Web App For Day Ahead Market Clearing Documentation

Release 0.1

Martyn van Dijke
Yuanlong Li

Jun 29, 2017

CONTENTS

Welcome to the documentation of the Wep application for clearing the day ahead market. Below you will vind various in depth explinations of how the web application works and is put togheter. The web application is largley implemented in `PyPSA` and `Django` to get a good understanding of both technologies we recommend to read them The web application is live at Documentation may also be found at..

CHAPTER

ONE

INSTALLATION

Getting Python

This web application uses python3 To install python on a debian based linux machine simply run:

```
sudo apt-get install python3
```

it's always helpful to use a virtual environment for your python installation.

Getting a solver for linear optimisation

The web app is known to work with the free software GLPK. The web application should also work using the Gurobi software (and whatever else Pyomo works with).

For Debian-based systems you can get GLPK with:

```
sudo apt-get install glpk-utils
```

There are similar packages for other GNU/Linux distributions. For Windows there is WinGLPK. For Mac OS X brew is your friend.

Installing the backend of the web app

If you have the Python package installed (`pip3`) just run:

```
sudo pip3 install -r requirements.txt
```

Please make sure to install the dependencies for Python 3, since the web application is programmed in Python 3

Backend dependencies

The web application relies on the following packages which are not contained in a standard Python installation:

- pypsa
- numpy
- scipy

- pandas
- networkx
- pyomo
- moreitertools
- Django

Using above installation technique these packages are installed.

Getting Nodejs

NodeJs can be installed by going to **'NodeJs<https://nodejs.org/en/download/ >'_** To install NodeJs run

```
sudo apt-get install node
```

Also webpack is needed wich may be installed using npm:

```
sudo npm install webpack -g
```

After installing Node.Js and webpack run

```
npm install
```

In the asset folder located in mysite/assets

Also make sure to run

```
webpack --config webpack.config.js --watch
```

In side the asset folder

CHAPTER

TWO

TUTORIAL

To use the web application visit .. Here you will be greeted with an welcome screen that should be similar with

Getting Started

To get started choose the model of your liking. 1. Simple 2. Network 3. Stochastic

Choose the model you want to start with. Data between models may not be shared, and needs to be manually transferred. Please be carefull when choosing your model !

What it can do

This web application can do :

- Market clearing
- Optimal network dispatching

The difference between the two models is explained here :

Simple

The elemnetaqry model can calculated :

- Nominal power
- Minimum power
- Naximum power

Network

The network constrained model can calculate :

- Nominal power
- Minimum power
- Maximum power
- Ramp up/ ramp down limits
- Min/max up / down time

- Initial capital cost optimization
- Start up / Start down cost optimization
- Load flow calculations (reactive power)
- Capital cost optimization
- Maximum voltage angle
- Minimum voltage angle

Using the web application

After you have chosen the model you want to use, you are greeted by a screen similar to this

INSERT IMAGE

You can save data to the server by clicking the option and the save button To retrieve the stored data click the option and the get button

Once you have drawn your desired layout, and made sure that all desired inputs are given. Click on the run button to send the frontend data to the backend for the optimization. Depending on the amount of generators, loads and busses it may take some time 0- 10/20 s to calculate the optimum. If the desired network is not possible you will be presented by an error screen, if this is not the case you will recieve the output.

CHAPTER

THREE

EXPLANATION OF FRONT AND BACKEND

This section will dive into the intergration of the frontend and backend of the overall system.

Overview

The figure below will provide an overview of the system hierachy, this figure also indicates how the diffrent systems (Python & Node.Js) are interconnected and operate. Using this figure the overall structure of the web application should become clear.

Backend Pyton

The backend is writen in Python 3.5 and may be digisted further. It consist as seen in the overview of two main part's the Django web server and the simple & network case. The simple and network case are programmed in one 'EconomicDispatcher'.

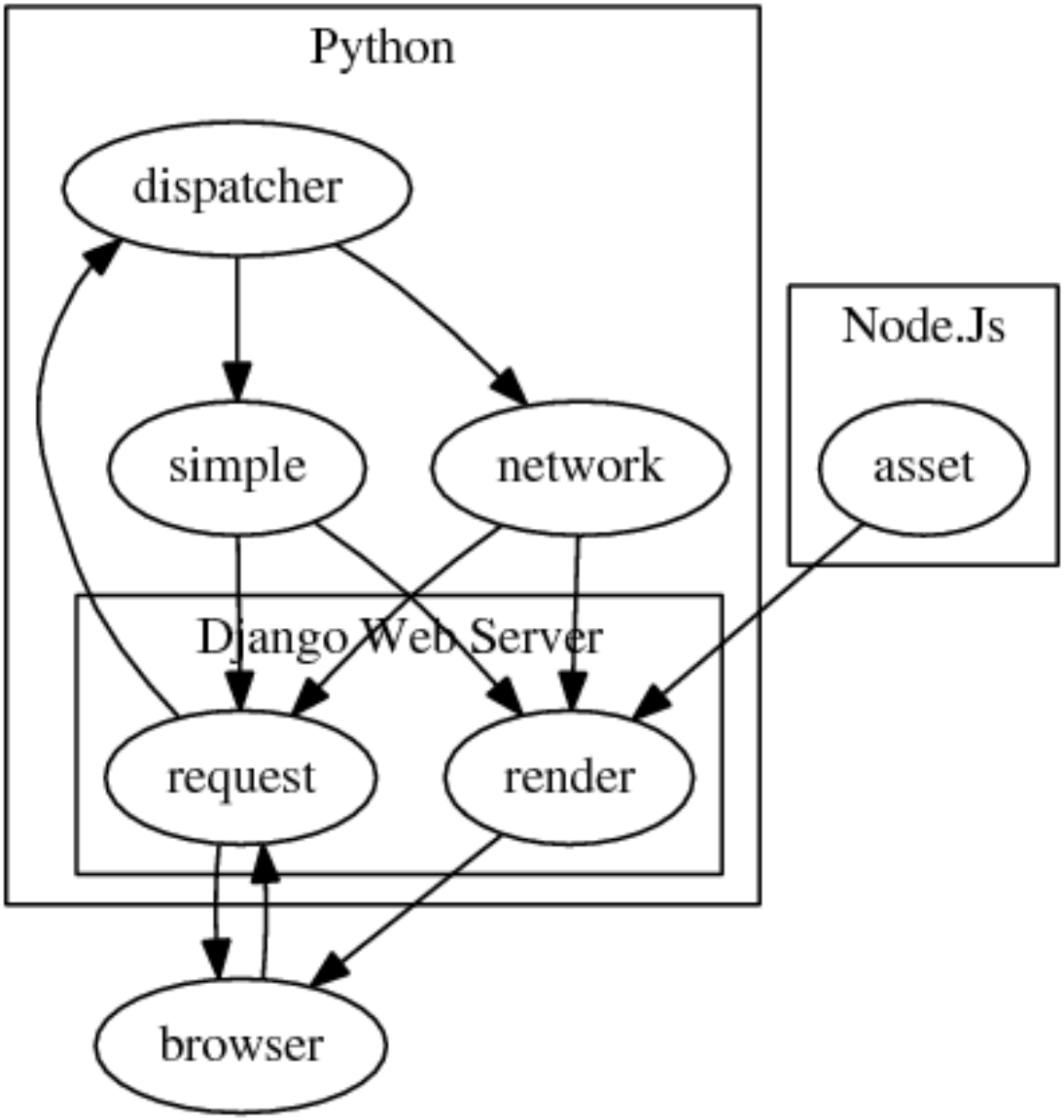
Django

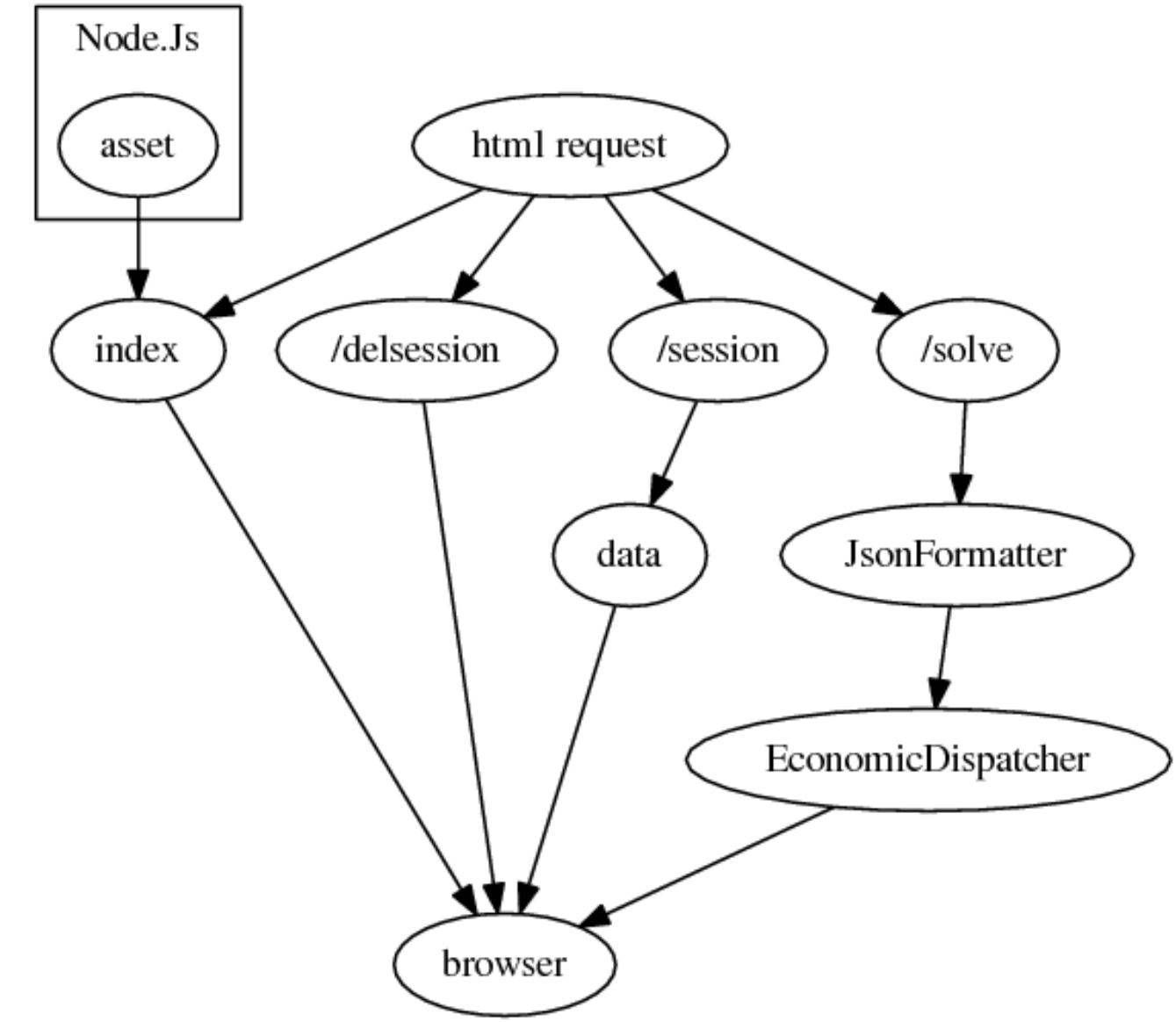
Django ensure that all html request are handled well, and is a web server framework. Django process the html requested and handles the data transfer from the front end to the backend and vice versa. Django also provides the simple session based storage. All html request are first handled by Django, and if an action is coupled to the html request Django will initiate that action.

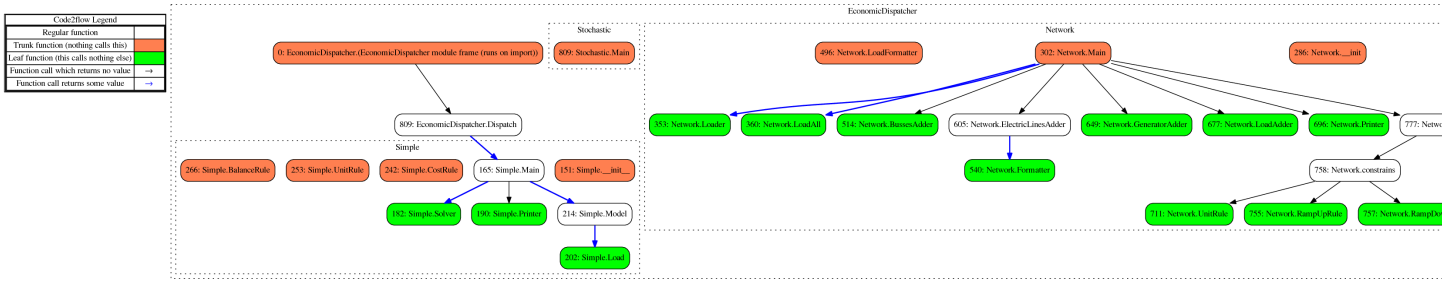
EconomicDispatcher

The inner workings of the 'EconomicDispatcher' may be seen in this figure An more in depth look into the simple and network case is made in ..

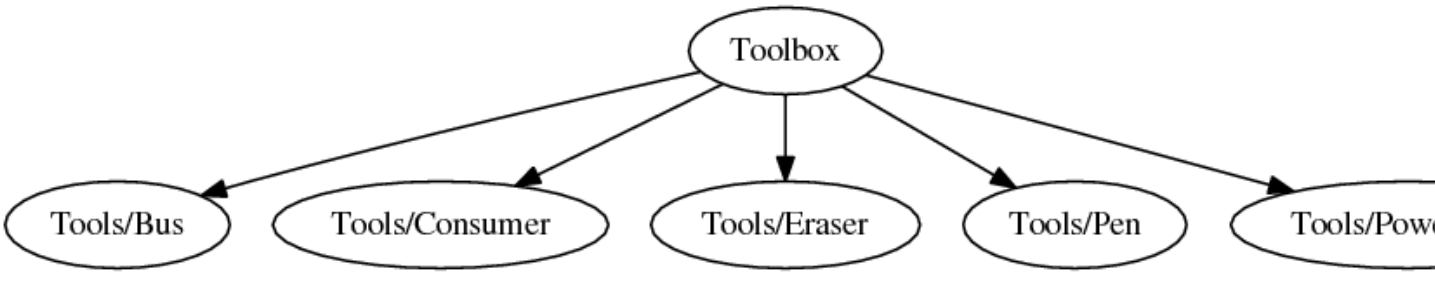
Frontend Node.Js

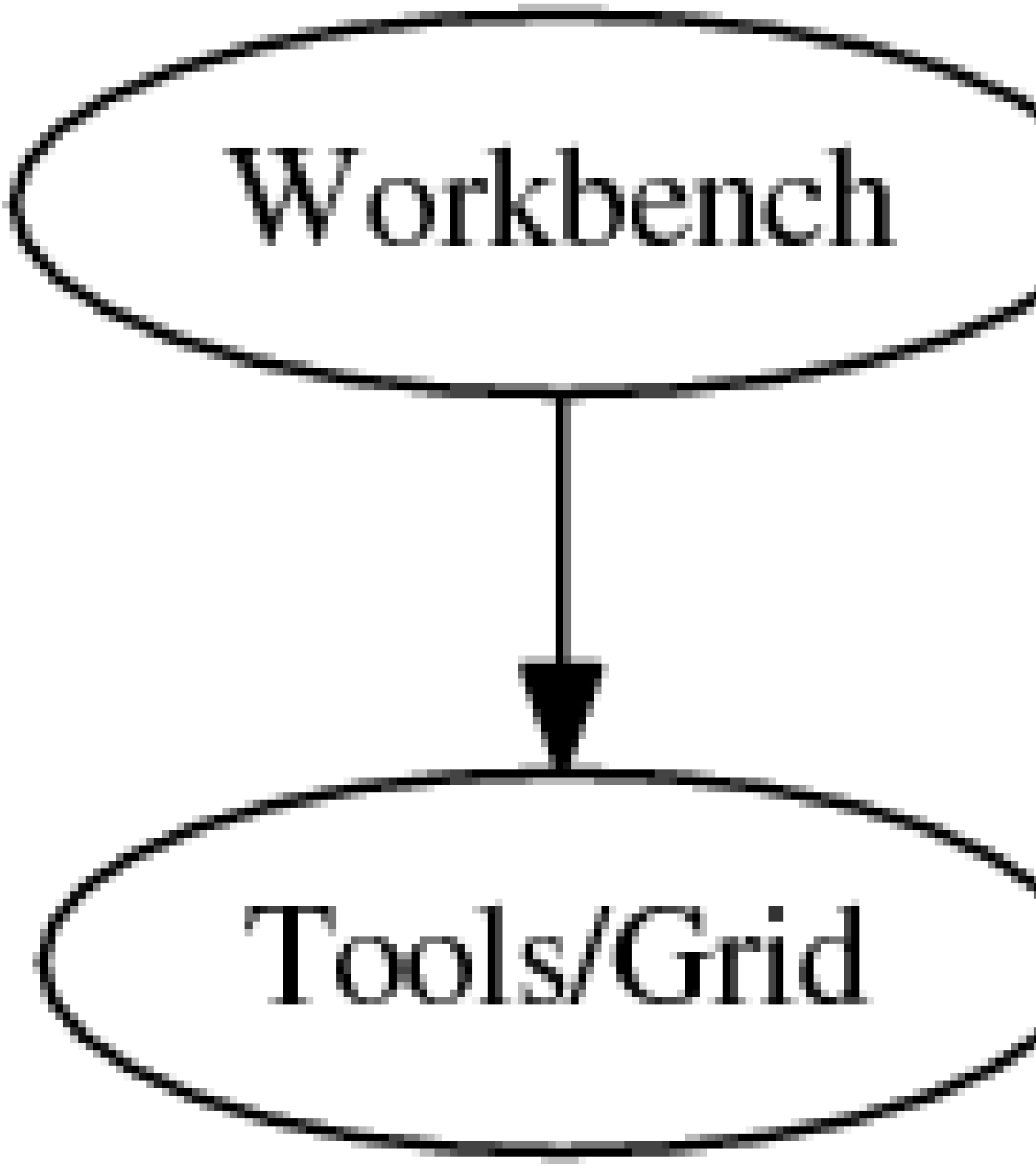

Python
dispatcher
simple
network
Node.Js
asset
Django Web Server
request
render
browser


Node.Js
asset
html request
index
/delsession
/session
/solve
data
JsonFormatter
EconomicDispatcher
browser


Code2flow Legend
Regular function
Trunk function (nothing calls this)
Leaf function (this calls nothing else)
Function call which returns no value
Function call returns some value
EconomicDispatcher
Stochastic
0: EconomicDispatcher.(EconomicDispatcher module frame (runs on import))
809: Stochastic.Main
809: EconomicDispatcher.Dispatch
Simple
266: Simple.BalanceRule
253: Simple.UnitRule
242: Simple.CostRule
165: Simple.Main
151: Simple.__init__
192: Simple.Solver
193: Simple.Printer
214: Simple.Model
202: Simple.Load
Network
496: Network.LoadFormatter
302: Network.Main
286: Network.__init
353: Network.Loader
360: Network.LoadAll
514: Network.BussesAdder
605: Network.ElectricLinesAdder
649: Network.GeneratorAdder
677: Network.LoadAdder
696: Network.Printer
540: Network.Formatter
758: Network.constrains
711: Network.UnitRule
755: Network.RampUpRule


Solver
Store
Tools/Bus
Tools/Consumer
Tools/Eraser
Tools/Grid
Tools/Pen
Tools/PowerPlant


Toolbox
Tools/Bus
Tools/Consumer
Tools/Eraser
Tools/Pen


Workbench
Tools/Grid

CHAPTER

FOUR

ELEMENTARY MODEL

The elementary model, uses the following objective function .. math:

```
\sum_{t=1}^{t=t_{end}} \Big( \sum_{i=1}^{i=j} P_{cost}[i] \cdot P[i,t]  \Big)
```

where t_{end} is the end time of the series, $P_{cost}[i]$ is the cost of generator at place i (e.g. the i th generator) $P[i,t]$ is the power that can be generated by an generator at time t

With the following constrains: The power delivered from the generator on time interval t cannot be larger then maximum power of the generator and should be higher then the minimum power the generator must deliver .. math:

```
P_{min}[i] \leq P[i,t] \leq P_{max}[i]
```

Where $P_{min}[i]$ is the minimum power generator i can deliver , $P[i,t]$ is the desired power at time t , $P_{max}[i]$ is the maximum power generator i can deliver

Generated power and load needed should be matched, the sum off all the powers at a time instance t needs to match the load at an time instance t .. math:

```
\sum_{i=1}^{i=j}  P[i,t]  = Load[t]
```

Where $Load[t]$ is the load at time instance t

As seen from above the elementary case does not incorporate technical constrains for the generators and the lines connecting them. This makes the linear program simpler and equires thus less computation time.

CHAPTER

FIVE

NETWORK MODEL

The network model is largely implemented in PyPSA it is recommend to read the full PyPSA documentation first. Wich can be found at ..

Components

Below you will find the components used in the web application, this list is a narrow sub list of the PyPSA documentation

component	list_name	description
Network	networks	Container for all components and functions which act upon the whole network.
Bus	buses	Electrically fundamental node where x-port objects attach.
Line	lines	Lines include distribution and transmission lines, overhead lines and cables.
LineType	line_types	Standard line types with per length values for impedances.
Link	links	Link between two buses with controllable active power - can be used for a transport power flow model OR as a simplified version of point-to-point DC connection OR as a lossy energy converter. NB: for a lossless bi-directional HVDC or transport link, set p_min_pu = -1 and efficiency = 1. NB: It is assumed that the links neither produce nor consume reactive power.
Load	loads	PQ power consumer.
Generator	generators	Power generator.

Network

The `Network` is the overall container for all components.

attribute	type	unit	default	description	status
name	string	n/a	n/a	Unique name	Input (required)
snapshots	list or pandas.Index	n/a	["now"]	List of snapshots or time steps. All time-dependent series quantities are indexed by `network.snapshots`. To reset the snapshots, call `network.set_snapshots(new_snapshots)`.	Input (optional)
buses	pandas.DataFrame	n/a	n/a	All static bus information compiled by PyPSA from inputs. Index is bus names, columns are bus attributes.	Output
buses_t	dictionary of pandas.DataFrames	n/a	n/a	All time-dependent bus information compiled by PyPSA from inputs. Dictionary keys are time-dependent series attributes, index is network.snapshots, columns are bus names.	Output
lines	pandas.DataFrame	n/a	n/a	All static line information compiled by PyPSA from inputs. Index is line names, columns are line attributes.	Output
lines_t	dictionary of pandas.DataFrames	n/a	n/a	All time-dependent line information compiled by PyPSA from inputs. Dictionary keys are time-dependent series attributes, index is network.snapshots, columns are line names.	Output
components	pandas.DataFrame	n/a	n/a	For each component type (buses, lines, etc.): static component information compiled by PyPSA from inputs. Index is component names, columns are component attributes.	Output
components_t	dictionary of pandas.DataFrames	n/a	n/a	For each component type (buses, lines, etc.): time-dependent component information compiled by PyPSA from inputs. Dictionary keys are time-dependent series attributes, index is network.snapshots, columns are component names.	Output
branches()	pandas.DataFrame	n/a	n/a	Dynamically generated concatenation of branch DataFrames: network.lines, network.transformers and network.links. Note that this is a copy and therefore changing entries will NOT update the original.	Output

Bus

The bus is the fundamental node of the network, to which components like loads, generators and transmission lines attach. It enforces energy conservation for all elements feeding in and out of it (i.e. like Kirchhoff's Current Law).

attribute	type	unit	default	description	status
name	string	n/a	n/a	Unique name	Input (required)
v_nom	float	kV	1.	Nominal voltage	Input (optional)
type	string	n/a	n/a	Placeholder for bus type. Not yet implemented.	Input (optional)
v_mag_pu_set	static or series	per unit	1.	Voltage magnitude set point, per unit of v_nom.	Input (optional)
v_mag_pu_min	float	per unit	0.	Minimum desired voltage, per unit of v_nom	Input (optional)
v_mag_pu_max	float	per unit	inf	Maximum desired voltage, per unit of v_nom	Input (optional)
control	string	n/a	PQ	P,Q,V control strategy for PF, must be "PQ", "PV" or "Slack". Note that this attribute is an output inherited from the controls of the generators attached to the bus; setting it directly on the bus will not have any effect.	Output
sub_network	string	n/a	n/a	Name of connected sub-network to which bus belongs. This attribute is set by PyPSA in the function network.determine_network_topology(); do not set it directly by hand.	Output
p	series	MW	0.	active power at bus (positive if net generation at bus)	Output
q	series	MVar	0.	reactive power (positive if net generation at bus)	Output
v_mag_pu	series	per unit	1.	Voltage magnitude, per unit of v_nom	Output
v_ang	series	radians	0.	Voltage angle	Output

Generator

attribute	type	unit	default	description	status
name	string	n/a	n/a	Unique name	Input (required)
bus	string	n/a	n/a	name of bus to which generator is attached	Input (required)
control	string	n/a	PQ	P,Q,V control strategy for PF, must be “PQ”, “PV” or “Slack”.	Input (optional)
p_nom	float	MW	0.	Nominal power for limits in OPF.	Input (optional)
p_min_pu	static or series	per unit	0.	The minimum output for each snapshot per unit of p_nom for the OPF (e.g. for variable renewable generators this can change due to weather conditions and compulsory feed-in; for conventional generators it represents a minimal dispatch). Note that if comittable is False and p_min_pu > 0, this represents a must-run condition.	Input (optional)
p_max_pu	static or series	per unit	1.	The maximum output for each snapshot per unit of p_nom for the OPF (e.g. for varialbe renewable generators this can change due to weather conditions; for conventional generators it represents a maximum dispatch).	Input (optional)
p_set	static or series	MW	0.	active power set point (for PF)	Input (optional)
q_set	static or series	MVar	0.	reactive power set point (for	Input (optional)

Lines

attribute	type	unit	default	description	status
name	string	n/a	n/a	Unique name	Input (required)
bus0	string	n/a	n/a	Name of first bus to which branch is attached.	Input (required)
bus1	string	n/a	n/a	Name of second bus to which branch is attached.	Input (required)
type	string	n/a	n/a	Name of line standard type. If this is not an empty string “”, then the line standard type impedance parameters are multiplied with the line length and divided/multiplied by num_parallel to compute x, r, etc. This will override any values set in r, x, and b. If the string is empty, PyPSA will simply read r, x, etc.	Input (optional)
x	float	Ohm	0.	Series reactance; must be non-zero for AC branch in linear power flow. If the line has series inductance L in Henries then $x = 2\pi f L$ where f is the frequency in Hertz. Series impedance $z = r + jx$ must be non-zero for the non-linear power flow. Ignored if type defined.	Input (required)
r	float	Ohm	0.	Series resistance; must be non-zero for DC branch in linear power flow. Series impedance $z = r + jx$ must	Input (required)

Loads

attribute	type	unit	default	description	status
name	string	n/a	n/a	Unique name	Input (required)
bus	string	n/a	n/a	Name of bus to which load is attached.	Input (required)
type	string	n/a	n/a	Placeholder for load type. Not yet implemented.	Input (optional)
p_set	static or series	MW	0.	Active power consumption (positive if the load is consuming power).	Input (optional)
q_set	static or series	MVar	0.	Reactive power consumption (positive if the load is inductive).	Input (optional)
sign	float	n/a	-1.	power sign (opposite sign to generator)	Input (optional)
p	series	MW	0.	active power at bus (positive if net load)	Output
q	series	MVar	0.	reactive power (positive if net load)	Output

CHAPTER

SIX

EXPERIMENTAL STAND ALONE VERSION

CHAPTER

SEVEN

CODE DOCUMENTATION

Economic Dispatcher

Purpose of this module is to dispatch between the different types of economic dispatchers and calculated the minimum objective function

EconomicDispatcher.**Dispatch**(*model*, **data*)

Args:

Returns: json file with the solutions of the model

class EconomicDispatcher.**Network**

Bases: object

Network constrained optimization problem

Parameters

- **file** –
- **solver** –

Returns Json file with simple marketclearing

Return type solotion

BussesAdder(*network*, *Buss*, *BussVnom*, *BussVmax*, *BussVmin*, *BussVset*, *BussType*)

Add all the buses to the network component

Parameters

- **()** (*BussType*) – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend

Returns:

ElectricLinesAdder(*network*, *ElectricLines*, *ElectricLinesR*, *ElectricLinesX*, *ElectricLinesCcap*, *ElectricLinesL*, *ElectricLinesPhaseShift*, *ElectricLinesAngMax*, *ElectricLinesAngMin*, *ElectricLinesLength*, *ElectricLinesConnection*, *ElectricLinesPmax*)

Add the lines between buses to the network component

Parameters

- **()** (*ElectricLinesPmax*) – pypsa network component
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend

Returns:

GeneratorAdder(*network*, *Generators*, *GeneratorConnection*, *PnomOrdered*, *PminOrdered*, *Pmax-Ordered*, *CostsOrdered*, *RampUp*, *RampDown*, *MaxDownTime*, *MinDownTime*, *Initstatus*, *StartUpCost*, *ShutdownCost*, *Effciency*, *CapCost*, *PsetPoint*, *QsetPoint*, *Time*)

Add the generators to the network component

Parameters

- **()** (*QsetPoint*) – pypsa network component
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend
- **()** – Pandas dataframework from frontend

Returns:

LoadAdder(*network*, *LoadName*, *Loads*, *LoadQ*, *LoadBus*)

Add loads to the network

ToDo : Qload

Parameters

- **LoadName** – List of Load names
- **Loads** – List of loads that need to be edited
- **LoadBus** – List of the busses where the load is connected to

Main(*model*, *ElectricLines*, *ElectricLinesR*, *ElectricLinesC*, *ElectricLinesX*, *Buss*, *BussVnom*, *BussType*, *BussVset*, *BussVmax*, *BussVmin*, *BussConnection*, *ElectricLinesCcap*, *ElectricLinesL*, *ElectricLinesPhaseShift*, *ElectricLinesAngMax*, *ElectricLinesAngMin*, *ElectricLinesLength*, *ElectricLinesConnection*, *ElectricLinesPmax*, *Generators*, *CostsOrdered*, *PnomOrdered*, *RampUp*, *RampDown*, *Effciency*, *MinDownTime*, *MaxDownTime*, *PminOrdered*, *GeneratorConnection*, *Time*, *PowerBlocks*, *PmaxOrdered*, *StartUpCost*, *ShutdownCost*, *Initstatus*, *CapCost*, *PsetPoint*, *QsetPoint*, *Loads*, *LoadP*, *LoadQ*, *LoadTime*, *LoadConnection*, *options*)

Main network function that calls necessary network functions

Parameters

- **()** (*options*) – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** –
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend

- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend
- **()** – Pandas Data Framework from the frontend

Returns:

Printer(*network*)

Debugger that prints info

Parameters **network** – pypsa main network component

Returns:

lopf(*network*)

Function that runs the optimization procces

Returns:

class EconomicDispatcher.**Simple**

Bases: object

Simple optimization problem, using only 3 basic constrains: Unit, Balance , Cost

Parameters

- **file** – file to be loaded with all the data
- **solver** – string that defines which optimization solver to be used e.g. the free software GLPK or the commercial software Gurobi

Returns:

BalanceRule(*model*, *t*)

Generated power and load needed should be matched, the sum off all the powers at a time instance t needs to match

$$\sum_{i=1}^{i=j} P[i,t] = Load[t]$$

Where $Load[t]$ is the load at time instance t

CostRule(*model*)

Defining the cost rule that pyomo uses

$$\sum_{t=1}^{t=t_{end}} \Big(\sum_{i=1}^{i=j} P_{cost}[i] \cdot P[i,t] \Big)$$

where t_{end} is the end time of the series, $P_{cost}[i]$ is the cost of generator at place i (e.g. the i th generator) $P[i,t]$ is the power that can be generated by an generator at time t

Load()
Function that loads the data from an excel sheets in order to further process it.

Main(*Pmax*, *Pmin*, *CostsOrdered*, *LoadP*)
Main function of the simple network model

Returns solotion

Model(*Pmax*, *Pmin*, *CostsOrdered*, *LoadP*)
Initialises the model in the pyomo framework

Printer(*model*)
For debugging purpose only prints the output of the result if the debug option has been set

Solver(*model*)
Solves the linear objective function

Returns results

UnitRule(*model*, *i*, *t*)

Power delivered from the generator on time interval t cannot be larger then maximum power of the generator and sl

$$P_{min}[i] \leq P[i,t] \leq P_{max}[i]$$

Where $P_{min}[i]$ is the minimum power generator i can deliver , $P[i,t]$ is the desired power at time t , $P_{max}[i]$ is the maximum power generator i can deliver

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e