MUSE: ModUlar energy system Simulation Environment Documentation

Release 0.8

Sustainable Gas Institute

Nov 10, 2020

CONTENTS:

CHAPTER

ONE

INSTALLATION

There are two ways to install MUSE: one for users who do not wish to modify the source code of MUSE, and another for developers who do.

Note: Windows users and developers may need to install Windows Build Tools. These tools include C/C++ compilers which are needed to build some python dependencies.

MacOS includes compilers by default, hence no action is needed for Mac users.

Linux users may need to install a C compiler, whether GNU gcc or Clang, as well python development packages, depending on their distribution.

1. Download Microsoft Visual C++ Build Tools from this link: https://visualstudio.microsoft.com/downloads/
2. Select your preferred edition. The "Community" is free and contains what is required.
3. Run the installer
4. Select: Workloads → Visual C++ build tools.
5. Install options: select only the "Windows 10 SDK" (assuming the computer is Windows 10)]

For further information, see this link: https://www.scivision.dev/python-windows-visual-c-14-required

1.1 For users

MUSE is developed using python, an open-source programming language, which means that there are two steps to the installation process. First, python should be installed. Then so should MUSE.

The simplest method to install python is by downloading the Anaconda distribution. Make sure to choose the appropriate operating system (e.g. windows), python version 3.7, and the 64 bit installer. Once this has been done follow the steps for the anaconda installer, as prompted.

After python is installed we can install MUSE. MUSE can be installed via the Anaconda Prompt (or any terminal on Mac and Linux). This is a command-line interface to python and the python eco-system. In the anaconda prompt, run:

```
python -m pip install --user git+https://github.com/SGIModel/StarMuse
```

It should now be possible to run muse. Again, this can be done in the anaconda prompt as follows:

```
python -m muse --help
```

Note: Although not strictly necessary, users are encouraged to create an Anaconda virtual environment and install MUSE there, as shown in *For developers*.

1.2 For developers

Although not strictly necessary, creating an Anaconda virtual environment is highly recommended. Anaconda will isolate users and developers from changes occuring on their operating system, and from conflicts between python packages. It also ensures reproducibility from day to day.

Create a virtual env including python with:

```
conda create -n muse python=3.7
```

Activate the environment with:

```
conda activate muse
```

Later, to recover the system-wide "normal" python, deactivate the environment with:

```
conda deactivate
```

The simplest approach is to first download the muse code with git:

```
git clone https://github.com/SGIModel/StarMuse.git muse
```

For interested users, there are plenty of good tutorials for git. Next, it is possible to install the working directory into the conda environment:

```
# On Linux and Mac
cd muse
conda activate muse
python -m pip install -e ".[dev,docs]"

# On Windows
dir muse
conda activate muse
python -m pip install -e ".[dev,docs]"
```

The quotation marks are needed on some systems or shells, and do not hurt on any. The downloaded code can then be modified. The changes will be automatically reflected in the conda environment.

Tests can be run with the command pytest, from the testing framework of the same name.

The documentation can be built with:

```
python setup.py docs
```

The main page for the documentation can then be found at *build\sphinx\html\index.html* (or *build/sphinx/html/index.html* on Mac and Linux). The file can viewed from any web browser.

CHAPTER

TWO

RUNNING YOUR FIRST EXAMPLE

In this section we run an example simulation of MUSE and visualise the results. There are a number of different examples in the source code, which can be found INSERT LINK HERE.

Once python and MUSE have been installed, we can run an example. To do this open anaconda prompt. Then change directory to where you have downloaded the MUSE source code.

Navigate to the following link for MacOS or Linux based operating systems:

`{MUSE_download_location}/StarMuse/run/example/default/`

Change `{MUSE_download_location}` to the location you downloaded MUSE to, for example `Users/{my_name}/Documents/` using the `cd` command, or "change directory" command. Once we have navigated to the directory containing the example settings `settings.toml` we can run the simulation using the following command in the anaconda prompt or terminal:

`python -m muse settings.toml`

If running correctly, your prompt should output text similar to that which can be found here.

It is also possible to run MUSE directly in python using the following code:

[ ]:
```
from muse import examples
model = examples.model("default")
model.run()
```

2.1 Results

If the default MUSE example has run successfully, you should now have a folder called `Results` in the same directory as `settings.toml`.

This directory should contain results for each sector (`Gas`,`Power` and `Residential`) as well as results for the entire simulation in the form of `MCACapacity.csv` and `MCAPrices.csv`.

- `MCACapacity.csv` contains information about the capacity each agent has for each technology per year.
- `MCAPrices.csv` has the price of each commodity per year and timeslice. eg. the cost of electricity at night for electricity in 2020.

Within each of the sector result folders, there is an output for `Capacity` for each commodity in each year. The years into the future, which the simulation has not run to, refers to the capacity as it retires. Within the `Residential` folder there is also a folder for `Supply` within each year. This refers to how much end-use commodity was output.

The output can be fully configurable, as shown in the developer guide *here*.

2.2 Visualisation

[1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Next, we load the dataset of interest to us for this example: the `MCACapacity.csv` file. We do this using pandas.

[2]:
```python
capacity_results = pd.read_csv("Results/MCAcapacity.csv")
capacity_results.head()
```

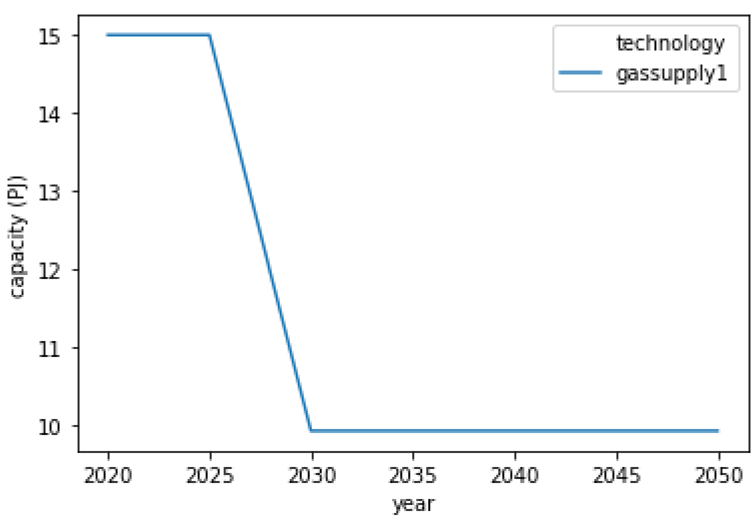
[2]:
```
   technology region agent      type       sector  capacity  year
0   gasboiler     R1    A1  retrofit  residential      10.0  2020
1     gasCCGT     R1    A1  retrofit        power       1.0  2020
2  gassupply1     R1    A1  retrofit          gas      15.0  2020
3   gasboiler     R1    A1  retrofit  residential       5.0  2025
4    heatpump     R1    A1  retrofit  residential      19.0  2025
```

Using the `head` command we print the first five rows of our dataset. Next, we will visualise each of the sectors, with capacity on the y-axis and year on the x-axis.

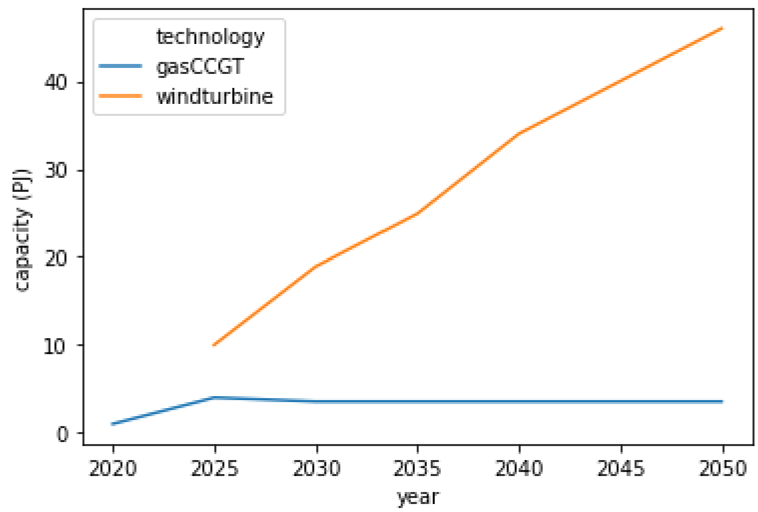
Don't worry too much about the code if some of it is unfamiliar. We effectively split the data into each sector and then plot a line plot for each.

[3]:
```python
for sector_name, results in capacity_results.groupby("sector"):
    print("{} sector".format(sector_name))
    sns.lineplot(data=results, x="year", y="capacity", hue="technology")
    plt.ylabel("capacity (PJ)")
    plt.show()
    plt.close()
```

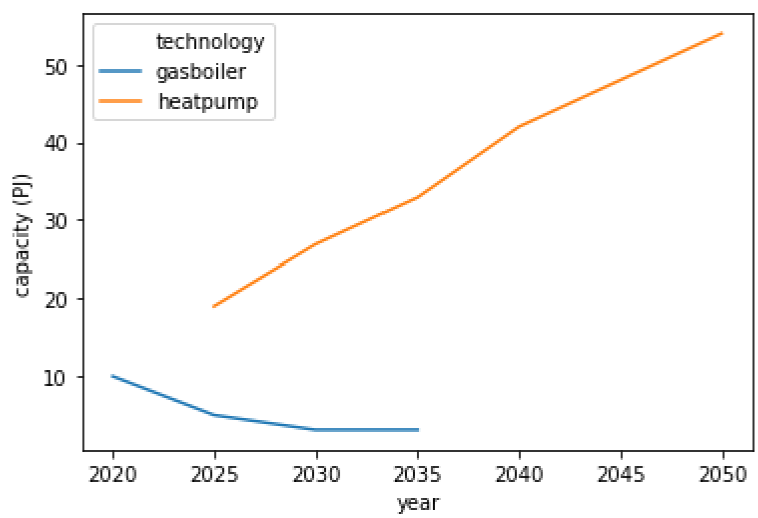
```
gas sector
```




```
power sector
```




```
residential sector
```



In this toy example, we can see that the end-use technology of choice in the residential sector becomes a heatpump. The heatpump displaces the gas boiler. Therefore, the supply of gas crashes due to a reduced demand. To account for the increase in demand for electricity, the agent invests heavily in wind turbines.

Note, that the units are in petajoules (PJ). MUSE requires consistent units across each of the sectors, and each of the input files (which we will see later). The model does not make any unit conversion internally.

2.3 Next steps

If you want to jump straight into customising your own example scenarios, head to the link *here*. If you would like a little bit of background based on how MUSE works first, head to the next section!

CHAPTER

THREE

MUSE OVERVIEW

Note: TODO: Potentially find introductory image to place here.

MUSE is an open source agent-based modelling environment that can be used to simulate change in an energy system over time. An example of the type of question MUSE can help in answering is:

- How may a carbon budget affect investments made in the power sector over the next 30 years?

MUSE can incorporate residential, power, industrial and conversion sectors, meaning many questions can be explored using MUSE, as per the wishes of the user.

MUSE is an agent-based modelling environment, where the agents are investors and consumers. In MUSE, this means that investment decisions are made from the point of view of the investor and consumer. These agents can be heterogenous, enabling for differering investment strategies between agents, as in the real world.

MUSE is technology rich and can model energy production, conversion and end-use technologies. So, for example, MUSE can enable the user to develop a power sector with solar photovoltaics, wind turbines and gas power plants which produce energy for appliances like electric stoves, heaters and lighting in the residential sector. Agents invest within these sectors, investing in technologies such as electric stoves in the residential sector or gas power plants in the power sectors. The investments made depend on the agent's investment strategies.

Every sector is a user configurable module. This means that a user can configure any number of sectors, cointaining custom, user-defined technologies and commodities. MUSE is fully data-driven, meaning that the configuration of the model is carried out using a selection of *Input Files*. This means that you are able to customise MUSE to your wishes by modifying these input files. In addition, MUSE can model any geographical region around the world and over any time scale, from a single year through to 100 years or more. Within a year, MUSE allows for a user-defined temporal granularity. This allows for the year to be split into different seasons and times, where energy demand may differ. Thus allowing us to model diurnal peaks in the demand, varying weekly and seasonally.

MUSE differs from the vast majority of energy systems models, which are intertemporal optimisation, by allowing agents to have "limited foresight". This enables these agents to invest under uncertainty of the future, as in the real world. In addition, MUSE is a "partial equilibrium" model, in the sense that it balances supply and demand of each energy commodity in the system.

3.1 What questions can MUSE answer?

MUSE allows for users to investigate how an energy system may evolve over a time period, based upon investors using different decision metrics or objectives such as the net present value, levelized cost of electricity or a custom-defined function. In addition to this, it can simulate how investors search for technology options, and how different objectives are combined to reach an investment decision.

The search for new technologies can depend on several factors such as agents' budgets, technology maturity or preferences on the fuel-type. For instance, an investor in the power sector may decide that they want to focus on renewable energy, whereas another may prefer the perceived most profitable option.

Examples of the questions MUSE can answer include:

- How may India's steel industry decarbonise?
- How might residential consumers change their investment decisions over time?
- How might a carbon tax impact investments made in the power sector?

3.2 How to use MUSE

There are a huge number of ways that MUSE could be used. The energy field is varied and diverse, and many different scenarios can be explored. Users can model the impact of changes in technology prices, demand, policy instruments, sector interactions and much, much more. People are always thinking of new ways that MUSE can be used. So, get creative!

A simulation model of a geographical region or world can be developed and is made up of the following features:

1. **Sectors** such as the power sector, gas production sector and the residential sector.
2. **Agents** such as a high-income subsection of the population in the UK or a risk-averse generation company. These agents are responsible for making investments in energy technologies.
3. **Technologies** which the agents choose to adopt. Technologies either produce an energy commodity (e.g. electricity), or a service demand (e.g. building space heating).
4. **Service demands** are demands that must be serviced such as lighting, heating or steel production.
5. **Market clearing algorithm** is the algorithm which determines global commodity prices based upon the balancing of supply and demand from each of the sectors. It must be noted, however, that only the conversion and supply sectors are able to modify prices; the demand sectors are price-takers, and so do not modify prices.
6. **Equilibrium prices** are the prices determined by the market clearing algorithm and can determine the investments made by agents in various sectors. This allows for the model to project how the system may develop over a time period.

These features are described in more detail in the rest of this documentation.

3.3 What are MUSE's unique features?

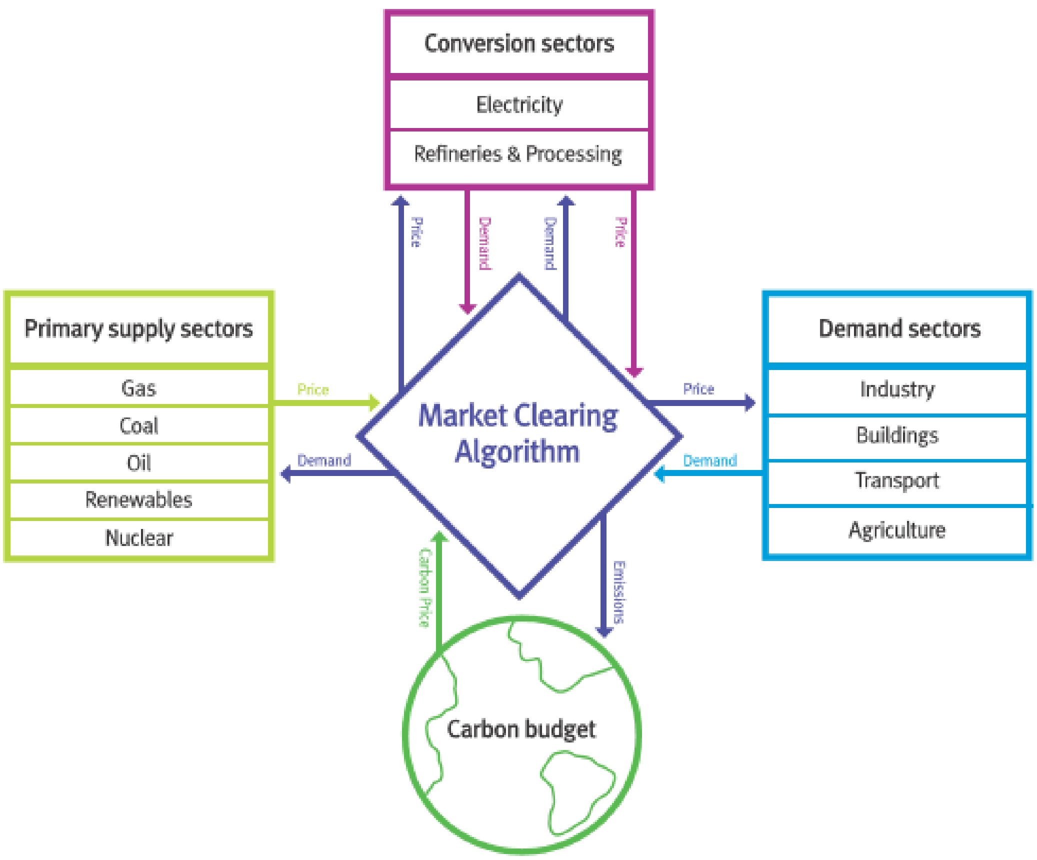
MUSE is a generalisable agent-based modelling environment and simulates energy transitions from the point of view of the investor and consumer agents. This means that users can define their own agents based upon their needs. The fact that MUSE is an agent-based model means that each of these agents can have different investment behaviours.

Additionally, agent-based models allow for agents to model imperfect information and limited foresight. An example of this is the ability to model the uncertainty residential users face when predicting the price of gas over the next 25 years. This is a unique feature to agent-based models when compared to intertemporal optimisation models and more closely models the real world. Many energy systems models are intertemporal optimisation models, which consider the viewpoint of a single benevolent decision maker, with perfect foresight and knowledge. These models optimise energy system investment and operation.

Whilst such intertemporal optimisation models are certainly useful, MUSE is different in that it models the incentives and challenges faced by investors. It can, therefore, be used to investigate different research questions, from the point of view of the investor and consumer. These questions are up to you, so impress us!

MUSE is completely open source, and ready for development.

3.4 Visualisation of MUSE



The figure above displays the key sectors of MUSE:

- Primary supply sectors; this allows to model diurnal peaks in the demand, varying weekly and seasonally.

- Conversion sectors
- Demand sectors
- Climate model (in the current model this is simplified by the use of a carbon budget.)
- Market clearing algorithm (MCA)

3.5 How MUSE works

MUSE works by iterating between sectors shown above to ensure that energy service demands are met by the technologies chosen by the agents. Next, we detail the calculations made by MUSE throughout the simulation.

1. The energy service demand is calculated. For example, how much electricity, gas and oil demand is there for cooking, building space heating and lighting in the residential sector? It must be noted, that this is only known after the energy service demand sector is solved and the technologies invested in are decided.
2. A demand sector is solved. That is, agents choose which end-use technologies to serve the demands in the sector. For example, electric stoves are compared to gas stoves to meet demand for cooking. These technologies are chosen based upon their:
   i. Search space (which technologies are they willing to consider?)
   ii. Their objectives (which metrics do they consider important?)
   iii. Their decision rules (how do they choose to combine their metrics if they have multiple?)
3. The decisions made by the agents in the demand sectors then leads to a certain level of demand for energy commodities, such as electricity, gas and oil, as a whole. This demand is then passed to the MCA.
4. The MCA then sends these demands to the sectors that supply these energy commodities (supply or conversion sectors).
5. The supply and conversion sectors are solved: agents in these sectors use the same approach (i.e. search space, objectives, decision rules) to decide which technologies to investment in to serve the energy commodity demand. For example, agents in the power sector may decide to invest in solar photovoltaics, wind turbines and gas power plants to service the electricity demand.
6. As a result of these decisions a price for each energy commodity is formed based upon supply and demand. This is passed to the MCA.
7. The MCA then sends these prices back to the demand sectors, which are solved again as above.
8. This process repeats itself until commodity supply and demand converges for each energy commodity. Once these converge, the model has found a "partial equilibrium" and it moves forward to the next time period.

CHAPTER FOUR

KEY MUSE COMPONENTS

MUSE is made up of five key components:

- Service Demand
- Technologies
- Sectors
- Agents
- Market Clearing Algorithm

In this section we will briefly explore what these components do and and how they interact.

4.1 Service Demand

The service demand is a user input which defines the demand that an end-use sector has. An example of this is the service demand commodity of heat or cooling that the residential sector requires. End-use in this case, refers to the energy which is utilised at the very final stage, after both extraction and conversion.

The estimate of the energy service is the first step. This estimate can be an exogenous input derived from the user, or correlations of GDP and population which reflect the socio-economic development of a region or country.

4.2 Technologies

Users are able to define any technology they wish for each of the energy sectors. Examples include power generators such as coal power plants, buses in the transport sector or lighting in the residential sector.

Each of the technologies are placed in their regions of interest, such as the USA or India. They are then defined by the following, but not limited to, variables:

- Capital costs
- Fixed costs
- Maximum capacity limit
- Maximum capaicty growth
- Lifetime of the technology
- Utilization factor
- Interest rate

Technologies, and their parameters are defined in the Technodata.csv file. For a full description of the input files, please refer to the *Techno-data* file.

4.3 Sectors

Sectors typically group areas of economic activity together, such as the residential sector, which might include all energy conusming activies of households. Possible examples of sectors are:

- Gas sector
- Power sector
- Residential sector
- Industrial sector

Each of these sectors contain their respective technologies which consume energy commodities. For example, the residential sector may consume electricity, gas or oil for a variety of different energy demands such as lighting, cooking and heating.

Each of the technologies, which consume a commodity, also output a different commodity or service demands. For example, a gas boiler consumes gas, but outputs heat and hot water.

4.4 Agents

Agents represent the investment decision makers in an energy system, for example consumers or companies. They invest in technologies that meet service demands, like heating, or produce other needed energy commodities, like electricity. These agents can be heterogenous, meaning that their investment priorities have the ability to differ.

As an example, a generation company could compare potential power generators based on their levelized cost of electricity, their net present value, by minimising the total capital cost, a mixture of these and/or any user-defined approach. This approach more closely matches the behaviour of real-life agents in the energy market, where companies, or people, have different priorities and constraints.

4.5 Market Clearing Algorithm

The market clearing algorithm (MCA) is the central component between the different supplies and demands of the energy system in question. The MCA iterates between the demand and supply of each of these sectors. Its role is to govern the endogenous price of commodities over the course of a simulation.

For a hypothetical example, the price of electricity is set to $70/MWh. However, at this price, the majority of residential agents prefer to heat their homes using gas. As a result of this, residential agents consume less electricity and more gas. This reduction in demand reduces the electricity price to $50/MWh. However, at this lower electricity price, some agents decide to invest in electric heating as opposed to gas. Eventually, the price converges on $60/MWh, where supply and demand for both electricity and gas are equal.

This is the principle of the MCA. It finds an equilibrium by iterating through each of the different sectors until an overall equilibrium is reached for each of the commodities. It is possible to run the MCA in a carbon budget mode, as well as exogenous mode. The carbon budget mode ensures that a carbon price limits the amount of carbon produce by the market. Whereas, the exogenous mode allows the carbon price to be set by the user.

CHAPTER

FIVE

CUSTOMISING MUSE TUTORIALS

Next, we show you how to customise MUSE to create your own scenarios.

We recommend following the tutorials step by step, as the files build on the previous example. If you prefer to jump straight in, your results may be different to the ones presented. To help you, we have provided the code to generate the various examples, in case you want to compare your code to ours.

5.1 Adding a new technology

5.1.1 Input Files

MUSE is made up of a number of different *input files*. These, however, can be broadly split into two:

- *Simulation settings*
- *Simulation data*

Simulation settings specify how a simulation should be run. For example, which sectors to run, for how many years and what to output.

Whereas, simulation data parametrises the technologies involved in the simulation, or the number and kinds of agents.

To create a customised case study it is necessary to edit both of these file types.

Simulation settings are specified in a TOML file. TOML is a simple, extensible and intuitive file format well suited for specifying small sets of complex data.

Simulation data is specified in CSV. This is a common format used for larger datasets, and is made up of columns and rows, with a comma used to differentiate between entries.

MUSE requires at least the following files to successfully run:

- a single *simulation settings TOML file* for the simulation as a whole
- a file indicating initial market price *projections*
- a file describing the *commodities in the simulation*
- for generalized sectors:
  - a file descring the *agents*
  - a file descring the *technologies*
  - a file descring the *input commodities* for each technology
  - a file descring the *output commodities* for each technology
  - a file descring the *existing capacity* of a given sector

- for each preset sector:
  - a csv file describing consumption for the duration of the simulation

For a full description of these files see the *input files section*. To see how to customise an example, continue on this page.

5.1.2 Addition of solar PV

In this section, we will add solar photovoltaics to the default model seen in the *example page*. To achieve this, we must modify some of the input files shown in the above section. These files can be found in the `StarMuse` folder at the following location:

`{muse_install_location}/src/muse/data/example/default`

Change `{muse_install_location}` to the location where you installed MUSE using your file browser. You can modify the files in your favourite spreadsheet editor or text editor such as Excel, Numbers, Notepad or TextEdit.

Technodata Input

Within the default folder there is the `settings.toml` file, input folder and technodata folder. To add a technology within the power sector, we must open the `technodata` folder followed by the `power` folder.

Next, we will edit the `CommIn.csv` file, which specifies the commodities consumed by solar photovoltaics.

The table below shows the original `CommIn.csv` version in normal text, and the added column and row in **bold**.

ProcessName	RegionName	Time	Level	electricity	gas	heat	CO2f	wind	**solar**
Unit	•	Year	•	PJ/PJ	PJ/PJ	PJ/PJ	kt/PJ	PJ/PJ	**PJ/PJ**
gasCCGT	R1	2020	fixed	0	1.67	0	0	0	**0**
windturbine	R1	2020	fixed	0	0	0	0	1	**0**
solarPV	**R1**	**2020**	**fixed**	**0**	**0**	**0**	**0**	**0**	**1**

We must first add a new row at the bottom of the file, to indicate the new solar photovoltaic technology:

- we call this technology `solarPV`
- place it in region `R1`
- the data in this row is associated to the year 2020
- the input type is fixed
- solarPV consumes solar

As the solar commodity has not been previously defined, we must define it by adding a column, which we will call solar. We fill out the entries in the solar column, ie. that neither `gasCCGT` nor `windturbine` consume solar.

We repeat this process for the file: `CommOut.csv`. This file specifies the output of the technology. In our case, solar photovoltaics only output `electricity`. This is unlike `gasCCGT` which also outputs `CO2f`, or carbon dioxide.

ProcessName	RegionName	Time	Level	electricity	gas	heat	CO2f	wind	solar
Unit	•	Year	•	PJ/PJ	PJ/PJ	PJ/PJ	kt/PJ	PJ/PJ	**PJ/PJ**
gasCCGT	R1	2020	fixed	1	0	0	91.67	0	**0**
windturbine	R1	2020	fixed	1	0	0	0	0	**0**
solarPV	**R1**	**2020**	**fixed**	**1**	**0**	**0**	**0**	**0**	**0**

Similar to the the `CommIn.csv`, we create a new row, and add in the solar commodity. We must ensure that we call our new commodity and technologies the same as the previous file for MUSE to successfully run. ie `solar` and `solarPV`

The next file to modify is the `ExistingCapacity.csv` file. This file details the existing capacity of each technology, per year. For this example, we will set the existing capacity to be 0.

ProcessName	RegionName	Unit	2020	2025	2030	2035	2040	2045	2050
gasCCGT	R1	PJ/y	1	1	0	0	0	0	0
windturbine	R1	PJ/y	0	0	0	0	0	0	0
solarPV	**R1**	**PJ/y**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Finally, the `technodata.csv` containts parametrisation data for the technology, such as the cost, growth constraints, lifetime of the power plant and fuel used. The technodata file is too long for it all to be displayed here, so we will truncate the full version.

Here, we will only define the parameters: `processName`, `RegionName`, `Time`, `Level`,`cap_par`, `Fuel`,`EndUse`,`Agent2` and `Agent1`

We shall copy the existing parameters from the `windturbine` technology for the remaining parameters that can be seen in the `technodata.csv` file for brevity. You can see the full file here INSERT LINK HERE

ProcessName	RegionName	Time	Level	cap_par	cap_exp	…	Fuel	EndUse	Agent2	Agent1
Unit	•	Year	•	MUS$2010/PJ_a	•	…	•	•	Retrofit	New
gasCCGT	R1	2020	fixed	23.78234399	1	…	gas	electricity	1	0
windturbine	R1	2020	fixed	36.30771182	1	…	wind	electricity	1	0
solarPV	**R1**	**2020**	**fixed**	**30**	**1**	…	**solar**	**electricity**	**1**	**0**

Global inputs

Next, navigate to the `input` folder, found at `{muse_installation_location}src/muse/data/example/default/input`.

We now must edit each of the files found here to add the new solar commodity. Due to space constraints we will not display all of the entries contained in the input files. The edited files can be viewed here INSERT LINK HERE however.

The `BaseYearExport.csv` file defines the exports in the base year. For our example we add a column to indicate that there is no export for solar. However, it is important that a column exists for our new commodity.

RegionName	Attribute	Time	electricity	gas	heat	CO2f	wind	**solar**
Unit	•	Year	PJ	PJ	PJ	kt	PJ	**PJ**
R1	Exports	2010	0	0	0	0	0	**0**
R1	Exports	2015	0	0	0	0	0	**0**
…	…	…	…	…	…	…	…	**…**
R1	Exports	2100	0	0	0	0	0	**0**

The `BaseYearImport.csv` file defines the imports in the base year. Similarly to `BaseYearExport.csv`, we add a column for solar in the `BaseYearImport.csv` file. Again, we indicate that solar has no imports.

RegionName	Attribute	Time	electricity	gas	heat	CO2f	wind	**solar**
Unit	•	Year	PJ	PJ	PJ	kt	PJ	**PJ**
R1	Imports	2010	0	0	0	0	0	**0**
R1	Imports	2015	0	0	0	0	0	**0**
…	…	…	…	…	…	…	…	**…**
R1	Imports	2100	0	0	0	0	0	**0**

The `GlobalCommodities.csv` file is the file which defines the commodities. Here we give the commodities a commodity type, CO2 emissions factor and heat rate. For this file, we will add the solar commodity, with zero CO2 emissions factor and a heat rate of 1.

Commodity	Commodity-Type	Commodity-Name	CommodityEmissionFactor_CO2	HeatRate	Unit
Electricity	Energy	electricity	0	1	PJ
Gas	Energy	gas	56.1	1	PJ
Heat	Energy	heat	0	1	PJ
Wind	Energy	wind	0	1	PJ
CO2fuelcomsbustion	Environmental	CO2f	0	1	kt
Solar	**Energy**	**solar**	**0**	**1**	**PJ**

The `projections.csv` file details the initial market prices for the commodities. The market clearing algorithm will update these throughout the simulation, however, an initial estimate is required to start the simulation. As solar energy is free, we will indicate this by adding a final column.

RegionName	Attribute	Time	electricity	gas	heat	CO2f	wind	**solar**
Unit	•	Year	MUS$2010/PJ	MUS$2010/PJ	MUS$2010/PJ	MUS$2010/kt	MUS$2010/kt	**MUS$2010/kt**
R1	CommodityPrice	2010	14.81481472	6.6759	100	0	0	**0**
R1	CommodityPrice	2015	17.89814806	6.914325	100	0.052913851	0	**0**
…	…	…	…	…	…	…	…	**…**
R1	CommodityPrice	2100	21.39814806	7.373485819	100	1.871299697	0	**0**

Running our customised simulation

Now we are able to run our simulation, with the new solar power technology.

To do this we run the same run command as previously in the anaconda command prompt:

```
python -m muse settings.toml
```

The output should be similar to the output here. However, expect the simulation to take slightly longer to run. This is due to the additional calculations made.

If the simulation has run successfully, you should now have a folder in the same location as your settings.toml file called Results. The next step is to visualise the results using the python visualisation package `seaborn` as well as the data analysis library `pandas`.

[2]:
```
import seaborn as sns
import pandas as pd
```

Next, we will import the `MCACapacity.csv` file into pandas and print the first 5 lines using the `head()` command.

Make sure to change the file path of `"../Results/MCACapacity.csv"` to where the `MCACapacity.csv` is on your computer, otherwise you will receive an error when you import the csv file.

[6]:
```
mca_capacity = pd.read_csv("../Results/MCACapacity.csv")
mca_capacity.head()
```

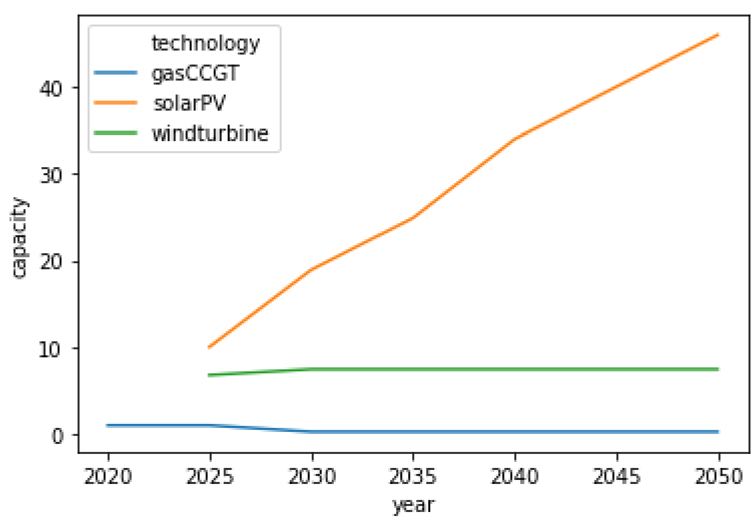
[6]:

	technology	region	agent	type	sector	capacity	year
0	gasboiler	R1	A1	retrofit	residential	10.0	2020
1	gasCCGT	R1	A1	retrofit	power	1.0	2020
2	gassupply1	R1	A1	retrofit	gas	15.0	2020
3	gasboiler	R1	A1	retrofit	residential	5.0	2025
4	heatpump	R1	A1	retrofit	residential	19.0	2025

We will only visualise the power sector in this example, as this was the only sector we changed. We, therefore, filter for this sector, and then visualise it using `seaborn`:

[7]:
```
power_capacity = mca_capacity[mca_capacity.sector=="power"]
sns.lineplot(data=power_capacity, x='year', y='capacity', hue="technology")
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd544a6e760>
```



We can now see that there is solarPV in addition to windturbine and gasCCGT, when compared to the example *here*! That's great and means it worked!

The difference in uptake of `solarPV` compared to `windturbine` is due to the fact that `solarPV` has a lower `cap_par` cost of 30, compared to the `windturbine`. Meaning that `solarPV` outcompetes both `windturbine` and `gasCCGT` in the electricity market.

Change Solar Price

Now, we will observe what happens if we increase the price of solar to be more expensive than wind. To achieve, this we have to modify the `Technodata.csv` file:

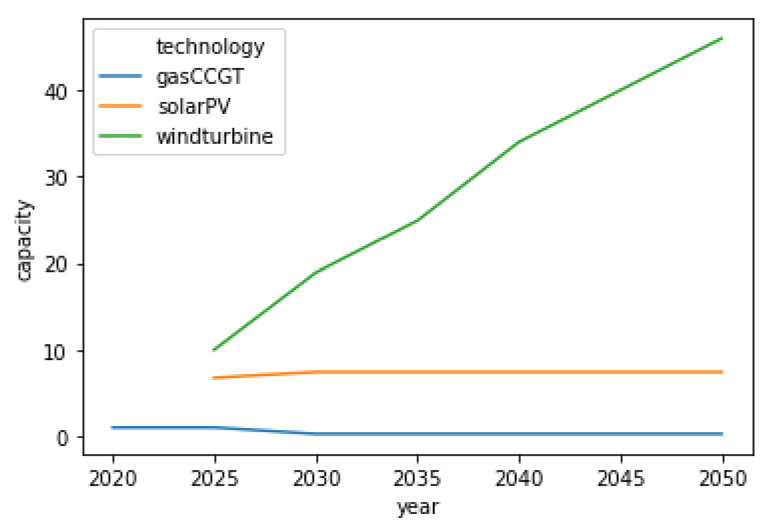
ProcessName	RegionName	Time	Level	cap_par	cap_exp	…	Fuel	EndUse	Agent2	Agent1
Unit	-	Year	-	MUS$2010/PJ_a	-	…	-	-	Retrofit	New
gasCCGT	R1	2020	fixed	23.78234399	1	…	gas	electricity	1	0
windturbine	R1	2020	fixed	36.30771182	1	…	wind	electricity	1	0
solarPV	R1	2020	fixed	**40**	1	…	solar	electricity	1	0

Here, we increase the `cap_par` variable by 10, to be a total of 40. We will now rerun the simulation, using the same command as previously and visualise the new results.

We must import the new `MCACapacity.csv` file again, and then visualise the results.

```
[8]: mca_capacity = pd.read_csv("../Results/MCACapacity.csv")
     power_capacity = mca_capacity[mca_capacity.sector=="power"]
     sns.lineplot(data=power_capacity, x='year', y='capacity', hue="technology")
```

```
[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd544ef7ee0>
```



Now, we can see that the technology `windturbine` outcompetes `solarPV` and `gasCCGT` due to the difference in price. The possibilities for creating your own scenarios are infinite.

For the full example with the completed input files see here INSERT LINK HERE

5.1.3 Next steps

In the next section we will add a new agent to the simulation.

5.2 Adding an agent

In this section, we will add a new agent called `A2`. This agent will be slightly different to the other agents in the `default` example, in that it will make investments based upon a mixture of levelised cost of electricity (LCOE) and net present value (NPV). These two objectives will be combined by calculating the mean of the two when comparing potential investment options.

To achieve this, we must modify the `Agents.csv` file in the directory:

```
{muse_install_location}/src/muse/data/example/default/technodata/Agents.csv
```

To do this, we will add two new rows to the file. To simplify the process, we copy the data from the first two rows of agent `A1`, changing only the rows: `Name`, `Objective1`, `Objective2`, `ObjData1`, `ObjData2` and `DecisionMethod`. The values we changed can be seen below. Again, we only show some of the rows due to space constraints, however see here for the full file.

AgentShare	Name	AgentNumber	RegionName	Objective1	Objective2	Objective3	ObjData1	ObjData2	...	DecisionMethod	...	Type
Agent1	A1	1	R1	LCOE			1		...	singleObj	...	New
Agent2	A1	2	R1	LCOE			1		...	singleObj	...	Retrofit
Agent1	**A2**	**1**	**R1**	**LCOE**	**NPV**		**1**	**1**	**...**	**mean**	**...**	**New**
Agent2	**A2**	**2**	**R1**	**LCOE**	**NPV**		**1**	**1**	**...**	**mean**	**...**	**Retrofit**

We will now save this file and run the new simulation model using the following command:

```
python -m muse settings.toml
```

Again, we use seaborn and pandas to analyse the data in the `Results` folder.

[6]:
```python
import pandas as pd
import seaborn as sns
```

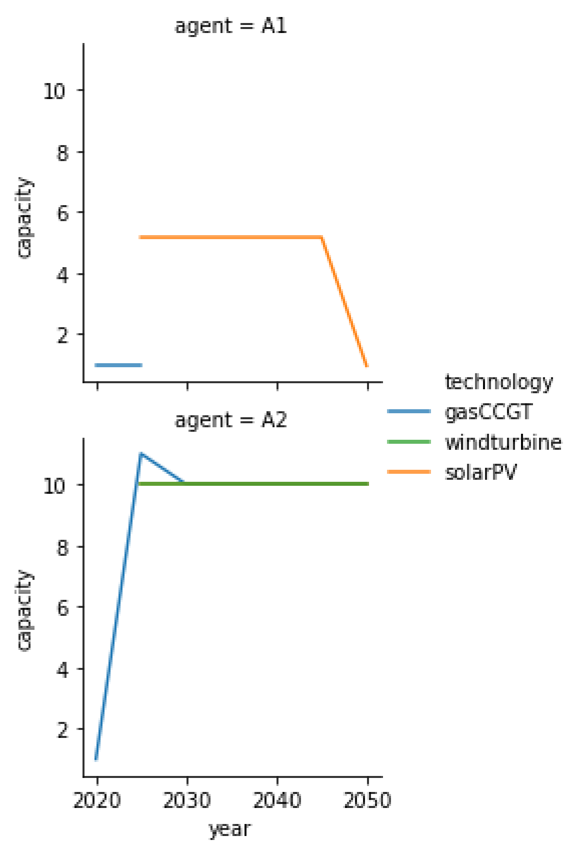
[19]:
```python
mca_capacity = pd.read_csv("../Results/MCACapacity.csv")
power_sector = mca_capacity[mca_capacity.sector=="power"]
power_sector.head()
```

[19]:
```
     technology region agent      type sector  capacity  year
2       gasCCGT     R1    A1  retrofit  power     1.000  2020
3       gasCCGT     R1    A2  retrofit  power     1.000  2020
10      gasCCGT     R1    A1  retrofit  power     1.000  2025
11  windturbine     R1    A1  retrofit  power     5.172  2025
12      gasCCGT     R1    A2  retrofit  power    11.000  2025
```

This time we can see that there is data for the new agent, `A2`. Next, we will visualise the investments made by each of the agents using seaborn's facetgrid command.

```
[20]: g=sns.FacetGrid(power_sector, row='agent')
      g.map(sns.lineplot, "year", "capacity", "technology")
      g.add_legend()
```

```
[20]: <seaborn.axisgrid.FacetGrid at 0x7f95819b4730>
```



In this scenario, agent `A1` is investing using LCOE, whereas agent `A2` is investing based on the mean of the objectives: LCOE and NPV in the same region. A different strategy is employed by these agents with `A2` investing in gasCCGT and windturbines, whereas `A1` invests in solarPV.

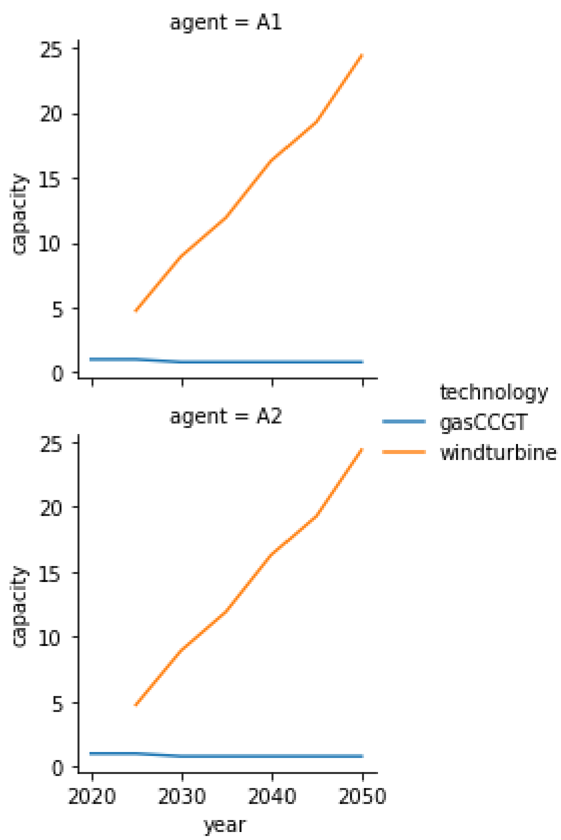
Next, we will see what occurs if the agents invest based upon the same investment strategy, with both investing using NPV. This requires to edit the `Agents.csv` file once more, to look like the following:

AgentShare	Name	Agent-Num-ber	Re-gion-Name	Ob-jec-tive1	Ob-jec-tive2	Ob-jec-tive3	Obj-Data1	Obj-Data2	…	Deci-sion-Method	…	Type
Agent1	A1	1	R1	LCOE			1		…	sin-gleObj	…	New
Agent2	A1	2	R1	LCOE			1		…	sin-gleObj	…	Retrofit
Agent1	A2	1	R1	**LCOE**			**1**		…	**sin-gleObj**	…	New
Agent2	A2	2	R1	**LCOE**			**1**		…	**sin-gleObj**	…	Retrofit

Again, this requires the re-running of the simulation, and visualisation like before:

[21]:
```
mca_capacity = pd.read_csv("../Results/MCACapacity.csv")
power_sector = mca_capacity[mca_capacity.sector=="power"]
g=sns.FacetGrid(power_sector, row='agent')
g.map(sns.lineplot, "year", "capacity", "technology")
g.add_legend()
```

[21]:
```
<seaborn.axisgrid.FacetGrid at 0x7f957e5f0970>
```



In this new scenario, with both agents running the same objective, very similar results can be seen, with a high investmwent in `windturbine`, none in `solarPV` and low `gasCCGT`. Have a play around with the files to see if you can come up with different scenarios!

5.2.1 Next steps

In the next section we will show you how to add a new region.

5.3 Adding a region

The next step is to add a region which we will call `R2`, however, this could equally be called `USA` or `India`. This requires a similar process to before of modifying the input simulation data. However, we will also have to change the `settings.toml` file to achieve this.

The process to change the `settings.toml` file is relatively simple. We just have to add our new region to the `regions` variable, in the 4th line of the `settings.toml` file, like so:

```
regions = ["R1", "R2"]
```

The process to change the input files, however, takes a bit more time. To achieve this, there must be data for each of the sectors for the new region. This, therefore, requires the modification of every *input file*.

Due to space constraints, we will not show you how to edit all of the files. However, you can access the modified files here INSERT LINK HERE.

Effectively, for this example, we will copy and paste the results for each of the input files from region `R1`, and change the name of the region for the new rows to `R2`.

However, as we are increasing the demand by adding a region, as well as modifying the costs of technologies, it may be the case that a higher growth in technology is required. For example, there may be no possible solution to meet demand without increasing the `windturbine` maximum allowed limit. We will therefore increase the allowed limits for `windturbine` in region `R2`.

We have placed two examples as to how to edit the residential sector below. Again, the edited data are highlighted in **bold**, with the original data in normal text.

The following file is the modified `/technodata/residential/CommIn.csv` file:

ProcessName	RegionName	Time	Level	electricity	gas	heat	CO2f	wind
Unit	•	Year	•	PJ/PJ	PJ/PJ	PJ/PJ	kt/PJ	PJ/PJ
gasboiler	R1	2020	fixed	0	1.16	0	0	0
heatpump	R1	2020	fixed	0.4	0	0	0	0
gasboiler	**R2**	**2020**	**fixed**	**0**	**1.16**	**0**	**0**	**0**
heatpump	**R2**	**2020**	**fixed**	**0.4**	**0**	**0**	**0**	**0**

Whereas the following file is the modified `/technodata/residential/ExistingCapacity.csv` file:

ProcessName	RegionName	Unit	2020	2025	2030	2035	2040	2045	2050
gasboiler	R1	PJ/y	10	5	0	0	0	0	0
heatpump	R1	PJ/y	0	0	0	0	0	0	0
gasboiler	**R2**	**PJ/y**	**10**	**5**	**0**	**0**	**0**	**0**	**0**
heatpump	**R2**	**PJ/y**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Below is the reduced `/technodata/power/technodata.csv` file, showing the increased capacity for `windturbine` in `R2`. For this, we highlight only the elements we changed from the rows in `R1`. The rest of the elements are the same for `R1` as they are for `R2`.

ProcessName	RegionName	...	MaxCapacityAddition	MaxCapacityGrowth	TotalCapacityLimit		Agent2	Agent1
Unit	•	...	PJ	%	PJ	...	Retrofit	New
gasCCGT	R1	...	2	0.02	60	...	1	0
windturbine	R1	...	2	0.02	60	...	1	0
solarPV	R1	...	2	0.02	60	...	1	0
gasCCGT	R2	...	2	0.02	60	...	1	0
windturbine	R2	...	**5**	**0.05**	**100**	...	1	0
solarPV	R2	...	2	0.02	60	...	1	0

Now, go ahead and amend all of the other input files for each of the sectors by copying and pasting the rows from `R1` and replacing the `RegionName` to `R2` for the new rows. All of the edited input files can be seen here.

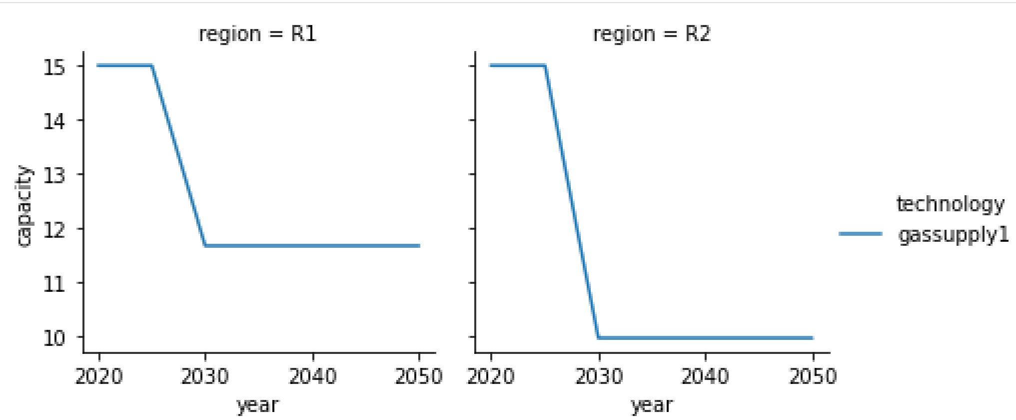
Again, we will run the results using the `python -m pip muse settings.toml` in anaconda prompt, and analyse the data as follows:

[1]:
```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
```

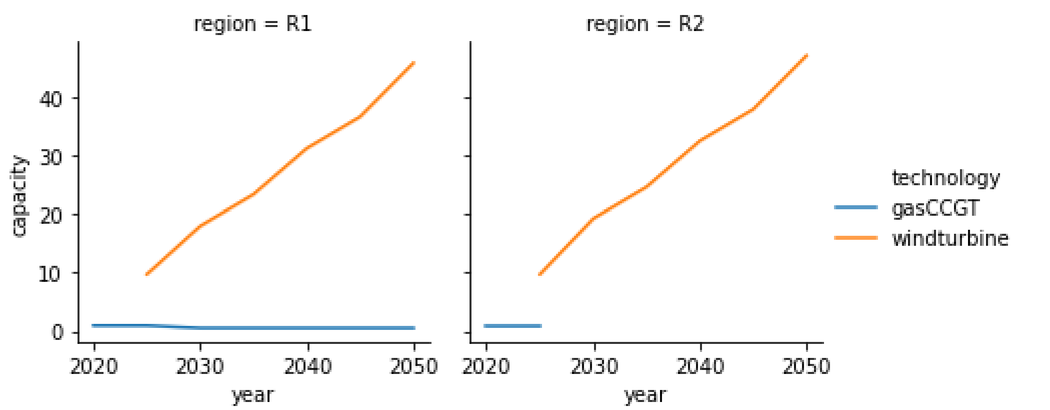
[2]:
```
mca_capacity = pd.read_csv("../tutorial-code/add-region/Results/MCACapacity.csv")

for name, sector in mca_capacity.groupby("sector"):
    print("{} sector:".format(name))
    g = sns.FacetGrid(data=sector, col="region")
    g.map(sns.lineplot, "year", "capacity", "technology")
    g.add_legend()
    plt.show()
```

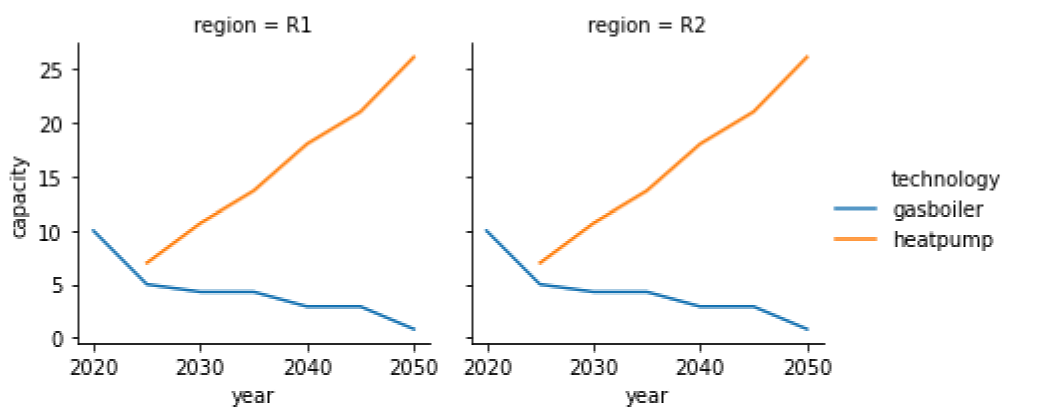
```
gas sector:
```




```
power sector:
```




```
residential sector:
```



Due to the similar natures of the two regions, with the parameters effectively copied and pasted between them, the results are very similar in both `R1` and `R2`. `gassupply1` drops significantly within the gas sector, due to the increasing demand of `heatpump` and falling demand of `gasboiler` in both region `R1` and `R2`. `windturbine` increases significantly to match this `heatpump` demand.

Have a play around with the various costs data in the technodata files for each of the sectors and technologies to see if different scenarios emerge. Although be careful. In some cases, the constraints on certain technologies will make it impossible for the demand to be met. Therefore you may have to relax these constraints.

5.3.1 Next steps

In the next section we modify the `settings.toml` file to change the timeslicing arrangements as well as project until 2040, instead of 2050, in two year timeslices.

5.4 Modification of time

In this section we will show you how to modify the timeslicing arrangement as well as change the time horizon and year intervals by modifying the `settings.toml` file.

5.4.1 Modify timeslicing

Timeslicing is the division of a single year into multiple different sections. For example, we could slice the year into different seasons, make a distinction between weekday and weekend or a distinction between morning and night. We do this as energy demand profiles can show a difference between these timeslices. eg. Electricity consumption is lower during the night than during the day.

To achieve this, we have to modify the `settings.toml` file, as well as the files within the preset folder: `Residential2020Consumption.csv` and `Residential2050Consumption.csv`. This is so that we can edit the demand for the residential sector for the new timeslices.

First we edit the `settings.toml` file to add two additional timeslices: early-morning and late-afternoon. We also rename afternoon to mid-afternoon. These settings can be found at the bottom of the `settings.toml` file.

An example of the changes is shown below:

```
[timeslices]
all-year.all-week.night = 1095
all-year.all-week.morning = 1095
all-year.all-week.mid-afternoon = 1095
all-year.all-week.early-peak = 1095
all-year.all-week.late-peak = 1095
all-year.all-week.evening = 1095
all-year.all-week.early-morning = 1095
all-year.all-week.late-afternoon = 1095
level_names = ["month", "day", "hour"]
```

Next, we modify both Residential Consumption files. Again, we put the text in bold for the modified entries. We must add the demand for the two additional timelsices, which we call timeslice 7 and 8. We make the demand for heat to be 2 for both of the new timeslices.

Below is the modified `Residential2020Consumption.csv` file:

	RegionName	ProcessName	Timeslice	electricity	gas	heat	CO2f	wind
0	R1	gasboiler	1	0	0	1	0	0
1	R1	gasboiler	2	0	0	1.5	0	0
2	R1	gasboiler	3	0	0	1	0	0
3	R1	gasboiler	4	0	0	1.5	0	0
4	R1	gasboiler	5	0	0	3	0	0
5	R1	gasboiler	6	0	0	2	0	0
6	**R1**	**gasboiler**	**7**	**0**	**0**	**2**	**0**	**0**
7	**R1**	**gasboiler**	**8**	**0**	**0**	**2**	**0**	**0**
0	R2	gasboiler	1	0	0	1	0	0
1	R2	gasboiler	2	0	0	1.5	0	0
2	R2	gasboiler	3	0	0	1	0	0
3	R2	gasboiler	4	0	0	1.5	0	0
4	R2	gasboiler	5	0	0	3	0	0
5	R2	gasboiler	6	0	0	2	0	0
6	**R2**	**gasboiler**	**7**	**0**	**0**	**2**	**0**	**0**
7	**R2**	**gasboiler**	**8**	**0**	**0**	**2**	**0**	**0**

We do the same for the `Residential2050Consumption.csv`, however this time we make the demand for heat in 2050 to both be 5 for the new timeslices. See here INSERT LINK HERE for the full file.

Once the relevant files have been edited, we are able to run the simulation model using `python -m muse settings.toml`.

Then, once run, we import the necessary packages:

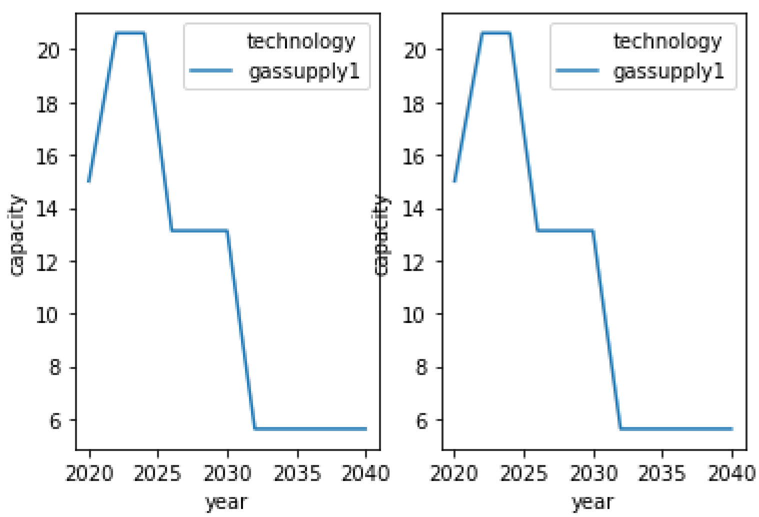
[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

and visualise the relevant data:

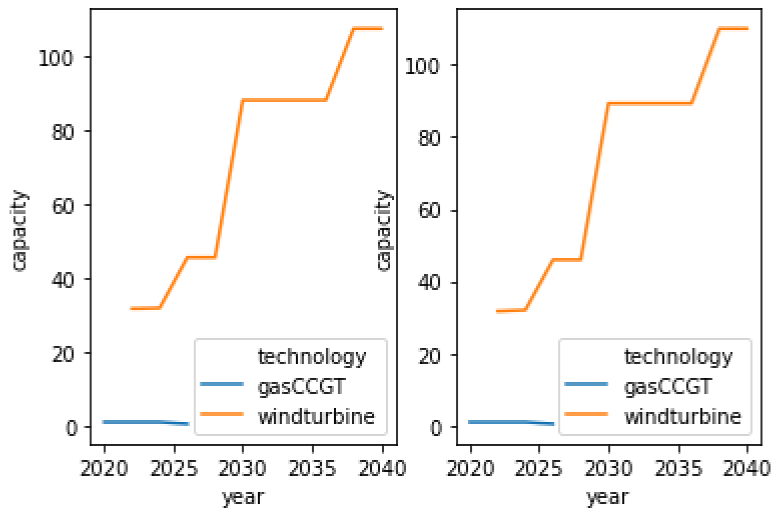
[2]:
```
mca_capacity = pd.read_csv("../tutorial-code/modify-timing-data/modify-time-framework/
↪Results/MCACapacity.csv")

for name, sector in mca_capacity.groupby("sector"):
    print("{} sector:".format(name))
    fig, ax =plt.subplots(1,2)
    sns.lineplot(data=sector[sector.region=="R1"], x="year", y="capacity", hue=
↪"technology", ax=ax[0])
    sns.lineplot(data=sector[sector.region=="R2"], x="year", y="capacity", hue=
↪"technology", ax=ax[1])
    plt.show()
    plt.close()
```

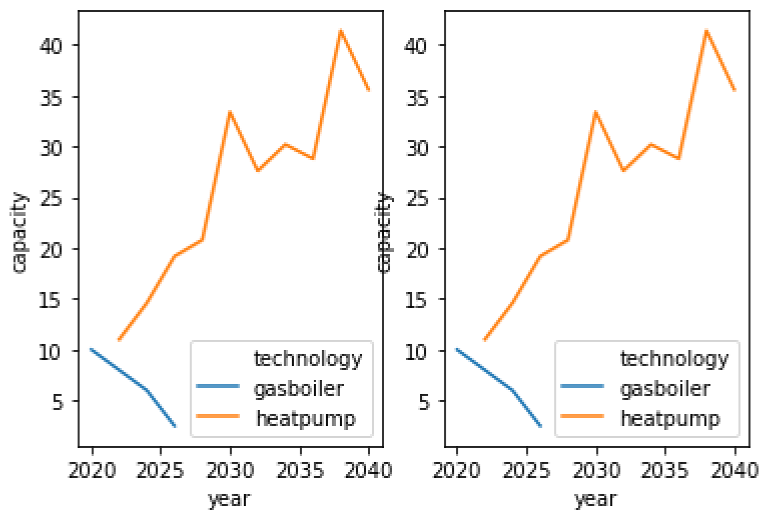
```
gas sector:
```




```
power sector:
```




```
residential sector:
```



Compared to the scenario where we added a *region*, there is a slight increase in solarPV in the power sector. However, the rest remains unchanged.

5.4.2 Modify time horizon and time periods

For the previous examples, we have run the scenario from 2020 to 2050, in 5 year time steps. This has been set at the top of the `settings.toml` file. However, we may want to run a more detailed scenario, with 2 year time steps, and up until the year 2040.

Making this change is quite simple as we only have two lines to change. We will modify line 2 and 3 of the `settings.toml` file, as follows:

```
# Global settings - most REQUIRED
time_framework = [2020, 2022, 2024, 2026, 2028, 2030, 2032, 2034, 2036, 2038, 2040]
foresight = 2   # Has to be a multiple of the minimum separation between the years in␣
↪time
```

The `time_framework` details each year in which we run the simulation. The `foresight` variable details how much foresight an agent has when making investments.

As we have modified the timeslicing arrangements there will be a change in the underlying demand for heating. This may require more electricity to service this demand. Therefore, we relax the constraints for growth in the power sector for all technologies and constraints in the `technodata/power/technodata.csv`, as is shown below:

ProcessName	RegionName	…	MaxCapacityAddition	MaxCapacityGrowth	TotalCapacityLimit	…	Agent1
Unit	•	…	PJ	%	PJ	…	New
gasCCGT	R1	…	**40**	**0.2**	**120**	…	0
windturbine	R1	…	**40**	**0.2**	**120**	…	0
solarPV	R1	…	**40**	**0.2**	**120**	…	0
gasCCGT	R2	…	**40**	**0.2**	**120**	…	0
windturbine	R2	…	**40**	**0.2**	**120**	…	0
solarPV	R2	…	**40**	**0.2**	**120**	…	0

We also modify the constraints defined in the `technodata.csv` file for the residential sector, as shown below:

ProcessName	RegionName	…	MaxCapacityAddition	MaxCapacityGrowth	TotalCapacityLimit	…	Agent1
Unit	•	…	PJ	%	PJ	…	New
gasboiler	R1	…	**60**	**0.5**	**120**	…	0
heatpump	R1	…	**60**	**0.5**	**120**	…	0
gasboiler	R2	…	**60**	**0.5**	**120**	…	0
heatpump	R2	…	**60**	**0.5**	**120**	…	0

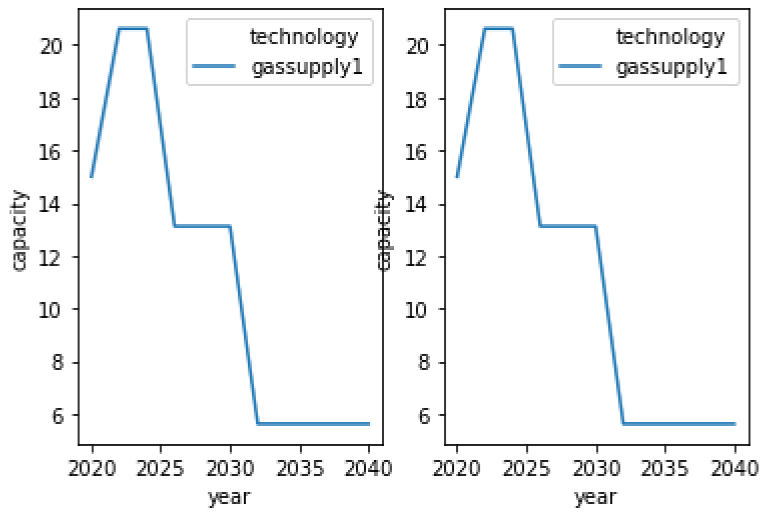
It must be noted, that this is a toy example. For modelling a real life scenario, data should be sought to ensure there remain realistic constraints.

For the full power sector `technodata.csv` file click here INSERT LINK HERE, and for the full residential sector `technodata.csv` file click here INSERT LINK HERE.

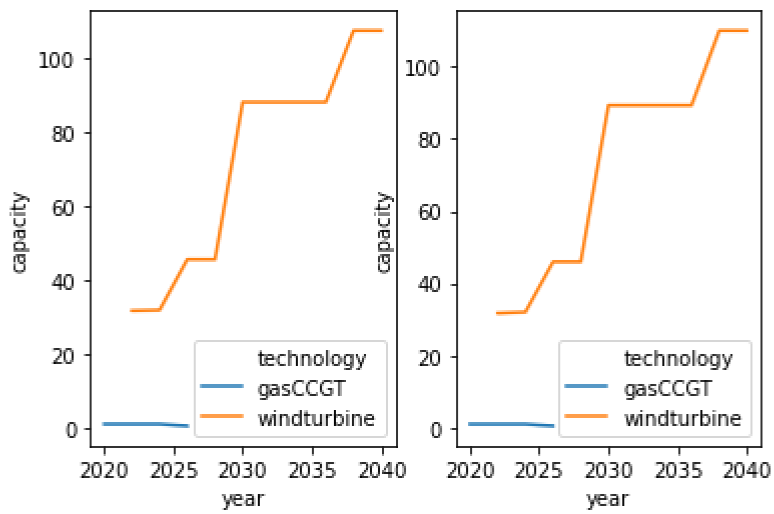
[3]:
```
mca_capacity = pd.read_csv("../tutorial-code/modify-timing-data/modify-time-framework/
↪Results/MCACapacity.csv")

for name, sector in mca_capacity.groupby("sector"):
    print("{} sector:".format(name))
    fig, ax =plt.subplots(1,2)
    sns.lineplot(data=sector[sector.region=="R1"], x="year", y="capacity", hue=
↪"technology", ax=ax[0])
    sns.lineplot(data=sector[sector.region=="R2"], x="year", y="capacity", hue=
↪"technology", ax=ax[1])
    plt.show()
    plt.close()
```

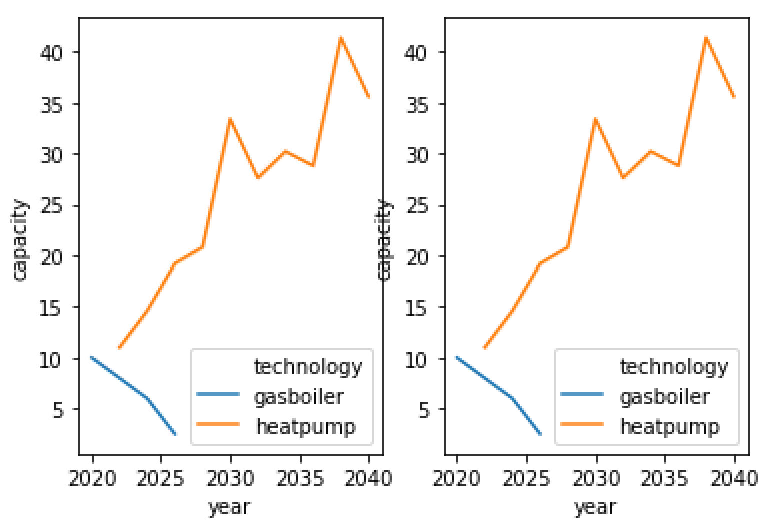
```
gas sector:
```




```
power sector:
```




```
residential sector:
```



5.4.3 Next steps

In the next section we detail how to add an exogenous service demand, such as demand for heating or cooking.

5.5 Adding a service demand

In this section, we will detail how to add a service demand to MUSE.

A service demand is an end-use demand. For example, in the residential sector, a service demand could be cooking. Houses require energy to cook food and a technology to service this demand, such as an electric stove.

This process consists of setting a demand, either through inputs derived from the user or correlations of GDP and population which reflect the socio-economic decvelopment of a region or country. In addition, a technology must be added to service the demand.

5.5.1 Addition of cooking demand

Firstly, we must add the demand section. In this example, we will add a cooking preset demand. To achieve this, we will now edit the `Residential2020Consumption.csv` and `Residential2050Consumption.csv` files, found within the `technodata/preset/` directory.

The `Residential2020Consumption.csv` file allows us to specify the demand in 2020 for each region and technology per timeslice. The `Residential2050Consumption.csv` file does the same but for the year 2050. The datapoints between these are interpolated.

Firstly, we must add the new service demand: `cook` as a column in these two files. Next, we add the demand. Again, the modified entries are in bold:

	RegionName	ProcessName	Timeslice	electricity	gas	heat	CO2f	wind	**cook**
0	R1	gasboiler	1	0	0	1	0	0	**0**
…	…	…	…	…	…	…	…	…	**…**
15	R2	gasboiler	8	0	0	2	0	0	**0**
16	**R1**	**electric_stove**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
17	**R1**	**electric_stove**	**2**	**0**	**0**	**0**	**0**	**0**	**2**
18	**R1**	**electric_stove**	**3**	**0**	**0**	**0**	**0**	**0**	**1**
19	**R1**	**electric_stove**	**4**	**0**	**0**	**0**	**0**	**0**	**1.5**
20	**R1**	**electric_stove**	**5**	**0**	**0**	**0**	**0**	**0**	**2**
21	**R1**	**electric_stove**	**6**	**0**	**0**	**0**	**0**	**0**	**3**
22	**R1**	**electric_stove**	**7**	**0**	**0**	**0**	**0**	**0**	**2**
23	**R1**	**electric_stove**	**8**	**0**	**0**	**0**	**0**	**0**	**3**
24	**R2**	**electric_stove**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
25	**R2**	**electric_stove**	**2**	**0**	**0**	**0**	**0**	**0**	**1**
26	**R2**	**electric_stove**	**3**	**0**	**0**	**0**	**0**	**0**	**1**
27	**R2**	**electric_stove**	**4**	**0**	**0**	**0**	**0**	**0**	**1.5**
28	**R2**	**electric_stove**	**5**	**0**	**0**	**0**	**0**	**0**	**2**
29	**R2**	**electric_stove**	**6**	**0**	**0**	**0**	**0**	**0**	**2**
30	**R2**	**electric_stove**	**7**	**0**	**0**	**0**	**0**	**0**	**2.5**
31	**R2**	**electric_stove**	**8**	**0**	**0**	**0**	**0**	**0**	**2**

For the purposes of brevity, we omitted the majority of the `gasboiler` entries. However, these remain unchanged, apart from a `0` entry in the cook column to indicate that a `gasboiler` does not meet `cook` demand.

We added an `electric_stove` process for each of the timeslices, which meets the `cook` demand. This can be seen through the addition of a positive number in the `cook` column.

The process is very similar for the `Residential2050Consumption.csv` file, however, for this example, we often placed larger numbers to indicate higher demand in 2050. For the complete file see the link here INCLUDE LINK HERE

Next, we must edit the files within the `input` folder. For this, we must add the `cook` service demand to each of these files.

First, we will amend the `BaseYearExport.csv` and `BaseYearImport.csv` files. For this, we say that there is no import or export of the `cook` service demand. A brief example is outlined below for `BaseYearExport.csv`:

RegionName	Attribute	Time	electricity	gas	heat	CO2f	wind	solar	**cook**
Unit	•	Year	PJ	PJ	PJ	kt	PJ	PJ	**PJ**
R1	Exports	2010	0	0	0	0	0	0	**0**
…	…	…	…	…	…	…	…	…	**…**
R2	Exports	2100	0	0	0	0	0	0	**0**

The same is true for the `BaseYearImport.csv` file:

RegionName	Attribute	Time	electricity	gas	heat	CO2f	wind	solar	**cook**
Unit	•	Year	PJ	PJ	PJ	kt	PJ	PJ	**PJ**
R1	Imports	2010	0	0	0	0	0	0	**0**
…	…	…	…	…	…	…	…	…	**…**
R2	Imports	2100	0	0	0	0	0	0	**0**

Next, we must edit the `GlobalCommodities.csv` file. This is where we define the new commodity `cook`. It tells MUSE the commodity type, name, emissions factor of CO2 and heat rate, amongst other things.

The example used for this tutorial is below:

Commodity	Commodity-Type	Commodity-Name	CommodityEmissionFac-tor_CO2	HeatRate	Unit
Electricity	Energy	electricity	0	1	PJ
Gas	Energy	gas	56.1	1	PJ
Heat	Energy	heat	0	1	PJ
Wind	Energy	wind	0	1	PJ
CO2fuelcomsbustion	Environmental	CO2f	0	1	kt
Solar	Energy	solar	0	1	PJ
Cook	**Energy**	**cook**	**0**	**1**	**PJ**

Finally, the `Projections.csv` file must be changed. This is a large file which details the expected cost of the technology in the first year of the simulation. Due to its size, we will only show two rows of the new column `cook`.

RegionName	Attribute	Time	…	**cook**
Unit	•	Year	…	**MUS$2010/kt**
R1	CommodityPrice	2010	…	**100**
…	…	…	…	**…**
R2	CommodityPrice	2100	…	**100**

We set every price of cook to be `100MUS$2010/kt`

5.5.2 Addition of cooking technology

Next, we must add a technology to service this new demand. This is achieved through a similar process as the section in the *"adding a new technology"* section. However, we must be careful to specify the end-use of the technology as `cook`.

For this example, we will add two competing technologies to service the cooking demand: `electric_stove` and `gas_stove` to the `Technodata.csv` file in `/technodata/residential/Technodata.csv`.

Again for the interests of space, we have omitted the existing `gasboiler` and `heatpump` technologies. But we copy the `gasboiler` row for `R1` and paste it for the new `electric_stove` for both `R1` and `R2`. For `gas_stove` we copy and paste the data for `heatpump` from region `R1` for both `R1` and `R2`.

An important modification, however, is specifying the end-use for these new technologies to be `cook` and not `heat`.

ProcessName	RegionName	Time	Level	cap_par	…	Fuel	EndUse	Agent2	Agent1
Unit	•	Year	•	MUS$2010/PJ_a		•	•	Retrofit	New
gasboiler	R1	2020	fixed	3.8	…	gas	heat	1	0
…	…	…	…	…	…	…	…	…	…
electric_stove	**R1**	**2020**	**fixed**	**3.8**	**…**	**electricity**	**cook**	**1**	**0**
electric_stove	**R2**	**2020**	**fixed**	**3.8**	**…**	**electricity**	**cook**	**1**	**0**
gas_stove	**R1**	**2020**	**fixed**	**8.8667**	**…**	**gas**	**cook**	**1**	**0**
gas_stove	**R2**	**2020**	**fixed**	**8.8667**	**…**	**gas**	**cook**	**1**	**0**

As can be seen we have added two technologies, in the two regions with different `cap_par` costs. We specified their respective fules, and the enduse for both is `cook`. For the full file please see here INSERT LINK HERE.

We must also add the data for these new technologies to the following files:

- `CommIn.csv`
- `CommOut.csv`
- `ExistingCapacity.csv`

This is largely a similar process to the tutorial shown in *"adding a new technology"*. We must add the input to each of the technologies (gas and electricity for `gas_stove` and `electric_stove` respectively), outputs of `cook` for both and the existing capacity for each technology in each region.

Due to the additional demand for gas and electricity brought on by the new `cook` demand, it is necessary to relax the growth constraints for `gassupply1` in the `technodata/gas/technodata.csv` file. For this example, we set this file as follows:

ProcessName	RegionName	Time	…	MaxCapacityAddition	MaxCapacityGrowth	TotalCapacityLimit	…	Agent1
Unit	•	Year	…	PJ	%	PJ	…	New
gassupply1	R1	2020	…	**100**	**5**	**500**	…	0
gassupply1	R2	2020	…	**100**	**5**	**120**	…	0

To prevent repetition of the *"adding a new technology"* section, we will leave the full files here INSERT LINK HERE.

Again, we run the simulation with our modified input files using the following command, in the relevant directory:

```
python -m pip muse settings.toml
```

Once this has run we are ready to visualise our results.

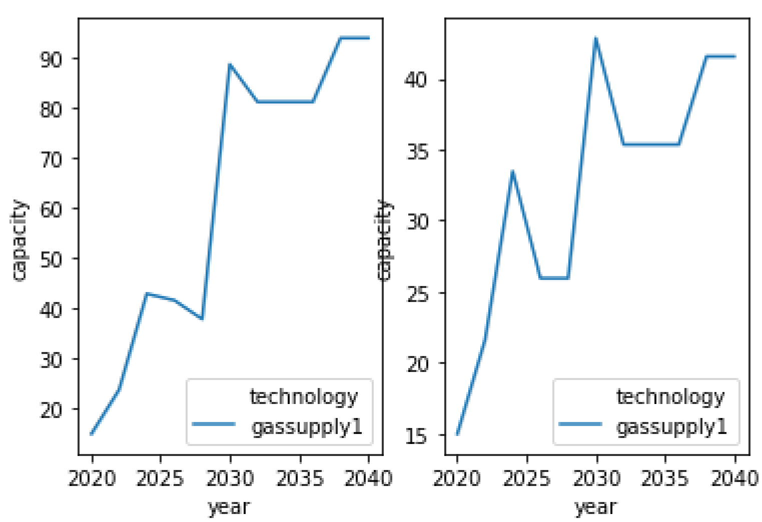
```
[2]: import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
```

```
[3]: mca_capacity = pd.read_csv("../tutorial-code/add-service-demand/Results/MCACapacity.
     ↪csv")
     mca_capacity.head()
```

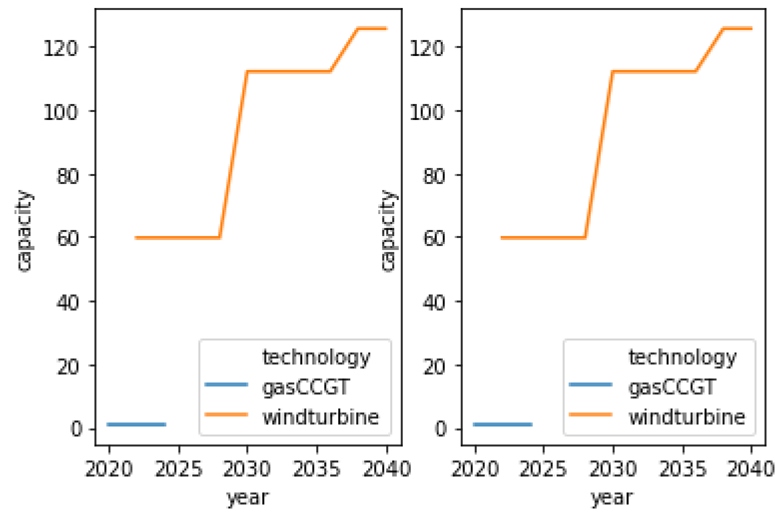
```
[3]:   technology region agent      type       sector  capacity  year
     0  gas_stove     R1    A1  retrofit  residential      10.0  2020
     1  gasboiler     R1    A1  retrofit  residential      10.0  2020
     2  gas_stove     R2    A1  retrofit  residential      10.0  2020
     3  gasboiler     R2    A1  retrofit  residential      10.0  2020
     4  gas_stove     R1    A2  retrofit  residential      10.0  2020
```

```
[4]: for name, sector in mca_capacity.groupby("sector"):
         print("{} sector:".format(name))
         fig, ax =plt.subplots(1,2)
         sns.lineplot(data=sector[sector.region=="R1"], x="year", y="capacity", hue=
     ↪"technology", ax=ax[0])
         sns.lineplot(data=sector[sector.region=="R2"], x="year", y="capacity", hue=
     ↪"technology", ax=ax[1])
         plt.show()
         plt.close()
```

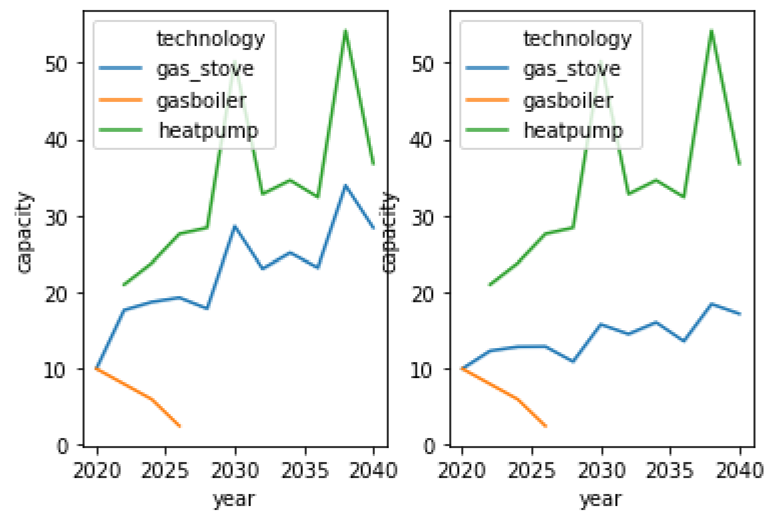
```
gas sector:
```




```
power sector:
```




```
residential sector:
```



We can see our new technology, the `gas_stove` is used over the `electric_stove`. Therefore, there is an increase in `gassupply1` to accommodate for this growth in demand. However, this is not enough to displace `windturbine` by `gasCCGT`.

5.5.3 Next steps

This brings us to the end of the user guide! Using the information explained in this tutorial, or following similar steps, you will be able to create complex scenarios of your choosing.

For the full code to generate the final results, see here INSERT LINK HERE.

CHAPTER

SIX

INPUT FILES

In this section we detail each of the files required to run MUSE. We include information based on how these files should be used, as well as the data that populates them.

6.1 TOML primer

The full specification for TOML files can be found here. A TOML file is separated into sections, with each section except the topmost introduced by a name in square brackets. Sections can hold key-value pairs, e.g. a name associated with a value. For instance:

```
general_string_attribute = "x"

[some_section]
section_attribute = 12

[some_section.subsection]
subsetion_attribute = true
```

TOML is quite flexible in how one can define sections and attributes. The following three examples are equivalent:

```
[sectors.residential.production]
name = "match"
costing = "prices"
```

```
[sectors.residential]
production = {"name": "match", "costing": "prices"}
```

```
[sectors.residential]
production.name = "match"
production.costing = "prices"
```

Additionally, TOML files can contain tabular data, specified row-by-row using double square bracket. For instance, below we define a table with two rows and a single *column* called *some_table_of_data* (though column is not quite the right term, TOML tables are made more flexible than most tabular formats. Rather, each row can be considered a dictionary).

```
[[some_table_of_data]]
a_key = "a value"

[[some_table_of_data]]
a_key = "another value"
```

As MUSE requires a number of data file, paths to files can be formated in a flexible manner. Paths can be formatted with shorthands for specific directories and are defined with curly-brackets. For example:

```
projection = '{path}/inputs/projection.csv'
timeslices_path = '{cwd}/technodata/timeslices.csv'
consumption_path = '{muse_sectors}/technodata/timeslices.csv'
```

path refers to the directory where the TOML file is located

cwd refers to the directory from which the muse simulation is launched

muse_sectors refers to the directory where default sectoral data is located

6.2 Simulation settings

This section details the TOML input for MUSE. The format for TOML files is described in this *previous section*. Here, however, we focus on sections and attributes that are specific to MUSE.

The TOML file can be read using `read_settings()`. The resulting data is used to construt the market clearing algorithm directly in the `MCA's factory function`.

6.2.1 Main section

This is the topmost section. It contains settings relevant to the simulation as a whole.

```
time_framework = [2020, 2025, 2030, 2035, 2040, 2045, 2050]
regions = ["USA"]
interpolation_mode = 'Active'
log_level = 'info'

equilibrium_variable = 'demand'
maximum_iterations = 100
tolerance = 0.1
tolerance_unmet_demand = -0.1
```

time_framework Required. List of years for which the simulation will run.

region Subset of regions to consider. If not given, defaults to all regions found in the simulation data.

interpolation_mode interpolation when reading the initial market. One of *linear*, *nearest*, *zero*, *slinear*, *quadratic*, *cubic*. Defaults to *linear*.

log_level: verbosity of the output.

equilibirum_variable whether equilibrium of *demand* or *prices* should be sought. Defaults to *demand*.

maximum_iterations Maximum number of iterations when searching for equilibrium. Defaults to 3.

tolerance Tolerance criteria when checking for equilibrium. Defaults to 0.1.

tolerance_unmet_demand Criteria checking whether the demand has been met. Defaults to -0.1.

excluded_commodities List of commodities excluded from the equilibrium considerations. Defaults to the list *[“CO2f”, “CO2r”, “CO2c”, “CO2s”, “CH4”, “N2O”, “f-gases”]*.

plugins Path or list of paths to extra python plugins, i.e. files with registered functions such as `register_output_quantity()`.

6.2.2 Carbon market

This section containts the settings related to the modelling of the carbon market. If omitted, it defaults to not including the carbon market in the simulation.

Example

```
[carbon_budget_control]
budget = []
```

budget Yearly budget. There should be one item for each year the simulation will run. In other words, if given and not empty, this is a list with the same length as *time_framework* from the main section. If not given or an empty list, then the carbon market feature is disabled. Defaults to an empty list.

method Method used to equilibrate the carbon market. Defaults to a simple iterative scheme. [INSERT OPTIONS HERE]

commodities Commodities that make up the carbon market. Defaults to an empty list.

control_undershoot Whether to control carbon budget undershoots. Defaults to True.

control_overshoot Whether to control carbon budget overshoots. Defaults to True.

method_options: Additional options for the specific carbon method.

6.2.3 Global input files

Defines the paths specific simulation data files. The paths can be formatted as explained in the *TOML primer*.

```
[global_input_files]
projections = '{path}/inputs/Projections.csv'
regions = '{path}/inputs/Regions.csv'
global_commodities = '{path}/inputs/MUSEGlobalCommodities.csv'
```

projections: Path to a csv file giving initial market projection. See *Initial Market Projection*.

regions: Path to a csv file describing the regions. See *Regional data*.

global_commodities: Path to a csv file describing the comodities in the simulation. See *Commodity Description*.

6.2.4 Timeslices

Time-slices represent a sub-year disaggregation of commodity demand. Generally, timeslices are expected to introduce several levels, e.g. season, day, or hour. The simplest is to show the TOML for the default timeslice:

```
[timeslices]
winter.weekday.night = 396
winter.weekday.morning = 396
winter.weekday.afternoon = 264
winter.weekday.early-peak = 66
winter.weekday.late-peak = 66
winter.weekday.evening = 396
winter.weekend.night = 156
winter.weekend.morning = 156
winter.weekend.afternoon = 156
winter.weekend.evening = 156
spring-autumn.weekday.night = 792
```

(continues on next page)

(continued from previous page)

```
spring-autumn.weekday.morning = 792
spring-autumn.weekday.afternoon = 528
spring-autumn.weekday.early-peak = 132
spring-autumn.weekday.late-peak = 132
spring-autumn.weekday.evening = 792
spring-autumn.weekend.night = 300
spring-autumn.weekend.morning = 300
spring-autumn.weekend.afternoon = 300
spring-autumn.weekend.evening = 300
summer.weekday.night = 396
summer.weekday.morning  = 396
summer.weekday.afternoon = 264
summer.weekday.early-peak = 66
summer.weekday.late-peak = 66
summer.weekday.evening = 396
summer.weekend.night = 150
summer.weekend.morning = 150
summer.weekend.afternoon = 150
summer.weekend.evening = 150
level_names = ["month", "day", "hour"]
```

This input introduces three levels, via `level_names`: `month`, `day`, `hours`. Other simulations may want fewer or more levels. The `month` level is split into three points of data, `winter`, `spring-autumn`, `summer`. Then `day` splits out weekdays from weekends, and so on. Each line indicates the number of hours for the relevant slice. It should be noted that the slices are not a cartesian products of each levels. For instance, there no `peak` periods during weekends. All that matters is that the relative weights (i.e. the number of hours) are consistent and sum up to a year.

The input above defines the finest times slice in the code. In order to define rougher timeslices we can introduce items in each levels that represent aggregates at that level. By default, we have the following:

```
[timeslices.aggregates]
all-day = ["night", "morning", "afternoon", "early-peak", "late-peak", "evening"]
all-week = ["weekday", "weekend"]
all-year = ["winter", "summer", "spring-autumn"]
```

Here, `all-day` aggregates the full day. However, one could potentially create aggregates such as:

```
[timeslices.aggregates]
daylight = ["morning", "afternoon", "early-peak", "late-peak"]
nightlife = ["evening", "night"]
```

Once the finest timeslice and its aggregates are given, it is possible for each sector to define the timeslice simply by refering to the slices it will use at each level.

```
[sectors.some_sector.timeslice_levels]
day = ["daylight", "nightlife"]
month = ["all-year"]
```

Above, `sectors.some_sector.timeslice_levels.week` defaults its value in the finest timeslice. Indeed, if the subsection `sectors.some_sector.timeslice_levels` is not given, then the sector will default to using the finest timeslices.

Similarly, it is possible to specify a timeslice for the mca by adding an *mca.timeslice_levels* section. However, be aware that if the MCA uses a rougher timeslice framework, the market will be expressed within it. Hence information from sectors with a finer timeslice framework will be lost.

6.2.5 Standard sectors

Sectors are declared in the TOML file by adding a subsection to the *sectors* section:

```
[sectors.residential]
type = 'default'
[sectors.power]
type = 'default'
```

Above, we've added two sectors, residential and power. The name of the subsection is only used for identification. In other words, it should be chosen to be meaningful to the user, since it will not affect the model itself.

Sectors are defined in `Sector`.

A sector accepts a number of attributes and subsections.

type Defines the kind of sector this is. *Standard* sectors are those with type "default". This value corresponds to the name with which a sector class is registerd with MUSE, via `register_sector()`. [INSERT OTHER OPTIONS HERE]

priority An integer denoting which sectors runs when. Lower values imply the sector will run earlier. If two sectors share the same priority. Later sectors can depend on earlier sectors for the their input. If two sectors share the same priority, then their order is not defined. Indeed, it should indicate that they can run in parallel. For simplicity, the keyword also accepts standard values:

- "preset": 0
- "demand": 10
- "conversion": 20
- "supply": 30
- "last": 100

Defaults to "last".

interpolation Interpolation method user when filling in missing values. Available interpolation methods depend on the underlying scipy method's kind attribute.

investment_production In its simplest form, this is the name of a method to compute the production from a sector, as used when splitting the demand across agents. In other words, this is the computation of the production which affects future investments. In it's more general form, *production* can be a subsection of its own, with a "name" attribute. For instance:

```
[sectors.residential.production]
name = "match"
costing = "prices"
```

MUSE provides two methods in `muse.production`:

- **share: the production is the maximum production for the existing capacity and** the technology's utilization factor. See `muse.production.maximum_production()`.
- **match: production and demand are matched according to a given cost metric. The** cost metric defaults to "prices". It can be modified by using the general form given above, with a "costing" attribute. The latter can be "prices", "gross_margin", or "lcoe". See `muse.production.demand_matched_production()`.

production can also refer to any custom production method registered with MUSE via `muse.production.register_production()`.

Defaults to "share".

dispatch_production The name of the production method used to compute the sector's output, as returned to the muse market clearing algorithm. In other words, this is computation of the production method which will affect other sectors.

It has the same format and options as the *production* attribute above.

demand_share A method used to split the MCA demand into seperate parts to be serviced by specific agents. There is currently only one option, "new_and_retro", corresponding to *new* and *retro* agents.

interactions Defines interactions between agents. These interactions take place right before new investments are computed. The interactions can be anything. They are expected to modify the agents and their assets. MUSE provides a default set of interactions that have *new* agents pass on their assets to the corresponding *retro* agent, and the *retro* agents pass on the make-up of their assets to the corresponding *new* agents.

interactions are specified as a *TOML array*, e.g. with double brackets. Each sector can specify an arbitrary number of interactaction, simply by adding an extra interaction row.

There are two orthogonal concepts to interactions:

- a *net* defines the set of agents that interact. A set can contain any number of agents, whether zero, two, or all agents in a sector. See `muse.interactions.register_interaction_net()`.
- an *interaction* defines how the net actually interacts. See `muse.interactions.register_agent_interaction()`.

In practice, we always consider sequences of nets (i.e. more than one net) that interact using the same interaction function.

Hence, the input looks something like the following:

```
[[sectors.commercial.interactions]]
net = 'new_to_retro'
interaction = 'transfer'
```

"new_to_retro" is a function that figures out all "new/retro" pairs of agents. Whereas "transfer" is a function that performs the transfer of assets and information between each pair.

Furthermore, it is possible to pass parameters to either the net of the interaction as follows:

```
[[sectors.commercial.interactions]]
net = {"name": "some_net", "param": "some value"}
interaction = {"name": "some_interaction", "param": "some other value"}
```

The parameters will depend on the net and interaction functions. Neither "new_to_retro" nor "transfer" take any arguments at this point. MUSE interaction facilities are defined in `muse.interactions`.

output Outputs are made up of several components. MUSE is designed to allow users to mix-and-match both how and what to save.

output is specified as a TOML array, e.g. with double brackets. Each sector can specify an arbitrary number of outputs, simply by adding an extra output row.

A single row looks like this:

```
[[sectors.commercial.outputs]]
filename = '{cwd}/Results/{Sector}/{Quantity}/{year}{suffix}'
quantity = "capacity"
sink = 'csv'
overwrite = true
```

The following attributes are available:

- **quantity: Name of the quantity to save. Currently, only *capacity* exists,** refering to `muse.outputs.capacity()`. However, users can customize and create further output quantities by registering with MUSE via `muse.outputs.register_output_quantity()`. See `muse.outputs` for more details.
- **sink: the sink is the place (disk, cloud, database, etc. . .) and format with which** the computed quantity is saved. Currently only sinks that save to files are implemented. The filename can specified via *filename*, as given below. The following sinks are available: "csv", "netcfd", "excel". However, more sinks can be added by interested users, and registered with MUSE via `muse.outputs.register_output_sink()`. See `muse.outputs` for more details.
- **filename: defines the format of the file where to save the data. There are several** standard values that are automatically substituted:
  - cwd: current working directory, where MUSE was started
  - path: directory where the TOML file resides
  - sector: name of the current sector (.e.g. "commercial" above)
  - Sector: capitalized name of the current sector
  - quantity: name of the quantity to save (as given by the quantity attribute)
  - Quantity: capitablized name of the quantity to save
  - year: current year
  - suffix: standard suffix/file extension of the sink

  Defaults to *{cwd}/{default_output_dir}/{Sector}/{Quantity}/{year}{suffix}*.
- **overwrite: If *False* MUSE will issue an error and abort, instead of** overwriting an existing file. Defaults to *False*. This prevents important output files from being overwritten.

technodata Path to a csv file containing the characterization of the technologies involved in the sector, e.g. lifetime, capital costs, etc. . . See *Techno-data*.

timeslice_levels Slices to consider in a level. If absent, defaults to the finest timeslices. See *Timeslices*

commodities_in Path to a csv file describing the inputs of each technology involved in the sector. See *General features*.

commodities_out Path to a csv file describing the outputs of each technology involved in the sector. See *Output Commodities*.

existing_capacity Path to a csv file describing the initial capacity of the sector. See *Existing Sectoral Capacity*.

agents Path to a csv file describing the agents in the sector. See *Agents*.

6.2.6 Preset sectors

The commodity production, commodity consumption and product prices of preset sectors are determined exogeneously. They are know from the start of the simulation and are not affected by the simulation.

Preset sectors are defined in `PresetSector`.

The three components, production, consumption, and prices, can be set independantly and not all three need to be set. Production and consumption default to zero, and prices default to leaving things unchanged.

The following defines a standard preset sector where consumption is defined as a function of macro-economic data, i.e. population and gdp.

```
[sectors.commercial_presets]
type = 'presets'
priority = 'presets'
timeslice_shares_path = '{path}/technodata/TimesliceShareCommercial.csv'
macrodrivers_path = '{path}/technodata/Macrodrivers.csv'
regression_path = '{path}/technodata/regressionparameters.csv'
timeslices_levels = {'day': ['all-day']}
forecast = [0, 5]
```

The following attributes are accepted:

type: See the attribute in the standard mode, type. *Preset* sectors are those with type "presets".

priority See the attribute in the standard mode, priority.

timeslices_levels: See the attribute in the standard mode, *Timeslices*.

consumption_path: CSV output files, one per year. This attribute can include wild cards, i.e. '*', which can match anything. For instance: *consumption_path = "{cwd}/Consumtion*.csv"* will match any csv file starting with "Consumption" in the current working directory. The file names must include the year for which it defines the consumption, e.g. *Consumption2015.csv*.

The CSV format should follow the following format:

Table 1: Consumption

	RegionName	ProcessName	TimeSlice	electricity	diesel	algae
0	USA	fluorescent light	1	1.9	0	0
1	USA	fluorescent light	2	1.8	0	0

The index column as well as "RegionName", "ProcessName", and "TimeSlice" must be present. Further columns are reserved for commodities. "TimeSlice" refers to the index of the timeslice.

supply_path: CSV file, one per year, indicating the amount of a commodities produced. It follows the same format as consumption_path.

supply_path: CSV file, one per year, indicating the amount of a commodities produced. It follows the same format as consumption_path.

prices_path: CSV file indicating the amount of a commodities produced. The format of the CSV files follows that of *Initial Market Projection*.

demand_path: Incompatible with consumption_path or macrodrivers_path. A CSV file containing the consumption in the same format as *Initial Market Projection*.

macrodrivers_path: Incompatible with consumption_path or demand_path. Path to a CSV file giving the profile of the macrodrivers. Also requires regression_path.

regression_path: Incompatible with consumption_path or demand_path. Path to a CSV file giving the regression parameters with respect to the macrodrivers. Also requires macrodrivers_path.

timeslice_shares_path Optional csv file giving shares per timeslice. Requires macrodrivers_path.

filters: Optional dictionary of entries by which to filter the consumption. Requires macrodrivers_path. For instance,

```
filters.region = ["USA", "ASEA"]
filters.commodity = ["algae", "fluorescent light"]
```

6.2.7 Legacy Sectors

Legacy sectors wrap sectors developed for a previous version of MUSE to the open-source version.

Preset sectors are defined in `PresetSector`.

The can be defined in the TOML file as follows:

```
[global_input_files]
macrodrivers = '{path}/input/Macrodrivers.csv'
regions = '{path}/input/Regions.csv'
global_commodities = '{path}/input/MUSEGlobalCommodities.csv'

[sectors.Industry]
type = 'legacy'
priority = 'demand'
agregation_level = 'month'
excess = 0

userdata_path = '{muse_sectors}/Industry'
technodata_path = '{muse_sectors}/Industry'
timeslices_path = '{muse_sectors}/Industry/TimeslicesIndustry.csv'
output_path = '{path}/output'
```

For historical reasons, the three *global_input_files* above are required. The sector itself can use the following attributes.

type: See the attribute in the standard mode, type. *Legacy* sectors are those with type "legacy".

priority See the attribute in the standard mode, priority.

agregation_level: Information relevant to the sector's timeslice.

excess: Excess factor used to model early obsolescence.

timeslices_path: Path to a timeslice *time_slices*.

userdata_path: Path to a directory with sector-specific data files.

technodata_path: Path to a technodata CSV file. See. *Techno-data*.

output_path: Path to a diretory where the sector will write output files.

6.3 Input Files

6.3.1 Initial Market Projection

MUSE needs an initial projection of the market prices for each period of the simulation.

- The price trajectory is needed if the MCA works in *equilibrium* mode as an initial trajectory for the base year of the simulation. The market will override the calculated prices obtained from each commodity equilibrium for all the future periods following the base year
- Similarly, if the market works in a *carbon budget* mode, the prices are used as a starting point. The only difference from the previous case is given by the fact that the MCA will be calculating an additional global market price for carbon dioxide (and additional pollutants if required)
- If the MCA works in an *exogenous* mode, it will use the initial market projection as the projection for the the base year and all the future periods of the simulation

The forward price trajectory should follow the structure reported in the table below.

Table 2: Initial market projections

RegionName	Attribute	Time	com1	com2	com3
Unit	•	Year	MUS$2010/PJ	MUS$2010/PJ	MUS$2010/PJ
region1	CommodityPrice	2010	20	1.9583	2
region1	CommodityPrice	2015	20	1.9583	2
region1	CommodityPrice	2020	20.38518042	1.996014941	2.038518042
region1	CommodityPrice	2025	20.77777903	2.034456234	2.077777903
region1	CommodityPrice	2030	21.17793872	2.073637869	2.117793872
region1	CommodityPrice	2035	21.58580508	2.113574105	2.158580508
region1	CommodityPrice	2040	22.00152655	2.154279472	2.200152655
region1	CommodityPrice	2045	22.42525441	2.195768786	2.242525441
region1	CommodityPrice	2050	22.85714286	2.238057143	2.285714286

RegionName represents the region ID and needs to be consistent across all the data inputs

Attribute defines the attribute type. In this case it refers to the CommodityPrice; it is relevant only for internal use

Time corresponds to the time periods of the simulation; the simulated time framework in the example goes from 2010 through to 2050 with a 5-year time step

com1, . . . , comN Any further columns represent the commodities modelled, as defined in the global commodities the row Unit reports the unit in which the technology consumption is defined; it is for the user internal reference only. The names *comX* should be replaced with the names of the commodities.

6.3.2 Regional data

MUSE requires the definition of the methodology used for investment and dispatch and alias demand matching. The methodology has to be defined by region and subregion, meant as a geographical subdivision in a region. Currently, the methodology definition is important for the legacy sectors only.

Below the generic structure of the input commodity file for the electric heater is shown:

Table 3: Methodology used in investment and demand matching

SectorName	RegionName	Subregion	sMethodologyPlanning	sMethodologyDispatch
Agriculture	region1	region1	NPV	DCF
Bioenergy	region1	region1	NPV	DCF
Industry	region1	region1	NPV	DCF
Residential	region1	region1	EAC	EAC
Commercial	region1	region1	EAC	EAC
Transport	region1	region1	LCOE	LCOE
Power	region1	region1	LCOE	LCOE
Refinery	region1	region1	LCOE	LCOE
Supply	region1	region1	LCOE	LCOE

SectorName represents the sector_ID and needs to be consistent across the data input files

RegionName represents the region ID and needs to be consistent across all the data inputs

Subregion represents the subregion ID and needs to be consistent across all the data inputs

sMethodologyPlanning reports the cost quantity used for making investments in new technologies in each sector (e.g. NPV stands for net present value, EAC stands for equivalent annual costs, LCOE stands for levelised cost of energy)

sMethodologyDispatch reports the cost quantity used for the demand matching using existing technologies in each sector (e.g. DCF stands for discounted cash flow, EAC stands for equivalent annual cost, LCOE stands for levelised cost of energy)

6.3.3 Commodity Description

MUSE handles a configurable number and type of commodities which are primarily used to represent energy, services, pollutants/emissions. The commodities for the simulation as a whole are defined in a csv file with the following structure.

Table 4: Global commodities

Commodity	Commodity-Type	Commodity-Name	CommodityEmissionFac-tor_CO2	HeatRate	Unit
Coal	Energy	hardcoal	94.6	29	PJ
Agricultural-residues	Energy	agrires	112	15.4	PJ

Commodity represents the extended name of a commodity

CommodityType defines the type of a commodity (i.e. energy, material or environmental)

CommodityName is the internal name used for a commodity inside the model.

CommodityEmissionFactor_CO2 is CO2 emission per unit of commodity flow

HeatRate represents the lower heating value of an energy commodity

Unit is the unit used as a basis for all the input data. More specifically the model allows a totally flexible way of defining the commodities. CommodityName is currently the only column used internally as it defines the names of commodities and needs to be used consistently across all the input data files. The remaining columns of the file are only relevant for the user internal reference for the original sets of assumptions used.

6.3.4 Techno-data

The techno-data includes the techno-economic characteristics of each technology such as capital, fixed and variable cost, lifetime, utilisation factor. The techno-data should follow the structure reported in the table. The column order is not important and additional input data can alsobe read in this format. In the table, the electric boiler used in households is taken as an example for a generic region, region1.

Table 5: Techno-data

ProcessName	RegionName	Time	Level	cap_par	cap_exp	fix_par	…
resBoilerElectric	region1	2010	fixed	3.81	1.00	0.38	…
resBoilerElectric	region1	2030	fixed	3.81	1.00	0.38	…

ProcessName represents the technology ID and needs to be consistent across all the data inputs

RegionName represents the region ID and needs to be consistent across all the data inputs

Time represents the period of the simulation to which the value applies; it needs to contain at least the base year of the simulation

Level characterises either a fixed or a flexible input type

cap_par, cap_exp are used in the capital cost estimation. Capital costs are calculated as:

$$\text{CAPEX} = \text{cap\_par} * (\text{Capacity})^{\text{cap\_exp}}$$

where the parameter cap_par is estimated at a selected reference size (i.e. Capref), such as:

$$\text{cap\_par} = \left(\frac{\text{CAPEXref}}{\text{Capref}}\right)^{\text{cap\_exp}}$$

Capref is decided by the modeller before filling the input data files.

This allows the model to take into account economies of scale. ie. As *Capacity* increases, the price of the technology decreases.

fix_par, fix_exp

are used in the fixed cost estimation. Fixed costs are calculated as:

$$\text{FOM} = \text{fix\_par} * (\text{Capacity})^{\text{fix\_exp}}$$

where the parameter fix_par is estimated at a selected reference size (i.e. Capref), such as:

$$\text{fix\_par} = \left(\frac{\text{FOMref}}{\text{Capref}}\right)^{\text{fix\_exp}}$$

Capref is decided by the modeller before filling the input data files.

var_par, var_exp are used in the variable costs estimation. These variable costs are capacity dependent Variable costs are calculated as:

$$\text{VAREX} = \text{cap\_par} * (\text{Capacity})^{\text{cap\_exp}}$$

where the parameter var_par is estimated at a selected reference size (i.e. Capref), such as:

$$\text{var\_par} = \left(\frac{\text{VARref}}{\text{Capref}}\right)^{\text{var\_exp}}$$

Capref is decided by the modeller before filling the input data files.

MaxCapacityAddition represents the maximum addition of installed capacity per technology, region, year.

MaxCapacityGrowth represents the maximum growth in capacity as a fraction of the installed capacity per technology, region and year.

TotalCapacityLimit represents the total capacity limit per technology, region and year.

TechnicalLife represents the number of years that a technology operates before it is decommissioned.

UtilizationFactor is the number of operating hours of a process over the maximum number of hours in a year.

ScalingSize represents the minimum size of a technology to be installed.

efficiency is calculated as the ratio between the total output commodities and the input commodities.

AvailabiliyYear defines the starting year of a technology; for example the value equals 1 when a technology would be available or 0 when a technology would not be available.

Type defines the type of a technology.

Fuel defines the fuel used by a technology.

EndUse defines the end use of a technology.

InterestRate is the technology interest rate.

Agent_0, . . . , Agent_N represent the allocation of the initial capacity to the each agent.

The input data has to be provided for the base year. Additional years within the time framework of the overall simulation can be defined. In this case, MUSE would interpolate the values between the provided periods and assume a constant value afterwards.

6.3.5 Time-slices

Note: This input file is only for legacy sectors. For anything else, please see simulation-settings.

Time-slices represent a sub-year disaggregation of commodity demand. They are fully flexible in number and names as to serve the specific representation of the commodity demand, supply, and supply cost profile in each energy sector. Each time slice is independent in terms of the number of represent hours, as long as it is meaningful for the users and their data inputs. 1 is the minimum number of time-slice as this would correspond to a full year. The time-slice definition of a sector affects the commodity price profile and the supply cost profile.

The csv file for the time-slice definition would report the length (in hours) of each time slice as characteristic to the selected sector to represent diurnal, weekly and seasonal variation of energy commodities, demand and supply, as shown in the table for 30 time-slices.

Table 6: Time-slices

AgLevel	SN	Month	Day	Hour	RepresentHours
Hour	1	Winter	Weekday	Night	396
Hour	2	Winter	Weekday	Morning	396
Hour	3	Winter	Weekday	Afternoon	264
Hour	4	Winter	Weekday	EarlyPeak	66
Hour	5	Winter	Weekday	LatePeak	66
Hour	6	Winter	Weekday	Evening	396
Hour	7	Winter	Weekend	Night	156
Hour	8	Winter	Weekend	Morning	156
Hour	9	Winter	Weekend	Afternoon	156
Hour	10	Winter	Weekend	Evening	156
Hour	11	SpringAutumn	Weekday	Night	792
Hour	12	SpringAutumn	Weekday	Morning	792
Hour	13	SpringAutumn	Weekday	Afternoon	528
Hour	14	SpringAutumn	Weekday	EarlyPeak	132
Hour	15	SpringAutumn	Weekday	LatePeak	132
Hour	16	SpringAutumn	Weekday	Evening	792
Hour	17	SpringAutumn	Weekend	Night	300
Hour	18	SpringAutumn	Weekend	Morning	300
Hour	19	SpringAutumn	Weekend	Afternoon	300
Hour	20	SpringAutumn	Weekend	Evening	300
Hour	21	Summer	Weekday	Night	396
Hour	22	Summer	Weekday	Morning	396
Hour	23	Summer	Weekday	Afternoon	264
Hour	24	Summer	Weekday	EarlyPeak	66
Hour	25	Summer	Weekday	LatePeak	66
Hour	26	Summer	Weekday	Evening	396
Hour	27	Summer	Weekend	Night	150
Hour	28	Summer	Weekend	Morning	150

continues on next page

Table 6 – continued from previous page

AgLevel	SN	Month	Day	Hour	RepresentHours
Hour	29	Summer	Weekend	Afternoon	150
Hour	30	Summer	Weekend	Evening	150

It reports the aggregation level of the sector time-slices (AgLevel), slice number (SN), seasonal time slices (Month), weekly time slices (Day), hourly profile (Hour), the amount of hours associated to each time slice (RepresentHours).

6.3.6 Input Commodities

Input commodities are the commodities consumed (also called consumables in MUSE) by each technology. They are defined in a csv file which describes the commodity inputs to each technology, calculated per unit of technology activity. See *below* for a description.

6.3.7 Output Commodities

Output commodities are the commodities produced (also called products in MUSE) by each technology. They are defined in a csv file which describes the commodity outputs from each technology, defined per unit of technology activity. Emissions, such as CO2 (produced from fuel combustion and reactions), CH4, N2O, F-gases, can also be accounted for in this file. See *below* for a description.

6.3.8 General features

To illustrate the data required for a generic technology in MUSE, the *electric boiler technology* is used as an example. The commodity flow for the electric boiler, capable to cover space heating and water heating energy service demands.

Fig. 1: The table below shows the basic data requirements for a typical technology, the electric boiler.

Technology: Electric Boiler ProcessName: resBoilerElectric	*Values*	*Units (input commodity unit/process activity unit)*
electricity	1.0	GWh/PJ
Output commodity	*Values*	*Unit (output commodity unit/process activity unit)*
Space cooling	0	PJ/PJ
Space heating	0.80	PJ/PJ
Water heating	0.20	PJ/PJ
Appliances	0	PJ/PJ
Lighting	0	PJ/PJ
Cooking	0	PJ/PJ
CO_2 from reaction	0	kt/PJ
CO_2 from combustion	0	kt/PJ
CO_2 captured	0	kt/PJ
CO_2 stored	0	kt/PJ
CH_4	0	kt/PJ
N_2O	0	kt/PJ
F-gases	0	kt/PJ

Below it is shown the generic structure of the input commodity file for the electric heater.

Table 7: Commodities used as consumables - Input commodities

ProcessName	RegionName	Time	Level	electricity
Unit	•	Year	•	GWh/PJ
resBoilerElectric	region1	2010	fixed	300
resBoilerElectric	region1	2030	fixed	290

ProcessName represents the technology ID and needs to be consistent across all the data inputs.

RegionName represents the region ID and needs to be consistent across all the data inputs.

Time represents the period of the simulation to which the value applies; it needs to contain at least the base year of the simulation.

Level characterises either a fixed or a flexible input type the following columns should contain the list of commodities the row.

Unit reports the unit in which the technology consumption is defined; it is for the user internal reference only.

The same structure for the csv file would also apply for the output commodity file. The input data has to be provided for the base year. Additional years within the time framework of the overall simulation can be defined. In this case, MUSE would interpolate the values between the provided periods and assume a constant value afterwards.

6.3.9 Existing Sectoral Capacity

For each technology, the decommissioning profile should be given to MUSE.

The csv file which provides the installed capacity in base year and the decommissioning profile in the future periods for each technology in a sector, in each region, should follow the structure reported in the table.

Table 8: Existing capacity of technologies: the residential boiler example

ProcessName	RegionName	Unit	2010	2020	2030	2040	2050
resBoilerElectric	region1	PJ/y	5	0.5	0	0	0
resBoilerElectric	region2	PJ/y	39	3.5	1	0.3	0

ProcessName represents the technology ID and needs to be consistent across all the data inputs.

RegionName represents the region ID and needs to be consistent across all the data inputs.

Unit reports the unit of the technology capacity; it is for the user internal reference only.

2010,. . . , 2050 represent the simulated periods.

6.3.10 Agents

In MUSE, an agent-based formulation was originally introduced for the residential and commercial building sectors [2019:sachs]. Agents are defined using a CSV file, with one agent per row, using a somewhat historical format meant specifically for retrofit and new-capacity agent pairs. This CSV file can be read using `read_csv_agent_parameters()`. The data is also interpreted to some degree in the factory functions `create_retrofit_agent()` and `create_newcapa_agent()`.

For instance, we have the following CSV table:

Name	Type	AgentShare	RegionName	Objective1	SearchRule	DecisionMethod	. . .
A1	New	Agent5	ASEAN	EAC	all	epsilonCon	. . .
A4	New	Agent6	ASEAN	CapitalCosts	existing	weightedSum	. . .
A1	Retrofit	Agent1	ASEAN	efficiency	all	epsilonCon	. . .
A2	Retrofit	Agent2	ASEAN	Emissions	similar	weightedSum	. . .

For simplicity, not all columns are included in the example above. Though all column listed below are currently required.

The columns have the following meaning:

Name Name shared by a retrofit and new-capacity agent pair.

Type One of "New" or "Retrofit". "New" and "Retrofit" agents make up a pair with a given *name*. The demand is split into two, with one part coming from decommissioned assets, and the other coming from everything else. "Retrofit" agents invest only to make up for decommissioned assets. They are often limited in the technologies they can consider (by *SearchRule*). "New" agents invest on the rest of the demand, and can often consider more general sets of technologies.

AgentShare Name of the share of the existing capacity assigned to this agent. Only meaningful for retrofit agents. The actual share itself can be found in *Techno-data*.

RegionName Region where an agent operates.

Objective1 First objective that an agent will try and maximize or minimize during investment. This objective should be one registered with `@register_objective`. The following objectives are available with MUSE:

- `comfort`: Comfort provided by a given technology. Comfort does not change during the simulation. It is obtained straightforwardly from *Techno-data*.
- `efficiency`: Efficiency of the technologies. Efficiency does not change during the simulation. It is obtained straightforwardly from *Techno-data*.
- `fixed_costs`: The fixed maintenance costs incurred by a technology. The costs are a function of the capacity required to fulfil the current demand.
- `capital_costs`: The capital cost incurred by a technology. The capital cost does not change during the simulation. It is obtained as a function of parameters found in *Techno-data*.
- `emission_cost`: The costs associated for emissions for a technology. The costs is a function both of the amount produced (equated to the total demand in this case) and of the prices associated with each pollutant. Aliased to "emission" for simplicity.
- `fuel_consumption_cost`: Costs of the fuels for each technology, where each technology is used to fulfil the whole demand.
- `lifetime_levelized_cost_of_energy`: LCOE over the lifetime of a technology. Aliased to "LCOE" for simplicity.
- `net_present_value`: Present value of all the costs of installing and operating a technology, minus its revenues, of the course of its lifetime. Aliased to "NPV" for simplicity.
- `equivalent_annual_cost`: Annualized form of the net present value. Aliased to "EAC" for simplicity.

The weight associated with this objective can be changed using *ObjData1*. Whether the objective should be minimized or maximized depends on *Objsort1*. Multiple objectives are combined using the *DecisionMethod*

Objective2 Second objective. See *Objective1*.

Objective3: Third objective. See *Objective1*.

ObjData1 A weight associated with the *first objective*. Whether it is used will depend in large part on the *decision method*.

ObjData2 A weight associated with the *second objective*. See *ObjData1*.

ObjData3 A weight associated with the *third objective*. See *ObjData1*.

Objsort1 Whether to maximize (*True*) or minimize (*False*) the *first objective*.

Objsort2 Whether to maximize (*True*) or minimize (*False*) the *second objective*.

Objsort3 Whether to maximize (*True*) or minimize (*False*) the *third objective*.

SearchRule The search rule allows users to par down the search space of technologies to those an agent is likely to consider. The search rule is any function with a given signature, and registered with MUSE via `@register_filter`. The following search rules, defined in `filters`, are available with MUSE:

- `same_enduse`: Only allow technologies that provide the same enduse as the current set of technologies owned by the agent.
- `identity`: Allows all current technologies. E.g. disables filtering. Aliased to "all".
- `similar_technology`: Only allows technologies that have the same type as current crop of technologies in the agent, as determined by "tech_type" in *Techno-data*. Aliased to "similar".
- `same_fuels`: Only allows technologies that consume the same fuels as the current crop of technologies in the agent. Aliased to "fueltype".
- `currently_existing_tech`: Only allows technologies that the agent already owns. Aliased to "existing".

- `currently_referenced_tech`: Only allows technologies that are currently present in the market with non-zero capacity.
- `maturity`: Only allows technologies that have achieved a given market share.

The implementation allows for combining these filters. However, the CSV data format described here does not.

DecisionMethod Decision methods reduce multiple objectives into a single scalar objective per replacement technology. They allow combining several objectives into a single metric through which replacement technologies can be ranked.

Decision methods are any function which follow a given signature and are registered via the decorator `@register_decision`. The following decision methods are available with MUSE, as implemented in `decisions`:

- `mean`: Computes the average across several objectives.
- `weighted_sum`: Computes a weighted average across several objectives.
- `lexical_comparion`: Compares objectives using a binned lexical comparison operator. Aliased to "lexo".
- `retro_lexical_comparion`: A binned lexical comparison function where the bin size is adjusted to ensure the current crop of technologies are competitive. Aliased to "retro_lexo".
- `epsilon_constraints`: A comparison method which ensures that first selects technologies following constraints on objectives 2 and higher, before actually ranking them using objective 1. Aliased to "epsilon" ad "epsilon_con".
- `retro_epsilon_constraints`: A variation on epsilon constraints which ensures that the current crop of technologies are not deselected by the constraints. Aliased to "retro_epsilon".
- `single_objective`: A decision method to allow ranking via a single objective.

The functions allow for any number of objectives. However, the format described here allows only for three.

Quantity A factor used to determine the demand share of "New" agents.

MaturityThreshold Parameter for the search rule `maturity`.

6.3.11 Indices and tables

- genindex
- modindex
- search

6.4 Indices and tables

- genindex
- modindex
- search

CHAPTER

SEVEN

ADVANCED GUIDE

7.1 Extending MUSE

One key feature of the generalized sector's implementation is that it should be easy to extend. As such, MUSE can be made to run custom python functions, as long as these inputs and output of the function follow a standard specific to each step. We will look at a few here.

Below is a list of possible hooks, referenced by their implementation in the MUSE model:

- `register_interaction_net` in `muse.interactions`: a list of lists of agents that interact together.
- `register_agent_interaction` in `muse.interactions`: Given a list of interacting agents, perform the interaction.
- `register_production` in `muse.production`: A method to compute the production from a sector, given the demand and the capacity.
- `register_initial_asset_transform` in `muse.hooks`: Allows any kind of transformation to be applied to the assets of an agent, prior to investing.
- `register_final_asset_transform` in `muse.hooks`: After computing the investment, this sets the assets that will be owned by the agents.
- `register_demand_share` in `muse.demand_share`: During agent investment, this is the share of the demand that an agent will try and satisfy.
- `register_filter` in `muse.filters`: A filter to remove technologies from consideration, during agent investment.
- `register_objective` in `muse.objectives`: A quantity which allows an agent to compare technologies during investment.
- `register_decision` in `muse.decisions`: A transformation applied to aggregate multiple objectives into a single objective during agent investment, e.g. via a weighted sum.
- `register_investment` in `muse.investment`: During agent investment, matches the demand for future investment using the decision metric above.
- `register_output_quantity` in `muse.output.sector`: A sectorial quantity to output for post-mortem analysis.
- `register_output_sink` in `muse.outputs`: A *place* to store an output quantity, e.g. a file with a given format, a database on premise or on the cloud, etc...
- `register_carbon_budget_fitter` in `muse.carbon_budget`
- `register_carbon_budget_method` in `muse.carbon_budget`
- `register_sector`: Registers a function that can create a sector from a muse configuration object.

7.1.1 Extending outputs

MUSE can be used to save custom quantities as well as data for analysis. There are two steps to this process:

- Computing the quantity of interest
- Store the quantity of interest in a sink

In practice, this means that we can compute any quantity, such as capacity or consumption of an energy source and save it to a csv file, or a netcdf file.

Output extension

To demonstrate this, we will compute a new edited quantity of consumption, then save it as a text file.

The current implementation of the quantity of consumption found in `muse.outputs.sector` filters out values of 0. In this example, we would like to maintain the values of 0, but do not want to edit the source code of MUSE.

This is rather simple to do using MUSE's hooks.

First we create a new function called `consumption_zero` as follows:

[1]:
```
from muse.outputs import register_output_quantity
from muse.outputs.sector import market_quantity
from xarray import Dataset, DataArray
from typing import Optional, List, Text

@register_output_quantity
def consumption_zero(
    market: Dataset,
    capacity: DataArray,
    technologies: Dataset,
):
    """Current consumption."""
    result = (
        market_quantity(market.consumption, sum_over="timeslice", drop=None)
        .rename("consumption")
        .to_dataframe()
        .round(4)
    )
    return result
```

The function we created takes three arguments. These arguments (`market`, `capacity` and `technology`) are mandatory for the `@register_output_quantity` hook. Other hooks require different arguments.

Whilst this function is very similar to the `consumption` function in `muse.outputs.sector`, we have modified it slightly by allowing for values of `0`.

The important part of this function is the `@register_output_quantity` decorator. This decorator ensures that this new quantity is addressable in the TOML file. Notice that we did not need to edit the source code to create our new function.

Next, we can create a sink to save the output quantity previously registered. For this example, this sink will simply dump the quantity it is given to a file, with the "Hello world!" message:

[2]:
```
from typing import Any, Text
from muse.outputs.sinks import register_output_sink, sink_to_file

@register_output_sink(name="txt")
```

(continues on next page)

(continued from previous page)

```
@sink_to_file(".txt")
def text_dump(data: Any, filename: Text) -> None:
    from pathlib import Path
    Path(filename).write_text(f"Hello world!\n\n{data}")
```

The code above makes use of two dectorators: `@register_output_sink` and `@sink_to_file`.

`@register_output_sink` registers the function with MUSE, so that the sink is addressable from a TOML file. The second one, `@sink_to_file`, is optional. This adds some nice-to-have features to sinks that are files. For example, a way to specify filenames and check that files cannot be overwritten, unless explicitly allowed to.

Next, we want to modify the TOML file to actually use this output type. To do this, we add a section to the output table:

```
[[sectors.residential.outputs]]
quantity = "consumption_zero"
sink = "txt"
filename = "{cwd}/{default_output_dir}/{Sector}{Quantity}{year}{suffix}"
```

The last line above allows us to specify the name of the file. We could also use `sector` above or `quantity`.

There can be as many sections of this kind as we like in the TOML file, which allow for multiple outputs.

Next, we first copy the default model provided with muse to a local subfolder called "model". Then we read the `settings.toml` file and modify it using python. You may prefer to modify the `settings.toml` file using your favorite text editor. However, modifying the file programmatically allows us to routinely run this notebook as part of MUSE's test suite and check that the tutorial it is still up to date.

[3]:
```
from pathlib import Path
from toml import load, dump
from muse import examples

model_path = examples.copy_model(overwrite=True)
settings = load(model_path / "settings.toml")
new_output = {
    "quantity": "consumption_zero",
    "sink":  "txt",
    "overwrite": True,
    "filename": "{cwd}/{default_output_dir}/{Sector}{Quantity}{year}{suffix}",
}
settings["sectors"]["residential"]["outputs"].append(new_output)
dump(settings, (model_path / "modified_settings.toml").open("w"))
settings
```

```
-- 2020-11-09 11:19:48 - muse.sectors.register - INFO
Sector legacy registered.

-- 2020-11-09 11:19:48 - muse.sectors.register - INFO
Sector preset registered, with alias presets.

-- 2020-11-09 11:19:48 - muse.sectors.register - INFO
Sector default registered.
```

[3]:
```
{'time_framework': [2020, 2025, 2030, 2035, 2040, 2045, 2050],
 'foresight': 5,
 'regions': ['R1'],
 'interest_rate': 0.1,
```

(continues on next page)

(continued from previous page)

```
'interpolation_mode': 'Active',
'log_level': 'info',
'equilibrium_variable': 'demand',
'maximum_iterations': 100,
'tolerance': 0.1,
'tolerance_unmet_demand': -0.1,
'outputs': [{'quantity': 'prices',
  'sink': 'aggregate',
  'filename': '{cwd}/{default_output_dir}/MCA{Quantity}.csv'},
 {'quantity': 'capacity',
  'sink': 'aggregate',
  'filename': '{cwd}/{default_output_dir}/MCA{Quantity}.csv'}],
'carbon_budget_control': {'budget': []},
'global_input_files': {'projections': '{path}/input/Projections.csv',
 'global_commodities': '{path}/input/GlobalCommodities.csv'},
'sectors': {'residential': {'type': 'default',
  'priority': 1,
  'dispatch_production': 'share',
  'technodata': '{path}/technodata/residential/Technodata.csv',
  'commodities_in': '{path}/technodata/residential/CommIn.csv',
  'commodities_out': '{path}/technodata/residential/CommOut.csv',
  'subsectors': {'retro_and_new': {'agents': '{path}/technodata/Agents.csv',
    'existing_capacity': '{path}/technodata/residential/ExistingCapacity.csv',
    'lpsolver': 'scipy',
    'constraints': ['max_production',
     'max_capacity_expansion',
     'demand',
     'search_space'],
    'demand_share': 'new_and_retro',
    'forecast': 5}},
  'outputs': [{'filename': '{cwd}/{default_output_dir}/{Sector}/{Quantity}/{year}
↪{suffix}',
    'quantity': 'capacity',
    'sink': 'csv',
    'overwrite': True},
   {'filename': '{cwd}/{default_output_dir}/{Sector}/{Quantity}/{year}{suffix}',
    'quantity': {'name': 'supply',
     'sum_over': 'timeslice',
     'drop': ['comm_usage', 'units_prices']},
    'sink': 'csv',
    'overwrite': True},
   {'quantity': 'consumption_zero',
    'sink': 'txt',
    'overwrite': True,
    'filename': '{cwd}/{default_output_dir}/{Sector}{Quantity}{year}{suffix}'}],
  'interactions': [{'net': 'new_to_retro', 'interaction': 'transfer'}]},
 'power': {'type': 'default',
  'priority': 2,
  'dispatch_production': 'share',
  'technodata': '{path}/technodata/power/Technodata.csv',
  'commodities_in': '{path}/technodata/power/CommIn.csv',
  'commodities_out': '{path}/technodata/power/CommOut.csv',
  'subsectors': {'retro_and_new': {'agents': '{path}/technodata/Agents.csv',
    'existing_capacity': '{path}/technodata/power/ExistingCapacity.csv',
    'lpsolver': 'scipy'}},
  'outputs': [{'filename': '{cwd}/{default_output_dir}/{Sector}/{Quantity}/{year}
↪{suffix}',
```

(continues on next page)

(continued from previous page)

```
     'quantity': 'capacity',
     'sink': 'csv',
     'overwrite': True}],
   'interactions': [{'net': 'new_to_retro', 'interaction': 'transfer'}]},
  'gas': {'type': 'default',
   'priority': 3,
   'dispatch_production': 'share',
   'technodata': '{path}/technodata/gas/Technodata.csv',
   'commodities_in': '{path}/technodata/gas/CommIn.csv',
   'commodities_out': '{path}/technodata/gas/CommOut.csv',
   'subsectors': {'retro_and_new': {'agents': '{path}/technodata/Agents.csv',
     'existing_capacity': '{path}/technodata/gas/ExistingCapacity.csv',
     'lpsolver': 'scipy'}},
   'outputs': [{'filename': '{cwd}/{default_output_dir}/{Sector}/{Quantity}/{year}
↪{suffix}',
     'quantity': 'capacity',
     'sink': 'csv',
     'overwrite': True}],
   'interactions': [{'net': 'new_to_retro', 'interaction': 'transfer'}]},
  'residential_presets': {'type': 'presets',
   'priority': 0,
   'consumption_path': '{path}/technodata/preset/*Consumption.csv'}},
 'timeslices': {'all-year': {'all-week': {'night': 1460,
    'morning': 1460,
    'afternoon': 1460,
    'early-peak': 1460,
    'late-peak': 1460,
    'evening': 1460}},
  'level_names': ['month', 'day', 'hour']}}
```

We can now run the simulation. There are two ways to do this. From the command-line, where we can do:

```
python3 -m muse data/commercial/modified_settings.toml
```

(note that slashes may be the other way on Windows). Or directly from the notebook:

[4]:
```
import logging
from muse.mca import MCA
logging.getLogger("muse").setLevel(0)
mca = MCA.factory(model_path / "modified_settings.toml")
mca.run();
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             148.9679256735
0.2598249156018     0.2598249156018     0.2598249156018     0.7495200004432  0.
↪2598249156018      120.5733849622
0.02399956829695    0.02399956829695    0.02399956829695    0.9210391224498  0.
↪02399956829695     4.780663765494
0.0181364461758     0.0181364461758     0.0181364461758     0.2509588065043  0.
↪0181364461758      7.107141691547
0.01499350833129    0.01499350833129    0.01499350833129    0.1921973185437  0.
↪01499350833129     70.77614035582
0.004968295711366   0.004968295711367   0.004968295711366   0.6857131120066  0.
↪004968295711366    164.7472224003
0.0006443120819652  0.0006443120819642  0.000644312081964   0.8804718592549  0.
↪0006443120819672   289.7109372802
```

(continues on next page)

(continued from previous page)

```
2.427431365313e-06  2.427431365276e-06  2.42743136527e-06   0.9976309182175  2.
↪427431365399e-06  310.7437190082
1.214286379284e-10  1.214286245985e-10  1.214286217581e-10  0.9999499778566  1.
↪214286284187e-10  310.7859704917
Optimization terminated successfully.
         Current function value: 310.785970
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪            148.9679256735
0.1806451257467     0.1806451257467     0.1806451257467     0.8281847212959  0.
↪1806451257467     169.6037982942
0.02684129624378    0.02684129624378    0.02684129624378    0.8635116331789  0.
↪02684129624378    123.2904723181
0.01081107082374    0.01081107082373    0.01081107082373    0.6279145428497  0.
↪01081107082373    293.3552576002
0.00150335333755    0.001503353337531   0.001503353337531   0.8785435425234  0.
↪001503353337582   618.5754737903
4.126548293386e-06  4.126548293452e-06  4.126548293469e-06  0.99729939758    4.
↪126548293473e-06  673.2429034259
2.063813940498e-10  2.063814559148e-10  2.063814538311e-10  0.9999499869317  2.
↪06381853248e-10   673.369600975
Optimization terminated successfully.
         Current function value: 673.369601
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪            225.7506433631
0.07675788061753    0.07675788061753    0.07675788061751    0.9271368109475  0.
↪07675788061753    1.307751786207
0.01889464099889    0.01889464099889    0.01889464099888    0.7970949255449  0.
↪01889464099889    19.4989221165
0.007543783963398   0.007543783963394   0.007543783963392   0.6153162486386  0.
↪007543783963394   16.52048983639
0.002504946781023   0.002504946781021   0.00250494678102    0.7004074675406  0.
↪002504946781021   72.319459457
0.0004445444355394  0.000444544435539   0.0004445444355388  0.8689995047748  0.
↪000444544435539   423.758860583
1.214501095331e-05  1.214501095325e-05  1.214501095324e-05  0.9820606335229  1.
↪214501095322e-05  675.1646942955
1.070439776445e-09  1.07043978707e-09   1.070439766673e-09  0.9999120772687  1.
↪070439801419e-09  681.9179209593
5.353929135063e-14  5.353592310399e-14  5.35373239097e-14   0.9999499866071  5.
↪352230256347e-14  681.9185224492
Optimization terminated successfully.
         Current function value: 681.918522
         Iterations: 8
```

```
-- 2020-11-09 11:19:59 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪            359.2443189825
```

(continues on next page)

(continued from previous page)

```
0.2131118149695     0.2131118149695     0.2131118149695     0.7960343319409  0.
↪2131118149695     262.2885392799
0.05758094728718    0.05758094728718    0.05758094728718    0.7523513573813  0.
↪05758094728718    13.16175379893
0.01048349585899    0.01048349585899    0.01048349585899    0.8203940076989  0.
↪01048349585899    9.036399448741
0.009005598049604   0.009005598049604   0.009005598049604   0.1488155054953  0.
↪009005598049604   19.40296370843
0.002317604003518   0.002317604003518   0.002317604003518   0.8098673571979  0.
↪002317604003518   226.5492459765
0.0009784310820962  0.0009784310820963  0.0009784310820963  0.5991550275447  0.
↪0009784310820971  289.4684048776
0.0001326875071986  0.0001326875071986  0.0001326875071986  0.8836343167337  0.
↪0001326875071987  358.6919881748
6.688262823007e-08  6.688262823276e-08  6.688262823391e-08  0.9995454959391  6.
↪688262822145e-08  365.8223849509
3.344208692489e-12  3.34420597683e-12   3.344204125008e-12  0.9999499989235  3.
↪344177862259e-12  365.8250744356
Optimization terminated successfully.
         Current function value: 365.825074
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            179.6221594912
0.06792119055695    0.06792119055695    0.06792119055695    0.9362017234058  0.
↪06792119055695    76.26287556551
0.009746786382626   0.009746786382626   0.009746786382626   0.8719786940257  0.
↪009746786382625   34.28929487798
0.00504956460969    0.005049564609689   0.005049564609689   0.5134005827998  0.
↪005049564609689   115.6145513358
0.003132107070668   0.003132107070663   0.003132107070663   0.3922116719677  0.
↪003132107070663   175.0444435627
0.0005331490663204  0.000533149066321   0.000533149066321   0.8631478238108  0.
↪0005331490663195  430.7265923665
6.809758501168e-06  6.809758501084e-06  6.809758501069e-06  0.9896962642248  6.
↪809758501216e-06  482.9615875673
3.566453823927e-10  3.56645399101e-10   3.566453982014e-10  0.9999476448412  3.
↪566453742836e-10  483.66367672
Optimization terminated successfully.
         Current function value: 483.663677
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            547.5170704708
0.06099397574481    0.06099397574481    0.06099397574482    0.944387525785   0.
↪06099397574482    192.4316112656
0.01386491315737    0.01386491315737    0.01386491315737    0.7896145976147  0.
↪01386491315737    3.157754203839
0.00713143943229    0.00713143943229    0.007131439432292   0.5116778496497  0.
↪007131439432291   6.913227303095
0.0007854492963609  0.0007854492963609  0.0007854492963611  0.9057153968575  0.
↪000785449296361   6.591304425235
0.0003968642772996  0.0003968642772997  0.0003968642772998  0.5195625901748  0.
↪0003968642772996  5.218772647071
5.276614006577e-06  5.276614006577e-06  5.276614006576e-06  0.9941140946865  5.
↪276614006565e-06  5.272614338781
```

(continues on next page)

(continued from previous page)

```
3.679893198137e-10  3.679893242575e-10  3.679893250277e-10  0.9999303117912  3.
↪679893357516e-10  5.266601980833
1.790517523351e-14  1.842347640038e-14  1.841384711876e-14  0.9999499611495  1.
↪844295319492e-14  5.266601636509
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             273.7585352354
0.06117571447458    0.06117571447458    0.06117571447455    0.9425787965419  0.
↪06117571447457    204.7212184456
0.009942269802024   0.009942269802025   0.00994226980202    0.8403658088806  0.
↪009942269802024   0.7760264390675
0.002147318631478   0.002147318631478   0.002147318631477   0.8277733790617  0.
↪002147318631478   3.894705138429
0.0009649606635229  0.0009649606635227  0.0009649606635223  0.5635818341147  0.
↪0009649606635226  2.700821086695
0.0002350111516202  0.0002350111516201  0.00023501115162    0.7866567764877  0.
↪0002350111516201  14.62347819316
5.857294640779e-05  5.857294640777e-05  5.857294640774e-05  0.8105927228255  5.
↪857294640777e-05  123.50656776
1.165473006364e-06  1.165473006407e-06  1.165473006407e-06  0.9835504451632  1.
↪165473006409e-06  151.6654340598
8.362816702708e-11  8.362816612134e-11  8.362816648317e-11  0.9999285085203  8.
↪36281608009e-11   152.3843167925
4.185697343433e-15  4.184037681176e-15  4.18416825892e-15   0.9999499678098  4.
↪181415889613e-15  152.3843678125
Optimization terminated successfully.
         Current function value: 152.384368
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             24.59260538017
0.01918847234907    0.01918847234907    0.01918847234907    0.9908439847925  0.
↪01918847234907    0.01449776485458
0.01221126742797    0.01221126742797    0.01221126742797    0.3735738645022  0.
↪01221126742797    0.1603023022985
0.009831906616685   0.009831906616685   0.009831906616685   0.2161492677935  0.
↪009831906616685   20.38072882765
0.001212407727123   0.001212407727023   0.001212407727023   0.8886055956883  0.
↪001212407727023   51.40232805869
4.730913157321e-06  4.73091315438e-06   4.730913154368e-06  0.9976178802846  4.
↪730913155107e-06  59.83474944668
2.366017130877e-10  2.366016942176e-10  2.366016879444e-10  0.9999499881554  2.
↪366016905478e-10  59.84200873085
Optimization terminated successfully.
         Current function value: 59.842009
         Iterations: 6
-- 2020-11-09 11:20:09 - muse.mca - WARNING
Check growth constraints for wind.

Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
```

(continues on next page)

(continued from previous page)

```
1.0                    1.0                    1.0                    -                1.0                ␣
↪            359.2443189825
0.2131118149695      0.2131118149695      0.2131118149695      0.7960343319409  0.
↪2131118149695       262.2885392799
0.05758094728718     0.05758094728718     0.05758094728718     0.7523513573813  0.
↪05758094728718      13.16175379893
0.01048349585899     0.01048349585899     0.01048349585899     0.8203940076989  0.
↪01048349585899      9.036399448741
0.009005598049604    0.009005598049604    0.009005598049604    0.1488155054953  0.
↪009005598049604     19.40296370843
0.002317604003518    0.002317604003518    0.002317604003518    0.8098673571979  0.
↪002317604003518     226.5492459765
0.0009784310820962   0.0009784310820963   0.0009784310820963   0.5991550275447  0.
↪0009784310820971    289.4684048776
0.0001326875071986   0.0001326875071986   0.0001326875071986   0.8836343167337  0.
↪0001326875071987    358.6919881748
6.688262823007e-08   6.688262823276e-08   6.688262823391e-08   0.9995454959391  6.
↪688262822145e-08    365.8223849509
3.344208692489e-12   3.34420597683e-12    3.344204125008e-12   0.9999499989235  3.
↪344177862259e-12    365.8250744356
Optimization terminated successfully.
         Current function value: 365.825074
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                    1.0                    1.0                    -                1.0                ␣
↪            179.6221594912
0.06792119055695     0.06792119055695     0.06792119055695     0.9362017234058  0.
↪06792119055695      76.26287556551
0.009746786382626    0.009746786382626    0.009746786382626    0.8719786940257  0.
↪009746786382625     34.28929487798
0.00504956460969     0.005049564609689    0.005049564609689    0.5134005827998  0.
↪005049564609689     115.6145513358
0.003132107070668    0.003132107070663    0.003132107070663    0.3922116719677  0.
↪003132107070663     175.0444435627
0.0005331490663204   0.000533149066321    0.000533149066321    0.8631478238108  0.
↪0005331490663195    430.7265923665
6.809758501168e-06   6.809758501084e-06   6.809758501069e-06   0.9896962642248  6.
↪809758501216e-06    482.9615875673
3.566453823927e-10   3.56645399101e-10    3.566453982014e-10   0.9999476448412  3.
↪566453742836e-10    483.66367672
Optimization terminated successfully.
         Current function value: 483.663677
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                    1.0                    1.0                    -                1.0                ␣
↪            547.5170704708
0.06099397574481     0.06099397574481     0.06099397574482     0.944387525785   0.
↪06099397574482      192.4316112656
0.01386491315737     0.01386491315737     0.01386491315737     0.7896145976147  0.
↪01386491315737      3.157754203839
0.00713143943229     0.00713143943229     0.007131439432292    0.5116778496497  0.
↪007131439432291     6.913227303095
0.0007854492963609   0.0007854492963609   0.0007854492963611   0.9057153968575  0.
↪000785449296361     6.591304425235
0.0003968642772996   0.0003968642772997   0.0003968642772998   0.5195625901748  0.
↪0003968642772996    5.218772647071
```

(continues on next page)

(continued from previous page)

```
5.276614006577e-06  5.276614006577e-06  5.276614006576e-06  0.9941140946865  5.
↪276614006565e-06  5.272614338781
3.679893198137e-10  3.679893242575e-10  3.679893250277e-10  0.9999303117912  3.
↪679893357516e-10  5.266601980833
1.790517523351e-14  1.842347640038e-14  1.841384711876e-14  0.9999499611495  1.
↪844295319492e-14  5.266601636509
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             273.7585352354
0.06117571447458    0.06117571447458    0.06117571447455    0.9425787965419  0.
↪06117571447457     204.7212184456
0.009942269802024   0.009942269802025   0.00994226980202    0.8403658088806  0.
↪009942269802024    0.7760264390675
0.002147318631478   0.002147318631478   0.002147318631477   0.8277733790617  0.
↪002147318631478    3.894705138429
0.0009649606635229  0.0009649606635227  0.0009649606635223  0.5635818341147  0.
↪0009649606635226   2.700821086695
0.0002350111516202  0.0002350111516201  0.00023501115162    0.7866567764877  0.
↪0002350111516201   14.62347819316
5.857294640779e-05  5.857294640777e-05  5.857294640774e-05  0.8105927228255  5.
↪857294640777e-05   123.50656776
1.165473006364e-06  1.165473006407e-06  1.165473006407e-06  0.9835504451632  1.
↪165473006409e-06   151.6654340598
8.362816702708e-11  8.362816612134e-11  8.362816648317e-11  0.9999285085203  8.
↪36281608009e-11    152.3843167925
4.185697343433e-15  4.184037681176e-15  4.18416825892e-15   0.9999499678098  4.
↪181415889613e-15   152.3843678125
Optimization terminated successfully.
         Current function value: 152.384368
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             24.59260538017
0.01918847234907    0.01918847234907    0.01918847234907    0.9908439847925  0.
↪01918847234907     0.01449776485458
0.01221126742797    0.01221126742797    0.01221126742797    0.3735738645022  0.
↪01221126742797     0.1603023022985
0.009831906616685   0.009831906616685   0.009831906616685   0.2161492677935  0.
↪009831906616685    20.38072882765
0.001212407727123   0.001212407727023   0.001212407727023   0.8886055956883  0.
↪001212407727023    51.40232805869
4.730913157321e-06  4.73091315438e-06   4.730913154368e-06  0.9976178802846  4.
↪730913155107e-06   59.83474944668
2.366017130877e-10  2.366016942176e-10  2.366016879444e-10  0.9999499881554  2.
↪366016905478e-10   59.84200873085
Optimization terminated successfully.
         Current function value: 59.842009
         Iterations: 6
```

```
-- 2020-11-09 11:20:18 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1943112264485     0.1943112264485     0.1943112264485     0.8115414580695  0.
↪1943112264485      293.1180738751
0.05737071259203    0.05737071259203    0.05737071259203    0.7291321197238  0.
↪05737071259203     17.07106219013
0.01062277327644    0.01062277327644    0.01062277327644    0.8173814867435  0.
↪01062277327644     12.44885046687
0.009064514495371   0.009064514495372   0.009064514495372   0.1550649500025  0.
↪009064514495372    28.00868491133
0.002334119807294   0.002334119807293   0.002334119807293   0.8051326751218  0.
↪002334119807295    275.8552661516
0.0006946146456239  0.0006946146456238  0.0006946146456238  0.7097377670861  0.
↪0006946146456242   332.7560201847
0.000253341221249   0.0002533412212489  0.0002533412212489  0.6718481634042  0.
↪0002533412212491   377.2933943083
1.33722226265e-06   1.337222262655e-06  1.337222262657e-06  0.9952598213341  1.
↪337222262651e-06   392.3599718847
6.775876566156e-11  6.775875940049e-11  6.775875524037e-11  0.9999493313269  6.
↪775868905564e-11   392.4568216146
Optimization terminated successfully.
         Current function value: 392.456822
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1030590385675     0.1030590385675     0.1030590385675     0.9011154818883  0.
↪1030590385675      219.2748221074
0.02654054666621    0.02654054666621    0.02654054666621    0.7752562655553  0.
↪02654054666621     276.7551091027
0.01427394633699    0.01427394633699    0.01427394633699    0.4912526709581  0.
↪01427394633699     607.4391713228
0.001218580492651   0.001218580492601   0.001218580492601   0.9248974186534  0.
↪001218580492676    1166.289755796
1.888403761715e-05  1.888403761622e-05  1.888403761624e-05  0.985551099478   1.
↪888403761735e-05   1241.740132447
1.545745837373e-09  1.545745372552e-09  1.545745362755e-09  0.999922927438   1.
↪545745648845e-09   1242.779847889
1.692223225044e-10  1.873153075245e-10  1.873152955153e-10  0.8801597673128  1.
↪817506825368e-10   1242.779937345
Optimization terminated successfully.
         Current function value: 1242.779937
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             652.1069903706
0.06097571429344    0.06097571429346    0.06097571429347    0.9444902662603  0.
↪06097571429346     285.2043200632
0.01533833555931    0.01533833555931    0.01533833555932    0.7609599230413  0.
↪01533833555932     2.7262007615
0.006316196361705   0.006316196361706   0.006316196361708   0.6163829523723  0.
↪006316196361707    8.634999065661
0.001219580332373   0.001219580332374   0.001219580332374   0.8323480537452  0.
↪001219580332374    8.307794697355
```

(continues on next page)

(continued from previous page)

```
0.0005645791470893  0.0005645791470901  0.0005645791470902  0.5500511910892  0.
↪0005645791470901  6.632034269378
0.0001849432525583  0.0001849432525588  0.0001849432525588  0.7137620753374  0.
↪0001849432525588  4.483601470858
2.517759634035e-05  2.517759634041e-05  2.517759634042e-05  0.9034269505801  2.
↪517759634041e-05  4.138476382855
3.098182398421e-08  3.098182439526e-08  3.09818244027e-08   0.9988322072353  3.
↪098182439405e-08  4.000023749419
1.548986338477e-12  1.549101875116e-12  1.549109003245e-12  0.9999499994251  1.
↪549106268784e-12  3.999950652469
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0               ␣
↪             24.59260538017
0.02944420803734    0.02944420803734    0.02944420803733    0.9778743187132  0.
↪02944420803734    27.12355968787
0.008204037532537   0.008204037532537   0.008204037532536   0.7626654784836  0.
↪008204037532537   110.4823556367
0.0004115277845134  0.0004115277844911  0.000411527784491   0.9587152757388  0.
↪0004115277845217  181.7019444318
2.842168973263e-08  2.842168968894e-08  2.84216896431e-08   0.9999314505927  2.
↪84216896475e-08   184.4443098122
1.421064808692e-12  1.421032025856e-12  1.420975506573e-12  0.999950002033   1.
↪421084481186e-12  184.444539173
Optimization terminated successfully.
         Current function value: 184.444539
         Iterations: 5
```

```
-- 2020-11-09 11:20:25 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0               ␣
↪             414.316476678
0.1943112264485     0.1943112264485     0.1943112264485     0.8115414580695  0.
↪1943112264485     293.1180738751
0.05737071259203    0.05737071259203    0.05737071259203    0.7291321197238  0.
↪05737071259203    17.07106219013
0.01062277327644    0.01062277327644    0.01062277327644    0.8173814867435  0.
↪01062277327644    12.44885046687
0.009064514495371   0.009064514495372   0.009064514495372   0.1550649500025  0.
↪009064514495372   28.00868491133
0.002334119807294   0.002334119807293   0.002334119807293   0.8051326751218  0.
↪002334119807295   275.8552661516
0.0006946146456239  0.0006946146456238  0.0006946146456238  0.7097377670861  0.
↪0006946146456242  332.7560201847
0.000253341221249   0.0002533412212489  0.0002533412212489  0.6718481634042  0.
↪0002533412212491  377.2933943083
1.33722226265e-06   1.337222262655e-06  1.337222262657e-06  0.9952598213341  1.
↪337222262651e-06  392.3599718847
6.775876566156e-11  6.775875940049e-11  6.775875524037e-11  0.9999493313269  6.
↪775868905564e-11  392.4568216146
Optimization terminated successfully.
```

(continues on next page)

(continued from previous page)

```
         Current function value: 392.456822
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0             ␣
↪            414.316476678
0.1030590385675     0.1030590385675     0.1030590385675     0.9011154818883  0.
↪1030590385675     219.2748221074
0.02654054666621    0.02654054666621    0.02654054666621    0.7752562655553  0.
↪02654054666621    276.7551091027
0.01427394633699    0.01427394633699    0.01427394633699    0.4912526709581  0.
↪01427394633699    607.4391713228
0.001218580492651   0.001218580492601   0.001218580492601   0.9248974186534  0.
↪001218580492676   1166.289755796
1.888403761715e-05  1.888403761622e-05  1.888403761624e-05  0.985551099478   1.
↪888403761735e-05  1241.740132447
1.545745837373e-09  1.545745372552e-09  1.545745362755e-09  0.999922927438   1.
↪545745648845e-09  1242.779847889
1.692223225044e-10  1.873153075245e-10  1.873152955153e-10  0.8801597673128  1.
↪817506825368e-10  1242.779937345
Optimization terminated successfully.
         Current function value: 1242.779937
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0             ␣
↪            652.1069903706
0.06097571429344    0.06097571429346    0.06097571429347    0.9444902662603  0.
↪06097571429346    285.2043200632
0.01533833555931    0.01533833555931    0.01533833555932    0.7609599230413  0.
↪01533833555932    2.7262007615
0.006316196361705   0.006316196361706   0.006316196361708   0.6163829523723  0.
↪006316196361707   8.634999065661
0.001219580332373   0.001219580332374   0.001219580332374   0.8323480537452  0.
↪001219580332374   8.307794697355
0.0005645791470893  0.0005645791470901  0.0005645791470902  0.5500511910892  0.
↪0005645791470901  6.632034269378
0.0001849432525583  0.0001849432525588  0.0001849432525588  0.7137620753374  0.
↪0001849432525588  4.483601470858
2.517759634035e-05  2.517759634041e-05  2.517759634042e-05  0.9034269505801  2.
↪517759634041e-05  4.138476382855
3.098182398421e-08  3.098182439526e-08  3.09818244027e-08   0.9988322072353  3.
↪098182439405e-08  4.000023749419
1.548986338477e-12  1.549101875116e-12  1.549109003245e-12  0.9999499994251  1.
↪549106268784e-12  3.999950652469
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0             ␣
↪            24.59260538017
0.02944420803734    0.02944420803734    0.02944420803733    0.9778743187132  0.
↪02944420803734    27.12355968787
0.008204037532537   0.008204037532537   0.008204037532536   0.7626654784836  0.
↪008204037532537   110.4823556367
0.0004115277845134  0.0004115277844911  0.000411527784491   0.9587152757388  0.
↪0004115277845217  181.7019444318
```

(continues on next page)

(continued from previous page)

```
2.842168973263e-08  2.842168968894e-08  2.84216896431e-08   0.9999314505927  2.
↪84216896475e-08   184.4443098122
1.421064808692e-12  1.421032025856e-12  1.420975506573e-12  0.999950002033   1.
↪421084481186e-12  184.444539173
Optimization terminated successfully.
         Current function value: 184.444539
         Iterations: 5
```

```
-- 2020-11-09 11:20:33 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪            454.6784294315
0.1862232334687     0.1862232334687     0.1862232334687     0.8174567476576  0.
↪1862232334687     282.4632604722
0.02870747975048    0.02870747975048    0.02870747975048    0.8875534078522  0.
↪02870747975048    46.09791383177
0.009705128410004   0.009705128410004   0.009705128410004   0.6723641007507  0.
↪009705128410004   35.13402203751
0.007693915131578   0.007693915131578   0.007693915131578   0.2197837130033  0.
↪007693915131578   80.93881539206
0.001699044794839   0.001699044794837   0.001699044794837   0.8250841182677  0.
↪00169904479484    298.2653882331
0.0005650008980062  0.0005650008980057  0.0005650008980057  0.678565948553   0.
↪0005650008980065  327.975809321
0.0002196033953225  0.0002196033953223  0.0002196033953223  0.6469207178836  0.
↪0002196033953226  358.4708363196
2.579861478512e-06  2.579861478501e-06  2.579861478502e-06  0.9887803221269  2.
↪579861478497e-06  368.5514208652
1.310502062448e-10  1.310502049574e-10  1.31050204896e-10   0.9999492039366  1.
↪310502128569e-10  368.6632645094
Optimization terminated successfully.
         Current function value: 368.663265
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪            454.6784294315
0.06717002148527    0.06717002148527    0.06717002148527    0.939336263039   0.
↪06717002148527    183.2225626364
0.0335506482529     0.0335506482529     0.0335506482529     0.5230681149682  0.
↪0335506482529     256.2098343218
0.01040420231972    0.01040420231972    0.01040420231972    0.6975119886622  0.
↪01040420231972    178.3301859411
0.004349337378734   0.004349337378734   0.004349337378734   0.6235886580387  0.
↪004349337378734   396.8309868051
0.002928936786464   0.002928936786463   0.002928936786463   0.3467952576409  0.
↪002928936786463   488.976382689
0.0001458208281248  0.0001458208281499  0.0001458208281499  0.9601001354313  0.
↪0001458208281409  665.3317152907
1.593588680453e-07  1.593588684743e-07  1.593588684755e-07  0.9989479829423  1.
↪593588661973e-07  675.8691141106
7.991316370683e-12  7.990632294946e-12  7.990631661844e-12  0.999949860274   7.
↪990813072693e-12  675.8826828943
Optimization terminated successfully.
```

(continues on next page)

(continued from previous page)

```
         Current function value: 675.882683
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0             ␣
↪             759.5666413636
0.06104093664241    0.06104093664241    0.0610409366424     0.9444854714867  0.
↪06104093664241    386.9519025016
0.01481511454072    0.01481511454072    0.01481511454072    0.7670183371838  0.
↪01481511454072    2.643589195991
0.005238876768342   0.005238876768342   0.005238876768341   0.6766638370173  0.
↪005238876768342   11.29316821413
0.001567745187159   0.001567745187159   0.001567745187159   0.7265977484523  0.
↪001567745187159   12.32846209581
0.0001184080214619  0.0001184080214619  0.0001184080214619  0.9328477549533  0.
↪0001184080214619  6.401954889258
1.02649122158e-05   1.026491221577e-05  1.026491221577e-05  0.959846524846   1.
↪026491221577e-05  6.140180339771
2.564197796788e-08  2.564197949727e-08  2.564197949173e-08  0.9988486248561  2.
↪56419794805e-08   6.06660177389
1.282243371032e-12  1.282166817908e-12  1.282163560611e-12  0.9999499974762  1.
↪282163552715e-12  6.066591804033
Optimization terminated successfully.
         Current function value: 6.066592
         Iterations: 8
```

```
-- 2020-11-09 11:20:40 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0             ␣
↪             454.6784294315
0.1862232334687     0.1862232334687     0.1862232334687     0.8174567476576  0.
↪1862232334687     282.4632604722
0.02870747975048    0.02870747975048    0.02870747975048    0.8875534078522  0.
↪02870747975048    46.09791383177
0.009705128410004   0.009705128410004   0.009705128410004   0.6723641007507  0.
↪009705128410004   35.13402203751
0.007693915131578   0.007693915131578   0.007693915131578   0.2197837130033  0.
↪007693915131578   80.93881539206
0.001699044794839   0.001699044794837   0.001699044794837   0.8250841182677  0.
↪00169904479484    298.2653882331
0.0005650008980062  0.0005650008980057  0.0005650008980057  0.678565948553   0.
↪0005650008980065  327.975809321
0.0002196033953225  0.0002196033953223  0.0002196033953223  0.6469207178836  0.
↪0002196033953226  358.4708363196
2.579861478512e-06  2.579861478501e-06  2.579861478502e-06  0.9887803221269  2.
↪579861478497e-06  368.5514208652
1.310502062448e-10  1.310502049574e-10  1.31050204896e-10   0.9999492039366  1.
↪310502128569e-10  368.6632645094
Optimization terminated successfully.
         Current function value: 368.663265
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0             ␣
↪             454.6784294315
```

(continues on next page)

(continued from previous page)

```
0.06717002148527    0.06717002148527    0.06717002148527    0.939336263039    0.
↪06717002148527    183.2225626364
0.0335506482529     0.0335506482529     0.0335506482529     0.5230681149682   0.
↪0335506482529     256.2098343218
0.01040420231972    0.01040420231972    0.01040420231972    0.6975119886622   0.
↪01040420231972    178.3301859411
0.004349337378734   0.004349337378734   0.004349337378734   0.6235886580387   0.
↪004349337378734   396.8309868051
0.002928936786464   0.002928936786463   0.002928936786463   0.3467952576409   0.
↪002928936786463   488.976382689
0.0001458208281248  0.0001458208281499  0.0001458208281499  0.9601001354313   0.
↪0001458208281409  665.3317152907
1.593588680453e-07  1.593588684743e-07  1.593588684755e-07  0.9989479829423   1.
↪593588661973e-07  675.8691141106
7.991316370683e-12  7.990632294946e-12  7.990631661844e-12  0.999949860274    7.
↪990813072693e-12  675.8826828943
Optimization terminated successfully.
         Current function value: 675.882683
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪             759.5666413636
0.06104093664241    0.06104093664241    0.0610409366424     0.9444854714867   0.
↪06104093664241    386.9519025016
0.01481511454072    0.01481511454072    0.01481511454072    0.7670183371838   0.
↪01481511454072    2.643589195991
0.005238876768342   0.005238876768342   0.005238876768341   0.6766638370173   0.
↪005238876768342   11.29316821413
0.001567745187159   0.001567745187159   0.001567745187159   0.7265977484523   0.
↪001567745187159   12.32846209581
0.0001184080214619  0.0001184080214619  0.0001184080214619  0.9328477549533   0.
↪0001184080214619  6.401954889258
1.02649122158e-05   1.026491221577e-05  1.026491221577e-05  0.959846524846    1.
↪026491221577e-05  6.140180339771
2.564197796788e-08  2.564197949727e-08  2.564197949173e-08  0.9988486248561   2.
↪56419794805e-08   6.06660177389
1.282243371032e-12  1.282166817908e-12  1.282163560611e-12  0.9999499974762   1.
↪282163552715e-12  6.066591804033
Optimization terminated successfully.
         Current function value: 6.066592
         Iterations: 8
```

```
-- 2020-11-09 11:20:46 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪             506.4464461439
0.1717574295871     0.1717574295871     0.1717574295871     0.8319963473544   0.
↪1717574295871     268.7406800963
0.06638577511032    0.06638577511032    0.06638577511033    0.6490768395246   0.
↪06638577511033    376.5617800925
0.0108119991109     0.01081199911089    0.01081199911089    0.8457086649906   0.
↪01081199911089    156.44831956
0.006166341998836   0.006166341998834   0.006166341998835   0.4575243211097   0.
↪006166341998835   243.5697308399
```

(continues on next page)

(continued from previous page)

```
0.0005157722017999  0.0005157722017955  0.0005157722017955  0.9303097249327  0.
↪0005157722018036  353.933860755
7.752032087913e-08  7.75203208986e-08   7.752032089586e-08  0.9998584907938  7.
↪75203208902e-08   354.2607816225
1.875610793654e-11  1.875609893145e-11  1.875610020548e-11  0.9997580492617  1.
↪875613913502e-11  354.2617798007
Optimization terminated successfully.
         Current function value: 354.261780
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             506.4464461439
0.1079826576971     0.1079826576971     0.1079826576971     0.8981297430914  0.
↪1079826576971     201.9066839097
0.04065945390789    0.04065945390789    0.04065945390789    0.6593890232973  0.
↪04065945390789    434.9407529189
0.01889246737726    0.01889246737727    0.01889246737727    0.5585864512458  0.
↪01889246737727    712.8967272059
0.001989080291767   0.001989080291864   0.001989080291864   0.9101975479096  0.
↪001989080291929   1386.629312789
2.319893722758e-06  2.31989372302e-06   2.319893722997e-06  0.9988740155508  2.
↪319893723235e-06  1475.934762389
2.096073130738e-10  2.096071489352e-10  2.096071783066e-10  0.9999096479509  2.
↪096074566229e-10  1476.090721225
Optimization terminated successfully.
         Current function value: 1476.090721
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             889.6273141593
0.0609484122893     0.06094841228931    0.0609484122893     0.9446306809213  0.
↪06094841228931    507.5981904767
0.01403712586721    0.01403712586721    0.01403712586721    0.7774983036897  0.
↪01403712586721    2.560790425808
0.004916790985586   0.004916790985586   0.004916790985586   0.6812990636392  0.
↪004916790985587   14.46599029197
0.001087493288514   0.001087493288517   0.001087493288517   0.8126332143986  0.
↪001087493288517   17.14374653765
0.0004935333934202  0.0004935333934214  0.0004935333934214  0.5558253633026  0.
↪0004935333934214  10.54120251124
0.0001121697661281  0.000112169766129   0.000112169766129   0.817643306912   0.
↪000112169766129   4.9896729167
2.712089127881e-05  2.712089127903e-05  2.712089127903e-05  0.7970501144483  2.
↪712089127903e-05  4.324997088558
1.122718419375e-07  1.122718402016e-07  1.122718401965e-07  0.99597588406    1.
↪122718401941e-07  4.000800850778
5.614947446286e-12  5.615029242581e-12  5.615032519692e-12  0.9999499875898  5.
↪615033454619e-12  3.999950693022
2.903000170299e-16  2.886166226175e-16  2.808194717321e-16  0.9999499790221  2.
↪807865903642e-16  3.999950650496
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
```

(continues on next page)

(continued from previous page)

```
1.0                 1.0                 1.0                 -                1.0                 ␣
↪            889.6273141593
0.06073185177466    0.06073185177467    0.06073185177466    0.9450179496586  0.
↪06073185177466    760.7133777158
0.00617243956643    0.006172439566431   0.006172439566431   0.8994093979063  0.
↪006172439566431   1.525116036565
0.002290102597025   0.002290102597025   0.002290102597025   0.6678934240013  0.
↪002290102597025   13.23877514086
0.0007463324176618  0.0007463324176611  0.0007463324176611  0.7077017235652  0.
↪0007463324176608  10.86935158651
0.0003923306880488  0.0003923306880484  0.0003923306880484  0.488114542318   0.
↪0003923306880483  6.236673613644
2.432640260798e-05  2.432640260791e-05  2.432640260791e-05  1.0              2.
↪43264026079e-05   0.3563903987643
6.281868977076e-08  6.281868977357e-08  6.281868977105e-08  0.9991491899851  6.
↪281868977102e-08  0.0001119022044274
3.378349078595e-12  3.378347740195e-12  3.378348155213e-12  0.9999462206524  3.
↪378348155213e-12  6.056769044982e-09
2.306615405556e-13  2.306653965292e-13  2.306614266427e-13  0.9320769632577  2.
↪306614266427e-13  4.595744934239e-10
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 9
```

```
-- 2020-11-09 11:20:54 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                 ␣
↪            506.4464461439
0.1717574295871     0.1717574295871     0.1717574295871     0.8319963473544  0.
↪1717574295871     268.7406800963
0.06638577511032    0.06638577511032    0.06638577511033    0.6490768395246  0.
↪06638577511033    376.5617800925
0.0108119991109     0.01081199911089    0.01081199911089    0.8457086649906  0.
↪01081199911089    156.44831956
0.006166341998836   0.006166341998834   0.006166341998835   0.4575243211097  0.
↪006166341998835   243.5697308399
0.0005157722017999  0.0005157722017955  0.0005157722017955  0.9303097249327  0.
↪0005157722018036  353.933860755
7.752032087913e-08  7.75203208986e-08   7.752032089586e-08  0.9998584907938  7.
↪75203208902e-08   354.2607816225
1.875610793654e-11  1.875609893145e-11  1.875610020548e-11  0.9997580492617  1.
↪875613913502e-11  354.2617798007
Optimization terminated successfully.
         Current function value: 354.261780
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                 ␣
↪            506.4464461439
0.1079826576971     0.1079826576971     0.1079826576971     0.8981297430914  0.
↪1079826576971     201.9066839097
0.04065945390789    0.04065945390789    0.04065945390789    0.6593890232973  0.
↪04065945390789    434.9407529189
0.01889246737726    0.01889246737727    0.01889246737727    0.5585864512458  0.
↪01889246737727    712.8967272059
```

(continues on next page)

(continued from previous page)

```
0.001989080291767   0.001989080291864   0.001989080291864   0.9101975479096  0.
↪001989080291929   1386.629312789
2.319893722758e-06  2.31989372302e-06   2.319893722997e-06  0.9988740155508  2.
↪319893723235e-06  1475.934762389
2.096073130738e-10  2.096071489352e-10  2.096071783066e-10  0.9999096479509  2.
↪096074566229e-10  1476.090721225
Optimization terminated successfully.
         Current function value: 1476.090721
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             889.6273141593
0.0609484122893     0.06094841228931    0.0609484122893     0.9446306809213  0.
↪06094841228931    507.5981904767
0.01403712586721    0.01403712586721    0.01403712586721    0.7774983036897  0.
↪01403712586721    2.560790425808
0.004916790985586   0.004916790985586   0.004916790985586   0.6812990636392  0.
↪004916790985587   14.46599029197
0.001087493288514   0.001087493288517   0.001087493288517   0.8126332143986  0.
↪001087493288517   17.14374653765
0.0004935333934202  0.0004935333934214  0.0004935333934214  0.5558253633026  0.
↪0004935333934214  10.54120251124
0.0001121697661281  0.000112169766129   0.000112169766129   0.817643306912   0.
↪000112169766129   4.9896729167
2.712089127881e-05  2.712089127903e-05  2.712089127903e-05  0.7970501144483  2.
↪712089127903e-05  4.324997088558
1.122718419375e-07  1.122718402016e-07  1.122718401965e-07  0.99597588406    1.
↪122718401941e-07  4.000800850778
5.614947446286e-12  5.615029242581e-12  5.615032519692e-12  0.9999499875898  5.
↪615033454619e-12  3.999950693022
2.903000170299e-16  2.886166226175e-16  2.808194717321e-16  0.9999499790221  2.
↪807865903642e-16  3.999950650496
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             889.6273141593
0.06073185177466    0.06073185177467    0.06073185177466    0.9450179496586  0.
↪06073185177466    760.7133777158
0.00617243956643    0.006172439566431   0.006172439566431   0.8994093979063  0.
↪006172439566431   1.525116036565
0.002290102597025   0.002290102597025   0.002290102597025   0.6678934240013  0.
↪002290102597025   13.23877514086
0.0007463324176618  0.0007463324176611  0.0007463324176611  0.7077017235652  0.
↪0007463324176608  10.86935158651
0.0003923306880488  0.0003923306880484  0.0003923306880484  0.488114542318   0.
↪0003923306880483  6.236673613644
2.432640260798e-05  2.432640260791e-05  2.432640260791e-05  1.0              2.
↪43264026079e-05   0.3563903987643
6.281868977076e-08  6.281868977357e-08  6.281868977105e-08  0.9991491899851  6.
↪281868977102e-08  0.0001119022044274
3.378349078595e-12  3.378347740195e-12  3.378348155213e-12  0.9999462206524  3.
↪378348155213e-12  6.056769044982e-09
2.306615405556e-13  2.306653965292e-13  2.306614266427e-13  0.9320769632577  2.
↪306614266427e-13  4.595744934239e-10
```

(continues on next page)

(continued from previous page)

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 9
```

```
-- 2020-11-09 11:21:02 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0            ␣
↪             568.8949309456
0.1574999940443     0.1574999940443     0.1574999940443     0.8462995490748  0.
↪1574999940443      259.3692323549
0.07523425397025    0.07523425397025    0.07523425397025    0.5547720211262  0.
↪07523425397025     453.5904211877
0.01015218722862    0.01015218722862    0.01015218722862    0.8831324835902  0.
↪01015218722862     172.5109988087
0.005597873343747   0.005597873343747   0.005597873343747   0.4782155730769  0.
↪005597873343747    257.1812475882
0.0004682435100659  0.0004682435100634  0.0004682435100634  0.9301215087302  0.
↪000468243510069    357.0084554784
6.611502592752e-08  6.611502591615e-08  6.611502591873e-08  0.999863385888   6.
↪611502594636e-08   355.7877874388
7.555502313764e-12  7.555508723405e-12  7.555505999397e-12  0.9998857217433  7.
↪555493074812e-12   355.7884668282
Optimization terminated successfully.
         Current function value: 355.788467
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0            ␣
↪             568.8949309456
0.07655552550188    0.07655552550188    0.07655552550188    0.9326533055761  0.
↪07655552550188     158.9571104913
0.02915610316818    0.02915610316819    0.02915610316819    0.6583861263339  0.
↪02915610316818     366.3114833866
0.01377238842158    0.01377238842158    0.01377238842158    0.5494198953479  0.
↪01377238842158     509.1861831748
0.002113311162461   0.002113311162461   0.002113311162461   0.8672288939288  0.
↪002113311162461    954.4881944043
1.802801647426e-06  1.802801647524e-06  1.802801647519e-06  0.9992160883437  1.
↪802801646136e-06   1007.960176974
1.128093610417e-10  1.128094830078e-10  1.128094936008e-10  0.9999374254592  1.
↪128097428918e-10   1008.067349764
Optimization terminated successfully.
         Current function value: 1008.067350
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0            ␣
↪             1027.252465794
0.06092848901518    0.06092848901523    0.06092848901522    0.9446907062507  0.
↪06092848901524     639.5911028223
0.01301865096831    0.01301865096832    0.01301865096832    0.7926600108064  0.
↪01301865096833     2.590461755952
0.00456587722928    0.004565877229283   0.004565877229283   0.6815721939632  0.
↪004565877229284    18.42445056578
```

(continues on next page)

(continued from previous page)

```
0.001192620177961   0.001192620177954   0.001192620177954   0.7720888734274  0.
↪001192620177954   21.2766378272
0.0004150762782539  0.0004150762782515  0.0004150762782515  0.6551231023534  0.
↪0004150762782516  10.90714854956
8.30397207983e-05   8.303972079784e-05  8.303972079783e-05  0.8551088375548  8.
↪303972079786e-05  4.883155784925
2.345269716473e-05  2.345269716461e-05  2.34526971646e-05   0.7548298132573  2.
↪345269716461e-05  4.346885094868
1.371159312911e-07  1.371159321168e-07  1.371159321194e-07  0.994331926271   1.
↪371159321123e-07  4.001183377585
6.859845575537e-12  6.859856824223e-12  6.859863126594e-12  0.9999499723157  6.
↪85986409236e-12   3.999950686436
6.215051942361e-16  3.43521130371e-16   3.430265101783e-16  0.9999499985889  3.
↪431205195905e-16  3.999950624744
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0    ␣
↪           513.6262328969
0.06112045297154    0.06112045297154    0.06112045297151    0.9426671718798  0.
↪06112045297154    286.961037666
0.0186024953335     0.0186024953335     0.01860249533349    0.7059824756003  0.
↪0186024953335     1.911789377935
0.006446621823764   0.006446621823763   0.00644662182376    0.6910883581126  0.
↪006446621823763   10.97980543166
0.0005501537566849  0.0005501537566955  0.0005501537566952  0.9377765193638  0.
↪0005501537566949  10.00501917633
4.007383875704e-06  4.007383875795e-06  4.007383875804e-06  0.9951127586026  4.
↪007383875797e-06  6.687770636962
2.007342726257e-10  2.007343535262e-10  2.007343506552e-10  0.9999499088827  2.
↪007343503785e-10  6.666585422706
2.161448037787e-10  2.007343535257e-10  2.007343507631e-10  2.456113364069e-461.
↪961302857585e-10  6.666585422564
6.009588090216e-12  2.723883580422e-12  2.723885134159e-12  0.9866729468426  2.
↪685910112544e-12  6.666584374146
Optimization terminated successfully.
         Current function value: 6.666584
         Iterations: 8
```

```
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'cholesky':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'cholesky'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'sym_pos':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'sym_pos'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/anaconda3/lib/python3.8/site-packages/scipy/optimize/_linprog_ip.py:
↪116: LinAlgWarning: Ill-conditioned matrix (rcond=1.62544e-36): result may not be␣
↪accurate.
  return sp.linalg.solve(M, r, sym_pos=sym_pos)
```

(continues on next page)

(continued from previous page)

```
-- 2020-11-09 11:21:10 - muse.mca - WARNING
Check growth constraints for wind.

Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0           ␣
↪             568.8949309456
0.1574999940443     0.1574999940443     0.1574999940443     0.8462995490748  0.
↪1574999940443      259.3692323549
0.07523425397025    0.07523425397025    0.07523425397025    0.5547720211262  0.
↪07523425397025     453.5904211877
0.01015218722862    0.01015218722862    0.01015218722862    0.8831324835902  0.
↪01015218722862     172.5109988087
0.005597873343747   0.005597873343747   0.005597873343747   0.4782155730769  0.
↪005597873343747    257.1812475882
0.0004682435100659  0.0004682435100634  0.0004682435100634  0.9301215087302  0.
↪000468243510069    357.0084554784
6.611502592752e-08  6.611502591615e-08  6.611502591873e-08  0.999863385888   6.
↪611502594636e-08   355.7877874388
7.555502313764e-12  7.555508723405e-12  7.555505999397e-12  0.9998857217433  7.
↪555493074812e-12   355.7884668282
Optimization terminated successfully.
         Current function value: 355.788467
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0           ␣
↪             568.8949309456
0.07655552550188    0.07655552550188    0.07655552550188    0.9326533055761  0.
↪07655552550188     158.9571104913
0.02915610316818    0.02915610316819    0.02915610316819    0.6583861263339  0.
↪02915610316818     366.3114833866
0.01377238842158    0.01377238842158    0.01377238842158    0.5494198953479  0.
↪01377238842158     509.1861831748
0.002113311162461   0.002113311162461   0.002113311162461   0.8672288939288  0.
↪002113311162461    954.4881944043
1.802801647426e-06  1.802801647524e-06  1.802801647519e-06  0.9992160883437  1.
↪802801646136e-06   1007.960176974
1.128093610417e-10  1.128094830078e-10  1.128094936008e-10  0.9999374254592  1.
↪128097428918e-10   1008.067349764
Optimization terminated successfully.
         Current function value: 1008.067350
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0           ␣
↪             1027.252465794
0.06092848901518    0.06092848901523    0.06092848901522    0.9446907062507  0.
↪06092848901524     639.5911028223
0.01301865096831    0.01301865096832    0.01301865096832    0.7926600108064  0.
↪01301865096833     2.590461755952
0.00456587722928    0.004565877229283   0.004565877229283   0.6815721939632  0.
↪004565877229284    18.42445056578
0.001192620177961   0.001192620177954   0.001192620177954   0.7720888734274  0.
↪001192620177954    21.2766378272
0.0004150762782539  0.0004150762782515  0.0004150762782515  0.6551231023534  0.
↪0004150762782516   10.90714854956
```

(continues on next page)

(continued from previous page)

```
8.30397207983e-05   8.303972079784e-05  8.303972079783e-05  0.8551088375548  8.
↪303972079786e-05  4.883155784925
2.345269716473e-05  2.345269716461e-05  2.34526971646e-05   0.7548298132573  2.
↪345269716461e-05  4.346885094868
1.371159312911e-07  1.371159321168e-07  1.371159321194e-07  0.994331926271   1.
↪371159321123e-07  4.001183377585
6.859845575537e-12  6.859856824223e-12  6.859863126594e-12  0.9999499723157  6.
↪85986409236e-12   3.999950686436
6.215051942361e-16  3.43521130371e-16   3.430265101783e-16  0.9999499985889  3.
↪431205195905e-16  3.999950624744
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0          ␣
↪             513.6262328969
0.06112045297154    0.06112045297154    0.06112045297151    0.9426671718798  0.
↪06112045297154    286.961037666
0.0186024953335     0.0186024953335     0.01860249533349    0.7059824756003  0.
↪0186024953335     1.911789377935
0.006446621823764   0.006446621823763   0.00644662182376    0.6910883581126  0.
↪006446621823763   10.97980543166
0.0005501537566849  0.0005501537566955  0.0005501537566952  0.9377765193638  0.
↪0005501537566949  10.00501917633
4.007383875704e-06  4.007383875795e-06  4.007383875804e-06  0.9951127586026  4.
↪007383875797e-06  6.687770636962
2.007342726257e-10  2.007343535262e-10  2.007343506552e-10  0.9999499088827  2.
↪007343503785e-10  6.666585422706
2.161448037787e-10  2.007343535257e-10  2.007343507631e-10  2.456113364069e-461.
↪961302857585e-10  6.666585422564
6.009588090216e-12  2.723883580422e-12  2.723885134159e-12  0.9866729468426  2.
↪685910112544e-12  6.666584374146
Optimization terminated successfully.
         Current function value: 6.666584
         Iterations: 8
```

```
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'cholesky':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'cholesky'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'sym_pos':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'sym_pos'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/anaconda3/lib/python3.8/site-packages/scipy/optimize/_linprog_ip.py:
↪116: LinAlgWarning: Ill-conditioned matrix (rcond=1.62544e-36): result may not be␣
↪accurate.
  return sp.linalg.solve(M, r, sym_pos=sym_pos)
-- 2020-11-09 11:21:16 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                ␣
↪             647.4402248427
0.1042702416317     0.1042702416317     0.1042702416317     0.9044651903652  0.
↪1042702416317      197.7436068274
0.04330783922975    0.04330783922975    0.04330783922975    0.6213338928986  0.
↪04330783922975     554.8458695929
0.02029339907042    0.02029339907042    0.02029339907042    0.5546164251713  0.
↪02029339907042     887.503418141
0.002059630009005   0.002059630008646   0.002059630008646   0.9123945707354  0.
↪002059630008855    1773.497308596
1.96164830749e-06   1.961648306624e-06  1.961648306623e-06  0.9995475315428  1.
↪96164830693e-06    1899.795955905
1.183904657193e-10  1.183907029537e-10  1.183907180056e-10  0.9999396473343  1.
↪183903667831e-10   1900.055708261
Optimization terminated successfully.
         Current function value: 1900.055708
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                ␣
↪             1199.542590081
0.06080970203684    0.06080970203684    0.06080970203685    0.9448463893022  0.
↪06080970203684     802.3521760538
0.01206790508619    0.01206790508619    0.01206790508619    0.8067148881887  0.
↪01206790508619     2.242531350203
0.004235304009507   0.004235304009507   0.004235304009507   0.6829892140296  0.
↪004235304009507    22.93309435174
0.001076107298274   0.001076107298272   0.001076107298272   0.7811880414723  0.
↪001076107298272    27.34472817154
0.0001381467026759  0.0001381467026756  0.0001381467026756  0.9534954297498  0.
↪0001381467026756   2.164474733514
0.0001283068196922  0.0001283068196919  0.0001283068196919  0.07280278667804 0.
↪0001283068196919   2.01175576967
1.709535377769e-06  1.709535377765e-06  1.709535377765e-06  0.9870423816225  1.
↪709535377765e-06   0.04804313560572
9.48541628326e-11   9.485418648369e-11  9.485418526931e-11  0.9999445934679  9.
↪485418526932e-11   2.66445492392e-06
8.795947527601e-12  8.795912717956e-12  8.795907275833e-12  0.9086823994024  8.
↪795907275834e-12   2.473302892196e-07
8.361396439476e-12  8.361363544519e-12  8.361358288329e-12  0.05441434596209 8.
↪361358288329e-12   2.360712463483e-07
1.014674559722e-12  1.014680661495e-12  1.014676648034e-12  0.8808930741304  1.
↪014676648034e-12   2.943765656853e-08
7.649517191012e-13  7.649501663219e-13  7.649518441425e-13  0.266685424728   7.
↪649518441426e-13   2.292855842523e-08
2.251056337644e-13  2.251034423868e-13  2.250976823208e-13  0.7199831364733  2.
↪250976823208e-13   9.225506253453e-09
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                ␣
↪             599.7712950406
0.06108522397884    0.06108522397884    0.06108522397884    0.9426921917133  0.
↪06108522397884     370.5338700223
```

(continues on next page)

(continued from previous page)

```
0.01718424163858    0.01718424163858    0.01718424163858    0.7269400839096  0.
↪01718424163858    1.816590685562
0.006448030539302   0.006448030539302   0.006448030539302   0.6618055567806  0.
↪006448030539302   13.33233954634
0.0007451641812888  0.0007451641813056  0.0007451641813056  0.9128367304104  0.
↪0007451641813056  13.46011423965
0.0002051849835872  0.0002051849835913  0.0002051849835913  0.726248232922   0.
↪0002051849835914  8.092503300126
3.421589486985e-06  3.421589487287e-06  3.421589487279e-06  1.0              3.
↪421589487245e-06  5.943322696325
2.883371020626e-10  2.88337212738e-10   2.883372076584e-10  0.999916005146   2.
↪883372071457e-10  5.933260704906
1.437342489421e-14  1.441626748189e-14  1.441700316141e-14  0.999949999407   1.
↪441794244359e-14  5.933260075848
Optimization terminated successfully.
         Current function value: 5.933260
         Iterations: 8
```

```
-- 2020-11-09 11:21:23 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0       ␣
↪             647.4402248427
0.1042702416317     0.1042702416317     0.1042702416317     0.9044651903652  0.
↪1042702416317     197.7436068274
0.04330783922975    0.04330783922975    0.04330783922975    0.6213338928986  0.
↪04330783922975    554.8458695929
0.02029339907042    0.02029339907042    0.02029339907042    0.5546164251713  0.
↪02029339907042    887.503418141
0.002059630009005   0.002059630008646   0.002059630008646   0.9123945707354  0.
↪002059630008855   1773.497308596
1.96164830749e-06   1.961648306624e-06  1.961648306623e-06  0.9995475315428  1.
↪96164830693e-06   1899.795955905
1.183904657193e-10  1.183907029537e-10  1.183907180056e-10  0.9999396473343  1.
↪183903667831e-10  1900.055708261
Optimization terminated successfully.
         Current function value: 1900.055708
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0       ␣
↪             1199.542590081
0.06080970203684    0.06080970203684    0.06080970203685    0.9448463893022  0.
↪06080970203684    802.3521760538
0.01206790508619    0.01206790508619    0.01206790508619    0.8067148881887  0.
↪01206790508619    2.242531350203
0.004235304009507   0.004235304009507   0.004235304009507   0.6829892140296  0.
↪004235304009507   22.93309435174
0.001076107298274   0.001076107298272   0.001076107298272   0.7811880414723  0.
↪001076107298272   27.34472817154
0.0001381467026759  0.0001381467026756  0.0001381467026756  0.9534954297498  0.
↪0001381467026756  2.164474733514
0.0001283068196922  0.0001283068196919  0.0001283068196919  0.07280278667804 0.
↪0001283068196919  2.01175576967
1.709535377769e-06  1.709535377765e-06  1.709535377765e-06  0.9870423816225  1.
↪709535377765e-06  0.04804313560572
```

(continues on next page)

(continued from previous page)

```
9.48541628326e-11   9.485418648369e-11  9.485418526931e-11  0.9999445934679  9.
↪485418526932e-11  2.66445492392e-06
8.795947527601e-12  8.795912717956e-12  8.795907275833e-12  0.9086823994024  8.
↪795907275834e-12  2.473302892196e-07
8.361396439476e-12  8.361363544519e-12  8.361358288329e-12  0.05441434596209 8.
↪361358288329e-12  2.360712463483e-07
1.014674559722e-12  1.014680661495e-12  1.014676648034e-12  0.8808930741304  1.
↪014676648034e-12  2.943765656853e-08
7.649517191012e-13  7.649501663219e-13  7.649518441425e-13  0.266685424728   7.
↪649518441426e-13  2.292855842523e-08
2.251056337644e-13  2.251034423868e-13  2.250976823208e-13  0.7199831364733  2.
↪250976823208e-13  9.225506253453e-09
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪           599.7712950406
0.06108522397884    0.06108522397884    0.06108522397884    0.9426921917133  0.
↪06108522397884    370.5338700223
0.01718424163858    0.01718424163858    0.01718424163858    0.7269400839096  0.
↪01718424163858    1.816590685562
0.006448030539302   0.006448030539302   0.006448030539302   0.6618055567806  0.
↪006448030539302   13.33233954634
0.0007451641812888  0.0007451641813056  0.0007451641813056  0.9128367304104  0.
↪0007451641813056  13.46011423965
0.0002051849835872  0.0002051849835913  0.0002051849835913  0.726248232922   0.
↪0002051849835914  8.092503300126
3.421589486985e-06  3.421589487287e-06  3.421589487279e-06  1.0              3.
↪421589487245e-06  5.943322696325
2.883371020626e-10  2.88337212738e-10   2.883372076584e-10  0.999916005146   2.
↪883372071457e-10  5.933260704906
1.437342489421e-14  1.441626748189e-14  1.441700316141e-14  0.999949999407   1.
↪441794244359e-14  5.933260075848
Optimization terminated successfully.
         Current function value: 5.933260
         Iterations: 8
```

```
-- 2020-11-09 11:21:29 - muse.mca - WARNING
Check growth constraints for wind.
```

We can now check that the simulation has created the files that we expect. We also check that our "Hello, world!" message has printed:

[5]:
```
all_txt_files = sorted((Path() / "Results").glob("Residential*.txt"))
assert "Hello world!" in all_txt_files[0].read_text()
all_txt_files
```

[5]:
```
[PosixPath('Results/ResidentialConsumption_Zero2020.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2025.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2030.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2035.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2040.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2045.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2050.txt')]
```

Our model output the files we were expecting and passed the `assert` statement, meaning that it could find the "Hello world!" messages in the outputs.

7.1.2 Adding TOML parameters to the outputs

It would be useful if we could pass parameters from the TOML file to our new functions `consumption_zero` and `text_dump`. For example, in our previous iteration the consumption output was aggregating the data by `"timeslice"`, by hardcoding the variable. We can pass a parameter which could do this by setting the `sum_over` parameter to be `True`. In addition, we could change the message output by a new `text_dump` function.

Not all hooks are this flexible (for historical reasons, rather than any intrinsic difficulty). However, for outputs, we can do this as follows:

[6]:
```
@register_output_quantity(overwrite=True)
def consumption_zero(
    market: Dataset,
    capacity: DataArray,
    technologies: Dataset,
    sum_over: Optional[List[Text]] = None,
    drop: Optional[List[Text]] = None,
    rounding: int = 4,
):
    """Current consumption."""
    result = (
        market_quantity(market.consumption, sum_over=sum_over, drop=drop)
        .rename("consumption")
        .to_dataframe()
        .round(rounding)
    )
    return result


@register_output_sink(name="txt", overwrite=True)
@sink_to_file(".txt")
def text_dump(
    data: Any,
    filename: Text,
    msg : Optional[Text] = "Hello, world!"
) -> None:
    from pathlib import Path
    Path(filename).write_text(f"{msg}\n\n{data}")
```

We simply added parameters as arguments to both of our functions: `consumption_zero` and `text_dump`.

Note: The overwrite argument allows us to overwrite previously defined registered functions. This is useful in a notebook such as this. But it should not be used in general. If overwrite were false, then the code would issue a warning and it would leave the TOML to refer to the original functions at the beginning of the notebook. This is useful when using custom modules.

Now we can modify the output section to take additional arguments:

```
[[sectors.commercial.outputs]]
quantity.name = "consumption_zero"
quantity.sum_over = "timeslice"
sink.name = "txt"
sink.filename = "{cwd}/{default_output_dir}/{Sector}{Quantity}{year}{suffix}"
```

(continues on next page)

(continued from previous page)

```
sink.msg = "Hello, you!"
sink.overwrite = True
```

Here, we still want to use the `consumption_zero` function and the `txt` sink. But we would like to change the message from "Hello world!" to "Hello you!" within the `TOML` file.

Now, both sink and quantity are dictionaries which can take any number of arguments. Previously, we were using a shorthand for convenience. Again, we create a new settings file, and run this with our new parameters, which interface with our new functions.

[7]:
```
from pathlib import Path
from toml import load, dump
from muse import examples

model_path = examples.copy_model(overwrite=True)
settings = load(model_path / "settings.toml")
settings["sectors"]["residential"]["outputs"] = [
    {
        "quantity":{
            "name": "consumption_zero",
            "sum_over": "timeslice"
        },
        "sink":{
            "name": "txt",
            "filename": "{cwd}/{default_output_dir}/{Sector}{Quantity}{year}{suffix}",
            "msg": "Hello, you!",
            "overwrite": True,
        }

    }
]

dump(settings, (model_path / "modified_settings_2.toml").open("w"))
settings
```

[7]:
```
{'time_framework': [2020, 2025, 2030, 2035, 2040, 2045, 2050],
 'foresight': 5,
 'regions': ['R1'],
 'interest_rate': 0.1,
 'interpolation_mode': 'Active',
 'log_level': 'info',
 'equilibrium_variable': 'demand',
 'maximum_iterations': 100,
 'tolerance': 0.1,
 'tolerance_unmet_demand': -0.1,
 'outputs': [{'quantity': 'prices',
   'sink': 'aggregate',
   'filename': '{cwd}/{default_output_dir}/MCA{Quantity}.csv'},
  {'quantity': 'capacity',
   'sink': 'aggregate',
   'filename': '{cwd}/{default_output_dir}/MCA{Quantity}.csv'}],
 'carbon_budget_control': {'budget': []},
 'global_input_files': {'projections': '{path}/input/Projections.csv',
  'global_commodities': '{path}/input/GlobalCommodities.csv'},
 'sectors': {'residential': {'type': 'default',
   'priority': 1,
   'dispatch_production': 'share',
```

(continues on next page)

(continued from previous page)

```
    'technodata': '{path}/technodata/residential/Technodata.csv',
    'commodities_in': '{path}/technodata/residential/CommIn.csv',
    'commodities_out': '{path}/technodata/residential/CommOut.csv',
    'subsectors': {'retro_and_new': {'agents': '{path}/technodata/Agents.csv',
      'existing_capacity': '{path}/technodata/residential/ExistingCapacity.csv',
      'lpsolver': 'scipy',
      'constraints': ['max_production',
       'max_capacity_expansion',
       'demand',
       'search_space'],
      'demand_share': 'new_and_retro',
      'forecast': 5}},
    'outputs': [{'quantity': {'name': 'consumption_zero',
       'sum_over': 'timeslice'},
      'sink': {'name': 'txt',
       'filename': '{cwd}/{default_output_dir}/{Sector}{Quantity}{year}{suffix}',
       'msg': 'Hello, you!',
       'overwrite': True}}],
    'interactions': [{'net': 'new_to_retro', 'interaction': 'transfer'}]},
   'power': {'type': 'default',
    'priority': 2,
    'dispatch_production': 'share',
    'technodata': '{path}/technodata/power/Technodata.csv',
    'commodities_in': '{path}/technodata/power/CommIn.csv',
    'commodities_out': '{path}/technodata/power/CommOut.csv',
    'subsectors': {'retro_and_new': {'agents': '{path}/technodata/Agents.csv',
      'existing_capacity': '{path}/technodata/power/ExistingCapacity.csv',
      'lpsolver': 'scipy'}},
    'outputs': [{'filename': '{cwd}/{default_output_dir}/{Sector}/{Quantity}/{year}
↪{suffix}',
      'quantity': 'capacity',
      'sink': 'csv',
      'overwrite': True}],
    'interactions': [{'net': 'new_to_retro', 'interaction': 'transfer'}]},
   'gas': {'type': 'default',
    'priority': 3,
    'dispatch_production': 'share',
    'technodata': '{path}/technodata/gas/Technodata.csv',
    'commodities_in': '{path}/technodata/gas/CommIn.csv',
    'commodities_out': '{path}/technodata/gas/CommOut.csv',
    'subsectors': {'retro_and_new': {'agents': '{path}/technodata/Agents.csv',
      'existing_capacity': '{path}/technodata/gas/ExistingCapacity.csv',
      'lpsolver': 'scipy'}},
    'outputs': [{'filename': '{cwd}/{default_output_dir}/{Sector}/{Quantity}/{year}
↪{suffix}',
      'quantity': 'capacity',
      'sink': 'csv',
      'overwrite': True}],
    'interactions': [{'net': 'new_to_retro', 'interaction': 'transfer'}]},
   'residential_presets': {'type': 'presets',
    'priority': 0,
    'consumption_path': '{path}/technodata/preset/*Consumption.csv'}},
  'timeslices': {'all-year': {'all-week': {'night': 1460,
     'morning': 1460,
     'afternoon': 1460,
     'early-peak': 1460,
     'late-peak': 1460,
```

(continues on next page)

(continued from previous page)

```
    'evening': 1460}},
  'level_names': ['month', 'day', 'hour']}}
```

We then run the simulation again:

[10]:
```
mca = MCA.factory(model_path / "modified_settings_2.toml")
mca.run();
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0           ␣
↪            148.9679256735
0.2598249156018     0.2598249156018     0.2598249156018     0.7495200004432  0.
↪2598249156018      120.5733849622
0.02399956829695    0.02399956829695    0.02399956829695    0.9210391224498  0.
↪02399956829695     4.780663765494
0.0181364461758     0.0181364461758     0.0181364461758     0.2509588065043  0.
↪0181364461758      7.107141691547
0.01499350833129    0.01499350833129    0.01499350833129    0.1921973185437  0.
↪01499350833129     70.77614035582
0.004968295711366   0.004968295711367   0.004968295711366   0.6857131120066  0.
↪004968295711366    164.7472224003
0.0006443120819652  0.0006443120819642  0.000644312081964   0.8804718592549  0.
↪0006443120819672   289.7109372802
2.427431365313e-06  2.427431365276e-06  2.42743136527e-06   0.9976309182175  2.
↪427431365399e-06   310.7437190082
1.214286379284e-10  1.214286245985e-10  1.214286217581e-10  0.9999499778566  1.
↪214286284187e-10   310.7859704917
Optimization terminated successfully.
         Current function value: 310.785970
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0           ␣
↪            148.9679256735
0.1806451257467     0.1806451257467     0.1806451257467     0.8281847212959  0.
↪1806451257467      169.6037982942
0.02684129624378    0.02684129624378    0.02684129624378    0.8635116331789  0.
↪02684129624378     123.2904723181
0.01081107082374    0.01081107082373    0.01081107082373    0.6279145428497  0.
↪01081107082373     293.3552576002
0.00150335333755    0.001503353337531   0.001503353337531   0.8785435425234  0.
↪001503353337582    618.5754737903
4.126548293386e-06  4.126548293452e-06  4.126548293469e-06  0.99729939758    4.
↪126548293473e-06   673.2429034259
2.063813940498e-10  2.063814559148e-10  2.063814538311e-10  0.9999499869317  2.
↪06381853248e-10    673.369600975
Optimization terminated successfully.
         Current function value: 673.369601
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0           ␣
↪            225.7506433631
0.07675788061753    0.07675788061753    0.07675788061751    0.9271368109475  0.
↪07675788061753     1.307751786207
0.01889464099889    0.01889464099889    0.01889464099888    0.7970949255449  0.
↪01889464099889     19.4989221165
```

(continues on next page)

(continued from previous page)

```
0.007543783963398   0.007543783963394   0.007543783963392   0.6153162486386  0.
↪007543783963394   16.52048983639
0.002504946781023   0.002504946781021   0.00250494678102    0.7004074675406  0.
↪002504946781021   72.319459457
0.0004445444355394  0.000444544435539   0.0004445444355388  0.8689995047748  0.
↪000444544435539   423.758860583
1.214501095331e-05  1.214501095325e-05  1.214501095324e-05  0.9820606335229  1.
↪214501095322e-05  675.1646942955
1.070439776445e-09  1.07043978707e-09   1.070439766673e-09  0.9999120772687  1.
↪070439801419e-09  681.9179209593
5.353929135063e-14  5.353592310399e-14  5.35373239097e-14   0.9999499866071  5.
↪352230256347e-14  681.9185224492
Optimization terminated successfully.
         Current function value: 681.918522
         Iterations: 8
```

```
-- 2020-11-09 11:32:00 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪            359.2443189825
0.2131118149695     0.2131118149695     0.2131118149695     0.7960343319409  0.
↪2131118149695     262.2885392799
0.05758094728718    0.05758094728718    0.05758094728718    0.7523513573813  0.
↪05758094728718    13.16175379893
0.01048349585899    0.01048349585899    0.01048349585899    0.8203940076989  0.
↪01048349585899    9.036399448741
0.009005598049604   0.009005598049604   0.009005598049604   0.1488155054953  0.
↪009005598049604   19.40296370843
0.002317604003518   0.002317604003518   0.002317604003518   0.8098673571979  0.
↪002317604003518   226.5492459765
0.0009784310820962  0.0009784310820963  0.0009784310820963  0.5991550275447  0.
↪0009784310820971  289.4684048776
0.0001326875071986  0.0001326875071986  0.0001326875071986  0.8836343167337  0.
↪0001326875071987  358.6919881748
6.688262823007e-08  6.688262823276e-08  6.688262823391e-08  0.9995454959391  6.
↪688262822145e-08  365.8223849509
3.344208692489e-12  3.34420597683e-12   3.344204125008e-12  0.9999499989235  3.
↪344177862259e-12  365.8250744356
Optimization terminated successfully.
         Current function value: 365.825074
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪            179.6221594912
0.06792119055695    0.06792119055695    0.06792119055695    0.9362017234058  0.
↪06792119055695    76.26287556551
0.009746786382626   0.009746786382626   0.009746786382626   0.8719786940257  0.
↪009746786382625   34.28929487798
0.00504956460969    0.005049564609689   0.005049564609689   0.5134005827998  0.
↪005049564609689   115.6145513358
0.00313210707068    0.003132107070663   0.003132107070663   0.3922116719677  0.
↪003132107070663   175.0444435627
0.0005331490663204  0.000533149066321   0.000533149066321   0.8631478238108  0.
↪0005331490663195  430.7265923665
```

(continues on next page)

(continued from previous page)

```
6.809758501168e-06  6.809758501084e-06  6.809758501069e-06  0.9896962642248  6.
↪809758501216e-06  482.9615875673
3.566453823927e-10  3.56645399101e-10   3.566453982014e-10  0.9999476448412  3.
↪566453742836e-10  483.66367672
Optimization terminated successfully.
         Current function value: 483.663677
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0       ␣
↪            547.5170704708
0.06099397574481    0.06099397574481    0.06099397574482    0.944387525785   0.
↪06099397574482    192.4316112656
0.01386491315737    0.01386491315737    0.01386491315737    0.7896145976147  0.
↪01386491315737    3.157754203839
0.00713143943229    0.00713143943229    0.007131439432292   0.5116778496497  0.
↪007131439432291   6.913227303095
0.0007854492963609  0.0007854492963609  0.0007854492963611  0.9057153968575  0.
↪000785449296361   6.591304425235
0.0003968642772996  0.0003968642772997  0.0003968642772998  0.5195625901748  0.
↪0003968642772996  5.218772647071
5.276614006577e-06  5.276614006577e-06  5.276614006576e-06  0.9941140946865  5.
↪276614006565e-06  5.272614338781
3.679893198137e-10  3.679893242575e-10  3.679893250277e-10  0.9999303117912  3.
↪679893357516e-10  5.266601980833
1.790517523351e-14  1.842347640038e-14  1.841384711876e-14  0.9999499611495  1.
↪844295319492e-14  5.266601636509
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0       ␣
↪            273.7585352354
0.06117571447458    0.06117571447458    0.06117571447455    0.9425787965419  0.
↪06117571447457    204.7212184456
0.009942269802024   0.009942269802025   0.00994226980202    0.8403658088806  0.
↪009942269802024   0.7760264390675
0.002147318631478   0.002147318631478   0.002147318631477   0.8277733790617  0.
↪002147318631478   3.894705138429
0.0009649606635229  0.0009649606635227  0.0009649606635223  0.5635818341147  0.
↪0009649606635226  2.700821086695
0.0002350111516202  0.0002350111516201  0.00023501115162    0.7866567764877  0.
↪0002350111516201  14.62347819316
5.857294640779e-05  5.857294640777e-05  5.857294640774e-05  0.8105927228255  5.
↪857294640777e-05  123.50656776
1.165473006364e-06  1.165473006407e-06  1.165473006407e-06  0.9835504451632  1.
↪165473006409e-06  151.6654340598
8.362816702708e-11  8.362816612134e-11  8.362816648317e-11  0.9999285085203  8.
↪36281608009e-11   152.3843167925
4.185697343433e-15  4.184037681176e-15  4.18416825892e-15   0.9999499678098  4.
↪181415889613e-15  152.3843678125
Optimization terminated successfully.
         Current function value: 152.384368
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
```

(continues on next page)

(continued from previous page)

```
1.0                 1.0                 1.0                 -                 1.0
↪             24.59260538017
0.01918847234907    0.01918847234907    0.01918847234907    0.9908439847925   0.
↪01918847234907    0.01449776485458
0.01221126742797    0.01221126742797    0.01221126742797    0.3735738645022   0.
↪01221126742797    0.1603023022985
0.009831906616685   0.009831906616685   0.009831906616685   0.2161492677935   0.
↪009831906616685   20.38072882765
0.001212407727123   0.001212407727023   0.001212407727023   0.8886055956883   0.
↪001212407727023   51.40232805869
4.730913157321e-06  4.73091315438e-06   4.730913154368e-06  0.9976178802846   4.
↪730913155107e-06  59.83474944668
2.366017130877e-10  2.366016942176e-10  2.366016879444e-10  0.9999499881554   2.
↪366016905478e-10  59.84200873085
Optimization terminated successfully.
         Current function value: 59.842009
         Iterations: 6
```

```
-- 2020-11-09 11:32:10 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step              Path
↪Parameter       Objective
1.0                 1.0                 1.0                 -                 1.0
↪             359.2443189825
0.2131118149695     0.2131118149695     0.2131118149695     0.7960343319409   0.
↪2131118149695     262.2885392799
0.05758094728718    0.05758094728718    0.05758094728718    0.7523513573813   0.
↪05758094728718    13.16175379893
0.01048349585899    0.01048349585899    0.01048349585899    0.8203940076989   0.
↪01048349585899    9.036399448741
0.009005598049604   0.009005598049604   0.009005598049604   0.1488155054953   0.
↪009005598049604   19.40296370843
0.002317604003518   0.002317604003518   0.002317604003518   0.8098673571979   0.
↪002317604003518   226.5492459765
0.0009784310820962  0.0009784310820963  0.0009784310820963  0.5991550275447   0.
↪0009784310820971  289.4684048776
0.0001326875071986  0.0001326875071986  0.0001326875071986  0.8836343167337   0.
↪0001326875071987  358.6919881748
6.688262823007e-08  6.688262823276e-08  6.688262823391e-08  0.9995454959391   6.
↪688262822145e-08  365.8223849509
3.344208692489e-12  3.34420597683e-12   3.344204125008e-12  0.9999499989235   3.
↪344177862259e-12  365.8250744356
Optimization terminated successfully.
         Current function value: 365.825074
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step              Path
↪Parameter       Objective
1.0                 1.0                 1.0                 -                 1.0
↪             179.6221594912
0.06792119055695    0.06792119055695    0.06792119055695    0.9362017234058   0.
↪06792119055695    76.26287556551
0.009746786382626   0.009746786382626   0.009746786382626   0.8719786940257   0.
↪009746786382625   34.28929487798
0.00504956460969    0.005049564609689   0.005049564609689   0.5134005827998   0.
↪005049564609689   115.6145513358
0.003132107070668   0.003132107070663   0.003132107070663   0.3922116719677   0.
↪003132107070663   175.0444435627
```

(continues on next page)

(continued from previous page)

```
0.0005331490663204  0.000533149066321   0.000533149066321   0.8631478238108  0.
↪0005331490663195  430.7265923665
6.809758501168e-06  6.809758501084e-06  6.809758501069e-06  0.9896962642248  6.
↪809758501216e-06  482.9615875673
3.566453823927e-10  3.56645399101e-10   3.566453982014e-10  0.9999476448412  3.
↪566453742836e-10  483.66367672
Optimization terminated successfully.
         Current function value: 483.663677
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            547.5170704708
0.06099397574481    0.06099397574481    0.06099397574482    0.944387525785   0.
↪06099397574482    192.4316112656
0.01386491315737    0.01386491315737    0.01386491315737    0.7896145976147  0.
↪01386491315737    3.157754203839
0.00713143943229    0.00713143943229    0.007131439432292   0.5116778496497  0.
↪007131439432291   6.913227303095
0.0007854492963609  0.0007854492963609  0.0007854492963611  0.9057153968575  0.
↪000785449296361   6.591304425235
0.0003968642772996  0.0003968642772997  0.0003968642772998  0.5195625901748  0.
↪0003968642772996  5.218772647071
5.276614006577e-06  5.276614006577e-06  5.276614006576e-06  0.9941140946865  5.
↪276614006565e-06  5.272614338781
3.679893198137e-10  3.679893242575e-10  3.679893250277e-10  0.9999303117912  3.
↪679893357516e-10  5.266601980833
1.790517523351e-14  1.842347640038e-14  1.841384711876e-14  0.9999499611495  1.
↪844295319492e-14  5.266601636509
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            273.7585352354
0.06117571447458    0.06117571447458    0.06117571447455    0.9425787965419  0.
↪06117571447457    204.7212184456
0.009942269802024   0.009942269802025   0.00994226980202    0.8403658088806  0.
↪009942269802024   0.7760264390675
0.002147318631478   0.002147318631478   0.002147318631477   0.8277733790617  0.
↪002147318631478   3.894705138429
0.0009649606635229  0.0009649606635227  0.0009649606635223  0.5635818341147  0.
↪0009649606635226  2.700821086695
0.0002350111516202  0.0002350111516201  0.00023501115162    0.7866567764877  0.
↪0002350111516201  14.62347819316
5.857294640779e-05  5.857294640777e-05  5.857294640774e-05  0.8105927228255  5.
↪857294640777e-05  123.50656776
1.165473006364e-06  1.165473006407e-06  1.165473006407e-06  0.9835504451632  1.
↪165473006409e-06  151.6654340598
8.362816702708e-11  8.362816612134e-11  8.362816648317e-11  0.9999285085203  8.
↪36281608009e-11   152.3843167925
4.185697343433e-15  4.184037681176e-15  4.18416825892e-15   0.9999499678098  4.
↪181415889613e-15  152.3843678125
Optimization terminated successfully.
         Current function value: 152.384368
         Iterations: 9
```

(continues on next page)

(continued from previous page)

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             24.59260538017
0.01918847234907    0.01918847234907    0.01918847234907    0.9908439847925  0.
↪01918847234907     0.01449776485458
0.01221126742797    0.01221126742797    0.01221126742797    0.3735738645022  0.
↪01221126742797     0.1603023022985
0.009831906616685   0.009831906616685   0.009831906616685   0.2161492677935  0.
↪009831906616685    20.38072882765
0.001212407727123   0.001212407727023   0.001212407727023   0.8886055956883  0.
↪001212407727023    51.40232805869
4.730913157321e-06  4.73091315438e-06   4.730913154368e-06  0.9976178802846  4.
↪730913155107e-06   59.83474944668
2.366017130877e-10  2.366016942176e-10  2.366016879444e-10  0.9999499881554  2.
↪366016905478e-10   59.84200873085
Optimization terminated successfully.
         Current function value: 59.842009
         Iterations: 6
```

```
-- 2020-11-09 11:32:19 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1943112264485     0.1943112264485     0.1943112264485     0.8115414580695  0.
↪1943112264485      293.1180738751
0.05737071259203    0.05737071259203    0.05737071259203    0.7291321197238  0.
↪05737071259203     17.07106219013
0.01062277327644    0.01062277327644    0.01062277327644    0.8173814867435  0.
↪01062277327644     12.44885046687
0.009064514495371   0.009064514495372   0.009064514495372   0.1550649500025  0.
↪009064514495372    28.00868491133
0.002334119807294   0.002334119807293   0.002334119807293   0.8051326751218  0.
↪002334119807295    275.8552661516
0.0006946146456239  0.0006946146456238  0.0006946146456238  0.7097377670861  0.
↪0006946146456242   332.7560201847
0.000253341221249   0.0002533412212489  0.0002533412212489  0.6718481634042  0.
↪0002533412212491   377.2933943083
1.33722226265e-06   1.337222262655e-06  1.337222262657e-06  0.9952598213341  1.
↪337222262651e-06   392.3599718847
6.775876566156e-11  6.775875940049e-11  6.775875524037e-11  0.9999493313269  6.
↪775868905564e-11   392.4568216146
Optimization terminated successfully.
         Current function value: 392.456822
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1030590385675     0.1030590385675     0.1030590385675     0.9011154818883  0.
↪1030590385675      219.2748221074
0.02654054666621    0.02654054666621    0.02654054666621    0.7752562655553  0.
↪02654054666621     276.7551091027
0.01427394633699    0.01427394633699    0.01427394633699    0.4912526709581  0.
↪01427394633699     607.4391713228
```

(continues on next page)

(continued from previous page)

```
0.001218580492651   0.001218580492601   0.001218580492601   0.9248974186534  0.
↪001218580492676   1166.289755796
1.888403761715e-05  1.888403761622e-05  1.888403761624e-05  0.985551099478   1.
↪888403761735e-05  1241.740132447
1.545745837373e-09  1.545745372552e-09  1.545745362755e-09  0.999922927438   1.
↪545745648845e-09  1242.779847889
1.692223225044e-10  1.873153075245e-10  1.873152955153e-10  0.8801597673128  1.
↪817506825368e-10  1242.779937345
Optimization terminated successfully.
         Current function value: 1242.779937
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             652.1069903706
0.06097571429344    0.06097571429346    0.06097571429347    0.9444902662603  0.
↪06097571429346    285.2043200632
0.01533833555931    0.01533833555931    0.01533833555932    0.7609599230413  0.
↪01533833555932    2.7262007615
0.006316196361705   0.006316196361706   0.006316196361708   0.6163829523723  0.
↪006316196361707   8.634999065661
0.001219580332373   0.001219580332374   0.001219580332374   0.8323480537452  0.
↪001219580332374   8.307794697355
0.0005645791470893  0.0005645791470901  0.0005645791470902  0.5500511910892  0.
↪0005645791470901  6.632034269378
0.0001849432525583  0.0001849432525588  0.0001849432525588  0.7137620753374  0.
↪0001849432525588  4.483601470858
2.517759634035e-05  2.517759634041e-05  2.517759634042e-05  0.9034269505801  2.
↪517759634041e-05  4.138476382855
3.098182398421e-08  3.098182439526e-08  3.09818244027e-08   0.9988322072353  3.
↪098182439405e-08  4.000023749419
1.548986338477e-12  1.549101875116e-12  1.549109003245e-12  0.9999499994251  1.
↪549106268784e-12  3.999950652469
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             24.59260538017
0.02944420803734    0.02944420803734    0.02944420803733    0.9778743187132  0.
↪02944420803734    27.12355968787
0.008204037532537   0.008204037532537   0.008204037532536   0.7626654784836  0.
↪008204037532537   110.4823556367
0.0004115277845134  0.0004115277844911  0.000411527784491   0.9587152757388  0.
↪0004115277845217  181.7019444318
2.842168973263e-08  2.842168968894e-08  2.84216896431e-08   0.9999314505927  2.
↪84216896475e-08   184.4443098122
1.421064808692e-12  1.421032025856e-12  1.420975506573e-12  0.999950002033   1.
↪421084481186e-12  184.444539173
Optimization terminated successfully.
         Current function value: 184.444539
         Iterations: 5
```

```
-- 2020-11-09 11:32:28 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪             414.316476678
0.1943112264485     0.1943112264485     0.1943112264485     0.8115414580695  0.
↪1943112264485      293.1180738751
0.05737071259203    0.05737071259203    0.05737071259203    0.7291321197238  0.
↪05737071259203     17.07106219013
0.01062277327644    0.01062277327644    0.01062277327644    0.8173814867435  0.
↪01062277327644     12.44885046687
0.009064514495371   0.009064514495372   0.009064514495372   0.1550649500025  0.
↪009064514495372    28.00868491133
0.002334119807294   0.002334119807293   0.002334119807293   0.8051326751218  0.
↪002334119807295    275.8552661516
0.0006946146456239  0.0006946146456238  0.0006946146456238  0.7097377670861  0.
↪0006946146456242   332.7560201847
0.000253341221249   0.0002533412212489  0.0002533412212489  0.6718481634042  0.
↪0002533412212491   377.2933943083
1.33722226265e-06   1.337222262655e-06  1.337222262657e-06  0.9952598213341  1.
↪337222262651e-06   392.3599718847
6.775876566156e-11  6.775875940049e-11  6.775875524037e-11  0.9999493313269  6.
↪775868905564e-11   392.4568216146
Optimization terminated successfully.
         Current function value: 392.456822
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪             414.316476678
0.1030590385675     0.1030590385675     0.1030590385675     0.9011154818883  0.
↪1030590385675      219.2748221074
0.02654054666621    0.02654054666621    0.02654054666621    0.7752562655553  0.
↪02654054666621     276.7551091027
0.01427394633699    0.01427394633699    0.01427394633699    0.4912526709581  0.
↪01427394633699     607.4391713228
0.001218580492651   0.001218580492601   0.001218580492601   0.9248974186534  0.
↪001218580492676    1166.289755796
1.888403761715e-05  1.888403761622e-05  1.888403761624e-05  0.985551099478   1.
↪888403761735e-05   1241.740132447
1.545745837373e-09  1.545745372552e-09  1.545745362755e-09  0.999922927438   1.
↪545745648845e-09   1242.779847889
1.692223225044e-10  1.873153075245e-10  1.873152955153e-10  0.8801597673128  1.
↪817506825368e-10   1242.779937345
Optimization terminated successfully.
         Current function value: 1242.779937
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪             652.1069903706
0.06097571429344    0.06097571429346    0.06097571429347    0.9444902662603  0.
↪06097571429346     285.2043200632
0.01533833555931    0.01533833555931    0.01533833555932    0.7609599230413  0.
↪01533833555932     2.7262007615
0.006316196361705   0.006316196361706   0.006316196361708   0.6163829523723  0.
↪006316196361707    8.634999065661
0.001219580332373   0.001219580332374   0.001219580332374   0.8323480537452  0.
↪001219580332374    8.307794697355
```

(continues on next page)

(continued from previous page)

```
0.0005645791470893  0.0005645791470901  0.0005645791470902  0.5500511910892  0.
↪0005645791470901  6.632034269378
0.0001849432525583  0.0001849432525588  0.0001849432525588  0.7137620753374  0.
↪0001849432525588  4.483601470858
2.517759634035e-05  2.517759634041e-05  2.517759634042e-05  0.9034269505801  2.
↪517759634041e-05  4.138476382855
3.098182398421e-08  3.098182439526e-08  3.09818244027e-08   0.9988322072353  3.
↪098182439405e-08  4.000023749419
1.548986338477e-12  1.549101875116e-12  1.549109003245e-12  0.9999499994251  1.
↪549106268784e-12  3.999950652469
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪            24.59260538017
0.02944420803734    0.02944420803734    0.02944420803733    0.9778743187132  0.
↪02944420803734    27.12355968787
0.008204037532537   0.008204037532537   0.008204037532536   0.7626654784836  0.
↪008204037532537   110.4823556367
0.0004115277845134  0.0004115277844911  0.000411527784491   0.9587152757388  0.
↪0004115277845217  181.7019444318
2.842168973263e-08  2.842168968894e-08  2.84216896431e-08   0.9999314505927  2.
↪84216896475e-08   184.4443098122
1.421064808692e-12  1.421032025856e-12  1.420975506573e-12  0.999950002033   1.
↪421084481186e-12  184.444539173
Optimization terminated successfully.
         Current function value: 184.444539
         Iterations: 5
```

```
-- 2020-11-09 11:32:37 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪            454.6784294315
0.1862232334687     0.1862232334687     0.1862232334687     0.8174567476576  0.
↪1862232334687     282.4632604722
0.02870747975048    0.02870747975048    0.02870747975048    0.8875534078522  0.
↪02870747975048    46.09791383177
0.009705128410004   0.009705128410004   0.009705128410004   0.6723641007507  0.
↪009705128410004   35.13402203751
0.007693915131578   0.007693915131578   0.007693915131578   0.2197837130033  0.
↪007693915131578   80.93881539206
0.001699044794839   0.001699044794837   0.001699044794837   0.8250841182677  0.
↪00169904479484    298.2653882331
0.0005650008980062  0.0005650008980057  0.0005650008980057  0.678565948553   0.
↪0005650008980065  327.975809321
0.0002196033953225  0.0002196033953223  0.0002196033953223  0.6469207178836  0.
↪0002196033953226  358.4708363196
2.579861478512e-06  2.579861478501e-06  2.579861478502e-06  0.9887803221269  2.
↪579861478497e-06  368.5514208652
1.310502062448e-10  1.310502049574e-10  1.31050204896e-10   0.9999492039366  1.
↪310502128569e-10  368.6632645094
Optimization terminated successfully.
```

(continues on next page)

(continued from previous page)

```
         Current function value: 368.663265
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0          ␣
↪            454.6784294315
0.06717002148527    0.06717002148527    0.06717002148527    0.939336263039   0.
↪06717002148527    183.2225626364
0.0335506482529     0.0335506482529     0.0335506482529     0.5230681149682  0.
↪0335506482529     256.2098343218
0.01040420231972    0.01040420231972    0.01040420231972    0.6975119886622  0.
↪01040420231972    178.3301859411
0.004349337378734   0.004349337378734   0.004349337378734   0.6235886580387  0.
↪004349337378734   396.8309868051
0.002928936786464   0.002928936786463   0.002928936786463   0.3467952576409  0.
↪002928936786463   488.976382689
0.0001458208281248  0.0001458208281499  0.0001458208281499  0.9601001354313  0.
↪0001458208281409  665.3317152907
1.593588680453e-07  1.593588684743e-07  1.593588684755e-07  0.9989479829423  1.
↪593588661973e-07  675.8691141106
7.991316370683e-12  7.990632294946e-12  7.990631661844e-12  0.999949860274   7.
↪990813072693e-12  675.8826828943
Optimization terminated successfully.
         Current function value: 675.882683
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0          ␣
↪            759.5666413636
0.06104093664241    0.06104093664241    0.0610409366424     0.9444854714867  0.
↪06104093664241    386.9519025016
0.01481511454072    0.01481511454072    0.01481511454072    0.7670183371838  0.
↪01481511454072    2.643589195991
0.005238876768342   0.005238876768342   0.005238876768341   0.6766638370173  0.
↪005238876768342   11.29316821413
0.001567745187159   0.001567745187159   0.001567745187159   0.7265977484523  0.
↪001567745187159   12.32846209581
0.0001184080214619  0.0001184080214619  0.0001184080214619  0.9328477549533  0.
↪0001184080214619  6.401954889258
1.02649122158e-05   1.026491221577e-05  1.026491221577e-05  0.959846524846   1.
↪026491221577e-05  6.140180339771
2.564197796788e-08  2.564197949727e-08  2.564197949173e-08  0.9988486248561  2.
↪56419794805e-08   6.06660177389
1.282243371032e-12  1.282166817908e-12  1.282163560611e-12  0.9999499974762  1.
↪282163552715e-12  6.066591804033
Optimization terminated successfully.
         Current function value: 6.066592
         Iterations: 8
```

```
-- 2020-11-09 11:32:46 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0          ␣
↪            454.6784294315
0.1862232334687     0.1862232334687     0.1862232334687     0.8174567476576  0.
↪1862232334687     282.4632604722
```

(continues on next page)

(continued from previous page)

```
0.02870747975048    0.02870747975048    0.02870747975048    0.8875534078522  0.
↪02870747975048    46.09791383177
0.009705128410004   0.009705128410004   0.009705128410004   0.6723641007507  0.
↪009705128410004   35.13402203751
0.007693915131578   0.007693915131578   0.007693915131578   0.2197837130033  0.
↪007693915131578   80.93881539206
0.001699044794839   0.001699044794837   0.001699044794837   0.8250841182677  0.
↪00169904479484    298.2653882331
0.0005650008980062  0.0005650008980057  0.0005650008980057  0.678565948553   0.
↪0005650008980065  327.975809321
0.0002196033953225  0.0002196033953223  0.0002196033953223  0.6469207178836  0.
↪0002196033953226  358.4708363196
2.579861478512e-06  2.579861478501e-06  2.579861478502e-06  0.9887803221269  2.
↪579861478497e-06  368.5514208652
1.310502062448e-10  1.310502049574e-10  1.31050204896e-10   0.9999492039366  1.
↪310502128569e-10  368.6632645094
Optimization terminated successfully.
         Current function value: 368.663265
         Iterations: 9
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪            454.6784294315
0.06717002148527    0.06717002148527    0.06717002148527    0.939336263039   0.
↪06717002148527    183.2225626364
0.0335506482529     0.0335506482529     0.0335506482529     0.5230681149682  0.
↪0335506482529     256.2098343218
0.01040420231972    0.01040420231972    0.01040420231972    0.6975119886622  0.
↪01040420231972    178.3301859411
0.004349337378734   0.004349337378734   0.004349337378734   0.6235886580387  0.
↪004349337378734   396.8309868051
0.002928936786464   0.002928936786463   0.002928936786463   0.3467952576409  0.
↪002928936786463   488.976382689
0.0001458208281248  0.0001458208281499  0.0001458208281499  0.9601001354313  0.
↪0001458208281409  665.3317152907
1.593588680453e-07  1.593588684743e-07  1.593588684755e-07  0.9989479829423  1.
↪593588661973e-07  675.8691141106
7.991316370683e-12  7.990632294946e-12  7.990631661844e-12  0.999949860274   7.
↪990813072693e-12  675.8826828943
Optimization terminated successfully.
         Current function value: 675.882683
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪            759.5666413636
0.06104093664241    0.06104093664241    0.0610409366424     0.9444854714867  0.
↪06104093664241    386.9519025016
0.01481511454072    0.01481511454072    0.01481511454072    0.7670183371838  0.
↪01481511454072    2.643589195991
0.005238876768342   0.005238876768342   0.005238876768341   0.6766638370173  0.
↪005238876768342   11.29316821413
0.001567745187159   0.001567745187159   0.001567745187159   0.7265977484523  0.
↪001567745187159   12.32846209581
0.0001184080214619  0.0001184080214619  0.0001184080214619  0.9328477549533  0.
↪0001184080214619  6.401954889258
1.02649122158e-05   1.026491221577e-05  1.026491221577e-05  0.959846524846   1.
↪026491221577e-05  6.140180339771
```

(continues on next page)

(continued from previous page)

```
2.564197796788e-08  2.564197949727e-08  2.564197949173e-08  0.9988486248561  2.
↪56419794805e-08    6.06660177389
1.282243371032e-12  1.282166817908e-12  1.282163560611e-12  0.9999499974762  1.
↪282163552715e-12   6.066591804033
Optimization terminated successfully.
         Current function value: 6.066592
         Iterations: 8
```

```
-- 2020-11-09 11:32:55 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                ␣
↪            506.4464461439
0.1717574295871     0.1717574295871     0.1717574295871     0.8319963473544  0.
↪1717574295871      268.7406800963
0.06638577511032    0.06638577511032    0.06638577511033    0.6490768395246  0.
↪06638577511033     376.5617800925
0.0108119991109     0.01081199911089    0.01081199911089    0.8457086649906  0.
↪01081199911089     156.44831956
0.006166341998836   0.006166341998834   0.006166341998835   0.4575243211097  0.
↪006166341998835    243.5697308399
0.0005157722017999  0.0005157722017955  0.0005157722017955  0.9303097249327  0.
↪0005157722018036   353.933860755
7.752032087913e-08  7.75203208986e-08   7.752032089586e-08  0.9998584907938  7.
↪75203208902e-08    354.2607816225
1.875610793654e-11  1.875609893145e-11  1.875610020548e-11  0.9997580492617  1.
↪875613913502e-11   354.2617798007
Optimization terminated successfully.
         Current function value: 354.261780
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                ␣
↪            506.4464461439
0.1079826576971     0.1079826576971     0.1079826576971     0.8981297430914  0.
↪1079826576971      201.9066839097
0.04065945390789    0.04065945390789    0.04065945390789    0.6593890232973  0.
↪04065945390789     434.9407529189
0.01889246737726    0.01889246737727    0.01889246737727    0.5585864512458  0.
↪01889246737727     712.8967272059
0.001989080291767   0.001989080291864   0.001989080291864   0.9101975479096  0.
↪001989080291929    1386.629312789
2.319893722758e-06  2.31989372302e-06   2.319893722997e-06  0.9988740155508  2.
↪319893723235e-06   1475.934762389
2.096073130738e-10  2.096071489352e-10  2.096071783066e-10  0.9999096479509  2.
↪096074566229e-10   1476.090721225
Optimization terminated successfully.
         Current function value: 1476.090721
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0                ␣
↪            889.6273141593
0.0609484122893     0.06094841228931    0.0609484122893     0.9446306809213  0.
↪06094841228931     507.5981904767
```

(continues on next page)

(continued from previous page)

```
0.01403712586721    0.01403712586721    0.01403712586721    0.7774983036897  0.
↪01403712586721    2.560790425808
0.004916790985586   0.004916790985586   0.004916790985586   0.6812990636392  0.
↪004916790985587   14.46599029197
0.001087493288514   0.001087493288517   0.001087493288517   0.8126332143986  0.
↪001087493288517   17.14374653765
0.0004935333934202  0.0004935333934214  0.0004935333934214  0.5558253633026  0.
↪0004935333934214  10.54120251124
0.0001121697661281  0.000112169766129   0.000112169766129   0.817643306912   0.
↪000112169766129   4.9896729167
2.712089127881e-05  2.712089127903e-05  2.712089127903e-05  0.7970501144483  2.
↪712089127903e-05  4.324997088558
1.122718419375e-07  1.122718402016e-07  1.122718401965e-07  0.99597588406    1.
↪122718401941e-07  4.000800850778
5.614947446286e-12  5.615029242581e-12  5.615032519692e-12  0.9999499875898  5.
↪615033454619e-12  3.999950693022
2.903000170299e-16  2.886166226175e-16  2.808194717321e-16  0.9999499790221  2.
↪807865903642e-16  3.999950650496
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪            889.6273141593
0.06073185177466    0.06073185177467    0.06073185177466    0.9450179496586  0.
↪06073185177466    760.7133777158
0.00617243956643    0.006172439566431   0.006172439566431   0.8994093979063  0.
↪006172439566431   1.525116036565
0.002290102597025   0.002290102597025   0.002290102597025   0.6678934240013  0.
↪002290102597025   13.23877514086
0.0007463324176618  0.0007463324176611  0.0007463324176611  0.7077017235652  0.
↪0007463324176608  10.86935158651
0.0003923306880488  0.0003923306880484  0.0003923306880484  0.488114542318   0.
↪0003923306880483  6.236673613644
2.432640260798e-05  2.432640260791e-05  2.432640260791e-05  1.0              2.
↪43264026079e-05   0.3563903987643
6.281868977076e-08  6.281868977357e-08  6.281868977105e-08  0.9991491899851  6.
↪281868977102e-08  0.0001119022044274
3.378349078595e-12  3.378347740195e-12  3.378348155213e-12  0.9999462206524  3.
↪378348155213e-12  6.056769044982e-09
2.306615405556e-13  2.306653965292e-13  2.306614266427e-13  0.9320769632577  2.
↪306614266427e-13  4.595744934239e-10
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 9
```

```
-- 2020-11-09 11:33:05 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪            506.4464461439
0.1717574295871     0.1717574295871     0.1717574295871     0.8319963473544  0.
↪1717574295871     268.7406800963
0.06638577511032    0.06638577511032    0.06638577511033    0.6490768395246  0.
↪06638577511033    376.5617800925
```

(continues on next page)

(continued from previous page)

```
0.0108119991109     0.01081199911089    0.01081199911089    0.8457086649906  0.
↪01081199911089     156.44831956
0.006166341998836   0.006166341998834   0.006166341998835   0.4575243211097  0.
↪006166341998835    243.5697308399
0.0005157722017999  0.0005157722017955  0.0005157722017955  0.9303097249327  0.
↪0005157722018036   353.933860755
7.752032087913e-08  7.75203208986e-08   7.752032089586e-08  0.9998584907938  7.
↪75203208902e-08    354.2607816225
1.875610793654e-11  1.875609893145e-11  1.875610020548e-11  0.9997580492617  1.
↪875613913502e-11   354.2617798007
Optimization terminated successfully.
         Current function value: 354.261780
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           506.4464461439
0.1079826576971     0.1079826576971     0.1079826576971     0.8981297430914  0.
↪1079826576971      201.9066839097
0.04065945390789    0.04065945390789    0.04065945390789    0.6593890232973  0.
↪04065945390789     434.9407529189
0.01889246737726    0.01889246737727    0.01889246737727    0.5585864512458  0.
↪01889246737727     712.8967272059
0.001989080291767   0.001989080291864   0.001989080291864   0.9101975479096  0.
↪001989080291929    1386.629312789
2.319893722758e-06  2.31989372302e-06   2.319893722997e-06  0.9988740155508  2.
↪319893723235e-06   1475.934762389
2.096073130738e-10  2.096071489352e-10  2.096071783066e-10  0.9999096479509  2.
↪096074566229e-10   1476.090721225
Optimization terminated successfully.
         Current function value: 1476.090721
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           889.6273141593
0.0609484122893     0.06094841228931    0.0609484122893     0.9446306809213  0.
↪06094841228931     507.5981904767
0.01403712586721    0.01403712586721    0.01403712586721    0.7774983036897  0.
↪01403712586721     2.560790425808
0.004916790985586   0.004916790985586   0.004916790985586   0.6812990636392  0.
↪004916790985587    14.46599029197
0.001087493288514   0.001087493288517   0.001087493288517   0.8126332143986  0.
↪001087493288517    17.14374653765
0.0004935333934202  0.0004935333934214  0.0004935333934214  0.5558253633026  0.
↪0004935333934214   10.54120251124
0.0001121697661281  0.000112169766129   0.000112169766129   0.817643306912   0.
↪000112169766129    4.9896729167
2.712089127881e-05  2.712089127903e-05  2.712089127903e-05  0.7970501144483  2.
↪712089127903e-05   4.324997088558
1.122718419375e-07  1.122718402016e-07  1.122718401965e-07  0.99597588406    1.
↪122718401941e-07   4.000800850778
5.614947446286e-12  5.615029242581e-12  5.615032519692e-12  0.9999499875898  5.
↪615033454619e-12   3.999950693022
2.903000170299e-16  2.886166226175e-16  2.808194717321e-16  0.9999499790221  2.
↪807865903642e-16   3.999950650496
Optimization terminated successfully.
```

(continues on next page)

(continued from previous page)

```
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪             889.6273141593
0.06073185177466    0.06073185177467    0.06073185177466    0.9450179496586  0.
↪06073185177466    760.7133777158
0.00617243956643    0.006172439566431   0.006172439566431   0.8994093979063  0.
↪006172439566431   1.525116036565
0.002290102597025   0.002290102597025   0.002290102597025   0.6678934240013  0.
↪002290102597025   13.23877514086
0.0007463324176618  0.0007463324176611  0.0007463324176611  0.7077017235652  0.
↪0007463324176608  10.86935158651
0.0003923306880488  0.0003923306880484  0.0003923306880484  0.488114542318   0.
↪0003923306880483  6.236673613644
2.432640260798e-05  2.432640260791e-05  2.432640260791e-05  1.0              2.
↪43264026079e-05   0.3563903987643
6.281868977076e-08  6.281868977357e-08  6.281868977105e-08  0.9991491899851  6.
↪281868977102e-08  0.0001119022044274
3.378349078595e-12  3.378347740195e-12  3.378348155213e-12  0.9999462206524  3.
↪378348155213e-12  6.056769044982e-09
2.306615405556e-13  2.306653965292e-13  2.306614266427e-13  0.9320769632577  2.
↪306614266427e-13  4.595744934239e-10
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 9
```

```
-- 2020-11-09 11:33:13 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪             568.8949309456
0.1574999940443     0.1574999940443     0.1574999940443     0.8462995490748  0.
↪1574999940443     259.3692323549
0.07523425397025    0.07523425397025    0.07523425397025    0.5547720211262  0.
↪07523425397025    453.5904211877
0.01015218722862    0.01015218722862    0.01015218722862    0.8831324835902  0.
↪01015218722862    172.5109988087
0.005597873343747   0.005597873343747   0.005597873343747   0.4782155730769  0.
↪005597873343747   257.1812475882
0.0004682435100659  0.0004682435100634  0.0004682435100634  0.9301215087302  0.
↪000468243510069   357.0084554784
6.611502592752e-08  6.611502591615e-08  6.611502591873e-08  0.999863385888   6.
↪611502594636e-08  355.7877874388
7.555502313764e-12  7.555508723405e-12  7.555505999397e-12  0.9998857217433  7.
↪555493074812e-12  355.7884668282
Optimization terminated successfully.
         Current function value: 355.788467
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪             568.8949309456
0.07655552550188    0.07655552550188    0.07655552550188    0.9326533055761  0.
↪07655552550188    158.9571104913
```

(continues on next page)

(continued from previous page)

```
0.02915610316818    0.02915610316819    0.02915610316819    0.6583861263339  0.
↪02915610316818    366.3114833866
0.01377238842158    0.01377238842158    0.01377238842158    0.5494198953479  0.
↪01377238842158    509.1861831748
0.002113311162461   0.002113311162461   0.002113311162461   0.8672288939288  0.
↪002113311162461   954.4881944043
1.802801647426e-06  1.802801647524e-06  1.802801647519e-06  0.9992160883437  1.
↪802801646136e-06  1007.960176974
1.128093610417e-10  1.128094830078e-10  1.128094936008e-10  0.9999374254592  1.
↪128097428918e-10  1008.067349764
Optimization terminated successfully.
         Current function value: 1008.067350
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           1027.252465794
0.06092848901518    0.06092848901523    0.06092848901522    0.9446907062507  0.
↪06092848901524    639.5911028223
0.01301865096831    0.01301865096832    0.01301865096832    0.7926600108064  0.
↪01301865096833    2.590461755952
0.00456587722928    0.004565877229283   0.004565877229283   0.6815721939632  0.
↪004565877229284   18.42445056578
0.001192620177961   0.001192620177954   0.001192620177954   0.7720888734274  0.
↪001192620177954   21.2766378272
0.0004150762782539  0.0004150762782515  0.0004150762782515  0.6551231023534  0.
↪0004150762782516  10.90714854956
8.30397207983e-05   8.303972079784e-05  8.303972079783e-05  0.8551088375548  8.
↪303972079786e-05  4.883155784925
2.345269716473e-05  2.345269716461e-05  2.34526971646e-05   0.7548298132573  2.
↪345269716461e-05  4.346885094868
1.371159312911e-07  1.371159321168e-07  1.371159321194e-07  0.994331926271   1.
↪371159321123e-07  4.001183377585
6.859845575537e-12  6.859856824223e-12  6.859863126594e-12  0.9999499723157  6.
↪85986409236e-12   3.999950686436
6.215051942361e-16  3.43521130371e-16   3.430265101783e-16  0.9999499985889  3.
↪431205195905e-16  3.999950624744
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           513.6262328969
0.06112045297154    0.06112045297154    0.06112045297151    0.9426671718798  0.
↪06112045297154    286.961037666
0.0186024953335     0.0186024953335     0.01860249533349    0.7059824756003  0.
↪0186024953335     1.911789377935
0.006446621823764   0.006446621823763   0.00644662182376    0.6910883581126  0.
↪006446621823763   10.97980543166
0.0005501537566849  0.0005501537566955  0.0005501537566952  0.9377765193638  0.
↪0005501537566949  10.00501917633
4.007383875704e-06  4.007383875795e-06  4.007383875804e-06  0.9951127586026  4.
↪007383875797e-06  6.687770636962
2.007342726257e-10  2.007343535262e-10  2.007343506552e-10  0.9999499088827  2.
↪007343503785e-10  6.666585422706
2.161448037787e-10  2.007343535257e-10  2.007343507631e-10  2.456113364069e-461.
↪961302857585e-10  6.666585422564
```

(continues on next page)

(continued from previous page)

```
6.009588090216e-12  2.723883580422e-12  2.723885134159e-12  0.9866729468426  2.
↪685910112544e-12  6.666584374146
Optimization terminated successfully.
         Current function value: 6.666584
         Iterations: 8
```

```
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'cholesky':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'cholesky'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'sym_pos':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'sym_pos'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/anaconda3/lib/python3.8/site-packages/scipy/optimize/_linprog_ip.py:
↪116: LinAlgWarning: Ill-conditioned matrix (rcond=1.62544e-36): result may not be␣
↪accurate.
  return sp.linalg.solve(M, r, sym_pos=sym_pos)
-- 2020-11-09 11:33:25 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪             568.8949309456
0.1574999940443     0.1574999940443     0.1574999940443     0.8462995490748  0.
↪1574999940443      259.3692323549
0.07523425397025    0.07523425397025    0.07523425397025    0.5547720211262  0.
↪07523425397025     453.5904211877
0.01015218722862    0.01015218722862    0.01015218722862    0.8831324835902  0.
↪01015218722862     172.5109988087
0.005597873343747   0.005597873343747   0.005597873343747   0.4782155730769  0.
↪005597873343747    257.1812475882
0.0004682435100659  0.0004682435100634  0.0004682435100634  0.9301215087302  0.
↪000468243510069    357.0084554784
6.611502592752e-08  6.611502591615e-08  6.611502591873e-08  0.999863385888   6.
↪611502594636e-08   355.7877874388
7.555502313764e-12  7.555508723405e-12  7.555505999397e-12  0.9998857217433  7.
↪555493074812e-12   355.7884668282
Optimization terminated successfully.
         Current function value: 355.788467
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0     ␣
↪             568.8949309456
0.07655552550188    0.07655552550188    0.07655552550188    0.9326533055761  0.
↪07655552550188     158.9571104913
0.02915610316818    0.02915610316819    0.02915610316819    0.6583861263339  0.
↪02915610316818     366.3114833866
0.01377238842158    0.01377238842158    0.01377238842158    0.5494198953479  0.
↪01377238842158     509.1861831748
0.002113311162461   0.002113311162461   0.002113311162461   0.8672288939288  0.
↪002113311162461    954.4881944043
```

(continues on next page)

(continued from previous page)

```
1.802801647426e-06  1.802801647524e-06  1.802801647519e-06  0.9992160883437  1.
↪802801646136e-06  1007.960176974
1.128093610417e-10  1.128094830078e-10  1.128094936008e-10  0.9999374254592  1.
↪128097428918e-10  1008.067349764
Optimization terminated successfully.
         Current function value: 1008.067350
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪            1027.252465794
0.06092848901518    0.06092848901523    0.06092848901522    0.9446907062507  0.
↪06092848901524    639.5911028223
0.01301865096831    0.01301865096832    0.01301865096832    0.7926600108064  0.
↪01301865096833    2.590461755952
0.00456587722928    0.004565877229283   0.004565877229283   0.6815721939632  0.
↪004565877229284   18.42445056578
0.001192620177961   0.001192620177954   0.001192620177954   0.7720888734274  0.
↪001192620177954   21.2766378272
0.0004150762782539  0.0004150762782515  0.0004150762782515  0.6551231023534  0.
↪0004150762782516  10.90714854956
8.30397207983e-05   8.303972079784e-05  8.303972079783e-05  0.8551088375548  8.
↪303972079786e-05  4.883155784925
2.345269716473e-05  2.345269716461e-05  2.34526971646e-05   0.7548298132573  2.
↪345269716461e-05  4.346885094868
1.371159312911e-07  1.371159321168e-07  1.371159321194e-07  0.994331926271   1.
↪371159321123e-07  4.001183377585
6.859845575537e-12  6.859856824223e-12  6.859863126594e-12  0.9999499723157  6.
↪85986409236e-12   3.999950686436
6.215051942361e-16  3.43521130371e-16   3.430265101783e-16  0.9999499985889  3.
↪431205195905e-16  3.999950624744
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪            513.6262328969
0.06112045297154    0.06112045297154    0.06112045297151    0.9426671718798  0.
↪06112045297154    286.961037666
0.0186024953335     0.0186024953335     0.01860249533349    0.7059824756003  0.
↪0186024953335     1.911789377935
0.006446621823764   0.006446621823763   0.00644662182376    0.6910883581126  0.
↪006446621823763   10.97980543166
0.0005501537566849  0.0005501537566955  0.0005501537566952  0.9377765193638  0.
↪0005501537566949  10.00501917633
4.007383875704e-06  4.007383875795e-06  4.007383875804e-06  0.9951127586026  4.
↪007383875797e-06  6.687770636962
2.007342726257e-10  2.007343535262e-10  2.007343506552e-10  0.9999499088827  2.
↪007343503785e-10  6.666585422706
2.161448037787e-10  2.007343535257e-10  2.007343507631e-10  2.456113364069e-461.
↪961302857585e-10  6.666585422564
6.009588090216e-12  2.723883580422e-12  2.723885134159e-12  0.9866729468426  2.
↪685910112544e-12  6.666584374146
Optimization terminated successfully.
         Current function value: 6.666584
         Iterations: 8
```

```
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'cholesky':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'cholesky'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/Documents/SGI/1-examples/example_model/model/Results/muse/src/muse/
↪investments.py:325: OptimizeWarning: Solving system with option 'sym_pos':True␣
↪failed. It is normal for this to happen occasionally, especially as the solution is␣
↪approached. However, if you see this frequently, consider setting option 'sym_pos'␣
↪to False.
  res = linprog(**adapter.kwargs, options=dict(disp=True))
/Users/alexkell/anaconda3/lib/python3.8/site-packages/scipy/optimize/_linprog_ip.py:
↪116: LinAlgWarning: Ill-conditioned matrix (rcond=1.62544e-36): result may not be␣
↪accurate.
  return sp.linalg.solve(M, r, sym_pos=sym_pos)
-- 2020-11-09 11:33:37 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0    ␣
↪             647.4402248427
0.1042702416317     0.1042702416317     0.1042702416317     0.9044651903652  0.
↪1042702416317      197.7436068274
0.04330783922975    0.04330783922975    0.04330783922975    0.6213338928986  0.
↪04330783922975     554.8458695929
0.02029339907042    0.02029339907042    0.02029339907042    0.5546164251713  0.
↪02029339907042     887.503418141
0.002059630009005   0.002059630008646   0.002059630008646   0.9123945707354  0.
↪002059630008855    1773.497308596
1.96164830749e-06   1.961648306624e-06  1.961648306623e-06  0.9995475315428  1.
↪96164830693e-06    1899.795955905
1.183904657193e-10  1.183907029537e-10  1.183907180056e-10  0.9999396473343  1.
↪183903667831e-10   1900.055708261
Optimization terminated successfully.
         Current function value: 1900.055708
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0    ␣
↪             1199.542590081
0.06080970203684    0.06080970203684    0.06080970203685    0.9448463893022  0.
↪06080970203684     802.3521760538
0.01206790508619    0.01206790508619    0.01206790508619    0.8067148881887  0.
↪01206790508619     2.242531350203
0.004235304009507   0.004235304009507   0.004235304009507   0.6829892140296  0.
↪004235304009507    22.93309435174
0.001076107298274   0.001076107298272   0.001076107298272   0.7811880414723  0.
↪001076107298272    27.34472817154
0.0001381467026759  0.0001381467026756  0.0001381467026756  0.9534954297498  0.
↪0001381467026756   2.164474733514
0.0001283068196922  0.0001283068196919  0.0001283068196919  0.07280278667804 0.
↪0001283068196919   2.01175576967
1.709535377769e-06  1.709535377765e-06  1.709535377765e-06  0.9870423816225  1.
↪709535377765e-06   0.04804313560572
9.48541628326e-11   9.485418648369e-11  9.485418526931e-11  0.9999445934679  9.
↪485418526932e-11   2.66445492392e-06
```

(continues on next page)

(continued from previous page)

```
8.795947527601e-12  8.795912717956e-12  8.795907275833e-12  0.9086823994024  8.
↪795907275834e-12  2.473302892196e-07
8.361396439476e-12  8.361363544519e-12  8.361358288329e-12  0.05441434596209 8.
↪361358288329e-12  2.360712463483e-07
1.014674559722e-12  1.014680661495e-12  1.014676648034e-12  0.8808930741304  1.
↪014676648034e-12  2.943765656853e-08
7.649517191012e-13  7.649501663219e-13  7.649518441425e-13  0.266685424728   7.
↪649518441426e-13  2.292855842523e-08
2.251056337644e-13  2.251034423868e-13  2.250976823208e-13  0.7199831364733  2.
↪250976823208e-13  9.225506253453e-09
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪            599.7712950406
0.06108522397884    0.06108522397884    0.06108522397884    0.9426921917133  0.
↪06108522397884    370.5338700223
0.01718424163858    0.01718424163858    0.01718424163858    0.7269400839096  0.
↪01718424163858    1.816590685562
0.006448030539302   0.006448030539302   0.006448030539302   0.6618055567806  0.
↪006448030539302   13.33233954634
0.0007451641812888  0.0007451641813056  0.0007451641813056  0.9128367304104  0.
↪0007451641813056  13.46011423965
0.0002051849835872  0.0002051849835913  0.0002051849835913  0.726248232922   0.
↪0002051849835914  8.092503300126
3.421589486985e-06  3.421589487287e-06  3.421589487279e-06  1.0              3.
↪421589487245e-06  5.943322696325
2.883371020626e-10  2.88337212738e-10   2.883372076584e-10  0.999916005146   2.
↪883372071457e-10  5.933260704906
1.437342489421e-14  1.441626748189e-14  1.441700316141e-14  0.999949999407   1.
↪441794244359e-14  5.933260075848
Optimization terminated successfully.
         Current function value: 5.933260
         Iterations: 8
```

```
-- 2020-11-09 11:33:46 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪            647.4402248427
0.1042702416317     0.1042702416317     0.1042702416317     0.9044651903652  0.
↪1042702416317     197.7436068274
0.04330783922975    0.04330783922975    0.04330783922975    0.6213338928986  0.
↪04330783922975    554.8458695929
0.02029339907042    0.02029339907042    0.02029339907042    0.5546164251713  0.
↪02029339907042    887.503418141
0.002059630009005   0.002059630008646   0.002059630008646   0.9123945707354  0.
↪002059630008855   1773.497308596
1.96164830749e-06   1.961648306624e-06  1.961648306623e-06  0.9995475315428  1.
↪96164830693e-06   1899.795955905
1.183904657193e-10  1.183907029537e-10  1.183907180056e-10  0.9999396473343  1.
↪183903667831e-10  1900.055708261
Optimization terminated successfully.
```

(continues on next page)

(continued from previous page)

```
         Current function value: 1900.055708
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             1199.542590081
0.06080970203684    0.06080970203684    0.06080970203685    0.9448463893022  0.
↪06080970203684    802.3521760538
0.01206790508619    0.01206790508619    0.01206790508619    0.8067148881887  0.
↪01206790508619    2.242531350203
0.004235304009507   0.004235304009507   0.004235304009507   0.6829892140296  0.
↪004235304009507   22.93309435174
0.001076107298274   0.001076107298272   0.001076107298272   0.7811880414723  0.
↪001076107298272   27.34472817154
0.0001381467026759  0.0001381467026756  0.0001381467026756  0.9534954297498  0.
↪0001381467026756  2.164474733514
0.0001283068196922  0.0001283068196919  0.0001283068196919  0.07280278667804 0.
↪0001283068196919  2.01175576967
1.709535377769e-06  1.709535377765e-06  1.709535377765e-06  0.9870423816225  1.
↪709535377765e-06  0.04804313560572
9.48541628326e-11   9.485418648369e-11  9.485418526931e-11  0.9999445934679  9.
↪485418526932e-11  2.66445492392e-06
8.795947527601e-12  8.795912717956e-12  8.795907275833e-12  0.9086823994024  8.
↪795907275834e-12  2.473302892196e-07
8.361396439476e-12  8.361363544519e-12  8.361358288329e-12  0.05441434596209 8.
↪361358288329e-12  2.360712463483e-07
1.014674559722e-12  1.014680661495e-12  1.014676648034e-12  0.8808930741304  1.
↪014676648034e-12  2.943765656853e-08
7.649517191012e-13  7.649501663219e-13  7.649518441425e-13  0.266685424728   7.
↪649518441426e-13  2.292855842523e-08
2.251056337644e-13  2.251034423868e-13  2.250976823208e-13  0.7199831364733  2.
↪250976823208e-13  9.225506253453e-09
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             599.7712950406
0.06108522397884    0.06108522397884    0.06108522397884    0.9426921917133  0.
↪06108522397884    370.5338700223
0.01718424163858    0.01718424163858    0.01718424163858    0.7269400839096  0.
↪01718424163858    1.816590685562
0.006448030539302   0.006448030539302   0.006448030539302   0.6618055567806  0.
↪006448030539302   13.33233954634
0.0007451641812888  0.0007451641813056  0.0007451641813056  0.9128367304104  0.
↪0007451641813056  13.46011423965
0.0002051849835872  0.0002051849835913  0.0002051849835913  0.726248232922   0.
↪0002051849835914  8.092503300126
3.421589486985e-06  3.421589487287e-06  3.421589487279e-06  1.0              3.
↪421589487245e-06  5.943322696325
2.883371020626e-10  2.88337212738e-10   2.883372076584e-10  0.999916005146   2.
↪883372071457e-10  5.933260704906
1.437342489421e-14  1.441626748189e-14  1.441700316141e-14  0.999949999407   1.
↪441794244359e-14  5.933260075848
Optimization terminated successfully.
         Current function value: 5.933260
```

(continues on next page)

(continued from previous page)

```
        Iterations: 8
```

```
-- 2020-11-09 11:33:58 - muse.mca - WARNING
Check growth constraints for wind.
```

And we can check the parameters were used accordingly:

[9]:
```
all_txt_files = sorted((Path() / "Results").glob("Residential*.txt"))
assert len(all_txt_files) == 7
assert "Hello, you!" in all_txt_files[0].read_text()
all_txt_files
```

[9]:
```
[PosixPath('Results/ResidentialConsumption_Zero2020.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2025.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2030.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2035.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2040.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2045.txt'),
 PosixPath('Results/ResidentialConsumption_Zero2050.txt')]
```

Again, we can see that the number of output files generated were as we expected and that our new message "Hello, you!" was found within these files. This means that our output and sink functions worked as expected.

7.1.3 Next steps

In the next section we will output a technology filter, to stop agents from investing in a certain technology, and a new metric to combine multiple objectives.

7.2 Further extending MUSE

[1]:
```
from xarray import Dataset, DataArray

from muse.agents import Agent
from muse.filters import register_filter

@register_filter
def no_ccgt_filter(
    agent: Agent,
    search_space: DataArray,
    technologies: Dataset,
    market: Dataset
) -> DataArray:
    """Excludes gasCCGT."""
    dropped_tech = search_space.where(search_space.replacement != "windturbine",␣
↪drop=True)
    return search_space & search_space.replacement.isin(dropped_tech.replacement)
```

[6]:
```
import logging
from muse.mca import MCA
from muse import examples

# model_path = examples.copy_model(overwrite=True)
```

(continues on next page)

(continued from previous page)

```
logging.getLogger("muse").setLevel(0)
mca = MCA.factory("model/settings.toml")
mca.run();
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             148.9679256735
0.2598249156018     0.2598249156018     0.2598249156018     0.7495200004432  0.
↪2598249156018      120.5733849622
0.02399956829695    0.02399956829695    0.02399956829695    0.9210391224498  0.
↪02399956829695     4.780663765494
0.0181364461758     0.0181364461758     0.0181364461758     0.2509588065043  0.
↪0181364461758      7.107141691547
0.01499350833129    0.01499350833129    0.01499350833129    0.1921973185437  0.
↪01499350833129     70.77614035582
0.004968295711366   0.004968295711367   0.004968295711366   0.6857131120066  0.
↪004968295711366    164.7472224003
0.0006443120819652  0.0006443120819642  0.000644312081964   0.8804718592549  0.
↪0006443120819672   289.7109372802
2.427431365313e-06  2.427431365276e-06  2.42743136527e-06   0.9976309182175  2.
↪427431365399e-06   310.7437190082
1.214286379284e-10  1.214286245985e-10  1.214286217581e-10  0.9999499778566  1.
↪214286284187e-10   310.7859704917
Optimization terminated successfully.
         Current function value: 310.785970
         Iterations: 8
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             148.9679256735
0.1806451257467     0.1806451257467     0.1806451257467     0.8281847212959  0.
↪1806451257467      169.6037982942
0.02684129624378    0.02684129624378    0.02684129624378    0.8635116331789  0.
↪02684129624378     123.2904723181
0.01081107082374    0.01081107082373    0.01081107082373    0.6279145428497  0.
↪01081107082373     293.3552576002
0.00150335333755    0.001503353337531   0.001503353337531   0.8785435425234  0.
↪001503353337582    618.5754737903
4.126548293386e-06  4.126548293452e-06  4.126548293469e-06  0.99729939758    4.
↪126548293473e-06   673.2429034259
2.063813940498e-10  2.063814559148e-10  2.063814538311e-10  0.9999499869317  2.
↪06381853248e-10    673.369600975
Optimization terminated successfully.
         Current function value: 673.369601
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             225.7506433631
0.07675788061753    0.07675788061753    0.07675788061751    0.9271368109475  0.
↪07675788061753     1.307751786207
0.01889464099889    0.01889464099889    0.01889464099888    0.7970949255449  0.
↪01889464099889     19.4989221165
0.007543783963398   0.007543783963394   0.007543783963392   0.6153162486386  0.
↪007543783963394    16.52048983639
```

(continues on next page)

(continued from previous page)

```
0.002504946781023   0.002504946781021   0.00250494678102    0.7004074675406  0.
↪002504946781021   72.319459457
0.0004445444355394  0.000444544435539   0.0004445444355388  0.8689995047748  0.
↪000444544435539   423.758860583
1.214501095331e-05  1.214501095325e-05  1.214501095324e-05  0.9820606335229  1.
↪214501095322e-05  675.1646942955
1.070439776445e-09  1.07043978707e-09   1.070439766673e-09  0.9999120772687  1.
↪070439801419e-09  681.9179209593
5.353929135063e-14  5.353592310399e-14  5.35373239097e-14   0.9999499866071  5.
↪352230256347e-14  681.9185224492
Optimization terminated successfully.
         Current function value: 681.918522
         Iterations: 8
```

```
-- 2020-11-09 15:08:48 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            359.2443189825
0.2131118149695     0.2131118149695     0.2131118149695     0.7960343319409  0.
↪2131118149695     262.2885392799
0.05758094728718    0.05758094728718    0.05758094728718    0.7523513573813  0.
↪05758094728718    13.16175379893
0.01048349585899    0.01048349585899    0.01048349585899    0.8203940076989  0.
↪01048349585899    9.036399448741
0.009005598049604   0.009005598049604   0.009005598049604   0.1488155054953  0.
↪009005598049604   19.40296370843
0.002317604003518   0.002317604003518   0.002317604003518   0.8098673571979  0.
↪002317604003518   226.5492459765
0.0009784310820962  0.0009784310820963  0.0009784310820963  0.5991550275447  0.
↪0009784310820971  289.4684048776
0.0001326875071986  0.0001326875071986  0.0001326875071986  0.8836343167337  0.
↪0001326875071987  358.6919881748
6.688262823007e-08  6.688262823276e-08  6.688262823391e-08  0.9995454959391  6.
↪688262822145e-08  365.8223849509
3.344208692489e-12  3.34420597683e-12   3.344204125008e-12  0.9999499989235  3.
↪344177862259e-12  365.8250744356
Optimization terminated successfully.
         Current function value: 365.825074
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            179.6221594912
0.06792119055695    0.06792119055695    0.06792119055695    0.9362017234058  0.
↪06792119055695    76.26287556551
0.009746786382626   0.009746786382626   0.009746786382626   0.8719786940257  0.
↪009746786382625   34.28929487798
0.00504956460969    0.005049564609689   0.005049564609689   0.5134005827998  0.
↪005049564609689   115.6145513358
0.003132107070668   0.003132107070663   0.003132107070663   0.3922116719677  0.
↪003132107070663   175.0444435627
0.0005331490663204  0.000533149066321   0.000533149066321   0.8631478238108  0.
↪0005331490663195  430.7265923665
```

(continues on next page)

(continued from previous page)

```
6.809758501168e-06  6.809758501084e-06  6.809758501069e-06  0.9896962642248  6.
↪809758501216e-06  482.9615875673
3.566453823927e-10  3.56645399101e-10   3.566453982014e-10  0.9999476448412  3.
↪566453742836e-10  483.66367672
Optimization terminated successfully.
         Current function value: 483.663677
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             547.5170704708
0.06099397574481    0.06099397574481    0.06099397574482    0.944387525785   0.
↪06099397574482    192.4316112656
0.01386491315737    0.01386491315737    0.01386491315737    0.7896145976147  0.
↪01386491315737    3.157754203839
0.00713143943229    0.00713143943229    0.007131439432292   0.5116778496497  0.
↪007131439432291   6.913227303095
0.0007854492963609  0.0007854492963609  0.0007854492963611  0.9057153968575  0.
↪000785449296361   6.591304425235
0.0003968642772996  0.0003968642772997  0.0003968642772998  0.5195625901748  0.
↪0003968642772996  5.218772647071
5.276614006577e-06  5.276614006577e-06  5.276614006576e-06  0.9941140946865  5.
↪276614006565e-06  5.272614338781
3.679893198137e-10  3.679893242575e-10  3.679893250277e-10  0.9999303117912  3.
↪679893357516e-10  5.266601980833
1.790517523351e-14  1.842347640038e-14  1.841384711876e-14  0.9999499611495  1.
↪844295319492e-14  5.266601636509
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             273.0918767992
0.007784675344484   0.007784675344483   0.007784675344496   0.9922157975144  0.
↪007784675344482   1.218389503684
0.0007796035919664  0.0007796035919663  0.0007796035919676  0.9305974535198  0.
↪0007796035919662  8.689368495091
0.0002432663284089  0.0002432663284089  0.0002432663284093  0.7114098156148  0.
↪0002432663284089  45.24959963709
7.450312562646e-05  7.450312562646e-05  7.450312562658e-05  0.7438433474555  7.
↪450312562644e-05  232.9668001869
1.824586184364e-06  1.824586184364e-06  1.824586184368e-06  0.9786324171177  1.
↪824586184364e-06  272.0879301696
1.362513233892e-10  1.362513236089e-10  1.36251323629e-10   0.9999263111913  1.
↪362513232934e-10  273.0918022961
6.81275233453e-15   6.813117868374e-15  6.81302616198e-15   0.9999499990795  6.
↪812566211363e-15  273.0918768406
Optimization terminated successfully.
         Current function value: 273.091877
         Iterations: 7
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             24.59260538017
```

(continues on next page)

(continued from previous page)

```
0.01678939478565    0.01678939478565    0.01678939478565    0.9931439495955   0.
↪01678939478565    0.6055223166906
0.009965674493075   0.009965674493075   0.009965674493078   0.4242810241262   0.
↪009965674493075   3.244714326293
0.003397925619967   0.003397925619967   0.003397925619968   0.7237970170791   0.
↪003397925619967   63.79567916947
2.830731029084e-05  2.830731029084e-05  2.830731029086e-05  0.9917790420884   2.
↪830731029084e-05  77.95525849671
1.416553419389e-09  1.416553580528e-09  1.416553598767e-09  0.999949961019    1.
↪416553321704e-09  78.09518029969
7.075972310081e-14  7.098971338123e-14  7.097069021718e-14  0.9999498833268   7.
↪082766760582e-14  78.095187309
Optimization terminated successfully.
         Current function value: 78.095187
         Iterations: 6
```

```
-- 2020-11-09 15:08:54 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0          ␣
↪             359.2443189825
0.2131118149695     0.2131118149695     0.2131118149695     0.7960343319409   0.
↪2131118149695     262.2885392799
0.05758094728718    0.05758094728718    0.05758094728718    0.7523513573813   0.
↪05758094728718    13.16175379893
0.01048349585899    0.01048349585899    0.01048349585899    0.8203940076989   0.
↪01048349585899    9.036399448741
0.009005598049604   0.009005598049604   0.009005598049604   0.1488155054953   0.
↪009005598049604   19.40296370843
0.002317604003518   0.002317604003518   0.002317604003518   0.8098673571979   0.
↪002317604003518   226.5492459765
0.0009784310820962  0.0009784310820963  0.0009784310820963  0.5991550275447   0.
↪0009784310820971  289.4684048776
0.0001326875071986  0.0001326875071986  0.0001326875071986  0.8836343167337   0.
↪0001326875071987  358.6919881748
6.688262823007e-08  6.688262823276e-08  6.688262823391e-08  0.9995454959391   6.
↪688262822145e-08  365.8223849509
3.344208692489e-12  3.34420597683e-12   3.344204125008e-12  0.9999499989235   3.
↪344177862259e-12  365.8250744356
Optimization terminated successfully.
         Current function value: 365.825074
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0          ␣
↪             179.6221594912
0.06792119055695    0.06792119055695    0.06792119055695    0.9362017234058   0.
↪06792119055695    76.26287556551
0.009746786382626   0.009746786382626   0.009746786382626   0.8719786940257   0.
↪009746786382625   34.28929487798
0.00504956460969    0.005049564609689   0.005049564609689   0.5134005827998   0.
↪005049564609689   115.6145513358
0.003132107070668   0.003132107070663   0.003132107070663   0.3922116719677   0.
↪003132107070663   175.0444435627
```

(continues on next page)

(continued from previous page)

```
0.0005331490663204  0.000533149066321   0.000533149066321   0.8631478238108  0.
↪0005331490663195  430.7265923665
6.809758501168e-06  6.809758501084e-06  6.809758501069e-06  0.9896962642248  6.
↪809758501216e-06  482.9615875673
3.566453823927e-10  3.56645399101e-10   3.566453982014e-10  0.9999476448412  3.
↪566453742836e-10  483.66367672
Optimization terminated successfully.
         Current function value: 483.663677
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             547.5170704708
0.06099397574481    0.06099397574481    0.06099397574482    0.944387525785   0.
↪06099397574482    192.4316112656
0.01386491315737    0.01386491315737    0.01386491315737    0.7896145976147  0.
↪01386491315737    3.157754203839
0.00713143943229    0.00713143943229    0.007131439432292   0.5116778496497  0.
↪007131439432291   6.913227303095
0.0007854492963609  0.0007854492963609  0.0007854492963611  0.9057153968575  0.
↪000785449296361   6.591304425235
0.0003968642772996  0.0003968642772997  0.0003968642772998  0.5195625901748  0.
↪0003968642772996  5.218772647071
5.276614006577e-06  5.276614006577e-06  5.276614006576e-06  0.9941140946865  5.
↪276614006565e-06  5.272614338781
3.679893198137e-10  3.679893242575e-10  3.679893250277e-10  0.9999303117912  3.
↪679893357516e-10  5.266601980833
1.790517523351e-14  1.842347640038e-14  1.841384711876e-14  0.9999499611495  1.
↪844295319492e-14  5.266601636509
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0         ␣
↪             273.0918767992
0.007784675344484   0.007784675344483   0.007784675344496   0.9922157975144  0.
↪007784675344482   1.218389503684
0.0007796035919664  0.0007796035919663  0.0007796035919676  0.9305974535198  0.
↪0007796035919662  8.689368495091
0.0002432663284089  0.0002432663284089  0.0002432663284093  0.7114098156148  0.
↪0002432663284089  45.24959963709
7.450312562646e-05  7.450312562646e-05  7.450312562658e-05  0.7438433474555  7.
↪450312562644e-05  232.9668001869
1.824586184364e-06  1.824586184364e-06  1.824586184368e-06  0.9786324171177  1.
↪824586184364e-06  272.0879301696
1.362513233892e-10  1.362513236089e-10  1.36251323629e-10   0.9999263111913  1.
↪362513232934e-10  273.0918022961
6.81275233453e-15   6.813117868374e-15  6.81302616198e-15   0.9999499990795  6.
↪812566211363e-15  273.0918768406
Optimization terminated successfully.
         Current function value: 273.091877
         Iterations: 7
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
```

(continues on next page)

(continued from previous page)

```
1.0                 1.0                 1.0                 -                1.0              ␣
↪             24.59260538017
0.01678939478565    0.01678939478565    0.01678939478565    0.9931439495955  0.
↪01678939478565    0.6055223166906
0.009965674493075   0.009965674493075   0.009965674493078   0.4242810241262  0.
↪009965674493075   3.244714326293
0.003397925619967   0.003397925619967   0.003397925619968   0.7237970170791  0.
↪003397925619967   63.79567916947
2.830731029084e-05  2.830731029084e-05  2.830731029086e-05  0.9917790420884  2.
↪830731029084e-05  77.95525849671
1.416553419389e-09  1.416553580528e-09  1.416553598767e-09  0.999949961019   1.
↪416553321704e-09  78.09518029969
7.075972310081e-14  7.098971338123e-14  7.097069021718e-14  0.9999498833268  7.
↪082766760582e-14  78.095187309
Optimization terminated successfully.
         Current function value: 78.095187
         Iterations: 6
```

```
-- 2020-11-09 15:09:00 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1943112264485     0.1943112264485     0.1943112264485     0.8115414580695  0.
↪1943112264485     293.1180738751
0.05737071259203    0.05737071259203    0.05737071259203    0.7291321197238  0.
↪05737071259203    17.07106219013
0.01062277327644    0.01062277327644    0.01062277327644    0.8173814867435  0.
↪01062277327644    12.44885046687
0.009064514495371   0.009064514495372   0.009064514495372   0.1550649500025  0.
↪009064514495372   28.00868491133
0.002334119807294   0.002334119807293   0.002334119807293   0.8051326751218  0.
↪002334119807295   275.8552661516
0.0006946146456239  0.0006946146456238  0.0006946146456238  0.7097377670861  0.
↪0006946146456242  332.7560201847
0.000253341221249   0.0002533412212489  0.0002533412212489  0.6718481634042  0.
↪0002533412212491  377.2933943083
1.33722226265e-06   1.337222262655e-06  1.337222262657e-06  0.9952598213341  1.
↪337222262651e-06  392.3599718847
6.775876566156e-11  6.775875940049e-11  6.775875524037e-11  0.9999493313269  6.
↪775868905564e-11  392.4568216146
Optimization terminated successfully.
         Current function value: 392.456822
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1030590385675     0.1030590385675     0.1030590385675     0.9011154818883  0.
↪1030590385675     219.2748221074
0.02654054666621    0.02654054666621    0.02654054666621    0.7752562655553  0.
↪02654054666621    276.7551091027
0.01427394633699    0.01427394633699    0.01427394633699    0.4912526709581  0.
↪01427394633699    607.4391713228
```

(continues on next page)

(continued from previous page)

```
0.001218580492651   0.001218580492601   0.001218580492601   0.9248974186534  0.
↪001218580492676   1166.289755796
1.888403761715e-05  1.888403761622e-05  1.888403761624e-05  0.985551099478   1.
↪888403761735e-05  1241.740132447
1.545745837373e-09  1.545745372552e-09  1.545745362755e-09  0.999922927438   1.
↪545745648845e-09  1242.779847889
1.692223225044e-10  1.873153075245e-10  1.873152955153e-10  0.8801597673128  1.
↪817506825368e-10  1242.779937345
Optimization terminated successfully.
         Current function value: 1242.779937
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪            652.1069903706
0.06096672005102    0.06096672005103    0.060966720051      0.9444993128467  0.
↪06096672005103    285.1503667166
0.01533649030442    0.01533649030442    0.01533649030441    0.760955159853   0.
↪01533649030442    2.726105681791
0.006320164709415   0.006320164709416   0.006320164709413   0.6160648365512  0.
↪006320164709416   8.631043022899
0.001197892469647   0.001197892469648   0.001197892469647   0.8360719893043  0.
↪001197892469648   8.304987261732
0.0005713687249976  0.0005713687249978  0.0005713687249976  0.5371632525845  0.
↪0005713687249978  6.593737888213
0.0001900730365094  0.0001900730365095  0.0001900730365094  0.7071525519853  0.
↪0001900730365095  4.503362444169
1.636573313793e-05  1.636573313793e-05  1.636573313793e-05  0.9478311770592  1.
↪636573313794e-05  4.082883819104
7.532302844599e-09  7.532302083082e-09  7.532302077642e-09  0.9995526057271  7.
↪532302082324e-09  3.99997015618
3.762057344252e-13  3.766204589771e-13  3.766274429159e-13  0.9999499983548  3.
↪766188718361e-13  3.99995064979
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪            24.59260538017
0.03041802557839    0.03041802557839    0.03041802557839    0.9772064682649  0.
↪03041802557839    24.19388314957
0.009080107307655   0.009080107307655   0.009080107307656   0.7409207447829  0.
↪009080107307655   102.5179052635
0.0005454700267143  0.000545470026725   0.000545470026725   0.95051584666    0.
↪0005454700267272  180.762473129
4.528574406413e-08  4.528574409137e-08  4.528574404608e-08  0.9999182057566  4.
↪528574406479e-08  184.4441432226
2.264218525495e-12  2.264162473465e-12  2.264155892645e-12  0.9999500028083  2.
↪264287232588e-12  184.4445388081
Optimization terminated successfully.
         Current function value: 184.444539
         Iterations: 5
```

```
-- 2020-11-09 15:09:07 - muse.mca - WARNING
Check growth constraints for wind.
```

(continues on next page)

(continued from previous page)

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1943112264485     0.1943112264485     0.1943112264485     0.8115414580695  0.
↪1943112264485      293.1180738751
0.05737071259203    0.05737071259203    0.05737071259203    0.7291321197238  0.
↪05737071259203     17.07106219013
0.01062277327644    0.01062277327644    0.01062277327644    0.8173814867435  0.
↪01062277327644     12.44885046687
0.009064514495371   0.009064514495372   0.009064514495372   0.1550649500025  0.
↪009064514495372    28.00868491133
0.002334119807294   0.002334119807293   0.002334119807293   0.8051326751218  0.
↪002334119807295    275.8552661516
0.0006946146456239  0.0006946146456238  0.0006946146456238  0.7097377670861  0.
↪0006946146456242   332.7560201847
0.000253341221249   0.0002533412212489  0.0002533412212489  0.6718481634042  0.
↪0002533412212491   377.2933943083
1.33722226265e-06   1.337222262655e-06  1.337222262657e-06  0.9952598213341  1.
↪337222262651e-06   392.3599718847
6.775876566156e-11  6.775875940049e-11  6.775875524037e-11  0.9999493313269  6.
↪775868905564e-11   392.4568216146
Optimization terminated successfully.
         Current function value: 392.456822
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             414.316476678
0.1030590385675     0.1030590385675     0.1030590385675     0.9011154818883  0.
↪1030590385675      219.2748221074
0.02654054666621    0.02654054666621    0.02654054666621    0.7752562655553  0.
↪02654054666621     276.7551091027
0.01427394633699    0.01427394633699    0.01427394633699    0.4912526709581  0.
↪01427394633699     607.4391713228
0.001218580492651   0.001218580492601   0.001218580492601   0.9248974186534  0.
↪001218580492676    1166.289755796
1.888403761715e-05  1.888403761622e-05  1.888403761624e-05  0.985551099478   1.
↪888403761735e-05   1241.740132447
1.545745837373e-09  1.545745372552e-09  1.545745362755e-09  0.999922927438   1.
↪545745648845e-09   1242.779847889
1.692223225044e-10  1.873153075245e-10  1.873152955153e-10  0.8801597673128  1.
↪817506825368e-10   1242.779937345
Optimization terminated successfully.
         Current function value: 1242.779937
         Iterations: 7
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             652.1069903706
0.06096672005102    0.06096672005103    0.060966720051      0.9444993128467  0.
↪06096672005103     285.1503667166
0.01533649030442    0.01533649030442    0.01533649030441    0.760955159853   0.
↪01533649030442     2.726105681791
```

(continues on next page)

(continued from previous page)

```
0.006320164709415   0.006320164709416   0.006320164709413   0.6160648365512  0.
↪006320164709416   8.631043022899
0.001197892469647   0.001197892469648   0.001197892469647   0.8360719893043  0.
↪001197892469648   8.304987261732
0.0005713687249976  0.0005713687249978  0.0005713687249976  0.5371632525845  0.
↪0005713687249978  6.593737888213
0.0001900730365094  0.0001900730365095  0.0001900730365094  0.7071525519853  0.
↪0001900730365095  4.503362444169
1.636573313793e-05  1.636573313793e-05  1.636573313793e-05  0.9478311770592  1.
↪636573313794e-05  4.082883819104
7.532302844599e-09  7.532302083082e-09  7.532302077642e-09  0.9995526057271  7.
↪532302082324e-09  3.99997015618
3.762057344252e-13  3.766204589771e-13  3.766274429159e-13  0.9999499983548  3.
↪766188718361e-13  3.99995064979
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪           24.59260538017
0.03041802557839    0.03041802557839    0.03041802557839    0.9772064682649  0.
↪03041802557839    24.19388314957
0.009080107307655   0.009080107307655   0.009080107307656   0.7409207447829  0.
↪009080107307655   102.5179052635
0.0005454700267143  0.000545470026725   0.000545470026725   0.95051584666    0.
↪0005454700267272  180.762473129
4.528574406413e-08  4.528574409137e-08  4.528574404608e-08  0.9999182057566  4.
↪528574406479e-08  184.4441432226
2.264218525495e-12  2.264162473465e-12  2.264155892645e-12  0.9999500028083  2.
↪264287232588e-12  184.4445388081
Optimization terminated successfully.
         Current function value: 184.444539
         Iterations: 5
```

```
-- 2020-11-09 15:09:12 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪           454.6784294315
0.1862232334687     0.1862232334687     0.1862232334687     0.8174567476576  0.
↪1862232334687     282.4632604722
0.02870747975048    0.02870747975048    0.02870747975048    0.8875534078522  0.
↪02870747975048    46.09791383177
0.009705128410004   0.009705128410004   0.009705128410004   0.6723641007507  0.
↪009705128410004   35.13402203751
0.007693915131578   0.007693915131578   0.007693915131578   0.2197837130033  0.
↪007693915131578   80.93881539206
0.001699044794839   0.001699044794837   0.001699044794837   0.8250841182677  0.
↪00169904479484    298.2653882331
0.0005650008980062  0.0005650008980057  0.0005650008980057  0.678565948553   0.
↪0005650008980065  327.975809321
0.0002196033953225  0.0002196033953223  0.0002196033953223  0.6469207178836  0.
↪0002196033953226  358.4708363196
```

(continues on next page)

(continued from previous page)

```
2.579861478512e-06  2.579861478501e-06  2.579861478502e-06  0.9887803221269  2.
↪579861478497e-06  368.5514208652
1.310502062448e-10  1.310502049574e-10  1.31050204896e-10   0.9999492039366  1.
↪310502128569e-10  368.6632645094
Optimization terminated successfully.
         Current function value: 368.663265
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪             454.6784294315
0.06717002148527    0.06717002148527    0.06717002148527    0.939336263039   0.
↪06717002148527    183.2225626364
0.0335506482529     0.0335506482529     0.0335506482529     0.5230681149682  0.
↪0335506482529     256.2098343218
0.01040420231972    0.01040420231972    0.01040420231972    0.6975119886622  0.
↪01040420231972    178.3301859411
0.004349337378734   0.004349337378734   0.004349337378734   0.6235886580387  0.
↪004349337378734   396.8309868051
0.002928936786464   0.002928936786463   0.002928936786463   0.3467952576409  0.
↪002928936786463   488.976382689
0.0001458208281248  0.0001458208281499  0.0001458208281499  0.9601001354313  0.
↪0001458208281409  665.3317152907
1.593588680453e-07  1.593588684743e-07  1.593588684755e-07  0.9989479829423  1.
↪593588661973e-07  675.8691141106
7.991316370683e-12  7.990632294946e-12  7.990631661844e-12  0.999949860274   7.
↪990813072693e-12  675.8826828943
Optimization terminated successfully.
         Current function value: 675.882683
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪             759.5666413636
0.06103168736637    0.06103168736636    0.06103168736636    0.9444947751848  0.
↪06103168736636    386.894766737
0.01481387164895    0.01481387164895    0.01481387164895    0.7670031635967  0.
↪01481387164895    2.63017161912
0.005246182165771   0.00524618216577    0.005246182165771   0.676128398925   0.
↪00524618216577    11.26512518349
0.001435544233241   0.00143554423324    0.00143554423324    0.7534457362106  0.
↪00143554423324    12.27554761078
0.000168669976258   0.000168669976258   0.000168669976258   0.8892431693931  0.
↪000168669976258   6.683524472165
1.71245789073e-05   1.712457890729e-05  1.712457890729e-05  0.9390213283973  1.
↪712457890728e-05  6.18232423823
6.830533706094e-08  6.830533980833e-08  6.830533980775e-08  0.9982184386442  6.
↪830533976625e-08  6.066658211057
3.416078908143e-12  3.415974879856e-12  3.415978735438e-12  0.9999499895809  3.
↪41597612292e-12   6.066591807906
Optimization terminated successfully.
         Current function value: 6.066592
         Iterations: 8
```

```
-- 2020-11-09 15:09:17 - muse.mca - WARNING
Check growth constraints for wind.
```

(continues on next page)

(continued from previous page)

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             454.6784294315
0.1862232334687     0.1862232334687     0.1862232334687     0.8174567476576  0.
↪1862232334687     282.4632604722
0.02870747975048    0.02870747975048    0.02870747975048    0.8875534078522  0.
↪02870747975048    46.09791383177
0.009705128410004   0.009705128410004   0.009705128410004   0.6723641007507  0.
↪009705128410004   35.13402203751
0.007693915131578   0.007693915131578   0.007693915131578   0.2197837130033  0.
↪007693915131578   80.93881539206
0.001699044794839   0.001699044794837   0.001699044794837   0.8250841182677  0.
↪00169904479484    298.2653882331
0.0005650008980062  0.0005650008980057  0.0005650008980057  0.678565948553   0.
↪0005650008980065  327.975809321
0.0002196033953225  0.0002196033953223  0.0002196033953223  0.6469207178836  0.
↪0002196033953226  358.4708363196
2.579861478512e-06  2.579861478501e-06  2.579861478502e-06  0.9887803221269  2.
↪579861478497e-06  368.5514208652
1.310502062448e-10  1.310502049574e-10  1.31050204896e-10   0.9999492039366  1.
↪310502128569e-10  368.6632645094
Optimization terminated successfully.
         Current function value: 368.663265
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             454.6784294315
0.06717002148527    0.06717002148527    0.06717002148527    0.939336263039   0.
↪06717002148527    183.2225626364
0.0335506482529     0.0335506482529     0.0335506482529     0.5230681149682  0.
↪0335506482529     256.2098343218
0.01040420231972    0.01040420231972    0.01040420231972    0.6975119886622  0.
↪01040420231972    178.3301859411
0.004349337378734   0.004349337378734   0.004349337378734   0.6235886580387  0.
↪004349337378734   396.8309868051
0.002928936786464   0.002928936786463   0.002928936786463   0.3467952576409  0.
↪002928936786463   488.976382689
0.0001458208281248  0.0001458208281499  0.0001458208281499  0.9601001354313  0.
↪0001458208281409  665.3317152907
1.593588680453e-07  1.593588684743e-07  1.593588684755e-07  0.9989479829423  1.
↪593588661973e-07  675.8691141106
7.991316370683e-12  7.990632294946e-12  7.990631661844e-12  0.999949860274   7.
↪990813072693e-12  675.8826828943
Optimization terminated successfully.
         Current function value: 675.882683
         Iterations: 8
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             759.5666413636
0.06103168736637    0.06103168736636    0.06103168736636    0.9444947751848  0.
↪06103168736636    386.894766737
```

(continues on next page)

(continued from previous page)

```
0.01481387164895    0.01481387164895    0.01481387164895    0.7670031635967  0.
↪01481387164895    2.63017161912
0.005246182165771   0.00524618216577    0.005246182165771   0.676128398925   0.
↪00524618216577    11.26512518349
0.001435544233241   0.00143554423324    0.00143554423324    0.7534457362106  0.
↪00143554423324    12.27554761078
0.000168669976258   0.000168669976258   0.000168669976258   0.8892431693931  0.
↪000168669976258   6.683524472165
1.71245789073e-05   1.712457890729e-05  1.712457890729e-05  0.9390213283973  1.
↪712457890728e-05  6.18232423823
6.830533706094e-08  6.830533980833e-08  6.830533980775e-08  0.9982184386442  6.
↪830533976625e-08  6.066658211057
3.416078908143e-12  3.415974879856e-12  3.415978735438e-12  0.9999499895809  3.
↪41597612292e-12   6.066591807906
Optimization terminated successfully.
         Current function value: 6.066592
         Iterations: 8
```

```
-- 2020-11-09 15:09:22 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             506.4464461439
0.1717574295871     0.1717574295871     0.1717574295871     0.8319963473544  0.
↪1717574295871     268.7406800963
0.06638577511032    0.06638577511032    0.06638577511033    0.6490768395246  0.
↪06638577511033    376.5617800925
0.0108119991109     0.01081199911089    0.01081199911089    0.8457086649906  0.
↪01081199911089    156.44831956
0.006166341998836   0.006166341998834   0.006166341998835   0.4575243211097  0.
↪006166341998835   243.5697308399
0.0005157722017999  0.0005157722017955  0.0005157722017955  0.9303097249327  0.
↪0005157722018036  353.933860755
7.752032087913e-08  7.75203208986e-08   7.752032089586e-08  0.9998584907938  7.
↪75203208902e-08   354.2607816225
1.875610793654e-11  1.875609893145e-11  1.875610020548e-11  0.9997580492617  1.
↪875613913502e-11  354.2617798007
Optimization terminated successfully.
         Current function value: 354.261780
         Iterations: 7
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter       Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             506.4464461439
0.1079826576971     0.1079826576971     0.1079826576971     0.8981297430914  0.
↪1079826576971     201.9066839097
0.04065945390789    0.04065945390789    0.04065945390789    0.6593890232973  0.
↪04065945390789    434.9407529189
0.01889246737726    0.01889246737727    0.01889246737727    0.5585864512458  0.
↪01889246737727    712.8967272059
0.001989080291767   0.001989080291864   0.001989080291864   0.9101975479096  0.
↪001989080291929   1386.629312789
2.319893722758e-06  2.31989372302e-06   2.319893722997e-06  0.9988740155508  2.
↪319893723235e-06  1475.934762389
```

(continues on next page)

(continued from previous page)

```
2.096073130738e-10  2.096071489352e-10  2.096071783066e-10  0.9999096479509  2.
↪096074566229e-10  1476.090721225
Optimization terminated successfully.
         Current function value: 1476.090721
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           889.6273141593
0.06094445869208    0.06094445869208    0.06094445869209    0.9446346580036  0.
↪06094445869208    507.5729159191
0.0140365256742     0.0140365256742     0.0140365256742     0.7774935142178  0.
↪0140365256742     2.547633056773
0.004912856564238   0.004912856564238   0.004912856564239   0.6815812213596  0.
↪004912856564237   14.44422608905
0.001078619945183   0.001078619945164   0.001078619945164   0.8144176821684  0.
↪001078619945165   17.16956450895
0.0005055468167838  0.000505546816775   0.0005055468167751  0.5422993251434  0.
↪0005055468167754  10.66487053069
0.0001219875076483  0.0001219875076462  0.0001219875076463  0.801041914532   0.
↪0001219875076463  5.108155023129
2.275536452812e-05  2.275536452773e-05  2.275536452773e-05  0.8511017161659  2.
↪275536452774e-05  4.268976835492
5.871314065763e-08  5.87131408013e-08   5.871314080683e-08  0.9974816389309  5.
↪871314078161e-08  4.000401347274
2.935817263126e-12  2.935852912569e-12  2.935856570528e-12  0.9999499966411  2.
↪935857873721e-12  3.999950673031
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           888.9606557232
0.06089606977121    0.06089606977121    0.06089606977121    0.9428879411484  0.
↪06089606977121    766.2404996017
0.005721125845739   0.005721125845738   0.005721125845739   0.9069719079827  0.
↪005721125845738   5.33096424619
0.002161096200032   0.002161096200032   0.002161096200032   0.6566312030915  0.
↪002161096200032   20.91332490724
0.0003725944226319  0.0003725944226319  0.0003725944226319  0.8524952692869  0.
↪0003725944226319  8.881368524706
0.0001720989711016  0.0001720989711015  0.0001720989711015  0.5509232609834  0.
↪0001720989711015  4.926154010814
1.220446310927e-05  1.22044631093e-05   1.220446310931e-05  1.0              1.
↪220446310931e-05  0.246666757403
2.473680488043e-08  2.473680488002e-08  2.47368048808e-08   0.9984936714025  2.
↪47368048808e-08   0.0001969335672683
1.907668751839e-12  1.907671450774e-12  1.90766909387e-12   0.9999228813462  1.
↪90766909387e-12   1.551815937867e-08
5.529091076088e-13  5.529110976982e-13  5.529090987305e-13  0.7177319323253  5.
↪529090987305e-13  5.40574204651e-09
2.202175165779e-13  2.202166746453e-13  2.202178545523e-13  0.6416601486717  2.
↪202178545522e-13  2.996616136035e-08
6.244002250284e-14  6.244046003975e-14  6.244022524357e-14  0.7587549421636  6.
↪244022524355e-14  8.360160383532e-08
```

(continues on next page)

(continued from previous page)

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 11
```

```
-- 2020-11-09 15:09:28 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            506.4464461439
0.1717574295871     0.1717574295871     0.1717574295871     0.8319963473544  0.
↪1717574295871      268.7406800963
0.06638577511032    0.06638577511032    0.06638577511033    0.6490768395246  0.
↪06638577511033     376.5617800925
0.0108119991109     0.01081199911089    0.01081199911089    0.8457086649906  0.
↪01081199911089     156.44831956
0.006166341998836   0.006166341998834   0.006166341998835   0.4575243211097  0.
↪006166341998835    243.5697308399
0.0005157722017999  0.0005157722017955  0.0005157722017955  0.9303097249327  0.
↪0005157722018036   353.933860755
7.752032087913e-08  7.75203208986e-08   7.752032089586e-08  0.9998584907938  7.
↪75203208902e-08    354.2607816225
1.875610793654e-11  1.875609893145e-11  1.875610020548e-11  0.9997580492617  1.
↪875613913502e-11   354.2617798007
Optimization terminated successfully.
         Current function value: 354.261780
         Iterations: 7
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            506.4464461439
0.1079826576971     0.1079826576971     0.1079826576971     0.8981297430914  0.
↪1079826576971      201.9066839097
0.04065945390789    0.04065945390789    0.04065945390789    0.6593890232973  0.
↪04065945390789     434.9407529189
0.01889246737726    0.01889246737727    0.01889246737727    0.5585864512458  0.
↪01889246737727     712.8967272059
0.001989080291767   0.001989080291864   0.001989080291864   0.9101975479096  0.
↪001989080291929    1386.629312789
2.319893722758e-06  2.31989372302e-06   2.319893722997e-06  0.9988740155508  2.
↪319893723235e-06   1475.934762389
2.096073130738e-10  2.096071489352e-10  2.096071783066e-10  0.9999096479509  2.
↪096074566229e-10   1476.090721225
Optimization terminated successfully.
         Current function value: 1476.090721
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0        ␣
↪            889.6273141593
0.06094445869208    0.06094445869208    0.06094445869209    0.9446346580036  0.
↪06094445869208     507.5729159191
0.0140365256742     0.0140365256742     0.0140365256742     0.7774935142178  0.
↪0140365256742      2.547633056773
0.004912856564238   0.004912856564238   0.004912856564239   0.6815812213596  0.
↪004912856564237    14.44422608905
```

(continues on next page)

(continued from previous page)

```
0.001078619945183   0.001078619945164   0.001078619945164   0.8144176821684  0.
↪001078619945165   17.16956450895
0.0005055468167838  0.000505546816775   0.0005055468167751  0.5422993251434  0.
↪0005055468167754  10.66487053069
0.0001219875076483  0.0001219875076462  0.0001219875076463  0.801041914532   0.
↪0001219875076463  5.108155023129
2.275536452812e-05  2.275536452773e-05  2.275536452773e-05  0.8511017161659  2.
↪275536452774e-05  4.268976835492
5.871314065763e-08  5.87131408013e-08   5.871314080683e-08  0.9974816389309  5.
↪871314078161e-08  4.000401347274
2.935817263126e-12  2.935852912569e-12  2.935856570528e-12  0.9999499966411  2.
↪935857873721e-12  3.999950673031
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 9
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0               ␣
↪           888.9606557232
0.06089606977121    0.06089606977121    0.06089606977121    0.9428879411484  0.
↪06089606977121    766.2404996017
0.005721125845739   0.005721125845738   0.005721125845739   0.9069719079827  0.
↪005721125845738   5.33096424619
0.002161096200032   0.002161096200032   0.002161096200032   0.6566312030915  0.
↪002161096200032   20.91332490724
0.0003725944226319  0.0003725944226319  0.0003725944226319  0.8524952692869  0.
↪0003725944226319  8.881368524706
0.0001720989711016  0.0001720989711015  0.0001720989711015  0.5509232609834  0.
↪0001720989711015  4.926154010814
1.220446310927e-05  1.22044631093e-05   1.220446310931e-05  1.0              1.
↪220446310931e-05  0.246666757403
2.473680488043e-08  2.473680488002e-08  2.47368048808e-08   0.9984936714025  2.
↪47368048808e-08   0.0001969335672683
1.907668751839e-12  1.907671450774e-12  1.90766909387e-12   0.9999228813462  1.
↪90766909387e-12   1.551815937867e-08
5.529091076088e-13  5.529110976982e-13  5.529090987305e-13  0.7177319323253  5.
↪529090987305e-13  5.40574204651e-09
2.202175165779e-13  2.202166746453e-13  2.202178545523e-13  0.6416601486717  2.
↪202178545522e-13  2.996616136035e-08
6.244002250284e-14  6.244046003975e-14  6.244022524357e-14  0.7587549421636  6.
↪244022524355e-14  8.360160383532e-08
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 11
```

```
-- 2020-11-09 15:09:33 - muse.mca - WARNING
Check growth constraints for wind.
```

```
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0               ␣
↪           568.8949309456
0.1574999940443     0.1574999940443     0.1574999940443     0.8462995490748  0.
↪1574999940443     259.3692323549
0.07523425397025    0.07523425397025    0.07523425397025    0.5547720211262  0.
↪07523425397025    453.5904211877
```

(continues on next page)

(continued from previous page)

```
0.01015218722862    0.01015218722862    0.01015218722862    0.8831324835902  0.
↪01015218722862    172.5109988087
0.005597873343747   0.005597873343747   0.005597873343747   0.4782155730769  0.
↪005597873343747   257.1812475882
0.0004682435100659  0.0004682435100634  0.0004682435100634  0.9301215087302  0.
↪000468243510069   357.0084554784
6.611502592752e-08  6.611502591615e-08  6.611502591873e-08  0.999863385888   6.
↪611502594636e-08  355.7877874388
7.555502313764e-12  7.555508723405e-12  7.555505999397e-12  0.9998857217433  7.
↪555493074812e-12  355.7884668282
Optimization terminated successfully.
         Current function value: 355.788467
         Iterations: 7
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪            568.8949309456
0.07655552550188    0.07655552550188    0.07655552550188    0.9326533055761  0.
↪07655552550188    158.9571104913
0.02915610316818    0.02915610316819    0.02915610316819    0.6583861263339  0.
↪02915610316818    366.3114833866
0.01377238842158    0.01377238842158    0.01377238842158    0.5494198953479  0.
↪01377238842158    509.1861831748
0.002113311162461   0.002113311162461   0.002113311162461   0.8672288939288  0.
↪002113311162461   954.4881944043
1.802801647426e-06  1.802801647524e-06  1.802801647519e-06  0.9992160883437  1.
↪802801646136e-06  1007.960176974
1.128093610417e-10  1.128094830078e-10  1.128094936008e-10  0.9999374254592  1.
↪128097428918e-10  1008.067349764
Optimization terminated successfully.
         Current function value: 1008.067350
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪            1027.252465794
0.06092556982108    0.06092556982105    0.06092556982104    0.9446936429103  0.
↪06092556982104    639.5720130299
0.01301817113373    0.01301817113372    0.01301817113372    0.7926577624161  0.
↪01301817113372    2.572827353367
0.004562995467295   0.004562995467293   0.004562995467292   0.6817945715562  0.
↪004562995467292   18.38802438984
0.001187610205527   0.001187610205525   0.001187610205525   0.7731355396853  0.
↪001187610205525   21.29230074562
0.0004286830611871  0.0004286830611869  0.0004286830611868  0.6428182053059  0.
↪0004286830611868  11.09687380181
9.112758693237e-05  9.112758693242e-05  9.112758693242e-05  0.8412920104849  9.
↪11275869324e-05   4.992047466728
2.247353056843e-05  2.247353056846e-05  2.247353056845e-05  0.7901238115153  2.
↪247353056845e-05  4.333289150699
1.030728394581e-07  1.030728396726e-07  1.030728396812e-07  0.9955619145694  1.
↪030728396813e-07  4.000897045216
5.155084903017e-12  5.15512049257e-12   5.155123215439e-12  0.9999499861783  5.
↪155122339295e-12  3.99995066737
2.970831115849e-16  2.567650272094e-16  2.577287629196e-16  0.999949996482   2.
↪580007603385e-16  3.99995062003
```

(continues on next page)

(continued from previous page)

```
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             513.6262328969
0.06112168433353    0.06112168433353    0.06112168433352    0.9426661283905  0.
↪06112168433353    286.3443459917
0.01863946704738    0.01863946704738    0.01863946704738    0.7054291768553  0.
↪01863946704738    1.912202845507
0.0064383053884     0.0064383053884     0.006438305388398   0.6923307144584  0.
↪0064383053884     10.96992216863
0.0005492326542255  0.0005492326542115  0.0005492326542114  0.9375350483781  0.
↪0005492326542115  9.941037067989
4.307299686793e-06  4.307299686644e-06  4.307299686641e-06  0.9947606446392  4.
↪307299686645e-06  6.689178746935
2.155106691573e-10  2.155106735225e-10  2.155106567961e-10  0.9999499661842  2.
↪15510656451e-10   6.666585491792
1.767668975714e-13  1.759148670162e-14  1.760582438766e-14  0.9999183071531  1.
↪503270896521e-14  6.66658436143
Optimization terminated successfully.
         Current function value: 6.666584
         Iterations: 7
-- 2020-11-09 15:09:39 - muse.mca - WARNING
Check growth constraints for wind.

Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             568.8949309456
0.1574999940443     0.1574999940443     0.1574999940443     0.8462995490748  0.
↪1574999940443     259.3692323549
0.07523425397025    0.07523425397025    0.07523425397025    0.5547720211262  0.
↪07523425397025    453.5904211877
0.01015218722862    0.01015218722862    0.01015218722862    0.8831324835902  0.
↪01015218722862    172.5109988087
0.005597873343747   0.005597873343747   0.005597873343747   0.4782155730769  0.
↪005597873343747   257.1812475882
0.0004682435100659  0.0004682435100634  0.0004682435100634  0.9301215087302  0.
↪000468243510069   357.0084554784
6.611502592752e-08  6.611502591615e-08  6.611502591873e-08  0.999863385888   6.
↪611502594636e-08  355.7877874388
7.555502313764e-12  7.555508723405e-12  7.555505999397e-12  0.9998857217433  7.
↪555493074812e-12  355.7884668282
Optimization terminated successfully.
         Current function value: 355.788467
         Iterations: 7
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             568.8949309456
0.07655552550188    0.07655552550188    0.07655552550188    0.9326533055761  0.
↪07655552550188    158.9571104913
```

(continues on next page)

(continued from previous page)

```
0.02915610316818    0.02915610316819    0.02915610316819    0.6583861263339  0.
↪02915610316818    366.3114833866
0.01377238842158    0.01377238842158    0.01377238842158    0.5494198953479  0.
↪01377238842158    509.1861831748
0.002113311162461   0.002113311162461   0.002113311162461   0.8672288939288  0.
↪002113311162461   954.4881944043
1.802801647426e-06  1.802801647524e-06  1.802801647519e-06  0.9992160883437  1.
↪802801646136e-06  1007.960176974
1.128093610417e-10  1.128094830078e-10  1.128094936008e-10  0.9999374254592  1.
↪128097428918e-10  1008.067349764
Optimization terminated successfully.
         Current function value: 1008.067350
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           1027.252465794
0.06092556982108    0.06092556982105    0.06092556982104    0.9446936429103  0.
↪06092556982104    639.5720130299
0.01301817113373    0.01301817113372    0.01301817113372    0.7926577624161  0.
↪01301817113372    2.572827353367
0.004562995467295   0.004562995467293   0.004562995467292   0.6817945715562  0.
↪004562995467292   18.38802438984
0.001187610205527   0.001187610205525   0.001187610205525   0.7731355396853  0.
↪001187610205525   21.29230074562
0.0004286830611871  0.0004286830611869  0.0004286830611868  0.6428182053059  0.
↪0004286830611868  11.09687380181
9.112758693237e-05  9.112758693242e-05  9.112758693242e-05  0.8412920104849  9.
↪11275869324e-05   4.992047466728
2.247353056843e-05  2.247353056846e-05  2.247353056845e-05  0.7901238115153  2.
↪247353056845e-05  4.333289150699
1.030728394581e-07  1.030728396726e-07  1.030728396812e-07  0.9955619145694  1.
↪030728396813e-07  4.000897045216
5.155084903017e-12  5.15512049257e-12   5.155123215439e-12  0.9999499861783  5.
↪155122339295e-12  3.99995066737
2.970831115849e-16  2.567650272094e-16  2.577287629196e-16  0.999949996482   2.
↪580007603385e-16  3.99995062003
Optimization terminated successfully.
         Current function value: 3.999951
         Iterations: 10
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪           513.6262328969
0.06112168433353    0.06112168433353    0.06112168433352    0.9426661283905  0.
↪06112168433353    286.3443459917
0.01863946704738    0.01863946704738    0.01863946704738    0.7054291768553  0.
↪01863946704738    1.912202845507
0.0064383053884     0.0064383053884     0.006438305388398   0.6923307144584  0.
↪0064383053884     10.96992216863
0.0005492326542255  0.0005492326542115  0.0005492326542114  0.9375350483781  0.
↪0005492326542115  9.941037067989
4.307299686793e-06  4.307299686644e-06  4.307299686641e-06  0.9947606446392  4.
↪307299686645e-06  6.689178746935
2.155106691573e-10  2.155106735225e-10  2.155106567961e-10  0.9999499661842  2.
↪15510656451e-10   6.666585491792
```

(continues on next page)

(continued from previous page)

```
1.767668975714e-13  1.759148670162e-14  1.760582438766e-14  0.9999183071531  1.
↪503270896521e-14  6.66658436143
Optimization terminated successfully.
         Current function value: 6.666584
         Iterations: 7
```

```
-- 2020-11-09 15:09:45 - muse.mca - WARNING
Check growth constraints for wind.
```

```
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             647.4402248427
0.1042702416317     0.1042702416317     0.1042702416317     0.9044651903652  0.
↪1042702416317     197.7436068274
0.04330783922975    0.04330783922975    0.04330783922975    0.6213338928986  0.
↪04330783922975    554.8458695929
0.02029339907042    0.02029339907042    0.02029339907042    0.5546164251713  0.
↪02029339907042    887.503418141
0.002059630009005   0.002059630008646   0.002059630008646   0.9123945707354  0.
↪002059630008855   1773.497308596
1.96164830749e-06   1.961648306624e-06  1.961648306623e-06  0.9995475315428  1.
↪96164830693e-06   1899.795955905
1.183904657193e-10  1.183907029537e-10  1.183907180056e-10  0.9999396473343  1.
↪183903667831e-10  1900.055708261
Optimization terminated successfully.
         Current function value: 1900.055708
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             1199.542590081
0.06080970203684    0.06080970203684    0.06080970203685    0.9448463893022  0.
↪06080970203684    802.3521760538
0.01206790508619    0.01206790508619    0.01206790508619    0.8067148881887  0.
↪01206790508618    2.242531350155
0.004235304009502   0.004235304009502   0.004235304009502   0.6829892140301  0.
↪004235304009502   22.93309435161
0.001076107298245   0.001076107298268   0.001076107298268   0.7811880414727  0.
↪001076107298267   27.34472817154
0.0001381467026663  0.0001381467026694  0.0001381467026694  0.9534954297556  0.
↪0001381467026692  2.164474733354
0.0001283068198708  0.0001283068198737  0.0001283068198737  0.07280278532081 0.
↪0001283068198735  2.0117557724
1.709535384529e-06  1.709535384566e-06  1.709535384568e-06  0.9870423815898  1.
↪709535384566e-06  0.04804313579566
9.45480215837e-11   9.454801752592e-11  9.454801899378e-11  0.9999447725614  9.
↪454801899363e-11  2.655850431687e-06
8.641189470312e-12  8.641191779805e-12  8.641199906593e-12  0.9099589255942  8.
↪64119990658e-12   2.42972823089e-07
8.227204364827e-12  8.227212664943e-12  8.22721600211e-12   0.05277665620137 8.
↪227216002098e-12  2.322662502381e-07
1.085487238602e-12  1.085487876362e-12  1.085489602398e-12  0.8709818852506  1.
↪085489602396e-12  3.141737288768e-08
7.818953396427e-13  7.81893890829e-13   7.818962895149e-13  0.3024720499645  7.
↪818962895138e-13  2.33935913105e-08
```

(continues on next page)

(continued from previous page)

```
2.33759416086e-13   2.337579557771e-13  2.337613758542e-13  0.7142468606499  2.
↪337613758539e-13  9.290715381741e-09
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             599.7712950406
0.06106893109436    0.06106893109436    0.06106893109435    0.9427080518507  0.
↪06106893109436    369.2205608767
0.0172839506186     0.0172839506186     0.0172839506186     0.7253158185948  0.
↪0172839506186     1.815008327441
0.006506405689011   0.006506405689011   0.00650640568901    0.660706520954   0.
↪006506405689012   13.26117009766
0.0007534849804658  0.0007534849804373  0.0007534849804372  0.9127389540888  0.
↪0007534849804374  13.3711836365
0.0002310664967685  0.0002310664967596  0.0002310664967596  0.7145302516257  0.
↪0002310664967597  7.652720903336
1.357431783524e-06  1.35743178356e-06   1.357431783558e-06  1.0              1.
↪357431783551e-06  5.273685381749
7.205799155642e-11  7.205807383533e-11  7.205806647959e-11  0.9999469527435  7.
↪205806692488e-11  5.266601996919
3.494750239057e-15  3.599609449551e-15  3.605486637761e-15  0.9999499638259  3.
↪603660446069e-15  5.26660163652
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
```

```
-- 2020-11-09 15:09:50 - muse.mca - WARNING
Check growth constraints for wind.
```

```
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             647.4402248427
0.1042702416317     0.1042702416317     0.1042702416317     0.9044651903652  0.
↪1042702416317     197.7436068274
0.04330783922975    0.04330783922975    0.04330783922975    0.6213338928986  0.
↪04330783922975    554.8458695929
0.02029339907042    0.02029339907042    0.02029339907042    0.5546164251713  0.
↪02029339907042    887.503418141
0.002059630009005   0.002059630008646   0.002059630008646   0.9123945707354  0.
↪002059630008855   1773.497308596
1.96164830749e-06   1.961648306624e-06  1.961648306623e-06  0.9995475315428  1.
↪96164830693e-06   1899.795955905
1.183904657193e-10  1.183907029537e-10  1.183907180056e-10  0.9999396473343  1.
↪183903667831e-10  1900.055708261
Optimization terminated successfully.
         Current function value: 1900.055708
         Iterations: 6
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0              ␣
↪             1199.542590081
```

(continues on next page)

(continued from previous page)

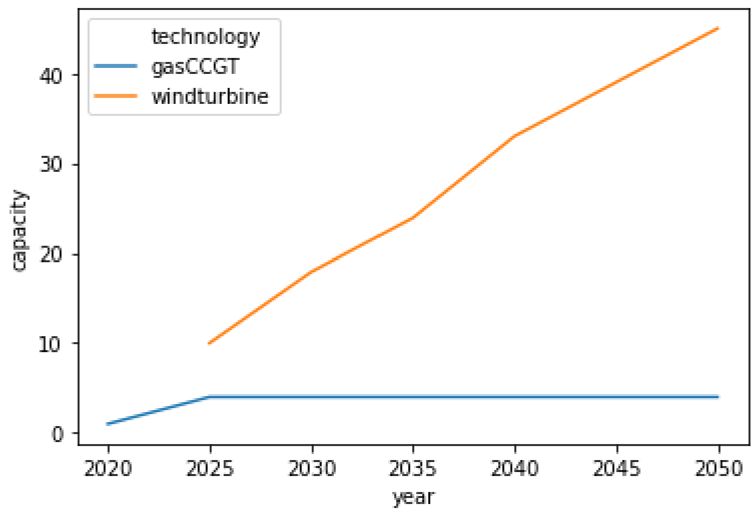
```
0.06080970203684    0.06080970203684    0.06080970203685    0.9448463893022  0.
↪06080970203684     802.3521760538
0.01206790508619    0.01206790508619    0.01206790508619    0.8067148881887  0.
↪01206790508618     2.242531350155
0.004235304009502   0.004235304009502   0.004235304009502   0.6829892140301  0.
↪004235304009502    22.93309435161
0.001076107298245   0.001076107298268   0.001076107298268   0.7811880414727  0.
↪001076107298267    27.34472817154
0.0001381467026663  0.0001381467026694  0.0001381467026694  0.9534954297556  0.
↪0001381467026692   2.164474733354
0.0001283068198708  0.0001283068198737  0.0001283068198737  0.07280278532081 0.
↪0001283068198735   2.0117557724
1.709535384529e-06  1.709535384566e-06  1.709535384568e-06  0.9870423815898  1.
↪709535384566e-06   0.04804313579566
9.45480215837e-11   9.454801752592e-11  9.454801899378e-11  0.9999447725614  9.
↪454801899363e-11   2.655850431687e-06
8.641189470312e-12  8.641191779805e-12  8.641199906593e-12  0.9099589255942  8.
↪64119990658e-12    2.42972823089e-07
8.227204364827e-12  8.227212664943e-12  8.22721600211e-12   0.05277665620137 8.
↪227216002098e-12   2.322662502381e-07
1.085487238602e-12  1.085487876362e-12  1.085489602398e-12  0.8709818852506  1.
↪085489602396e-12   3.141737288768e-08
7.818953396427e-13  7.81893890829e-13   7.818962895149e-13  0.3024720499645  7.
↪818962895138e-13   2.33935913105e-08
2.33759416086e-13   2.337579557771e-13  2.337613758542e-13  0.7142468606499  2.
↪337613758539e-13   9.290715381741e-09
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 13
HIHIHIHIHI
Primal Feasibility  Dual Feasibility    Duality Gap         Step             Path␣
↪Parameter      Objective
1.0                 1.0                 1.0                 -                1.0      ␣
↪           599.7712950406
0.06106893109436    0.06106893109436    0.06106893109435    0.9427080518507  0.
↪06106893109436     369.2205608767
0.0172839506186     0.0172839506186     0.0172839506186     0.7253158185948  0.
↪0172839506186      1.815008327441
0.006506405689011   0.006506405689011   0.00650640568901    0.660706520954   0.
↪006506405689012    13.26117009766
0.0007534849804658  0.0007534849804373  0.0007534849804372  0.9127389540888  0.
↪0007534849804374   13.3711836365
0.0002310664967685  0.0002310664967596  0.0002310664967596  0.7145302516257  0.
↪0002310664967597   7.652720903336
1.357431783524e-06  1.35743178356e-06   1.357431783558e-06  1.0              1.
↪357431783551e-06   5.273685381749
7.205799155642e-11  7.205807383533e-11  7.205806647959e-11  0.9999469527435  7.
↪205806692488e-11   5.266601996919
3.494750239057e-15  3.599609449551e-15  3.605486637761e-15  0.9999499638259  3.
↪603660446069e-15   5.26660163652
Optimization terminated successfully.
         Current function value: 5.266602
         Iterations: 8
```

```
-- 2020-11-09 15:09:55 - muse.mca - WARNING
Check growth constraints for wind.
```

```
[8]: import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot

     results = pd.read_csv("Results/MCACapacity.csv")
     results
     sns.lineplot(data=results[results.sector=="power"], x='year', y='capacity', hue=
     ↪'technology')
```

```
[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe8f77df460>
```



7.3 Indices and tables

- genindex
- modindex
- search

CHAPTER

EIGHT

API

MUSE model.

8.1 Market Clearing Algorithm

8.1.1 Main MCA

class muse.mca.**FindEquilibriumResults**(*converged: bool*, *market: Dataset*, *sectors: List[AbstractSector]*)

Result of find equilibrium.

converged: bool

Alias for field number 0

market: Dataset

Alias for field number 1

sectors: List[AbstractSector]

Alias for field number 2

class muse.mca.**MCA**(*sectors: List[muse.sectors.abstract.AbstractSector]*, *market: xarray.core.dataset.Dataset*, *outputs: Optional[Callable[[List[muse.sectors.abstract.AbstractSector], xarray.core.dataset.Dataset], Any]] = None*, *time_framework: Sequence[int] = [2010, 2020, 2030, 2040, 2050, 2060, 2070, 2080, 2090]*, *equilibrium: bool = True*, *expect_equilibrium: bool = True*, *equilibrium_variable: str = 'demand'*, *maximum_iterations: int = 3*, *tolerance: float = 0.1*, *tolerance_unmet_demand: float = - 0.1*, *excluded_commodities: Optional[Sequence[str]] = None*, *carbon_budget: Optional[Sequence] = None*, *carbon_price: Optional[Sequence] = None*, *carbon_commodities: Optional[Sequence[str]] = None*, *debug: bool = False*, *control_undershoot: bool = True*, *control_overshoot: bool = True*, *carbon_method: str = 'fitting'*, *method_options: Optional[Mapping] = None*)

Market Clearing Algorithm.

The market clearing algorithm is the main object implementing the MUSE model. It is responsible for orchestrating how the sectors are run, how they interface with one another, with the general market and the carbon market.

calibrate_legacy_sectors()

Run a calibration step in the lagacy sectors.

TODO: Remove when LegacySectors are no longer needed.

classmethod factory(*settings: Union[str, pathlib.Path, Mapping, Any]*) → muse.mca.MCA
Loads MCA from input settings and input files.

Parameters **settings** – namedtuple with the global MUSE input settings.

Returns The loaded MCA

find_equilibrium(*market: xarray.core.dataset.Dataset, sectors: Optional[List[muse.sectors.abstract.AbstractSector]] = None*) → muse.mca.FindEquilibriumResults
Specialised version of the find_equilibrium function.

Parameters **market** – Commodities market, with the prices, supply, consumption and demand.

Returns A tuple with the updated market (prices, supply, consumption and demand) and sector.

run() → None
Initiates the calculation, starting with the loop over years.

This method starts the main MUSE loop, going over the years of the simulation. Internally, it runs the carbon budget loop, which updates the carbon prices, if needed, and the equilibrium loop, which tries to reach an equilibrium between prices, demand and supply.

Returns None

update_carbon_budget(*market: xarray.core.dataset.Dataset, year_idx: int*) → float
Specialised version of the update_carbon_budget function.

Parameters

- **market** – Commodities market, with the prices, supply, consumption and demand.
- **year_idx** – Index of the year of interest.

Returns An updated market with prices, supply, consumption and demand.

update_carbon_price(*market*) → Optional[float]
Calculates the updated carbon price, if required.

If the emission calculated for the next time period is larger than the limit, then the carbon price needs to be updated to ensure that whatever the sectors do, the carbon budget limit is not exceeded.

Parameters **market** – Market, with the prices, supply, consumption and demand.

Returns The new carbon price or None

class muse.mca.**SingleYearIterationResult**(*market: Dataset, sectors: List[AbstractSector]*)
Result of iterating over sectors for a year.

Convenience tuple naming naming the return values from of `single_year_iteration()`.

market: Dataset
Alias for field number 0

sectors: List[AbstractSector]
Alias for field number 1

muse.mca.**check_demand_fulfillment**(*market: xarray.core.dataset.Dataset, tol: float, excluded_commodities: Optional[Sequence] = None*) → bool
Checks if the supply will fulfill all the demand in the future.

If it does not, it logs a warning.

Parameters

- **market** – Commodities market, with the prices, supply, consumption and demand.

- **tol** – Tolerance for the unmet demand.

Returns True if the supply fulfils the demand; False otherwise

`muse.mca.`**`check_equilibrium`**(*market: xarray.core.dataset.Dataset*, *int_market: xarray.core.dataset.Dataset*, *tolerance: float*, *equilibrium_variable: str*, *year: Optional[int] = None*) → bool

Checks if equilibrium has been reached.

This function checks if the difference in either the demand or the prices between iterations if smaller than certain tolerance. If is, then it is assumed that the process has converged.

Parameters

- **market** – The market values in this iteration.
- **int_market** – The market values in the previous iteration.
- **tolerance** – Tolerance for reaching equilibrium.
- **equilibrium_variable** – Variable to use to calculate the equilibrium condition.
- **year** – year for which to check changes. Default to minimum year in market.

Returns True if converged, False otherwise.

`muse.mca.`**`find_equilibrium`**(*market: xarray.core.dataset.Dataset*, *sectors: List[muse.sectors.abstract.AbstractSector]*, *maxiter: int = 3*, *tol: float = 0.1*, *equilibrium_variable: str = 'demand'*, *tol_unmet_demand: float = - 0.1*, *excluded_commodities: Optional[Sequence] = None*, *expect_equilibrium: bool = True*) → muse.mca.FindEquilibriumResults

Runs the equilibrium loop.

If convergence is reached, then the function returns the new market. If the maximum number of iterations are reached, then a warning issued in the log and the function returns with the current status.

Parameters

- **market** – Commodities market, with the prices, supply, consumption and demand.
- **sectors** – A list of the sectors participating in the simulation.
- **maxiter** – Maximum number of iterations.
- **tol** – Tolerance for reaching equilibrium.
- **equilibrium_variable** – Variable to use to calculate the equilibrium condition.
- **tol_unmet_demand** – Tolerance for the unmet demand.
- **excluded_commodities** – Commodities to be excluded in check_demand_fulfillment
- **expect_equilibrium** – if equilibrium should be reached. Useful to testing.

Returns A tuple with the updated market (prices, supply, consumption and demand), sectors, and convergence status.

`muse.mca.`**`single_year_iteration`**(*market: xarray.core.dataset.Dataset*, *sectors: List[muse.sectors.abstract.AbstractSector]*) → muse.mca.SingleYearIterationResult

Runs one iteration of the sectors (runs each sector once).

Parameters

- **market** – An initial market with prices, supply, consumption.
- **sectors** – A list of the sectors participating in the simulation.

Returns A tuple with the new market and sectors.

8.1.2 Carbon Budget

`muse.carbon_budget.`**`CARBON_BUDGET_FITTERS: MutableMapping[str, Callable[[numpy.ndarray, nump`**

Dictionary of carbon budget fitters.

`muse.carbon_budget.`**`CARBON_BUDGET_FITTERS_SIGNATURE`**

carbon budget fitters signature.

alias of Callable[[numpy.ndarray, numpy.ndarray, int], float]

`muse.carbon_budget.`**`CARBON_BUDGET_METHODS`**

Dictionary of carbon budget methods checks.

`muse.carbon_budget.`**`CARBON_BUDGET_METHODS_SIGNATURE`**

carbon budget fitters signature.

alias of Callable[[xarray.core.dataset.Dataset, list, Callable, xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray], float]

`muse.carbon_budget.`**`create_sample`**(*carbon_price*, *current_emissions*, *budget*, *size=4*)

Calculates a sample of carbon prices to estimate the adjusted carbon price.

For each of these prices, the equilibrium loop will be run, obtaining a new value for the emissions. Out of those price-emissions pairs, the final carbon price will be estimated.

Parameters

- **`carbon_price`** – Current carbon price
- **`current_emissions`** – Current emissions
- **`budget`** – Carbon budget
- **`size`** – Number of points in the sample

Returns An array with the sample prices.

`muse.carbon_budget.`**`exp_guess_and_weights`**(*prices: numpy.ndarray*, *emissions: numpy.ndarray*, *budget: int*) → tuple

Estimates initial values for the exponential fitting algorithm and the weights.

The points closest to the budget are used to estimate the initial guess. They also have the highest weight.

Parameters

- **`prices`** – An array with the sample carbon prices
- **`emissions`** – An array with the corresponding emissions
- **`budget`** – The carbon budget for the time period

Returns The initial guess and weights

`muse.carbon_budget.`**`exponential`**(*prices: numpy.ndarray*, *emissions: numpy.ndarray*, *budget: int*) → float

Fits the prices-emissions pairs to an exponential function.

Once that is done, an optimal carbon price is estimated

Parameters

- **`prices`** – An array with the sample carbon prices
- **`emissions`** – An array with the corresponding emissions

- **budget** – The carbon budget for the time period

Returns The optimal carbon price.

`muse.carbon_budget.`**`linear`**(*prices: numpy.ndarray*, *emissions: numpy.ndarray*, *budget: int*) → float

Fits the prices-emissions pairs to a linear function.

Once that is done, an optimal carbon price is estimated

Parameters

- **prices** – An array with the sample carbon prices
- **emissions** – An array with the corresponding emissions
- **budget** – The carbon budget for the time period

Returns The optimal carbon price.

`muse.carbon_budget.`**`linear_guess_and_weights`**(*prices: numpy.ndarray*, *emissions: numpy.ndarray*, *budget: int*) → tuple

Estimates initial values for the linear fitting algorithm and the weights.

The points closest to the budget are used to estimate the initial guess. They also have the highest weight.

Returns The initial guess and weights

`muse.carbon_budget.`**`refine_new_price`**(*market: xarray.core.dataset.Dataset*, *historic_price: xarray.core.dataarray.DataArray*, *carbon_budget: xarray.core.dataarray.DataArray*, *sample: numpy.ndarray*, *price: float*, *commodities: list*, *price_too_high_threshold: float*) → float

Refine the value of the carbon price to ensure it is not too high or low. :param market: Market, with the prices, supply, consumption and demand. :param historic_price: DataArray with the historic carbon prices. :param carbon_budget: DataArray with the carbon budget. :param sample: Sample carbon price points. :param price: Current carbon price, to be refined. :param commodities: List of carbon-related commodities. :param price_too_high_threshold: Threshold to decide what is a price too high.

Returns A refined carbon price.

`muse.carbon_budget.`**`register_carbon_budget_fitter`**(*function: Callable[[numpy.ndarray, numpy.ndarray, int], float] = None*)

Decorator to register a carbon budget function.

`muse.carbon_budget.`**`register_carbon_budget_method`**(*function: Callable[[xarray.core.dataset.Dataset, list, Callable, xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray], float] = None*)

Decorator to register a carbon budget function.

`muse.carbon_budget.`**`update_carbon_budget`**(*carbon_budget: Sequence[float]*, *emissions: float*, *year_idx: int*, *over: bool = True*, *under: bool = True*) → float

Adjust the carbon budget in the far future if emissions too high or low.

Returns An adjusted threshold for the far future year

8.2 Sectors and associated functionality

Define a sector, e.g. aggregation of agents.

There are three main kinds of sectors classes, encompassing three use cases:

- `Sector`: The main workhorse sector of the model. It contains only on kind of data, namely the agents responsible for holding assets and investing in new assets.
- `PresetSector`: A sector that is meant to generate demand for the sectors above using a fixed formula or schedule.
- `LegacySector`: A wrapper around the original MUSE sectors.

All the sectors derive from `AbstractSector`. The `AbstractSector` defines two abstract functions which should be declared by derived sectors. Abstract here means a common programming practice where some concept in the code (e.g. a sector) is given an explicit interface, with the goal of making it easier for other programmers to use and implement the concept.

- `AbstractSector.factory()`: Creates a sector from input data
- `AbstractSector.next()`: A function which takes a market (demand, supply, prices) and returns a market. What happens within could be anything, though it will likely constists of dispatch and investment.

New sectors can be registered with the MUSE input files using `muse.sectors.register.register_sector()`.

`@muse.sectors.register.`**`register_sector`**(*sector_class: Optional[Type[muse.sectors.abstract.AbstractSector]] = None, name: Optional[Union[str, Sequence[str]]] = None*) → Type[muse.sectors.abstract.AbstractSector]

Registers a sector so it is available MUSE-wide.

Example

```
>>> from muse.sectors import AbstractSector, register_sector
>>> @register_sector(name="MyResidence")
... class ResidentialSector(AbstractSector):
...     pass
```

8.2.1 AbstractSector

class `muse.sectors.`**`AbstractSector`**

Abstract base class for sectors.

Sectors are part of type hierarchy with `AbstractSector` at the apex: all sectors should derive from `AbstractSector` directly or indirectly.

MUSE only requires two things of a sector. Sector should be instanstiable via a `factory()` function. And they should be callable via `next()`.

`AbstractSector` declares an interface with these two functions. Sectors which derive from it will be warned if either method is not implemented.

abstract classmethod factory(*name: str, settings: Any*) → muse.sectors.abstract.AbstractSector

Creates class from settings named-tuple.

abstract next(*mca_market: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Advance sector by one time period.

8.2.2 Sector

class muse.sectors.sector.**Sector**(*name: str*, *technologies: xarray.core.dataset.Dataset*, *subsectors: Sequence[muse.sectors.subsector.Subsector] = []*, *timeslices: Optional[pandas.core.indexes.multi.MultiIndex] = None*, *interactions: Optional[Callable[[Sequence[muse.agents.agent.AbstractAgent]], None]] = None*, *interpolation: str = 'linear'*, *outputs: Optional[Callable] = None*, *supply_prod: Optional[Callable[[xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], xarray.core.dataarray.DataArray]] = None*)

Base class for all sectors.

property agents
Iterator over all agents in the sector.

property capacity
Aggregates capacity across agents.

The capacities are aggregated leaving only two dimensions: asset (technology, installation date, region), year.

static convert_market_timeslice(*market: xarray.core.dataset.Dataset*, *timeslice: pandas.core.indexes.multi.MultiIndex*, *intensive: Union[str, Tuple[str]] = 'prices'*) → xarray.core.dataset.Dataset
Converts market from one to another timeslice.

classmethod factory(*name: str*, *settings: Any*) → muse.sectors.sector.Sector
Creates class from settings named-tuple.

property forecast
Maximum forecast horizon across agents.

If no agents with a "forecast" attribute are found, defaults to 5. It cannot be lower than 1 year.

interactions
Interactions between agents.

Called right before computing new investments, this function should manage any interactions between agents, e.g. passing assets from *new* agents to *retro* agents, and maket make-up from *retro* to *new*.

Defaults to doing nothing.

The function takes the sequence of agents as input, and returns nothing. It is expected to modify the agents in-place.

See also:

muse.interactions

interpolation: Mapping[Text, Any]
Interpolation method and arguments when computing years.

market_variables(*market: xarray.core.dataset.Dataset*, *technologies: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Computes resulting market: production, consumption, and costs.

name: Text
Name of the sector.

next (*mca_market: xarray.core.dataset.Dataset*, *time_period: Optional[int] = None*, *current_year: Optional[int] = None*) → xarray.core.dataset.Dataset
Advance sector by one time period.

Parameters

- **mca_market** – Market with `demand`, `supply`, and `prices`.
- **time_period** – Length of the time period in the framework. Defaults to the range of `mca_market.year`.

Returns A market containing the `supply` offered by the sector, it's attendant `consumption` of fuels and materials and the associated `costs`.

outputs: Callable
A function for outputing data for post-mortem analysis.

subsectors: Sequence[Subsector]
Subsectors controlled by this object.

supply_prod
Computes production as used to return the supply to the MCA.

It can be anything registered with `@register_production`.

technologies: xr.Dataset
Parameters describing the sector's technologies.

timeslices: Optional[pd.MultiIndex]
Timeslice at which this sector operates.

If None, it will operate using the timeslice of the input market.

8.2.3 Subsector

class muse.sectors.subsector.**Subsector** (*agents: Sequence[muse.agents.agent.Agent]*, *commodities: Sequence[str]*, *demand_share: Optional[Callable] = None*, *constraints: Optional[Callable] = None*, *name: str = 'subsector'*, *forecast: int = 5*)
Agent group servicing a subset of the sectorial commodities.

8.2.4 PresetSector

class muse.sectors.preset_sector.**PresetSector** (*presets: xarray.core.dataset.Dataset*, *interpolation_mode: str = 'linear'*, *name: str = 'preset'*)
Sector with outcomes fixed from the start.

classmethod factory (*name: str*, *settings: Any*) → muse.sectors.preset_sector.PresetSector
Constructs a PresetSectors from input data.

interpolation_mode: Text
Interpolation method

name
Name by which to identify a sector

next (*mca_market: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Advance sector by one time period.

presets: Dataset
Market across time and space.

8.2.5 LegacySector

class muse.sectors.legacy_sector.**LegacySector** (*name: str*, *old_sector*, *timeslices: Dict*, *commodities: Dict*, *commodity_price: xarray.core.dataarray.DataArray*, *static_trade: xarray.core.dataarray.DataArray*, *regions: Sequence*, *time_framework: numpy.ndarray*, *mode: str*, *excess: Union[int, float]*, *market_iterative: str*, *sectors_dir: str*, *output_dir: str*)

calibrated
Flag if the sector has gone through the calibration process.

commodities
Commodities for each sector, as well as global commodities.

commodity_price
Initial price of all the commodities.

dims
Order of the input and output dimensions.

excess
Allowed excess of capacity.

classmethod factory (*name: str*, *settings: Any*, ***kwargs*) → muse.sectors.legacy_sector.LegacySector
Creates class from settings named-tuple.

property global_commodities
List of all commodities used by the MCA.

inputs (*consumption: xarray.core.dataarray.DataArray*, *prices: xarray.core.dataarray.DataArray*, *supply: xarray.core.dataarray.DataArray*)
Converts xarray to MUSE numpy input arrays.

static load_timeslices_and_aggregation (*timeslices*, *sectors*) → Tuple[dict, str]
Loads all sector timeslices and finds the finest one.

market_iterative
——> TODO what's this parameter?

mode
If 'Calibration', the sector runs in calibration mode

name
Name of the sector

next (*market: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset
Adapter between the old and the new.

old_sector
Legacy sector method to run the calculation

output_dir
Outputs directory.

outputs (*consumption: numpy.ndarray*, *prices: numpy.ndarray*, *supply: numpy.ndarray*) → xarray.core.dataset.Dataset
Converts MUSE numpy outputs to xarray.

regions
Regions taking part in the simulation.

property sector_commodities
List of all commodities used by the Sector.

property sector_timeslices
List of all commodities used by the MCA.

sectors_dir
Sectors directory.

static_trade
Static trade needed for the conversion and supply sectors.

time_framework
Time framework of the complete simulation.

timeslices
Timeslices for sectors and mca.

8.2.6 Production

Various ways and means to compute production.

Production is the amount of commodities produced by an asset. However, depending on the context, it could be computed several ways. For instace, it can be obtained straight from the capacity of the asset. Or it can be obtained by matching for the same commodities with a set of assets.

Production methods can be registered via the `@register_production` production decorator. Registering a function makes the function accessible from MUSE's input file. Production methods are not expected to modify their arguments. Furthermore they should conform the following signatures:

```
@register_production
def production(
    market: xr.Dataset, capacity: xr.DataArray, technologies: xr.Dataset, **kwargs
) -> xr.DataArray:
    pass
```

param market Market, including demand and prices.

param capacity The capacity of each asset within a market.

param technologies A dataset characterising the technologies of the same assets.

param **kwargs Any number of keyword arguments

returns A *xr.DataArray* with the amount produced for each good from each asset.

`muse.production.`**PRODUCTION_SIGNATURE**
Production signature.

alias of Callable[[xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], xarray.core.dataarray.DataArray] `muse.production.`**`demand_matched_production`**(*market: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *costs: str = 'prices'*) → xarray.core.dataarray.DataArray

Production from matching demand via annual lcoe.

`muse.production.`**`factory`**(*settings: Union[str, Mapping] = 'maximum_production'*, ***kwargs*) → Callable[[xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], xarray.core.dataarray.DataArray]

Creates a production functor.

This function's raison d'être is to convert the input from a TOML file into an actual functor usable within the model, i.e. it converts data into logic.

Parameters

- **`name`** – Registered production method to create. The name is resolved when the function returned by the factory is called. Hence, it could refer to a function yet to be registered when this factory method is called.
- **`**kwargs`** – any keyword argument the production method accepts.

`muse.production.`**`maximum_production`**(*market: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*) → xarray.core.dataarray.DataArray

Production when running at full capacity.

Full capacity is limited by the utilitization factor. For more details, see `muse.quantities.maximum_production()`.

`muse.production.`**`register_production`**(*function: Callable[[xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], xarray.core.dataarray.DataArray] = None*)

Decorator to register a function as a production method.

See also:

`muse.production`

`muse.production.`**`supply`**(*market: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*) → xarray.core.dataarray.DataArray

Service current demand equally from all assets.

"Equally" means that equivalent technologies are used to the same percentage of their respective capacity.

8.2.7 Agent Interactions

Modes of interactions between agents.

Interactions between agents are modelled via two orthogonal concepts:

- a *net* is a set of agents which interact in some way
- an *interaction* proper is a function that takes a net and actually performs the interaction.

Hence, there are two registrators in this this module, `register_interaction_net()`, and `register_agent_interaction()`. The first registers functions that take the full set of agents as input and returns a sequence of nets. It is expected each net of the sequence will be applied the same interaction. The second registrator registers the interaction proper: it takes agents as arguments and returns nothing. It is expected to modify the agents in-place.

`muse.interactions.`**`factory`**(*inputs: Optional[Sequence[Union[Mapping, Tuple[str, str]]]] = None*) → Callable[[Sequence[muse.agents.agent.AbstractAgent]], None]

Creates an interaction functor.

`muse.interactions.`**`new_to_retro_net`**(*agents: Sequence[muse.agents.agent.Agent], first_category: str = 'newcapa'*) → Sequence[Sequence[muse.agents.agent.Agent]]

Interactions between new and retrofit agents.

`muse.interactions.`**`register_agent_interaction`**(*function: Callable[[muse.agents.agent.Agent, muse.agents.agent.Agent], None]*)

Decorator to register an agent to agent(s) interaction function.

An agent interaction function takes at least two agents and makes them interact in some way.

An agent interaction function also takes as argument a sector object. This object should not be modified in any way. But it can be queried for parameters, if the specific agent interaction function requires it. This is most likely the same configuration object passed on to the interaction net function.

`muse.interactions.`**`register_interaction_net`**(*function: Callable[[Sequence[muse.agents.agent.Agent]], Sequence[Sequence[muse.agents.agent.Agent]]]*)

Decorator to register a function computing interaction nets.

An interaction net function takes as input the list of all agents and returns the list of all interactions, where an interaction is a list of at least two interacting agents.

An interactiont-net function also takes as argument a sector object. This object should not be modified in any way. But it can be queried for parameters, if the specific interaction-net function requires it.

`muse.interactions.`**`transfer_assets`**(*from_: muse.agents.agent.Agent, to_: muse.agents.agent.Agent*) → None

Transfer assets from first agent to second agent.

8.3 Agents and associated functionalities

Holds all building agents.

`muse.agents.factories.`**`agents_factory`**(*params_or_path: Union[str, pathlib.Path, List], capacity: Union[xarray.core.dataarray.DataArray, str, pathlib.Path], technologies: xarray.core.dataset.Dataset, regions: Optional[Sequence[str]] = None, year: Optional[int] = None, **kwargs*) → List[muse.agents.agent.Agent]

Creates a list of agents for the chosen sector.

`muse.agents.factories.`**`create_newcapa_agent`**(*capacity: xarray.core.dataarray.DataArray, year: int, region: str, search_rules: Union[str, Sequence[str]] = 'all', interpolation: str = 'linear', merge_transform: Union[str, Mapping, Callable] = 'new', quantity: float = 0.3, housekeeping: Union[str, Mapping, Callable] = 'noop', **kwargs*)

Creates newcapa agent from muse primitives.

muse.agents.factories.**create_retrofit_agent**(*technologies: xarray.core.dataset.Dataset, capacity: xarray.core.dataarray.DataArray, share: str, year: int, region: str, interpolation: str = 'linear', decision: Union[Callable, str, Mapping] = 'mean', **kwargs*)

Creates retrofit agent from muse primitives.

muse.agents.factories.**factory**(*existing_capacity_path: Optional[Union[str, pathlib.Path]] = None, agent_parameters_path: Optional[Union[str, pathlib.Path]] = None, technodata_path: Optional[Union[str, pathlib.Path]] = None, sector: Optional[str] = None, sectors_directory: Union[str, pathlib.Path] = PosixPath('/Users/alexkell/Documents/SGI/2-documentation/StarMuse/docs/data'), baseyear: int = 2010*) → List[muse.agents.agent.Agent]

Reads list of agents from standard MUSE input files.

class muse.agents.agent.**AbstractAgent**(*name: str = 'Agent', region: str = '', assets: Optional[xarray.core.dataset.Dataset] = None, interpolation: str = 'linear', category: Optional[str] = None*)

Base class for all agents.

assets

Current stock of technologies.

category

Attribute to classify different sets of agents.

filter_input(*dataset: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray], year: Optional[Union[Sequence[int], int]] = None, **kwargs*) → Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]

Filter inputs for usage in agent.

For instance, filters down to agent's region, etc.

interpolation

Interpolation method.

name

Name associated with the agent

abstract next(*technologies: xarray.core.dataset.Dataset, market: xarray.core.dataset.Dataset, demand: xarray.core.dataarray.DataArray, time_period: int = 1*)

Iterates agent one turn.

The goal is to figure out from market variables which technologies to invest in and by how much.

region

Region the agent operates in

tolerance = 1e-12

tolerance criteria for floating point comparisons.

uuid

A unique identifier for the agent.

class muse.agents.agent.**Agent**(*name: str = 'Agent', region: str = 'USA', assets: Optional[xarray.core.dataset.Dataset] = None, interpolation: str = 'linear', search_rules: Optional[Callable] = None, objectives: Optional[Callable] = None, decision: Optional[Callable] = None, year: int = 2010, maturity_threshhold: float = 0, forecast: int = 5, housekeeping: Optional[Callable] = None, merge_transform: Optional[Callable] = None, demand_threshhold: Optional[float] = None, category: Optional[str] = None, **kwargs*)

Agent that is capable of computing a search-space and a cost metric.

This agent will not perform any investment itself.

decision

Creates single decision objective from one or more objectives.

demand_threshhold

Threshhold below which the demand share is zero.

This criteria avoids fulfilling demand for very small values. If None, then the criteria is not applied.

forecast

Number of years to look into the future for forecating purposed.

property forecast_year

Year to consider when forecasting.

housekeeping

Tranforms applied on the assets at the start of each iteration.

It could mean keeping the assets as are, or removing assets with no capacity in the current year and beyond, etc... It can be any function registered with `register_initial_asset_transform()`.

maturity_threshhold

Market share threshhold.

Threshhold when and if filtering replacement technologies with respect to market share.

merge_transform

Tranforms applied on the old and new assets.

It could mean using only the new assets, or merging old and new, etc... It can be any function registered with `register_final_asset_transform()`.

next(*technologies: xarray.core.dataset.Dataset, market: xarray.core.dataset.Dataset, demand: xarray.core.dataarray.DataArray, time_period: int = 1*) → Optional[xarray.core.dataset.Dataset]

Iterates agent one turn.

The goal is to figure out from market variables which technologies to invest in and by how much.

This function will modify *self.assets* and increment *self.year*. Other attributes are left unchanged. Arguments to the function are never modified.

objectives

One or more objectives by which to decide next investments.

search_rules: Callable

Search rule(s) determining potential replacement technologies.

This is a string referring to a filter, or a sequence of strings referring to multiple filters, applied one after the other. Any function registered via *muse.filters.register_filter* can be used to filter the search space.

year

Current year.

The year is incremented by one everytime next is called.

class muse.agents.agent.**InvestingAgent**(**args, constraints: Optional[Callable] = None, investment: Optional[Callable] = None, **kwargs*)

Agent that performs investment for itself.

_compute_new_assets(*demand: xarray.core.dataarray.DataArray, search: xarray.core.dataset.Dataset, technologies: xarray.core.dataset.Dataset, market: xarray.core.dataset.Dataset, time_period: int, current_year: int*) → xarray.core.dataarray.DataArray

Computes investment and retirement profile.

add_assets(*newassets: xarray.core.dataset.Dataset*)

Add new assets to the agent.

constraints

Creates a set of constraints limiting investment.

invest

Method to use when fulfilling demand from rated set of techs.

next(*technologies: xarray.core.dataset.Dataset, market: xarray.core.dataset.Dataset, demand: xarray.core.dataarray.DataArray, time_period: int = 1*)

Iterates agent one turn.

The goal is to figure out from market variables which technologies to invest in and by how much.

This function will modify *self.assets* and increment *self.year*. Other attributes are left unchanged. Arguments to the function are never modified.

8.3.1 Objectives

Valuation functions for replacement technologies.

Objectives are used to compare replacement technologies. They should correspond to a single well defined economic concept. Multiple objectives can later be combined via decision functions.

Objectives should be registered via the `@register_objective` decorator. This makes it possible to refer to them by name in agent input files, and nominally to set extra input parameters.

The `factory()` function creates a function that calls all objectives defined in its input argument and returns a dataset with each objective as a separate data array.

Objectives are not expected to modify their arguments. Furthermore they should conform the following signatures:

```
@register_objective
def comfort(
    agent: Agent,
    demand: xr.DataArray,
    search_space: xr.DataArray,
    technologies: xr.Dataset,
    market: xr.Dataset,
    **kwargs
) -> xr.DataArray:
    pass
```

param agent the agent relevant to the search space. The filters may need to query the agent for parameters, e.g. the current year, the interpolation method, the tolerance, etc.

param demand Demand to fulfill.

param search_space A boolean matrix represented as a `xr.DataArray`, listing replacement technologies for each asset.

param technologies A data set characterising the technologies from which the agent can draw assets.

param market Market variables, such as prices or current capacity and retirement profile.

param kwargs Extra input parameters. These parameters are expected to be set from the input file.

> **Warning:** The standard *agent csv file* does not allow to set these parameters.

returns A dataArray with at least one dimension corresponding to `replacement`. Only the technologies in `search_space.replacement` should be present. Furthermore, if an `asset` dimension is present, then it should correspond to `search_space.asset`. Other dimensions can be present, as long as the subsequent decision function nows how to reduce them.

`muse.objectives.`**`capacity_to_service_demand`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Minimum capacity required to fulfill the demand.

`muse.objectives.`**`capital_costs`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Capital costs for input technologies.

The capital costs are computed as $a * b^\alpha$, where a is "cap_par" from the *Techno-data*, b is the "scaling_size", and α is "cap_exp". In other words, capital costs are constant across the simulation for each technology.

`muse.objectives.`**`comfort`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Comfort value provided by technologies.

`muse.objectives.`**`efficiency`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Efficiency of the technologies.

`muse.objectives.`**`emission_cost`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Emission cost for each technology when fultfilling whole demand.

Given the demand share D, the emissions per amount produced E, and the prices per emittant P, then emissions costs C are computed as:

$$C = \sum_s \left(\sum_c D\right) \left(\sum_c EP\right),$$

with s the timeslices and c the commodity.

`muse.objectives.`**`equivalent_annual_cost`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, **args*, ***kwargs*)

Equivalent annual costs (or annualized cost) of a technology.

This is the cost that, if it were to occur equally in every year of the project lifetime, would give the same net present cost as the actual cash flow sequence associated with that component. The cost is computed using the annualized cost expression given by HOMER Energy.

Parameters

- **`agent`** – The agent of interest
- **`search_space`** – The search space space for replacement technologies
- **`technologies`** – All the technologies
- **`market`** – The market parameters

Returns xr.DataArray with the EAC calculated for the relevant technologies

`muse.objectives.`**`factory`**(*settings: Union[str, Mapping, Sequence[Union[str, Mapping]]] = 'LCOE'*) → Callable

Creates a function computing multiple objectives.

The input can be a single objective defined by its name alone. Or it can be a single objective defined by a dictionary which must include at least a "name" item, as well as any extra parameters to pass to the objective. Or it can be a sequence of objectives defined by name or by dictionary.

`muse.objectives.`**`fixed_costs`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Fixed costs associated with a technology.

Given a factor α and an exponent β, the fixed costs F are computed from the `capacity fulfilling the current demand` C as:

$$F = \alpha * C^\beta$$

α and β are "fix_par" and "fix_exp" in *Techno-data*, respectively.

`muse.objectives.`**`fuel_consumption_cost`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, **args*, ***kwargs*)

Cost of fuels when fulfilling whole demand.

`muse.objectives.`**`lifetime_levelized_cost_of_energy`**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, **args*, ***kwargs*)

Levelized cost of energy (LCOE) of technologies over their lifetime.

It follows the *simpified LCOE* given by NREL.

Parameters

- **agent** – The agent of interest
- **search_space** – The search space space for replacement technologies
- **technologies** – All the technologies
- **market** – The market parameters

Returns xr.DataArray with the LCOE calculated for the relevant technologies

muse.objectives.**net_present_value**(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, **args*, ***kwargs*)

Net present value (NPV) of the relevant technologies.

The net present value of a Component is the present value of all the revenues that a Component earns over its lifetime minus all the costs of installing and operating it. Follows the definition of the net present cost given by HOMER Energy.

- energy commodities INPUTS are related to fuel costs
- environmental commodities OUTPUTS are related to environmental costs
- material and service commodities INPUTS are related to consumable costs
- fixed and variable costs are given as technodata inputs and depend on the installed capacity and production (non-environmental), respectively
- capacity costs are given as technodata inputs and depend on the installed capacity

Note: Here, the installation year is always agent.year, since objectives compute the NPV for technologies to be installed in the current year. A more general NPV computation (which would then live in quantities.py) would have to refer to installation year of the technology.

Parameters

- **agent** – The agent of interest
- **search_space** – The search space space for replacement technologies
- **technologies** – All the technologies
- **market** – The market parameters

Returns xr.DataArray with the NPV calculated for the relevant technologies

muse.objectives.**register_objective**(*function: Callable[[muse.agents.agent.Agent, xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, xarray.core.dataset.Dataset, Any], xarray.core.dataarray.DataArray]*)

Decorator to register a function as a objective.

Registers a function as a objective so that it can be applied easily when sorting technologies one against the other.

The input name is expected to be in lower_snake_case, since it ought to be a python function. CamelCase, lowerCamelCase, and kebab-case names are also registered.

8.3.2 Search Rules

Various search-space filters.

Search-space filters return a modified matrix of booleans, with dimension *(asset, replacement)*, where *asset* refer to technologies currently managed by the agent, and *replacement* to all technologies the agent could consider, prior to filtering.

Filters should be registered using the decorator `register_filter()`. The registration makes it possible to call then from the agent by specifying the *search_rule* attribute. The *search_rule* attribute is string or list of strings specifying the filters to apply one after the other when considering the search space.

Filters are not expected to modify any of their arguments. They should all follow the same signature:

```
@register_filter
def search_space_filter(
    agent: Agent,
    search_space: xr.DataArray,
    technologies: xr.Dataset,
    market: xr.Dataset
) -> xr.DataArray:
    pass
```

param agent the agent relevant to the search space. The filters may need to query the agent for parameters, e.g. the current year, the interpolation method, the tolerance, etc.

param search_space the current search space.

param technologies A data set characterising the technologies from which the agent can draw assets.

param market Market variables, such as prices or current capacity and retirement profile.

returns A new search space with the same data-type as the input search-space, but with potentially different values.

In practice, an initial search space is created by calling a function with the signature given below, and registered with `register_initializer()`. The initializer function returns a search space which is passed on to a chain of filters, as done in the `factory()` function.

Functions creating initial search spaces should have the following signature:

```
@register_initializer
def search_space_initializer(
    agent: Agent,
    demand: xr.DataArray,
    technologies: xr.Dataset,
    market: xr.Dataset
) -> xr.DataArray:
    pass
```

param agent the agent relevant to the search space. The filters may need to query the agent for parameters, e.g. the current year, the interpolation method, the tolerance, etc.

param demand share of the demand per existing reference technology (e.g. assets).

param technologies A data set characterising the technologies from which the agent can draw assets.

param market Market variables, such as prices or current capacity and retirement profile.

returns An initial search space

`muse.filters.compress`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, ***kwargs*) → xarray.core.dataarray.DataArray

Compress search space to include only potential technologies.

This operation reduces the *size* of the search space along the *replacement* dimension, such that are left only technologies that will be considered as replacement for at least by one asset. Unlike most filters, it does not change the data, but rather changes how the data is represented. In other words, this is mostly an *optimization* for later steps, to avoid unnecessary computations.

`muse.filters.currently_existing_tech`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*) → xarray.core.dataarray.DataArray

Only consider technologies that currently exist in the market.

This filter only allows technologies that exists in the market and have non- zero capacity in the current year. See *currently_referenced_tech* for a similar filter that does not check the capacity.

`muse.filters.currently_referenced_tech`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*) → xarray.core.dataarray.DataArray

Only consider technologies that are currently referenced in the market.

This filter will allow any technology that exists in the market, even if it currently sits at zero capacity (unlike *currently_existing_tech* which requires non-zero capacity in the current year).

`muse.filters.factory`(*settings: Optional[Union[str, Mapping, Sequence[Union[str, Mapping]]]] = None*, *separator: str = '->'*)

Creates filters from input TOML data.

The input data is standardized to a list of dictionaries where each dictionary contains at least one member, "name".

The first dictionary specifies the initial function which creates the search space from the demand share, the market, and the dataset describing technologies in the sectors.

The next entries are applied in turn and transform the search space in some way. In other words the process is more or less:

```
search_space = initial_filter(
    agent, demand, technologies=technologies, market=market
)
for afilter in filters:
    search_space = afilter(
        agent, search_space, technologies=technologies, market=market
    )
return search_space
```

`initial_filter` is simply first filter given on input, if that filter is registered with `register_initializer()`. Otherwise, `initialize_from_technologies()` is automatically inserted.

`muse.filters.identity`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Returns search space as given.

`muse.filters.initialize_from_technologies`(*agent: muse.agents.agent.Agent*, *demand: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, **args*, ***kwargs*)

Initialize a search space from existing technologies.

`muse.filters.maturity`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, *enduse_label: str = 'service'*, ***kwargs*) → xarray.core.dataarray.DataArray

Only allows technologies that have achieve a given market share.

Specifically, the market share refers to the capacity for each end- use.

`muse.filters.reduce_asset`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, ***kwargs*) → xarray.core.dataarray.DataArray

Reduce over assets.

`muse.filters.register_filter`(*function: Callable[[muse.agents.agent.Agent, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, xarray.core.dataset.Dataset], xarray.core.dataarray.DataArray]*) → Callable

Decorator to register a function as a filter.

Registers a function as a filter so that it can be applied easily when constraining the technology search-space.

The name that the function is registered with defaults to the function name. However, it can also be specified explicitly as a *keyword* argument. In any case, it must be unique amongst all search-space filters.

`muse.filters.register_initializer`(*function: Callable[[muse.agents.agent.Agent, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, xarray.core.dataset.Dataset], xarray.core.dataarray.DataArray]*) → Callable

Decorator to register a function as a search-space initializer.

`muse.filters.same_enduse`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, **args*, *enduse_label: str = 'service'*, ***kwargs*) → xarray.core.dataarray.DataArray

Only allow for technologies with at least the same end-use.

`muse.filters.same_fuels`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, **args*, ***kwargs*)

Filters technologies with the same fuel type.

`muse.filters.similar_technology`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, **args*, ***kwargs*)

Filters technologies with the same type.

`muse.filters.with_asset_technology`(*agent: muse.agents.agent.Agent*, *search_space: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *market: xarray.core.dataset.Dataset*, ***kwargs*) → xarray.core.dataarray.DataArray

Search space *also* contains its asset technology for each asset.

8.3.3 Decision Methods

Decision methods combining several objectives into ones.

Decisions methods create a single scalar from multiple objectives. To be available from the input, functions implementing decision methods should follow a specific signature:

```
@register_decision
def weighted_sum(objectives: Dataset, parameters: Any, **kwargs) -> DataArray:
    pass
```

param objectives An dataset where each array is a separate objective

param parameters parameters, such as weigths, whether to minimize or maximize, the names of objectives to consider, etc.

param kwargs Extra input parameters. These parameters are expected to be set from the input file.

> **Warning:** The standard *agent csv file* does not allow to set these parameters.

returns A data array with ranked replacement technologies.

`muse.decisions.`**`epsilon_constraints`**(*objectives: xarray.core.dataset.Dataset, parameters: Sequence[Tuple[str, bool, float]], mask: Optional[Any] = None*) → xarray.core.dataarray.DataArray

Minimizes first objective subject to constraints on other objectives.

The parameters are a sequence of tuples *(name, minimize, epsilon)*, where *name* is the name of the objective, *minimze* is *True* if minimizing and false if maximizing that objective, and *epsilon* is the constraint. The first objective is the one that will be minimized according to:

Given objectives $O^{(i)}_t$, with $i \in [|1, N|]$ and t the replacement technologies, this function computes the ranking with respect to t:

$$\mathrm{ranking}_{O^{(i)}_t < \epsilon_i} O^{(0)}_t$$

The first tuple can be restricted to *(name, minimize)*, since *epsilon* is ignored.

The result is the matrix $O^{(0)}$ modified such minimizing over the replacement dimension value would take into account the constraints and the optimization direction (minimize or maximize). In other words, calling *result.rank('replacement')* will yield the expected result.

`muse.decisions.`**`factory`**(*settings: Union[str, Mapping] = 'mean'*) → Callable

Creates a decision method based on the input settings.

`muse.decisions.`**`lexical_comparison`**(*objectives: xarray.core.dataset.Dataset, parameters: Union[Sequence[Tuple[str, bool, float]], Sequence[Tuple[str, float]]]*) → xarray.core.dataarray.DataArray

Lexical comparison over the objectives.

Lexical comparison operates by binning the objectives into bins of width w_i = min_j(p_i o_i^j). Once binned, dimensions other than *asset* and *technology* are reduced by taking the max, e.g. the largest constraint. Finally, the objectives are ranked lexographically, in the order given by the parameters.

The result is an array of tuples which can subsquently be compared lexicographically.

`muse.decisions.mean`(*objectives: xarray.core.dataset.Dataset*, **args*, ***kwargs*) → xarray.core.dataarray.DataArray

Mean over objectives.

`muse.decisions.register_decision`(*function: Callable[[xarray.core.dataset.Dataset, Sequence[Tuple[str, bool, float]]], xarray.core.dataarray.DataArray]*, *name: str*)

Decorator to register a function as a decision.

Registers a function as a decision so that it can be applied easily when aggregating different objectives together.

`muse.decisions.retro_epsilon_constraints`(*objectives: xarray.core.dataset.Dataset*, *parameters: Sequence[Tuple[str, bool, float]]*) → xarray.core.dataarray.DataArray

Epsilon constraints where the current tech is included.

Modifies the parameters to the function such that the existing technologies are always competitive.

`muse.decisions.retro_lexical_comparison`(*objectives: xarray.core.dataset.Dataset*, *parameters: Union[Sequence[Tuple[str, bool, float]], Sequence[Tuple[str, float]]]*) → xarray.core.dataarray.DataArray

Lexical comparison over the objectives.

Lexical comparison operates by binning the objectives into bins of width w_i = p_i o_i, where i are the current assets. Once binned, dimensions other than *asset* and *replacement* are reduced by taking the max, e.g. the largest constraint. Finally, the objectives are ranked lexographically, in the order given by the parameters.

The result is an array of tuples which can subsquently be compared lexicographically.

`muse.decisions.single_objective`(*objectives: xarray.core.dataset.Dataset*, *parameters: Union[str, Tuple[str, bool], Tuple[str, bool, float], Sequence[Tuple[str, bool, float]]]*) → xarray.core.dataarray.DataArray

Single objective decision method.

It only decides on minimization vs maximization and multiplies by a given factor. The input parameters can take the following forms:

- Standard sequence *[(objective, direction, factor)]*, in which case it must have only one element.
- A single string: defaults to standard sequence *[(string, 1, 1)]*
- A tuple (string, bool): defaults to standard sequence *[(string, direction, 1)]*
- A tuple (string, bool, factor): defaults to standard sequence *[(string, direction, factor)]*

`muse.decisions.weighted_sum`(*objectives: xarray.core.dataset.Dataset*, *parameters: Mapping[str, float]*) → xarray.core.dataarray.DataArray

Weighted sum over normalized objectives.

The objectives are each normalized to [0, 1] over the *replacement* dimension. Furthermore, the dimensions other than *asset* and *replacement* are reduced by taking the mean.

More specifically, the objective function is:

$$\sum_m c_m \frac{A_m - \min(A_m)}{\max(A_m) - \min(A_m)}$$

where sum runs over the different objectives, c_m is a scalar coefficient, A_m is a matrix with dimensions (existing tech, replacemnt tech). *max(A)* and *min(A)* return the largest and smallest component of the input matrix. If c_m is positive, then that particular objective is minimized, whereas if it is negative, that particular objective is maximized.

8.3.4 Investment Methods

Investment decision.

An investment determines which technologies to invest given a metric to determine preferred technologies, a corresponding search space of technologies, and the demand to fulfill.

Investments should be registered via the decorator *register_investment*. The registration makes it possible to call investments dynamically through *compute_investment*, by specifying the name of the investment. It is part of MUSE's plugin platform.

Investments are not expected to modify any of their arguments. They should all have the following signature:

```
@register_investment
def investment(
    costs: DataArray,
    search_space: DataArray,
    technologies: Dataset,
    constraints: List[Constraint],
    year: int,
    **kwargs
) -> DataArray:
    pass
```

param costs specifies for each *asset* which *replacement* technology should be invested in preferentially. This should be an integer or floating point array with dimensions *asset* and *replacement*.

param search_space an *asset* by *replacement* matrix defining allowed and disallowed replacement technologies for each asset

param technologies a dataset containing all constant data characterizing the technologies.

param constraints a list of constraints as defined in `constraints`.

param year the current year.

returns A data array with dimensions *asset* and *technology* specifying the amount of newly invested capacity.

`muse.investments.`**`INVESTMENT_SIGNATURE`**

Investment signature.

alias of Callable[[xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, List[xarray.core.dataset.Dataset], Any], xarray.core.dataarray.DataArray]

`muse.investments.`**`cliff_retirement_profile`**(*technical_life: xarray.core.dataarray.DataArray, current_year: int = 0, protected: int = 0, interpolation: str = 'linear', **kwargs*) → xarray.core.dataarray.DataArray

Cliff-like retirement profile from current year.

Computes the retirement profile of all technologies in `technical_life`. Assets with a technical life smaller than the input time-period should automatically be renewed.

Hence, if `technical_life <= protected`, then effectively, the technical life is rewritten as `technical_life * n` with `n = int(protected // technical_life) + 1`.

We could just return an array where each year is repesented. Instead, to save memory, we return a compact view of the same where years where no change happens are removed.

Parameters

- **technical_life** – lifetimes for each technology
- **current_year** – current year
- **protected** – The technologies are assumed to be renewed between years *current_year* and *current_year + protected*
- ****kwargs** – arguments by which to filter technical_life, if any.

Returns A boolean DataArray where each each element along the year dimension is true if the technology is still not retired for the given year.

muse.investments.**register_investment**(*function: Callable[[xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, List[xarray.core.dataset.Dataset], Any], xarray.core.dataarray.DataArray]*) → Callable[[xarray.core.dataarray.DataArray, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, List[xarray.core.dataset.Dataset], Any], xarray.core.dataarray.DataArray]

Decorator to register a function as an investment.

8.3.5 Demand Share

Demand share computations.

The demand share splits a demand amongst agents. It is used within a sector to assign part of the input MCA demand to each agent.

Demand shares functions should be registered via the decorator *register_demand_share*.

Demand share functions are not expected to modify any of their arguments. They should all have the following signature:

```
@register_demand_share
def demand_share(
    agents: Sequence[AbstractAgent],
    market: xr.Dataset,
    technologies: xr.Dataset,
    **kwargs
) -> xr.DataArray:
    pass
```

param agents a sequence of agent relevant to the demand share procedure. The agent can be queried for parameters specific to the demand share procedure. For instance, :py:func`new_and_retro` will query the agents for the assets they own, the region they are contained with, their category (new or retrofit), etc…

param market Market variables, including prices, consumption and supply.

param technologies a dataset containing all constant data characterizing the technologies.

param kwargs Any number of keyword arguments that can parametrize how the demand is shared. These keyword arguments can be modified from the TOML file.

returns The unmet consumption. Unless indicated, all agents will compete for a the full demand. However, if there exists a coordinate "agent" of dimension "asset" giving the `uuid` of the agent, then agents will only service that par of the demand.

`muse.demand_share.DEMAND_SHARE_SIGNATURE`

Demand share signature.

alias of Callable[[Sequence[muse.agents.agent.AbstractAgent], xarray.core.dataset.Dataset, xarray.core.dataset.Dataset, Any], xarray.core.dataarray.DataArray]

`muse.demand_share.new_and_retro`(*agents: Sequence[muse.agents.agent.AbstractAgent]*, *market: xarray.core.dataset.Dataset*, *technologies: xarray.core.dataset.Dataset*, *production: Union[str, Mapping, Callable] = 'maximum_production'*, *current_year: Optional[int] = None*, *forecast: int = 5*) → xarray.core.dataarray.DataArray

Splits demand across new and retro agents.

The input demand is split amongst both *new* and *retro* agents. *New* agents get a share of the increase in demand for the forecast year, whereas *retrofi* agents are assigned a share of the demand that occurs from decommissioned assets.

Parameters

- **agents** – a list of all agents. This list should mainly be used to determine the type of an agent and the assets it owns. The agents will not be modified in any way.
- **market** – the market for which to satisfy the demand. It should contain at-least `consumption` and `supply`. It may contain `prices` if that is of use to the production method. The `consumption` reflects the demand for the commodities produced by the current sector.
- **technologies** – quantities describing the technologies.

Pseudo-code:

1. the capacity is reduced over agents and expanded over timeslices (extensive quantity) and aggregated over agents. Generally:

$$A^r_{a,s} = w_s \sum_i A^{r,i}_a$$

   with w_s a weight associated with each timeslice and determined via `muse.timeslices.convert_timeslice()`.

2. An intermediate quantity, the `unmet demand` U is defined from $P[\mathcal{M}, \mathcal{A}]$, a function giving the production for a given market $\mathcal{M}$, the associated consumption $\mathcal{C}$, and aggregate assets $\mathcal{A}$:

$$U[\mathcal{M}, \mathcal{A}] = \max(\mathcal{C} - P[\mathcal{M}, \mathcal{A}], 0)$$

   where max operates element-wise, and indices have been dropped for simplicity. The resulting expression has the same indices as the consumption $\mathcal{C}^r_{c,s}$.

   P is any function registered with `@register_production`.

3. the *new* demand N is defined as:

$$N = \min\left(\mathcal{C}^r_{c,s}(y + \Delta y) - \mathcal{C}^r_{c,s}(y), U[\mathcal{M}^r(y + \Delta y), \mathcal{A}^r_{a,s}(y)]\right)$$

4. the *retrofit* demand R is defined from the identity

$$C^r_{c,s}(y + \Delta y) = P[\mathcal{M}^r(y + \Delta y), \mathcal{A}^r_{a,s}(y + \Delta y)] + N^r_{c,s} + R^r_{c,s}$$

   In other words, it is the share of the forecasted consumption that is serviced neither by the current assets still present in the forecast year, nor by the *new* agent.

5. **then each *new* agent gets a share of N proportional to it's** share of the `production`, $P[\mathcal{A}^{r,i}_{a,s}(y)]$. Then the share of the demand for new agent i is:

$$N^{i,r}_{c,s,t}(y) = N^r_{c,s}\frac{\sum_\iota P[\mathcal{A}^{r,i}_{s,t,\iota}(y)]}{\sum_{i,t,\iota} P[\mathcal{A}^{r,i}_{s,t,\iota}(y)]}$$

6. **similarly, each *retrofit* agent gets a share of N proportional to it's** share of the `decommissioning demand`, $D^{r,i}_{t,c}$. Then the share of the demand for retrofit agent i is:

$$R^{i,r}_{c,s,t}(y) = R^r_{c,s}\frac{\sum_\iota \mathcal{D}^{i,r}_{t,c,\iota}(y)}{\sum_{i,t,\iota} \mathcal{D}^{i,r}_{t,c,\iota}(y)}$$

Note that tin the last two steps, the assets owned by the agent are aggregated over the installation year. The effect is that the demand serviced by agents is disaggregated over each technology, rather than not over each *model* of each technology.

See also:

indices, *Quantities*, Agent investments, `decommissioning_demand()`, `maximum_production()`

`muse.demand_share.`**`register_demand_share`**(*function: Callable[[Sequence[muse.agents.agent.AbstractAgent], xarray.core.dataset.Dataset, xarray.core.dataset.Dataset, Any], xarray.core.dataarray.DataArray]*)

Decorator to register a function as a demand share calculation.

`muse.demand_share.`**`unmet_demand`**(*market: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *production: Union[str, Mapping, Callable] = 'maximum_production'*)

Share of the demand that cannot be serviced by the existing assets.

$$U[\mathcal{M}, \mathcal{A}] = \max(\mathcal{C} - P[\mathcal{M}, \mathcal{A}], 0)$$

$\max$ operates element-wise, and indices have been dropped for simplicity. The resulting expression has the same indices as the consumption $\mathcal{C}^r_{c,s}$.

P is any function registered with `@register_production`.

`muse.demand_share.`**`unmet_forecasted_demand`**(*agents: Sequence[muse.agents.agent.AbstractAgent]*, *market: xarray.core.dataset.Dataset*, *technologies: xarray.core.dataset.Dataset*, *current_year: Optional[int] = None*, *production: Union[str, Mapping, Callable] = 'maximum_production'*, *forecast: int = 5*) → xarray.core.dataarray.DataArray

Forecast demand that cannot be serviced by non-decommissioned current assets.

8.3.6 Constraints:

Investment constraints.

Constraints on investements ensure that investements match some given criteria. For instance, the constraints could ensure that only so much of a new asset can be built every year.

Functions to compute constraints should be registered via the decorator `register_constraints()`. This registration step makes it possible for constraints to be declared in the TOML file.

Generally, LP solvers accept linear constraint defined as:

$$Ax \leq b$$

with A a matrix, x the decision variables, and b a vector. However, these quantities are dimensionless. They do no have timeslices, assets, or replacement technologies, or any other dimensions that users have set-up in their model. The crux is to translates from MUSE's data-structures to a consistent dimensionless format.

In MUSE, users can register constraints functions that return fully dimensional quantities. The matrix operator is split over the capacity decision variables and the production decision variables:

$$A_c. * x_c + A_p. * x_p \leq b$$

The operator $.*$ means the standard elementwise multiplication of xarray, including automatic broadcasting (adding missing dimensions by repeating the smaller matrix along the missing dimension). Constraint functions return the three quantities A_c, A_p, and b. These three quantities will often not have the same dimension. E.g. one might include timeslices where another might not. The transformation from A_c, A_p, b to A and b happens as described below.

- b remains the same. It defines the rows of A.
- x_c and x_p are concatenated one on top of the other and define the columns of A.
- A is split into a left submatrix for capacities and a right submatrix for production, following the concatenation of x_c and x_p
- Any dimension in $A_c. * x_c$ ($A_p. * x_p$) that is also in b defines diagonal entries into the left (right) submatrix of A.
- Any dimension in $A_c. * x_c$ ($A_p. * x_b$) and missing from b is reduce by summation over a row in the left (right) submatrix of A. In other words, those dimension do become part of a standard tensor reduction or matrix multiplication.

There are two additional rules. However, they are likely to be the result of an inefficient defininition of A_c, A_p and b.

- Any dimension in A_c (A_b) that is neither in b nor in x_c (x_p) is reduced by summation before consideration for the elementwise multiplication. For instance, if d is such a dimension, present only in A_c, then the problem becomes $(\sum_d A_c). * x_c + A_p. * x_p \leq b$.
- Any dimension missing from $A_c. * x_c$ ($A_p. * x_p$) and present in b is added by repeating the resulting row in A.

Constraints are registered using the decorator `register_constraints()`. The decorated functions must follow the following signature:

```
@register_constraints
def constraints(
    demand: xr.DataArray,
    assets: xr.Dataset,
    search_space: xr.DataArray,
    market: xr.Dataset,
    technologies: xr.Dataset,
    year: Optional[int] = None,
    **kwargs,
) -> Constraint:
    pass
```

demand: The demand for the sectors products. In practice it is a demand share obtained in `demand_share`. It is a data-array with dimensions including *asset*, *commodity*, *timeslice*.

assets: The capacity of the assets owned by the agent.

search_space: A matrix *asset* vs *replacement* technology defining which replacement technologies will be considered for each existing asset.

market: The market as obtained from the MCA.

technologies: Technodata characterizing the competing technologies.

year: current year.

****kwargs:** Any other parameter.

class muse.constraints.**ScipyAdapter**(*c: numpy.ndarray, to_muse: Callable[[numpy.ndarray], xarray.core.dataset.Dataset], bounds: Tuple[Optional[float], Optional[float]] = 0, inf, A_ub: Optional[numpy.ndarray] = None, b_ub: Optional[numpy.ndarray] = None, A_eq: Optional[numpy.ndarray] = None, b_eq: Optional[numpy.ndarray] = None*)

Creates the input for the scipy solvers.

Example

Lets give a fist simple example. The constraint `max_capacity_expansion()` limits how much each capacity can be expanded in a given year.

```
>>> from muse import examples
>>> from muse.quantities import maximum_production
>>> from muse.timeslices import convert_timeslice
>>> from muse import constraints as cs
>>> res = examples.sector("residential", model="medium")
>>> market = examples.residential_market("medium")
>>> search = examples.search_space("residential", model="medium")
>>> assets = next(a.assets for a in res.agents if a.category == "retrofit")
>>> market_demand =  0.8 * maximum_production(
...     res.technologies.interp(year=2025),
...     convert_timeslice(
...         assets.capacity.sel(year=2025).groupby("technology").sum("asset"),
...         market.timeslice,
...     ),
... ).rename(technology="asset")
>>> costs = search * np.arange(np.prod(search.shape)).reshape(search.shape)
>>> constraint = cs.max_capacity_expansion(
...     market_demand, assets, search, market, res.technologies,
... )
```

The constraint acts over capacity decision variables only:

```
>>> assert constraint.production.data == np.array(0)
>>> assert len(constraint.production.dims) == 0
```

It is an upper bound for a straightforward sum over the capacities for a given technology. The matrix operator is simply the identity:

```
>>> assert constraint.capacity.data == np.array(1)
>>> assert len(constraint.capacity.dims) == 0
```

And the upperbound is exanded over the replacement technologies, but not over the assets. Hence the assets will be summed over in the final constraint:

```
>>> assert (constraint.b.data == np.array([500.0, 55.0, 55.0, 500.0 ])).all()
>>> assert set(constraint.b.dims) == {"replacement"}
>>> assert constraint.kind == cs.ConstraintKind.UPPER_BOUND
```

As shown above, it does not bind the production decision variables. Hence, production is zero. The matrix operator for the capacity is simply the identity. Hence it can be inputed as the dimensionless scalar 1. The upper bound is simply the maximum for replacement technology (and region, if that particular dimension exists in the problem).

The lp problem then becomes:

```
>>> technologies = res.technologies.interp(year=market.year.min() + 5)
>>> inputs = cs.ScipyAdapter.factory(
...     technologies, costs, market.timeslice, constraint
... )
```

The decision variables are always constrained between zero and infinity:

```
>>> assert inputs.bounds == (0, np.inf)
```

The problem is an upper-bound one. There are no equality constraints:

```
>>> assert inputs.A_eq is None
>>> assert inputs.b_eq is None
```

The upper bound matrix and vector, and the costs are consistent in their dimensions:

```
>>> assert inputs.c.ndim == 1
>>> assert inputs.b_ub.ndim == 1
>>> assert inputs.A_ub.ndim == 2
>>> assert inputs.b_ub.size == inputs.A_ub.shape[0]
>>> assert inputs.c.size == inputs.A_ub.shape[1]
>>> assert inputs.c.ndim == 1
```

In practice, `lp_costs()` helps us define the decision variables (and `c`). We can verify that the sizes are consistent:

```
>>> lpcosts = cs.lp_costs(technologies, costs, market.timeslice)
>>> capsize = lpcosts.capacity.size
>>> prodsize = lpcosts.production.size
>>> assert inputs.c.size == capsize + prodsize
```

The upper bound itself is over each replacement technology:

```
>>> assert inputs.b_ub.size == lpcosts.replacement.size
```

The production decision variables are not involved:

```
>>> from pytest import approx
>>> assert inputs.A_ub[:, capsize:] == approx(0)
```

The matrix for the capacity decision variables is a sum over assets for a given replacement technology. Hence, each row is constituted of zeros and ones and sums to the number of assets:

```
>>> assert inputs.A_ub[:, :capsize].sum(axis=1) == approx(lpcosts.asset.size)
>>> assert set(inputs.A_ub[:, :capsize].flatten()) == {0.0, 1.0}
```

`muse.constraints.`**`demand`**(*demand: xarray.core.dataarray.DataArray*, *assets: xarray.core.dataset.Dataset*, *search_space: xarray.core.dataarray.DataArray*, *market: xarray.core.dataset.Dataset*, *technologies: xarray.core.dataset.Dataset*, *year: Optional[int] = None*, *forecast: int = 5*, *interpolation: str = 'linear'*) → xarray.core.dataset.Dataset

Constraints production to meet demand.

`muse.constraints.`**`factory`**(*settings: Optional[Union[str, Mapping, Sequence[str], Sequence[Union[str, Mapping]]]] = None*) → Callable

Creates a list of constraints from standard settings.

The standard settings can be a string naming the constraint, a dictionary including at least "name", or a list of strings and dictionaries.

`muse.constraints.`**`lp_constraint`**(*constraint: xarray.core.dataset.Dataset*, *lpcosts: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Transforms the constraint to LP data.

The goal is to create from `lpcosts.capacity`, `constraint.capacity`, and `constraint.b` a 2d-matrix `constraint` vs `decision variables`.

1. **The dimensions of `constraint.b` are the constraint dimensions. They are** renamed `"c(xxx)"`.
2. **The dimensions of `lpcosts` are the decision-variable dimensions. They are** renamed `"d(xxx)"`.
3. **`set(b.dims).intersection(lpcosts.xxx.dims)` are diagonal** in constraint dimensions and decision variables dimension, with `xxx` the capacity or the production
4. **`set(constraint.xxx.dims) - set(lpcosts.xxx.dims) - set(b.dims)` are reduced by** summation, with `xxx` the capacity or the production
5. **`set(lpcosts.xxx.dims) - set(constraint.xxx.dims) - set(b.dims)` are added for** expansion, with `xxx` the capacity or the production

See `muse.constraints.lp_constraint_matrix()` for a more detailed explanation of the transformations applied here.

`muse.constraints.`**`lp_constraint_matrix`**(*b: xarray.core.dataarray.DataArray*, *constraint: xarray.core.dataarray.DataArray*, *lpcosts: xarray.core.dataarray.DataArray*)

Transforms one constraint block into an lp matrix.

The goal is to create from `lpcosts`, `constraint`, and `b` a 2d-matrix of constraints vs decision variables.

1. **The dimensions of `b` are the constraint dimensions. They are renamed** `"c(xxx)"`.
2. **The dimensions of `lpcosts` are the decision-variable dimensions. They are** renamed `"d(xxx)"`.
3. **`set(b.dims).intersection(lpcosts.dims)` are diagonal** in constraint dimensions and decision variables dimension
4. **`set(constraint.dims) - set(lpcosts.dims) - set(b.dims)` are reduced by** summation
5. **`set(lpcosts.dims) - set(constraint.dims) - set(b.dims)` are added for** expansion
6. **`set(b.dims) - set(constraint.dims) - set(lpcosts.dims)` are added for** expansion. Such dimensions only make sense if they consist of one point.

The result is the constraint matrix, expanded, reduced and diagonalized for the conditions above.

Example:

Lets first setup a constraint and a cost matrix:

```
>>> from muse import examples
>>> from muse import constraints as cs
>>> res = examples.sector("residential", model="medium")
>>> technologies = res.technologies
>>> market = examples.residential_market("medium")
>>> search = examples.search_space("residential", model="medium")
>>> assets = next(a.assets for a in res.agents if a.category ==
↪"retrofit")
>>> demand = None # not used in max production
>>> constraint = cs.max_production(demand, assets, search, market,␣
↪technologies)
>>> lpcosts = cs.lp_costs(
...     (
...         technologies
...         .interp(year=market.year.min() + 5)
...         .drop_vars("year")
...         .sel(region=assets.region)
...     ),
...     costs=search * np.arange(np.prod(search.shape)).
↪reshape(search.shape),
...     timeslices=market.timeslice,
... )
```

For a simple example, we can first check the case where b is scalar. The result ought to be a single row of a matrix, or a vector with only decision variables:

```
>>> from pytest import approx
>>> result = cs.lp_constraint_matrix(
...     xr.DataArray(1), constraint.capacity, lpcosts.capacity
... )
>>> assert result.values == approx(-1)
>>> assert set(result.dims) == {f"d({x})" for x in lpcosts.
↪capacity.dims}
>>> result = cs.lp_constraint_matrix(
...     xr.DataArray(1), constraint.production, lpcosts.production
... )
>>> assert set(result.dims) == {f"d({x})" for x in lpcosts.
↪production.dims}
>>> assert result.values == approx(1)
```

As expected, the capacity vector is 1, whereas the production vector is -1. These are the values the `max_production()` is set up to create.

Now, let's check the case where `b` is the one from the `max_production()` constraint. In that case, all the dimensions should end up as constraint dimensions: the production for each timeslice, region, asset, and replacement technology should not outstrip the capacity assigned for the asset and replacement technology.

```
>>> result = cs.lp_constraint_matrix(
...     constraint.b, constraint.capacity, lpcosts.capacity
... )
>>> decision_dims = {f"d({x})" for x in lpcosts.capacity.dims}
>>> constraint_dims = {
...     f"c({x})" for x in set(lpcosts.production.dims).
↪union(constraint.b.dims)
... }
```

(continues on next page)

(continued from previous page)

```
>>> assert set(result.dims) == decision_dims.union(constraint_dims)
```

The `max_production()` constraint on the production side is the identy matrix with a factor -1. We can easily check this by stacking the decision and constraint dimensions in the result:

```
>>> result = cs.lp_constraint_matrix(
...     constraint.b, constraint.production, lpcosts.production
... )
>>> decision_dims = {f"d({x})" for x in lpcosts.production.dims}
>>> assert set(result.dims) == decision_dims.union(constraint_dims)
>>> stacked = result.stack(d=sorted(decision_dims),␣
↪c=sorted(constraint_dims))
>>> assert stacked.shape[0] == stacked.shape[1]
>>> assert stacked.values == approx(np.eye(stacked.shape[0]))
```

muse.constraints.**lp_costs**(*technologies: xarray.core.dataset.Dataset*, *costs: xarray.core.dataarray.DataArray*, *timeslices: xarray.core.dataarray.DataArray*) → xarray.core.dataset.Dataset

Creates costs for solving with scipy's LP solver.

Example

We can now construct example inputs to the funtion from the sample model. The costs will be a matrix where each assets has a candidate replacement technology.

```
>>> from muse import examples
>>> technologies = examples.technodata("residential", model="medium")
>>> search_space = examples.search_space("residential", model="medium")
>>> timeslices = examples.sector("residential", model="medium").timeslices
>>> costs = (
...     search_space
...     * np.arange(np.prod(search_space.shape)).reshape(search_space.shape)
... )
```

The function returns the LP vector split along capacity and production variables.

```
>>> from muse.constraints import lp_costs
>>> lpcosts = lp_costs(
...     technologies.sel(year=2020, region="R1"), costs, timeslices
... )
>>> assert "capacity" in lpcosts.data_vars
>>> assert "production" in lpcosts.data_vars
```

The capacity costs correspond exactly to the input costs:

```
>>> assert (costs == lpcosts.capacity).all()
```

The production is zero in this context. It does not enter the cost function of the LP problem:

```
>>> assert (lpcosts.production == 0).all()
```

They should correspond to a data-array with dimensions `(asset, replacement)` (and possibly `region` as well).

```
>>> lpcosts.capacity.dims
('asset', 'replacement')
```

The production costs are zero by default. However, the production expands over not only the dimensions of the capacity, but also the `timeslice` during which production occurs and the `commodity` produced.

```
>>> lpcosts.production.dims
('timeslice', 'asset', 'replacement', 'commodity')
```

`muse.constraints.`**`max_capacity_expansion`**(*demand: xarray.core.dataarray.DataArray*, *assets: xarray.core.dataset.Dataset*, *search_space: xarray.core.dataarray.DataArray*, *market: xarray.core.dataset.Dataset*, *technologies: xarray.core.dataset.Dataset*, *year: Optional[int] = None*, *forecast: Optional[int] = None*, *interpolation: str = 'linear'*) → xarray.core.dataset.Dataset

Max-capacity addition, max-capacity growth, and capacity limits constraints.

Limits by how much the capacity of each technology owned by an agent can grow in a given year. This is a constraint on the agent's ability to invest in a technology.

Let $L_t^r(y)$ be the total capacity limit for a given year, technology, and region. $G_t^r(y)$ is the maximum growth. And $W_t^r(y)$ is the maximum additional capacity. $y = y_0$ is the current year and $y = y_1$ is the year marking the end of the investment period.

Let $\mathcal{A}^{i,r}_{t,\iota}(y)$ be the current assets, before invesment, and let $\Delta\mathcal{A}^{i,r}_t$ be the future investements. The the constraint on agent i are given as:

$$
\begin{aligned}
L_t^r(y_0) - \sum_\iota \mathcal{A}^{i,r}_{t,\iota}(y_1) &\geq \Delta\mathcal{A}^{i,r}_t \\
(y_1 - y_0 + 1)G_t^r(y_0)\sum_\iota \mathcal{A}^{i,r}_{t,\iota}(y_0) - \sum_\iota \mathcal{A}^{i,r}_{t,\iota}(y_1) &\geq \Delta\mathcal{A}^{i,r}_t \\
(y_1 - y_0)W_t^r(y_0) &\geq \Delta\mathcal{A}^{i,r}_t
\end{aligned}
$$

The three constraints are combined into a single one which is returned as the maximum capacity expansion, $\Gamma^{r,i}_t$. The maximum capacity expansion cannot impose negative investments: Maximum capacity addition:

$$\Gamma^{r,i}_t \geq 0$$

`muse.constraints.`**`max_production`**(*demand: xarray.core.dataarray.DataArray*, *assets: xarray.core.dataset.Dataset*, *search_space: xarray.core.dataarray.DataArray*, *market: xarray.core.dataset.Dataset*, *technologies: xarray.core.dataset.Dataset*, *year: Optional[int] = None*) → xarray.core.dataset.Dataset

Constructs constraint between capacity and maximum production.

Constrains the production decision variable by the maximum production for a given capacity.

`muse.constraints.register_constraints`(*function: Optional[Callable[[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, xarray.core.dataset.Dataset, Any], Optional[xarray.core.dataset.Dataset]]]*) → Callable[[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, xarray.core.dataset.Dataset, Any], Optional[xarray.core.dataset.Dataset]]

Registers a constraint with MUSE.

See `muse.constraints`.

`muse.constraints.search_space`(*demand: xarray.core.dataarray.DataArray, assets: xarray.core.dataset.Dataset, search_space: xarray.core.dataarray.DataArray, market: xarray.core.dataset.Dataset, technologies: xarray.core.dataset.Dataset, year: Optional[int] = None, forecast: int = 5*) → Optional[xarray.core.dataset.Dataset]

Removes disabled technologies.

8.3.7 Initial and Final Asset Transforms

Pre and post hooks on agents.

`muse.hooks.asset_merge_factory`(*settings: Union[str, Mapping] = 'new'*) → Callable

Returns a function for merging new investments into assets.

Available merging functions should be registered with `@register_final_asset_transform`.

`muse.hooks.clean`(*agent: muse.agents.agent.Agent, assets: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Removes empty assets.

`muse.hooks.housekeeping_factory`(*settings: Union[str, Mapping] = 'noop'*) → Callable

Returns a function for performing initial housekeeping.

For instance, remove technologies with no capacity now or in the future. Available housekeeping functions should be registered with `@register_initial_asset_transform`.

`muse.hooks.merge_assets`(*old_assets: xarray.core.dataset.Dataset, new_assets: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Adds new assets to old along asset dimension.

New assets are assumed to be unequivalent to any old_assets. Indeed, it is expected that the asset dimension does not have coordinates (i.e. it is a combination of coordinates, such as technology and installation year).

After merging the new assets, quantities are back-filled along the year dimension. Further missing values (i.e. future years the old_assets did not take into account) are set to zero.

`muse.hooks.new_assets_only`(*old_assets: xarray.core.dataset.Dataset, new_assets: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Returns newly invested assets and ignores old assets.

`muse.hooks.noop`(*agent: muse.agents.agent.Agent, assets: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Return assets as they are.

`muse.hooks.`**`old_assets_only`**(*old_assets: xarray.core.dataset.Dataset*, *new_assets: xarray.core.dataset.Dataset*) → xarray.core.dataset.Dataset

Returns old assets and ignores newly invested assets.

`muse.hooks.`**`register_final_asset_transform`**(*function: Callable[[xarray.core.dataset.Dataset, xarray.core.dataset.Dataset], xarray.core.dataset.Dataset]*) → Callable

Decorator to register a function to merge new investments into current assets.

The transform is applied a the very end of the agent iteration. It can be any function which takes as input the current set of assets, the new assets, and any number of keyword arguments. The function must return a "merge" of the two assets.

For instance, the new assets could completely replace the old assets (`new_assets_only()`), or they could be summed to the old assets (`merge_assets()`).

`muse.hooks.`**`register_initial_asset_transform`**(*function: Callable[[muse.agents.agent.Agent, xarray.core.dataset.Dataset], xarray.core.dataset.Dataset]*) → Callable

Decorator to register a function for cleaning or transforming assets.

The transformation is applied at the start of each iteration. It any function which take an agent and assets as input and any number of keyword arguments, and returns the transformed assets. The agent should not be modified. It is only there to query the current year, the region, etc.

8.4 Reading the inputs

Ensemble of functions to read MUSE data.

`muse.readers.toml.`**`read_settings`**(*settings_file: Union[str, pathlib.Path, IO[str], Mapping]*, *path: Optional[Union[str, pathlib.Path]] = None*) → Any

Loads the input settings for any MUSE simulation.

Loads a MUSE settings file. This must be a TOML formatted file. Missing settings are loaded from the DEFAULT_SETTINGS. Custom pythom modules, if present, are loaded and checks are run to validate the settings and ensure that they are compatible with a MUSE simulation.

Arguments: settings_file: A string or a Path to the settings file

Returns A dictionary with the settings

Ensemble of functions to read MUSE data.

`muse.readers.csv.`**`read_attribute_table`**(*path: Union[str, pathlib.Path]*) → xarray.core.dataarray.DataArray

Read a standard MUSE csv file for price projections.

`muse.readers.csv.`**`read_csv_agent_parameters`**(*filename*) → List

Reads standard MUSE agent-declaration csv-files.

Returns a list of dictionaries, where each dictionary can be used to instantiate an agent in `muse.agents.factories.factory()`.

`muse.readers.csv.`**`read_csv_outputs`**(*paths: Union[str, pathlib.Path, Sequence[Union[str, pathlib.Path]]]*, *columns: str = 'commodity'*, *indices: Sequence[str] = 'RegionName', 'ProcessName', 'Timeslice'*, *drop: Sequence[str] = 'Unnamed: 0'*) → xarray.core.dataset.Dataset

Read standard MUSE output files for consumption or supply.

`muse.readers.csv.`**`read_csv_timeslices`**(*path: Union[str, pathlib.Path]*, ***kwargs*) → xarray.core.dataarray.DataArray

Reads timeslice information from input.

`muse.readers.csv.`**`read_global_commodities`**(*path: Union[str, pathlib.Path]*) → xarray.core.dataset.Dataset

Reads commodities information from input.

`muse.readers.csv.`**`read_initial_assets`**(*filename: Union[str, pathlib.Path]*) → xarray.core.dataarray.DataArray

Reads and formats data about initial capacity into a dataframe.

`muse.readers.csv.`**`read_initial_market`**(*projections: Union[xarray.core.dataarray.DataArray, pathlib.Path, str]*, *base_year_import: Optional[Union[str, pathlib.Path, xarray.core.dataarray.DataArray]] = None*, *base_year_export: Optional[Union[str, pathlib.Path, xarray.core.dataarray.DataArray]] = None*, *timeslices: Optional[xarray.core.dataarray.DataArray] = None*) → xarray.core.dataset.Dataset

Read projections, import and export csv files.

`muse.readers.csv.`**`read_io_technodata`**(*filename: Union[str, pathlib.Path]*) → xarray.core.dataset.Dataset

Reads process inputs or ouputs.

There are four axes: (technology, region, year, commodity)

`muse.readers.csv.`**`read_macro_drivers`**(*path: Union[str, pathlib.Path]*) → xarray.core.dataset.Dataset

Reads a standard MUSE csv file for macro drivers.

`muse.readers.csv.`**`read_regression_parameters`**(*path: Union[str, pathlib.Path]*) → xarray.core.dataset.Dataset

Reads the regression parameters from a standard MUSE csv file.

`muse.readers.csv.`**`read_technodictionary`**(*filename: Union[str, pathlib.Path]*) → xarray.core.dataset.Dataset

Reads and formats technodata into a dataset.

There are three axes: technologies, regions, and year.

`muse.readers.csv.`**`read_technologies`**(*technodata_path_or_sector: Optional[Union[str, pathlib.Path]] = None*, *comm_out_path: Optional[Union[str, pathlib.Path]] = None*, *comm_in_path: Optional[Union[str, pathlib.Path]] = None*, *commodities: Optional[Union[str, pathlib.Path, xarray.core.dataset.Dataset]] = None*, *sectors_directory: Union[str, pathlib.Path] = PosixPath('/Users/alexkell/Documents/SGI/2-documentation/StarMuse/docs/data')*) → xarray.core.dataset.Dataset

Reads data characterising technologies from files.

Parameters

- **`technodata_path_or_sector`** – If *comm_out_path* and *comm_in_path* are not given, then this argument refers to the name of the sector. The three paths are then determined using standard locations and name. Specifically, thechnodata looks for a "technodataSECTORNAME.csv" file in the standard location for that sector. However, if *comm_out_path* and *comm_in_path* are given, then this should be the path to the the technodata file.

- **comm_out_path** – If given, then refers to the path of the file specifying output commmodities. If not given, then defaults to "commOUTtechnodataSECTORNAME.csv" in the relevant sector directory.
- **comm_in_path** – If given, then refers to the path of the file specifying input commmodities. If not given, then defaults to "commINtechnodataSECTORNAME.csv" in the relevant sector directory.
- **commodities** – Optional. If commodities is given, it should point to a global commodities file, or a dataset akin to reading such a file with *read_global_commodities*. In either case, the information pertaining to commodities will be added to the technologies dataset.
- **sectors_directory** – Optional. If *paths_or_sector* is a string indicating the name of the sector, then this is a path to a directory where standard input files are contained.

Returns A dataset with all the characteristics of the technologies.

`muse.readers.csv.`**`read_timeslice_shares`**(*path: Union[str, pathlib.Path] = PosixPath('/Users/alexkell/Documents/SGI/2-documentation/StarMuse/docs/data'), sector: Optional[str] = None, timeslice: Union[str, pathlib.Path, xarray.core.dataarray.DataArray] = 'Timeslices{sector}.csv'*) → xarray.core.dataset.Dataset

Reads sliceshare information into a xr.Dataset.

Additionaly, this function will try and recover the timeslice multi- index from a import file "Timeslices{sector}.csv" in the same directory as the timeslice shares. Pass *None* if this behaviour is not required.

`muse.decorators.`**`SETTINGS_CHECKS: Mapping[str, Callable[[dict], None]] = {'check_budget_para`**

Dictionary of settings checks.

`muse.decorators.`**`SETTINGS_CHECKS_SIGNATURE`**

settings checks signature.

alias of Callable[[dict], None]

`muse.decorators.`**`register_settings_check`**(*function: Callable[[dict], None]*)

Decorator to register a function as a settings check.

Registers a function as a settings check so that it can be applied easily when validating the MUSE input settings.

There is no restriction on the function name, although is should be in lower_snake_case, as it is a python function.

8.5 Writing Outputs

`muse.outputs.`**`register_output_quantity`**(*function: Callable[[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, Any], Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray]] = None*) → Callable

Registers a function to compute an output quantity.

`muse.outputs.`**`register_output_sink`**(*function: Callable[[Union[xarray.core.dataarray.DataArray, pandas.core.frame.DataFrame], int, Any], Optional[str]] = None*) → Callable

Registers a function to save quantities.

8.5.1 Sinks

Sinks where output quantities can be stored.

Sinks take as argument a DataArray and store it somewhere. Additionally they take a dictionary as argument. This dictionary will always contains the items ('quantity', 'sector', 'year') referring to the name of the quantity, the name of the calling sector, the current year. They may contain additional parameters which depend on the actual sink, such as 'filename'.

Optionally, a description of the storage (filename, etc) can be returned.

The signature of a sink is:

```
@register_output_sink(name="netcfd")
def to_netcfd(quantity: DataArray, config: Mapping) -> Optional[Text]:
    pass
```

exception `muse.outputs.sinks.`**`FiniteResourceException`**
Raised when a finite resource is exceeded.

`muse.outputs.sinks.`**`OUTPUT_SINKS: MutableMapping[str, Union[Callable[[Union[xarray.core.data`**
Stores a quantity somewhere.

`muse.outputs.sinks.`**`OUTPUT_SINK_SIGNATURE`**
Signature of functions used to save quantities.

alias of Callable[[Union[xarray.core.dataarray.DataArray, pandas.core.frame.DataFrame], int, Any], Optional[str]]

class `muse.outputs.sinks.`**`YearlyAggregate`**(*final_sink: Optional[MutableMapping[str, Any]] = None*, *sector: str = ''*, *axis='year'*, ***kwargs*)
Incrementally aggregates data from year to year.

`muse.outputs.sinks.`**`register_output_sink`**(*function: Callable[[Union[xarray.core.dataarray.DataArray, pandas.core.frame.DataFrame], int, Any], Optional[str]] = None*) → Callable
Registers a function to save quantities.

`muse.outputs.sinks.`**`sink_to_file`**(*suffix: str*)
Simplifies sinks to files.

The decorator takes care of figuring out the path to the file, as well as trims the configuration dictionary to include only parameters for the sink itself. The decorated function returns the path to the output file.

`muse.outputs.sinks.`**`to_csv`**(*quantity: Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray]*, *filename: str*, ***params*) → None
Saves data array to csv format, using pandas.to_csv.

Parameters

- **`quantity`** – The data to be saved
- **`filename`** – File to which the data should be saved
- **`params`** – A configuration dictionary accepting any argument to *pandas.to_csv*

`muse.outputs.sinks.`**`to_excel`**(*quantity: Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray]*, *filename: str*, ***params*) → None
Saves data array to csv format, using pandas.to_excel.

Parameters

- **`quantity`** – The data to be saved

- **filename** – File to which the data should be saved
- **params** – A configuration dictionary accepting any argument to *pandas.to_excel*

`muse.outputs.sinks.`**`to_netcdf`**(*quantity: Union[xarray.core.dataarray.DataArray, pandas.core.frame.DataFrame], filename: str, **params*) → None

Saves data array to csv format, using xarray.to_netcdf.

Parameters

- **quantity** – The data to be saved
- **filename** – File to which the data should be saved
- **params** – A configuration dictionary accepting any argument to *xarray.to_netcdf*

8.5.2 Sectorial Outputs

Output quantities.

Functions that compute sectorial quantities for post-simulation analysis should all follow the same signature:

```
@register_output_quantity
def quantity(
    capacity: xr.DataArray,
    market: xr.Dataset,
    technologies: xr.Dataset
) -> Union[xr.DataArray, DataFrame]:
    pass
```

They take as input the current capacity profile, aggregated across a sectoar, a dataset containing market-related quantities, and a dataset characterizing the technologies in the market. It returns a single xr.DataArray object.

The function should never modify it's arguments.

`muse.outputs.sector.`**`OUTPUTS_PARAMETERS`**

Acceptable Datastructures for outputs parameters

alias of Union[str, Mapping]

`muse.outputs.sector.`**`OUTPUT_QUANTITIES`**

Quantity for post-simulation analysis.

`muse.outputs.sector.`**`OUTPUT_QUANTITY_SIGNATURE`**

Signature of functions computing quantities for later analysis.

alias of Callable[[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, Any], Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray]]

`muse.outputs.sector.`**`capacity`**(*market: xarray.core.dataset.Dataset, capacity: xarray.core.dataarray.DataArray, technologies: xarray.core.dataset.Dataset, rounding: int = 4*) → pandas.core.frame.DataFrame

Current capacity.

`muse.outputs.sector.`**`consumption`**(*market: xarray.core.dataset.Dataset, capacity: xarray.core.dataarray.DataArray, technologies: xarray.core.dataset.Dataset, sum_over: Optional[List[str]] = None, drop: Optional[List[str]] = None, rounding: int = 4*) → xarray.core.dataarray.DataArray

Current consumption.

`muse.outputs.sector.costs`(*market: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *sum_over: Optional[List[str]] = None*, *drop: Optional[List[str]] = None*, *rounding: int = 4*) → xarray.core.dataarray.DataArray

Current costs.

`muse.outputs.sector.factory`(**parameters: Union[str, Mapping]*, *sector_name: str = 'default'*) → Callable[[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], List[Any]]

Creates outputs functions for post-mortem analysis.

Each parameter is a dictionary containing the following:

- quantity (mandatory): name of the quantity to output. Mandatory.
- sink (optional): name of the storage procedure, e.g. the file format or database format. When it cannot be guessed from *filename*, it defaults to "csv".
- filename (optional): path to a directory or a file where to store the quantity. In the latter case, if sink is not given, it will be determined from the file extension. The filename can incorporate markers. By default, it is "{default_output_dir}/{sector}{year}{quantity}{suffix}".
- any other parameter relevant to the sink, e.g. *pandas.to_csv* keyword arguments.

For simplicity, it is also possible to given lone strings as input. They default to *{'quantity': string}* (and the sink will default to "csv").

`muse.outputs.sector.register_output_quantity`(*function: Callable[[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, Any], Union[pandas.core.frame.DataFrame, xarray.core.dataarray.DataArray]] = None*) → Callable

Registers a function to compute an output quantity.

`muse.outputs.sector.supply`(*market: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *sum_over: Optional[List[str]] = None*, *drop: Optional[List[str]] = None*, *rounding: int = 4*) → xarray.core.dataarray.DataArray

Current supply.

8.6 Quantities

Collection of functions to compute model quantities.

This module is meant to collect functions computing quantities of interest to the model, e.g. lcoe, maximum production for a given capacity, etc, especially where these functions are used in different areas of the model.

`muse.quantities.annual_levelized_cost_of_energy`(*prices: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *interpolation: str = 'linear'*, *fill_value: Union[int, str] = 'extrapolate'*, ***filters*) → xarray.core.dataarray.DataArray

Levelized cost of energy (LCOE) of technologies on each given year.

It mostly follows the simplified LCOE given by NREL. However, the units are sometimes different. In the argument description, we use the following:

- [h]: hour
- [y]: year
- [$]: unit of currency
- [E]: unit of energy
- [1]: dimensionless

Parameters

- **prices** – [$/(Eh)] the price of all commodities, including consumables and fuels. This dataarray contains at least timeslice and commodity dimensions.
- **technologies** – Describe the technologies, with at least the following parameters:
  - cap_par: [$/E] overnight capital cost
  - interest_rate: [1]
  - fix_par: [$/(Eh)] fixed costs of operation and maintenance costs
  - var_par: [$/(Eh)] variable costs of operation and maintenance costs
  - **fixed_inputs: [1] == [(Eh)/(Eh)] ratio indicating the amount of commodity** consumed per units of energy created.
  - **fixed_outputs: [1] == [(Eh)/(Eh)] ration indicating the amount of** environmental pollutants produced per units of energy created.
- **interpolation** – interpolation method.
- **fill_value** – Fill value for values outside the extrapolation range.
- ****filters** – Anything by which prices can be filtered.

Returns The lifetime LCOE in [$/(Eh)] for each technology at each timeslice.

muse.quantities.**capacity_in_use**(*production: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *max_dim: Optional[Union[str, Tuple[str]]] = 'commodity'*, ***filters*)

Capacity-in-use for each asset, given production.

Conceptually, this operation is the inverse of *production*.

Parameters

- **production** – Production from each technology of interest.
- **technologies** – xr.Dataset describing the features of the technologies of interests. It should contain *fixed_outputs* and *utilization_factor*. It's shape is matched to *capacity* using *muse.utilities.broadcast_techs*.
- **max_dim** – reduces the given dimensions using *max*. Defaults to "commodity". If None, then no reduction is performed.
- **filters** – keyword arguments are used to filter down the capacity and technologies. Filters not relevant to the quantities of interest, i.e. filters that are not a dimension of *capacity* or *techologies*, are silently ignored.

Returns Capacity-in-use for each technology, whittled down by the filters.

`muse.quantities.`**`consumption`**(*technologies: xarray.core.dataset.Dataset, production: xarray.core.dataarray.DataArray, prices: Optional[xarray.core.dataarray.DataArray] = None, **kwargs*) → xarray.core.dataarray.DataArray

Commodity consumption when fulfilling the whole production.

Currently, the consumption is implemented for commodity_max == +infinity. If prices are not given, then flexible consumption is *not* considered.

`muse.quantities.`**`costed_production`**(*demand: xarray.core.dataset.Dataset, costs: xarray.core.dataarray.DataArray, capacity: xarray.core.dataarray.DataArray, technologies: xarray.core.dataset.Dataset, with_minimum_service: bool = True*) → xarray.core.dataarray.DataArray

Computes production from ranked assets.

The assets are ranked according to their cost. The asset with least cost are allowed to service the demand first, up to the maximum production. By default, the mininum service is applied first.

`muse.quantities.`**`decommissioning_demand`**(*technologies: xarray.core.dataset.Dataset, capacity: xarray.core.dataarray.DataArray, year: Optional[Sequence[int]] = None*) → xarray.core.dataarray.DataArray

Computes demand from process decommissioning.

If *year* is not given, it defaults to all years in capacity. If there are more than two years, then decommissioning is with respect to first (or minimum) year.

Let $M_t^r(y)$ be the retrofit demand, $^{(s)}\mathcal{D}_t^r(y)$ be the decommissioning demand at the level of the sector, and $A^r_{t,\iota}(y)$ be the assets owned by the agent. Then, the decommissioning demand for agent i is :

$$\mathcal{D}^{r,i}_{t,c}(y) = \sum_\iota \alpha^r_{t,\iota}\beta^r_{t,\iota,c} \left(A^{i,r}_{t,\iota}(y) - A^{i,r}_{t,\iota,c}(y+1)\right)$$

given the utilization factor $\alpha_{t,\iota}$ and the fixed output factor $\beta_{t,\iota,c}$.

Furthermore, decommissioning demand is non-zero only for end-use commodities.

ncsearch-nohlsearch).. SeeAlso: indices, *Quantities*, `maximum_production()` `is_enduse()`

`muse.quantities.`**`demand_matched_production`**(*demand: xarray.core.dataarray.DataArray, prices: xarray.core.dataarray.DataArray, capacity: xarray.core.dataarray.DataArray, technologies: xarray.core.dataset.Dataset, **filters*) → xarray.core.dataarray.DataArray

Production matching the input demand.

Parameters

- **`demand`** – demand to match.
- **`prices`** – price from which to compute the annual levelized cost of energy.
- **`capacity`** – capacity from which to obtain the maximum production constraints.
- **`**filters`** – keyword arguments with which to filter the input datasets and data arrays., e.g. region, or year.

`muse.quantities.`**`emission`**(*production: xarray.core.dataarray.DataArray, fixed_outputs: xarray.core.dataarray.DataArray*)

Computes emission from current products.

Emissions are computed as *sum(product) * fixed_outputs*.

Parameters

- **production** – Produced goods. Only those with non-environmental products are used when computing emissions.
- **fixed_outputs** – factor relating total production to emissions. For convenience, this can also be a *technologies* dataset containing *fixed_output*.

Returns A data array containing emissions (and only emissions).

muse.quantities.**gross_margin**(*technologies: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, *prices: xarray.core.dataset.Dataset*) → xarray.core.dataarray.DataArray

profit of increasing the production by one unit.

- energy commodities INPUTS are related to fuel costs
- environmental commodities OUTPUTS are related to environmental costs
- variable costs is given as technodata inputs
- non-environmental commodities OUTPUTS are related to revenues

muse.quantities.**lifetime_levelized_cost_of_energy**(*prices: xarray.core.dataarray.DataArray*, *technologies: xarray.core.dataset.Dataset*, *installation_year: Optional[int] = None*, ***filters*)

Levelized cost of energy (LCOE) of technologies over their lifetime.

It mostly follows the *simplified LCOE* given by NREL. However, the units are sometimes different. In the argument description, we use the following:

- [h]: hour
- [y]: year
- [$]: unit of currency
- [E]: unit of energy
- [1]: dimensionless

Parameters

- **prices** – [$/(Eh)] the price of all commodities, including consumables and fuels. This dataarray contains at least timeslice and commodity dimensions.
- **technologies** – Describe the technologies, with at least the following parameters:
  - technical life: [a] lifetime of each technology
  - cap_par: [$/E] overnight capital cost
  - interest_rate: [1]
  - fix_par: [$/(Eh)] fixed costs of operation and maintenance costs
  - var_par: [$/(Eh)] variable costs of operation and maintenance costs
  - **fixed_inputs: [1] == [(Eh)/(Eh)] ratio indicating the amount of commodity** consumed per units of energy created.
  - **fixed_outputs: [1] == [(Eh)/(Eh)] ration indicating the amount of** environmental pollutants produced per units of energy created.

- **installation_year** – year when the technologies are installed. If not given, it defaults to the first year in *prices*. This should be a single value, there is currently no provision for computing LCOE over different installation years.

Returns The lifetime LCOE in [$/(Eh)] for each technology at each timeslice.

muse.quantities.**maximum_production**(*technologies: xarray.core.dataset.Dataset*, *capacity: xarray.core.dataarray.DataArray*, ***filters*)

Production for a given capacity.

Given a capacity $\mathcal{A}_{t,\iota}^r$, the utilization factor $\alpha_{t,\iota}^r$ and the the fixed outputs of each technology $\beta_{t,\iota,c}^r$, then the result production is:

$$P_{t,\iota}^r = \alpha_{t,\iota}^r \beta_{t,\iota,c}^r \mathcal{A}_{t,\iota}^r$$

The dimensions above are only indicative. The function should work with many different input values, e.g. with capacities expanded over time-slices t or agents i.

Parameters

- **capacity** – Capacity of each technology of interest. In practice, the capacity can refer to asset capacity, the max capacity, or the capacity-in-use.
- **technologies** – xr.Dataset describing the features of the technologies of interests. It should contain *fixed_outputs* and *utilization_factor*. It's shape is matched to *capacity* using *muse.utilities.broadcast_techs*.
- **filters** – keyword arguments are used to filter down the capacity and technologies. Filters not relevant to the quantities of interest, i.e. filters that are not a dimension of *capacity* or *techologies*, are silently ignored.

Returns *capacity * fixed_outputs * utilization_factor*, whittled down according to the filters and the set of technologies in *capacity*.

muse.quantities.**supply**(*capacity: xarray.core.dataarray.DataArray*, *demand: xarray.core.dataarray.DataArray*, *technologies: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]*, *interpolation: str = 'linear'*, *production_method: Optional[Callable] = None*) → xarray.core.dataarray.DataArray

Production and emission for a given capacity servicing a given demand.

Supply includes two components, end-uses outputs and environmental pollutants. The former consists of the demand that the current capacity is capable of servicing. Where there is excess capacity, then service is assigned to each asset a share of the maximum production (e.g. utilization across similar assets is the same in percentage). Then, environmental pollutants are computing as a function of commodity outputs.

Parameters

- **capacity** – number/quantity of assets that can service the demand
- **demand** – amount of each end-use required. The supply of each process will not exceed it's share of the demand.
- **technologies** – factors bindings the capacity of an asset with its production of commodities and environmental pollutants.

Returns A data array where the commodity dimension only contains actual outputs (i.e. no input commodities).

muse.quantities.**supply_cost**(*production: xarray.core.dataarray.DataArray*, *lcoe: xarray.core.dataarray.DataArray*, *asset_dim: Optional[str] = 'asset'*) → xarray.core.dataarray.DataArray

Supply cost given production and the levelized cost of energy.

In practice, the supply cost is the weighted average LCOE over assets (*asset_dim*), where the weights are the production.

Parameters

- **production** – Amount of goods produced. In practice, production can be obtained from the capacity for each asset via the method *muse.quantities.production*.
- **lcoe** – Levelized cost of energy for each good produced. In practice, it can be obtained from market prices via *muse.quantities.annual_levelized_cost_of_energy* or *muse.quantities.lifetime_levelized_cost_of_energy*.
- **asset_dim** – Name of the dimension(s) holding assets, processes or technologies.

8.7 Demand Matching Algorithm

Collection of demand-matching algorithms.

At it's simplest, the demand matching algorithm solves the following problem,

- given a demand for a commodity D_d, with $d \in \mathcal{D}$
- given processes to supply these commodities, with an associated cost per process, $C_{d,i}$, with $i \in \mathcal{I}$

Match demand and supply while minimizing the associated cost.

$$\begin{aligned} \min_X \sum_{d,i} C_{d,i} X_{d,i} \\ X_{d,i} \geq 0 \\ \sum_o X_o \geq D_d \end{aligned}$$

The basic algorithm proceeds as follows:

1. sort all costs $C_{d,i}$ accross both d and i
2. for each cost c_0 in order:
   1. find the set of indices $\mathcal{C} \subseteq \mathcal{D} \cup \mathcal{I}$ for which

      $$\forall (d,i) \in \mathcal{C} \quad C_{d,i} == c_0$$

   2. determine the partial result for the current cost

      $$\forall (d,i) \in \mathcal{C} \quad X_{d,i} = \frac{D_d}{|i \in \mathcal{C}|}$$

      Where $|i \in \mathcal{C}|$ indicates the number of indices i in $\mathcal{C}$.

However, in practice, the problem to solve often contains constraints, e.g. a constraint on production $\sum_d X_{d,i} \leq M_i$. The algorithms in this module try and solve these constrained problems one way or another.

`muse.demand_matching.`**`demand_matching`**(*demand: xarray.core.dataarray.DataArray, cost: xarray.core.dataarray.DataArray, *constraints: xarray.core.dataarray.DataArray, protected_dims: Optional[Set] = None*) → xarray.core.dataarray.DataArray

Demand matching over heterogenous dimensions.

This algorithm enables demand matching while enforcing constraints on how much an asset can produce. Any set of dimensions can be matched. The algorithm is general with respect to the dimensions in demand and cost. It also enforces constraints over sets of indices.

$$\begin{aligned}
\min_X \sum_{d,i} C_{d,i} X_{d,i} \\
X_{d,i} \geq 0 \\
\sum_i X_{d,i} \geq D_d \\
M^{(\alpha)}_{(d,i)\in\mathcal{R}^{(\alpha)}} \geq \sum_{(d,i)\notin\mathcal{R}^{(\alpha)}} X_{d,i}
\end{aligned}$$

Where α is an index running over constraints, $\mathcal{R}^{(\alpha)} \subseteq \mathcal{D} \cup \mathcal{I}$ is a subset of indices.

The algorithm proceeds as described in `muse.demand_matching`. However, an extra step is added to ensure that the solutions falls within the convex-hull formed by the constraints. This projects the current solution onto the constraint. Hence, the solution will depend on the order in which the constraints are given.

1. sort all costs $C_{d,m}$ accross both d and m
2. for each cost c_0 in order:
   1. find the set of indices $\mathcal{C}$

      $$\begin{aligned}
      \mathcal{C} \subseteq \mathcal{D} \cup \mathcal{I} \\
      \forall (d,i) \in \mathcal{C} \quad C_{d,i} == c_0
      \end{aligned}$$

   2. determine an interim partial result for the current cost

      $$\forall (d,i) \in \mathcal{C} \quad \delta X_{d,i} = \frac{1}{|i \in \mathcal{C}|}\left(D_d - \sum_{j\in\mathcal{I}} X_{d,j}\right)$$

      Where $|i \in \mathcal{C}|$ indicates the number of i indices in $\mathcal{C}$. The expression in the parenthesis is the currently unserviced demand.

   3. Loop over each constraint α. Below we drop the index α over constraints for simplicity.
      1. Determine the excess over the constraint:

         $$E_{(d,i)\in\mathcal{R}} = \max\left\{0, \sum_{(d,i)\notin\mathcal{R}} \left(X_{d,i} + \delta X_{d,i}\right) - M_{(d,i)\in\mathcal{R}}\right\}$$

      2. Correct δX as follows:

         $$\begin{aligned}
         \forall (d,i) \in \mathcal{C} \cap \mathcal{R} \quad \delta X\prime_{d,i} = E_{(d,i)} \frac{\delta X_{(d,i)}}{\sum_{(e,j)\in\mathcal{C}\cap\mathcal{R}} \delta X_{(e,j)}} \\
         \forall (d,i) \notin \mathcal{R}, (d,i) \in \mathcal{C} \quad \delta X\prime_{d,i} = 0
         \end{aligned}$$

      3. Set $\delta X = \max(0, \delta X - \delta X\prime)$

A more complex problem would see independant dimensions for each quantity. In that, case we can reduce to the original problem as shown here

$$
\begin{aligned}
C_{d,i,c} &= \min_c C\prime_{d,i,c} \\
D_d &= \sum_{d\prime} D\prime_{d,d\prime} \\
M_r &= \sum_m M\prime_{r,m} \\
X_{d,d\prime,i,m,c} &= (C\prime_{d,i,c} == C_{d,i}) \frac{M\prime_{r,m}}{M_r} \frac{D\prime_{d,d\prime}}{D_d} X_{d,i}
\end{aligned}
$$

A dimension could be shared by all quantities, in which case each point along that dimension is treated as independant.

Similarly, if a dimension is shared only by the demand and a constraint but not by the cost, then the problem can be reduced a set of problems independant along that direction.

Parameters

- **demand** – Demand to match with production. It should have the same physical units as *max_production*.
- **cost** – Cost to minimize while fulfiling the demand.
- ***constraints** – each item is a seperate constraint M_r.

Returns An array with the joint dimensionality of *max_production*, *cost*, and *demand*, containing the supply that fulfills the demand. The units of this supply are the same as *demand* and *max_production*.

8.8 Miscellaneous

8.8.1 Timeslices

Timeslice utility functions.

`muse.timeslices.`**`aggregate_transforms`**(*settings: Optional[Union[Mapping, str]] = None, timeslice: Optional[xarray.core.dataarray.DataArray] = None*) → Dict[Tuple, numpy.ndarray]

Creates dictionay of transforms for aggregate levels.

The transforms are used to create the projectors towards the finest timeslice.

Parameters

- **timeslice** – a `DataArray` with the timeslice dimension.
- **settings** – A dictionary mapping the name of an aggregate with the values it aggregates, or a string that toml will parse as such. If not given, only the unit transforms are returned.

Returns A dictionary of transforms for each possible slice to it's corresponding finest timeslices.

Example

```
>>> toml = """
...     [timeslices]
...     spring.weekday = 5
...     spring.weekend = 2
...     autumn.weekday = 5
...     autumn.weekend = 2
...     winter.weekday = 5
...     winter.weekend = 2
...     summer.weekday = 5
...     summer.weekend = 2
...
...     [timeslices.aggregates]
...     spautumn = ["spring", "autumn"]
...     week = ["weekday", "weekend"]
... """
>>> from muse.timeslices import reference_timeslice, aggregate_transforms
>>> ref = reference_timeslice(toml)
>>> transforms = aggregate_transforms(toml, ref)
>>> transforms[("spring", "weekend")]
array([0, 1, 0, 0, 0, 0, 0, 0])
>>> transforms[("spautumn", "weekday")]
array([1, 0, 1, 0, 0, 0, 0, 0])
>>> transforms[("autumn", "week")].T
array([0, 0, 1, 1, 0, 0, 0, 0])
>>> transforms[("spautumn", "week")].T
array([1, 1, 1, 1, 0, 0, 0, 0])
```

`muse.timeslices.`**`convert_timeslice`**(*x: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], ts: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, pandas.core.indexes.multi.MultiIndex], quantity: Union[muse.timeslices.QuantityType, str] = <QuantityType.EXTENSIVE: 'extensive'>, finest: Optional[xarray.core.dataarray.DataArray] = None, transforms: Optional[Dict[Tuple, numpy.ndarray]] = None*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Adjusts the timeslice of x to match that of ts.

The conversion can be done in on of two ways, depending on whether the quantity is extensive or intensive. See *QuantityType*.

Example

Lets define three timeslices from finest, to fine, to rough:

```
>>> toml = """
...     ["timeslices"]
...     winter.weekday.day = 5
...     winter.weekday.night = 5
...     winter.weekend.day = 2
...     winter.weekend.night = 2
...     summer.weekday.day = 5
```

(continues on next page)

(continued from previous page)

```
...     summer.weekday.night = 5
...     summer.weekend.day = 2
...     summer.weekend.night = 2
...     level_names = ["semester", "week", "day"]
...     aggregates.allday = ["day", "night"]
...     aggregates.allweek = ["weekend", "weekday"]
...     aggregates.allyear = ["winter", "summer"]
... """
>>> from muse.timeslices import setup_module
>>> from muse.readers import read_timeslices
>>> setup_module(toml)
>>> finest_ts = read_timeslices()
>>> fine_ts = read_timeslices(dict(week=["allweek"]))
>>> rough_ts = read_timeslices(dict(semester=["allyear"], day=["allday"]))
```

Lets also define to other data-arrays to demonstrate how we can play with dimensions:

```
>>> from numpy import array
>>> x = DataArray(
...     [5, 2, 3],
...     coords={'a': array([1, 2, 3], dtype="int64")},
...     dims='a'
... )
>>> y = DataArray([1, 1, 2], coords={'b': ["d", "e", "f"]}, dims='b')
```

We can now easily convert arrays with different dimensions. First, lets check conversion from an array with no timeslices:

```
>>> from xarray import ones_like
>>> from muse.timeslices import convert_timeslice, QuantityType
>>> z = convert_timeslice(x, finest_ts, QuantityType.EXTENSIVE)
>>> z.round(6)
<xarray.DataArray (timeslice: 8, a: 3)>
array([[0.892857, 0.357143, 0.535714],
       [0.892857, 0.357143, 0.535714],
       [0.357143, 0.142857, 0.214286],
       [0.357143, 0.142857, 0.214286],
       [0.892857, 0.357143, 0.535714],
       [0.892857, 0.357143, 0.535714],
       [0.357143, 0.142857, 0.214286],
       [0.357143, 0.142857, 0.214286]])
Coordinates:
  * timeslice  (timeslice) MultiIndex
  - semester   (timeslice) object 'winter' 'winter' ... 'summer' 'summer'
  - week       (timeslice) object 'weekday' 'weekday' ... 'weekend' 'weekend'
  - day        (timeslice) object 'day' 'night' 'day' ... 'night' 'day' 'night'
  * a          (a) int64 1 2 3
>>> z.sum("timeslice")
<xarray.DataArray (a: 3)>
array([5., 2., 3.])
Coordinates:
  * a        (a) int64 1 2 3
```

As expected, the sum over timeslices recovers the original array.

In the case of an intensive quantity without a timeslice dimension, the operation does not do anything:

```
>>> convert_timeslice([1, 2], rough_ts, QuantityType.INTENSIVE)
[1, 2]
```

More interesting is the conversion between different timeslices:

```
>>> from xarray import zeros_like
>>> zfine = x + y + zeros_like(fine_ts.timeslice, dtype=int)
>>> zrough = convert_timeslice(zfine, rough_ts)
>>> zrough.round(6)
<xarray.DataArray (timeslice: 2, a: 3, b: 3)>
array([[[17.142857, 17.142857, 20.       ],
        [ 8.571429,  8.571429, 11.428571],
        [11.428571, 11.428571, 14.285714]],

       [[ 6.857143,  6.857143,  8.       ],
        [ 3.428571,  3.428571,  4.571429],
        [ 4.571429,  4.571429,  5.714286]]])
Coordinates:
  * timeslice  (timeslice) MultiIndex
  - semester   (timeslice) object 'allyear' 'allyear'
  - week       (timeslice) object 'weekday' 'weekend'
  - day        (timeslice) object 'allday' 'allday'
  * a          (a) int64 1 2 3
  * b          (b) <U1 'd' 'e' 'f'
```

We can check that nothing has been added to z (the quantity is `EXTENSIVE` by default):

```
>>> from numpy import all
>>> all(zfine.sum("timeslice").round(6) == zrough.sum("timeslice").round(6))
<xarray.DataArray ()>
array(True)
```

Or that the ratio of weekdays to weekends makes sense: >>> weekdays = (… zrough … .unstack("timeslice") … .sel(week="weekday") … .stack(timeslice=["semester", "day"]) … .squeeze() …) >>> weekend = (… zrough … .unstack("timeslice") … .sel(week="weekend") … .stack(timeslice=["semester", "day"]) … .squeeze() …) >>> bool(all((weekend * 5).round(6) == (weekdays * 2).round(6))) True

`muse.timeslices.`**`reference_timeslice`**(*settings: Union[Mapping, str], level_names: Sequence[str] = 'month', 'day', 'hour', name: str = 'timeslice'*) → xarray.core.dataarray.DataArray

Reads reference timeslice from toml like input.

Parameters

- **`settings`** – A dictionary of nested dictionaries or a string that toml will interpret as such. The nesting specifies different levels of the timeslice. If a dictionary and it contains "timeslices" key, then the associated value is used as the root dictionary. Ultimately, the most nested values should be relative weights for each slice in the timeslice, e.g. the corresponding number of hours.
- **`level_names`** – Hints indicating the names of each level. Can also be given a "level_names" key in `settings`.
- **`name`** – name of the reference array

Returns A `DataArray` with dimension *timeslice* and values representing the relative weight of each timeslice.

Example

```
>>> from muse.timeslices import reference_timeslice
>>> reference_timeslice(
...     """
...     [timeslices]
...     spring.weekday = 5
...     spring.weekend = 2
...     autumn.weekday = 5
...     autumn.weekend = 2
...     winter.weekday = 5
...     winter.weekend = 2
...     summer.weekday = 5
...     summer.weekend = 2
...     level_names = ["season", "week"]
...     """
... )
<xarray.DataArray (timeslice: 8)>
array([5, 2, 5, 2, 5, 2, 5, 2])
Coordinates:
  * timeslice  (timeslice) MultiIndex
  - season     (timeslice) object 'spring' 'spring' ... 'summer' 'summer'
  - week       (timeslice) object 'weekday' 'weekend' ... 'weekday' 'weekend'
```

`muse.timeslices.`**`represent_hours`**(*timeslices: xarray.core.dataarray.DataArray, nhours: Union[int, float] = 8765.82*) → xarray.core.dataarray.DataArray

Number of hours per timeslice.

Parameters

- **`timeslices`** – The timeslice for which to compute the number of hours
- **`nhours`** – The total number of hours represented in the timeslice. Defaults to the average number of hours in year.

`muse.timeslices.`**`setup_module`**(*settings: Union[str, Mapping]*)

Sets up module singletons.

`muse.timeslices.`**`timeslice_projector`**(*x: Union[xarray.core.dataarray.DataArray, pandas.core.indexes.multi.MultiIndex], finest: Optional[xarray.core.dataarray.DataArray] = None, transforms: Optional[Dict[Tuple, numpy.ndarray]] = None*) → xarray.core.dataarray.DataArray

Project time-slice to standardized finest time-slices.

Returns a matrix from the input timeslice `x` to the `finest` timeslice, using the input `transforms`. The latter are a set of transforms that map indices from one timeslice to indices in another.

Example

Lets define the following timeslices and aggregates:

```
>>> toml = """
...     ["timeslices"]
...     winter.weekday.day = 5
...     winter.weekday.night = 5
...     winter.weekend.day = 2
...     winter.weekend.night = 2
...     winter.weekend.dusk = 1
...     summer.weekday.day = 5
...     summer.weekday.night = 5
...     summer.weekend.day = 2
...     summer.weekend.night = 2
...     summer.weekend.dusk = 1
...     level_names = ["semester", "week", "day"]
...     aggregates.allday = ["day", "night"]
... """
>>> from muse.timeslices import (
...     reference_timeslice,  aggregate_transforms
... )
>>> ref = reference_timeslice(toml)
>>> transforms = aggregate_transforms(toml, ref)
>>> from pandas import MultiIndex
>>> input_ts = DataArray(
...     [1, 2, 3],
...     coords={
...         "timeslice": MultiIndex.from_tuples(
...             [
...                 ("winter", "weekday", "allday"),
...                 ("winter", "weekend", "dusk"),
...                 ("summer", "weekend", "night"),
...             ],
...             names=ref.get_index("timeslice").names,
...         ),
...     },
...     dims="timeslice"
... )
>>> input_ts
<xarray.DataArray (timeslice: 3)>
array([1, 2, 3])
Coordinates:
  * timeslice  (timeslice) MultiIndex
  - semester   (timeslice) object 'winter' 'winter' 'summer'
  - week       (timeslice) object 'weekday' 'weekend' 'weekend'
  - day        (timeslice) object 'allday' 'dusk' 'night'
```

The input timeslice does not have to be complete. In any case, we can now compute a transform, i.e. a matrix that will take this timeslice and transform it to the equivalent times in the finest timeslice:

```
>>> from muse.timeslices import timeslice_projector
>>> timeslice_projector(input_ts, ref, transforms)
<xarray.DataArray 'projector' (finest_timeslice: 10, timeslice: 3)>
array([[1, 0, 0],
       [1, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
        [0, 1, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 1],
        [0, 0, 0]])
Coordinates:
  * finest_timeslice  (finest_timeslice) MultiIndex
  - finest_semester   (finest_timeslice) object 'winter' 'winter' ... 'summer'
  - finest_week       (finest_timeslice) object 'weekday' ... 'weekend'
  - finest_day        (finest_timeslice) object 'day' 'night' ... 'night' 'dusk'
  * timeslice         (timeslice) MultiIndex
  - semester          (timeslice) object 'winter' 'winter' 'summer'
  - week              (timeslice) object 'weekday' 'weekend' 'weekend'
  - day               (timeslice) object 'allday' 'dusk' 'night'
```

It is possible to give as input an array which does not have a timeslice of its own:

```
>>> nots = DataArray([5.0, 1.0, 2.0], dims="a", coords={'a': [1, 2, 3]})
>>> timeslice_projector(nots, ref, transforms).T
<xarray.DataArray (timeslice: 1, finest_timeslice: 10)>
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
Coordinates:
  * finest_timeslice  (finest_timeslice) MultiIndex
  - finest_semester   (finest_timeslice) object 'winter' 'winter' ... 'summer'
  - finest_week       (finest_timeslice) object 'weekday' ... 'weekend'
  - finest_day        (finest_timeslice) object 'day' 'night' ... 'night' 'dusk'
Dimensions without coordinates: timeslice
```

8.8.2 Commodities

Methods and types around commodities.

class `muse.commodities.`**`CommodityUsage`**(*value*)

Flags to specify the different kinds of commodities.

For details on how `enum`'s work, see python's documentation. In practice, `CommodityUsage` centralizes in one place the different kinds of commodities that are meaningfull to the generalized sector, e.g. commodities that are consumed by the sector, and commodities that produced by the sectors, as well commodities that are, somehow, *environmental*.

With the exception of `CommodityUsage.OTHER`, flags can be combined in any fashion. `CommodityUsage.PRODUCT | CommodityUsage.CONSUMABLE` is a commodity that is both consumed and produced by a sector. `CommodityUsage.ENVIRONMENTAL | CommodityUsage.ENERGY | CommodityUsage.CONSUMABLE` is an environmental energy commodity consumed by the sector.

`CommodityUsage.OTHER` is an alias for *no* flag. It is meant for commodities that should be ignored by the sector.

CONSUMABLE = 1

Commodity which can be consumed by the sector.

ENERGY = 8

Commodity which is a fuel for this or another sector.

ENVIRONMENTAL = 4
Commodity which is a pollutant.

OTHER = 0
Not relevant for current sector.

PRODUCT = 2
Commodity which can be produced by the sector.

muse.commodities.**check_usage**(*data: Sequence[muse.commodities.CommodityUsage], flag: Optional[Union[str, muse.commodities.CommodityUsage]], match: str = 'all'*) → numpy.ndarray

Match usage flags with input data array.

Parameters

- **data** – sequence for which to match flags elementwise.
- **flag** – flag or combination of flags to match. The input can be a string, such as "product | environmental", or a CommodityUsage instance. Defaults to "other".
- **match** – one of: - "all": should all flag match. Default. - "any", should match at least one flags. - "exact", should match each flag and nothing else.

Examples

```
>>> from muse.commodities import CommodityUsage, check_usage
>>> data = [
...     CommodityUsage.OTHER,
...     CommodityUsage.PRODUCT,
...     CommodityUsage.ENVIRONMENTAL | CommodityUsage.PRODUCT,
...     CommodityUsage.ENVIRONMENTAL,
... ]
```

Matching "all":

```
>>> check_usage(data, CommodityUsage.PRODUCT).tolist()
[False, True, True, False]
```

```
>>> check_usage(data, CommodityUsage.ENVIRONMENTAL).tolist()
[False, False, True, True]
```

```
>>> check_usage(
...     data, CommodityUsage.ENVIRONMENTAL | CommodityUsage.PRODUCT
... ).tolist()
[False, False, True, False]
```

Matching "any":

```
>>> check_usage(data, CommodityUsage.PRODUCT, match="any").tolist()
[False, True, True, False]
```

```
>>> check_usage(data, CommodityUsage.ENVIRONMENTAL, match="any").tolist()
[False, False, True, True]
```

```
>>> check_usage(data, "environmental | product", match="any").tolist()
[False, True, True, True]
```

Matching "exact":

```
>>> check_usage(data, "PRODUCT", match="exact").tolist()
[False, True, False, False]
```

```
>>> check_usage(data, CommodityUsage.ENVIRONMENTAL, match="exact").tolist()
[False, False, False, True]
```

```
>>> check_usage(data, "ENVIRONMENTAL | PRODUCT", match="exact").tolist()
[False, False, True, False]
```

Finally, checking no flags has been set can be done with:

```
>>> check_usage(data, CommodityUsage.OTHER, match="exact").tolist()
[True, False, False, False]
>>> check_usage(data, None, match="exact").tolist()
[True, False, False, False]
```

`muse.commodities.`**`is_consumable`**(*data: Sequence[muse.commodities.CommodityUsage]*) → numpy.ndarray

Any consumable.

Examples

```
>>> from muse.commodities import CommodityUsage, is_consumable
>>> data = [
...     CommodityUsage.CONSUMABLE,
...     CommodityUsage.PRODUCT,
...     CommodityUsage.ENVIRONMENTAL,
...     CommodityUsage.PRODUCT | CommodityUsage.CONSUMABLE,
...     CommodityUsage.ENVIRONMENTAL | CommodityUsage.PRODUCT,
... ]
>>> is_consumable(data).tolist()
[True, False, False, True, False]
```

`muse.commodities.`**`is_enduse`**(*data: Sequence[muse.commodities.CommodityUsage]*) → numpy.ndarray

Non-environmental product.

Examples

```
>>> from muse.commodities import CommodityUsage, is_enduse
>>> data = [
...     CommodityUsage.CONSUMABLE,
...     CommodityUsage.PRODUCT,
...     CommodityUsage.ENVIRONMENTAL,
...     CommodityUsage.PRODUCT | CommodityUsage.CONSUMABLE,
...     CommodityUsage.ENVIRONMENTAL | CommodityUsage.PRODUCT,
... ]
>>> is_enduse(data).tolist()
[False, True, False, True, False]
```

`muse.commodities.`**`is_fuel`**(*data: Sequence[muse.commodities.CommodityUsage]*) → numpy.ndarray

Any consumable energy.

Examples

```
>>> from muse.commodities import CommodityUsage, is_fuel
>>> data = [
...     CommodityUsage.CONSUMABLE,
...     CommodityUsage.PRODUCT,
...     CommodityUsage.ENERGY,
...     CommodityUsage.ENERGY | CommodityUsage.CONSUMABLE,
...     CommodityUsage.ENERGY | CommodityUsage.CONSUMABLE
...         | CommodityUsage.ENVIRONMENTAL,
...     CommodityUsage.ENERGY | CommodityUsage.CONSUMABLE
...         | CommodityUsage.PRODUCT,
...     CommodityUsage.ENERGY | CommodityUsage.PRODUCT,
... ]
>>> is_fuel(data).tolist()
[False, False, False, True, True, True, False]
```

`muse.commodities.`**`is_material`**(*data: Sequence[muse.commodities.CommodityUsage]*) → numpy.ndarray

Any non-energy non-environmental consumable.

Examples

```
>>> from muse.commodities import CommodityUsage, is_material
>>> data = [
...     CommodityUsage.CONSUMABLE,
...     CommodityUsage.PRODUCT,
...     CommodityUsage.ENERGY,
...     CommodityUsage.ENERGY | CommodityUsage.CONSUMABLE,
...     CommodityUsage.CONSUMABLE | CommodityUsage.ENVIRONMENTAL,
...     CommodityUsage.ENERGY | CommodityUsage.CONSUMABLE
...         | CommodityUsage.PRODUCT,
...     CommodityUsage.CONSUMABLE | CommodityUsage.PRODUCT,
... ]
>>> is_material(data).tolist()
[True, False, False, False, False, False, True]
```

`muse.commodities.`**`is_other`**(*data: Sequence[muse.commodities.CommodityUsage]*) → numpy.ndarray

No flags are set.

Examples

```
>>> from muse.commodities import CommodityUsage, is_other
>>> data = [
...     CommodityUsage.OTHER,
...     CommodityUsage.PRODUCT,
...     CommodityUsage.PRODUCT | CommodityUsage.OTHER,
... ]
>>> is_other(data).tolist()
[True, False, False]
```

`muse.commodities.`**`is_pollutant`**(*data: Sequence[muse.commodities.CommodityUsage]*) → numpy.ndarray

Environmental product.

Examples

```
>>> from muse.commodities import CommodityUsage, is_pollutant
>>> data = [
...     CommodityUsage.CONSUMABLE,
...     CommodityUsage.PRODUCT,
...     CommodityUsage.ENVIRONMENTAL,
...     CommodityUsage.PRODUCT | CommodityUsage.CONSUMABLE,
...     CommodityUsage.ENVIRONMENTAL | CommodityUsage.PRODUCT,
... ]
>>> is_pollutant(data).tolist()
[False, False, False, False, True]
```

8.8.3 Regression functions

Functions and functors to compute macro-drivers.

class muse.regressions.**Exponential**(**args, **kwds*)

Regression function: exponential

This functor is a regression function registered with MUSE as 'exponential'.

class muse.regressions.**ExponentialAdj**(**args, **kwds*)

Regression function: exponentialadj

This functor is a regression function registered with MUSE as 'exponentialadj'.

class muse.regressions.**Linear**(**args, **kwds*)

a * population + b * (gdp - gdp[2010]/population[2010] * population)

class muse.regressions.**Logistic**(**args, **kwds*)

Regression function: logistic

This functor is a regression function registered with MUSE as 'logistic'.

class muse.regressions.**LogisticSigmoid**(**args, **kwds*)

Regression function: logisticsigmoid

This functor is a regression function registered with MUSE as 'logisticsigmoid'.

class muse.regressions.**Loglog**(**args, **kwds*)

Regression function: loglog

This functor is a regression function registered with MUSE as 'loglog'.

muse.regressions.**endogenous_demand**(*regression_parameters: Union[str, pathlib.Path, xarray.core.dataset.Dataset], drivers: Union[str, pathlib.Path, xarray.core.dataset.Dataset], sector: Optional[Union[str, Sequence]] = None, **kwargs*) → xarray.core.dataset.Dataset

Endogenous demand based on macro drivers and regression parameters.

muse.regressions.**factory**(*regression_parameters: Union[str, pathlib.Path, xarray.core.dataset.Dataset], sector: Optional[Union[str, Sequence[str]]] = None*) → muse.regressions.Regression

Creates regression functor from standard MUSE data for given sector.

muse.regressions.**register_regression**(*Functor: Optional[muse.regressions.Regression] = None, name: Optional[str] = None*) → muse.regressions.Regression

Registers a functor with MUSE regressions.

Regression functors are registered with MUSE so that the functors can be called easily on created.

functor name that the functor is registered with defaults to the snake_case version of the functor name. However, it can also be specified explicitly as a *keyword* argument. In any case, it must be unique amongst all registered regression functor.

8.8.4 Functionality Registration

Registrators that allow pluggable data to logic transforms.

muse.registration.**registrator**(*decorator: Callable = None*, *registry: MutableMapping = None*, *logname: Optional[str] = None*, *loglevel: Optional[str] = 'Debug'*) → Callable

A decorator to create a decorator that registers functions with MUSE.

This is a decorator that takes another decorator as an argument. Hence it returns a decorator. It simplifies and standardizes creating decorators to register functions with muse.

The registrator expects as non-optional keyword argument a registry where the resulting decorator will register functions.

Furthermore, the final function (the one passed to the decorator passed to this function) will emit a standardized log-call.

Example

At it's simplest, creating a registrator and registrating happens by first declaring a registry.

```
>>> REGISTRY = {}
```

In general, it will be a variable owned directly by a module, hence the all-caps. Creating the registrator then follows:

```
>>> from muse.registration import registrator
>>> @registrator(registry=REGISTRY, logname='my stuff',
...              loglevel='Info')
... def register_mystuff(function):
...     return function
```

This registrator does nothing more than register the function. A more interesting example is given below. Then a function can be registered:

```
>>> @register_mystuff(name='yoyo')
... def my_registered_function(a, b):
...     return a + b
```

The argument 'yoyo' is optional. It adds aliases for the function in the registry. In any case, functions are registered with default aliases corresponding to standard name variations, e.g. CamelCase, camelCase, and kebab-case, as illustrated below:

```
>>> REGISTRY['my_registered_function'] is my_registered_function
True
>>> REGISTRY['my-registered-function'] is my_registered_function
True
>>> REGISTRY['yoyo'] is my_registered_function
True
```

A more interesting case would involve the registrator automatically adding functionality to the input function. For instance, the inputs could be manipulated and the result of the function could be automatically transformed to a string:

```
>>> from muse.registration import registrator
>>> @registrator(registry=REGISTRY)
... def register_mystuff(function):
...     from functools import wraps
...
...     @wraps(function)
...     def decorated(a, b) -> str:
...         result = function(2 * a, 3 * b)
...         return str(result)
...
...     return decorated
```

```
>>> @register_mystuff
... def other(a, b):
...     return a + b
```

```
>>> isinstance(REGISTRY['other'](-3, 2), str)
True
>>> REGISTRY['other'](-3, 2) == "0"
True
```

8.8.5 Utilities

Collection of functions and stand-alone algorithms.

`muse.utilities.agent_concatenation`(*data: Mapping[Hashable, Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]], dim: str = 'asset', name: str = 'agent', fill_value: Any = 0*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Concatenates input map along given dimension.

Example

Lets create sets of random assets to work with. We set the seed so that this test can be reproduced exactly.

```
>>> from muse.examples import random_agent_assets
>>> rng = np.random.default_rng(1234)
>>> assets = {i: random_agent_assets(rng) for i in range(5)}
```

The concatenation will create a new dataset (or datarray) combining all the inputs along the dimension "asset". The origin of each datum is retained in a new coordinate "agent" with dimension "asset".

```
>>> from muse.utilities import agent_concatenation
>>> aggregate = agent_concatenation(assets)
>>> aggregate
<xarray.Dataset>
Dimensions:     (asset: 19, year: 12)
Coordinates:
```

(continues on next page)

(continued from previous page)

```
  * year         (year) int64 2033 2035 2036 2037 2039 ... 2046 2047 2048 2049
    technology   (asset) <U9 'oven' 'stove' 'oven' ... 'stove' 'oven' 'thermomix'
    region       (asset) <U9 'Brexitham' 'Brexitham' ... 'Brexitham' 'Brexitham'
    agent        (asset) ... 0 0 0 0 0 1 1 1 2 2 2 2 3 3 3 4 4 4 4
    installed    (asset) int64 2030 2025 2030 2010 2030 ... 2025 2030 2010 2025
Dimensions without coordinates: asset
Data variables:
    capacity     (asset, year) float64 26.0 26.0 26.0 56.0 ... 62.0 62.0 62.0
```

Note that the *dtype* of the capacity has changed from integers to floating points. This is due to how `xarray` performs the operation.

We can check that all the data from each agent is indeed present in the aggregate.

```
>>> for agent, inventory in assets.items():
...     assert (aggregate.sel(asset=aggregate.agent == agent) == inventory).all()
```

However, it should be noted that the data is not always strictly equivalent: dimensions outside of "assets" (most notably "year") will include all points from all agents. Missing values for the "year" dimension are forward filled (and backfilled with zeros). Others are left with "NaN".

`muse.utilities.`**`aggregate_technology_model`**(*data: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], dim: str = 'asset', drop: Union[str, Sequence[str]] = 'installed'*) → Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]

Aggregate together assets with the same installation year.

The assets of a given agent, region, and technology but different installation year are grouped together and summed over.

Example

We first create a random set of agent assets and aggregate them. Some of these agents own assets from the same technology but potentially with different installation year. This function will aggregate together all assets of a given agent with same technology.

```
>>> from muse.examples import random_agent_assets
>>> from muse.utilities import agent_concatenation, aggregate_technology_model
>>> rng = np.random.default_rng(1234)
>>> agent_assets = {i: random_agent_assets(rng) for i in range(5)}
>>> assets = agent_concatenation(agent_assets)
>>> reduced = aggregate_technology_model(assets)
```

We can check that the tuples (agent, technology) are unique (each agent works in a single region):

```
>>> ids = list(zip(reduced.agent.values, reduced.technology.values))
>>> assert len(set(ids)) == len(ids)
```

And we can check they correspond to the right summation:

```
>>> for agent, technology in set(ids):
...     techsel = assets.technology == technology
...     agsel = assets.agent == agent
...     expected = assets.sel(asset=techsel & agsel).sum("asset")
...     techsel = reduced.technology == technology
```

(continues on next page)

(continued from previous page)

```
...     agsel = reduced.agent == agent
...     actual = reduced.sel(asset=techsel & agsel)
...     assert len(actual.asset) == 1
...     assert (actual == expected).all()
```

muse.utilities.**avoid_repetitions**(*data: xarray.core.dataarray.DataArray*, *dim: str = 'year'*) → xarray.core.dataarray.DataArray

list of years such that there is no repetition in the data.

It removes the central year of any three consecutive years where all data is the same. This means the original data can be reobtained via a linear interpolation or a forward fill.

The first and last year are always preserved.

muse.utilities.**broadcast_techs**(*technologies: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]*, *template: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset]*, *dimension: str = 'asset'*, *interpolation: str = 'linear'*, *installed_as_year: bool = True*, ***kwargs*) → Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]

Broadcasts technologies to the shape of template in given dimension.

The dimensions of the technologies are fully explicit, in that each concept 'technology', 'region', 'year' (for year of issue) is a separate dimension. However, the dataset or data arrays representing other quantities, such as capacity, are often flattened out with coordinates 'region', 'installed', and 'technology' represented in a single 'asset' dimension. This latter representation is sparse if not all combinations of 'region', 'installed', and 'technology' are present, whereas the former represention makes it easier to select a subset of the same.

This function broadcast the first represention to the shape and coordinates of the second.

Parameters

- **technologies** – The dataset to broadcast
- **template** – the dataset or data-array to use as a template
- **dimension** – the name of the dimensiom from *template* over which to broadcast
- **interpolation** – interpolation method used across *year*
- **installed_as_year** – if the coordinate *installed* exists, then it is applied to the *year* dimension of the technologies dataset
- **kwargs** – further arguments are used initial filters over the *technologies* dataset.

muse.utilities.**clean_assets**(*assets: xarray.core.dataset.Dataset*, *years: Union[int, Sequence[int]]*)

Cleans up and prepares asset for current iteration.

- adds current and forecast year by backfilling missing entries
- removes assets for which there is no capacity now or in the future

muse.utilities.**coords_to_multiindex**(*data: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]*, *dimension: str = 'asset'*) → Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]

Creates a multi-index from flattened multiple coords.

`muse.utilities.`**`filter_input`**(*dataset: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray], year: Optional[Union[int, Iterable[int]]] = None, interpolation: str = 'linear', **kwargs*) → Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]

Filter inputs, taking care to interpolate years.

`muse.utilities.`**`filter_with_template`**(*data: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray], template: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset], asset_dimension: str = 'asset', **kwargs*)

Filters data to match template.

If the *asset_dimension* is present in *template.dims*, then the call is forwarded to *broadcast_techs*. Otherwise, the set of dimensions and indices in common between *template* and *data* are determined, and the resulting call is forwarded to *filter_input*.

Parameters

- **`data`** – Data to transform
- **`template`** – Data from which to figure coordinates and dimensions
- **`asset_dimension`** – Name of the dimension which if present indicates the format is that of an *asset* (see *broadcast_techs*)
- **`kwargs`** – passed on to *broadcast_techs* or *filter_input*

Returns *data* transformed to match the form of *template*

`muse.utilities.`**`future_propagation`**(*data: xarray.core.dataarray.DataArray, future: xarray.core.dataarray.DataArray, threshhold: float = 1e-12, dim: str = 'year'*) → xarray.core.dataarray.DataArray

Propagates values into the future.

Example

`Data` should be an array with at least one dimension, "year":

```
>>> coords = dict(year=list(range(2020, 2040, 5)), fuel=["gas", "coal"])
>>> data = xr.DataArray(
...     [list(range(4)), list(range(-5, -1))],
...     coords=coords,
...     dims=("fuel", "year")
... )
```

`future` is an array with exactly one year in its `year` coordinate, or that coordinate must correspond to a scalar. That one year should also be present in `data`.

```
>>> future = xr.DataArray(
...     [1.2, -3.95], coords=dict(fuel=coords['fuel'], year=2025), dims="fuel",
... )
```

This function propagates into `data` values from `future`, but only if those values differed for the current year beyond a given threshhold:

```
>>> from muse.utilities import future_propagation
>>> future_propagation(data, future, threshhold=0.1)
<xarray.DataArray (fuel: 2, year: 4)>
array([[ 0. ,  1.2,  1.2,  1.2],
       [-5. , -4. , -3. , -2. ]])
Coordinates:
  * year     (year) ... 2020 2025 2030 2035
  * fuel     (fuel) <U4 'gas' 'coal'
```

Above, the data for coal is not sufficiently different given the threshhold. hence, the future values for coal remain as they where.

The dimensions of `future` do not have to match exactly those of `data`. Standard broadcasting is used if they do not match:

```
>>> future_propagation(data, future.sel(fuel="gas", drop=True), threshhold=0.1)
<xarray.DataArray (fuel: 2, year: 4)>
array([[ 0. ,  1.2,  1.2,  1.2],
       [-5. ,  1.2,  1.2,  1.2]])
Coordinates:
  * year     (year) ... 2020 2025 2030 2035
  * fuel     (fuel) <U4 'gas' 'coal'
>>> future_propagation(data, future.sel(fuel="coal", drop=True), threshhold=0.1)
<xarray.DataArray (fuel: 2, year: 4)>
array([[ 0.  , -3.95, -3.95, -3.95],
       [-5.  , -4.  , -3.  , -2.  ]])
Coordinates:
  * year     (year) ... 2020 2025 2030 2035
  * fuel     (fuel) <U4 'gas' 'coal'
```

`muse.utilities.`**`lexical_comparison`**(*objectives: xarray.core.dataset.Dataset, binsize: xarray.core.dataset.Dataset, order: Optional[Sequence[Hashable]] = None, bin_last: bool = True*) → xarray.core.dataarray.DataArray

Lexical comparison over the objectives.

Lexical comparison operates by binning the objectives into bins of width *binsize*. Once binned, dimensions other than *asset* and *technology* are reduced by taking the max, e.g. the largest constraint. Finally, the objectives are ranked lexographically, in the order given by the parameters.

Parameters

- **`objectives`** – xr.Dataset containing the objectives to rank
- **`binsize`** – bin size, minimization direction (+ -> minimize, - -> maximize), and (optionally) order of lexicographical comparison. The order is the one given *binsize.data_vars* if the argument *order* is None.
- **`order`** – Optional array indicating the order in which to rank the tuples.
- **`bin_last`** – Whether the last metric should be binned, or whether it should be left as a the type it already is (e.g. no flooring and no turning to integer.)

Result: An array of tuples which can subsquently be compared lexicographically.

`muse.utilities.`**`merge_assets`**(*capa_a: xarray.core.dataarray.DataArray, capa_b: xarray.core.dataarray.DataArray, interpolation: str = 'linear', dimension: str = 'asset'*) → xarray.core.dataarray.DataArray

Merge two capacity arrays.

`muse.utilities.`**`multiindex_to_coords`**(*data: Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray], dimension: str = 'asset'*)

Flattens multi-index dimension into multi-coord dimension.

`muse.utilities.`**`nametuple_to_dict`**(*nametup: Union[Mapping, NamedTuple]*) → Mapping

Transforms a nametuple of type GenericDict into an OrderDict.

`muse.utilities.`**`reduce_assets`**(*assets: Union[xarray.core.dataarray.DataArray, xarray.core.dataset.Dataset, Sequence[Union[xarray.core.dataset.Dataset, xarray.core.dataarray.DataArray]]], coords: Optional[Union[str, Sequence[str], Iterable[str]]] = None, dim: str = 'asset', operation: Optional[Callable] = None*) → xarray.core.dataarray.DataArray

Combine assets along given asset dimension.

This method simplifies combining assets accross multiple agents, or combining assets across a given dimension. By default, it will sum together assets from the same region which have the same technology and the same installation date. In other words, assets are identified by the technology, installation year and region. The reduction happens over other possible coordinates, e.g. the owning agent.

More specifically, assets are often indexed using what xarray calls a **dimension without coordinates**. In practice, there are still coordinates associated with assets, e.g. *technology* and *installed* (installation year or version), but the value associated with these coordinates are not unique. There may be more than one asset with the same technology or installation year.

For instance, with assets per agent defined as $A^{i,r}_o$, with i an agent index, r a region, o is the coordinates identified in `coords`, and T the transformation identified by `operation`, then this function computes:

$$R_{r,o} = T[\{A^{i,r}_o; \forall i\}]$$

If T is the sum operation, then:

$$R_{r,o} = \sum_i A^{i,r}_o$$

Example

Lets construct assets that do have duplicates assets. First we construct the dimensions, using fake data:

```
>>> data = xr.Dataset()
>>> data['year'] = 'year', [2010, 2015, 2017]
>>> data['installed'] = 'asset', [1990, 1991, 1991, 1990]
>>> data['technology'] = 'asset', ['a', 'b', 'b', 'c']
>>> data['region'] = 'asset', ['x', 'x', 'x', 'y']
>>> data = data.set_coords(('installed', 'technology', 'region'))
```

We can check there are duplicate assets in this coordinate system:

```
>>> processes = set(
...     zip(data.installed.values, data.technology.values, data.region.values)
... )
>>> len(processes) < len(data.asset)
True
```

Now we can easily create a fake two dimensional quantity per process and per year:

```
>>> data['capacity'] = ('year', 'asset'), np.arange(3 * 4).reshape(3, 4)
```

The point of *reduce_assets* is to aggregate assets that refer to the same process:

```
>>> reduce_assets(data.capacity)
<xarray.DataArray 'capacity' (year: 3, asset: 3)>
array([[ 0,  3,  3],
       [ 4,  7, 11],
       [ 8, 11, 19]])
Coordinates:
  * year        (year) ... 2010 2015 2017
    installed   (asset) ... 1990 1990 1991
    technology  (asset) <U1 'a' 'c' 'b'
    region      (asset) <U1 'x' 'y' 'x'
Dimensions without coordinates: asset
```

We can also specify explicitly which coordinates in the 'asset' dimension should be reduced, and how:

```
>>> reduce_assets(
...     data.capacity,
...     coords=('technology', 'installed'),
...     operation = lambda x: x.mean(dim='asset')
... )
<xarray.DataArray 'capacity' (year: 3, asset: 3)>
array([[ 0. ,  1.5,  3. ],
       [ 4. ,  5.5,  7. ],
       [ 8. ,  9.5, 11. ]])
Coordinates:
  * year        (year) ... 2010 2015 2017
    technology  (asset) <U1 'a' 'b' 'c'
    installed   (asset) ... 1990 1991 1990
Dimensions without coordinates: asset
```

`muse.utilities.`**`tupled_dimension`**(*array: numpy.ndarray*, *axis: int*)
    Transforms one axis into a tuples.

8.8.6 Examples

Example models and datasets.

Helps create and run small standard models from the command-line or directly from python.

To run from the command-line:

```
python -m muse --model default
```

Other models may be available. Check the command-line help:

```
python -m muse --help
```

The same models can be instanciated in a python script as follows:

```
from muse import example
model = example.model("default")
model.run()
```

`muse.examples.`**`model`**(*name: str = 'default'*) → muse.mca.MCA
    Fully constructs a given example model.

`muse.examples.`**`technodata`**(*sector: str*, *model: str = 'default'*) → xarray.core.dataset.Dataset

Technology for a sector of a given example model.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX