Annexes

Calendrier des activités

26 février 2024 : conférence de l'Agence Internationale de l'Énergie.

21 mars 2024 : contre-conférence sur le nucléaire organisée par 5 ONG.

29 avril 2024 : Steering Comittee pour le projet VEKA auquel j'ai assisté en ligne.

Le reste de l'organisation de mon stage a déjà été expliquée dans la Présentation de l'institution d'accueil.

Calendrier des contacts avec le/la maître de stage UCL

1er mars 2024 : 1er point de contact avec Prof. Contino.

26 mars 2024 : 2ème point de contact avec Prof. Contino.

24 avril 2024 : 3ème point de contact avec Prof. Contino.

Adresses des différents contacts établis

Je n'ai pas établi de contacts durant mon stage. Pour le projet VEKA, je n'ai pas eu d'interaction en tant que telle avec le client. Pour le projet de stockage par air comprimé, je n'ai pas moi-même discuté avec le client et je ne peux pas mentionner son nom pour raison de confidentialité.

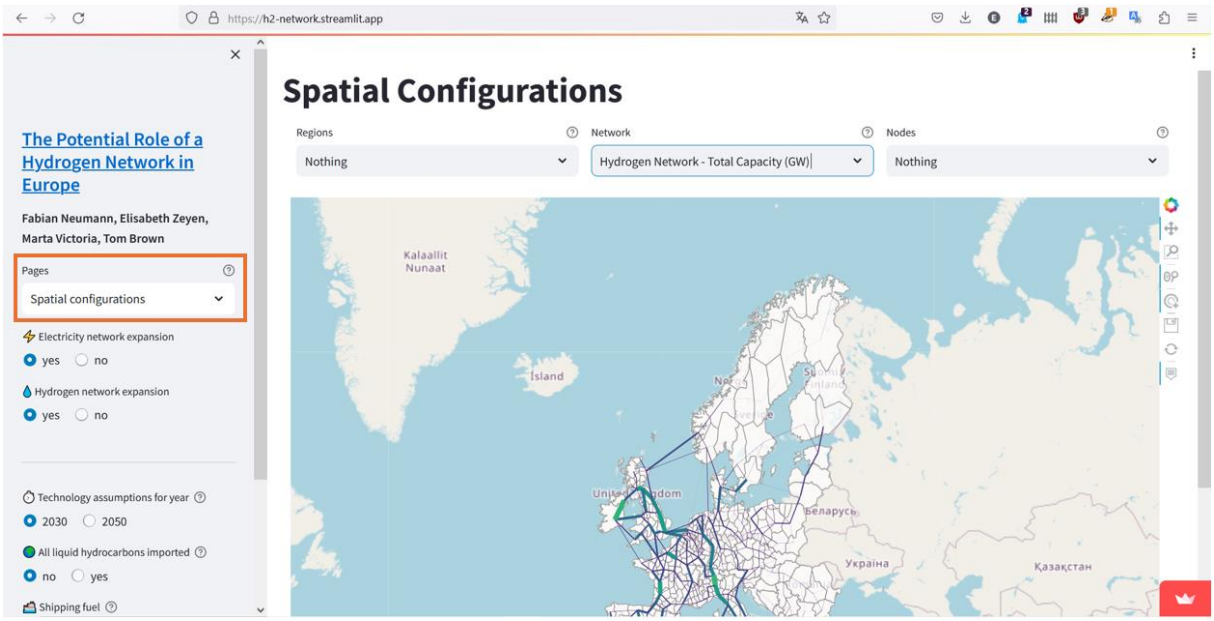
Streamlit interactive maps documentation

1. Intro

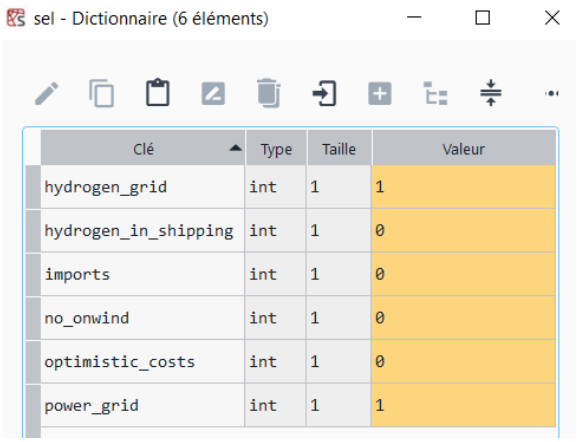
An example of interactive maps for PyPSA built with Streamlit can be found here, the GitHub here and the main file is “streamlit_app.py”.

Unfortunately, I was not able to create a stable conda environment that contains all the necessary packages for this GitHub **and** PyPSA (so PyPSA is missing).

In this section, I will only focus on the "Pages = Spatial configuration" option.



All the options on the side (defined in `with st.sidebar`) correspond to different PyPSA runs, therefore to different configurations of the European energy network. Depending on the cells checked in the options, this will modify the `sel` dictionary (also defined in `with st.sidebar`).



Clé	Type	Taille	Valeur
hydrogen_grid	int	1	1
hydrogen_in_shipping	int	1	0
imports	int	1	0
no_onwind	int	1	0
optimistic_costs	int	1	0
power_grid	int	1	1

This dictionary is passed as a parameter to functions `make_hydrogen_graph(**kwargs)` and `make_electricity_graph(**kwargs)`. These 2 functions have very similar operation.

2. Xarray

We start by reading related data by using the xarray package.

```python
def make_hydrogen_graph(**kwargs):

    ds = xr.open_dataset("data/hydrogen-network.nc")
```

Which gives the variable *ds*.

```
IPdb [2]: ds
Out  [2]:
<xarray.Dataset> Size: 2MB
Dimensions:                (power_grid: 2, hydrogen_grid: 2,
                            optimistic_costs: 2, imports: 2,
                            hydrogen_in_shipping: 2, no_onwind: 2, line: 372)
Coordinates:
  * power_grid             (power_grid) int64 16B 0 1
  * hydrogen_grid          (hydrogen_grid) int64 16B 0 1
  * optimistic_costs       (optimistic_costs) int64 16B 0 1
  * imports                (imports) int64 16B 0 1
  * hydrogen_in_shipping   (hydrogen_in_shipping) int64 16B 0 1
  * no_onwind              (no_onwind) int64 16B 0 1
  * line                   (line) <U7 10kB '1' '2' '3' ... '14804+1' '14822+1'
Data variables:
    bus0                   (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) <U6 571kB ...
    bus1                   (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) <U6 571kB ...
    length                 (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
    s_nom                  (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
    s_nom_max              (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
    carrier                (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) <U2 190kB ...
    s_nom_opt              (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
```

A good explanation of xarray is given here but a quick explanation of xarray can be useful. Let us start with *xarray.DataArray*. Here is an example: the temperature (K) at 5 timesteps for 12 (3 latitudes and 4 longitudes) locations can be stored in an *DataArray*.

xarray.DataArray (**time**: 5, **lat**: 3, **lon**: 4)

```
array([[[293.54540344, 284.21456978, 290.56946559, 291.14002034],
        [280.27792471, 288.80459871, 279.59725505, 282.94722765],
        [281.240509  , 276.5719258 , 282.69555401, 283.83783917]],

       [[280.41284877, 282.57514193, 286.22459609, 274.26439822],
        [282.61282235, 284.36482901, 286.17258599, 282.61130087],
        [292.43916457, 288.90546266, 279.66094407, 279.52174471]],

       [[282.9685738 , 278.51659892, 278.41202118, 282.94755646],
        [283.40763326, 296.67529767, 289.77672519, 272.23552105],
        [289.81845633, 287.45710119, 282.60207573, 279.63464563]],

       [[284.38842476, 277.41341624, 277.95770866, 281.06974285],
        [287.78403522, 284.61860784, 287.38836747, 286.00244169],
        [285.39251273, 282.65682517, 282.5116848 , 285.51485671]],

       [[283.29857529, 281.38079867, 276.72779204, 283.24694677],
        [286.52766919, 278.32713261, 287.11438781, 283.15556069],
        [277.80650789, 280.19002728, 275.69232531, 292.34379177]]])
```

▾ Coordinates:

time	(time)	datetime64[ns]	2018-01-01 ... 2018-01-05
lat	(lat)	float64	25.0 40.0 55.0
lon	(lon)	float64	-120.0 -100.0 -80.0 -60.0

▾ Indexes:

time	PandasIndex
lat	PandasIndex
lon	PandasIndex

▸ Attributes: (0)

Imagine that for the same coordinates (time, lat, lon), we want to record the pressure (hPa).

xarray.DataArray (**time**: 5, **lat**: 3, **lon**: 4)

```
array([[[1003.55086445, 1000.28433974, 1005.47413663,  995.77347468],
        [ 998.21204837, 1001.04170237, 1002.79185434, 1002.21831449],
        [1002.01780481,  996.07231203,  994.4818565 ,  999.17029591]],

       [[1005.53451048, 1002.67305177,  992.74558233,  993.3526019 ],
        [ 996.41126883, 1000.0211118 , 1000.48376802,  996.00524732],
        [1003.10846469, 1000.02418755,  995.84385628,  999.38946301]],

       [[ 998.06186149,  989.26563118, 1003.39658411,  999.92443832],
        [ 998.61595409, 1004.16173736, 1003.96838445,  994.9209871 ],
        [1002.22484811,  998.87277465,  996.93477925, 1002.73216326]],

       [[1001.45839468,  993.22181335,  991.88552604,  999.5784647 ],
        [1009.87415301,  995.47019748, 1003.60497356,  996.35861308],
        [ 998.4835137 ,  999.49214612, 1005.51246653,  999.78617478]],

       [[ 999.31787318, 1007.99240475, 1008.28582582, 1001.41458571],
        [1001.6287121 ,  999.38672149,  993.48877449,  998.14262489],
        [1003.44226691, 1005.3513651 , 1000.72891534, 1002.73612725]]])
```

▾ Coordinates:

time	(time)	datetime64[ns]	2018-01-01 ... 2018-01-05
lat	(lat)	float64	25.0 40.0 55.0
lon	(lon)	float64	-120.0 -100.0 -80.0 -60.0

▾ Indexes:

time	PandasIndex
lat	PandasIndex
lon	PandasIndex

▾ Attributes:

As those *DataArrays* share coordinates, we can store them in a *Dataset*. *Datasets* are containers similar to Python dictionaries; each *Dataset* can hold one or more *DataArrays*.

```
ds = xr.Dataset(data_vars={'Temperature': temp, 'Pressure': pressure})
ds
```

xarray.Dataset

▸ Dimensions: (**time**: 5, **lat**: 3, **lon**: 4)

▾ Coordinates:

time	(time)	datetime64[ns]	2018-01-01 ... 2018-01-05
lat	(lat)	float64	25.0 40.0 55.0
lon	(lon)	float64	-120.0 -100.0 -80.0 -60.0

▾ Data variables:

Temperature	(time, lat, lon)	float64	293.5 284.2 290.6 ... 275.7 292.3
Pressure	(time, lat, lon)	float64	1.004e+03 1e+03 ... 1.003e+03

▸ Indexes: (3)

▸ Attributes: (0)

In addition, you can access DataArrays through a dictionary syntax.

```
ds['Pressure']
```

xarray.DataArray 'Pressure' (**time**: 5, **lat**: 3, **lon**: 4)

```
array([[[1003.55086445, 1000.28433974, 1005.47413663,  995.77347468],
        [ 998.21204837, 1001.04170237, 1002.79185434, 1002.21831449],
        [1002.01780481,  996.07231203,  994.4818565 ,  999.17029591]],

       [[1005.53451048, 1002.67305177,  992.74558233,  993.3526019 ],
        [ 996.41126883, 1000.0211118 , 1000.48376802,  996.00524732],
        [1003.10846469, 1000.02418755,  995.84385628,  999.38946301]],

       [[ 998.06186149,  989.26563118, 1003.39658411,  999.92443832],
        [ 998.61595409, 1004.16173736, 1003.96838445,  994.9209871 ],
        [1002.22484811,  998.87277465,  996.93477925, 1002.73216326]]
```

Let's take back the *ds* variable, representing the electricity network. Each **dimension** (but lines) has 2 possible **coordinates**: {0,1}.

```
IPdb [2]: ds
Out  [2]:
<xarray.Dataset> Size: 2MB
Dimensions:                (power_grid: 2, hydrogen_grid: 2,
                            optimistic_costs: 2, imports: 2,
                            hydrogen_in_shipping: 2, no_onwind: 2, line: 372)
Coordinates:
  * power_grid             (power_grid) int64 16B 0 1
  * hydrogen_grid          (hydrogen_grid) int64 16B 0 1
  * optimistic_costs       (optimistic_costs) int64 16B 0 1
  * imports                (imports) int64 16B 0 1
  * hydrogen_in_shipping   (hydrogen_in_shipping) int64 16B 0 1
  * no_onwind              (no_onwind) int64 16B 0 1
  * line                   (line) <U7 10kB '1' '2' '3' ... '14804+1' '14822+1'
Data variables:
    bus0                   (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) <U6 571kB ...
    bus1                   (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) <U6 571kB ...
    length                 (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
    s_nom                  (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
    s_nom_max              (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
    carrier                (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) <U2 190kB ...
    s_nom_opt              (power_grid, hydrogen_grid, optimistic_costs, imports, hydrogen_in_shipping, no_onwind, line) float64 190kB ...
```

Dimensions of our space.
In a classic orthonormal coordinate system, we would have (x,y,z).

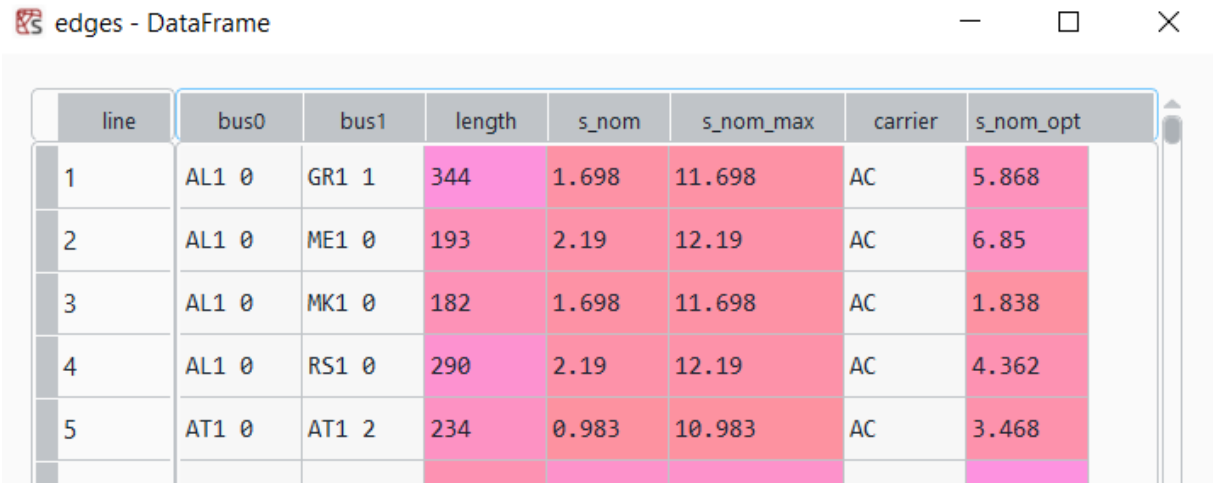
Those dimensions are the same as the keys stored in the dictionary *sel*. Which means that we can select the network options we want (i.e., the coordinates of the dimensions of the *DataSet*) thanks to the method *sel* (not the *sel* dictionary (which is here given as the *kwargs* argument)).

```
def make_hydrogen_graph(**kwargs):

    ds = xr.open_dataset("data/hydrogen-network.nc")
    edges = ds.sel(**kwargs, drop=True).to_pandas()
```

All the dimensions (but lines) will be fixed to 1 coordinate and the **lines coordinates** correspond to the lines of our electrical network.

After converting to a *DataFrame*, we find a familiar structure.



line	bus0	bus1	length	s_nom	s_nom_max	carrier	s_nom_opt
1	AL1 0	GR1 1	344	1.698	11.698	AC	5.868
2	AL1 0	ME1 0	193	2.19	12.19	AC	6.85
3	AL1 0	MK1 0	182	1.698	11.698	AC	1.838
4	AL1 0	RS1 0	290	2.19	12.19	AC	4.362
5	AT1 0	AT1 2	234	0.983	10.983	AC	3.468

Personally, I was unable to extract the relevant information by directly passing the file: *elec_s181_37m_lv3.0__3H-I-T-H-B-A-CCL_2030.nc* instead of the *hydrogen-network.nc* file.

Instead, I created (in another file and with another conda env containing PyPSA) a csv file containing the desired information. Then I read this csv file into the main python script.

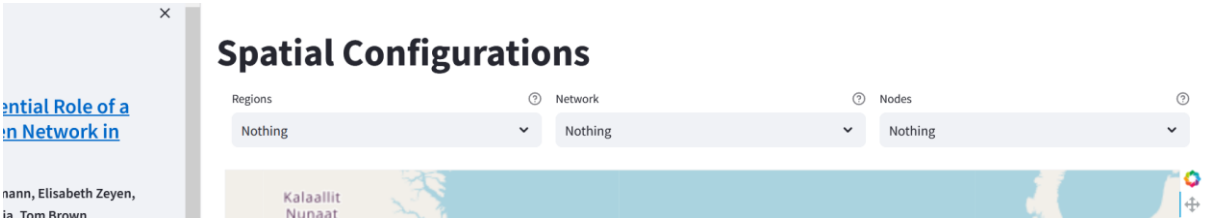
```
edges = pd.read_csv("data_VEKA/hydrogen-network.csv")
```

For the moment, I am not using the xarray package, but it could be useful to contain the 2030, 2040, 2050 data.

3. Columns

a. Network

Once we have selected "*Page*" as "*Spatial configuration*" and all the options in the side bar, we still can choose 3 options of display : Regions, Network and Nodes.



In the code, those options are defined as follows:

```
    _, col1, col2, col3, _ = st.columns([1,30,30,30,1])

    with col1:
        r_sel = st.selectbox(
            "Regions", options,
            help='Choose which data should be shown in choropleth.',
            format_func=parse_spatial_options
        )

    with col2:
        n_sel = st.selectbox(
            "Network", network_options,
            help='Choose which network data should be displayed.',
            format_func=parse_spatial_options
        )

    with col3:
        b_sel = st.selectbox(
            "Nodes", options,
            help='Choose which data should be shown on nodes.',
            format_func=parse_spatial_options
        )
```

r_sel will store our display choice for the regions, *n_sel* for the network and *b_sel* for the nodes. For debugging purposes, we can thus fix them. Here are some possibilities.

```
r_sel = ('Capacity (GW)', 'Onshore Wind')
n_sel = ('Hydrogen Network', 'Total Capacity (GW)')
b_sel = ('Capacity (GW)', 'Onshore Wind')
b_sel = ('Nothing',)
```

The complete list of possible values (and therefore the values of the drop-down menus) can be found in the *data/config.yaml* file.

If you want to display the hydrogen network, the *if* condition in the following code must be true.

```
    if n_sel[0] == "Hydrogen Network" and sel["hydrogen_grid"]:

        H = make_hydrogen_graph(**sel)

        pos = load_positions()

        scale = pd.Series(nx.get_edge_attributes(H, n_sel[1])).max() / 10

        network_plot = hvnx.draw(
            H,
            pos=pos,
            responsive=True,
            geo=True,
            node_size=5,
            node_color='k',
            edge_color=n_sel[1],
            inspection_policy="edges",
            edge_width=hv.dim(n_sel[1]) / scale,
        ).opts(**opts)

        plot *= network_plot
```

b. Regions

As you can see, the *plot* variable is not created at this point in the code but simply multiplied (*=). This is because it is defined higher in the code.

```
    gdf = load_regions()
```

This function reads a GeoJSON file representing the geographical division of Europe for simulation with PyPSA.

```
IPdb [3]: gdf
Out  [3]:
                                                geometry   name
name
AL1 0  POLYGON ((20.32207 39.91318, 20.39703 39.81809...  AL1 0
AT1 0  POLYGON ((10.08288 47.35907, 10.20928 47.37248...  AT1 0
AT1 1  POLYGON ((16.43396 47.39685, 16.42435 47.34520...  AT1 1
AT1 2  POLYGON ((13.58439 47.44950, 13.89111 47.12503...  AT1 2
AT1 3  POLYGON ((15.06195 46.64956, 15.00439 46.63684...  AT1 3
...                                                  ...    ...
SE2 5  MULTIPOLYGON (((13.58082 55.38882, 13.50343 55...  SE2 5
SE2 6  MULTIPOLYGON (((15.40763 62.87254, 14.99205 64...  SE2 6
SE2 7  MULTIPOLYGON (((22.40528 65.54194, 22.27491 65...  SE2 7
SI1 0  POLYGON ((15.60435 46.16700, 15.58988 46.11352...  SI1 0
SK1 0  POLYGON ((17.61911 47.82923, 17.57260 47.82954...  SK1 0
```

Then, the data that we want to display (*r_sel*) by territory is added as a column to this DataFrame *gdf* (and later, a color is associated to it).

```
        col = " - ".join(r_sel)
        gdf[col] = df[r_sel].reindex(gdf.index)
        gdf["Region"] = gdf.index
```

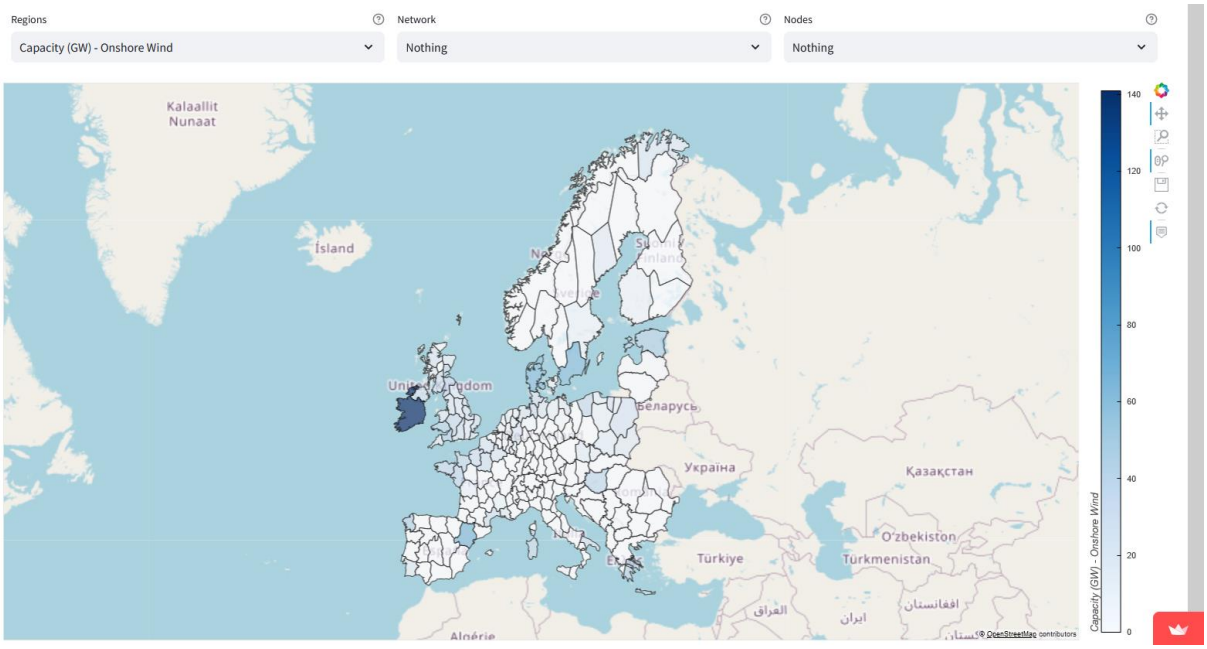
plot is then defined:

```
    plot = gdf.hvplot(
        geo=True,
        projection='GOOGLE_MERCATOR',
        height=720,
        tiles=config["tiles"],
        **kwargs
    ).opts(**opts)
```

Thus, if:

```
r_sel = ('Capacity (GW)', 'Onshore Wind')
```

We obtain:



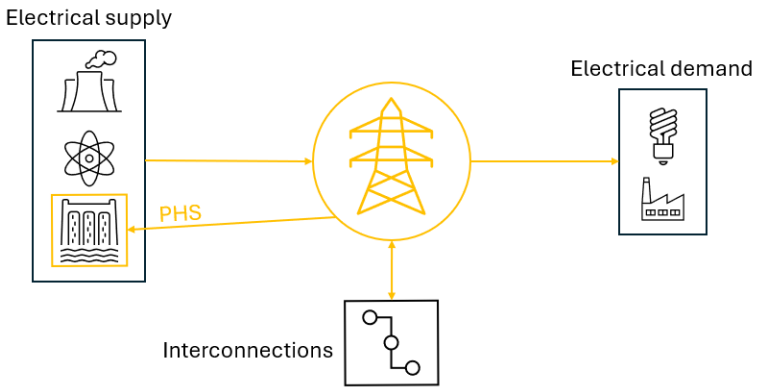
This means that we can display any value we want on the GeoJSON polygons !

Note: if we focus only on the Spatial Configuration view, much of the original code can be removed.

Balancing Networks

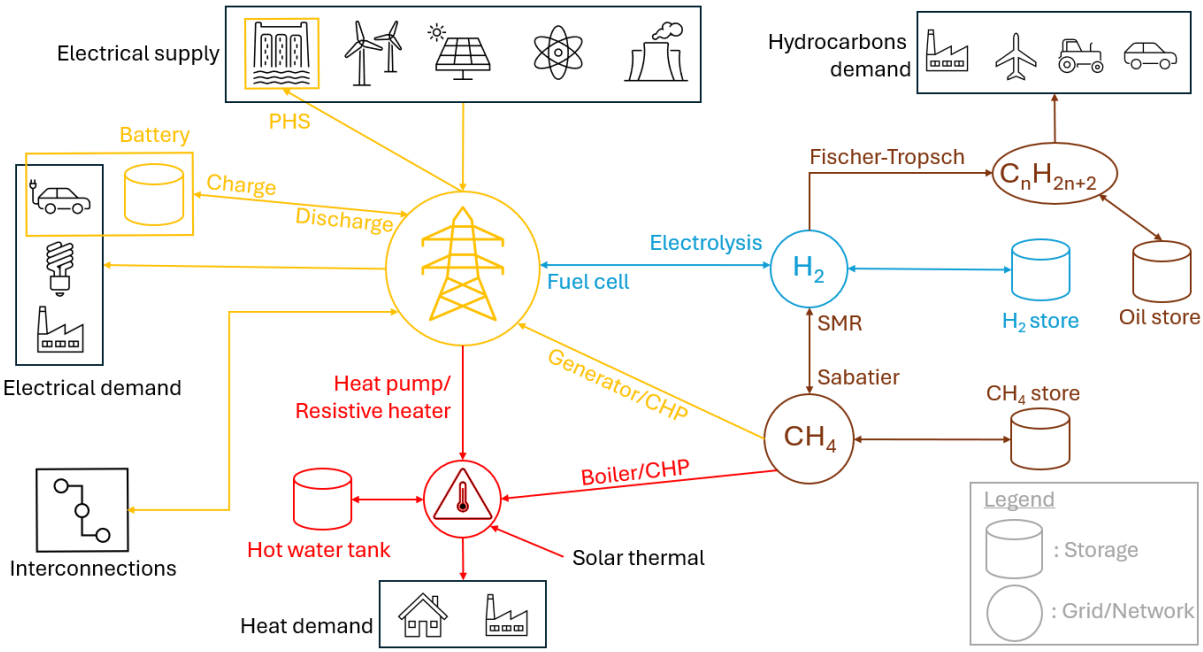
Electricity as such cannot be stored. For the electrical grid to function, electricity consumption must always be equivalent to electricity production. As soon as an individual consumes 1 kilowatt, 1 kilowatt must be simultaneously produced by a generation unit on the grid. Thermal power plants (gas, coal, ...), dams, and nuclear power plants are called controllable: their output can be adjusted according to demand. They can perform load following, meaning they can adapt to variations in demand.

Historically, the operation of the electricity grid was simpler. There were dams and thermal power stations upstream which produced electricity and downstream, an electrical demand. Electricity demand determined electricity production. In the event of a production surplus, part of the electricity could be used to pump water into the upper basin of a Pumped-Storage Hydroelectricity (PHS) plant. In the event of a surplus or shortage, a country could supply or consume electricity to/from its neighboring countries via interconnections.



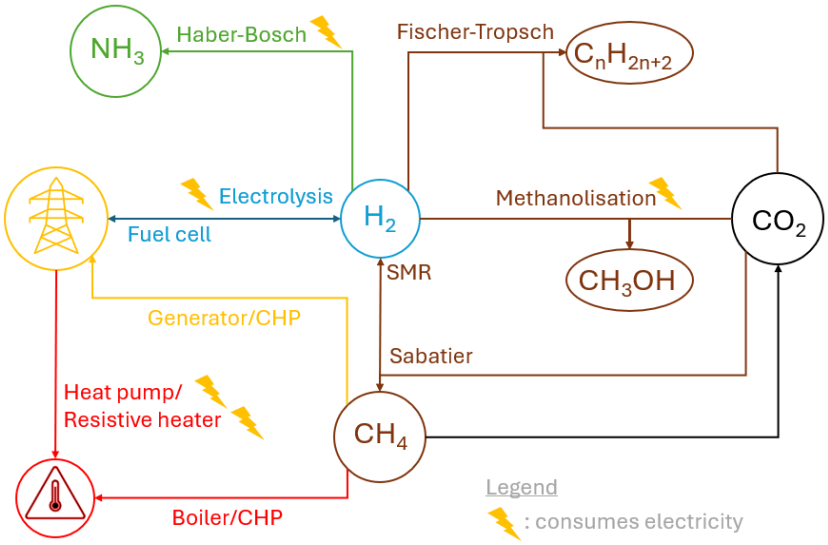
The current and future electricity network is and will be more complicated. Indeed, wind turbines and solar panels (Photovoltaic, PV) are called intermittent: their production depends on weather conditions and day/night cycles. The more an energy system is made up of renewable energies, the less there are controllable units which allow supply to be adapted to demand. Other balancing capacities are therefore necessary. These are capable of either increasing production, shifting consumption to a more opportune moment when production is higher or converting electricity to store it during surplus periods for redistribution during shortages. The combination of these elements constitutes the flexibility of the electricity network.

To understand the flexibility of the electricity network and balancing capabilities, it is necessary to understand the topology of the energy network as a whole and, in particular, how it was modeled in PyPSA-Eur. The Figure below is a simplified view of the PyPSA-Eur energy network, and it does not include all the flows and interactions, but it allows to understand the essentials for balancing.



The network is organized into grids/networks of carriers. These networks can exchange material or energy with each other via physical or chemical processes. Each network has incoming (production), outgoing (consumption) and bidirectional (storage) flows.

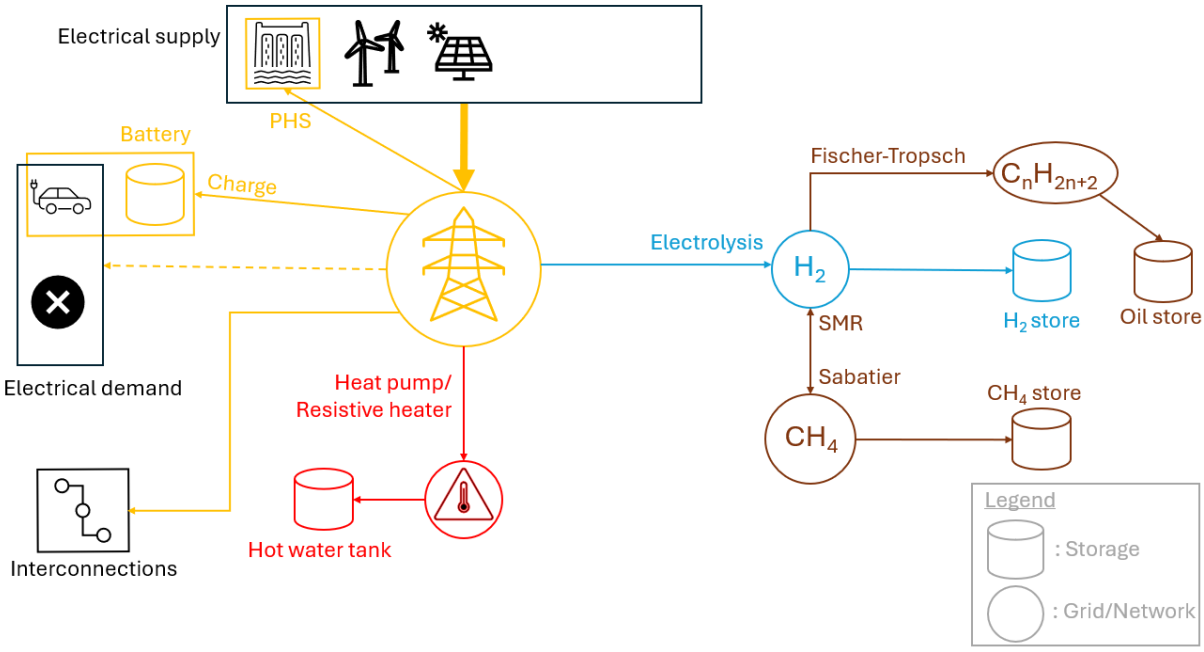
They were not all present in the Figure above so as not to overload information, but PyPSA-Eur models the synthesis of several molecules from electricity, H_2 and CO_2 (figure below). Processes that are modeled as consuming electricity are accompanied by a small flash. Every leaving arrow from CH_4 produces CO_2 as well. SMR stands for Steam Methane Reforming and CHP for Combine Heat and Power generator.



This network configuration makes it possible to store surplus production to redistribute it later during periods of scarcity. Indeed, there are 2 phenomena that can unbalance the electricity network: a surplus of production compared to demand, or a lack of production compared to demand.

1. Surplus of production

This is the least serious situation. On sunny and windy days, PV and wind turbines can produce more than the electricity demand. In this case, we can either curtail the production or balance the energy system by finding other uses for this surplus of production. The figure below shows other uses of this excess electricity.



During period of abundance, excess electricity can be transferred to neighboring countries via interconnexions if they need it.

It is interesting to operate heat pumps and resistive heaters during periods of abundance. If the instantaneous heat demand is already satisfied, this excess heat can be stored in the Hot Water Tanks.

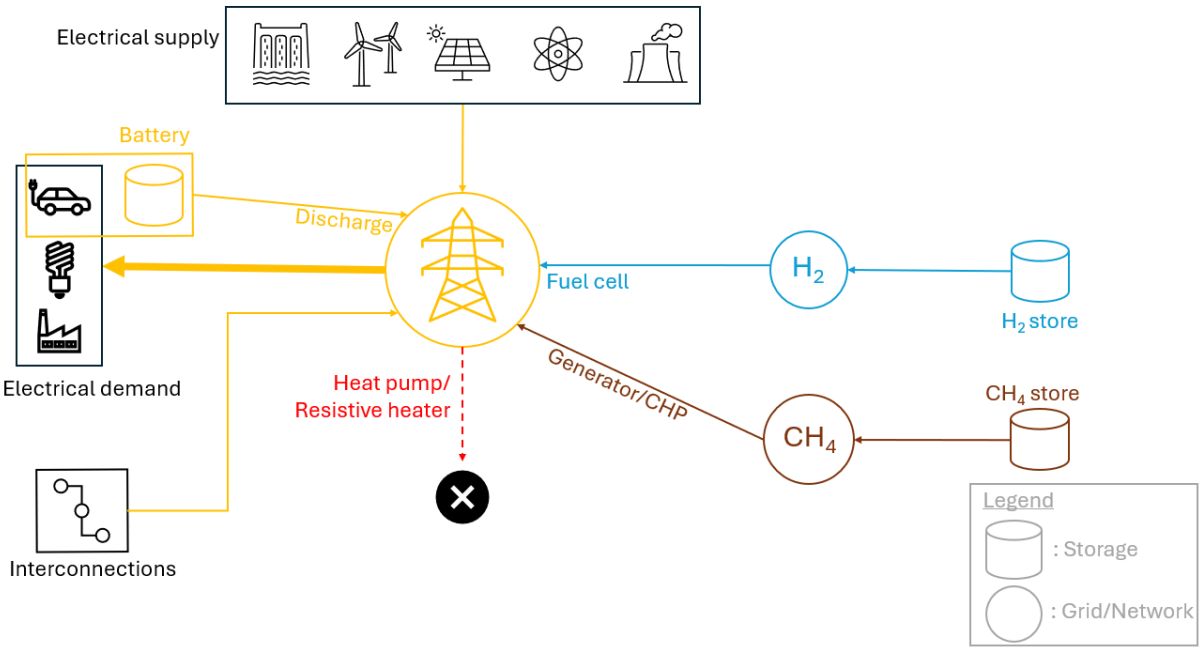
Industrial batteries and Battery Electric Vehicles (BEV) can be charged during this period. Same for PHS.

This is also the ideal time to run all the industrial processes described in the figure with molecules and consuming electricity: electrolysis, methanolisation and Haber-Bosch.

Unfortunately, if either the storage capacities for these molecules are full, or the devices to fill them are already running at full power, the electricity production will have to be curtailed. Even more unfortunate when we look at the next point.

2. Lack of production

This is the most serious situation. If for any reason (for example, lack of wind and sun), production becomes lower than consumption, other sources of production must be quickly found and/or consumption must be reduced to balance the electrical grid. The figure below illustrates the possibilities for that balancing.



Thermal plants, nuclear plants, Hydro, PHS (all controllable units) can modulate their power upwards to increase production. In a 100% renewable system, only Hydro and PHS remain.

If neighboring countries produce excess electricity, some can be imported to the national grid.

The use of HPs and resistive heaters could be temporarily stopped (and moved to a more convenient time). The same logic applies to electrolysis, Haber-Bosch and methanolisation. The charging of BEVs can also be delayed. All these electrical consumptions which can be delayed without serious consequences constitute the variable load.

BEVs via Vehicle-to-Grid (V2G) and industrial batteries could deliver power to the electricity grid. Here we see the dual role that BEVs can play.

Generators/CHP could burn CH_4, and fuel cells could oxidize H_2 to produce electricity. In a 100% renewable system, only H_2 can be used to produce electricity.

Unfortunately, if production remains too low, part of the demand will have to be eliminated (i.e. load shedding occurs, which is not modelled in PyPSA-Eur). If nothing is done, a blackout could occur (not modelled neither; PyPSA-Eur always refine solutions to meet demand).