Web App For Day Ahead Market Clearing Documentation

Release 0.1

Martyn van Dijke
Yuanlong Li

Mar 07, 2017

CONTENTS

CHAPTER

ONE

INSTALLATION

Getting Python

For those rolling their own on unix-like systems (GNU/Linux, Mac OS X) it's always helpful to use a virtual environment for your python installation (and even easier to use with a virtualenv-burrito), in case you accidentally trash something.

Getting a solver for linear optimisation

The web app is known to work with the free software GLPK and the non-free software Gurobi (and whatever else Pyomo works with).

For Debian-based systems you can get GLPK with:

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

and there are similar packages for other GNU/Linux distributions.

For Windows there is WinGLPK. For Mac OS X brew is your friend.

Installing the web app

If you have the Python package installer `pip` wich is recommended then just run:

```
pip install -r requirements.txt
```

On linux machines please be sure to use adminitriva rights:

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

Dependencies

PyPSA relies on the following packages which are not contained in a standard Python installation:

- numpy
- scipy
- pandas

- networkx
- pyomo
- moreitertools
- Django

CHAPTER

TWO

RUNNING THE WEB APP

Setting everything up

Make sure port 80 is still open on your machine and is not being blocked by you firewall

CHAPTER

THREE

CODE DOCUMENTATION

Economic Dispatcher

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

class EconomicDispatcher.**Network**

Bases: object

Network contrained optimization problem

Parameters

- **file** –
- **solver** –

Returns Json file with simple marketclearing

Return type solotion

BussesAdder(*network*, *UniqueElectricBusses*)

Add all the unique busses to the network

Parameters

- **network** – pypsa main network component
- **UniqueElectricBusses** – List of all of the unique busses that needs to be added to the network.

Returns:

ElectricLinesAdder(*network*, *ElectricLines*, *ElectricLinesX*, *ElectricBusses*)

Add the electric lines between the bus and the other busses

Parameters

- **network** – pypsa main network component
- **ElectricLines** – List of all the electric lines that need to be added
- **ElectricLinesX** – List of all the electric ressitance that need to be added
- **ElectricBusses** – List of all the electric busses that the lines need to connect to

Returns:

Formatter(*i*, *k*, *Lines*, *ElectricLines*, *Slines*, *Rlines*, *Glines*, *Blines*, *CapCostLines*, *LinesLength*, *Num_Parrels*, *VangMin*, *VangMax*, *V_nom*)

Split the list of lines into individual line components, if present also split the extra options of the lines into individual components

Parameters

- **()** (*k*) –
- **Lines** – List of Lines
- **ElectricLines** –
- **Slines** – List of
- **Rlines** – List of ressitance
- **Glines** – List of conductance
- **Blines** – List of
- **CapCostLines** – Capital cost of
- **LinesLength** – List of length of lines
- **Num_Parrels** – List of the number of lines in parrel
- **VangMin** – List of the minimim angle
- **VangMax** – List of the maxumim anlge
- **V_nom** – List of v_nom

Function returns the individual extra options

Returns LineS, LineR, LineG, LineB, LineCost, LineLength, LineAngMax, LineAngMin

GeneratorAdder(*network*, *Generators*, *UnitData*, *ElectricBusses*, *PnomOrdered*, *PminOrdered*, *PmaxOrdered*, *CostsOrdered*)

Add the generators to the network

Parameters

- **Generators** – List of generators
- **UnitData** – Pandas dataframe of the excel sheet
- **ElectricBusses** – List of Electric busses
- **PnomOrdered** – Ordered list of the nominal power
- **PminOrdered** – Ordered list of the minimal norminal power
- **PmaxOrdered** – Ordered list of the max norminal power
- **CostsOrdered** – List of the marginal cost of operation

Returns:

GeneratorFormatter(*UnitData*, *i*)

Loads extra options for the generators and formats them into the desired value

Parameters

- **Sheet1** – Pandas dataframework sheet with the generator values
- **i** – number of generators

Returns Efficieny, CapCost, power, QPower

LinesFormatter()
Loads extra options for the lines and formats them into the desired value

Returns Slines, Rlines, Glines, Blines, CapCostLines, LinesLength, Num_Parrels, VangMin, VangMax, V_no

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

Add loads to the network

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

Returns:

LoadAll()
Get all the data from excel and turn this data into list without the index (first coloum). If the excel data contains any nan's replace them with the default value.

Returns: ElectricBusses,ElectricLines,ElectricLinesX,Generators,CostsOrdered,PminOrdered,PmaxOrdered,Pnom
Loads, LoadBus, LoadName,LoadTime, UnitDataColunms, LoadColunms, UniqueElectricBusses

LoadFormatter(*Load*, *i*)
Loads extra options for the load and formats them into the desired value

Parameters

- **Load** – Pandas dataframework sheet with the load values
- **i** – number of loads

Returns Qload

Loader()
Loads only the excel sheet and returns the sheets of the excel as pandas dataframeworks

Returns Sheet1,Sheet2

Main()
Main function for the Network model, this function will load all the sub functions in correct order.

Returns solution

Printer(*network*)
Debugger that prints info

Parameters **network** – pypsa main network component

Returns:

RampDownRule(*network*, *snaphots*)
Todo ..

Parameters

- **()** (*snaphots*) –
- **()** –

Returns:

RampUpRule(*network*, *snaphots*)
Todo ..

Parameters

- **()** (*snaphots*) –
- **()** –

Returns:

UnitRule(*network*, *snaphots*)
Todo ..

Parameters

- **()** (*snaphots*) –
- **()** –

Returns:

constrains(*network*, *snapshots*, *Unit*, *RampUp*, *RampDown*)
Function that calulates extra constrains if they have been selected

Parameters

- **()** (*snapshots*) –
- **()** –
- **Unit** – boolean if the unit data rule needs to be added
- **RampUp** – boolean if rampup contraint needs to be calculated
- **RampDown** – boolean if rampdown contraint needs to be calculated

Returns:

lopf(*network*, *Unit*, *RampUp*, *RampDown*)
Function that runs the optimization procces

Parameters

- **Unit** – boolean if the unit data rule needs to be added
- **RampUp** – boolean if rampup contraint needs to be calculated
- **RampDown** – boolean if rampdown contraint needs to be calculated

Returns:

class EconomicDispatcher.**Simple**(*file*, *solver*)
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

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

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(*file*, *solver*)
Main function of the simple network model

Returns solotion

Model()
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

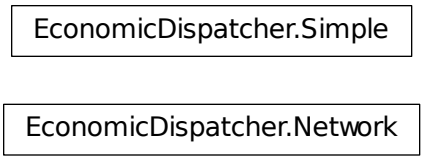
UnitRule(*model*, *i*, *t*)

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

EconomicDispatcher Economic Dispatcher


EconomicDispatcher.Simple
EconomicDispatcher.Network

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

EconomicDispatcher, **??**