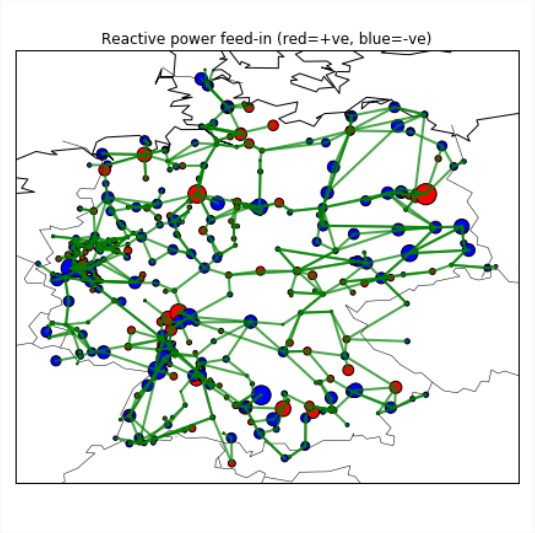
Python for Power System Analysis (PyPSA)

Tom Brown, Jonas Hörsch, David Schlachtberger
Frankfurt Institute for Advanced Studies (FIAS), University of Frankfurt

6th Open Energy Modelling Initiative Workshop, Frankfurt, 19th April 2017

Python for Power System Analysis (PyPSA)

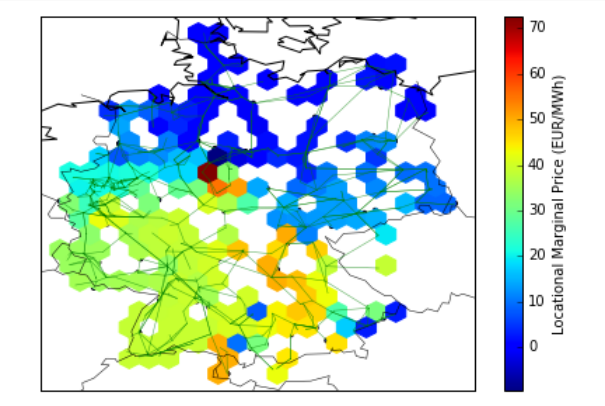
- Developed at Frankfurt Institute of Advanced Studies by Tom Brown, Jonas Hörsch and David Schlactberger for the CoNDyNet project.
- Fills missing gap between load flow software (e.g. MATPOWER, PYPOWER, PowerFactory) and energy system simulation software (e.g. TIMES, calliope, oemof, OSeMOSYS).
- Good grid modelling is increasingly important, for integration of renewables and possible electrification of transport and heating.



Python for Power System Analysis (PyPSA)

The FIAS software PyPSA is online at `http://pypsa.org/` and on github. It can do:

- Static power flow
- Linear optimal power flow (multiple periods, unit commitment, storage, coupling to other sectors)
- Security-constrained linear optimal power flow
- Total electricity system investment optimisation



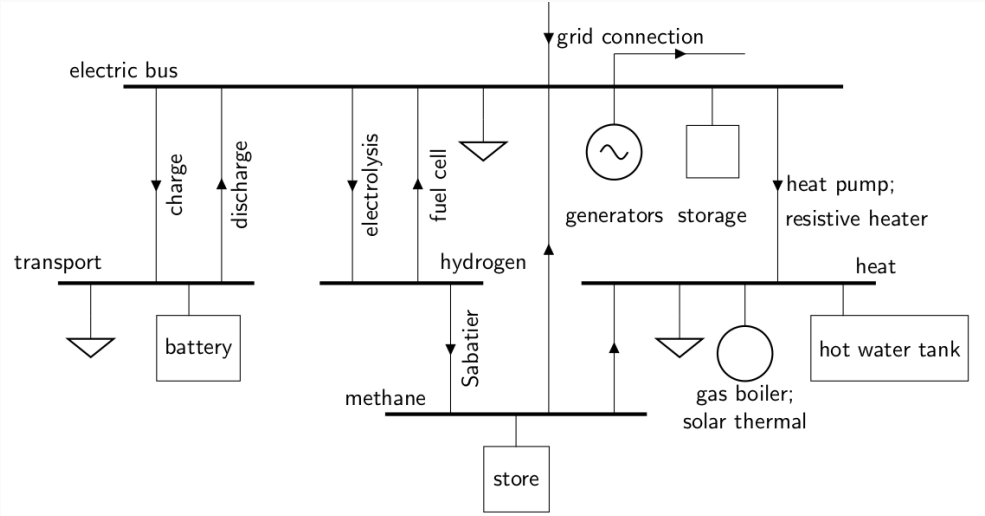
Models

It has models for:

- meshed multiply-connected AC and DC networks, with controllable converters between AC and DC networks
- standard types for lines and transformers following the implementation in pandapower
- conventional dispatchable generators with unit commitment
- generators with time-varying power availability, such as wind and solar generators
- storage units with efficiency losses
- simple hydroelectricity with inflow and spillage
- coupling with other energy carriers
- basic components out of which more complicated assets can be built, such as Combined Heat and Power (CHP) units, heat pumps, resistive Power-to-Heat (P2H), Power-to-Gas (P2G), battery electric vehicles (BEVs)

Models

Example of sector-coupling:



Components

Network	Container for all other network components.
Bus	Fundamental nodes to which all other components attach.
Load	A consumer of energy.
Generator	Generator whose feed-in can be flexible subject to minimum loading or minimum down and up times, or variable according to a given time series of availability.
Storage Unit	A device which can shift energy from one time to another, subject to efficiency losses.
Store	A more fundamental storage object with no restrictions on charging or discharging power.
Shunt Impedance	An impedance in shunt to a bus.
Line	A branch which connects two buses of the same voltage.
Transformer	A branch which connects two buses of different voltages.
Link	A branch with a controllable power flow between two buses.

Technical details

PyPSA is written and tested to be compatible with Python 2.7 and Python 3.5.

It leans heavily on the following Python packages:

- pandas for storing data about components and time series
- numpy and scipy for calculations, such as linear algebra and sparse matrix calculations
- pyomo for preparing optimisation problems (currently only linear)
- networkx for some network calculations
- py.test for unit testing
- logging for managing messages

Design

- Network object is the overall container
- Components (buses, lines, transformers, links, generators, storage units) have static properties (nominal power, etc.) stored in pandas DataFrames
- Time-varying data (e.g. load, solar/wind availability) also stored in pandas DataFrames, indexed by a list of 'snapshots'
- Pyomo for optimisation
- Internal use of per unit power system quantities
- Set points (e.g. for loads, generators) are stored separately from actual dispatch points
- No GUI, use Jupyter notebooks

Power flow

In a power flow calculation, the user specifies the power dispatch of all dispatchable components (loads, generators, storage units, stores and links) and then PyPSA computes the resulting voltages in the network and hence the power flows in passive branches (lines and transformers) based on their impedances. For an AC network we must have:

$$S_n = V_n I_n^* = \sum_m V_n Y_{nm}^* V_m^*$$

where $S_n = P_n + jQ_n$ is the apparent power at node n, I_n is the complex current, V_n is the complex voltage and Y_{nm} is the bus admittance matrix.

For short overhead transmission lines that are close to their natural loading, these equations can be linearised to:

$$P_n = \sum_m (KBK^T)_{nm}\theta_m - \sum_\ell K_{n\ell} b_\ell \theta_\ell^{\text{shift}}$$

where K is the incidence matrix of the network, B are the series susceptances, θ_n are the voltage angles and $\theta_\ell^{\text{shift}}$ are transformer phase-shifts.

PyPSA: Full non-linear power flow with Newton-Raphson

Benchmarked against PYPOWER and IEEE test cases:

```
import pypsa
from pypower.api import case118 as case

ppc = case()
network = pypsa.Network()
network.import_from_pypower_ppc(ppc)
network.pf()

print(network.generators_t.p)
print(network.generators_t.q)
print(network.buses_t.v_ang)
print(network.buses_t.v_mag)
```

Optimal power flow

PyPSA minimises the costs of generator dispatch $g_{n,s,t}$ at each node at each time t, and investment in generator $G_{n,s}$ and transmission capacity F_ℓ:

$$\min_{\substack{G_{n,s},F_\ell,\\ g_{n,s,t},f_{\ell,t}}} \left[\sum_{n,s,t} w_t o_{n,s} g_{n,s,t} + \sum_{n,s} c_{n,s} G_{n,s} + \sum_{\ell} c_\ell F_\ell \right]$$

subject to physical constraints (meeting energy demand, generator limits, storage consistency, physical flow conditions, thermal limits of branches).

Main constraints 1/2

- Demand $d_{n,t}$ is always met by generation or transmission

$$d_{n,t} = \sum_{s} g_{n,s,t} + \sum_{\ell \in n} f_{\ell,t}$$

- Dispatch $g_{n,s,t}$ cannot exceed availability $\bar{g}_{n,s,t}$

$$0 \leq g_{n,s,t} \leq \bar{g}_{n,s,t} \leq \bar{g}_{n,s}$$

- Installed capacity cannot exceed the installable potential $\hat{g}_{n,s}$

$$\bar{g}_{n,s} \leq \hat{g}_{n,s}$$

- CO_2 constraint is respected

$$\sum_{n,s,t} \frac{1}{\eta_{n,s}} g_{n,s,t} e_{n,s} \leq CAP$$

Main constraints 2/2

- Storage consistency

$$e_{n,s,t} = \eta_0^{w_t} e_{n,s,t-1} - \eta_1 w_t \left[g_{n,s,t}\right]^- + \eta_2^{-1} w_t \left[g_{n,s,t}\right]^+$$

- Flows $f_{\ell,t}$ have to by physical, i.e. obey Kirchhoff's Current and Voltage Laws for the linear load flow.
- New cycle-based formulations of the linear power flow are provided which are up to 20 times faster than standard angle-based formulations, see arXiv:1704.01881.
- Transmission flows cannot exceed capacities

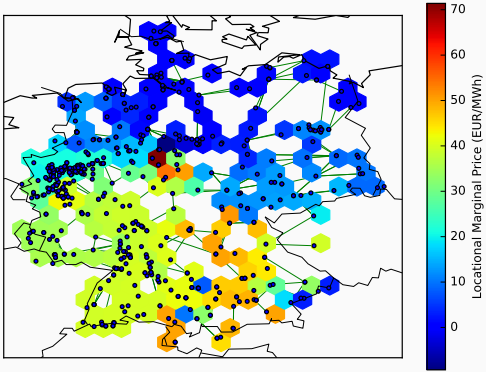
$$|f_{\ell,t}| \leq \bar{P}_\ell$$

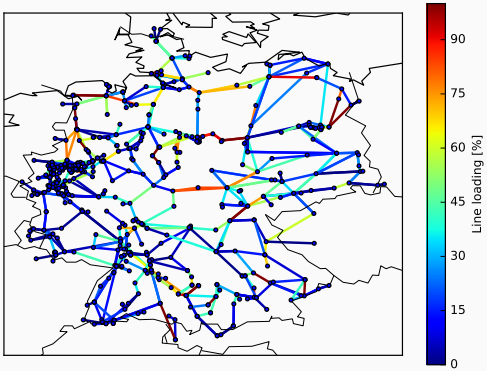
Example: linear OPF for a mixed AC-DC network

```
import pypsa

csv_folder_name=‘‘path/to/ac-dc-meshed/ac-dc-data’’

network = pypsa.Network(csv_folder_name)

solver_name = ‘‘glpk’’

network.lopf(snapshots=network.snapshots,solver_name)

print(network.generators.p_nom)  #series for each gen

print(network.generators_t.p)  #dataframe for each gen, snapshot
```

Example of LOPF in Germany

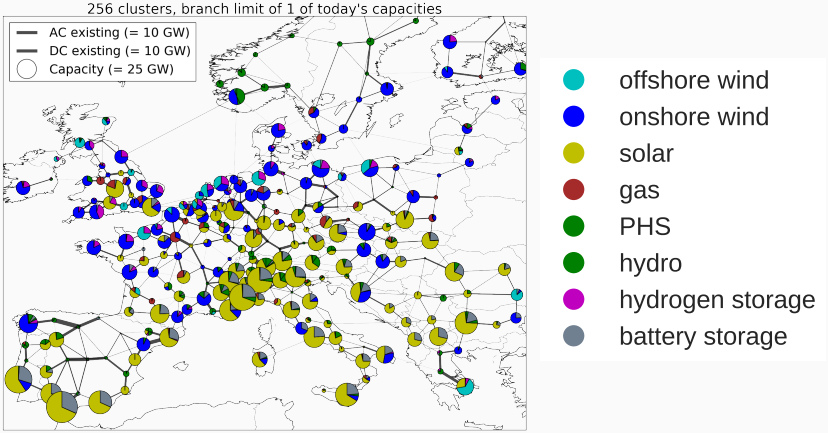
Using the **SciGRID dataset**, nodal pricing for a day of high wind shows divergent prices in North and South Germany due to line overloading:





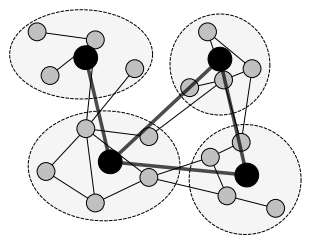
Example of investment optimisation in Europe

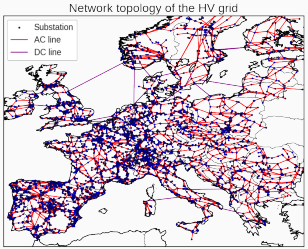
Using the **GridKit dataset**, investment optimisation for a 95% CO_2 reduction compared to 1990, with no grid expansion allowed, results in solar+batteries in Southern Europe and wind+hydrogen storage in the North:

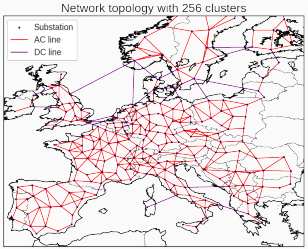


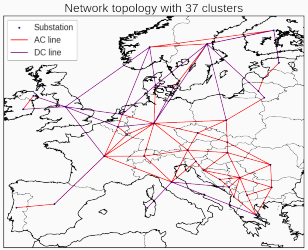
Network Clustering

Can cluster down the network, to reduce resolution while retaining important transmission lines:









Python for Power System Analysis (PyPSA)

- Documentation and example Jupyter notebooks showcasing open data, like SciGRID: `http://pypsa.org/`
- Github: `https://github.com/FRESNA/PyPSA`
- Mailing list: `https://groups.google.com/forum/#!forum/pypsa`

PyPSA users

PyPSA is being actively used by around a dozen institutions (that we know of...) and the website has been visited by people from 120+ countries: