Just a few imports to start :)

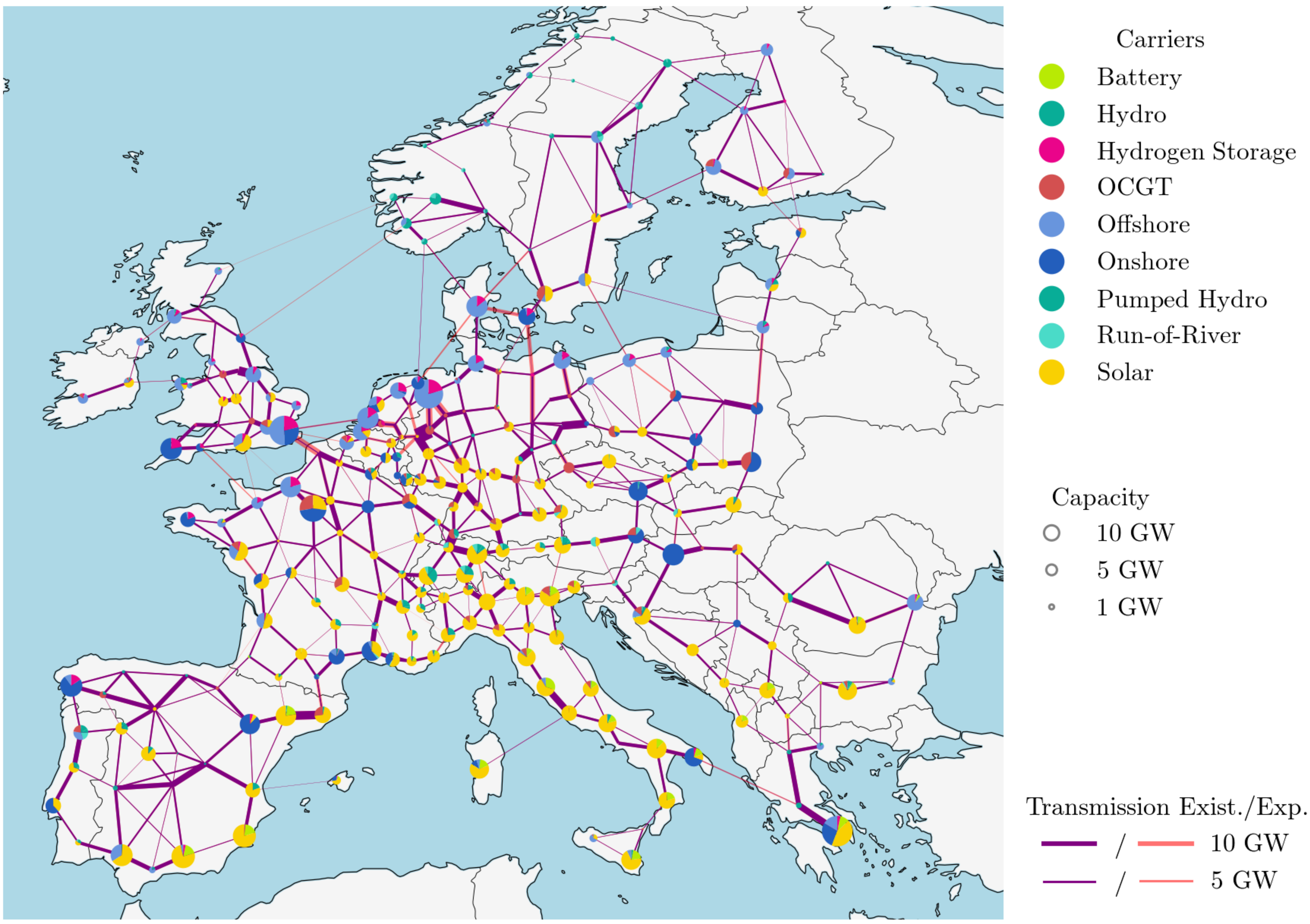
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('bmh')
%matplotlib inline
```

Python for Power System Analysis (PyPSA)

Free Software for Planning Energy Systems with High Shares of Renewables

Fabian Neumann, Karlsruhe Institute of Technology (KIT)

- **Website:** www.pypsa.org
- **Documentation:** www.pypsa.readthedocs.io
- **Github:** www.github.com/pypsa/pypsa
- **Group Homepage:** www.iai.kit.edu/english/esm.php

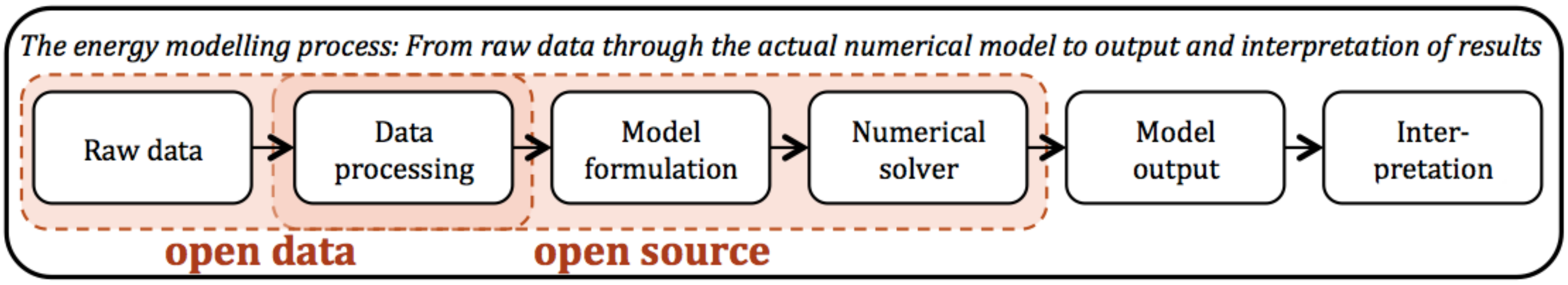


The Energy System Modelling Group @ KIT

Research Agenda:

- **Cost-effective pathways** to reduce greenhouse gas emissions with cross-sectoral energy system models
- **Co-optimisation** of generation, storage and transmission infrastructure to analyse the multitude of investment trade-offs
- **Power grid reinforcement** requirements with large shares of wind and solar generation, and increased energy trade
- **Algorithms** to increase the tractability of energy system and grid optimisation (spatial, temporal clustering, model reduction)
- **Open Source** software development and open data to enhance tranparency and reproducibility

Open Energy System Modelling:



There's an initiative for that: www.openmod-initiative.org/

Today:

- [PyPSA (www.github.com/pypsa/pypsa)](www.github.com/pypsa/pypsa)**:** Power System Analysis Toolbox
- [PyPSA-Eur (www.github.com/pypsa/pypsa-eur)](www.github.com/pypsa/pypsa-eur)**:** Open model data set of the European transmission system *(if time allows)*

Warm-up Questions

- familiar with power flow equation (non-linear and linearised)?
- familiar with optimisation problems?
- user of **any** modelling language (GAMS, JuMP, pyomo, ...)?
- user of Python?
- user of the pandas?
- user of the pyomo?
- heard of PyPSA?
- user of PyPSA?

Please Do Ask Questions Anytime and Interrupt Me!

So let's get started with PyPSA...

PyPSA bridges load flow analysis and energy system modelling.

Power Flow Software	Energy System Modelling Software
MATPOWER (MATLAB)	TIMES
PYPOWER (Python)	OSeMOSYS
pandapower (Python)	calliope
PowerFactory (commercial)	oemof

PyPSA can do:

- static power flow
- linear optimal power flow
    - multiple periods
    - unit commitment
    - storage
    - coupling to other sectors/carriers
- security-constrained linear optimal power flow
- total energy system investment optimisation
    - co-optimization of generation, transmission and storage expansion
    - with linear optimal power flow
    - **no unit commitment** if generator is extendable *(this would make the problem nonlinear)*

> https://pypsa.readthedocs.io/en/latest/introduction.html (https://pypsa.readthedocs.io/en/latest/introduction.html)

Installation

PyPSA is compatible with **Python 2** and **Python 3**!

Solver *(depending on your system)*

```
sudo apt-get install coinor-cbc
```

or

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

PyPSA with `conda` or `pip` or manually *(for the brave)*

```
conda env create -n mypypsa
conda activate mypypsa
conda install -c conda-forge pypsa
```

or

```
pip install pypsa
```

Overview of Network Components

In [2]:
```python
import pypsa
```

The `Network` is an overall container for all network components.

In [3]:
```python
network = pypsa.Network()
```

Per unit values of voltage and impedance are used internally for network calculations. It is assumed internally that the base power is **1 MVA**. The base voltage depends on the component.

> https://pypsa.readthedocs.io/en/latest/design.html#unit-conventions (https://pypsa.readthedocs.io/en/latest/design.html#unit-conventions)

Buses

The bus is the fundamental node to which all loads, generators, storage units, lines, transformers and HVDC links attach.

In [4]:
```
network.components['Bus']['description']
```
Out[4]:
```
'Electrically fundamental node where x-port objects attach.'
```

In [5]:
```
network.components['Bus']['attrs'];
```

Let's add three buses

In [6]:
```
for i in range(3):
    network.add("Bus","My bus {}".format(i))
network.buses;
```

In [7]:
```
type(network.buses)
```
Out[7]:
```
pandas.core.frame.DataFrame
```

To enable efficient calculations on the different dimensions of the data, data is stored in memory using `pandas DataFrames` (http://pandas.pydata.org/).

Carrier

In [8]:
```
network.components['Carrier']['description']
```
Out[8]:
```
'Energy carrier, such as AC, DC, heat, wind, PV or coal. Buses have direct carriers and Generators indic
ate their primary energy carriers.  The Carrier can track properties relevant for global constraints, su
ch as CO2 emissions.'
```

In [9]:
```
network.components['Carrier']['attrs'];
```

Generators

In [10]:
```
network.components['Generator']['description']
```
Out[10]:
```
'Power generator.'
```

In [11]:
```
network.components['Generator']['attrs'];
```

Let's add a generator at `bus 0`

In [12]:
```
network.add("Generator","My gen 0",
            bus="My bus 0",
            p_set=100)
network.generators;
```

Global Constraint

In [13]:
```
network.components['GlobalConstraint']['description']
```
Out[13]:
```
'Constraints for OPF that affect many components, such as CO2 emission constraints.'
```

In [14]:
```
network.components['GlobalConstraint']['attrs'];
```

Line

In [15]:
```
network.components['Line']['description']
```
Out[15]:
```
'Lines include distribution and transmission lines, overhead lines and cables.'
```

In [16]:
```
network.components['Line']['attrs'];
```

Let's add three lines in a ring

In [17]:
```
for i in range(3):
    network.add("Line","My line {}".format(i),
                bus0="My bus {}".format(i),
                bus1="My bus {}".format((i+1)%3),
                x=0.0001)
network.lines;
```

Line Types

In [18]:
```
network.components['LineType']['description']
```
Out[18]:
```
'Standard line types with per length values for impedances.'
```

In [19]:
```
network.components['LineType']['attrs'];
```

In [20]:
```
network.line_types;
```

Transformer

In [21]:
```
network.components['Transformer']['description']
```
Out[21]:
```
'2-winding transformer.'
```

In [22]:
```
network.components['Transformer']['attrs'];
```

Transformer Types

In [23]:
```
network.components['TransformerType']['description']
```
Out[23]:
```
'Standard 2-winding transformer types.'
```

In [24]:
```
network.components['TransformerType']['attrs'];
```

In [25]:
```
network.transformer_types;
```

Link

In [26]:
```
network.components['Link']['description']
```
Out[26]:
```
'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 lossle
ss bi-directional HVDC or transport link, set p_min_pu = -1 and efficiency = 1. NB: It is assumed that t
he links neither produce nor consume reactive power.'
```

In [27]:
```
network.components['Link']['attrs'];
```

Load

In [28]:
```
network.components['Load']['description']
```
Out[28]:
```
'PQ power consumer.'
```

In [29]:
```
network.components['Load']['attrs'];
```

Let's add a load at `bus 2`

In [30]:
```
network.add("Load","My load",
            bus="My bus 2",
            p_set=100)
network.loads;
```

Storage Unit

In [31]:
```
network.components['StorageUnit']['description']
```
Out[31]:
```
'Storage unit with fixed nominal-energy-to-nominal-power ratio.'
```

In [32]:
```
network.components['StorageUnit']['attrs'];
```

Store

In [33]:
```
network.components['Store']['description']
```
Out[33]:
```
'Generic store, whose capacity may be optimised.'
```

In [34]:
```
network.components['Store']['attrs'];
```

Shunt Impedance

In [35]:
```
network.components['ShuntImpedance']['description']
```
Out[35]:
```
'Shunt y = g + jb.'
```

In [36]:
```
network.components['ShuntImpedance']['attrs'];
```

Network (revisited)

In [37]:
```
network.components['Network']['attrs'];
```

Power Flow

(no solver required)

Full non-linear power flow

> [https://pypsa.readthedocs.io/en/latest/power_flow.html#full-non-linear-power-flow (https://pypsa.readthedocs.io/en/latest/power_flow.html#full-non-linear-power-flow)](https://pypsa.readthedocs.io/en/latest/power_flow.html#full-non-linear-power-flow)

In [38]:
```
# show docstring
network.pf();
```
```
INFO:pypsa.pf:Slack bus for sub-network 0 is My bus 0
INFO:pypsa.pf:Performing non-linear load-flow on AC sub-network SubNetwork 0 for snapshots Index(['now
'], dtype='object')
INFO:pypsa.pf:Newton-Raphson solved in 3 iterations with error of 0.000000 in 0.037454 seconds
```

Linear power flow

> [https://pypsa.readthedocs.io/en/latest/power_flow.html#linear-power-flow (https://pypsa.readthedocs.io/en/latest/power_flow.html#linear-power-flow)](https://pypsa.readthedocs.io/en/latest/power_flow.html#linear-power-flow)

For AC networks, it is assumed for the linear power flow that

- reactive power decouples
- there are no voltage magnitude variations
- voltage angles differences across branches are small
- branch resistances are much smaller than branch reactances.

In [39]:
```
# show docstring
network.lpf();
```
```
INFO:pypsa.pf:Slack bus for sub-network 0 is My bus 0
INFO:pypsa.pf:Performing linear load-flow on AC sub-network SubNetwork 0 for snapshot(s) Index(['now'],
dtype='object')
```

Let's look at some results

In [40]:
```
network.lines_t.p0
```
Out[40]:

	My line 0	My line 1	My line 2
now	33.333333	33.333333	-66.666667

In [41]:
```
network.buses_t.v_ang*180/np.pi
```

Out[41]:

	My bus 0	My bus 1	My bus 2
now	0.0	-0.190986	-0.381972

Optimal Power Flow

(solver required)

Model Formulation with Pyomo

To enable portability between solvers (e.g. free `Cbc` / `glpk` , or commercial `Gurobi` / `CPLEX`), the OPF is formulated using the Python optimisation modelling package [pyomo (www.pyomo.org)](www.pyomo.org) (which can be thought of as a Python version of [GAMS (www.gams.com)](www.gams.com)).

Let's make a few modifications to the power flow example...

In [42]:
```
network.generators.at['My gen 0', 'p_nom'] = 100
```

In [43]:
```
network.generators.at['My gen 0', 'marginal_cost'] = 50
```

In [44]:
```
network.lines.s_nom = 55
```

In [45]:
```
network.add("Generator","My candidate gen 1",
            bus="My bus 1",
            p_nom=0,
            p_nom_extendable=True,
            marginal_cost=25)
```

In [46]:
```
#network.remove('Generator','My candidate gen 1')
```

Linear Optimal Power Flow

Let's move to the documentation for more details on the optimisation problem formulation:

> [https://pypsa.readthedocs.io/en/latest/optimal_power_flow.html#linear-optimal-power-flow (https://pypsa.readthedocs.io/en/latest/optimal_power_flow.html#linear-optimal-power-flow)](https://pypsa.readthedocs.io/en/latest/optimal_power_flow.html#linear-optimal-power-flow)

In [47]:
```
#show docstring
network.lopf();
#network.lopf(solver_name='cbc');
#network.lopf(formulation='kirchhoff')
```

```
INFO:pypsa.pf:Slack bus for sub-network 0 is My bus 0
INFO:pypsa.opf:Performed preliminary steps
INFO:pypsa.opf:Building pyomo model using `angles` formulation
INFO:pypsa.opf:Solving model using glpk
INFO:pypsa.opf:Optimization successful

# ==========================================================
# = Solver Results                                         =
# ==========================================================
# ----------------------------------------------------------
#   Problem Information
# ----------------------------------------------------------
Problem:
- Name: unknown
  Lower bound: 3375.0
  Upper bound: 3375.0
  Number of objectives: 1
  Number of constraints: 16
  Number of variables: 10
  Number of nonzeros: 28
  Sense: minimize
# ----------------------------------------------------------
#   Solver Information
# ----------------------------------------------------------
Solver:
- Status: ok
  Termination condition: optimal
  Statistics:
    Branch and bound:
      Number of bounded subproblems: 0
      Number of created subproblems: 0
  Error rc: 0
  Time: 0.028525590896606445
# ----------------------------------------------------------
#   Solution Information
# ----------------------------------------------------------
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Let's have a glance at the results:

In [48]:
```
network.generators.p_nom
```

Out[48]:
```
My gen 0              100.0
My candidate gen 1      0.0
Name: p_nom, dtype: float64
```

In [49]:
```
network.generators.p_nom_opt
```

Out[49]:
```
My gen 0              100.0
My candidate gen 1     65.0
Name: p_nom_opt, dtype: float64
```

In [50]:
```
network.generators_t.p
```

Out[50]:

	My gen 0	My candidate gen 1
now	35.0	65.0

In [51]:
```
network.lines_t.p0
```

Out[51]:

	My line 0	My line 1	My line 2
now	-10.0	55.0	-45.0

In [52]:
```
network.lines_t.mu_upper
```

Out[52]:

	My line 0	My line 1	My line 2
now	-0.0	75.0	-0.0

In [53]:
```
network.buses_t.marginal_price
```

Out[53]:

	My bus 0	My bus 1	My bus 2
now	50.0	25.0	75.0

Data Import and Export

"Manually"

In [54]:
```
# show docstring
network.madd('Generator', ['g1', 'g2'], bus=['My bus 0']*2);
```

In [55]:
```
# show docstring
network.mremove('Generator', ['g1', 'g2'])
```

Importing from PYPOWER/MATPOWER Test Cases

In [56]:
```
from pypower.api import case30
ppc = case30()
network_case30 = pypsa.Network()
network_case30.import_from_pypower_ppc(ppc)
```

```
WARNING:pypsa.io:Warning: Note that when importing from PYPOWER, some PYPOWER features not supported: ar
eas, gencosts, component status
```

In [57]:
```
network_case30.generators.head();
```

In [58]:
```
network_case30.lines.head();
```

Saving and Reloading PyPSA Networks

In [59]:
```
# show docstring
n = pypsa.Network('networks/pypsa-eur-example.nc')
```

```
WARNING:pypsa.io:
Importing PyPSA from older version of PyPSA than current version 0.14.1.
Please read the release notes at https://pypsa.org/doc/release_notes.html
carefully to prepare your network for import.

INFO:pypsa.io:Imported network pypsa-eur-example.nc has buses, carriers, generators, global_constraints,
lines, links, loads, storage_units
```

In [60]:
```
n.export_to_csv_folder('networks/pypsa-eur-example')
```

```
WARNING:pypsa.io:Directory networks/pypsa-eur-example does not exist, creating it
INFO:pypsa.io:Exported network pypsa-eur-example has carriers, lines, generators, links, loads, storage_
units, buses, global_constraints
```

In [61]:
```
n_csv = pypsa.Network('networks/pypsa-eur-example')
```

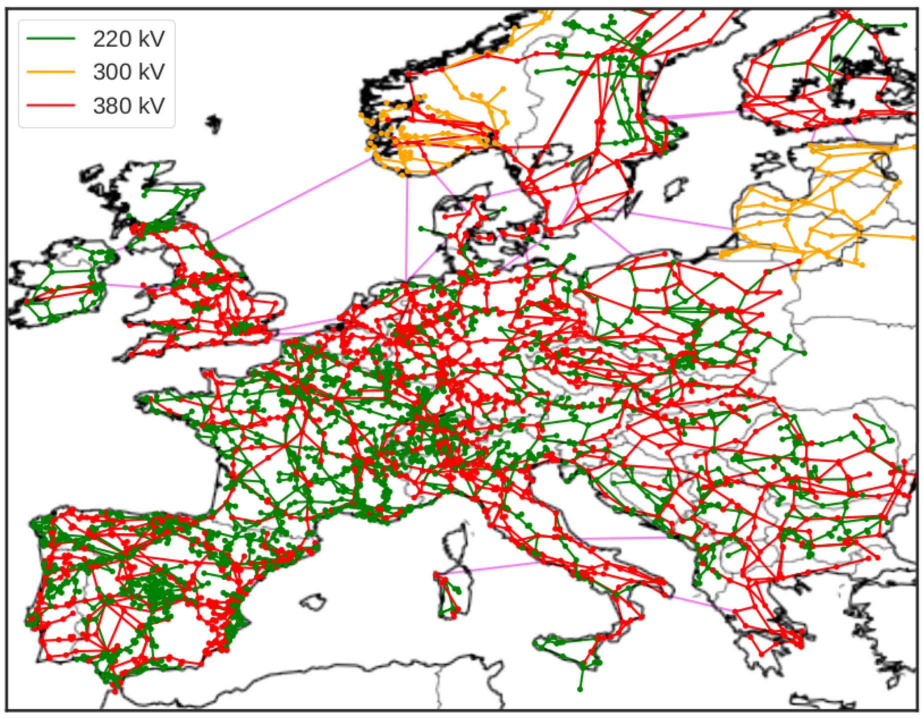
```
INFO:pypsa.io:Imported network pypsa-eur-example has buses, carriers, generators, global_constraints, li
nes, links, loads, storage_units
```

Overall: PyPSA Networks can be stored as `netCDF` , `CSV` or `HDF5` .

> [https://pypsa.readthedocs.io/en/latest/import_export.html (https://pypsa.readthedocs.io/en/latest/import_export.html)](https://pypsa.readthedocs.io/en/latest/import_export.html)

- `netCDF` files take up less space than `CSV` files and are faster to load.
- `netCDF` is also preferred over `HDF5` because `netCDF` is structured more cleanly, is easier to use from other programming languages, can limit float precision to save space and supports lazy loading.
- `CSV` might be easier to use with `Excel` .

Evaluating Larger Power Networks with a PyPSA-Eur Network

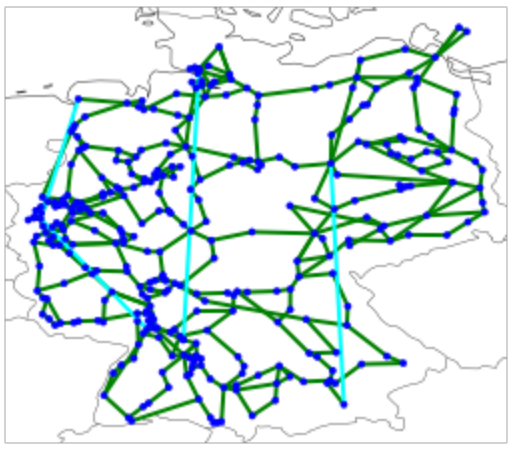


- Grid data based on **GridKit** extraction of ENTSO-E interactive map
- **powerplantmatching** tool combines open databases using matching algorithm DUKE
- Renewable energy time series from open **atlite**, based on Aarhus University REatlas
- Geographic potentials for RE from land use databases processed with **glaes**
- Optional: time series aggregation with **tsam**
- Basic **validation** performed in Hörsch et al 'PyPSA-Eur: An Open Optimisation Model of the European Transmission System'
- `https://github.com/PyPSA/pypsa-eur`

The following example (a German extract of `PyPSA-Eur` , I happen to have run recently.

All the outputs required nothing more than invoking `n.lopf()` with a commercial solver.

In [62]:
```
n.plot();
```



In [63]:
```
for c in n.iterate_components(list(n.components.keys())[2:]):
    print("Component '{}' has {} entries".format(c.name,len(c.df)))
```

```
Component 'Bus' has 333 entries
Component 'Carrier' has 12 entries
Component 'GlobalConstraint' has 1 entries
Component 'Line' has 466 entries
Component 'LineType' has 31 entries
Component 'TransformerType' has 14 entries
Component 'Link' has 4 entries
Component 'Load' has 256 entries
Component 'Generator' has 804 entries
Component 'StorageUnit' has 535 entries
```

In [64]:
```
n.snapshots[:5]
```

Out[64]:
```
DatetimeIndex(['2013-01-01 00:00:00', '2013-01-01 02:00:00',
               '2013-01-01 04:00:00', '2013-01-01 06:00:00',
               '2013-01-01 08:00:00'],
              dtype='datetime64[ns]', name='name', freq=None)
```

In [65]:
```
n.snapshots[-5:]
```

Out[65]:
```
DatetimeIndex(['2013-12-31 14:00:00', '2013-12-31 16:00:00',
               '2013-12-31 18:00:00', '2013-12-31 20:00:00',
               '2013-12-31 22:00:00'],
              dtype='datetime64[ns]', name='name', freq=None)
```

In [66]:
```
n.snapshot_weightings.head()
```

Out[66]:
```
name
2013-01-01 00:00:00    2.0
2013-01-01 02:00:00    2.0
2013-01-01 04:00:00    2.0
2013-01-01 06:00:00    2.0
2013-01-01 08:00:00    2.0
Name: weightings, dtype: float64
```

In [67]:
```
len(n.snapshots)*2 == 365*24
```

Out[67]:
```
True
```

In [68]:
```
n.generators.head();
```

In [69]:
```
n.lines.head();
```

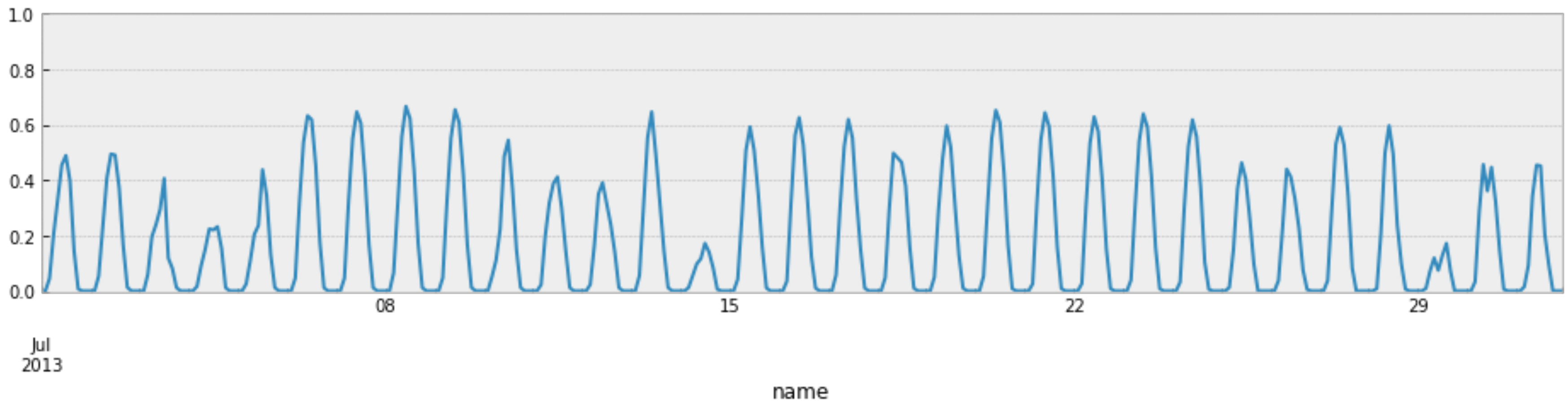
In [70]:
```
n.generators.head();
```

In [71]:
```
n.links.head();
```

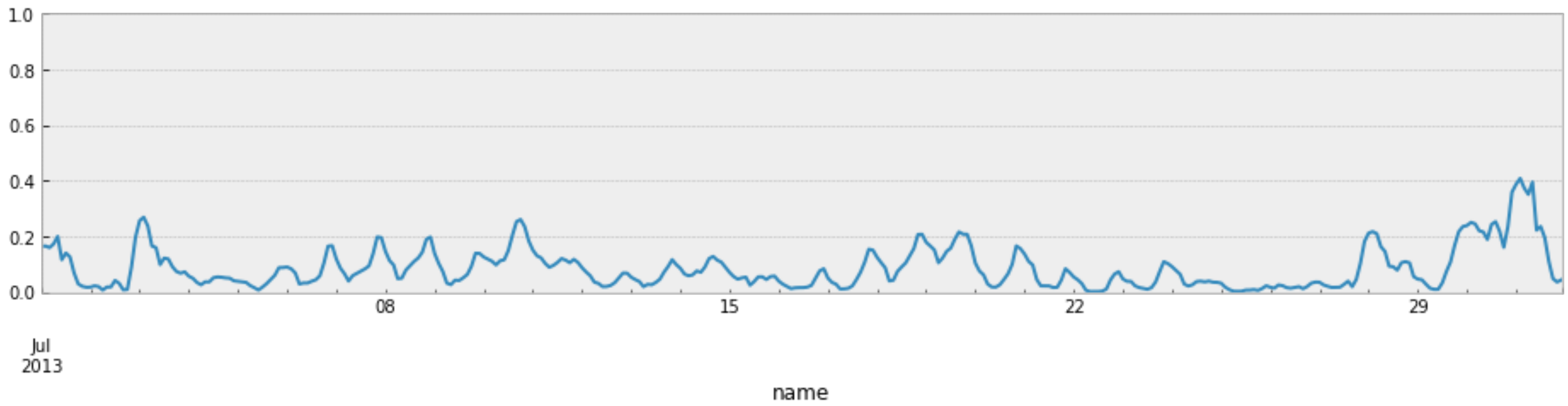
Renewables Availability Time Series:

In [72]:
```
time = '2013-7'
name = '8066 solar' # 3304 onwind, 8066 solar
selection = 'onwind'
```

In [73]:
```
n.generators_t.p_max_pu.loc[time,name].plot(figsize=(16,3), ylim=[0,1]);
```

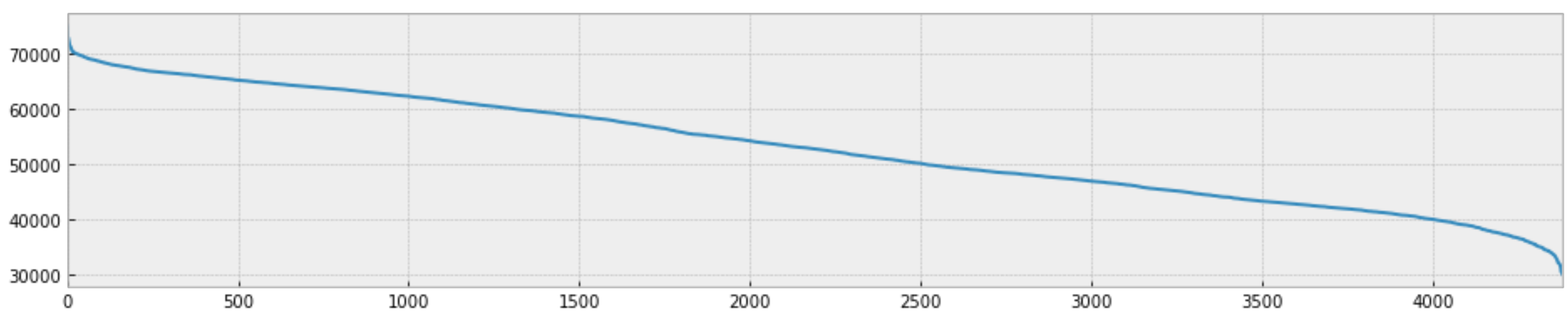


In [74]:
```
n.generators_t.p_max_pu.filter(like=selection).mean(axis=1)[time].plot(figsize=(16,3), ylim=[0,1]);
```



Duration Curves

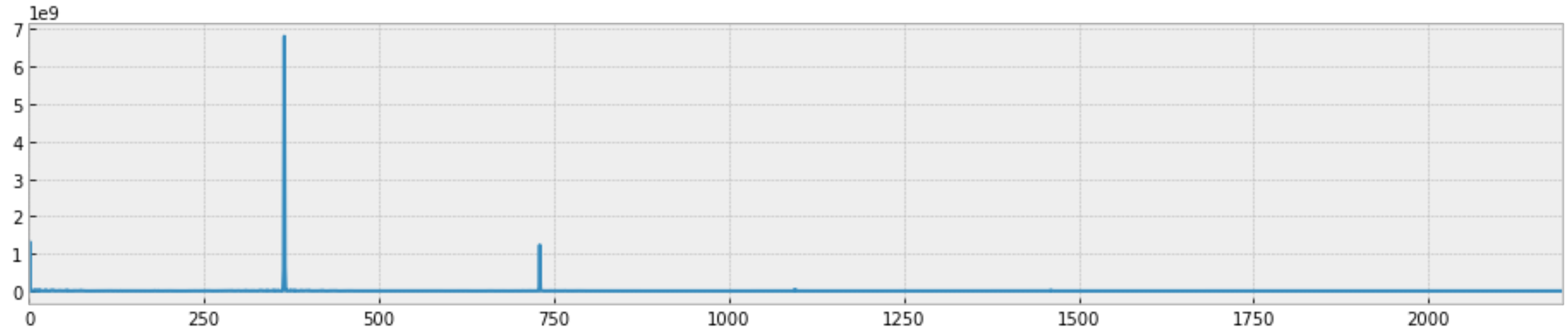
In [75]:
```
n.loads_t.p_set.sum(axis=1).sort_values(ascending=False).reset_index(drop=True).plot(figsize=(16,3));
```



Fourier Transform

In [76]:
```
def frequencies(d):
    return abs(pd.Series(np.fft.rfft(d - d.mean()), index=np.fft.rfftfreq(len(d), d=1./len(d)))**2)
```

In [77]:
```
data = n.generators_t.p_max_pu.filter(like='solar').sum(axis=1)
frequencies(data).plot(figsize=(16,3));
```



Total System Costs:

In [78]:
```
n.objective / 1e9 # billion Euros
```

Out[78]:
```
37.7411839004
```

CO_2 price - dual of carbon cap:

In [79]:
```
n.global_constraints
```

Out[79]:

name	sense	constant	mu	type	carrier_attribute
CO2Limit	<=	18000000.0	320.895923	primary_energy	co2_emissions

Transmission Line Expansion:

In [80]:
```
(n.lines.s_nom_opt - n.lines.s_nom).head(10)
```

Out[80]:
```
name
9636        0.023099
11025       0.057421
11460       0.008194
18          0.020455
697      1138.227279
681       197.912532
7492        0.006281
682         0.002081
556         0.026027
850       495.038598
dtype: float64
```

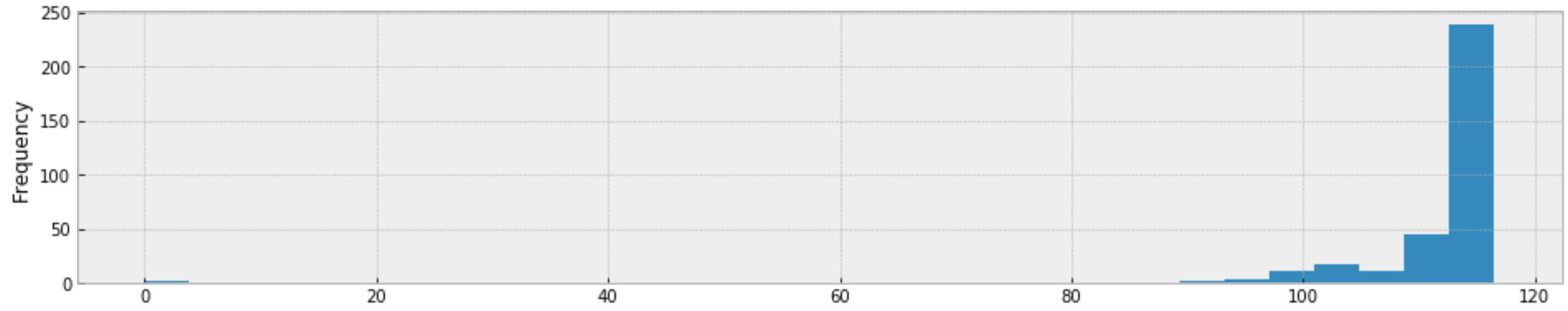
Optimal Generator/Storage Capacities

In [81]:
```
n.storage_units.groupby('carrier').p_nom_opt.sum()
```

Out[81]:
```
carrier
H2         26829.079068
PHS         6688.420000
battery     4647.495124
hydro        124.000000
Name: p_nom_opt, dtype: float64
```

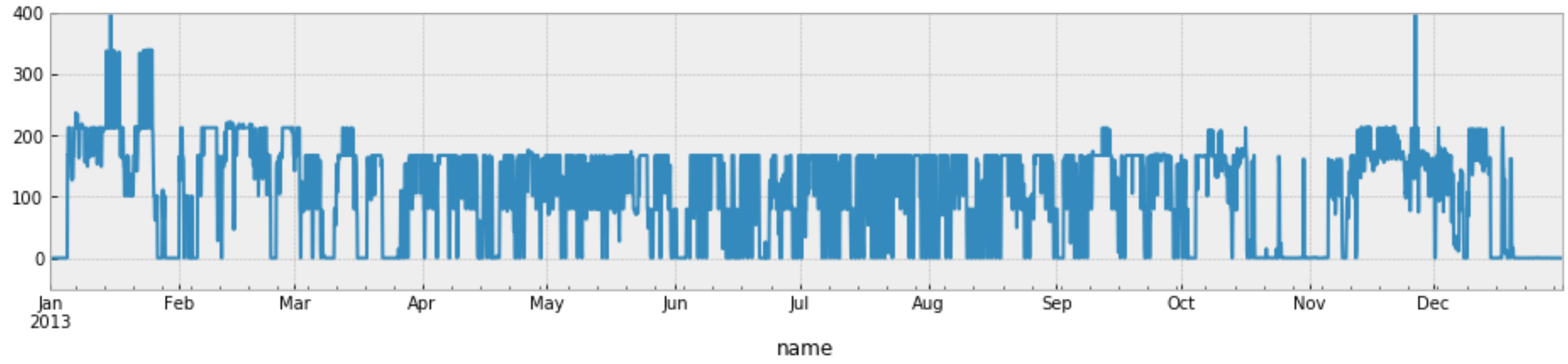
Distribution of Average Locational Marginal Prices:

In [82]:
```
n.buses_t.marginal_price.mean().plot.hist(bins=30, figsize=(16,3));
```



Locational Marginal Prices / Nodal Prices at a Single Location:

In [83]:
```
n.buses_t.marginal_price['3305'].plot(ylim=[-50,400], figsize=(16,3));
```

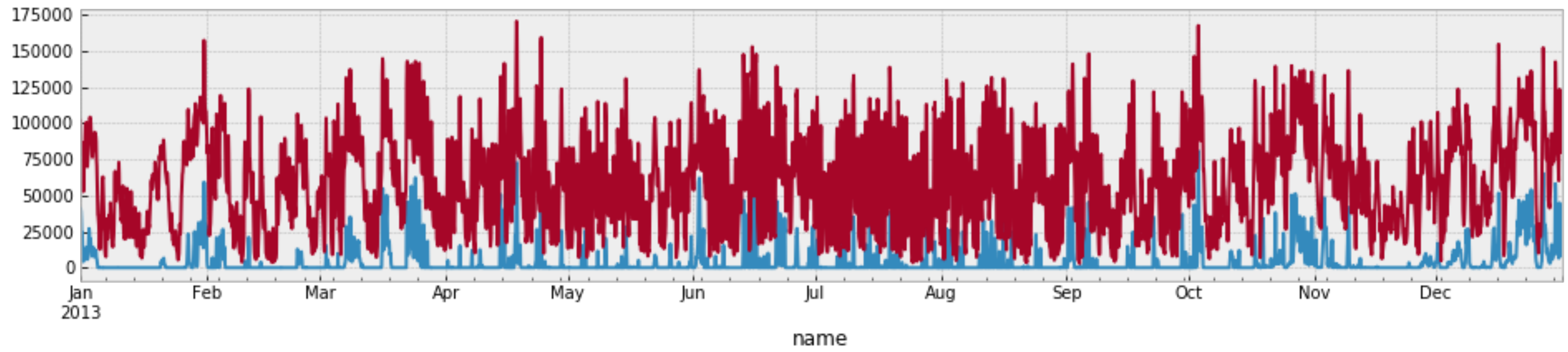


Curtailment of Renewable Generators:

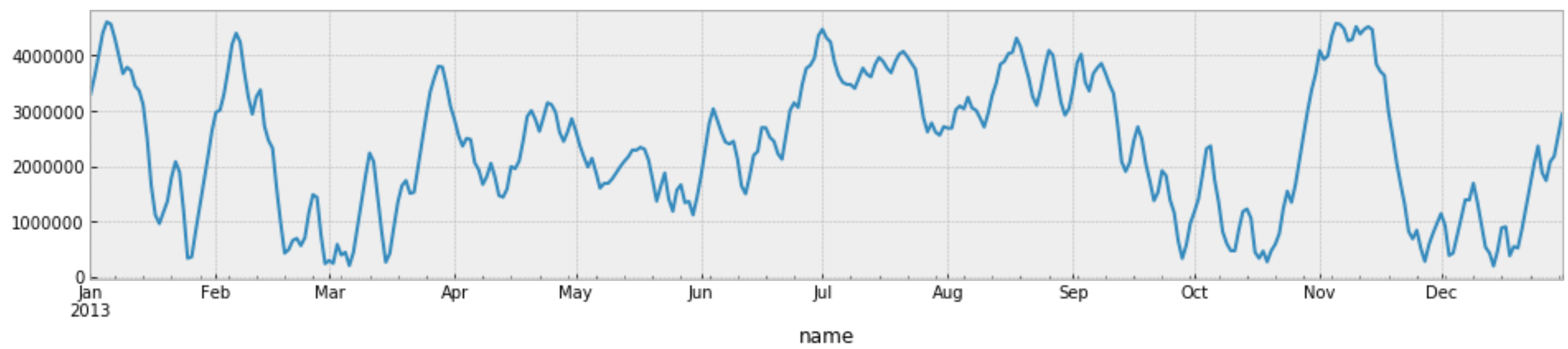
In [84]:
```
def curtailment(network):
    available= network.generators_t.p_max_pu\
        .multiply(
            network.generators.p_nom_opt.filter(regex='ror|solar|wind')
        )

    generated = network.generators_t.p.filter(regex='ror|solar|wind')

    return available - generated, available
```

In [85]:
```
curt, available = curtailment(n)
```

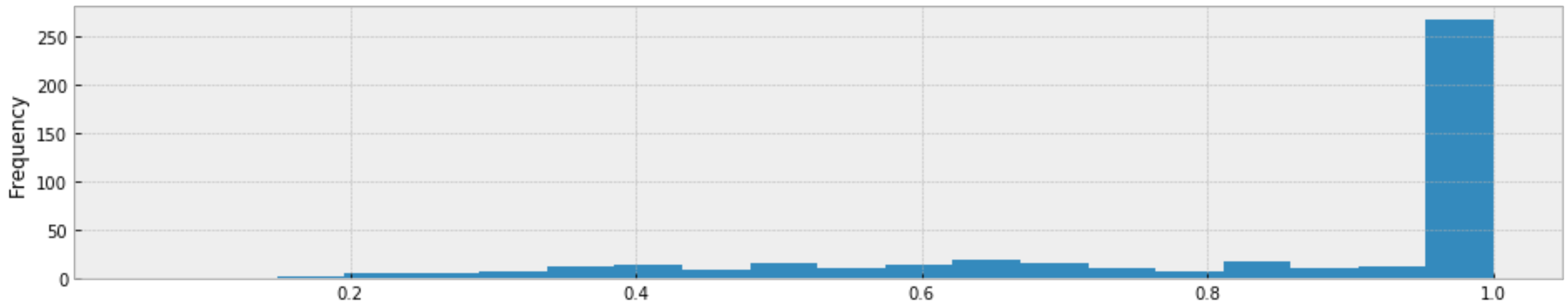
In [86]:
```
fig, ax = plt.subplots(figsize=(16,3))
curt.sum(axis=1).plot(ax=ax)
available.sum(axis=1).plot(ax=ax);
```



In [87]:
```
n.storage_units_t.state_of_charge.sum(axis=1).resample('D').mean().plot(figsize=(16,3));
```



In [88]:
```
line_loading = (n.lines_t.p0.abs().max() / n.lines.s_nom_opt / n.lines.s_max_pu)
line_loading.plot.hist(bins=20,figsize=(16,3));
```



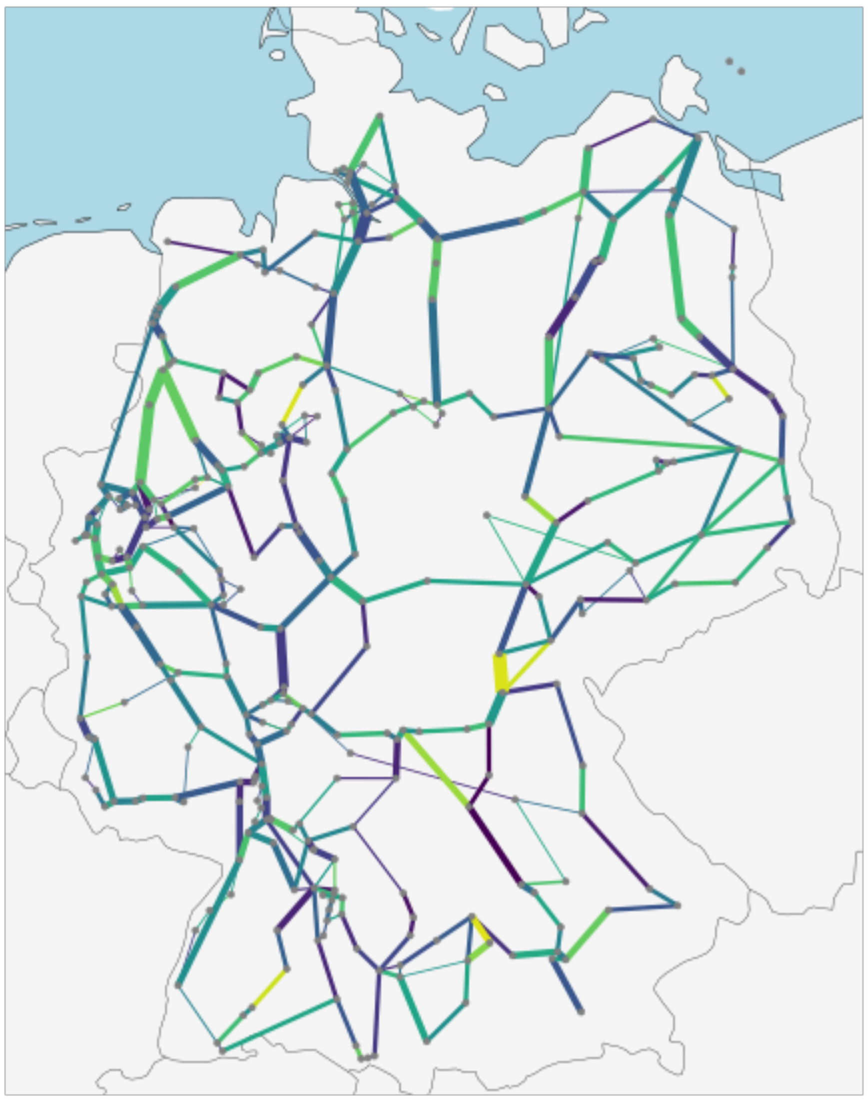
Plotting Networks

In [89]:
```
import cartopy.crs as ccrs
```

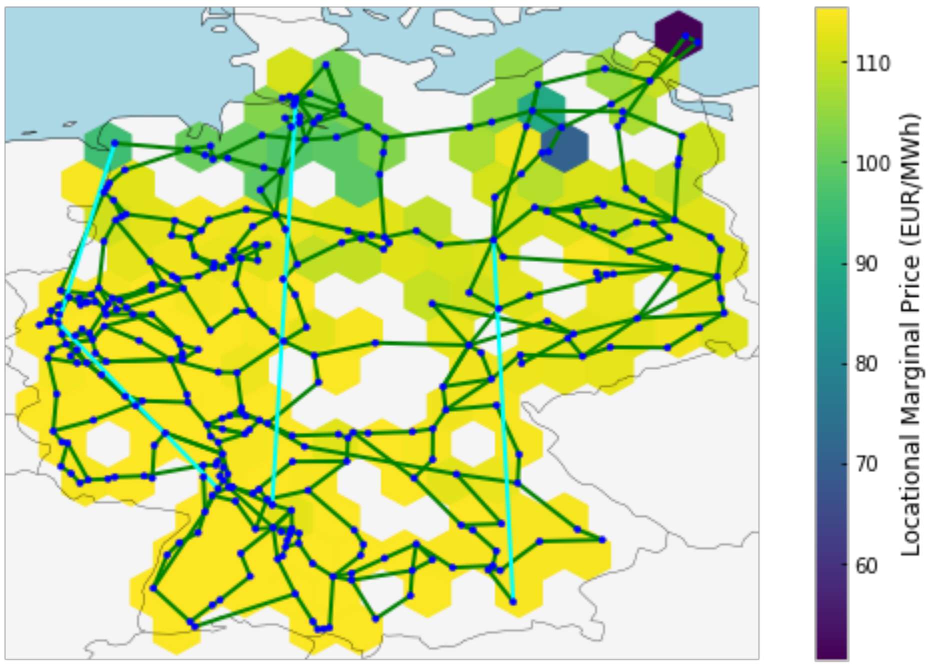
In [90]:
```
loading = (n.lines_t.p0.abs().mean().sort_index() / (n.lines.s_nom_opt*n.lines.s_max_pu).sort_index()).
fillna(0.)
```

In [91]:
```
fig,ax = plt.subplots(
            figsize=(10,10),
            subplot_kw={"projection": ccrs.epsg(4839)}
        )

n.plot(ax=ax,
       bus_colors='gray',
       branch_components=["Line"],
       line_widths=n.lines.s_nom_opt/1.5e3,
       line_colors=loading,
       line_cmap=plt.cm.viridis,
       color_geomap=True)

ax.axis('off')
```

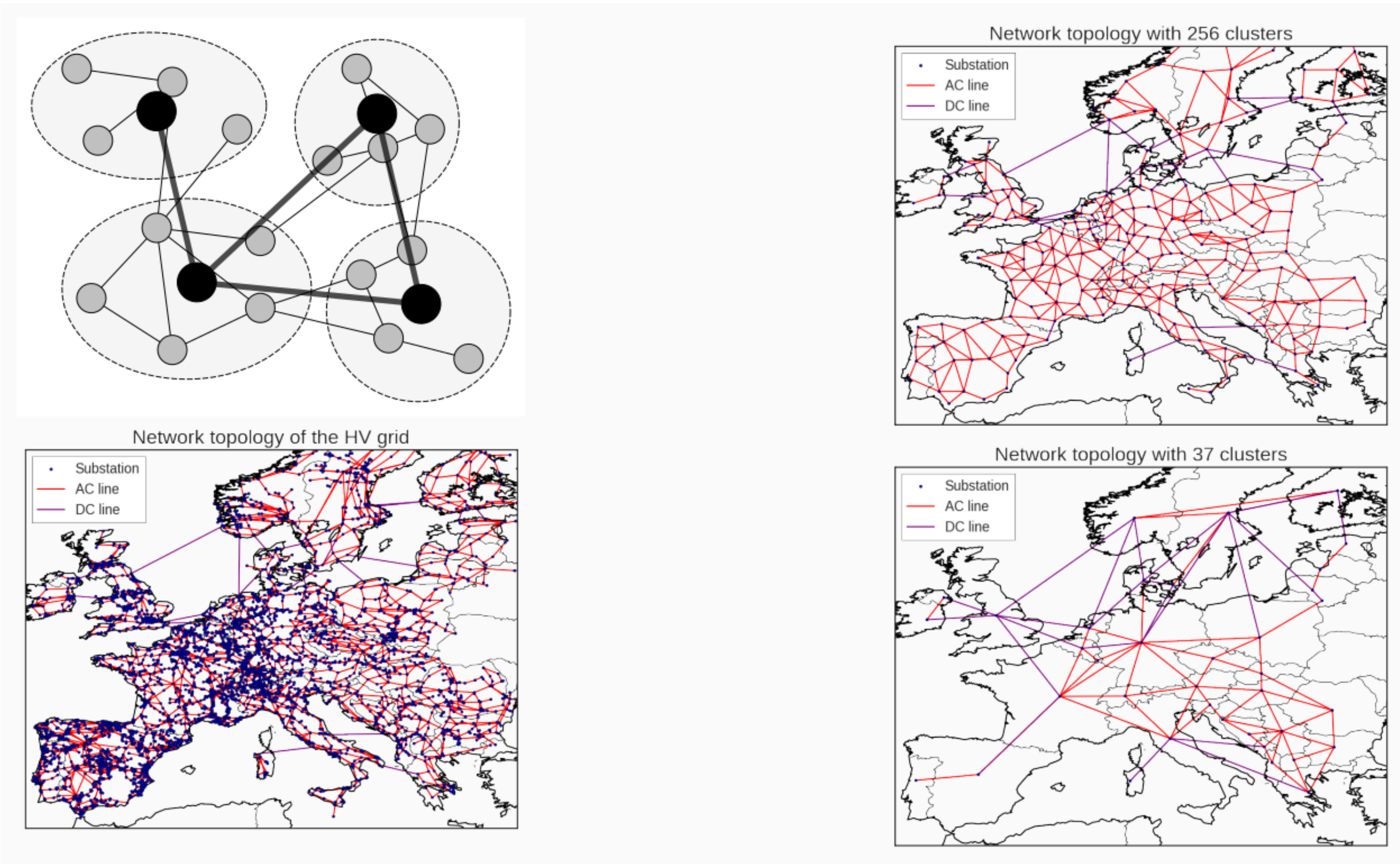
Out[91]: (-343718.144836487, 332070.74740666285, -404143.0985175424, 453552.8288801354)



In [92]:
```
fig,ax = plt.subplots(
            figsize=(10,6),
            subplot_kw={"projection": ccrs.PlateCarree()}
         )

n.plot(ax=ax, color_geomap=True)

marginal_prices = n.buses_t.marginal_price.mean(axis=0)

hb = ax.hexbin(
        n.buses.x,
        n.buses.y,
        gridsize=15,
        C=marginal_prices,
        cmap=plt.cm.viridis
     )

cb = fig.colorbar(hb, ax=ax)
cb.set_label('Locational Marginal Price (EUR/MWh)')
```



Network Clustering



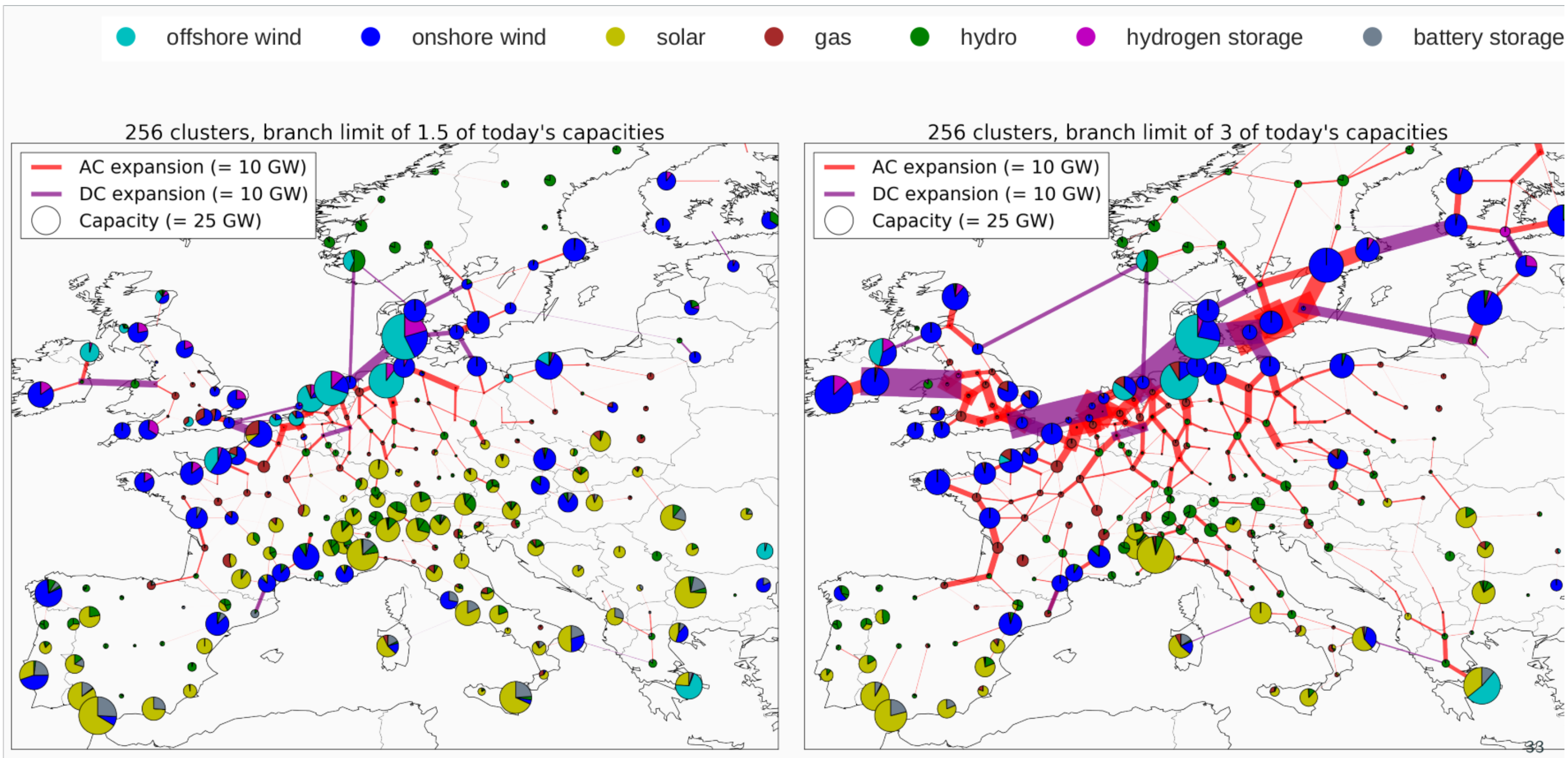
Comparison

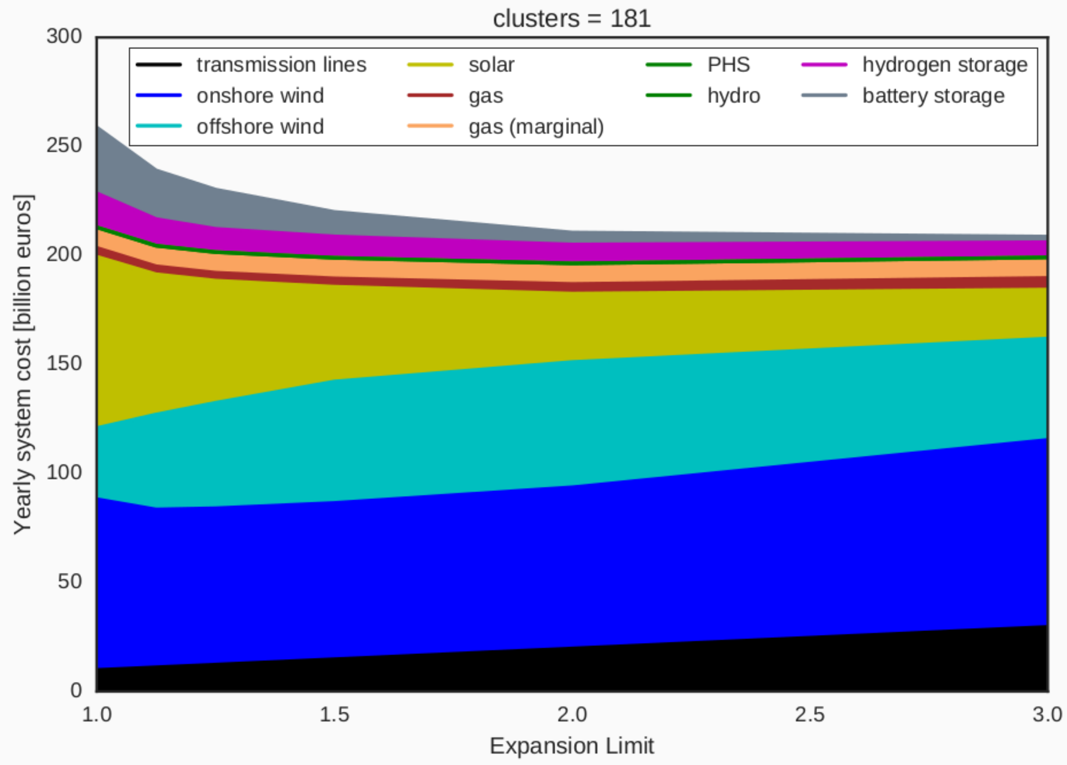
(Source: Dr. Tom Brown)

					Grid Analysis			Economic Analysis							
	Software	Version	Citation	Free Software	Power Flow	Continuation Power Flow	Dynamic Analysis	Transport Model	Linear OPF	SCLOPF	Nonlinear OPF	Multi-Period Optimisation	Unit Commitment	Investment Optimisation	Other Energy Sectors
Power system tools	MATPOWER	6.0	[?]	✓	✓	✓		✓	✓		✓				
	NEPLAN	5.5.8	[?]		✓		✓	✓	✓	✓	✓				✓
	pandapower	1.4.0	[?]	✓	✓			✓	✓		✓				
	PowerFactory	2017	[?]		✓		✓		✓	✓	✓				
	PowerWorld	19	[?]		✓		✓	✓	✓	✓	✓				
	PSAT	2.1.10	[?]	✓	✓	✓	✓		✓		✓	✓	✓		
	PSS/E	33.10	[?]		✓		✓		✓	✓	✓				
	PSS/SINCAL	13.5	[?]		✓		✓				✓				✓
	PYPOWER	5.1.2	[?]	✓	✓			✓	✓		✓				
	PyPSA	0.9.0		✓	✓			✓	✓	✓		✓	✓	✓	✓
Energy system tools	calliope	0.5.2	[?]	✓				✓				✓		✓	✓
	minpower	4.3.10	[?]	✓				✓	✓			✓	✓		
	MOST	6.0	[?]	✓	✓	✓		✓	✓	✓	✓	✓	✓		
	oemof	0.1.4	[?]	✓				✓				✓	✓	✓	✓
	OSeMOSYS	2017	[?]	✓				✓				✓		✓	✓
	PLEXOS	7.400	[?]					✓	✓	✓		✓	✓	✓	✓
	PowerGAMA	1.1	[?]	✓				✓	✓			✓			
	PRIMES	2017	[?]					✓	✓			✓	✓	✓	✓
	TIMES	2017	[?]					✓	✓			✓	✓	✓	✓
	urbs	0.7	[?]	✓				✓				✓	✓	✓	✓

Further Exemplary Results

(Source: Dr. Tom Brown)





- Big **non-linear cost reduction** as grid is expanded, from 82€/MWh to 66€/MWh (drop of 50 bill. €/a)
- Most of cost reduction happens with **25% grid expansion** compared to today's grid; costs rather flat once capacity has doubled
- Need for solar and batteries decrease significantly as grid expanded; with cost-optimal grid, system is dominated by wind

Source: Schlachtberger et al, 2017, Hörsch et al, 2017

Conclusion

- Who else uses PyPSA?
- How to ask questions?
- How to report bugs?
- How to contribute?

How to find this Jupyter Notebook and Data

> www.neumann.fyi/assets/pypsa-tutorial.html
>
> Dropbox link for the rest

Copyright

In [ ]: