\right| > \Delta_P, P_{agg}(k) \leftarrow P_P(k)$

Sort EVs by NfC in descending order

While $P_{agg}(k) < P_{agg}^{max}$ and NfC = C_{QoS}

- Charge EVs with NfC = C_{QoS}

If $P_{agg}(k) \leq P_O(k)$

1. $S \leftarrow$ **Select** EVs with $C_{full} <$ NfC $< C_{QoS}$
2. $TargetAction \leftarrow$ Charge

    - Integer case:
      (Allocate available power to EVs with highest NfC)
      For EV i in S:

      If $P_O(k) \leq P_{agg}(k)$, **Break**
      $P_{EV}^{i} \leftarrow P_{EV}^{max}$
      $P_{agg}(k) \leftarrow P_{agg}(k) + P_{EV}^{max}$

Else (meaning $P_O(k) < P_{agg}(k)$)

1. **Sort** remaining EVs by WtS in descending order
2. $S \leftarrow$ **Select** EVs with WtS $> C_{noS}$
3. $TargetAction \leftarrow$ Discharge

    - Integer case:
      (Allocate required power to EVs with highest WtS)
      For EV i in S:

      If $P_{agg}(k) \leq P_O(k)$, **Break**
      $P_{EV}^{i} \leftarrow -P_{EV}^{max}$
      $P_{agg}(k) \leftarrow P_{agg}(k) - P_{EV}^{max}$

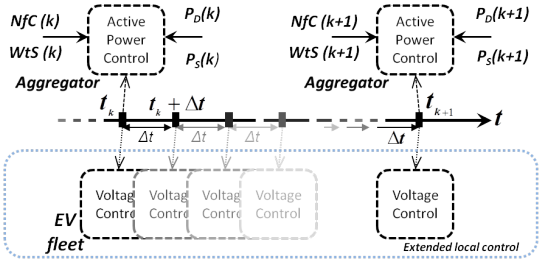


Figure 2.19 – VOS control with additional local voltage control

The modeled grid has several feeders, each one to be feeding a number of residential and commercial consumer, and dispersed photovoltaic panels. For the lines, trans-

formers, loads and photovoltaic panels modeling, the existing standard models from PowerFactory library are used. Load profiles also, have been chosen from existing residential and commercial categories. The simulation time is set for 24 hours from July 21^{th} midnight to July 22^{th} midnight. The location of the grid is in Munich area, and solar irradiation for the photovoltaics complies with the simulation date and time and place. The simulation has been performed in different scenarios:

- $No\ EVs$: grid without EVs.
- $No\ Control$: the same grid and configuration in the presence of EVs, without any control on the charging procedure.
- $Preceding\ VOS$: applying VOS to control EV charging.
- $VOS\ plus\ Voltage\ Control$: extended VOS with adding local control on voltage.

2.3 Results

2.3.1 Voltage

Figure 2.20 shows the LV side (0.4 kV) voltage deviation of a generic feeder. It can be seen that the presence of EVs causes voltage reductions in the network, mitigated by the VOS application and further limited in their range of variation by the additional voltage control.

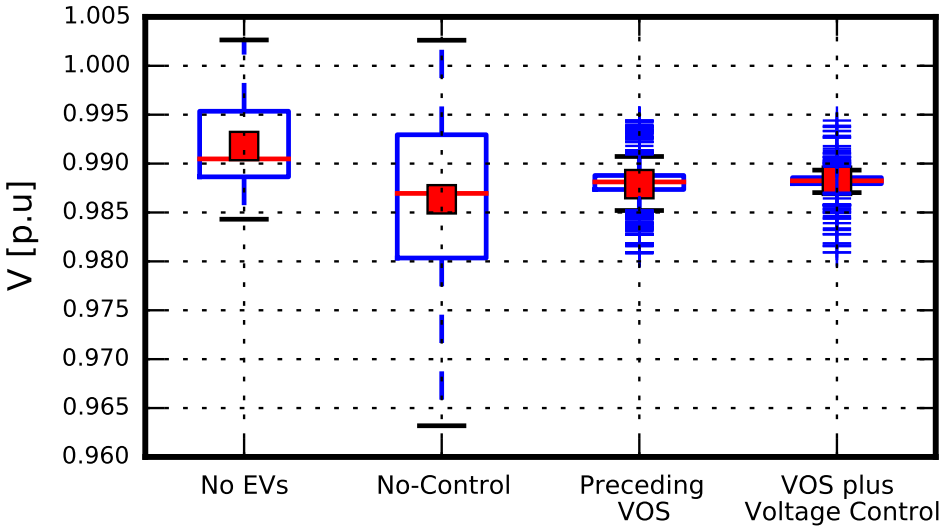


Figure 2.20 – Voltage deviation in a generic transformer

2.3.2 Loading

For another randomly selected feeder, the transformer loading is studied in Figure 2.21 The target active power is set to 85 per cent of the transformer nominal power and the results are shown in per cent of the transformer loading.

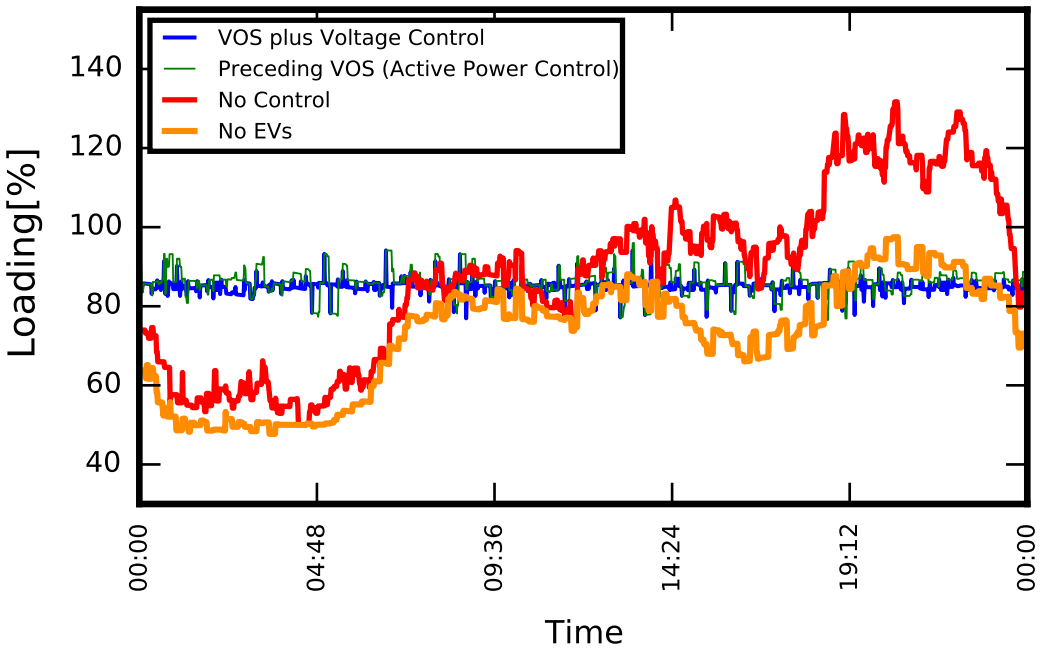


Figure 2.21 – Loading for a generic transformer

2.3.3 Visual results

Fig. 2.22 illustrates the resulting quick-overview videos for this experiment, in several versions. The test grid configuration and its presentation in geographical map and each feeding area can also be seen here.

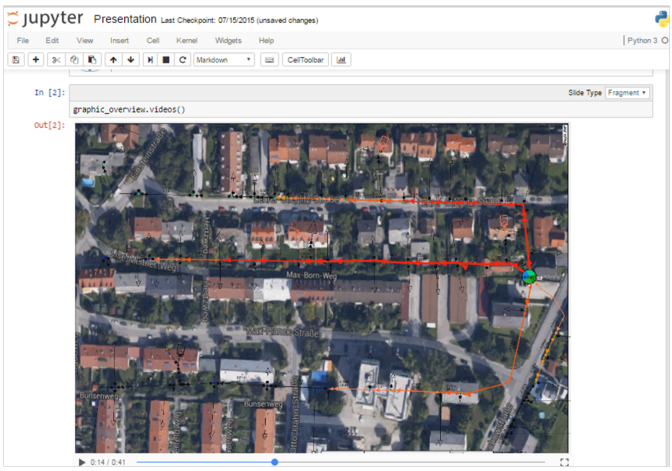


Figure 2.22 – Quick overview on Google Map platform.

Open source OpenStreetMap application is available within the simulator framework and as such the quick overview:

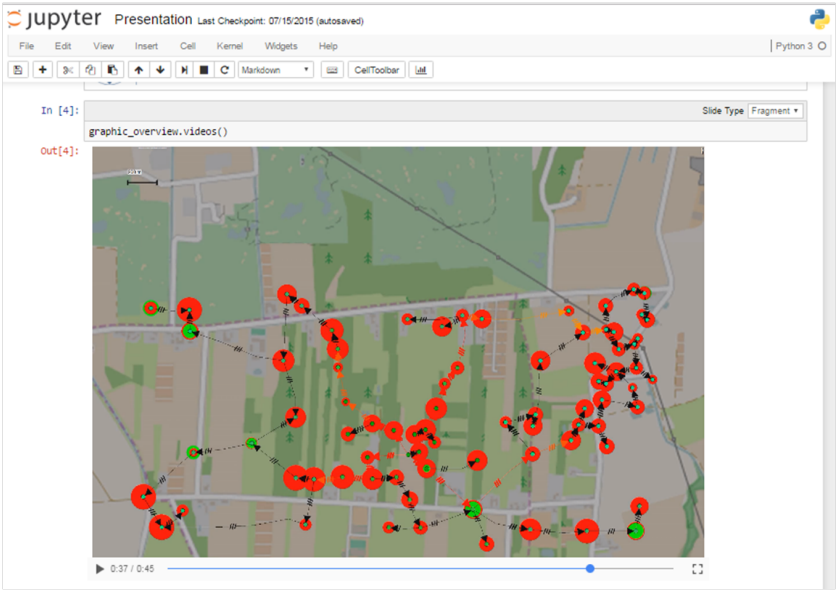


Figure 2.23 – Animations made as quick overview containing several metrics results, visually.

For the same simulation heatmap analysis is available as well:

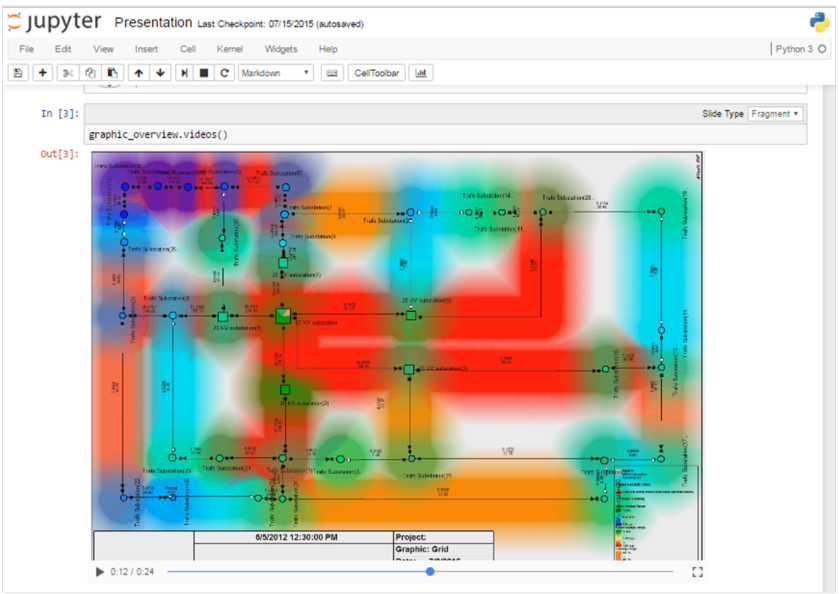


Figure 2.24 – Heatmap on MV layout.

Individual images taken for every time step from each spot of interest in simulation are also included in Graphic Board folder:

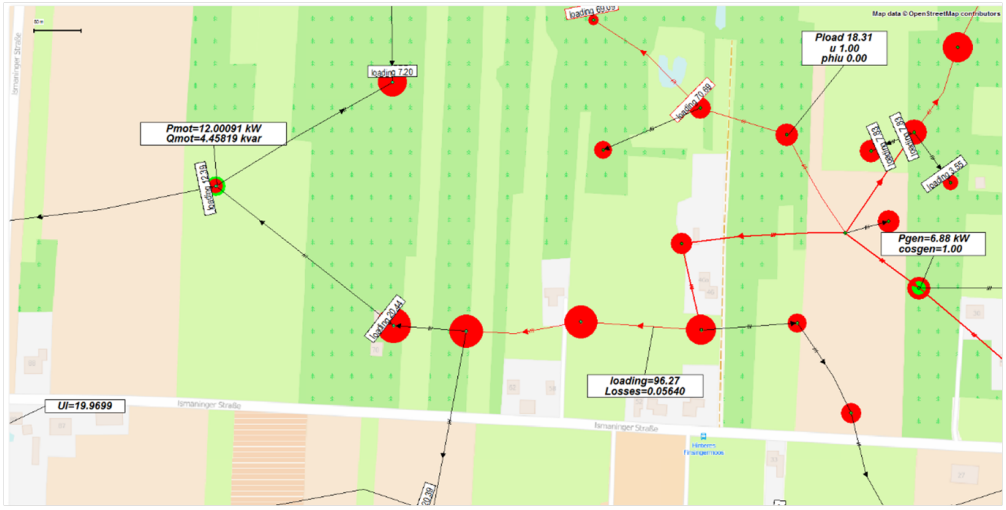


Figure 2.25 – Separate images in Graphic Board folder.

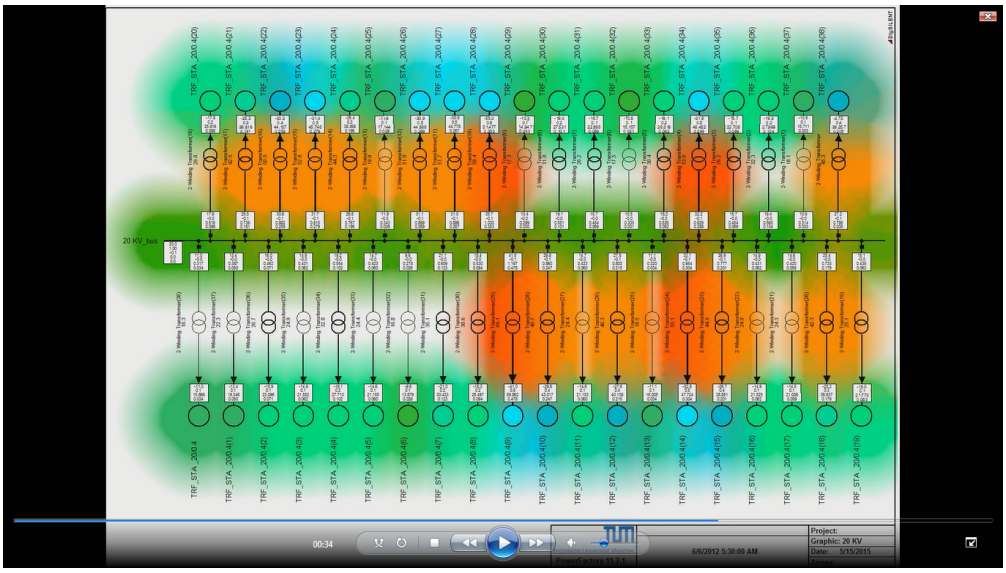


Figure 2.26 – Visual result for transformers.

2.4 Conclusion

The voltage quality, which had been corrected by original VOS active power control, is still improved by adding voltage control. The VOS charging approach also keeps the existing grid within nominal condition, especially in peak consumption hours.

In the loading feeder diagram, the advantage of adding local control is more evident, i.e. instant step increasing/decreasing loads are being responded by EV charger controllers' reaction. The results show that, in presence of EVs within the DN, there will be a random power consumption, heavily based on EV holders behavior. Without control, available power equipment will not be responsive and thermal condition of the lines and transformers may be violated. This issue must be managed in an appropriate framework, and here it is shown that the VOS approach is a potential solution to reduce voltage variations. Moreover using VOS smart charging approach with additional local control, can potentially optimize the use of existing power equipment such as transformer and lines. The result visualization introduced in this work, provides results in different terms. Instantaneous values of loads, local generation, line loading, power losses and voltage magnitude can be observed in quick-overview videos, in several versions, this could be done for all constructive elements separately, for instance all substations, feeders or a single transformer of interest after every simulation.

3 Different study-cases implemented by co-simulator

As already has been discussed, this work aimed to create a powerful tool to address the *smart grids'* simulation and analyses necessities, this is valid about all domains and fields of modeling. In this chapter some different study cases being modeled and launched with the proposed co-simulation framework are shown.

3.1 Short Circuit

Fault analyses are absolutely tied with the electrical systems, as well as smart grid and distribution network. In order to provide flexibility to co-simulator, a **Short circuit** module is include.

3.1.1 Short circuit module

This module provides access to PowerFactory fault functions, including different types of faults in the considered grid. One can set the fault type, location, transformer and generator's neutral connection in the user-interface and run the simulation easily. The results will be obtained in sub-transient, transient and steady-state, and are achievable numerical and graphically.

Here it is simulated a three-phase short circuit in the grid's transformers by random and iterated this simulation for 500 times. In each iteration a transformer in the grid is selected at random, and fault is simulated in two scenarios:

1. **Directly Earthed Neutral**
2. **Isolated Neutral**

3.1.1.1 Initial Fault Current

The results in terms of initial fault current are indicated in Fig. 3.1, showing the spots of initial current in **A**. As it can be easily seen, in the directly connected transformer's neutral case, the current is higher as the considered transformer is located in a closer layer to the external grid.

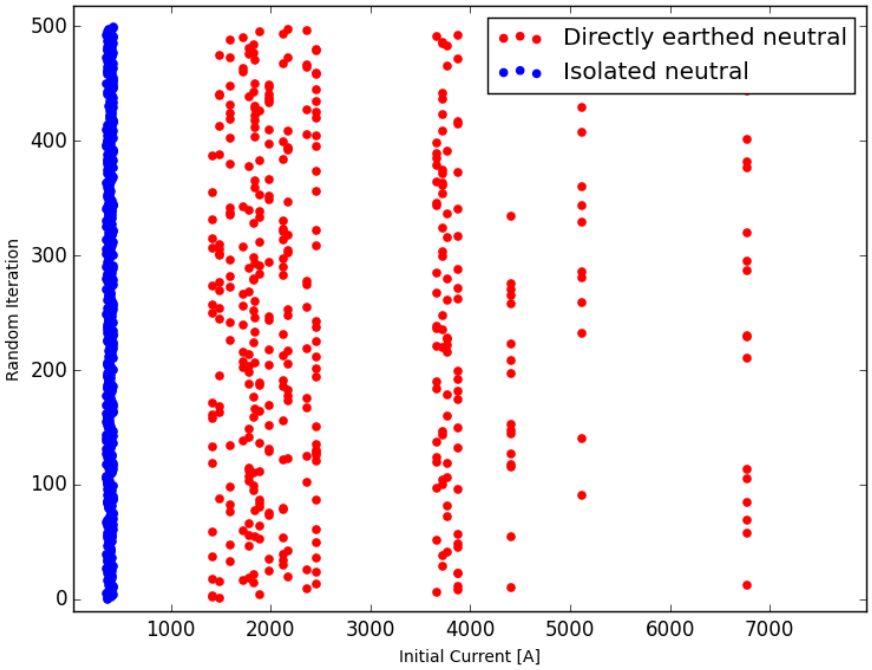


Figure 3.1 – Fault initial current, simulated 500 times in substations randomly twice, regarding neutral connection.

3.1.1.2 Peak Faut Current

Fig. 3.2 shows the results of maximum current.

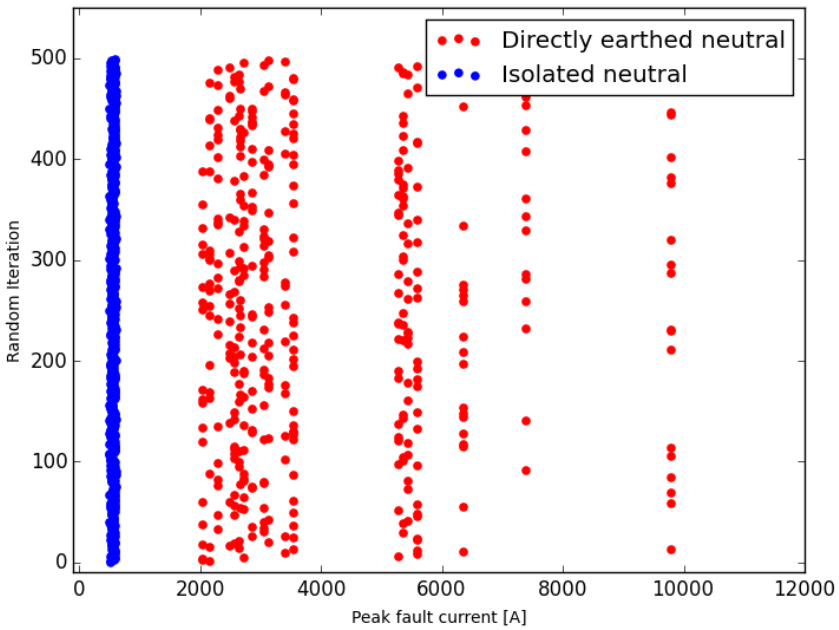


Figure 3.2 – Peak current, simulated 500 times in substations randomly twice, regarding neutral connection.

3.1.1.3 Neutral Connected through Series R-X

This time renewable energy resources are connected to the earth through a series resistance-inductance and the value of these elements will be varied in each random selection stage. Again fault is occurring in different buses. The results are provided in Fig. 3.3.

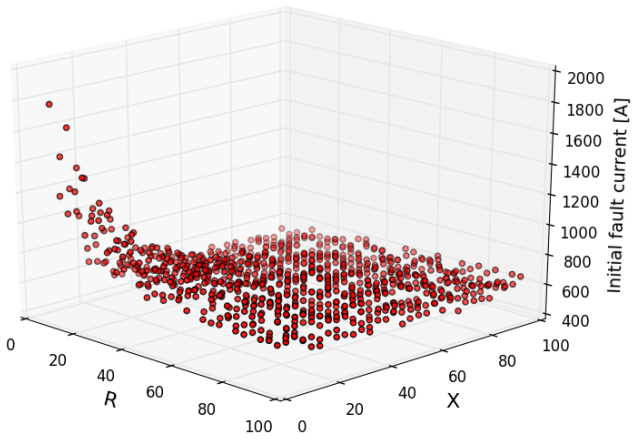


Figure 3.3 – Initial current, in case of generators earthed with R-X.

3.2 Optimization

Optimization problems also can be solved in this framework. This part is dedicated to two study cases that are simulated in the test grid.

3.2.1 Optimal Distributed Generator Placement

A simulation is launched to determine which node is the optimum to host a given distributed generator, the optimization is as following;

$$\text{Minimize} \quad \sum_{d\in 7}\sum_{h\in 24}\left(\sum_{l=1}^{N_L} P^{l}_{loss}(h,\Gamma)+\sum_{\tau=1}^{N_T} P^{l}_{loss}(h,\Gamma)\right) \tag{3.1}$$

$$\text{subject to} \quad \sum_{i=1}^{N_L} d_i(k)-\sum_{j=1}^{N_G} g_j(k)=0, \qquad \forall \quad \text{time} \quad k \tag{3.2}$$

$$P^{min}_{DG}\le P^{i}_{DG}\le P^{max}_{DG}, \qquad \forall \quad \text{i} \quad \in N_B \tag{3.3}$$

$$I^{L}_{j}\le I^{L_{max}}_{j}, \qquad \forall \quad \text{j} \quad j\in N_L \tag{3.4}$$

The optimization problem (3.1) is due to minimize active power losses in all distribution lines $\boldsymbol{N_L}$ and transformers $\boldsymbol{N_T}$, that are in turn function of time $\boldsymbol{h}$ and day of the week $\boldsymbol{d}$, as well as system configuration $\boldsymbol{\Gamma}$, subject to (3.2) which expresses power balance at every time step, (3.3): the capability curve of the distributed generator, and

also to (3.4): the maximum loading threshold of lines.

The results in terms of power losses minimization and voltage deviation are reported in Fig. 3.4 and Fig. 3.5:

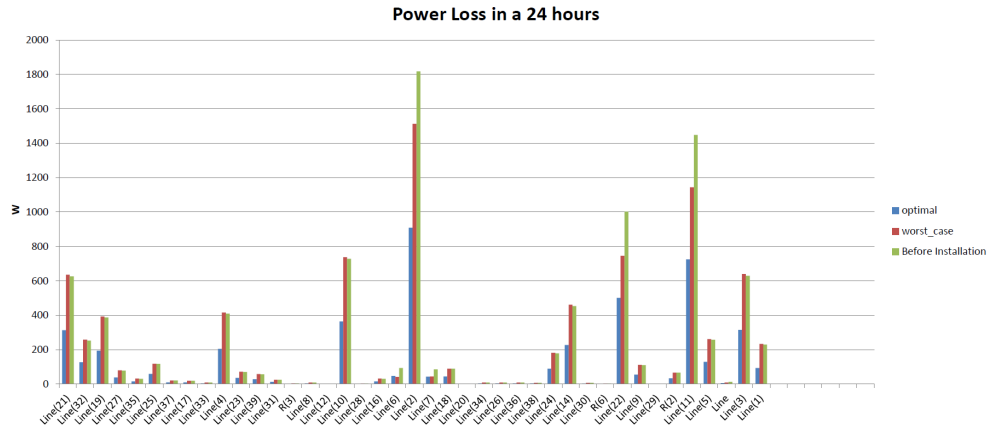


Figure 3.4 – Distributed generator optimal placement to reduce losses

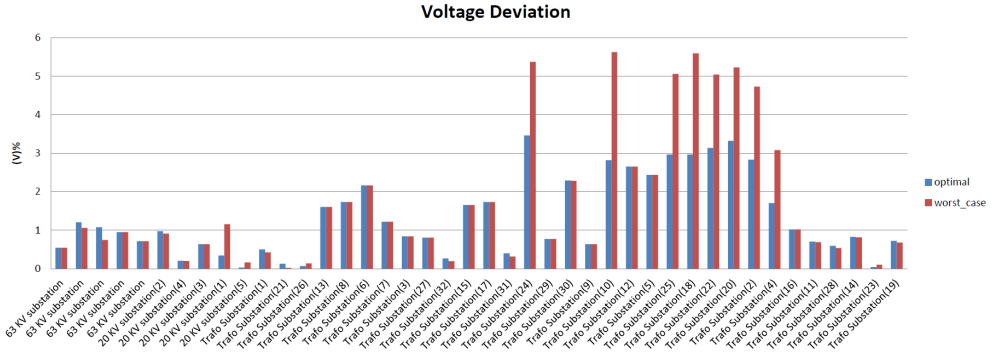


Figure 3.5 – Distributed generator optimal placement effect on voltage deviation

As already discussed from this framework of co-simulation result visualization is driven, from the geographical view, by the location of the optimum node to distributed generator placement, as shown in Fig. 3.5;

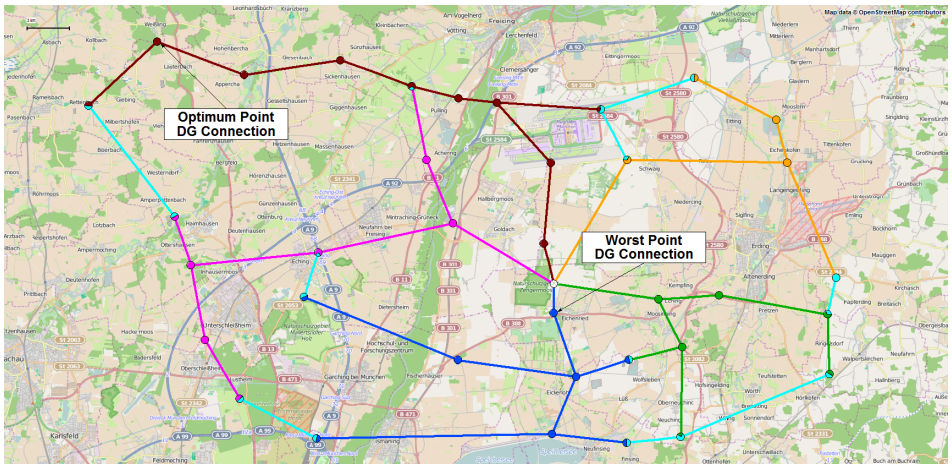


Figure 3.6 – Optimum and worst nodes to place distributed generator

3.2.2 Optimal Capacitor Placement

For the same distribution grid we determine the optimal location (bus) for a capacitor. Again, the optimization problem takes into account a loop over the week, the optimization is as following;

$$\text{Minimize} \qquad \sum_{d\in 7}\sum_{h\in 24}\left(\sum_{i=1}^{N_B} V_{dev}^{i}(S)\right) \tag{3.5}$$

$$\text{subject to} \qquad V^{min} \leq \left|V^{i}\right| \leq V^{max}, \qquad \forall \quad \text{i} \quad \in N_B \tag{3.6}$$

$$\tag{3.7}$$

Equation (3.5) minimizes the voltage deviation for the bus considered in this grid as a shunt capacitor with specific size and reactive power is being included. The results are shown in Fig. 3.7

The result in terms of power losses minimization and voltage deviation is reported in Fig. 3.7 and Fig.3.8.

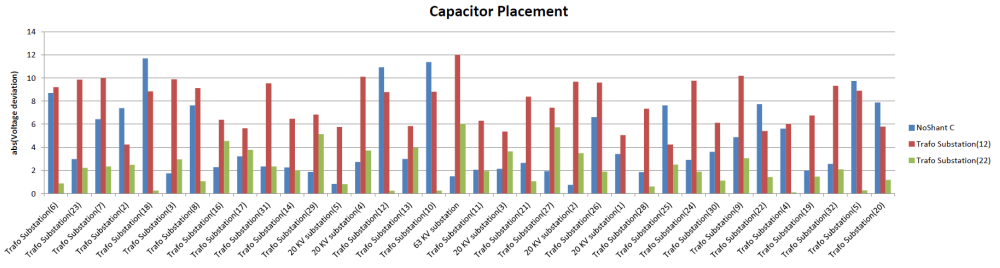


Figure 3.7 – Power factor correction shunt capacitor optimal placement

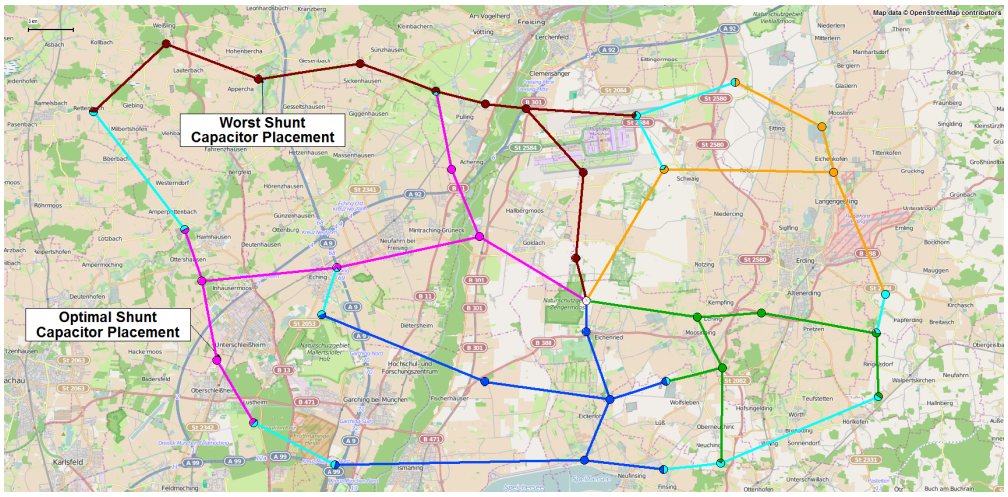


Figure 3.8 – Shunt capacitor placement in geographical view

4 Visualization Feature for Matlab-Based Tools

The most in use simulation tool for electrical engineers is Matlab. When it comes to simulate an electrical grid, always there is a lack of network visualization, and as mentioned earlier, smart grid simulation highlights the necessities of having an overall and quick view on the nodes and lines, are being studied.

This chapter presents a way to visualize electrical grid within Matlab-based analyses.

4.1 GraphPlot Objects

The most adapted way to depict electrical grid is to get use of GraphPlot object in Matlab. Graph plots are the primary way to visualize graphs and networks created using the graph and digraph (directed graph) functions, according to Matlab.

4.1.1 Graph Object in Matlab

Likewise other languages Matlab provide Graph objects, that basically are set and intended for the graph theory, which are mathematical structures used to model pairwise relations between objects. In mathematics a graph is made up of nodes, or points which are connected by edges, arcs, or lines. A graph may be undirected, meaning that there is no distinction between the two nodes associated with each edge, or its edges may be directed from one vertex to another. This could be validated for the electrical grid:

$$G = (N, E) \tag{4.1}$$

where Graph object G, is built by a set of Nodes N and related edges E, to form n Nodes and m edges (lines).

After you create a GraphPlot object, you can modify aspects of the plot by changing its property values. This is particularly useful for modifying the display of the graph nodes or edges.

4.1.1.1 Example

```
  %% Elementary instance of how to create a basical graph
>>S = {'a' 'a' 'b'};
>>T = {'b' 'c' 'c'};
>>g = graph(S,T)

g =
    graph with properties:

        Edges: [3x1 table]
        Nodes: [3x1 table]

>>e = g.Edges

        EndNodes
        ----------

        'a'    'b'
        'a'    'c'
        'b'    'c'

>>n = g.Nodes

n =
        Name
        ----

        'a'
        'b'
        'c'
```

We refer the Nodes of Graph object to the grid nodes, and the Edges to the power lines. A Graph object which have direction-less edges connecting the nodes, can exploit related object functions and methods to perform queries against the object, that in its turn are somehow the same we use for graph theory applied to an electrical grid.

Some Graph Objects **properties** and **functions** are fit and useful to be used for electrical grid theories. Some examples of those are shown and later are used for the proposed method.

4.1.2 GraphPlot Class Properties

To add or remove nodes from the graph, one can use the *addnode* or *rmnode* object functions, and *addedge* or *rmedge* do the same for the edges. Furthermore *Variable-*

Names, *VariableDescriptions* or *RowNames* properties of an object instance from *Graph class* are useful in order to create a general framework.

The edge *Weight* is another important properties, specially to be used in ***HeatMap*** analysis, where we may refer weight property to *loading*, *overloading* or even *losses*. Still aditional properties to the Graph class could be added by the user itself.

```
%% Example of Edge properties
>> G.Edges.Properties
    Description: ''
    VariableDescriptions: {}
    VariableUnits: {}
    DimensionNames: {'Row'  'Variable'}
    UserData: []
    RowNames: {}
    VariableNames: {'EndNodes'  'Weight'}
```

4.1.3 Graph Object Functions

An overview of the existing functions is the following:

Matlab method	utility
findedge	Locate edge in graph
rmedge	Locate node in graph
numedges	Number of nodes in graph
numnodes	Number of nodes in graph
subgraph	Extract subgraph
bfsearch	Breadth-first graph search
dfsearch	Depth-first graph search
conncomp	Connected graph components
maxflow	Maximum flow in graph
shortestpath	Shortest path between two single nodes
shortestpathtree	Shortest path tree from node
distances	Shortest path distances of all node pairs
adjacency	Graph adjacency matrix
incidence	Graph incidence matrix
laplacian	Graph Laplacian matrix
neighbors	Neighbors of graph node
nearest	Nearest neighbors within radius

4.1.4 Plot Function

Create a graph using a sparse adjacency matrix, and then plot the graph. This function also posses different properties and methods that are being used to picture a grid, and additionally with *Dynamic Plotting*, could be updated and create an animated *HeatMap* figure.

```
1 %% Example of plot a graph
2 >> figure
3 >> p = plot(G)
```

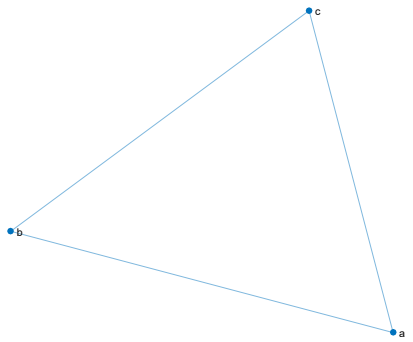


Figure 4.1 – Matlab Graph Class

```
%% All properties of a GraphPlot with the default values
p
    Annotation: [1x1 matlab.graphics.eventdata.Annotation]
    ArrowSize: 7
    BeingDeleted: 'off'
    BusyAction: 'queue'
    ButtonDownFcn: ''
    Children: [0x0 GraphicsPlaceholder]
    CreateFcn: ''
    DeleteFcn: ''
    DisplayName: ''
    EdgeAlpha: 0.5000
    EdgeCData: []
    EdgeColor: [0 0.4470 0.7410]
    EdgeLabel: {}
    EdgeLabelMode: 'manual'
    HandleVisibility: 'on'
    HitTest: 'on'
    Interruptible: 'on'
    LineStyle: '-'
    LineWidth: 0.5000
    Marker: 'o'
    MarkerSize: 4
    NodeCData: []
    NodeColor: [0 0.4470 0.7410]
    NodeLabel: {'a'  'b'  'c'}
    NodeLabelMode: 'auto'
    Parent: [1x1 Axes]
    PickableParts: 'visible'
    Selected: 'off'
    SelectionHighlight: 'on'
    ShowArrows: 'off'
    Tag: ''
    Type: 'graphplot'
    UIContextMenu: [0x0 GraphicsPlaceholder]
    UserData: []
    Visible: 'on'
    XData: [0.7362 -1.0743 0.3381]
    YData: [-0.8155 -0.2298 1.0453]
```

Now, we can modify the properties and accordingly the graph appears as needed.

4.1.4.1 Properties as Array and Matrices

It is possible to set properties for every element such as nodes and lines of the grid separately. In this case we should define arrays of *weights*, *sizes*, *styles* regarding the attribute is being assigned to a line or node. For the colors instead one should set a matrix of $n \times 3$ and $m \times 3$ for a grid containing n nodes and m edges

```
%% Modifying some properties
>> p.NodeColor = ones(g.numnodes,3);
>> p.NodeColor(:,1) = [0 1 0];
>> p.NodeColor(:,2) = [1 0 1];
>> p.NodeColor(:,3) = [0 0 1];
>> p.EdgeColor = zeros(g.numedges,3);
>> p.EdgeColor(:,1) = [0 1 0];
>> p.EdgeColor(:,2) = [1 0 1];
>> p.EdgeColor(:,3) = [0 0 1];

>> p.Marker = {'s' 'o' '^'};
>> p.MarkerSize = [8 25 16];
>> p.LineWidth = [4 1 2];
```

layout(H) changes the layout of graph plot H by using an automatic choice of layout method based on the structure of the graph. The layout function modifies the XData and YData properties of H.

4.2 HeatMap Analysis

In order to develop a high efficient visualization characteristic, we aim to create *HeatMap* feature analysis class, using properties and methods of *Graph*, *Plot* and *GraphPlot* objects. To do so we start with an example and work out step by step. This example which will be a simple script in Matlab, includes an instance distribution network and to assess that, *Backward-forward-sweep load flow (BFS)* equations.

4.2.1 Distribution Network Structure

Since the case of study and example is in distribution network, the grid is within *radial* category and its representation will be done through *incidence matrix* and its *inverse*.

Regarding the conventions of naming the lines and nodes, the grid will be formed as:

Line	From Bus	To Bus
1	*Slack*	*1*
2	*1*	*2*
3	*2*	*3*
4	*2*	*4*
5	*3*	*5*
6	*3*	*6*
7	*4*	*7*
8	*4*	*8*
9	*8*	*9*
10	*8*	*10*

Table 4.1 – A simple example grid visualization

- the incidence matrix will be built by visual inspection, using the conventional values 1 for the sending node and -1 for the ending node.
- the *gamma* matrix (the inverse of the incidence matrix) will be the one in use, in power flow analysis.

```
%% Incidence matrix forming

NumNodi = 10;
L=-eye(NumNodi);
L(2,1)=1;
L(3,2)=1;
L(4,2)=1;
L(5,3)=1;
L(6,3)=1;
L(7,4)=1;
L(8,4)=1;
L(9,8)=1;
L(10,8)=1;
```

```
L =

    -1     0     0     0     0     0     0     0     0     0
     1    -1     0     0     0     0     0     0     0     0
     0     1    -1     0     0     0     0     0     0     0
     0     1     0    -1     0     0     0     0     0     0
     0     0     1     0    -1     0     0     0     0     0
     0     0     1     0     0    -1     0     0     0     0
     0     0     0     1     0     0    -1     0     0     0
     0     0     0     1     0     0     0    -1     0     0
     0     0     0     0     0     0     0     1    -1     0
     0     0     0     0     0     0     0     1     0    -1
```

The further script visualizes the grid structure:

- first, the *sources* and *targets* are extracted from *gamma* matrix
- then a Graph object and GraphPlot will be created

```
%% extracting sources and target from L matrix

new_col = zeros(1,size(L,2));
new_row = zeros(1,size(L,1)+1)';
L = [new_col;L];
L = [new_row,L];
L(1,1) = -1;
L(2,1) = 1;

S =[];
T =[];

for u=1:size(L,1)
    for v =1:size(L,2)
        if L(v,u) == 1
            S = [S,u];
            T=[T,v];
        end
    end
end

%% PlotGraph depicts the grid
figure('name', 'Physical Network Structure','NumberTitle','off');
set(gcf,'units','normalized','outerposition',[0 0 1 1])
G = graph(S,T);
p = plot(G);
```

4.2.2 Properties Setting

Here we set names of the nodes and lines according to the incidence matrix and regarding how the Graph object puts them in order.

```
%% name setting
EdgesNames = {};
for j=1:size(T,2)
    name = strcat({'('},num2str(T(j)-1), {')'});
    EdgesNames=[EdgesNames,name];
end

NodeNames = {};
for j=1:size(L,1)
    name = num2str(j-1);
    NodeNames=[NodeNames,name];
end
NodeNames(1) = {'La rete esterna'};

%% line color setting
edge_colors = zeros(size(T,2),3);
```

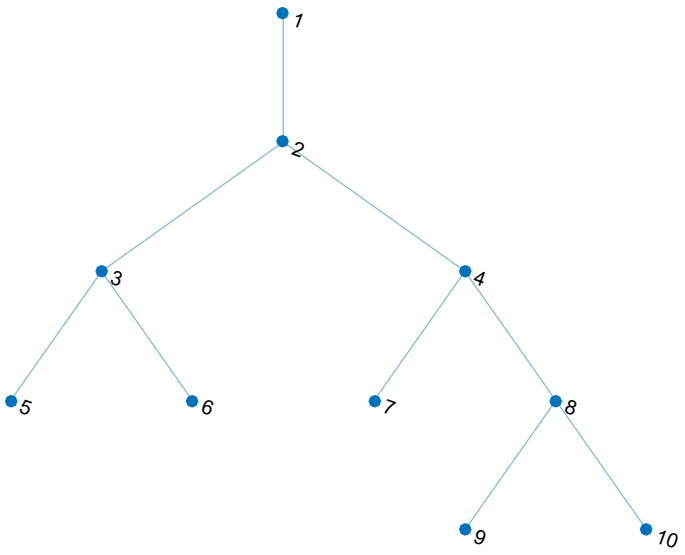


Figure 4.2 – Physical network by graph

```
edge_colors(:,3) = 1 ;

%% node color setting
node_colors = zeros(size(L,2),3);
node_colors(:,3) = 1 ;
node_colors(1,2) = 1 ;

%% graph
figure('name', 'Physical Network Structure','NumberTitle','off');
set(gcf,'units','normalized','outerposition',[0 0 1 1])
G = graph(S,T);
p = plot(G,'Layout','layered','EdgeLabel', EdgesNames,'NodeLabel',NodeNames);
axis off

%% attribute setting
p.MarkerSize = ones(1,size(L,1))*5;
p.MarkerSize(1) = p.MarkerSize(1,1)*2;
p.Marker = ['s', repmat({'o'}, 1, size(L,1)-1)];

p.NodeColor = node_colors;
p.EdgeColor = edge_colors;

p.NodeLabel = NodeNames;
p.NodeLabel(1) = {'La rete esterna'};
```

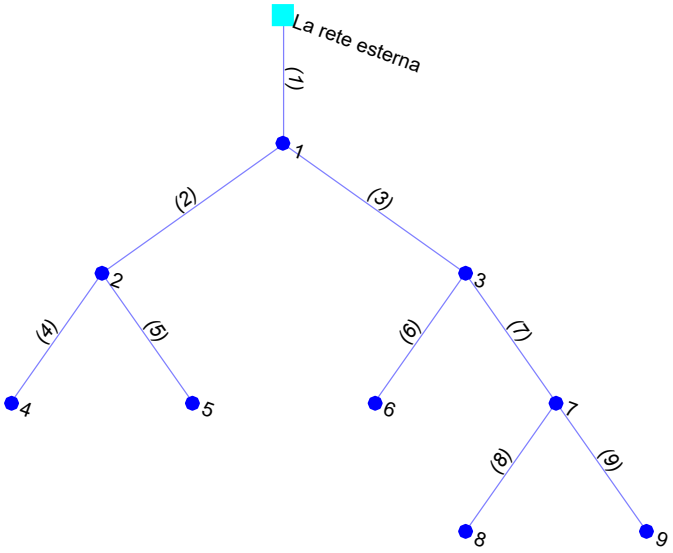


Figure 4.3 – Adapting GraphPlot properties

4.2.3 BFS Power Flow

We set the network parameters and launch the BFS Power flow.

$$V^{(k)} = V_0 1 - \Gamma Z_B \Gamma^T i_s^{(k)} \tag{4.2}$$

where V is vector of nodal voltages, Z_B is the impedance matrix, and Γ is the "filter" applied to the matrix Z_B, and i_B is the branch current vector.

The output result of such calculation will be a $M \times N$ vector (matrix) for voltage, active and reactive power, loading, losses and other variable of interest. M stands for number of buses and N represents the step number of simulation.

The function `visualize.m` converts the vector (matrix) results into visual aspects;

- Active power generation in nodes are being represented as a green circle.
- Loads in nodes are being represented as a red circle.
- Circles' diameter is proportional to amplitude of the active power in those nodes.

- Edge's thickness represents its loading level.

- The color of the edge is representing loading level of line scaled to its maximum thermal limit.

Running this function, the results appears as shown in Fig. 4.4;

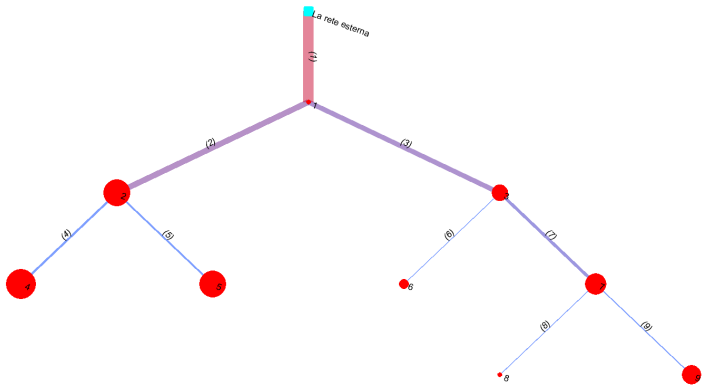


Figure 4.4 – Heatmap

Finally, it is possible to combine visual and numerical result to have both intuitive colored figure and also the exact numeric value of the variable. Figure 4.5 is an example of a 36 bus system which has been calculated and shows the results such as numerical value of voltage in p.u. and line loading in percent in addition to the already defined options.

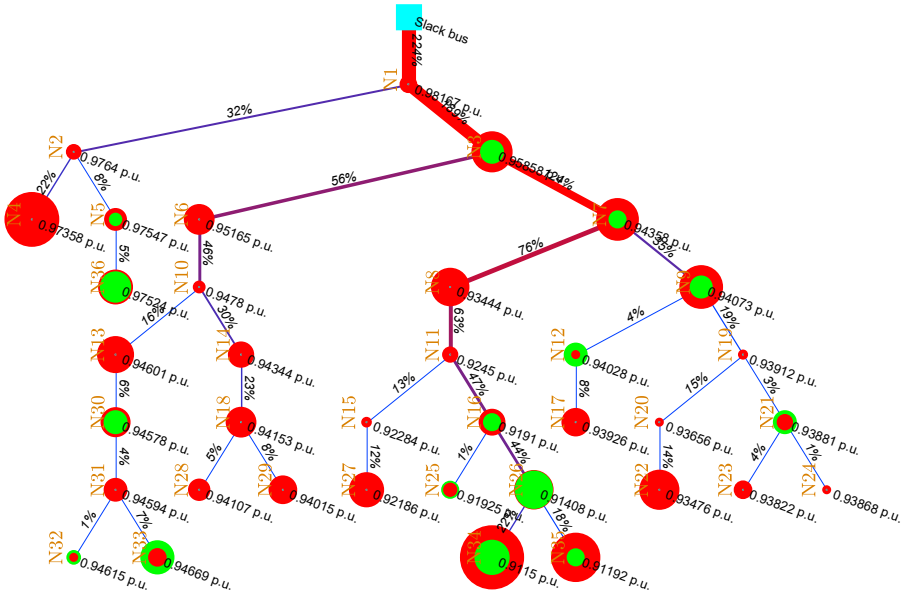


Figure 4.5 – Visual and numerical result

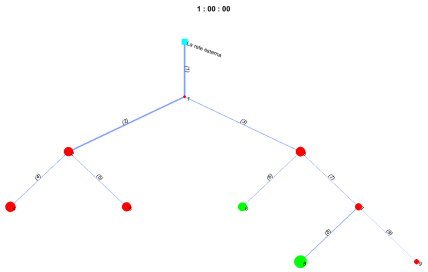
4.2.4 Dynamic Graph Visualization

Large dynamic networks are targets of analysis in many fields. Tracking temporal changes at scale in these networks is challenging due in part to the fact that small changes can be missed or drowned-out by the rest of the network. Controlling and modifying numerical attributes of nodes/edges, and their temporal evolution, we gain a dynamic electrical grid view in which the attributes such as *color*, *width*, *size* and *shape* are subject to the user-defined evolution, based on the specific element, method or analysis like *loading*, *voltage deviation*, *power losses*, *reactive power*, *active power consumption/generation* and so on.

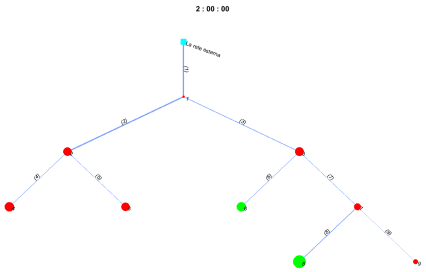
In every step of the simulation we adapt the normalized value of the results (e.g. loading for the lines) to a specific color and size, and then update the attirbutes of the graph object.

This is applied to the nodes also, where for example the loads are assigned with ***red*** color, and generation is specified with ***green*** color.

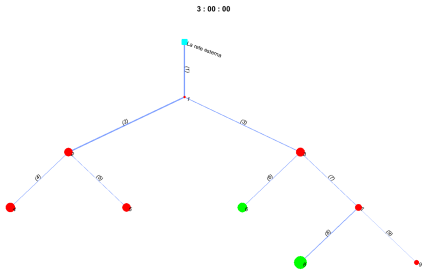
We apply BFS to the presented instance grid for a 24 hours of simulation, so M becomes 24 and output matrix will have dimension 10×24.



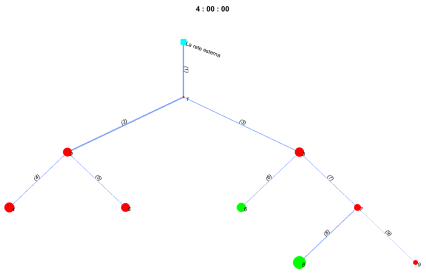
(a) 00 : 00



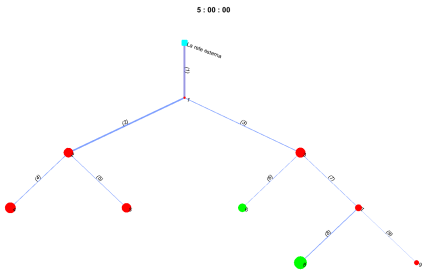
(b) 01 : 00



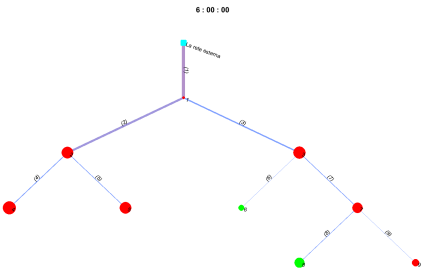
(c) 02 : 00



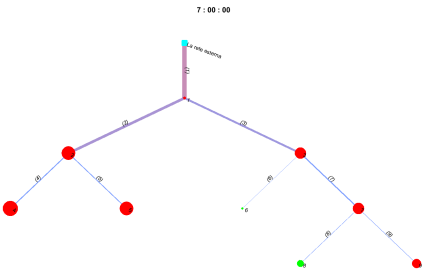
(d) 03 : 00



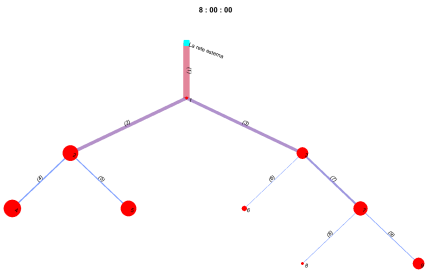
(e) 04 : 00



(f) 05 : 00



(g) 06 : 00



(h) 07 : 00

4.3 Power System Component Icon

To get more visualization quality we go further and try to create power system component conventional icons, as in some powerful softwares. To do so we use **patch** class.

4.3.1 Matlab Patch Class

patch(X,Y,C) creates one or several filled polygons through using the elements of X and Y as the coordinates for each vertex. Patch is used to connect vertices in the determined order. To create a single polygon, specify X and Y as the vectors. To create several polygons, specify X and Y as matrices where every column corresponds to a polygon. C is intended to determine the polygon colors.

The function `PlotCustMark.m` converts input coordinations to an icon using patch object.

```
function patchHndl = plotCustMark(xData,yData,markerDataX,markerDataY,markerSize,
   lineThick, face_color)


xData = reshape(xData,length(xData),1) ;
yData = reshape(yData,length(yData),1) ;
markerDataX = markerSize * reshape(markerDataX,1,length(markerDataX)) ;
markerDataY = markerSize * reshape(markerDataY,1,length(markerDataY)) ;

%% prepare and plot the patches
markerEdgeColor = [0 0 0] ;
markerFaceColor = face_color ;

vertX = repmat(markerDataX,length(xData),1) ; vertX = vertX(:) ;
vertY = repmat(markerDataY,length(yData),1) ; vertY = vertY(:) ;

vertX = repmat(xData,length(markerDataX),1) + vertX ;
vertY = repmat(yData,length(markerDataY),1) + vertY ;

faces = 0:length(xData):length(xData)*(length(markerDataY)-1) ;
faces = repmat(faces,length(xData),1) ;
faces = repmat((1:length(xData))',1,length(markerDataY)) + faces ;

patchHndl = patch('Faces',faces,'Vertices',[vertX vertY]);
set(patchHndl,'FaceColor',markerFaceColor,'LineWidth', lineThick, 'EdgeColor',
  markerEdgeColor) ;
hold on
```

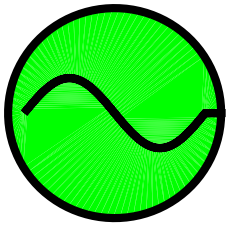
and once pass the following codes to it:

```
slot = 100;
theta = linspace(0,2*pi,slot);
ro = ones(1,slot);
[x_array, y_array] = pol2cart(theta,ro);
acSignX = linspace(2*pi, 0,slot);
acSignY = sin(acSignX);
```

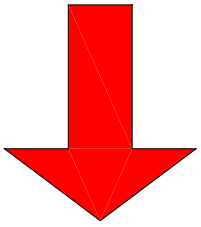
```
x_array = [x_array, linspace(max(x_array)*0.85, min(x_array)*0.85,slot),...
flip(linspace(max(x_array)*0.85, min(x_array)*0.85,slot))];
y_array = [y_array, acSignY/3, flip(acSignY/3)];
plotCustMark(x, y, x_array, y_array, size, 1.5, [0 1 0]);
```

image 4.6i will appear. In fact `x_array` and `y_array` indicate a path of painting in patch.

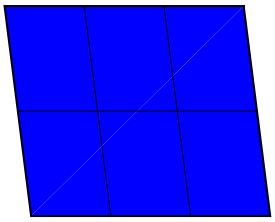
In this way we create some other essential icons as are reported in following figure:



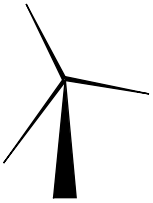
(i) Generic AC alternator icon



(j) Active load



(k) photovoltaics



(l) Wind tower

Figure 4.6 – Power system icon created in Matlab

4.3.2 WinArt Matlab

Applying this method we can imitate features of other tools. Here we present what WinArt does by using Matlab. Launching proposed cosed the defined grid in excel files will be seen visually as in Figure 4.7.

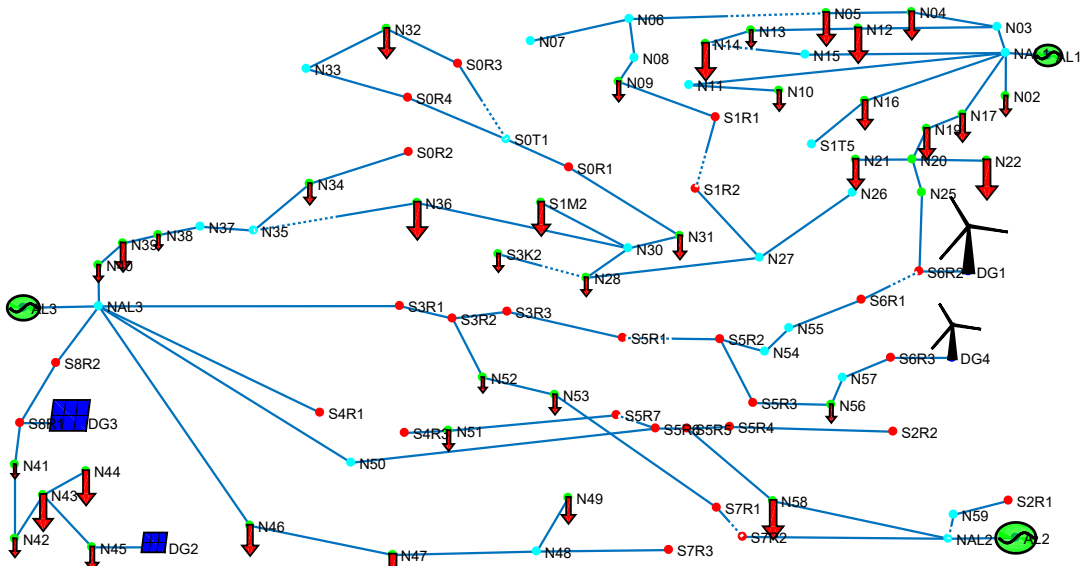


Figure 4.7 – Grid demonstration with electrical icons

A appendix

The material reported in this section is taken from the DigSilent PowerFactory User's Manual, and it included in this Appendix to recall the contents that have been specifically used for developing the approach presented in the thesis.

A.0.1 PowerFactory Software Concept

A.0.1.1 Single Database Concept

This concept has the following characteristics:

- Optimal data organization and project definitions for performing any type of calculation, storage of settings, diagrams and visualization options or software operation sequences,
- no need for tedious organization of several files for defining the various analysis aspects and project execution work flows,
- Database environment fully integrates all necessary data, such as that required for defining cases, scenarios, variants, single-line graphics, outputs, run conditions, calculation options, graphic or user defined models; saving a project includes everything required to rerun all defined cases at a later stage,
- Access to all data via a comfortable and powerful data manager,with object browser, plus various type of diagrams is possible also,
- Comprehensive data model supporting of all analyses functions.

A.0.1.2 User Roles

Access to user information through a user accounting system Protection of data through different types of access rights Folder sharing between users with “read-only” access; this is especially useful for libraries and network base cases which should be administrated only by authorized personnel.

A.0.1.3 Multi-User Operation and Team working

The main features are:

- Multi-user data administration supporting MS-SQL or ORACLE databases support of user accounting, access rights and data sharing, featuring the powerful option of allowing several users to work on the same project in a coordinated way; this demonstrates the concept of non-redundant data management in PowerFactory.
- Management of multi-user data editing via the definition of a base project, project versions and derived projects (virtual projects).
- Support of version control including rollback functions and merge/compare tools Network Variations, Expansion Stages Management and Operational Scenarios.
- Support of time-stamped network variations.
- Variation scheduler for easy handling of sub-projects.
- Definition of study cases and operational scenarios.
- Activation of network stages according to study time; this automatically addresses the handling of power system components according to their commissioning and de-commissioning dates.

A.0.1.4 Multi-Level Models

Data describing network models such as cables, machines, loads, transformers, etc., are subdivided into element data and type data which point to libraries. All data to be entered are grouped into basic data (data required for all calculations) and function level data (data required only for executing specific calculations).
Data are simply entered in physical quantities rather than in per unit values, minimizing the need for manual recalculation and conversion of data.

Verification of input data occurs with detailed warning and error messages.
Integrated calculators are available for asynchronous machines, cable data and tower configurations.

A.0.1.5 Batch Mode, Engine Mode and Interfaces

Theses model enable:

- Fully interactive windowing mode according to the latest, proven standards Engine mode for background operation.
- Various communication features to exchange data with other applications such as GIS, SCADA and real-time control systems via OPC, shared memory, DGS (CSV, ODBC), etc.
- Hybrid operation switching between background and windowing mode according to users' needs.
- Data exchange via CIM, PSS/E, UCTE and many other file formats.
- The Engine Manager component provides access to multiple PowerFactory Engines via web services.
- The built-in queuing and scheduling simplifies the Engine integration into other applications.

A.0.2 Power Equipment Models, Grid Representations and Power Equipment

A.0.2.1 Grid Models

Meshed and radial AC systems with 1-, 2-, 3-, and 4-phases
Meshed and radial DC systems
Combined AC and DC system modelling
Model validity from LV up to ultra-high voltage

A.0.2.2 Phase Technologies

Single phase with/without neutral
Two-phase with/without neutral

Bi-phase with/without neutral
Three-phase with/without neutral

A.0.2.3 Substations

Simple terminal models to be used for “node and branch” representation, marshalling panels, terminal blocks, terminal strips, clamping bars, joints and junctions
Complex substation models with the provision of various standard busbar configurations such as single- and double busbars with/without tie-breakers, bypass busbars, 1½ busbar systems and flexible busbar configurations according to user-specific needs
Secondary Substation object which provides templates with a broad variety of predefined secondary substation configurations
Templates for holding any type of user-specific busbar configuration, including preconfigured protection schemes

A.0.2.4 Generators and Sources

Synchronous and asynchronous generator
Doubly-fed induction generator
Static generator (for PV, fuel cell, wind generator, battery storage, etc.)
External grid
AC voltage source
AC current source
2-terminal AC voltage source
PV system
Impulse source

A.0.2.5 Loads

General load model (for HV and MV-feeders)
Complex load model (for feeders with a large number of induction motors)
Low voltage load (can be assigned across line and cable sections)
Medium voltage load, representing a distribution transformer together with a reduced load/generation model
Synchronous and asynchronous motor load

A.0.2.6 Reactive Power Compensation

Static Var Compensator (SVC)
Shunt/Filter (RLC, RL, C, RLCRp, RLCCRp)
Series RLC filter

A.0.2.7 Branch models

Overhead line and cable models (p-models and distributed parameter models)
Circuits and line sub-sections
Mutual data, line couplings, tower geometries
2-, 2-N-winding transformer and auto transformer 3-winding transformer, booster transformer
Series reactor, series capacitor and common impedance

A.0.2.8 DC Models

1-terminal and 2-terminal DC voltage source and DC current source
DC/DC converter
Inductive DC-coupling
DC machine
DC battery

A.0.2.9 Power Electronics Devices

Thyristor/Diode converter models Self-commutated converter models (VSC-converter)
DC valve (for building individual converter topologies) Softstarter

A.0.2.10 Switches and Substation Equipment

Circuit Breaker and Disconnector
Load-Break-Disconnector
Load-Switch
Grounding Switch
Fuse
NEC/NER, grounding devices
Surge arrester

A.0.2.11 Composite Models

Composite node models, e.g. representing complex substations
Composite branch models
Template library for handling composite models

A.0.2.12 Parameter characteristics

Time characteristics and discrete characteristics
Scalar, vector and matrix characteristics
File references and polygons
Continuous and discrete triggers
Frequency and time scales

A.0.2.13 Controllers

Station controller, secondary controller (SCO), virtual power plant
Tap controller, shunt controller
User-definable capability diagrams and controllers

A.0.2.14 Organisation and Grouping

Site, station, substation, area, zone
Feeder, branch, bay
Operator, owner
Boundaries

A.0.2.15 Operational Library

Substation running arrangements
CB ratings
Thermal ratings
Library of faults/contingencies
Library of (planned) outages

A.0.2.16 Others

Protection relays with over 30 basic protection function blocks Manufacturer-specific relay library with relay models from all major manufacturers

CT, VT and various measurement transducers (P, Q, f, etc.)
Fourier source, harmonic source, FFT
Clock, sample and hold, sample and hold noise generator

A.0.3 Network Diagrams and Graphic Features

Data visualization is becoming an increasingly important component of analytics in the age of big data.

Electrical network is not excluded from this essential trend, but traditional architectures and infrastructures are not up to this challenge. Big data has a lot of potential for power electric systems – but it can also be a burden. IT teams are faced with ever-growing requests for data, and decision makers are frustrated because it can take hours to get answers to questions.

Visual analytics – when paired with big data – can have a significant impact on the analyses gain insight from its data. But in order to take advantage of visual analytics, we first need to address several challenges related to visualization;

1. Meeting the need for speed
2. Understanding the data
3. Addressing data quality
4. Displaying meaningful results
5. Dealing with outliers

One of the most focused issues in this work addresses a real and important challenge in the power system analysis; grid visualization. PowerFactory empowers the user to have desired diagram and network graphic representation.

A.0.3.1 Categories of Network Diagrams

Simplified Single Line Diagrams with various options for a schematic view of substation topology and switching status
Detailed Single Line Diagrams showing all switches (circuit breakers and dis-connectors)
Intelligent Overview Diagrams providing a node and branch representation of the network; can be schematically, geographically or semi-geographically arranged
Visual representation of the network by using GPS coordinates to generate geographic diagrams

A.0.3.2 General Features

Handle mixed representations of Detailed Single Line Diagrams, Simplified Single Line Diagrams and Overview Diagrams
Access equipment editing menus in the single line diagram via cursor selection of the appropriate element, region or composite model Zoom-in or zoom-out of area networks or composite model graphics
Initiate calculation events directly within the graphical environment, including circuit breaker switching, fault implementation and other data changes
Option to immediately reflect any editing activity on the graphical level
Display any calculation results immediately in result boxes in single line diagrams.
All program variables and signals can be displayed according to a highly flexible user definition for various object categories and analysis functions
Display any calculation result to be defined on various functional levels and categories for any object
Insert freely-configured result displays
Provision of auxiliary graphics editing for enhanced documentation
Perform copy/paste operation on single objects and groups
View and operate several graphic windows with different layers and grid
sections simultaneously; utilize several graphical representations of the same system simultaneously
Spread large diagrams over several pages
Support of pre-defined and user-defined graphical layers
Support the use of multiple layers, which can be reorder, hide and move offering a high degree of flexibility; import/export annotation layers is possible
Graphical representation of protection devices and neutral connections
Placement of user-definable icons as buttons for executing DPL scripts; this way users can create custom panels of frequently-executed DPL-initiated commands

A.0.3.3 Coloring of Network Diagrams

Provision of various coloring modes according to topology criteria such as areas, zones, owners, operators, routes, station connectivity, energizing status, boundaries/interior regions, isolated grids, etc.
Coloring options to display voltage levels, equipment loading and operation ranges
Define coloring based on AC/DC equipment category and phase technology
Display of grid modifications and variants, recording of expansion stage modifications, missing grid connections
Provision of feeder coloring and path definitions
User-defined filters based on complex equations or DPL (DIgSilent custom program-

ming languages) scripts

A.0.3.4 User-definable Symbols

Support of user-definable symbols based on standard graphical formats (.wmf,.bmp). E.g. use your own symbols for wind turbines, PV panels, hydro units, etc.
Define specific graphical representation for transformer, shunts, circuit breakers, isolators to fit individual needs

A.0.3.5 Composite Graphics

Elements can be grouped together and stored as Composite Graphics. Typical applications are standard bus-bar arrangements, switchboard configurations, HVDC structures, PV panels, typical wind turbine configurations or complete wind parks.

Composite Graphics can be easily handled via the Template Manager. Templates can be populated with type and element data. For drawing Composite Graphics, the Template Manager is operated as Drawing Tool Box.

A.0.3.6 Virtual Instruments

DIgSILENT PowerFactory applies the concept of Virtual Instruments (VI) as a tool for displaying any calculated result or variable. Results may be displayed in the form of bar graphs, plotted curves, or even tables of values, with all of these representations being completely user-definable.

VIs are used to display protection curves, harmonics analysis results or to view electrical variables from any location in the network single line diagram, and any model variable during RMS and EMT simulations. Many VIs provide additional built-in functionality such as curve labelling and measuring, scaling, curve fitting, filtering and digitiser functions. Typical Virtual Instruments available are; x-t, x-y and 2y axis plots
Bar diagrams, harmonic distortion diagram
Overcurrent-time-diagrams, distance-time diagrams
Vector and path diagram
Relay plots
Voltage sag diagram, waveform diagram
Eigenvalue and phasor diagram and FFT plot
Scales and measurement diagram
Bitmaps, buttons, DPL-command buttons, digital display

Curve-digitising diagram
Text label

Bibliography

[1] https://www.mathworks.com/help/map/examples/creating-map-displays-with-latitude-and-longitude-data.html

[2] J.E.S. de Haan, P. H. Nguyen, W. Kling, and P. F. Ribeiro, “Social Interaction Interface for Performance Analysis of Smart Grids" , *IEEE International Workshop on Smart Grid Modeling and Simulation (SGMS)*, 2011.

[3] M.A. Al Faruque and F. Ahourai, “GridMat", *Innovative Smart Grid Technologies Conference (ISGT)*, 2014.

[4] W. Guo, J. Lin, Y. Song, and X. Chen, “Network Based on Co-simulation of GridLAB-D and Matlab", *International Conference on Power System Technology (POWERCON)*, 2014.

[5] A. Makhmalbaf, J. Fuller, V. Srivastava, S. Ciraci, and J. Daily, “Using Open Source Co-Simulation of Detailed Whole Building with the Power System to Study Smart Grid Applications", *IEEE Technologies for Sustainability (SusTech)*, 2014, pp. 192-198.

[6] EnergyPlus simulation program, https://energyplus.net/

[7] J.C. Fuller, S.E. McHann, and W. Sunderman, “Using Open Source Modeling Tools To Enhance Engineering Analysis", *Rural Electric Power Conference (REPC)*, 2014.

[8] R. Roche, S. Natarajan, A. Bhattacharyya and S. Suryanarayanan, “A Framework for Co-simulation of AI Tools with Power Systems Analysis Software", *23rd International Workshop on Database and Expert Sytems Applications*, 2012.

[9] S. Lehnhoff, O. Nannen, S. Rohjans, and F. Schlögl “Exchangeability of Power Flow Simulators in Smart Grid Co-Simulations with mosaik", *Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 2015.

[10] DIgSILENT PowerFactory simulation program, http://www.digsilent.de/index.php/products-powerfactory.html

[11] Mosaik simulation program, https://mosaik.offis.de/

[12] X. Kong and X. Yu, “Co-Simulation of a Marine Electrical Power System using PowerFactory and MATLAB/Simulink", *Electric Ship Technologies Symposium (ESTS)*, 2013.

[13] M. Stifter, R. Schwalbe, F. Andren, and T. Strasser, “Steady-State Co-Simulation with PowerFactory", *Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 2013.

[14] X. Sun, Y. Chen, J. Liu, and S. Huang, “A co-simulation platform for smart grid considering interaction between information and power systems", *Innovative Smart Grid Technologies Conference (ISGT)*, 2014.

[15] H. Lin, S. Veda, and S. Shukla, “Global Event-Driven Co-Simulation Framework for Interconnected Power System and Communication Network", *IEEE Trans. on Smart Grid*, vol. 3, no. 3, pp. 1444-1456, 2012.

[16] D. Bhor, K. Angappan and K.M. Sivalingam, “A Co-Simulation Framework for Smart Grid Wide-Area Monitoring Networks", *Conference on Communication Systems and Networks (COMSNETS)*, 2014.

[17] D. Bian, M. Kulzu, M. Pipattanasomporn, S, Rahman, and Y. Wu, “Real-time Co-simulation Platform using OPAL-RT and OPNET for Analyzing Smart Grid Performance", *IEEE PES General Meeting*, 2015.

[18] M. Stevic, Li. W, M. Ferdowsi, A. Benigni, F. Ponci, and A. Monti, “A two-step simulation approach for joint analysis of power systems and communication infrastructures", *Innovative Smart Grid Technologies Europe*, 2013.

[19] J.A. Martinez, F. de León, A. Mehrizi-Sani, M.H. Nehrir, and C. Wang, “Tools for Analysis and Design of Distributed Resources-Part II", *IEEE Trans. on Power Delivery*, vol. 26, no. 2, pp. 1653-1662, 2011.

[20] T.J. Overbye and J.D. Weber, “Visualizing the electric grid",*IEEE Spectrum*, vol. 38, no. 2, pp. 52-58, 2001.

[21] A. Khanna and P. Cuffe, “Visualization of Power System Data and Structure", *Frontiers in Engineering*, 2015.

[22] J. Overbye, A.D. Wiegmann, and M. Davis, “Visualization of Power Systems and Components", Ph.D. dissertation, PSERC, 2005.

[23] PYPOWER, a port of MATPOWER to the Python programming language, https://pypi.python.org/pypi/PYPOWER

[24] MATLAB Engine for Python, http://de.mathworks.com/help/matlab/matlab-engine-for-python.html

[25] A. Bosovic, M. Music and S. Sadovic, “Analysis of the impacts of plug-in electric vehicle charging on the part of a real medium voltage distribution network", *Innovative Smart Grid Technol. Europe*, 2014.

[26] C.H. Tie and C.K. Gan, “The impact of electric vehicle charging on a residential low voltage distribution network in Malaysia", *Innovative Smart Grid Technologies*, 2014.

[27] P. Fernandez and L. Roman, “Assessment of the Impact of Plug-in Electric Vehicles on Distribution Networks", *IEEE Trans. on Power Systems*, vol. 26, no. 1, pp. 206-213, 2011.

[28] V. del Razo, C. Goebel, and H.A. Jacobsen, “Electric Vehicle Real-Time Charging Control with Vehicle-Originating-Signals", *IEEE Trans. on Transportation Electrification*, vol. 1, no. 2, pp. 150-167, 2015.

[29] J. Rivera, P. Wolfrum, S. Hirche, C. Goebel, and H.-A. Jacobsen, “Alternating direction method of multipliers for decentralized electric vehicle charging control", *IEEE 52nd Annual Conference on Decision and Control (CDC)*, 2013.

[30] V. del Razo, C. Goebel, and H.A. Jacobsen, “Reducing communication requirements for electric vehicle charging using vehicle-originating-signals", *International Conference on Smart Grid Communications (SmartGridComm)*, 2014.