Capacity Expansion Planning with pypsa

Contents

> **Note**
>
> Also in this tutorial, you might want to refer to the PyPSA documentation: https://pypsa.readthedocs.io.

From electricity market modelling to capacity expansion planning

Review the problem formulation of the electricity market model. Below you can find an adapted version where the capacity li Back to top n promoted to **decision variables** with corresponding terms in the *objective function* and *new constraints for their expansion limits* (e.g. wind and solar potentials). This is known as **capacity expansion problem**.

$$\min_{g,e,f,G,E,F} \quad \sum_{i,s,t} w_t o_s g_{i,s,t} + \sum_{i,s} c_s G_{i,s} + c_{r,\text{dis/charge}} G_{i,r,\text{dis/charge}} + c_r E_{i,r} + c_\ell F_\ell$$

such that

$$\begin{aligned}
d_{i,t} &= \sum_s g_{i,s,t} - \sum_\ell K_{i\ell} f_{\ell,t} \\
0 &\leq g_{i,s,t} \leq \hat{g}_{i,s,t} G_{i,s} \\
0 &\leq g_{i,r,t,\text{dis/charge}} \leq G_{i,r,\text{dis/charge}} && \text{storage} \\
0 &\leq e_{i,r,t} \leq E_r && \text{stor} \\
e_{i,r,t} &= \eta^0_{i,r,t} e_{i,r,t-1} + \eta^1_r g_{i,r,t,\text{charge}} - \frac{1}{\eta^2_r} g_{i,r,t,\text{discharge}} && \text{st} \\
-F_\ell &\leq f_{\ell,t} \leq F_\ell \\
0 &= \sum_\ell C_{\ell c} x_\ell f_{\ell,t} \\
\underline{G}_{i,s} &\leq G_{i,s} \leq \overline{G}_{i,s} && \text{generator capacity} \\
\underline{G}_{i,r,\text{dis/charge}} &\leq G_{i,r,\text{dis/charge}} \leq \overline{G}_{i,r,\text{dis/charge}} && \text{storage power capacity} \\
\underline{E}_{i,r} &\leq E_{i,r} \leq \overline{E}_{i,r} && \text{storage energy} \\
\underline{F}_\ell &\leq F_\ell \leq \overline{F}_\ell && \text{line capacity}
\end{aligned}$$

New decision variables for capacity expansion planning:

- $G_{i,s}$ is the generator capacity at bus i, technology s,
- F_ℓ is the transmission capacity of line ℓ,
- $G_{i,r,\text{dis-/charge}}$ denotes the charge and discharge capacities of storage unit r at bus i,
- $E_{i,r}$ is the energy capacity of storage r at bus i and time step t.

New parameters for capacity expansion planning:

- $c_\star$ is the capital cost of technology $\star$ at bus i
- w_t is the weighting of time step t (e.g. number of hours it represents)
- $\underline{G}_\star, \underline{F}_\star, \underline{E}_\star$ are the minimum capacities per technology and location/connection.
- $\underline{G}_\star, \underline{F}_\star, \underline{E}_\star$ are the maximum capacities per technology and location.

Back to top

> **Note**
>
> For a full reference to the optimisation problem description, see https://pypsa.readthedocs.io/en/latest/optimal_power_flow.html

> **Note**
>
> If you have not yet set up Python on your computer, you can execute this tutorial in your browser via Google Colab. Click on the rocket in the top right corner and launch "Colab". If that doesn't work download the `.ipynb` file and import it in Google Colab.
>
> Then install the following packages by executing the following command in a Jupyter cell at the top of the notebook.
>
> ```
> !pip install pypsa pandas matplotlib highspy
> ```

First things first! We need a few packages for this tutorial:

```
import pypsa
import pandas as pd
import matplotlib.pyplot as plt

plt.style.use("bmh")
```

Prerequisites: handling technology data and costs

We maintain a database (PyPSA/technology-data) which collects assumptions and projections for energy system technologies (such as costs, efficiencies, lifetimes, etc.) for given years, which we can load into a `pandas.DataFrame`. This requires some pre-processing to load (e.g. converting units, setting defaults, re-arranging dimensions):

Back to top

```
year = 2030
url = f"https://raw.githubusercontent.com/PyPSA/technology-data/master/outputs/cost
costs = pd.read_csv(url, index_col=[0, 1])
```

```
costs.loc[costs.unit.str.contains("/kW"), "value"] *= 1e3
costs.unit = costs.unit.str.replace("/kW", "/MW")

defaults = {
    "FOM": 0,
    "VOM": 0,
    "efficiency": 1,
    "fuel": 0,
    "investment": 0,
    "lifetime": 25,
    "CO2 intensity": 0,
    "discount rate": 0.07,
}
costs = costs.value.unstack().fillna(defaults)

costs.at["OCGT", "fuel"] = costs.at["gas", "fuel"]
costs.at["CCGT", "fuel"] = costs.at["gas", "fuel"]
costs.at["OCGT", "CO2 intensity"] = costs.at["gas", "CO2 intensity"]
costs.at["CCGT", "CO2 intensity"] = costs.at["gas", "CO2 intensity"]
```

Let's also write a small utility function that calculates the **annuity** to annualise investment costs. The formula is

$$a(r, n) = \frac{r}{1 - (1 + r)^{-n}}$$

where r is the discount rate and n is the lifetime.

```
def annuity(r, n):
    return r / (1.0 - 1.0 / (1.0 + r) ** n)
```

```
annuity(0.07, 20)
```

```
0.09439292574325567
```

Back to top

Based on this, we can calculate the short-term marginal generation costs (STMGC, €/MWh):

```
costs["marginal_cost"] = costs["VOM"] + costs["fuel"] / costs["efficiency"]
```

and the annualised investment costs (`capital_cost` in PyPSA terms, €/MW/a):

```
annuity = costs.apply(lambda x: annuity(x["discount rate"], x["lifetime"]), axis=1)
```

```
costs["capital_cost"] = (annuity + costs["FOM"] / 100) * costs["investment"]
```

Loading time series data

We are also going to need some time series for wind, solar and load. For now, we are going to recycle the time series we used at the beginning of the course. They are given for Germany in the year 2015.

```
url = (
    "https://tubcloud.tu-berlin.de/s/pKttFadrbTKSJKF/download/time-series-lecture-2
)
ts = pd.read_csv(url, index_col=0, parse_dates=True)
```

```
ts.head(3)
```

	load	onwind	offwind	solar	prices
2015-01-01 00:00:00	41.151	0.1566	0.7030	0.0	NaN
2015-01-01 01:00:00	40.135	0.1659	0.6875	0.0	NaN
2015-01-01 02:00:00	39.106	0.1746	0.6535	0.0	NaN

Let's convert the load time series from GW to MW, the base unit of PyPSA:

```
ts.load *= 1e3
```

Back to top

We are also going to adapt the temporal resolution of the time series, e.g. sample only every other hour, to save some time:

```
resolution = 4
ts = ts.resample(f"{resolution}h").first()
```

Simple capacity expansion planning example

> **Note**
>
> See also https://model.energy.

In this tutorial, we want to build a replica of model.energy. This tool calculates the cost of meeting a constant electricity demand from a combination of wind power, solar power and storage for different regions of the world.

We deviate from model.energy by including offshore wind generation and electricity demand profiles rather than a constant electricity demand. Also, we are going to start with Germany only. You can adapt the code to other countries as an exercise.

Model Initialisation

For building the model, we start again by initialising an empty network.

```
n = pypsa.Network()
```

Then, we add a single bus...

```
n.add("Bus", "electricity")
```

```
Index(['electricity'], dtype='object')
```

...and tell the `pypsa.Network` object `n` what the snapshots of the model will be using the utility function `n.set_snapshots()`.

Back to top

```
n.set_snapshots(ts.index)
```

```
n.snapshots
```

```
DatetimeIndex(['2015-01-01 00:00:00', '2015-01-01 04:00:00',
               '2015-01-01 08:00:00', '2015-01-01 12:00:00',
               '2015-01-01 16:00:00', '2015-01-01 20:00:00',
               '2015-01-02 00:00:00', '2015-01-02 04:00:00',
               '2015-01-02 08:00:00', '2015-01-02 12:00:00',
               ...
               '2015-12-30 08:00:00', '2015-12-30 12:00:00',
               '2015-12-30 16:00:00', '2015-12-30 20:00:00',
               '2015-12-31 00:00:00', '2015-12-31 04:00:00',
               '2015-12-31 08:00:00', '2015-12-31 12:00:00',
               '2015-12-31 16:00:00', '2015-12-31 20:00:00'],
              dtype='datetime64[ns]', name='snapshot', length=2190, freq='4h')
```

The weighting of the snapshots (e.g. how many hours they represent, see w_t in problem formulation above) can be set in `n.snapshot_weightings`.

```
n.snapshot_weightings.head(3)
```

	objective	stores	generators
snapshot			
2015-01-01 00:00:00	1.0	1.0	1.0
2015-01-01 04:00:00	1.0	1.0	1.0
2015-01-01 08:00:00	1.0	1.0	1.0

```
n.snapshot_weightings.loc[:, :] = resolution
```

```
n.snapshot_weightings.head(3)
```

Back to top

	objective	stores	generators
snapshot			
2015-01-01 00:00:00	4.0	4.0	4.0
2015-01-01 04:00:00	4.0	4.0	4.0
2015-01-01 08:00:00	4.0	4.0	4.0

Adding Components

Then, we add all the technologies we are going to include as carriers.

```
carriers = [
    "onwind",
    "offwind",
    "solar",
    "OCGT",
    "hydrogen storage underground",
    "battery storage",
]

n.add(
    "Carrier",
    carriers,
    color=["dodgerblue", "aquamarine", "gold", "indianred", "magenta", "yellowgreen
    co2_emissions=[costs.at[c, "CO2 intensity"] for c in carriers],
)
```

```
Index(['onwind', 'offwind', 'solar', 'OCGT', 'hydrogen storage underground',
       'battery storage'],
      dtype='object')
```

Next, we add the demand time series to the model.

```
n.add(
    "Load",
    "demand",
    bus="electricity",
    p_set=ts.load,
)
```

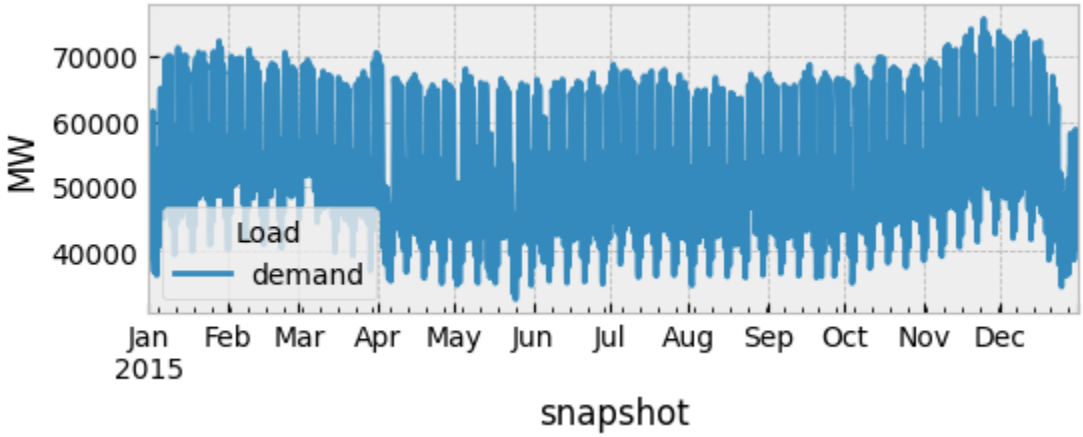
Back to top

```
Index(['demand'], dtype='object')
```

Let's have a check whether the data was read-in correctly.

```
n.loads_t.p_set.plot(figsize=(6, 2), ylabel="MW")
```

```
<Axes: xlabel='snapshot', ylabel='MW'>
```



We are going to add one dispatchable generation technology to the model. This is an open-cycle gas turbine (OCGT) with CO_2 emissions of 0.2 t/MWh$_{th}$.

```
n.add(
    "Generator",
    "OCGT",
    bus="electricity",
    carrier="OCGT",
    capital_cost=costs.at["OCGT", "capital_cost"],
    marginal_cost=costs.at["OCGT", "marginal_cost"],
    efficiency=costs.at["OCGT", "efficiency"],
    p_nom_extendable=True,
)
```

```
Index(['OCGT'], dtype='object')
```

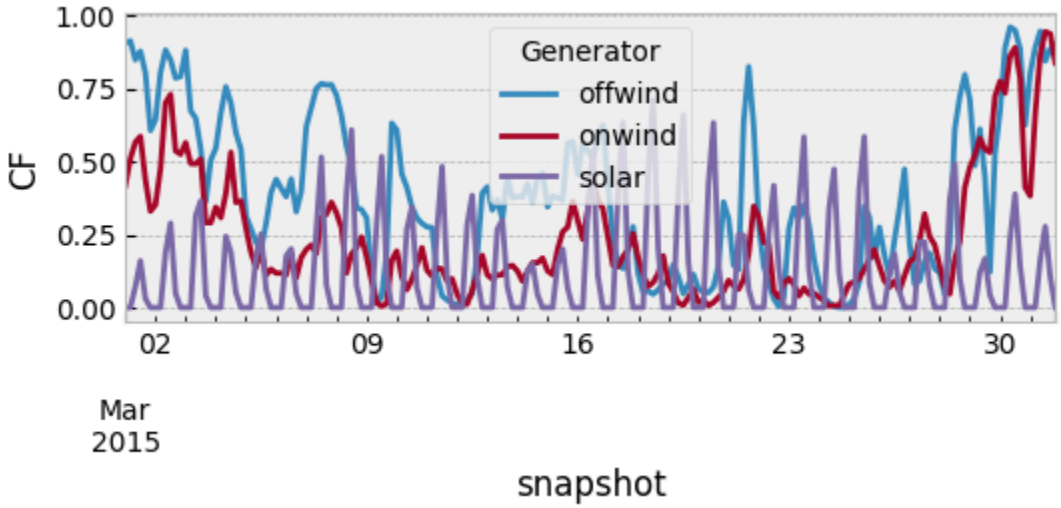
Adding the variable renewable generators works almost identically, but we also need to supply the capacity factors to the mo Back to top ibute `p_max_pu`.

```
for tech in ["onwind", "offwind", "solar"]:
    n.add(
        "Generator",
        tech,
        bus="electricity",
        carrier=tech,
        p_max_pu=ts[tech],
        capital_cost=costs.at[tech, "capital_cost"],
        marginal_cost=costs.at[tech, "marginal_cost"],
        efficiency=costs.at[tech, "efficiency"],
        p_nom_extendable=True,
    )
```

So let's make sure the capacity factors are read-in correctly.

```
n.generators_t.p_max_pu.loc["2015-03"].plot(figsize=(6, 2), ylabel="CF")
```

```
<Axes: xlabel='snapshot', ylabel='CF'>
```



Model Run

Then, we can already solve the model for the first time. At this stage, the model does not have any storage or emission limits implemented. It's going to look for the least-cost combination of variable renewables and the gas turbine to supply demand.

Back to top

```
n.optimize(solver_name="highs")
```

```
WARNING:pypsa.consistency:The following buses have carriers which are not defined:
Index(['electricity'], dtype='object', name='Bus')
```

```
WARNING:pypsa.consistency:The following buses have carriers which are not defined:
Index(['electricity'], dtype='object', name='Bus')
```

```
/opt/hostedtoolcache/Python/3.11.10/x64/lib/python3.11/site-packages/linopy/common.

coords for dimension(s) ['Generator'] is not aligned with the pandas object. Previo

INFO:linopy.model: Solve problem using Highs solver
```

```
INFO:linopy.io: Writing time: 0.13s
```

```
INFO:linopy.solvers:Log file at /tmp/highs.log
```

```
Running HiGHS 1.7.2 (git hash: 184e327): Copyright (c) 2024 HiGHS under MIT licence
Coefficient ranges:
  Matrix [1e-04, 4e+00]
  Cost   [4e-02, 2e+05]
  Bound  [0e+00, 0e+00]
  RHS    [3e+04, 8e+04]
Presolving model
9900 rows, 7714 cols, 23130 nonzeros  0s
9900 rows, 7714 cols, 23130 nonzeros  0s
Presolve : Reductions: rows 9900(-9818); columns 7714(-1050); elements 23130(-19624
Solving the presolved LP
Using EKK dual simplex solver - serial
  Iteration        Objective     Infeasibilities num(sum)
          0     0.0000000000e+00 Pr: 2190(1.25606e+09) 0s
       7615     3.1624531941e+10 Pr: 0(0) 0s
Solving the original LP from the solution after postsolve
Model   status      : Optimal
Simplex   iterations: 7615
Objective value     :  3.1624531941e+10
HiGHS run time      :          0.16
Writing the solution to /tmp/linopy-solve-u2o81ptv.sol
```

Back to top

```
INFO:linopy.constants: Optimization successful:
Status: ok
Termination condition: optimal
Solution: 8764 primals, 19718 duals
Objective: 3.16e+10
Solver model: available
Solver message: optimal
```

```
INFO:pypsa.optimization.optimize:The shadow-prices of the constraints Generator-ext
```

```
('ok', 'optimal')
```

Model Evaluation

The total system cost in billion Euros per year:

```
n.objective / 1e9
```

```
31.62453194099228
```

The optimised capacities in GW:

```
n.generators.p_nom_opt.div(1e3)  # GW
```

```
Generator
OCGT       70.096357
onwind     -0.000000
offwind    49.326062
solar      85.967820
Name: p_nom_opt, dtype: float64
```

The total energy generation by technology in GW:

Back to top

```
n.snapshot_weightings.generators @ n.generators_t.p.div(1e6)  # TWh
```

```
Generator
OCGT       235.778879
onwind       0.000000
offwind    150.379873
solar       92.766988
Name: generators, dtype: float64
```

While we get the objective value through `n.objective`, in many cases we want to know how the costs are distributed across the technologies. We can use the statistics module for this:

```
(n.statistics.capex() + n.statistics.opex()).div(1e6)
```

```
component  carrier
Generator  OCGT       18596.014468
           offwind     8613.359135
           solar       4415.158338
dtype: float64
```

Possibly, we are also interested in the total emissions:

```
emissions = (
    n.generators_t.p
    / n.generators.efficiency
    * n.generators.carrier.map(n.carriers.co2_emissions)
)  # t/h
```

```
n.snapshot_weightings.generators @ emissions.sum(axis=1).div(1e6)  # Mt
```

```
113.86394645337052
```

Plotting Optimal Dispatch

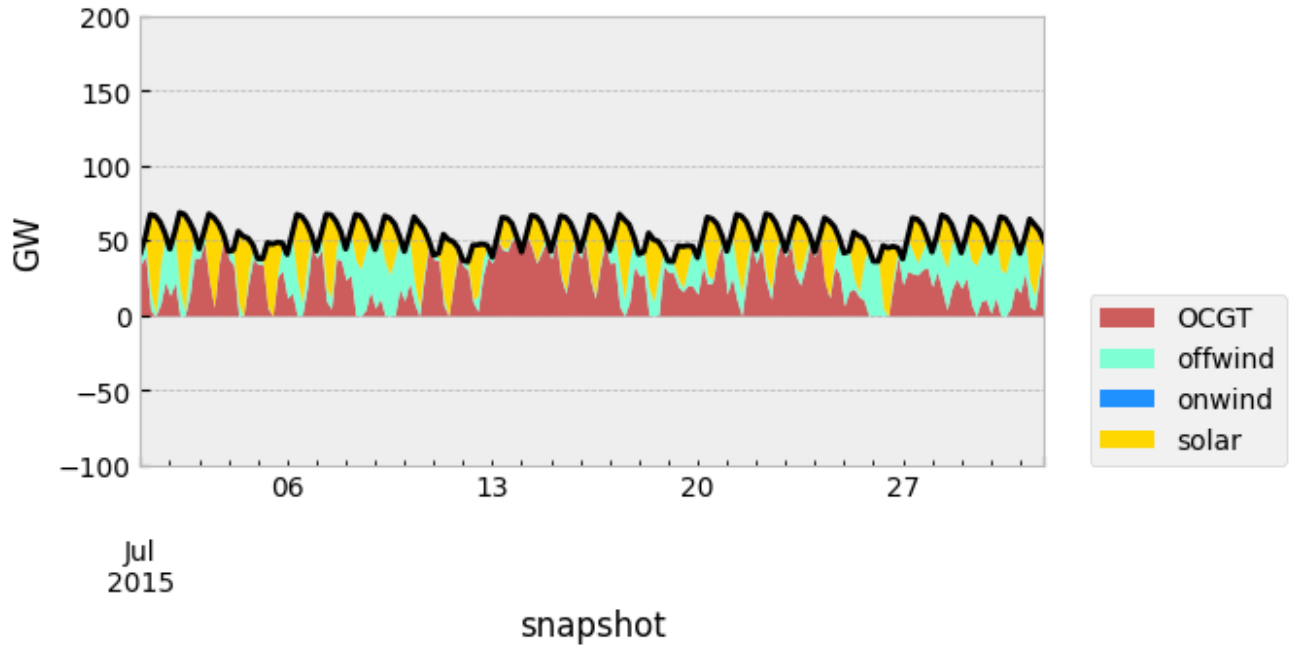
This function takes the network object `n` as an argument and, optionally, a time frame. We want to plot the load time series, and stacked area charts for electricity feed-in and storage charging. Technologies should be colo[Back to top]. color defined in `n.carriers`.

```
def plot_dispatch(n, time="2015-07"):
    p = (
        n.statistics.energy_balance(aggregate_time=False)
        .groupby("carrier")
        .sum()
        .div(1e3)
        .drop("-")
        .T
    )

    fig, ax = plt.subplots(figsize=(6, 3))

    color = p.columns.map(n.carriers.color)

    p.where(p > 0).loc[time].plot.area(
        ax=ax,
        linewidth=0,
        color=color,
    )

    charge = p.where(p < 0).dropna(how="all", axis=1).loc[time]

    if not charge.empty:
        charge.plot.area(
            ax=ax,
            linewidth=0,
            color=charge.columns.map(n.carriers.color),
        )

    n.loads_t.p_set.sum(axis=1).loc[time].div(1e3).plot(ax=ax, c="k")

    plt.legend(loc=(1.05, 0))
    ax.set_ylabel("GW")
    ax.set_ylim(-100, 200)
```

Let's test it:

```
plot_dispatch(n)
```

Back to top



Adding Storage Units

Alright, but there are a few important components missing for a system with high shares of renewables? What about short-term storage options (e.g. batteries) and long-term storage options (e.g. hydrogen storage)? Let's add them too.

First, the battery storage. We are going to assume a fixed energy-to-power ratio of 6 hours, i.e. if fully charged, the battery can discharge at full capacity for 6 hours. For the capital cost, we have to factor in both the capacity and energy cost of the storage. We are also going to enforce a cyclic state-of-charge condition, i.e. the state of charge at the beginning of the optimisation period must equal the final state of charge.

```
n.add(
    "StorageUnit",
    "battery storage",
    bus="electricity",
    carrier="battery storage",
    max_hours=6,
    capital_cost=costs.at["battery inverter", "capital_cost"]
    + 6 * costs.at["battery storage", "capital_cost"],
    efficiency_store=costs.at["battery inverter", "efficiency"],
    efficiency_dispatch=costs.at["battery inverter", "efficiency"],
    p_nom_extendable=True,
    cyclic_state_of_charge=True,
)
```

Back to top

```
Index(['battery storage'], dtype='object')
```

Second, the hydrogen storage. This one is composed of an electrolysis to convert electricity to hydrogen, a fuel cell to re-convert hydrogen to electricity and underground storage (e.g. in salt caverns). We assume an energy-to-power ratio of 168 hours, such that this type of storage can be used for weekly balancing.

```
capital_costs = (
    costs.at["electrolysis", "capital_cost"]
    + costs.at["fuel cell", "capital_cost"]
    + 168 * costs.at["hydrogen storage underground", "capital_cost"]
)

n.add(
    "StorageUnit",
    "hydrogen storage underground",
    bus="electricity",
    carrier="hydrogen storage underground",
    max_hours=168,
    capital_cost=capital_costs,
    efficiency_store=costs.at["electrolysis", "efficiency"],
    efficiency_dispatch=costs.at["fuel cell", "efficiency"],
    p_nom_extendable=True,
    cyclic_state_of_charge=True,
)
```

```
Index(['hydrogen storage underground'], dtype='object')
```

Ok, lets run the again, now with storage, and see what's changed.

```
n.optimize(solver_name="highs")
```

▶ Show code cell output

```
n.generators.p_nom_opt  # MW
```

Back to top

```
Generator
OCGT       70096.356831
onwind        -0.000000
offwind    49326.061998
solar      85967.819687
Name: p_nom_opt, dtype: float64
```

```
n.storage_units.p_nom_opt  # MW
```

```
StorageUnit
battery storage                -0.0
hydrogen storage underground   -0.0
Name: p_nom_opt, dtype: float64
```

Nothing! The objective value is the same, and no storage is built.

Adding emission limits

The gas power plant offers sufficient and cheap enough backup capacity to run in periods of low wind and solar generation. But what happens if this source of flexibility disappears. Let's model a 100% renewable electricity system by adding a CO_2 emission limit as global constraint:

```
n.add(
    "GlobalConstraint",
    "CO2Limit",
    carrier_attribute="co2_emissions",
    sense="<=",
    constant=0,
)
```

```
Index(['CO2Limit'], dtype='object')
```

When we run the model now...

Back to top

```
n.optimize(solver_name="highs")
```

► Show code cell output

...and inspect the capacities built...

```
n.generators.p_nom_opt  # MW
```

```
Generator
OCGT           -0.000000
onwind     267431.119057
offwind     61878.836900
solar      288960.778468
Name: p_nom_opt, dtype: float64
```

```
n.storage_units.p_nom_opt  # MW
```

```
StorageUnit
battery storage                 50461.162073
hydrogen storage underground    48721.593436
Name: p_nom_opt, dtype: float64
```

```
n.storage_units.p_nom_opt.div(1e3) * n.storage_units.max_hours  # GWh
```

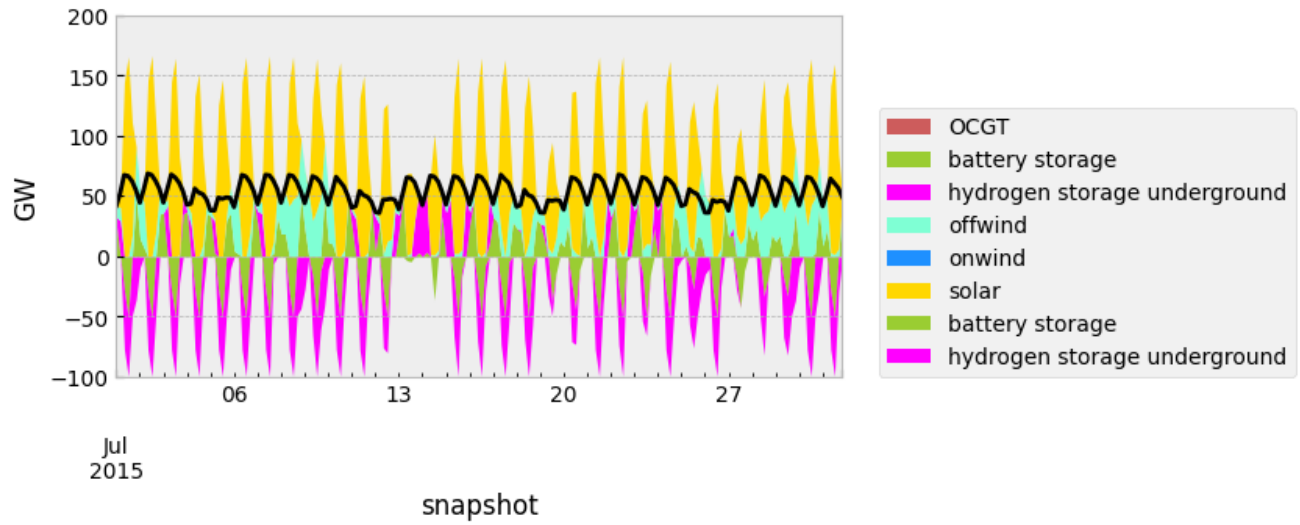
```
StorageUnit
battery storage                   302.766972
hydrogen storage underground    8185.227697
dtype: float64
```

... we now see some storage. So how does the optimised dispatch of the system look like?

```
plot_dispatch(n)
```

```
/opt/hostedtoolcache/Python/3.11.10/x64/lib/python3.11/site-packages/pandas/plottin

Attempting to set identical low and high ylims makes transformation singular; autom
```

Back to top



We are also keen to see what technologies constitute the largest cost components. For that we're going to define a small helper function:

```
def system_cost(n):
    tsc = pd.concat([n.statistics.capex(), n.statistics.opex()], axis=1)
    return tsc.sum(axis=1).droplevel(0).div(1e9).round(2)  # billion €/a
```

```
system_cost(n)
```

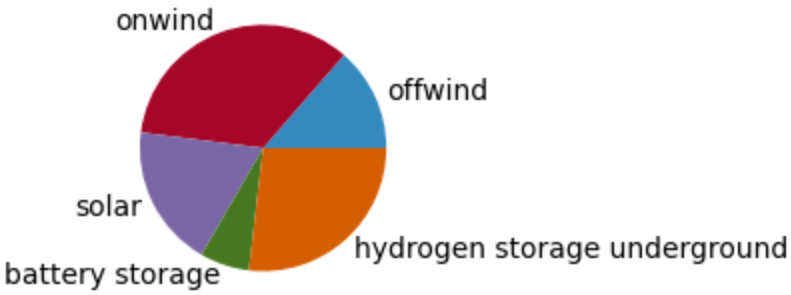
```
carrier
offwind                          10.81
onwind                           27.29
solar                            14.84
battery storage                   5.15
hydrogen storage underground     21.32
dtype: float64
```

This series, we can then process into plots, e.g. a pie chart:

```
system_cost(n).plot.pie(figsize=(2, 2))
```

```
<Axes: >
```

Back to top



or use to compute the cost per unit of electricity consumed:

```
demand = n.snapshot_weightings.generators @ n.loads_t.p_set.sum(axis=1)
```

```
system_cost(n).sum() * 1e9 / demand.sum()
```

```
165.80858652533482
```

```
n.export_to_netcdf("network-new.nc");
```

```
INFO:pypsa.io:Exported network 'network-new.nc' contains: generators, buses, global
```

Warning

Always consider, that the load data is given in units of power (MW) and if your resolution is not hourly, you need to multiply by the snapshot weighting to get the energy consumed!

Sensitivity Analysis

Sensitivity analyses constitute a core activity of energy system modelling. Below, you can find sensitivity analyses regarding the

Back to top

1. variation in allowed CO_2 emissions
2. variation in solar overnight costs

3. variation in offshore wind potentials

```
sensitivity = {}
for co2 in [150, 100, 50, 25, 0]:
    n.global_constraints.loc["CO2Limit", "constant"] = co2 * 1e6
    n.optimize(solver_name="highs")
    sensitivity[co2] = system_cost(n)
```

▶ Show code cell output

```
df = pd.DataFrame(sensitivity).T  # billion Euro/a
df.plot.area(
    stacked=True,
    linewidth=0,
    color=df.columns.map(n.carriers.color),
    figsize=(4, 4),
    xlim=(0, 150),
    xlabel=r"CO$_2$ emissions [Mt/a]",
    ylabel="System cost [bn€/a]",
    ylim=(0, 100),
)
plt.legend(frameon=False, loc=(1.05, 0))
```

Exercises

Explore the model by changing the assumptions and available technologies. Here are a few inspirations:

- Rerun the model with cost assumptions for 2050.
- What if either hydrogen or battery storage cannot be expanded?
- What if you can either only build solar or only build wind?
- What if we had a flat electricity demand profile (like in model.energy, i.e. average the demand time series)?
- Vary the energy-to-power ratio of the hydrogen storage. What ratio leads to lowest costs?
- How bad is the 4-hourly resolution used for demonstration? How does it compare to hourly or 2-hourly resolution?
- On model.energy, you can download capacity factors for onshore wind and solar for any region in the world. Drop offshore wind from the model and use the onshore wind and solar prices from another region of the world. Try a few. What changes?

Back to top

- Add nuclear as another dispatchable low-emission generation technology (similar to OCGT). Perform a sensitivity analysis trying to answer how low the capital cost of a nuclear plant would need to be to be chosen.

```
n.export_to_netcdf("network-cem.nc");
```

Back to top