1 A note on Jupyter Notebooks

If this is your first time with a Jupyter Notebook, please read on. Otherwise, skip to the next section.

Jupyter notebooks allow to mix text, data and code, and they either run on local server on your computer or in a cloud hosted Jupyter node. Most universities run Jupyter nodes which integrate with their campus wide login services.

Notebooks can be coupled to github repositories so that students find a preconfigured working environment.

The code examples below are meant to be executed step by step (just activate a code cell with mouse click and hit shift+enter). Code/Text cells can be edited which allows you to change model parameters and play with the code.

2 Install the ESBMTK libraries

In order to install the esbmtk library, please execute the following code. You only need to do this once. The library will remain installed as long as this account is not deleted. This command will generate quite a bit of output but there should be no error messages.

```
# Install a pip package in the current Jupyter kernel
import sys
# installl esbmtk in your jupyter account
!{sys.executable} -m pip install esbmtk

# update esbmtk if necessary
# !{sys.executable} -m pip install --upgrade esbmtk
```

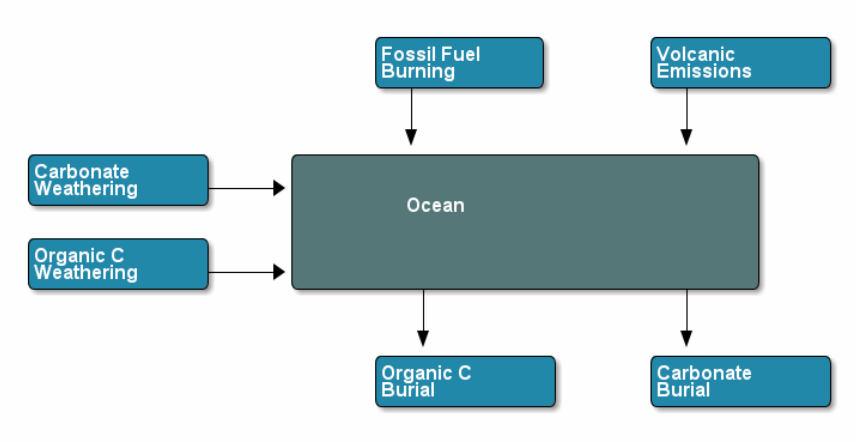
3 A worked example

Note that this example is not mean to be scientifically correct. It simply demonstrates various ESBMTK features

In the following example we will set up a simple carbon cycle model where the ocean is represented by a single box

The data forcing the anthropogenic carbon flux will be read from a csv file. Interaction with external data is handled through the external data object which allows to integrate external data into the model framework. It can then be used to generate a signal, or it can be associated with a reservoir so that the data is plotted with the reservoir data.

The model consists of four sources, two sinks, and one reservoir. We will read external data from spreadsheet which contains values for the CO_2 from fossil fuel burning, and then evaluate the response of the ocean to this perturbation.



3.1 Setting up the model

We need to load all required libraries and all classes we want to use. Interaction with the model classes is done through keyword/value pairs. Use `help()` to inquire about the supported keyword value pairs.

ESBMTK is unit aware. The units are used to map all input data to internal units. The type of internal units needs to specified when creating the model object. The time unit is derived from the timestep variable. I.e., if the timestep is given in seconds, all other time related input will be mapped to seconds. Similarly you have to define the base mass unit. Typically, this will be moles, but other units like grams etc can also be used. At present ESBMTK cannot convert between different types of mass units (e.g., kg to moles). The usual prefixes like kilo, mega, milli etc are recognized. Volume units like `l` or `m**3` are recognized as well. ESBMTK also defines the sverdrup ("Sv")

Once the input units are mapped to base units specified by the model object, all data will be reported in the base units. The only exception is the `object.plot()` method which will transform the data back into the original unit. In other words, if your timestep is in years, but you specify your endtime in kyrs, the time axis will be plotted in kyrs. Similarly for all other data, with the exception of the isotope delta values.

The below code loads predefined definitions for carbon, but you can also define your own element and species objects.

```
from esbmtk import Model, Element, Species, Reservoir
from esbmtk import Signal, Connect, Source, Sink, ExternalData

# create model
Model(
    name="C_Cycle",      # model name
    stop="1000 yrs",     # end time of model
    timestep=" 1 yr",    # base unit for time
    mass_unit = "mol",   # base unit for mass
    volume_unit = "l",   # base unit for volume
    element="Carbon",     # load default element and species definitions
    offset="1751 yrs"    # map to external timescale
)
```

3.2 Using external data to initialize a signal

```
Signal(name = "ACR",               # Signal name
       species = CO2,              # Species
       filename = "emissions.csv" # filename
)
```

Once a signal instance has been created, it can be passed to a connector object in order to associate it with a flux (see the first connection below as an example).

3.3 Sources, Sinks and Reservoirs

The fundamental model object is the reservoir. Reservoirs are connected to each other by one or more fluxes. Fluxes are created implicitly by connecting two reservoirs.

Connecting a reservoir with a Source or Sink also creates a flux, but unlike reservoirs, sources and sinks do not have any associated data. They are merely there to allow the creation of a flux.

```
Source(name="Fossil_Fuel_Burning", species=CO2)
Source(name="Carbonate_Weathering", species=CO2)
Source(name="Organic_Weathering", species=CO2)
Source(name="Volcanic", species=CO2)
Sink(name="Carbonate_burial", species=CaCO3)
Sink(name="OM_burial", species=OM)

Reservoir(
    name="Ocean",                # Name of reservoir
    species=DIC,                 # Species handle
    delta=2,                     # initial delta
    concentration="2.6 mmol/l", # cocentration
    volume="1.332E18 m**3",      # reservoir size (m^3)
)
```

We now have all the model objects, and the only thing which is left to be done, is define how objects are connected to each other.

3.4 Connecting sources, reservoirs and sinks

The first statement below, connects the source `Fossil_Fuel_Burning` with the reservoir `Ocean`. This will create a flux with the name `Fossil_Fuel_-Burning_to_Ocean`. The rate and delta keywords indicate that this flux will be zero. However, we provide the process list keyword `pl = [ACR]` in order to associate the fossil fuel burning emission signal with this flux This data will be added to the `Fossil_Fuel_Burning_to_Ocean` flux (since the process is additive, the initial flux has to be zero!)

The type of flux depends on how we specify the connection. In the previous example we provided a signal, so the flux will change with time according to the signal data. If you look at the connection between `Carbonate_-Weathering` and `Ocean` below, we specify a given rate and delta value. So this flux will not change over time. If you look at the connection between `Ocean` and `OM_burial` the connection specifies a constant flux but with an `alpha = -26.3`. This indicates that this flux involves a fixed isotope offset relative to the upstream reservoir, i.e., the isotope ratio of this flux will

change dynamically in response to the isotope ratio of the reservoir, but with a constant offset.

The carbonate burial flux additionally specifies a reference value for the DIC concentration. The model will modify this flux in such away that the reservoirs returns to this concentration setpoint. The `k_cocentration` parameter defines how fast the ocean returns to the reference value.

```
# connect source to reservoir
Connect(
    source=Fossil_Fuel_Burning,  # source of flux
    sink=Ocean,                  # target of flux
    rate="0 mol/yr",             # weathering flux in
    delta=0,                     # set a default flux
    pl=[ACR],                    # process list, here the anthropogenic carbon release
    scale=0.5                    # assume that the ocean uptke is half of the ACR
)

Connect(
    source=Carbonate_Weathering,  # source of flux
    sink=Ocean,                   # target of flux
    rate="12.3E12 mol/yr",        # weathering flux in
    delta=0,                      # isotope ratio
    plot="no",
)

Connect(
    source=Organic_Weathering,  # source of flux
    sink=Ocean,                 # target of flux
    rate="4.0E12 mol/yr",       # flux rate
    delta=-20,                  # isotope ratio
    plot="no",
)

Connect(
    source=Volcanic,      # source of flux
    sink=Ocean,           # target of flux
    rate="6.0E12 mol/yr", # flux rate
    delta=-5,             # isotope ratio
    plot="no",
)

Connect(
    source=Ocean,          # source of flux
```

```
    sink=OM_burial,         # target of flux
    rate="4.2E12 mol/yr",   # burial rate
    alpha=-26.32,           # fractionation factor
)

Connect(
    source=Ocean,           # source of flux
    sink=Carbonate_burial, # target of flux
    rate="18.1E12 mol/yr", # burial rate
    ref_value="2.6 mmol/l",
    k_concentration = 1000,
    alpha=0,                # set the isotope fractionation
)
```

3.5 Add some external data to compare against

External data can be read from a csv file, which must contain 3 columns. Run `help(External_data)` for details.

```
ExternalData(name="measured_carbon_isotopes",
             filename = "measured_c_isotopes.csv",
             legend = "Dean et al. 2014",
             offset = "1750 yrs",
             reservoir = Ocean
             )
```

3.6 Running the model

The model is executed via the `run()` method. The results can be displayed withe the `plot_data()` method which will generate an overview graph for each reservoir. Export of the results to a csv file is done via the `save_data()` method which will create csv file for each reservoir.

```
# Run the model
C_Cycle.run()

# plot the results
C_Cycle.plot_data()
# save the results
C_Cycle.save_data()
```