2020_notebook_optimization_SOLUTIONS

April 13, 2020

Table of Contents

0.1 Methods for technical and economic energy analysis, week 9

1 Linear Optimization for Energy System Modeling

1.1 Teachers

Martin Soini, room B604, Martin.Soini@unige.ch Arthur Rinaldi, room B604, Arthur.Rinaldi@unige.ch

1.2 Subjects and objectives of this assignment

- Define a linear optimization model for a stylized national energy system.
- Explore the impact of the CO2 emissions price on the optimal composition of the system in the context of an energy transition.
- Analyze the effect of adding large amounts of solar power to the system.
- BONUS Exercise: Analyze the effect of storage on the value of solar power.

1.3 Group size

2 people per group

1.4 Final product

100 points in total (+25 bonus points); the distribution of points per task can be found in the Word template - a report (please use the provided Word template and preserve the provided structure; no LaTeX) - an Excel file including your analysis (please use the provided Excel template and preserve the provided structure; no Python/Pandas) - The Jupyter Notebook is only relevant for grading if you work on the last (BONUS) exercise. However, please submit it in any case. - Your may write the report in either English or French. You decide. However, if you are not comfortable expressing yourself in English, please answer in French. The most important thing is to clearly express your ideas so they can be understood while grading.

1.5 Relevant research literature (on Moodle)

- de Sisternes *et al.*: **The value of energy storage in decarbonizing the electricity sector**, Applied Energy 175 (2016) 368–379
- Hirth: **The market value of variable renewables**, Energy Economics 38 (2013) 218–236
- Schlecht and Weigt: **Linking Europe: The Role of the Swiss Electricity Transmission Grid until 2050**, Swiss Journal of Economics and Statistics, Volume 151, Issue 2, pp 125–165

1.6 Submission deadline

Since this is the last lecture of the course, you have an additional week to complete the assignment. The report, the Excel analysis file, and the Jupyter notebook (*.ipynb) have to be uploaded to

Moodle on Wednesday 15 May 2019 at 17:00 at the latest. Any submission later than this date will not be reviewed.

1.7 Introduction

In this assignment we build a simple optimization model representing a stylized national power system. While the representation of the system is extremely simplified, it allows to investigate some basic mechanisms which one would also encounter in a more detailed model.

This system consists of * existing nuclear, coal, gas, wind and solar power plants with a given installed power capacity; only the running (variable) costs of these plants are considered. * Additionally, the optimized solution may include new gas, wind, and solar capacity, whose capacity is optimized and causes certain fixed costs.

The operation (power output) of all of these plants is optimized for a single year. This year is approximated by four seasons. Therefore, we implicitly assume that within each of the seasons the operation of all plants is constant (only one power production level for nuclear power in spring, summer, etc.).

This power system is loosely based on the German power supply. This system is a suitable example due to the diversity of the power plants and the *relatively* small interconnection capacity to neighboring countries. In contrast, modelling the Swiss power supply is much more difficult. This is due to the heavy reliance on hydro power and the large amount of energy exchange with the neighboring countries: Modelling the Swiss grid generally requires at least modelling the neighboring countries as well.

Despite its simple structure, this model allows to capture some interesting features related to the expansion and profitability of wind and solar power:

- The optimal system configuration after a nuclear and coal phase-out strongly depends on the CO2 emissions price. Both wind/solar and natural gas power plants could make up for the reduced generation capacity. Here we investigate how the optimal mix electricity mix (gas or wind/solar) of a future power system changes for increasing emissions prices.
- Based on these results, we evaluate the total cost and value of the new wind, solar, and gas plants.
- Then we turn to the case where new solar is forced into the system *exogenously*, i.e. defined by us instead of being optimized by the model.
- Finally, in the BONUS exercise we consider the case of additional storage capacity in the model. This storage allows to "spread out" the solar production in summer over the whole year. It therefore affects the value of this resource.

In our model we aggregate all power plants which are using a certain fuel. For example, we assume that there is 11'000 MW of nuclear power. In a real power system, there would be 8-11 single nuclear power plants, each with their own operating schedule and individual technical properties. The same holds for the coal, gas, wind and solar plants. Treating them all as a single combined power plant is a common approximation. It makes the model much simpler while still allowing us to learn interesting things. However, through these simplifications we miss out on many technical

details. *Whether or not these details are relevant depends on the research questions we are asking! It is very important to align the model design with its purpose.*

All power plants are characterized by their efficiency, the fuel cost, the CO2 intensity of the fuel, and their existing capacity. In addition, newly built power plants (new gas, wind, and solar) also have an annualized capital cost (i.e. annual share of the capital investment cost). As mentioned in the lecture, this annual cost is calculated from the discount rate, the power plant lifetime, and the total capital cost per unit of installed capacity (i.e. what you need to pay upfront in order to get the power plant built). Wind and solar plants have seasonal capacity factors. They determine how much solar/wind power can be produced during each season.

Important:

The model is implemented in Python. You won't need to come up with code yourself, but you will write additional constraints based on the examples provided to you (Bonus task). Since these model components must be correct for rest of the assignment, you will be assisted to get this right in any case. Nevertheless, you should give it a try yourself. You don't need to understand in detail why things are implemented in Python the way they are. However, it is very important that you understand what the different parts of the model code are doing. The structure of an optimization model as the one shown here is always the same, independent of the programming language.

All model input and output data required for the analysis are generated for you, i.e. you don't have to take care of extracting the data from the model. Please copy the tables to the Excel file for analysis.

Tasks are labeled as either **discussion** or **analysis**. Discussion tasks don't require additional data analysis or plots.

The BONUS exercise requires some additional changes to the model, in order to include storage capacity in the system. Similar to the BONUS Task 3.b, this requires the definition of an additional constraint.

Very important:

You can reset the model at any time by selecting "Run->Run all above selected cell" in the Jupyter notebook. (For example if you get errors because model components have already been defined.)

The analysis is done in Excel. The most advanced Excel function required to perform the analysis is `SUMIFS`. If you are not familiar with it, please make sure you understand how it works and what it can do. For example, have a look at the example sheet in the Excel template file. Of course you are welcome to use any other Excel approach (pivot tables, `INDEX(.., MATCH())`, etc). However, the analysis must be done in the provided Excel template (one sheet per exercise).

All plots require a legend (if there is more than one plot series) and axes labels. Points will be deducted if plots are not properly formatted.

Copied content from the lecture slides will not be counted as part of the answer. If you feel the need to do copy material from the slides, please ask for clarifications.

Only base discussions on the input data! Don't speculate on system properties which are not explicitly defined within the assignment. In this assignment we don't discuss the limitations of the model and to what extent it realistically represents a power system.

Many of the tasks ask for comparisons with previous tasks. These are mandatory. Points are deducted if the comparisons are missing.

[1]:
```
'''
Run this cell. Apart from this, you can ignore it.
It contains Python dependencies and the convenience function to extract the␣
 ↪data from the model.
'''

%matplotlib inline

import itertools
import pyomo.environ as po
from pyomo.core.base.objective import SimpleObjective
from pyomo.core.base.constraint import IndexedConstraint
import matplotlib.pyplot as plt
from pyomo.opt import SolverFactory
solver = SolverFactory('glpk')

import pandas as pd
pd.set_option('display.max_rows', None)
pd.options.display.float_format = '{:.4f}'.format
pd.set_option('display.multi_sparse', False)
pd.options.display.max_columns = 500
from IPython.core.display import display, HTML

def get_data(comp, index):
    """" Extract values from a Pyomo object and returns a table. """
    if not isinstance(comp, IndexedConstraint):
        df = pd.Series(m.obj.expr() if isinstance(comp, SimpleObjective) else␣
 ↪comp.extract_values()).rename('value')
        df.loc[df < 0] = 0
    else:
        data = {key: m.dual[comp[key]] for key in comp}
        df = pd.Series(data).rename('value')

    if index:
        df.index.names = index
        df = df.reset_index()

    return pd.DataFrame(df)
```

1.8 Definition of the input data

First we define all the input data that our model will make use of. The only thing you need to do here is to understand what the data stand for and to run all cells so the data is defined.

We start with basic cost and power plant data: * The CO2 intensity of the fuel (units: tCO2/MWhfuel) serves to calculate the emission cost associated with the production of one unit of electric energy * The fuel prices in EUR/MWhfuel * The power plant efficiency (MWhelectricity/MWhfuel) * The existing power plant capacity (MWelectricity)

[2]:
```
co2_intensity_gas = 0.202  # t_MWh_fuel
co2_intensity_coal = 0.341  # t_MWh_fuel

co2_intensity = {'solar': 0, 'solar_new': 0, 'wind': 0, 'wind_new': 0,␣
↪'nuclear': 0,  # t_MWh_fuel
                 'coal': co2_intensity_coal, 'gas': co2_intensity_gas,␣
↪'gas_new': co2_intensity_gas}  # t_MWh_fuel

fuel_cost_gas = 32.5  # EUR/MWh_fuel
fuel_cost_coal = 8.5  # EUR/MWh_fuel
fuel_cost_nuclear = 1.9  # EUR/MWh_fuel

fuel_cost = {'solar': 0, 'solar_new': 0, 'wind': 0, 'wind_new': 0,  # EUR/
↪MWh_fuel
             'nuclear': fuel_cost_nuclear, 'gas': fuel_cost_gas,  # EUR/MWh_fuel
             'coal': fuel_cost_coal, 'gas_new': fuel_cost_gas}  # EUR/MWh_fuel

eff = {'solar': 1, 'solar_new': 1, 'wind': 1, 'wind_new': 1,  # -
       'nuclear': 0.33, 'gas': 0.45, 'coal': 0.4, 'gas_new': 0.6}  # -

capacity_old = {'solar_new': 0, 'wind_new': 0, 'gas_new': 0,  # MW
                'solar': 40000, 'wind': 45000, 'nuclear': 11000,  # MW
                'gas': 30000, 'coal': 25000}  # MW
```

The annuity factor mentioned in the lecture is calculated from the discount rate and the power plant lifetime. For simplicity, we assume the same lifetime for wind, solar, and new gas plants.

This allows us to express the investment cost in units of EUR/MWelectricity/**year** and to consider a single year only. This means that we implicitly assume that the power plant operation this year repeats each year.

[3]:
```
discount_rate = 0.06  # (-)
lifetime = 25  # (years)

annuity_factor = ((1 + discount_rate) ** lifetime * discount_rate
                  / ((1 + discount_rate) ** lifetime - 1))  # 1/year

fixed_cost_wind = 2100000  # EUR/MW
```

```
fixed_cost_solar = 1500000  # EUR/MW
fixed_cost_gas = 1100000  # EUR/MW

fixed_cost = {'solar_new': fixed_cost_solar * annuity_factor,  # EUR/MW/year
              'wind_new': fixed_cost_wind * annuity_factor,  # EUR/MW/year
              'gas_new': fixed_cost_gas * annuity_factor}  # EUR/MW/year
```

Some of the input data have a time dependency: * The capacity factor (unitless) for wind and solar tells us how much average power can be produced during each season for each unit of installed capacity. Please have a look at the lecture slides for a more detailed discussion of the capacity factor. * The demand (MW) is the minimum total power which needs to be produced during each season.

[4]:
```
cf_wind = {'0_spring': 0.175, '1_summer': 0.131, '2_fall': 0.186, '3_winter': 0.
↪259}  # -
cf_solar = {'0_spring': 0.161, '1_summer': 0.19, '2_fall': 0.098, '3_winter': 0.
↪051}  # -
cf = {**{(plant, season): cf for season, cf in cf_wind.items() for plant in␣
↪['wind', 'wind_new']},  # -
      **{(plant, season): cf for season, cf in cf_solar.items() for plant in␣
↪['solar', 'solar_new']}}  # -

demand = {'0_spring': 45249, '1_summer': 45503, '2_fall': 48444, '3_winter':␣
↪47064}  # MW
```

Finally, we need to define the weight of the seasons: We will simplify a year by considering four average time slots (seasons). Within each season we optimize *power*. In order to get *energy*, we require the duration of each season (hour/year). For example, this is used to calculate the total yearly cost of energy production in the objective function below. In our case, we assume that all seasons have equal length, i.e. one fourth of the total number of hours of a year (8760): 8760 hours / 4 = 2190 hours: For example, if a power plant produces 1 MW of average power in spring, it produces 2190 MWh/year of electric energy.

When performing the analysis, please be very aware of the physical units you are dealing with!

[5]:
```
season_weight = {'0_spring': 2190, '1_summer': 2190, '2_fall': 2190, '3_winter':
↪ 2190}  # hours/year
```

1.9 Building the model

Now we have defined all necessary data to build the actual model. As discussed in the lecture, it consists of sets (indices), parameters (input data), variables (what's being optimized), the objective function (the value we want to minimize), and constraints.

First we define the model object `po.ConcreteModel` and call it `m`.

The second line is necessary to extract the electricity prices from the model later on. You may ignore this.

[6]:
```
m = po.ConcreteModel()
m.dual = po.Suffix(direction=po.Suffix.IMPORT)
```

1.9.1 Sets

The main **sets** of the model are the seasons and the power plants. We also define two subsets of the power plants: * `new_power_plants` and * `wind_solar`.

This is necessary because we define parameters and constraints which are *only* defined for new power plants (like the `fixed_cost`) or *only* for wind and solar (like the capacity factor `cf`). We express that the elements of the subsets must be contained in the `power_plants` set by using the `within=m.power_plants` argument.

[7]:
```
m.seasons = po.Set(initialize=['0_spring', '1_summer', '2_fall', '3_winter'])
m.power_plants = po.Set(initialize=['solar_new', 'wind_new', 'gas_new',␣
↪'solar', 'wind', 'nuclear', 'gas', 'coal'])
m.new_power_plants = po.Set(within=m.power_plants, initialize=['solar_new',␣
↪'wind_new', 'gas_new'])
m.wind_solar = po.Set(within=m.power_plants, initialize=['wind', 'wind_new',␣
↪'solar', 'solar_new'])
```

1.9.2 Parameters

The definition of the **parameters** consists largely in creating model components `po.Param` based on the data defined above.

Parameters are typically defined by a set and the input data. An exception is the CO2 price: It is just a single value and doesn't depend on any sets (only one price for all seasons and all power plants).

Products of sets evaluate to all combinations of the corresponding items, e.g. `m.power_plants * m.seasons` corresponds to all power plants in all seasons. Try to run `list(m.power_plants * m.seasons)` in a new cell to see what this looks like.

The `mutable=True` argument allows for changes to the parameter data after its definition. Later in this assignment we will change the existing power plant capacity and the CO2 price. Because of this, we have to define the parameters as mutable right from the start.

Please make sure you understand why each parameter is defined the way it is (why it depends on its corresponding sets). You can inspect parameters by executing `m.demand.display()` in a new cell.

[8]:
```
m.price_co2 = po.Param(initialize=5, mutable=True)

m.season_weight = po.Param(m.seasons, initialize=season_weight)
```

```
m.eff = po.Param(m.power_plants, initialize=eff)
m.co2_intensity = po.Param(m.power_plants, initialize=co2_intensity)
m.capacity_old = po.Param(m.power_plants, initialize=capacity_old, mutable=True)
m.fuel_cost = po.Param(m.power_plants, initialize=fuel_cost)
m.fixed_cost = po.Param(m.new_power_plants, initialize=fixed_cost)
m.cf = po.Param(m.power_plants * m.seasons, initialize=cf, default=1)
m.demand = po.Param(m.seasons, initialize=demand)
```

1.9.3 Variables

In this simple model we optimize two types of variables:

- the power output from all power plants during each season
- the added capacity for wind, solar and new natural gas plants

All variables must be greater than zero: `bounds=(0, None)`.

All variables must be real numbers (not integers or boolean values, i.e. 0/1). This is the default in Pyomo, so we don't need to include it explicitly.

[9]:
```
m.pwr = po.Var(m.power_plants * m.seasons, bounds=(0, None))
m.cap_new = po.Var(m.new_power_plants, bounds=(0, None))
```

1.9.4 Objective function

The objective function (i.e. what we want to minimize, in this case) is the sum of all yearly costs of the system.

The Python summation over all items in a set "`sum( ... for item in set)`" does exactly what you would expect it to do if you read it out loud.

[10]:
```
total_variable_cost = sum(m.season_weight[season] * m.pwr[plant, season]
                          * (m.fuel_cost[plant] + m.price_co2 * m.
 ↪co2_intensity[plant])
                          / m.eff[plant] for plant in m.power_plants for season␣
 ↪in m.seasons)

total_fixed_cost = sum(m.cap_new[plant] * m.fixed_cost[plant] for plant in m.
 ↪new_power_plants)

m.obj = po.Objective(expr=total_variable_cost + total_fixed_cost, sense=po.
 ↪minimize)
```

Exercise 1: Understanding the objective function

Task 1.a (Discussion): Based on the code above, your own reasoning, and the discussion in the lecture, explain the total cost in detail: * What are the units of each factor

and term? How do they add up to yield a total cost? Explain in words. What is the unit of the total cost? * What role does the efficiency (`eff`) play?

Exercise 2: Inspection of the input data.

Understanding the input data is crucial to interpret the model results. This is the longest exercise of this assignment.

The code in the next cell produces the tables with the necessary input data for the following tasks.

Task 2.a (Analysis/Discussion): Calculate the total variable cost (fuels and emissions) per produced MWh for each of the power plants (units EUR/MWhelectricity), i.e. how much does it cost to produce one MWh of electricity using each of the plants? Report the values and describe your reasoning *in words*. **Task 2.b (Analysis/Discussion)**: Calculate the levelized cost of new wind and solar plants per produced MWh. Report the values and describe your reasoning *in words*. **Task 2.c (Analysis/Discussion)**: Which price on CO2 emissions would be necessary to make new wind and solar plants competitive with the existing coal plants, only considering levelized costs of the former and total variable costs of the latter? Report the values and describe your reasoning *in words*. Hint: Competitive means that the levelized costs are equal. Important: Find the equation to calculate this, don't just try different values. **Task 2.d (Analysis/Discussion)**: Which price on CO2 emissions would be necessary to make new gas power plants competitive with respect to coal power plants and old gas power plants? Assume that the new gas plants produce electricity at full power all year round. What changes if they only produce power during 1, 2, or 3 seasons? Report all values for 1-4 seasons. In your own words, describe what it means that the competitiveness of the new plants depends on the number of seasons during which they produce power. Hint: The total cost of the new gas plants consists of fuel, CO2 prices, and the investment cost; use the same approach as in Task 2.c. **Task 2.e (Analysis/Discussion)**: Plot the capacity factors of the new wind and new solar plants. What do you notice when comparing the two curves qualitatively? Calculate the *yearly* capacity factor of both new wind and new solar from the seasonal capacity factors provided to you. How can you calculate the capacity factor in the more general case where not all seasons have the same length? Describe your reasoning *in words*. Hint: Remember how the capacity factor was defined in the lecture.

Important: The cost comparisons and the capacity factors will be important later when we analyze the results of the optimization model. Please make sure you finish this exercise before proceeding.

[11]:
```
"""Generates output tables to be copied to Excel."""

print('\n\n**Variable costs**\n')
df = pd.concat(
[get_data(m.co2_intensity, ['plant']).set_index('plant').value.
 ↪rename('co2_intensity'),
 get_data(m.eff, ['plant']).set_index('plant').value.rename('eff'),
 get_data(m.fuel_cost, ['plant']).set_index('plant').value.rename('fuel_cost'),
```

```
 ], axis=1, sort=True).assign(price_co2=get_data(m.price_co2, []).value.
 ↪values[0])
display(HTML(df.reset_index().rename(columns={'index': 'plant'}).
 ↪to_html(index=False)))

print('\n\n**Capacity factors**\n')
df = get_data(m.cf, ['plant', 'season']).pivot_table(index='plant',␣
 ↪columns='season', values='value')
df.columns = df.columns.rename(None)
display(HTML(df.reset_index().to_html(index=False)))

print('\n\n**Annualized fixed costs**\n')
df = get_data(m.fixed_cost, ['plant']).set_index('plant').
 ↪rename(columns={'value': 'fixed_cost'})
display(HTML(df.reset_index().to_html(index=False)))
```

```
**Variable costs**
```

```
<IPython.core.display.HTML object>
```

```
**Capacity factors**
```

```
<IPython.core.display.HTML object>
```

```
**Annualized fixed costs**
```

```
<IPython.core.display.HTML object>
```

1.9.5 Constraints

Back to the optimization model: So far we built a functioning model. However it doesn't do anything useful yet. Try to run it in a new cell: `solution = solver.solve(m)`. You can check whether the run was successful by having a look at the "termination condition", which should be `'optimal': solution.solver.termination_condition.key`. If you inspect the power variables (`m.pwr.display()`) and the objective (`m.obj.display()`) you will notice that they are all zero or `None`. This because we haven't included any constraints yet to define how much power must be produced during each season.

The demand of the system (power, MW) must be satisfied through electricity generated by the power plants. This can be formulated as a constraint for each of the seasons in the model. In words: **For each season, the sum of the power generation from the power plants (`m.pwr`) must be equal or larger the demand (`m.demand`).** As already discussed in the lecture, this is expressed as shown below:

```
def demand_constraint_equation(m, season):
    return sum(m.pwr[pp, season] for pp in m.power_plants) >= m.demand[season]
m.demand_constraint = po.Constraint(m.seasons, rule=demand_constraint_equation)
```

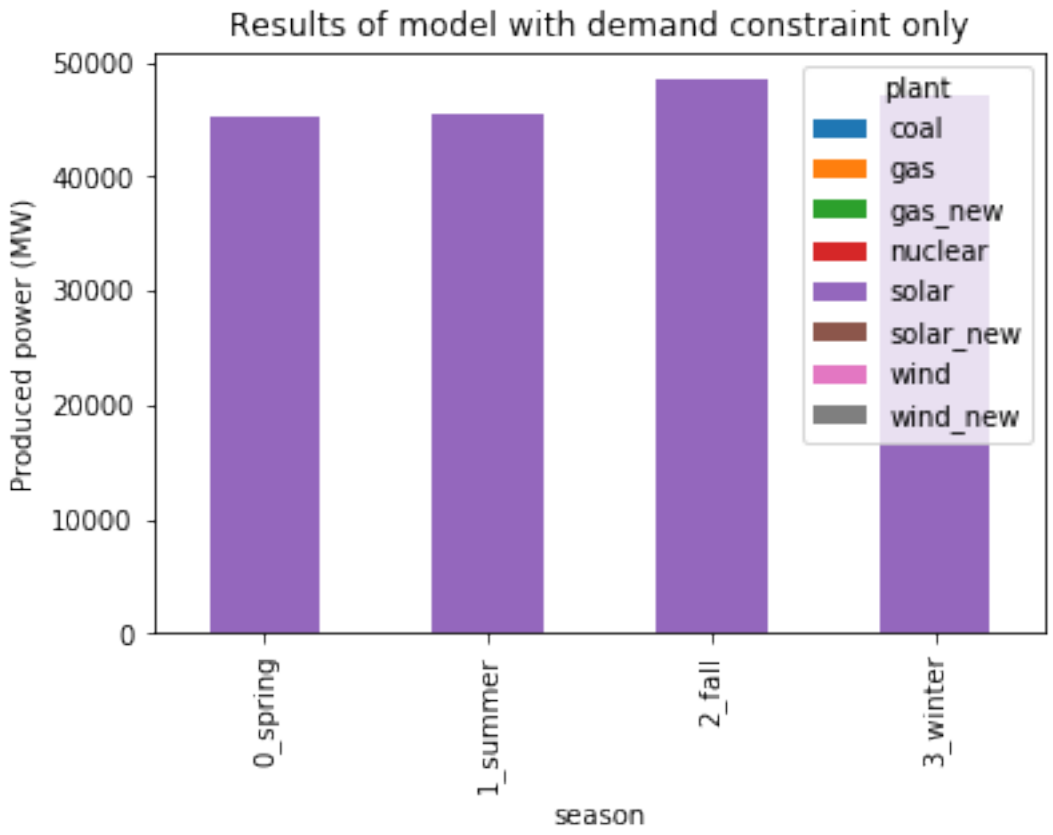
The actual equation (*sum of power greater/equal demand*) is written in a separate function (`def demand_constraint_equation(m, ...)`). Then it is used to define the actual model constraint `po.Constraint(...)`. Note the first parameter `m.seasons` in `po.Constraint(...`: It expresses the **For each season ...** part of the constraint formulation.

[12]:
```
def demand_constraint_equation(m, season):
    return sum(m.pwr[pp, season] for pp in m.power_plants) >= m.demand[season]

m.demand_constraint = po.Constraint(m.seasons, rule=demand_constraint_equation)
```

Then we run the model (`solution = solver.solve(m)` in the cell below). As you can see in the plot, only solar power is used to cover the demand. Currently, there are no limitations on the availability of this resource included in the model: Solar power can be used to provide unlimited amounts of zero-cost electricity. Note that the same would hold for any of the other wind and solar plants, or any combination thereof. In fact, an unlimited number of solutions exists to the model, all with total cost zero. If we increased the "fuel cost" of solar from 0 to something very small (like 1e-8), the solver would pick any of the other wind or solar plants. In these cases the solution is random, or depends on the internal workings of the solver.

[13]:
```
solution = solver.solve(m)

print('~'*30)
print('Value of objective function/total cost:')
m.obj.display()
print('~'*30)

ax = get_data(m.pwr, ['plant', 'season']).pivot_table(index='season',␣
 ↪columns='plant', values='value').plot.bar(stacked=True);
ax.set_ylabel('Produced power (MW)');
ax.set_title('Results of model with demand constraint only');
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Value of objective function/total cost:
obj : Size=1, Index=None, Active=True
    Key  : Active : Value
    None :   True :   0.0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```



1.9.6 Other constraints

Now you know everything to define the rest of the necessary constraints.

Note 1: You need two additional constraints to make this a meaningful model. Have a look at the lecture slides for inspiration. **Note 2:** Don't make any assumptions on the parameter values: Include the `m.capacity_old` parameter everywhere, even if it is zero for new power plants. **Note 2:** It should help to first formulate the constraints in words, starting with **For each ...** (as above) **Note 3:** If your constraint description starts with **For each power plant and each season ...** you can use the combined set `m.power_plants * m.seasons` (just like for the `m.cf` parameter). In this case the equation function would take three arguments: `def ..._equation(m, plant, season)`, as compared to the two in the case above: `def demand_constraint_equation(m, season)`. **Note 4:** The `m.cap_new` variable is defined for new plants only. You can make a constraint behave differently depending on whether or not it is being defined for a new power plant by checking as follows:

```
def ..._equation(m, plant, ...):
    if plant in m.new_power_plants:
        # do something
    else:
        # do something else
```

Note 5: This is the only exercise of this assignment where you have to work with Python code (except for the BONUS exercise). However, this needs to be correct, since the whole rest of the exercise depends on it. Please ask to get this checked.

Exercise 3: Adding the additional constraints.

Task 3.a (Discussion): Describe the additional two constraints *in your own words*. Explain: Why is the constraint needed? What does it enforce?

BONUS Task 3.b (Coding): Add the two additional constraints to the model, in the same way as the supply_constraint above. Please copy the code to the report. This is a bonus task. If you want to skip it, please ask for the code in order to proceed with the assignment.

[14]:
```
# <add additional constraints here>

def capacity_constraint_equation(m, plant, season):
    if plant in m.new_power_plants:
        return m.pwr[plant, season] <= m.capacity_old[plant] + m.cap_new[plant]
    else:
        return m.pwr[plant, season] <= m.capacity_old[plant]

m.capacity_constraint = po.Constraint(m.power_plants * m.seasons,␣
 ↪rule=capacity_constraint_equation)

def cf_constraint_equation(m, plant, season):
    if plant in m.new_power_plants:
        return m.pwr[plant, season] == m.cf[plant, season] * (m.
 ↪capacity_old[plant] + m.cap_new[plant])
    else:
        return m.pwr[plant, season] == m.cf[plant, season] * (m.
 ↪capacity_old[plant])

m.cf_constraint = po.Constraint(m.wind_solar, m.seasons,␣
 ↪rule=cf_constraint_equation)
```

Now we can run the model in the cell below. The total yearly cost (objective function `m.obj.display()`) should be 5514059936.625001. If it is different, something is wrong.

[15]:
```
solution = solver.solve(m, tee=True)

print('~' * 60)
print('Objective value: ')
m.obj.display()
print('~' * 60)
```

```
GLPSOL: GLPK LP/MIP Solver, v4.65
Parameter(s) specified in the command line:
 --write /tmp/tmpindhteaq.glpk.raw --wglp /tmp/tmp9hj1d2yd.glpk.glp --cpxlp
```

```
 /tmp/tmpu277hs21.pyomo.lp
Reading problem data from '/tmp/tmpu277hs21.pyomo.lp'…
53 rows, 36 columns, 101 non-zeros
322 lines were read
Writing problem data to '/tmp/tmp9hj1d2yd.glpk.glp'…
257 lines were written
GLPK Simplex Optimizer, v4.65
53 rows, 36 columns, 101 non-zeros
Preprocessing…
24 rows, 27 columns, 64 non-zeros
Scaling…
 A: min|aij| =  5.100e-02  max|aij| =  1.000e+00  ratio =  1.961e+01
GM: min|aij| =  4.752e-01  max|aij| =  2.104e+00  ratio =  4.428e+00
EQ: min|aij| =  2.258e-01  max|aij| =  1.000e+00  ratio =  4.428e+00
Constructing initial basis…
Size of triangular part is 24
      0: obj =   0.000000000e+00 inf =   1.769e+05 (4)
      6: obj =   2.570162315e+10 inf =   0.000e+00 (0)
*    15: obj =   5.514059937e+09 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.1 Mb (75077 bytes)
Writing basic solution to '/tmp/tmpindhteaq.glpk.raw'…
98 lines were written
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Objective value:
obj : Size=1, Index=None, Active=True
    Key  : Active : Value
    None :   True : 5514059936.625001
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Exercise 4: Analyzing the model results.

The code below produces the tables with the necessary data for the following tasks. Please copy the tables to Excel and perform the analysis there.

Task 4.a (Analysis): Draw a stacked bar plot which shows the average produced power for each season and each power plant (one plot, four bars (seasons), stacked by power plant). **Task 4.b (Analysis)**: Calculate the total yearly capacity factor of each plant and plot them as a single bar plot (one bar for each plant). **Task 4.c (Discussion)**: Compare the capacity factors of the nuclear, hard coal and gas plants to the respective variable costs of these plants (task 2.a). Given the calculated costs, explain why some plants have higher capacity factors than others. **Task 4.d (Discussion)**: Are any new wind, solar, or gas plants added to the system? Explain why or why not for each of the three plants. Take into account the results from task 2.c and 2.d. Which price on CO2 emissions would be required to make them competitive? How does the price assumed in the model compare to this? (Note: If you don't remember the CO2 price currently used by the model, you can either scroll back up, or execute `m.price_co2.display()` in a new cell)

[16]:
```
"""Generates output tables to be copied to Excel."""
print('\n\n**Produced power**\n')
df = get_data(m.pwr, ['plant', 'season']).pivot_table(index='season',␣
 ↪columns='plant', values='value')
df.columns = df.columns.rename(None)
display(HTML(df.reset_index().to_html(index=False)))

print('\n\n**Old capacity**\n')
df = get_data(m.capacity_old, ['plant']).set_index('plant').T[df.columns]
display(HTML(df.to_html()))

print('\n\n**New capacity**\n')
df = get_data(m.cap_new, ['plant']).set_index('plant').T
display(HTML(df.to_html()))
```

Produced power

<IPython.core.display.HTML object>

Old capacity

<IPython.core.display.HTML object>

New capacity

<IPython.core.display.HTML object>

1.10 Modeling an energy transition

Next, we analyze a transition to a nuclear-free and cleaner power system by assuming the phaseout of nuclear power and a strong reduction (-50%) of the coal power capacity. We can change the parameters of the existing model directly, as in the next cell.

[17]:
```
m.capacity_old['nuclear'] = 0
m.capacity_old['coal'] = capacity_old['coal'] * 0.5
```

While there is still sufficient capacity in the system to cover all demand, the reduction of the cheap nuclear and coal power plants is a strong driver for the installation of new power plants.

The optimal choice of which new plants (gas or wind or solar) and how much capacity to install, strongly depends on the CO2 emission price: A higher price on CO2 makes less efficient plants and plants using fuels with high carbon intensity more expensive.

With the model runs in the next 2 cells we compare the impact of a switch from 14 to 15 EUR/t of CO2.

[18]:
```
m.price_co2 = 14
solution = solver.solve(m)

df = get_data(m.pwr, ['plant', 'season']).pivot_table(index='season',␣
 ↪columns='plant', values='value')
df.columns = df.columns.rename(None)
print('\n\n**Produced power at 14 EUR/t_CO2**\n')
display(HTML(df.reset_index().to_html(index=False)))

print('\n\n**New installed capacity at 14 EUR/t_CO2**\n')
display(HTML(get_data(m.cap_new, ['plant']).to_html(index=False)))
```

Produced power at 14 EUR/t_CO2

<IPython.core.display.HTML object>

New installed capacity at 14 EUR/t_CO2

<IPython.core.display.HTML object>

[19]:
```
m.price_co2 = 15
solution = solver.solve(m)
df = get_data(m.pwr, ['plant', 'season']).pivot_table(index='season',␣
 ↪columns='plant', values='value')
df.columns = df.columns.rename(None)
print('\n\n** Produced power at 15 EUR/t_CO2 **\n')
display(HTML(df.reset_index().to_html(index=False)))

print('\n\n** New installed capacity at 15 EUR/t_CO2 **\n')
display(HTML(get_data(m.cap_new, ['plant']).to_html(index=False)))
```

** Produced power at 15 EUR/t_CO2 **

```
<IPython.core.display.HTML object>
```

```
** New installed capacity at 15 EUR/t_CO2 **
```

```
<IPython.core.display.HTML object>
```

> ***Exercise 5:*** Comparison of the 14 and 15 EUR/tCO2 model runs
>
> **Task 5.a (Analysis)**: Compare the results (produced average power from all plants) of the models in 2 stacked bar plots (two plots, four bars (seasons) in each plot, stacked by power plant)
>
> **Task 5.b (Discussion)**: What are the differences in terms of power production between the cases with 14 and 15 EUR/tCO2? Why is it beneficial to avoid production from old gas plants by building new ones? In the 14 EUR/t case, what limits the investment in new gas plants compared to the 15 EUR/t case? Use the results from task 2.d. to answer these questions.

1.10.1 Variation of the CO2 price

Next, we have a more general look at the CO2 price impact: We vary it for 11 values between 0 and 200 EUR/t in 20 EUR/t steps.

The data from the different model runs are saved in the tables `table_pwr` and `table_cap` and are displayed after the model runs.

In the code below, please read the comments (`# ...`) to understand what we are doing there.

[ ]:
```python
# Initialize the empty output data tables that we append to in the loop.
table_pwr = pd.DataFrame(columns=['plant', 'season', 'price_co2', 'value'])
table_cap = pd.DataFrame(columns=['plant', 'price_co2', 'value'])

# Loop over the CO2 prices.
for prco2 in range(0, 201, 20): # this gives [0, 20, 40, ..., 200]

    # Set the CO2 price parameter value. This changes the model parameter value
    # and therefore also the expression in the objective function!
    m.price_co2 = prco2

    # Perform the model run.
    solution = solver.solve(m)

    # Print some output to check whether the solution is optimal.
    print(f'{prco2:3d}: Termination condition = {solution.solver.
 ↪termination_condition.key}')
```

```
    # Append the data to the tables.
    table_pwr = table_pwr.append(get_data(m.pwr, ['plant', 'season']).
 ↪assign(price_co2=prco2), sort=True)
    table_cap = table_cap.append(get_data(m.cap_new, ['plant']).
 ↪assign(price_co2=prco2), sort=True)

# Print the output table as usual.
print('\n\n** Newly installed capacity **\n')
df = table_cap.sort_values(['price_co2', 'plant']).reset_index(drop=True)
display(HTML(df.to_html(index=False)))

print('\n\n** Produced power **\n')
df = table_pwr.sort_values(['price_co2', 'plant', 'season']).
 ↪reset_index(drop=True)
display(HTML(df.to_html(index=False)))
```

Exercise 6: Optimal electricity mix for changing CO2 prices.

This analysis is based on the data from the last model runs printed above.

Task 6.a (Analysis): Draw a stacked bar chart of the total produced energy (sum over all seasons) in **TWh/yr** as a function of the CO2 emission price (Consider: What are the units of the output data? How do you obtain TWh/yr from that?). Tip: Trends are easier to see if you set the bar gap width to 0%.

The following tasks are largely based on the comparison with results from exercise 2. **Task 6.b (Discussion)**: An abrupt transition in the optimal system configuration occurs for a certain CO2 price range, first to much more new gas power, then to much more wind power. Does this transition occur roughly where you would expect it? Explain in words. Take into account the results from exercise 2. **Task 6.c (Discussion)**: Once the installation of new wind and solar capacity becomes more beneficial than the installation of new gas plants, it is initially only wind power that's being installed. Why is wind more beneficial than solar power? Take into account the results from exercise 2. **BONUS Task 6.d**: Qualitatively, what drives the increasingly important role of solar power starting from a certain CO2 price? Hint: Get back to this question after working on Exercise 8.

1.10.2 Electricity price/shadow price of demand

We now look at the shadow prices of the demand constraint (the "electricity prices" of the model). By definition, we could obtain the shadow price of the demand constraint by adding a single unit of demand in each season separately, and then compare the total cost (see lecture slides). For simplicity, we extract the electricity prices directly from the model. **Since the demand constraint is defined in units of MW, the unit of a "additional cost due to a unit of demand increase" is EUR/MW. Please keep this in mind in order to obtain the correct units in the task below.**

Note that this is a subset of the model runs analyzed in Exercise 6 (for 3 selected CO2 emission prices). We run them again in order to extract the required output data.

In the code below, please read the comments (`# ...`) to understand what we are doing there.

[21]:
```
# Initialize empty tables for the model results
table_pwr = pd.DataFrame(columns=['plant', 'season', 'price_co2', 'value'])
table_cap = pd.DataFrame(columns=['plant', 'price_co2', 'value'])
table_price = pd.DataFrame(columns=['season', 'price_co2', 'value'])

# Loop over the selected CO2 prices
for prco2 in [20, 80, 200]:

    # Set the CO2 price
    m.price_co2 = prco2

    # Solve the model
    solver.solve(m)

    # Append the model results to the tables
    table_pwr = table_pwr.append(get_data(m.pwr, ['plant', 'season']).
 ↪assign(price_co2=prco2), sort=True)
    table_cap = table_cap.append(get_data(m.cap_new, ['plant']).
 ↪assign(price_co2=prco2), sort=True)
    table_price = table_price.append(get_data(m.demand_constraint, ['season']).
 ↪assign(price_co2=prco2), sort=True)

print('\n\n** Demand constraint shadow prices **\n')
df = table_price.pivot_table(index='price_co2', columns='season',␣
 ↪values='value').reset_index()
df.columns = df.columns.rename(None)
display(HTML(df.to_html(index=False)))

print('\n\n** Power production (sum over all seasons) **\n')
df = table_pwr.pivot_table(index=['plant'], columns='price_co2',␣
 ↪values='value', aggfunc=sum).reset_index()
df.columns = df.columns.rename(None)
display(HTML(df.to_html(index=False)))

print('\n\n** New installed capacity of gas power plants **\n')
df = table_cap.query('plant == "gas_new"')
display(HTML(df.to_html(index=False)))
```

```
** Demand constraint shadow prices **
```

```
<IPython.core.display.HTML object>
```

```
** Power production (sum over all seasons) **
```

```
<IPython.core.display.HTML object>
```

```
** New installed capacity of gas power plants **
```

```
<IPython.core.display.HTML object>
```

> ***Exercise 7:*** Electricity/Shadow prices.
>
> **Task 7.a (Analysis)**: Plot the electricity prices (EUR/**MWh**) as a function of the season (line plot). Include all emission prices in the same plot. **Task 7.b (Discussion)**: Discuss the qualitative change of the price profile for increasing power prices. Why do the electricity prices change only rather little for a four-fold increase of emission prices from 20 to 80 EUR/MWh? What could be the reason for the strongly different 200 EUR/tCO2 profile? **Hint**: Try to adjust the CO2 prices in task 2.a and compare the levelized costs to the electricity prices.

1.10.3 Variation of wind capacity

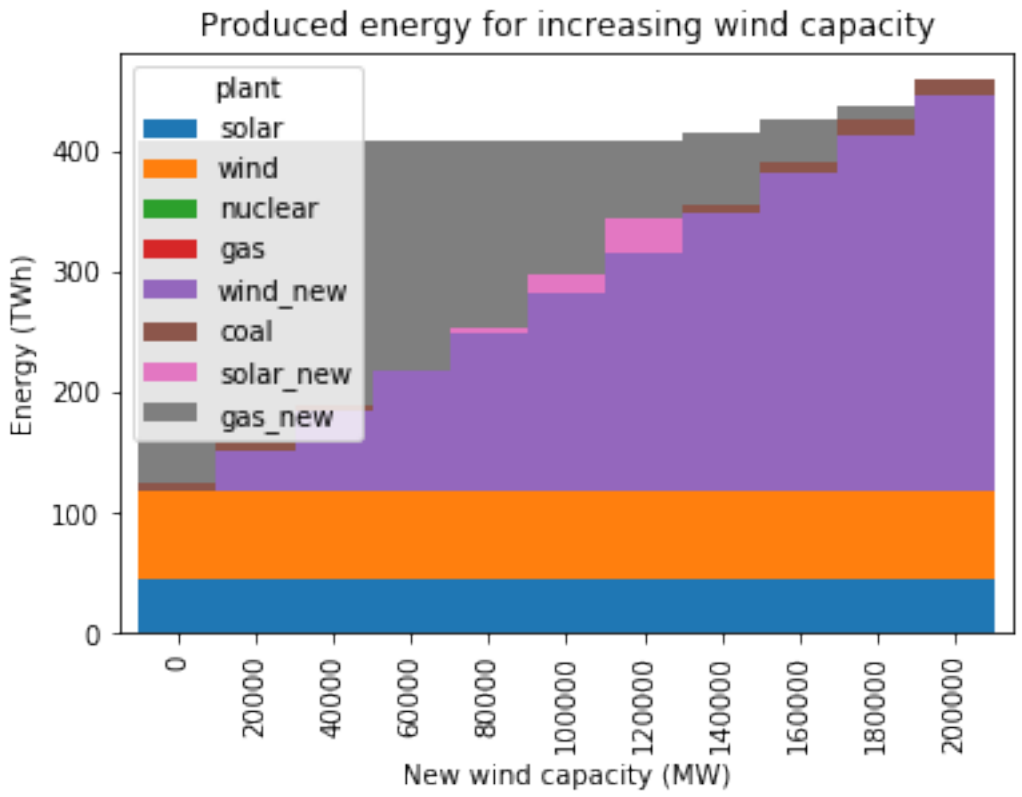
In the lecture we discussed that the net value of optimized capacity in the linear model is always zero. In the last model runs used above, we could have shown that the revenue (produced energy times electricity prices) is exactly equal the total fixed and variable cost of new gas, new wind, and new solar plants.

In this part of the assignment, we have a deeper look at this by considering the case where wind turbine capacity is forced into the power system. Then we analyze how the value of wind power changes for increasing wind turbine capacity, i.e. if we go from zero to capacities much higher than the optimum. Before starting, try to guess what happens as we increase the wind capacity: * to the revenue of wind power * to the total system cost

Please have a look at the code below and especially read the comments (`# ...`) to understand what we are doing here.

[22]:
```
# We now change the capacity of new wind power manually, instead of optimizing␣
 ↪it.
# Therefore, we need to "fix" the variable so the solver cannot change its␣
 ↪value.
m.cap_new['wind_new'].value = 0
m.cap_new['wind_new'].fix()
```

```
# We use a constant CO2 price.
m.price_co2 = 120

# Initialize empty tables for the results
table_pwr = pd.DataFrame(columns=['plant', 'season', 'cap_wind', 'value'])
table_price = pd.DataFrame(columns=['season', 'cap_wind', 'value'])
table_obj = pd.DataFrame(columns=['cap_wind', 'value'])

# Loop over the wind capacity from 0 to 200'000 MW in 20'000 MW steps:
for cap_wind in range(0, 200001, 20000):

    # set the wind capacity model parameter
    m.cap_new['wind_new'].value = cap_wind

    # solve the model
    solver.solve(m)

    # append the results to the tables
    table_pwr = table_pwr.append(get_data(m.pwr, ['plant', 'season']).
 ↪assign(cap_wind=cap_wind), sort=True)
    table_price = table_price.append(get_data(m.demand_constraint, ['season']).
 ↪assign(cap_wind=cap_wind), sort=True)
    table_obj = table_obj.append(get_data(m.obj, []).
 ↪assign(cap_wind=int(cap_wind)), sort=True)

# plotting the produced energy
data_plot = (2190 / 1e6 * table_pwr.pivot_table(index='cap_wind',␣
 ↪columns='plant', values='value', aggfunc=sum))
ordered_columns = ['solar', 'wind', 'nuclear', 'gas', 'wind_new', 'coal',␣
 ↪'solar_new', 'gas_new']
ax = data_plot[ordered_columns].plot.bar(stacked=True, width=1, title='Produced␣
 ↪energy for increasing wind capacity')
ax.set_ylabel('Energy (TWh)'); ax.set_xlabel('New wind capacity (MW)');
```



Exercise 8: Specific value of wind power forced into the power system.

The code in the cell below prints the required data for this exercise. Note that we are only interested in the newly added wind turbine capacity ("wind_new"). The old wind turbines are fixed and just a part of the overall system. Don't consider them.

Task 8.a (Analysis): Using the shadow price of demand/electricity prices (EUR/MW) from the table printed below and the capacity factors from Exercise 2, calculate the specific revenue of the new wind turbines (revenue per installed capacity, EUR/MW/year, single line plot with markers). Plot the specific wind turbine revenue as a function of the installed new wind capacity. Also plot the shadow prices for the 4 seasons separately (all in the same plot) as a function of the new wind capacity (single bar plot with 11 bar groups, not stacked). **Task 8.b (Analysis)**: Plot the total cost (objective function) as a function of the installed wind turbine capacity (line plot, units: billion (1e9) EUR/year). **Task 8.c (Discussion)**: Describe the plotted total cost (task 8.b). What is going on here? Does the minimum cost occur where you would expect it? Explain using the results of exercise 6. **Task 8.d (Discussion)**: What do you observe in the plotted specific revenue (task 8.a)? Explain what's going on (steps!) by comparing to the plotted electricity prices (task 8.a). What is the specific revenue of wind turbined around the optimum from task 8.b? Compare to the annualized investment cost of wind turbines. Describe your conclusions in your own words. **Task 8.e (Discussion)**: So far we have only considered *new* wind turbine capacity. What is the specific value (EUR/MW/year) of the old wind turbine plants? Explain your reasoning.

(No additional calculations required! Think how the calculation would change if you did it explicitly.) **Task 8.f (Discussion)**: What do these results mean in a real power system? Can you think of how this change in specific revenue might be problematic?

```
[ ]: print('\n\n** Electricity prices for each season and all model runs. **\n')
     display(HTML(table_price.to_html(index=False)))

     print('\n\n** Total objective function value. **\n')
     display(HTML(table_obj.to_html(index=False)))
```

1.10.4 BONUS EXERCISE: Does storage add value to wind power?

Now we want to repeat the last exercise and assess the impact of large amounts of pumped hydro storage in the system. Using storage we can transfer energy from one time slot to another. This generally results in a "smoothing" of the electricity prices, i.e. high prices get lower, low prices get higher, price differences get smaller. This is because storage charging creates additional demand during certain seasons. This demand needs to be satisifed using more expensive generators. During the discharging seasons, the demand is lowered, causing electricity prices to drop.

Here we want to investigate the impact of this effect on the value of wind power.

To keep things simple, we assume that the storage has a limited power capacity (10'000 MW) but an unconstrained energy capacity. This might correspond to pumped hydro plants with a certain turbine size but whose lakes are large enough to satisfy all our storage needs.

IN THE JUPYTER NOTEBOOK:

1. We keep the old model and adjust some parameters and constraints.
2. Define the following variables and parameters which will define the storage asset (**note**: Since there is only one storage plant, we just use single value parameters, i.e. we don't define sets for storage):
   - **Parameters**:
     - Storage round trip efficiency (fraction of the charged energy which is available after discharging): `m.storage_eff = po.Param(initialize=0.75)`
     - Storage power capacity: `m.storage_cap = po.Param(initialize=10000, mutable=True)`
   - **Variables**: Here we can use the storage power capacity as an upper bound to the charging and discharging power, instead of defining an extra constraint. Like this we can deactivate the storage by setting the capacity parameter zero (`m.storage_cap = 0`):
     - Storage charging/pumping during each season: `m.charging = po.Var(m.seasons, bounds=(0, m.storage_cap))`
     - Storage discharging/turbining during each season: `m.discharging = po.Var(m.seasons, bounds=(0, m.storage_cap))`

- Stored energy during each season: `m.stored_energy = po.Var(m.seasons, bounds=(0, None))`

3. Modify the constraints of the original model to include the additional storage operation. **Note**: If you add terms to the demand constraint equation, make sure that the `m.demand[season]` is the only term on the right-hand side. Otherwise the electricity prices will be negative.

4. Add the single additional constraint which is necessary to describe the storage operation. Have a look at the lecture slides. Tip: to access the "previous season" you can use a dictionary like this:

   ```
   previous_season = {'3_winter': '2_fall', '0_spring': '3_winter', '1_summer': '0_spring', '
   ```

   and then `m.stored_energy[previous_season[season]]`

5. Major parts of the required code are already implemented in the cell below. Add the relevant completions where it says so.

Exercise 10: Impact of storage on wind turbine revenue.

Task 10.a (Discussion): Describe the changes to the original model constraints when including storage (in words). Describe the reasoning behind the additional storage constraint (in words). **Task 10.b (Analysis)**: Plot the charging and discharging power (MW, data printed below) for 60'000 and 140'000 MW of new wind power (one plot, four bars). **Task 10.c (Analysis)**: Like in Task 8.a, calculate the specific revenue of wind power (EUR/MW/year) with and without storage and plot the two curves in a single plot (as function of the wind power capacity). **Task 10.d (Analysis)**: Plot the electricity prices (EUR/MWh) as a function of the season for 140'000 MW of new wind power capacity with and without storage (plot series in one plot). **Task 10.e (Discussion)**: Is the wind power revenue larger or smaller if there is storage in the system? Why? Explain by starting from the impact of wind power on prices, then the impact of storage on prices, then the impact of prices on wind power revenue. Base your discussion on plot 9.c and the plot of the wind power capacity factors from task 1.e. What do you think happens at the highest wind power capacity?

[ ]:
```
#####################################################################
# < ADD HERE: new storage parameters and variables >

m.storage_eff = po.Param(initialize=0.75)
m.storage_cap = po.Param(initialize=10000, mutable=True)

m.charging = po.Var(m.seasons, bounds=(0, m.storage_cap))
m.discharging = po.Var(m.seasons, bounds=(0, m.storage_cap))
m.stored_energy = po.Var(m.seasons, bounds=(0, None))

#####################################################################

# deleting the demand constraint so we can add the modified one below
```

```
m.del_component('demand_constraint')

#####################################################################
# < ADD HERE: modified/original constraints of the old model >

def demand_constraint_equation(m, season):
    return sum(m.pwr[pp, season] for pp in m.power_plants) + m.
 ↪discharging[season] - m.charging[season] >= m.demand[season]
m.demand_constraint = po.Constraint(m.seasons, rule=demand_constraint_equation)

#####################################################################


########################################
# < ADD HERE: new storage constraint >

previous_season = {'3_winter': '2_fall',
                   '0_spring': '3_winter',
                   '1_summer': '0_spring',
                   '2_fall': '1_summer'}

def storage_constraint_equation(m, season):
    return m.stored_energy[previous_season[season]] == m.stored_energy[season]␣
 ↪+ m.storage_eff * m.charging[season] - m.discharging[season]
m.storage_constraint = po.Constraint(m.seasons,␣
 ↪rule=storage_constraint_equation)

########################################

# intiialize output tables
table_pwr = pd.DataFrame(columns=['plant', 'season', 'cap_wind', 'value'])
table_price = pd.DataFrame(columns=['season', 'cap_wind', 'value'])
table_charging = pd.DataFrame(columns=['season', 'cap_wind', 'value'])
table_discharging = pd.DataFrame(columns=['season', 'cap_wind', 'value'])

# Loop over the wind capacity from 0 to 200'000 MW in 20'000 MW steps:
for cap_wind in range(0, 200001, 20000):

    for storage_cap in (0, 10000):

        ␣
 ↪#############################################################################################
        # ADD HERE: < modify model parameters (m.capacity_old and m.
 ↪storage_cap) and run model >

        m.cap_new['wind_new'] = cap_wind
        m.storage_cap = storage_cap
```

```
        solution = solver.solve(m)

    ␣
↪#############################################################################################

        table_pwr = table_pwr.append(get_data(m.pwr, ['plant', 'season']).
↪assign(cap_wind=cap_wind, storage_cap=storage_cap), sort=True)
        table_price = table_price.append(get_data(m.demand_constraint,␣
↪['season']).assign(cap_wind=cap_wind, storage_cap=storage_cap), sort=True)
        table_charging = table_charging.append(get_data(m.charging, ['season']).
↪assign(cap_wind=cap_wind, storage_cap=storage_cap), sort=True)
        table_discharging = table_discharging.append(get_data(m.discharging,␣
↪['season']).assign(cap_wind=cap_wind, storage_cap=storage_cap), sort=True)

print('\n\n**Charging power (MW) for 200\'000 and 160\'000 MW of wind turbine␣
↪capacity**\n')
df = table_charging.query('cap_wind in [200000, 140000] and storage_cap ==␣
↪"10000"').sort_values(['cap_wind', 'season'])
display(HTML(df.to_html(index=False)))
print('\n\n**Discharging power (MW) for 200\'000 and 140\'000 MW of wind␣
↪turbine capacity**\n')
df = table_discharging.query('cap_wind in [200000, 140000] and storage_cap ==␣
↪"10000"').sort_values(['cap_wind', 'season'])
display(HTML(df.to_html(index=False)))

print('\n\n**Shadow prices in EUR/MW**\n')
df = table_price
display(HTML(df.to_html(index=False)))
```

[ ]:
```
ordered_columns = ['solar', 'wind', 'nuclear', 'gas', 'wind_new', 'coal',␣
↪'solar_new', 'gas_new']
data_plot = (2190 / 1e6 *
table_pwr.query('storage_cap == 0').pivot_table(index='cap_wind',␣
↪columns='plant', values='value', aggfunc=sum)
)
display(data_plot)
ax = data_plot[ordered_columns].plot.bar(stacked=True, width=1)
ax.set_ylabel('Energy (TWh)'); ax.set_title('Produced energy for increasing␣
↪wind turbine capacity');
```