OpenSCM two layer model contains implementations of the two layer radiative forcing driven models by Held et al., Geoffroy et al. and Bloch-Johnson et al.

OpenSCM Two Layer Model was designed to provide a clean, modularised, extensible interface for one of the most commonly used simple climate models. It was used in Phase 1 of the Reduced Complexity Model Intercomparison Project as a point of comparison for the other participating models.

The FaIR model implements a mathematically equivalent model (under certain assumptions) but does not provide as clear a conversion between the two-layer model and the two-timescale response as is provided here. We hope that this implementation could interface with other simple climate models like FaIR to allow simpler exploration of the combined behaviour of interacting climate components with minimal coupling headaches.

As implemented here, the “OpenSCM Two Layer Model” interface is target at researchers and as an education tool. Users from other fields are of course encouraged to use it if they find it helpful.

OpenSCM two layer model is free software under a BSD 3-Clause License, see LICENSE.

If you have any issues or would like to discuss a feature request, please raise them in the OpenSCM Two Layer Model issue tracker. If your issue is a feature request or a bug, please use the templates available, otherwise, simply open a normal issue.
OpenSCM two layer model has only two dependencies:
  • scmdatagreater than or equal to 0.9
  • tqdm
OpenSCM two layer model can be installed with pip

```
pip install openscm-twolayermodel
```

If you also want to run the example notebooks install additional dependencies using

```
pip install "openscm-twolayermodel[notebooks]"
```

**Coming soon** OpenSCM two layer model can also be installed with conda

```
conda install -c conda-forge openscm-twolayermodel
```

We do not ship our tests with the packages. If you wish to run the tests, you must install from source (or download the tests separately and run them on your installation).

### 1.1 Installing from source

To install from source, simply clone the repository and then install it using pip e.g. `pip install ".[dev]"`. Having done this, the tests can be run with `pytest tests` or using the Makefile (`make test` will run only the code tests, `make checks` will run the code tests and all other tests e.g. linting, notebooks, documentation).

For more details, see the development section of the docs.
Here we provide examples of OpenSCM two layer model’s behaviour and usage. The source code of these usage examples is available in the folder docs/source/usage of the GitHub repository.

2.1 Basic demos

2.1.1 Getting Started

This notebook demonstrates the OpenSCM Two Layer Model repository’s basic functionality.

We start with imports, their need will become clearer throughout the notebook.

```python
import inspect
import numpy as np
from openscm_units import unit_registry
from scmdata import ScmRun
import openscm_twolayermodel
from openscm_twolayermodel import ImpulseResponseModel, TwoLayerModel
from openscm_twolayermodel.base import Model
```

As with most Python packages, the version of openscm_twolayermodel being used can always be checked as shown below. This is very helpful for debugging.

```python
# NBVAL_IGNORE_OUTPUT
openscm_twolayermodel.__version__
```

2.2.3

OpenSCM Two Layer Model has two key classes: ImpulseResponseModel and TwoLayerModel. These are implementations of the two major variants of the two-layer model found in the literature. We can see that they both have a common base class using the inspect package.

```python
inspect.getmro(ImpulseResponseModel)
```
These classes can both be used in the same way. We demonstrate the most basic usage here, more comprehensive usage is demonstrated in other notebooks.

The first thing we need is our effective radiative forcing driver. This should be an `ScmRun <https://scmdata.readthedocs.io/en/latest/data.html#the-scmrun-class>`__ instance.

```python
run_length = 200

driver = ScmRun(
    data=np.arange(run_length) * 4 / 70,
    index=1850 + np.arange(run_length),
    columns={
        "unit": "W/m^2",
        "model": "idealised",
        "scenario": "1pctCO2",
        "region": "World",
        "variable": "Effective Radiative Forcing",
    },
)

driver
```

```
<scmdata.ScmRun (timeseries: 1, timepoints: 200)>
Time:
   Start: 1850-01-01T00:00:00
   End: 2049-01-01T00:00:00
Meta:
   model region scenario unit variable
   0 idealised World 1pctCO2 W/m^2 Effective Radiative Forcing
```

```
# NBVAL_IGNORE_OUTPUT
driver.lineplot()
```

```
<AxesSubplot:xlabel='time', ylabel='W/m^2'>
```
Then we can initialise instances of our models and run them.

```python
# NBVAL_IGNORE_OUTPUT
two_layer = TwoLayerModel(lambda0=4 / 3 * unit_registry("W/m^2/delta_degC"))
res_two_layer = two_layer.run_scenarios(driver)

impulse_response = ImpulseResponseModel(d1=10 * unit_registry("yr"))
res_impulse_response = impulse_response.run_scenarios(driver)

res = res_two_layer.append(res_impulse_response)
res.head()
```

```
scenarios: 0it [00:00, ?it/s]
scenarios: 0it [00:00, ?it/s]
```

```python
res.head()
```

```
scenarios: 0it [00:00, ?it/s]
```

```python
res = res_two_layer.append(res_impulse_response)
res.head()
```

```
scenarios: 0it [00:00, ?it/s]
scenarios: 0it [00:00, ?it/s]
```

```
1850-01-01 | a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
d1 (a) | d2 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius

.../ meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_

..._degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)

...region run_idx scenario unit variable
two_layer NaN 0.0 NaN 1200.0 50.0 1.0 0.8 1.333333 NaN idealised NaN World

...0 1pctCO2 W/m^2 Effective Radiative Forcing 0.0

delta_degC Surface Temperature|Upper 0.0

...Surface Temperature|Lower 0.0
```

(continues on next page)
W/m² Heat Uptake 0.0
NaN
two_timescale_impulse_response 10.0
400.0 NaN NaN 1.0 NaN
NaN NaN 0.4 idealised 0.3
0.4

1pctCO2 W/m² Effective Radiative Forcing 0.0

time

1851-01-01

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
d1 (a)
d2 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2)
lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt)
q2 (delta_degree_Celsius * meter ** 2 / watt)
region run_idx scenario unit variable
0.0 two_layer NaN
NaN 1200.0 50.0 1.0 0.8
1.333333
NaN NaN
0.8

1pctCO2 W/m² Effective Radiative Forcing 0.057143

delta_degC Surface Temperature|Upper 0.000000

Surface Temperature|Lower 0.000000

W/m² Heat Uptake 0.000000
NaN
two_timescale_impulse_response 10.0
400.0 NaN NaN 1.0 NaN
NaN NaN 0.4 idealised 0.3
0.4

1pctCO2 W/m² Effective Radiative Forcing 0.057143

time

1852-01-01
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model d1 (a) d2 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt) region run_idx scenario unit variable

| 0.0 | two_layer | NaN |
| NaN | 1200.0 | 50.0 | 1.0 | 0.8 |
| NaN | 1.33333 | idealised NaN |
| 0 | 1pctCO2 | W/m^2 | Effective Radiative Forcing | 0.114286 |

| delta_degC | Surface Temperature|Upper | 0.008626 |
| Surface Temperature|Lower | 0.000000 |
| W/m^2 | Heat Uptake | 0.057143 |
| NaN | two_timescale_impulse_response | 10.0 |
| 400.0 | NaN | NaN | NaN |
| NaN | idealised 0.3 |
| 0 | 1pctCO2 | W/m^2 | Effective Radiative Forcing | 0.114286 |

| time |
| 1853-01-01 |

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model d1 (a) d2 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt) region run_idx scenario unit variable

| 0.0 | two_layer | NaN |
| NaN | 1200.0 | 50.0 | 1.0 | 0.8 |
| NaN | 1.33333 | idealised NaN |
| 0 | 1pctCO2 | W/m^2 | Effective Radiative Forcing | 0.171429 |

| delta_degC | Surface Temperature|Upper | 0.023100 |
| Surface Temperature|Lower | 0.000043 |
安宁
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
  d1 (a)
  d2 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius /
  meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)
  region run_idx scenario unit variable
  0.0 two_layer NaN NaN
  NaN 1200.0 50.0 1.0 0.8 idealised NaN
  NaN
  1.333333 NaN World
  0 1pctCO2 W/m^2 Effective Radiative Forcing 0.285714

  delta_degC Surface Temperature|Upper 0.062689

  Surface Temperature|Lower 0.000368

  W/m^2 Heat Uptake 0.173178
  NaN two_timescale_impulse_response 10.0 NaN
  400.0 NaN NaN 1.0 NaN idealised 0.3
  NaN 0.4 World
  0 1pctCO2 W/m^2 Effective Radiative Forcing 0.285714

time

  1856-01-01 \n
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
  d1 (a)
  d2 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius /
  meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)
  region run_idx scenario unit variable
  0.0 two_layer NaN NaN
  NaN 1200.0 50.0 1.0 0.8 idealised NaN
  NaN
  1.333333 NaN World
  0 1pctCO2 W/m^2 Effective Radiative Forcing 0.342857

  delta_degC Surface Temperature|Upper 0.085676

  Surface Temperature|Lower 0.000681
(continues on next page)
### 1857-01-01

<table>
<thead>
<tr>
<th>delta_degC</th>
<th>Surface Temperature</th>
<th>Upper</th>
<th>0.109924</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Temperature</td>
<td>Lower</td>
<td>0.001109</td>
<td></td>
</tr>
<tr>
<td>W/m²</td>
<td>Heat Uptake</td>
<td>0.228623</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>two_timescale_impulse_response</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>400.0</td>
<td>NaN</td>
<td>NaN</td>
<td>1.0</td>
</tr>
<tr>
<td>NaN</td>
<td>idealised</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1pctCO2</td>
<td>W/m²</td>
<td>Effective Radiative Forcing</td>
</tr>
</tbody>
</table>

### 1858-01-01

<table>
<thead>
<tr>
<th>delta_degC</th>
<th>Surface Temperature</th>
<th>Upper</th>
<th>0.109924</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Temperature</td>
<td>Lower</td>
<td>0.001109</td>
<td></td>
</tr>
<tr>
<td>W/m²</td>
<td>Heat Uptake</td>
<td>0.228623</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>two_timescale_impulse_response</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>400.0</td>
<td>NaN</td>
<td>NaN</td>
<td>1.0</td>
</tr>
<tr>
<td>NaN</td>
<td>idealised</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1pctCO2</td>
<td>W/m²</td>
<td>Effective Radiative Forcing</td>
</tr>
</tbody>
</table>
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
d1 (a)
\(d2\) (a) \(dl\) (meter) \(du\) (meter) efficacy (dimensionless) \(\eta\) (watt / delta_degree_Celsius / meter ** 2) model
\(q1\) (delta_degree_Celsius / meter ** 2 / watt) \(q2\) (delta_degree_Celsius / meter ** 2 / watt)
region run_idx scenario unit variable

<table>
<thead>
<tr>
<th>0.0</th>
<th>two_layer</th>
<th>NaN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td>1200.0</td>
<td>50.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

0 1pctCO2 W/m^2 Effective Radiative Forcing 0.457143

delta_degC Surface Temperature|Upper 0.135040

Surface Temperature|Lower 0.001656

W/m^2 Heat Uptake 0.253435

0 1pctCO2 W/m^2 Effective Radiative Forcing 0.457143

time

1859-01-01

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
\(d1\) (a)
\(d2\) (a) \(dl\) (meter) \(du\) (meter) efficacy (dimensionless) \(\eta\) (watt / delta_degree_Celsius / meter ** 2) model
\(q1\) (delta_degree_Celsius / meter ** 2 / watt) \(q2\) (delta_degree_Celsius / meter ** 2 / watt)
region run_idx scenario unit variable

<table>
<thead>
<tr>
<th>0.0</th>
<th>two_layer</th>
<th>NaN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td>1200.0</td>
<td>50.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

0 1pctCO2 W/m^2 Effective Radiative Forcing 0.514286

delta_degC Surface Temperature|Upper 0.160761

Surface Temperature|Lower 0.002328
<table>
<thead>
<tr>
<th>Time</th>
<th>run_idx</th>
<th>scenario</th>
<th>unit</th>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0</td>
<td>1pctCO2</td>
<td>W/m²</td>
<td>Effective Radiative Forcing</td>
<td>0.514286</td>
</tr>
<tr>
<td>1200.0</td>
<td>0</td>
<td>50.0</td>
<td>1.0</td>
<td>0.8</td>
<td>0.3</td>
</tr>
<tr>
<td>1.333333</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>idealised Wa</td>
<td>NaN</td>
</tr>
<tr>
<td>0.0</td>
<td>0</td>
<td>1pctCO2</td>
<td>W/m²</td>
<td>Effective Radiative Forcing</td>
<td>...</td>
</tr>
</tbody>
</table>
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
   d1 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius /
   meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)
   region run_idx scenario unit variable
   0.0 two_layer NaN NaN
   NaN d1 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius /
   meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)
   0.8 idealised NaN
   1.0 NaN NaN NaN
   0.3 idealised NaN
   0.4 World

0 1pctCO2 W/m^2 Effective Radiative Forcing 10.857143

... delta_degC Surface Temperature|Upper 5.710809

... Surface Temperature|Lower 1.937427

... W/m^2 Heat Uptake 3.230641

... two_timescale_impulse_response 10.0

2041-01-01 \a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model
   d1 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius /
   meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)
   region run_idx scenario unit variable
   0.0 two_layer NaN NaN
   NaN d1 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius /
   meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)
   NaN NaN NaN
   NaN NaN NaN
   0.3 idealised NaN
   0.4 World

0 1pctCO2 W/m^2 Effective Radiative Forcing 10.914286

... delta_degC Surface Temperature|Upper 5.744627

... Surface Temperature|Lower 1.956414

(continues on next page)
### Climate Model Components

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>W/m² Heat Uptake</td>
<td>3.242731</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two timescale impulse response</td>
<td>10.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>NaN</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>NaN</td>
<td>idealised 0.3</td>
<td></td>
</tr>
<tr>
<td>World</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1pctCO2 W/m² Effective Radiative Forcing</td>
<td>10.914286</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| time                   |                    |                     |                                       |

<table>
<thead>
<tr>
<th>value</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2042-01-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model

| d1 (a) | d2 (a) dl (meter) du (meter) efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt) region run_idx scenario unit variable |
|--------|--------|----------------|---------------------------------|----------------|----------------|----------------|----------------|----------------|----------------|--------|----------------|----------------|---------------|---------------|
| NaN    | 1200.0 | 50.0          | 1.0                             | 0.8             | NaN            | NaN            | NaN            | 0.4            | idealised 0.3 | NaN              | NaN            | NaN            | NaN            | NaN            |
| 1.33333 |       |               |                                 |                 | NaN            | NaN            | 1.33333       | 0.8            | World          | 1.33333         | 0.8              | World          | 1.33333         | 0.8            |
| 0.0    | 1pctCO2 | W/m²          | Effective Radiative Forcing 10.914286 |                 | 1.33333       | 0.8            | 1.33333       | 0.8            | World          | 1.33333         | 0.8              | World          | 1.33333         | 0.8            |

<table>
<thead>
<tr>
<th>delta_degC</th>
<th>Surface Temperature</th>
<th>Upper</th>
<th>5.778474</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Surface Temperature</th>
<th>Lower</th>
<th>1.975476</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| time                   |                    |                     |                                       |

<table>
<thead>
<tr>
<th>value</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2043-01-01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)

---
### 2.1. Basic demos

<table>
<thead>
<tr>
<th>Region</th>
<th>Run ID</th>
<th>Scenario</th>
<th>Unit</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>surface</td>
<td>0</td>
<td>1pctCO2</td>
<td>W/m(^2)</td>
<td>Effective Radiative Forcing</td>
<td>11.028571</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>delta_degC</td>
<td>Surface Temperature</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>W/m(^2)</td>
<td>Heat Uptake</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>two_layer</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>surface</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>two_layer</td>
<td>NaN</td>
</tr>
</tbody>
</table>

...
### Two Layer Model Documentation

#### 0.2.3+0.g6e5f9fe.dirty

(continued from previous page)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>W/m²² Heat Uptake</td>
<td>3.278774</td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>400.0</td>
<td>NaN</td>
</tr>
<tr>
<td>NaN</td>
<td>1.0</td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>0.4</td>
</tr>
<tr>
<td>0</td>
<td>1pctCO2</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>1200.0</td>
<td>50.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>1.333333</td>
</tr>
<tr>
<td>0</td>
<td>1pctCO2</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
</tbody>
</table>

#### 2045-01-01 \ 

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model</td>
<td>d1 (a)</td>
</tr>
<tr>
<td>d2 (a) d1 (meter) du (meter) efficacy (dimensionless) et (watt / delta_degree_Celsius</td>
<td></td>
</tr>
<tr>
<td>/ meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model q1 (delta</td>
<td></td>
</tr>
<tr>
<td>_degree_Celsius * meter ** 2 / watt) q2 (delta_degree_Celsius * meter ** 2 / watt)</td>
<td></td>
</tr>
<tr>
<td>region run_idx scenario unit variable</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>1200.0</td>
<td>50.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.8</td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>1.333333</td>
</tr>
<tr>
<td>0</td>
<td>1pctCO2</td>
</tr>
</tbody>
</table>

#### 2046-01-01 \ 

(continues on next page)
0.0 two_layer NaN
  NaN 1200.0 50.0 1.0 0.8
  1.333333 idealised NaN
  0 1pctCO2 W/m^2 Effective Radiative Forcing 11.200000

delta_degC Surface Temperature|Upper 5.914140

Surface Temperature|Lower 2.052465

W/m^2 Heat Uptake 3.302615

2047-01-01 \

0.0 two_layer NaN
  NaN 1200.0 50.0 1.0 0.8
  1.333333 idealised NaN
  0 1pctCO2 W/m^2 Effective Radiative Forcing 11.257143

2.1. Basic demos

Surface Temperature|Lower 2.071897
<table>
<thead>
<tr>
<th>time</th>
<th>2048-01-01</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model</td>
<td>d1 (a)</td>
</tr>
<tr>
<td>d2 (a) dl (meter) du (meter) efficacy (dimensionless)</td>
<td>eta (watt / delta_degree_Celsius / meter ** 2) model</td>
</tr>
<tr>
<td>lambda0 (watt / delta_degree_Celsius / meter ** 2) model</td>
<td>q1 (delta_degree_Celsius * meter ** 2 / watt)</td>
</tr>
<tr>
<td>q2 (delta_degree_Celsius * meter ** 2 / watt)</td>
<td></td>
</tr>
<tr>
<td>region run_idx scenario unit variable</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>two_layer</td>
<td>NaN</td>
</tr>
<tr>
<td>NaN</td>
<td>1.333333</td>
</tr>
<tr>
<td>NaN</td>
<td>0.8</td>
</tr>
<tr>
<td>world</td>
<td>0.4</td>
</tr>
<tr>
<td>0 1pctCO2 W/m^2 Effective Radiative Forcing</td>
<td>11.314286</td>
</tr>
</tbody>
</table>

| delta_degC Surface Temperature|Upper     | 5.982141 |
| Surface Temperature|Lower     | 2.091402 |

<table>
<thead>
<tr>
<th>time</th>
<th>2049-01-01</th>
</tr>
</thead>
<tbody>
<tr>
<td>a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model</td>
<td>d1 (a)</td>
</tr>
<tr>
<td>d2 (a) dl (meter) du (meter) efficacy (dimensionless)</td>
<td>eta (watt / delta_degree_Celsius / meter ** 2) model</td>
</tr>
<tr>
<td>lambda0 (watt / delta_degree_Celsius / meter ** 2) model</td>
<td>q1 (delta_degree_Celsius * meter ** 2 / watt)</td>
</tr>
<tr>
<td>q2 (delta_degree_Celsius * meter ** 2 / watt)</td>
<td></td>
</tr>
<tr>
<td>region run_idx scenario unit variable</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>variable</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>two_layer</td>
<td>NaN</td>
</tr>
<tr>
<td>NaN</td>
<td>1.333333</td>
</tr>
<tr>
<td>NaN</td>
<td>0.8</td>
</tr>
<tr>
<td>world</td>
<td>0.4</td>
</tr>
<tr>
<td>0 1pctCO2 W/m^2 Effective Radiative Forcing</td>
<td>11.314286</td>
</tr>
</tbody>
</table>
Now we can plot our outputs and compare (of course, we can make these two models the same if we're clever about how we set the parameters, see the impulse response equivalence notebook).

```python
res.filter(variable="Surface Temperature*").lineplot(
    hue="climate_model", style="variable"
)
```

```python
<AxesSubplot:xlabel='time', ylabel='delta_degC'>
```
2.1.2 Running scenarios

Here we show how multiple scenarios can be run using the OpenSCM Two Layer Model package.

```python
import os.path
import numpy as np
import pandas as pd
from openscm_units import unit_registry as ur
import tqdm.autonotebook as tqdm
from scmdata import ScmRun, run_append
```
from openscm_twolayermodel import TwoLayerModel

import matplotlib.pyplot as plt

For this we use RCMIP effective radiative forcing data.

```python
DATA_PATH = os.path.join(
    "..",
    "..",
    "..",
    "tests",
    "test-data",
    "rcmip-radiative-forcing-annual-means-v4-0-0.csv",
)
```

```text
DATA_PATH:

'..
```

```python
# NBVAL_IGNORE_OUTPUT
scenarios = ScmRun(DATA_PATH, lowercase_cols=True).filter(
    scenario="historical", keep=False
)
```

```text
scenarios:

<scmdata.ScmRun (timeseries: 480, timepoints: 751)>

Time:

Start: 1750-01-01T00:00:00
End: 2500-01-01T00:00:00

Meta:

<table>
<thead>
<tr>
<th>activity_id</th>
<th>mip_era</th>
<th>model</th>
<th>region</th>
<th>scenario</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>AIM</td>
<td>World</td>
<td>rcp60</td>
</tr>
<tr>
<td>1</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>AIM</td>
<td>World</td>
<td>rcp60</td>
</tr>
<tr>
<td>2</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>AIM</td>
<td>World</td>
<td>rcp60</td>
</tr>
<tr>
<td>3</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>AIM</td>
<td>World</td>
<td>rcp60</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>494</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>unspecified</td>
<td>World</td>
<td>historical-cmip5</td>
</tr>
<tr>
<td>495</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>unspecified</td>
<td>World</td>
<td>historical-cmip5</td>
</tr>
<tr>
<td>496</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>unspecified</td>
<td>World</td>
<td>historical-cmip5</td>
</tr>
<tr>
<td>497</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>unspecified</td>
<td>World</td>
<td>historical-cmip5</td>
</tr>
<tr>
<td>498</td>
<td>not_applicable</td>
<td>CMIP5</td>
<td>unspecified</td>
<td>World</td>
<td>historical-cmip5</td>
</tr>
</tbody>
</table>

variable

| 0 | Radiative Forcing |
| 1 | Radiative Forcing|Anthropogenic |
| 2 | Radiative Forcing|Anthropogenic|Aerosols |
```

(continues on next page)
We can then run them, for a number of parameter settings, as shown.

```python
# NBVAL_IGNORE_OUTPUT
a_values = np.array([0, 0.01]) * ur("W/m^2/delta_degC^2")
a_values
```

```output
(0.0 0.01) watt (delta_degree_Celsius^2 · meter^2)
```

```python
# NBVAL_IGNORE_OUTPUT
runner = TwoLayerModel()
output = []
for a in tqdm(a_values, desc="Parameter settings"):
    runner.a = a
    output.append(runner.run_scenarios(scenarios))
output = run_append(output)
```

```output
Parameter settings: 0%| | 0/2 [00:00<?, ?it/s]
scenarios: 0it [00:00, ?it/s]
scenarios: 0it [00:00, ?it/s]
```

```output
<scmdata.ScmRun (timeseries: 80, timepoints: 751)>
Time:
    Start: 1750-01-01T00:00:00
    End: 2500-01-01T00:00:00
Meta:
a (watt / delta_degree_Celsius ** 2 / meter ** 2) activity_id  
  0 0.00 not_applicable
  1 0.00 not_applicable
  2 0.00 not_applicable
  3 0.00 not_applicable
  4 0.00 not_applicable
  ... ... ... ... ...
  75 0.01 not_applicable
  76 0.01 not_applicable
  77 0.01 not_applicable
  78 0.01 not_applicable
  79 0.01 not_applicable
  climate_model dl (meter) du (meter) efficacy (dimensionless)  
```

(continues on next page)
<table>
<thead>
<tr>
<th>Region</th>
<th>Run_idx</th>
<th>Scenario</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>World</td>
<td>0</td>
<td>ssp370</td>
<td>W/m^2</td>
</tr>
<tr>
<td>World</td>
<td>1</td>
<td>ssp370</td>
<td>delta_degC</td>
</tr>
<tr>
<td>World</td>
<td>2</td>
<td>ssp370</td>
<td>delta_degC</td>
</tr>
<tr>
<td>World</td>
<td>3</td>
<td>ssp370</td>
<td>W/m^2</td>
</tr>
<tr>
<td>World</td>
<td>4</td>
<td>ssp370-lowNTCF-aerchemmip</td>
<td>W/m^2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>World</td>
<td>75</td>
<td>ssp534-over</td>
<td>W/m^2</td>
</tr>
<tr>
<td>World</td>
<td>76</td>
<td>ssp585</td>
<td>W/m^2</td>
</tr>
<tr>
<td>World</td>
<td>77</td>
<td>ssp585</td>
<td>delta_degC</td>
</tr>
<tr>
<td>World</td>
<td>78</td>
<td>ssp585</td>
<td>delta_degC</td>
</tr>
<tr>
<td>World</td>
<td>79</td>
<td>ssp585</td>
<td>W/m^2</td>
</tr>
</tbody>
</table>

2.1. Basic demos
## 0 Effective Radiative Forcing
## 1 Surface Temperature|Upper
## 2 Surface Temperature|Lower
## 3 Heat Uptake
## 4 Effective Radiative Forcing
## ...
## 75 Heat Uptake
## 76 Effective Radiative Forcing
## 77 Surface Temperature|Upper
## 78 Surface Temperature|Lower
## 79 Heat Uptake

[80 rows x 15 columns]

```python
# NBVAL_IGNORE_OUTPUT
pkwargs = dict(
    hue="scenario", style="a (watt / delta_degree_Celsius ** 2 / meter ** 2)"
)
fig = plt.figure(figsize=(12, 18))
ax = fig.add_subplot(211)
output.filter(variable="Surface Temperature|Upper").lineplot(**pkwargs, ax=ax)

ax = fig.add_subplot(212)
output.filter(variable="Heat Uptake").lineplot(**pkwargs, ax=ax)
```

```bash
[6]: <AxesSubplot:xlabel='time', ylabel='W/m^2'>
```
2.1. Basic demos

---

[Image of two graphs showing data trends over time]
2.2 More detail

2.2.1 Impulse response equivalence

In this notebook we explore the equivalence between the two-layer model and a two-timescale impulse response approach.

Background

Following Geoffroy et al., 2013, Part 2, with notation altered to match our implementation, the two-layer model with efficacy ($\epsilon$) and state-dependent climate feedback can be written as

\begin{align}
C \frac{dT}{dt} &= F - (\lambda_0 - aT)T - \epsilon\eta(T - T_D) \\
C_D \frac{dT_D}{dt} &= \eta(T - T_D)
\end{align}

If the state-dependent feedback factor, $a$, is non-zero, the two-layer model and impulse response approaches are not equivalent. However, if $a = 0$, they become the same.

Hereafter we assume $a = 0$, however this assumption should not be forgotten. In the case $a = 0$, the two-layer model can be written (adding an $\epsilon$ for the deep-ocean equation too for simplicity later).

\begin{align}
C \frac{dT}{dt} &= F - \lambda_0T - \epsilon\eta(T - T_D) \\
\epsilon C_D \frac{dT_D}{dt} &= \epsilon\eta(T - T_D)
\end{align}

In matrix notation we have

\[ \frac{dX}{dt} = AX + B \]

where \( X = \begin{pmatrix} T \\ T_D \end{pmatrix} \), \( A = \begin{bmatrix} -\lambda_0 + \epsilon\eta & \frac{\epsilon\eta}{\epsilon C_D} \\ \frac{\epsilon\eta}{\epsilon C_D} & -\frac{\epsilon\eta^2}{\epsilon C_D} \end{bmatrix} \) and \( B = \begin{pmatrix} F \\ 0 \end{pmatrix} \).

As shown in Geoffroy et al., 2013, Part 1, \( A \) can be diagonalised i.e. written in the form \( A = \Phi D \Phi^{-1} \), where \( D \) is a diagonal matrix. Applying the solution given in Geoffroy et al., 2013, Part 1 to our impulse response notation, we have

\[ D = \begin{bmatrix} -1/	au_1 & 0 \\ 0 & -1/	au_2 \end{bmatrix} \]

and

\[ \Phi = \begin{bmatrix} 1 & 1 \\ \phi_1 & \phi_2 \end{bmatrix} \]

where

\[ \tau_1 = \frac{CC_D}{2\lambda_0\eta} (b - \sqrt{\delta}) \]

\[ \tau_2 = \frac{CC_D}{2\lambda_0\eta} (b + \sqrt{\delta}) \]

\[ \phi_1 = \frac{C}{2\epsilon\eta} (b^* - \sqrt{\delta}) \]
\[ \phi_2 = \frac{C}{2e\eta} (b^* + \sqrt{\delta}) \]  

(2.11)

\[ b = \frac{\lambda_0 + e\eta}{C} + \frac{\eta}{C_D} \]  

(2.12)

\[ b^* = \frac{\lambda_0 + e\eta}{C} - \frac{\eta}{C_D} \]  

(2.13)

\[ \delta = b^2 - 4 \frac{\lambda_0 \eta}{CC_D} \]  

(2.14)

Given this, we can re-write the system as

\[ \frac{dX}{dt} = \Phi D \Phi^{-1} X + B \]  

(2.15)

\[ \Phi^{-1} \frac{dX}{dt} = D \Phi^{-1} X + \Phi^{-1} B \]  

(2.16)

\[ \frac{dY}{dt} = DY + \Phi^{-1} B \]  

(2.17)

Defining \( Y = \begin{pmatrix} T_1 \\ T_2 \end{pmatrix} \), we have

\[ \frac{d}{dt} \begin{pmatrix} T_1 \\ T_2 \end{pmatrix} = \begin{bmatrix} -\frac{1}{\tau_1} & 0 \\ 0 & -\frac{1}{\tau_2} \end{bmatrix} \begin{pmatrix} T_1 \\ T_2 \end{pmatrix} + \frac{1}{\phi_2 - \phi_1} \begin{bmatrix} \phi_2 & -1 \end{bmatrix} \begin{pmatrix} F \\ 0 \end{pmatrix} \]  

(2.19)

or,

\[ \frac{dT_1}{dt} = -\frac{T_1}{\tau_1} + \frac{\phi_2}{\phi_2 - \phi_1} \frac{F}{C} \]  

(2.20)

\[ \frac{dT_2}{dt} = -\frac{T_2}{\tau_2} - \frac{\phi_1}{\phi_2 - \phi_1} \frac{F}{C} \]  

(2.21)

Re-writing, we have,

\[ \frac{dT_1}{dt} = \frac{1}{\tau_1} \left( \frac{\tau_1 \phi_2}{\phi_2 - \phi_1} \frac{F}{C} - T_1 \right) \]  

(2.22)

\[ \frac{dT_2}{dt} = \frac{1}{\tau_2} \left( -\tau_2 \phi_1 \frac{F}{\phi_2 - \phi_1} - T_2 \right) \]  

(2.23)

We can compare this to the notation of Millar et al., 2017 and see that

\[ d_1 = \tau_1 \]  

(2.24)

\[ d_2 = \tau_2 \]  

(2.25)

\[ q_1 = \frac{\tau_1 \phi_2}{C(\phi_2 - \phi_1)} \]  

(2.26)

\[ q_2 = -\frac{\tau_2 \phi_1}{C(\phi_2 - \phi_1)} \]  

(2.27)

Hence we have redemonstrated the equivalence of the two-layer model and a two-timescale impulse response model. Given the parameters of the two-layer model, we can now trivially derive the equivalent parameters of the two-timescale model. Doing the reverse is possible, but requires some more work in order to make a useable route drop out.
The first step is to follow Geoffroy et al., 2013, Part 1, and define two extra constants

\[ a_1 = \frac{\phi_2 \tau_1}{C(\phi_2 - \phi_1)} \lambda_0 \]  
\[ a_2 = -\frac{\phi_1 \tau_2}{C(\phi_2 - \phi_1)} \lambda_0 \]  

These constants have the useful property that \( a_1 + a_2 = 1 \) (proof in Appendix A).

From above, we also see that

\[ a_1 = \lambda_0 q_1 \]  
\[ a_2 = \lambda_0 q_2 \]  

Hence

\[ a_1 + a_2 = \lambda_0 q_1 + \lambda_0 q_2 = 1 \]  
\[ \lambda_0 = \frac{1}{q_1 + q_2} \]  

Next we calculate \( C \) via

\[ C = \frac{q_1}{d_1} + \frac{q_2}{d_2} = \frac{\phi_2}{C(\phi_2 - \phi_1)} - \frac{\phi_1}{C(\phi_2 - \phi_1)} = \frac{1}{C} \]  
\[ C = \frac{d_1 d_2}{q_1 d_2 + q_2 d_1} \]  

We then use further relationships from Table 1 of Geoffroy et al., 2013, Part 1 (proof is left to the reader) to calculate the rest of the constants.

Firstly,

\[ \tau_1 a_1 + \tau_2 a_2 = \frac{C + \epsilon C_D}{\lambda_0} \]  
\[ \epsilon C_D = \lambda_0 (\tau_1 a_1 + \tau_2 a_2) - C \]  

and then finally,

\[ \tau_1 a_1 + \tau_2 a_2 = \frac{C + \epsilon C_D}{\lambda_0} \]  
\[ \epsilon \eta = \frac{\epsilon C_D}{\tau_1 a_2 + \tau_2 a_1} \]  

The final thing to notice here is that \( C_D, \epsilon \) and \( \eta \) are not uniquely-defined. This makes sense, as shown by Geoffroy et al., 2013, Part 2, the introduction of the efficacy factor does not alter the behaviour of the system (it is still the same mathematical system) and so it is impossible for simply the two-timescale temperature response to uniquely define all three of these quantities. It can only define the products \( \epsilon C_D \) and \( \epsilon \eta \). Hence when translating from the two-timescale model to the two-layer model with efficacy, an explicit choice for the efficacy must be made. This does not alter the temperature response but it does alter the implied ocean heat uptake of the two-timescale model.

Long story short, when deriving two-layer model parameters from a two-timescale model, one must specify the efficacy.

Given that \( \mathbf{Y} = \Phi^{-1} \mathbf{X} \) i.e. \( \mathbf{X} = \Phi \mathbf{Y} \), we can also relate the impulse response boxes to the two layers.

\[
\begin{pmatrix}
T \\
T_D
\end{pmatrix} = \begin{pmatrix}
1 & 1 \\
\phi_1 & \phi_2
\end{pmatrix}
\begin{pmatrix}
T_1 \\
T_2
\end{pmatrix}
\]  
\[
= \begin{pmatrix}
T_1 + T_2 \\
\phi_1 T_1 + \phi_2 T_2
\end{pmatrix}
\]

Finally, the equivalent of the two-timescale and two-layer models al-
allows us to also calculate the heat uptake of a two-timescale impulse response model. It is given by

\[
\text{Heat uptake} \quad = \quad C \frac{dT}{dt} + C_D \frac{dT_D}{dt} \\
= \quad F - \lambda_0 T + (1 - \epsilon) \eta (T - T_D) \\
= \quad F - \lambda_0 (T_1 + T_2) + (1 - \epsilon) \eta ((1 - \phi_1)T_1 + (1 - \phi_2)T_2) \\
= \quad F - \lambda_0 (T_1 + T_2) - \eta (\epsilon - 1)((1 - \phi_1)T_1 + (1 - \phi_2)T_2)
\]

### Running the code

Here we actually run the two implementations to explore their similarity.

[1]:

```python
import datetime as dt
import numpy as np
import pandas as pd
from openscm_units import unit_registry as ur
from scmdata.run import ScmRun, run_append
from openscm_twolayermodel import ImpulseResponseModel, TwoLayerModel
import matplotlib.pyplot as plt
```

First we define a scenario to run.

[2]:

```python
time = np.arange(1750, 2501)
forcing = 0.05 * np.sin(time / 15 * 2 * np.pi) + 3.0 * time / time.max()
inp = ScmRun(
    data=forcing,
    index=time,
    columns={
        "scenario": "test_scenario",
        "model": "unspecified",
        "climate_model": "junk input",
        "variable": "Effective Radiative Forcing",
        "unit": "W/m^2",
        "region": "World",
    },
)
inp
```

[2]:

```
<scmdata.ScmRun (timeseries: 1, timepoints: 751)>
Time:
Start: 1750-01-01T00:00:00
End: 2500-01-01T00:00:00
Meta:
climate_model    model region    scenario    unit
(continues on next page)
```
Next we run the two-layer model. In order for it to be convertible to a two-timescale model, we must turn state-dependence off (a=0).

```python
[4]: two_layer_config = {
    "du": 55 * ur("m"),
    "efficacy": 1.2 * ur("dimensionless"),
    # "efficacy": 1.0 * ur("dimensionless"),
    "a": 0 * ur("W/m^2/delta_degC^2"),
}
```

```python
[5]: # NBVAL_IGNORE_OUTPUT
twolayer = TwoLayerModel(**two_layer_config)
res_twolayer = twolayer.run_scenarios(inp)
res_twolayer
```

```
0it [00:00, ?it/s]
```

```
<scmdata.ScmRun (timeseries: 4, timepoints: 751)>
Time:
    Start: 1750-01-01T00:00:00
    End: 2500-01-01T00:00:00
Meta:
    a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model \
    0 0.0 two_layer
    1 0.0 two_layer
    2 0.0 two_layer
```

(continues on next page)
3  0.0  two_layer

<table>
<thead>
<tr>
<th></th>
<th>dl (meter)</th>
<th>du (meter)</th>
<th>efficacy (dimensionless)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1200</td>
<td>55</td>
<td>1.2</td>
</tr>
<tr>
<td>1</td>
<td>1200</td>
<td>55</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>1200</td>
<td>55</td>
<td>1.2</td>
</tr>
<tr>
<td>3</td>
<td>1200</td>
<td>55</td>
<td>1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>eta (watt / delta_degree_Celsius / meter ** 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.8</td>
</tr>
<tr>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lambda0 (watt / delta_degree_Celsius / meter ** 2)</th>
<th>model region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.246667</td>
<td>unspecified World</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>run_idx</th>
<th>scenario</th>
<th>unit</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>test_scenario</td>
<td>W/m^2</td>
<td>Effective Radiative Forcing</td>
</tr>
<tr>
<td>1</td>
<td>test_scenario</td>
<td>delta_degC</td>
<td>Surface Temperature</td>
</tr>
<tr>
<td>2</td>
<td>test_scenario</td>
<td>delta_degC</td>
<td>Surface Temperature</td>
</tr>
<tr>
<td>3</td>
<td>test_scenario</td>
<td>W/m^2</td>
<td>Heat Uptake</td>
</tr>
</tbody>
</table>

[6]: `# NBVAL_IGNORE_OUTPUT`
    `fig = plt.figure(figsize=(16, 9))`
    `ax = fig.add_subplot(121)`
    `res_twolayer.filter(variable="*Temperature*").lineplot(hue="variable", ax=ax)`
    `ax = fig.add_subplot(122)`
    `res_twolayer.filter(variable="Heat Uptake").lineplot(hue="variable", ax=ax)`

[6]: `<AxesSubplot:xlabel='time', ylabel='W/m^2'>`
Next we get the parameters with which we get the equivalent impulse response model.

```python
[7]: two_timescale_paras = twolayer.get_impulse_response_parameters()
two_timescale_paras
```

```python
[7]: {
'd1': 103454323.57029569 <Unit('joule / watt')>,
'd2': 11181891933.114195 <Unit('joule / watt')>,
'q1': 0.4465999986742509 <Unit('delta_degree_Celsius * meter ** 2 / watt')>,
'q2': 0.3555390387589074 <Unit('delta_degree_Celsius * meter ** 2 / watt')>,
'efficacy': 1.2 <Unit('dimensionless')>
```

```python
[8]: # NBVAL_IGNORE_OUTPUT

impulse_response = ImpulseResponseModel(**two_timescale_paras)
res_impulse_response = impulse_response.run_scenarios(inp)
res_impulse_response

scenarios: 0it [00:00, ?it/s]
```

```python
[8]: <scmdata.ScmRun (timeseries: 5, timepoints: 751)>

Time:
  Start: 1750-01-01T00:00:00
  End: 2500-01-01T00:00:00

Meta:

<table>
<thead>
<tr>
<th>climate_model</th>
<th>d1 (joule / watt)</th>
<th>d2 (joule / watt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>two_timescale_impulse_response</td>
<td>1.034543e+08</td>
</tr>
<tr>
<td>1</td>
<td>two_timescale_impulse_response</td>
<td>1.034543e+08</td>
</tr>
<tr>
<td>2</td>
<td>two_timescale_impulse_response</td>
<td>1.034543e+08</td>
</tr>
<tr>
<td>3</td>
<td>two_timescale_impulse_response</td>
<td>1.034543e+08</td>
</tr>
<tr>
<td>4</td>
<td>two_timescale_impulse_response</td>
<td>1.034543e+08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>efficacy (dimensionless)</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

(continues on next page)
0 1.2 unspecified
1 1.2 unspecified
2 1.2 unspecified
3 1.2 unspecified
4 1.2 unspecified

\[
q_1 \text{ (delta_degree_Celsius} \times \text{meter}^2 / \text{watt)} \ \ \\
0 \quad 0.4466 \ \\
1 \quad 0.4466 \ \\
2 \quad 0.4466 \ \\
3 \quad 0.4466 \ \\
4 \quad 0.4466
\]

\[
q_2 \text{ (delta_degree_Celsius} \times \text{meter}^2 / \text{watt)} \text{ region run_idx} \ \\
0 \quad 0.355539 \text{ World 0} \ \\
1 \quad 0.355539 \text{ World 0} \ \\
2 \quad 0.355539 \text{ World 0} \ \\
3 \quad 0.355539 \text{ World 0} \ \\
4 \quad 0.355539 \text{ World 0}
\]

**scenario unit variable**

<table>
<thead>
<tr>
<th>0</th>
<th>test_scenario</th>
<th>W/m^2</th>
<th>Effective Radiative Forcing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>test_scenario</td>
<td>delta_degC</td>
<td>Surface Temperature</td>
</tr>
<tr>
<td>2</td>
<td>test_scenario</td>
<td>delta_degC</td>
<td>Surface Temperature</td>
</tr>
<tr>
<td>3</td>
<td>test_scenario</td>
<td>delta_degC</td>
<td>Surface Temperature</td>
</tr>
<tr>
<td>4</td>
<td>test_scenario</td>
<td>W/m^2</td>
<td>Heat Uptake</td>
</tr>
</tbody>
</table>

![Figure](image_url)

```python
# NBVAL_IGNORE_OUTPUT
fig = plt.figure(figsize=(16, 9))

ax = fig.add_subplot(121)
res_impulse_response.filter(variable="*Temperature*").lineplot(
    hue="variable", ax=ax
)

ax = fig.add_subplot(122)
res_impulse_response.filter(variable="Heat Uptake").lineplot(
    hue="variable", ax=ax
)
```

```python
 AXES: subplot: xlabel='time', ylabel='W/m^2'
```
We can compare the two responses as well.

```python
combined = run_append([res_impulse_response, res_twolayer])
combined
```

```
<scmdata.ScmRun (timeseries: 9, timepoints: 751)>
```

Time:
```
Start: 1750-01-01T00:00:00
End: 2500-01-01T00:00:00
```

Meta:
```
 climate_model | d1 (joule / watt) | d2 (joule / watt) |
----------------|------------------|------------------|
   0           | 1.034543e+08     | 1.118189e+10     |
   1           | 1.034543e+08     | 1.118189e+10     |
   2           | 1.034543e+08     | 1.118189e+10     |
   3           | 1.034543e+08     | 1.118189e+10     |
   4           | 1.034543e+08     | 1.118189e+10     |
   5           | 0.0              | NaN              |
   6           | 0.0              | NaN              |
   7           | 0.0              | NaN              |
   8           | 0.0              | NaN              |
```

(continues on next page)
<table>
<thead>
<tr>
<th></th>
<th>two_layer</th>
<th>NaN</th>
<th>NaN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NaN</td>
<td>NaN</td>
<td>1.2</td>
</tr>
<tr>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>NaN</td>
<td>NaN</td>
<td>1.2</td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
<td>NaN</td>
<td>1.2</td>
</tr>
<tr>
<td>4</td>
<td>NaN</td>
<td>NaN</td>
<td>1.2</td>
</tr>
<tr>
<td>5</td>
<td>1200.0</td>
<td>55.0</td>
<td>1.2</td>
</tr>
<tr>
<td>6</td>
<td>1200.0</td>
<td>55.0</td>
<td>1.2</td>
</tr>
<tr>
<td>7</td>
<td>1200.0</td>
<td>55.0</td>
<td>1.2</td>
</tr>
<tr>
<td>8</td>
<td>1200.0</td>
<td>55.0</td>
<td>1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>eta (watt / delta_degree_Celsius / meter ** 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NaN</td>
</tr>
<tr>
<td>1</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>NaN</td>
</tr>
<tr>
<td>3</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>0.8</td>
</tr>
<tr>
<td>6</td>
<td>0.8</td>
</tr>
<tr>
<td>7</td>
<td>0.8</td>
</tr>
<tr>
<td>8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>lambda0 (watt / delta_degree_Celsius / meter ** 2) model</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NaN unspecified</td>
</tr>
<tr>
<td>1</td>
<td>NaN unspecified</td>
</tr>
<tr>
<td>2</td>
<td>NaN unspecified</td>
</tr>
<tr>
<td>3</td>
<td>NaN unspecified</td>
</tr>
<tr>
<td>4</td>
<td>NaN unspecified</td>
</tr>
<tr>
<td>5</td>
<td>1.246667 unspecified</td>
</tr>
<tr>
<td>6</td>
<td>1.246667 unspecified</td>
</tr>
<tr>
<td>7</td>
<td>1.246667 unspecified</td>
</tr>
<tr>
<td>8</td>
<td>1.246667 unspecified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>q1 (delta_degree_Celsius * meter ** 2 / watt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.4466</td>
</tr>
<tr>
<td>1</td>
<td>0.4466</td>
</tr>
<tr>
<td>2</td>
<td>0.4466</td>
</tr>
<tr>
<td>3</td>
<td>0.4466</td>
</tr>
<tr>
<td>4</td>
<td>0.4466</td>
</tr>
<tr>
<td>5</td>
<td>NaN</td>
</tr>
<tr>
<td>6</td>
<td>NaN</td>
</tr>
<tr>
<td>7</td>
<td>NaN</td>
</tr>
<tr>
<td>8</td>
<td>NaN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>q2 (delta_degree_Celsius * meter ** 2 / watt) region run_idx</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.355539 World 0</td>
</tr>
<tr>
<td>1</td>
<td>0.355539 World 0</td>
</tr>
<tr>
<td>2</td>
<td>0.355539 World 0</td>
</tr>
<tr>
<td>3</td>
<td>0.355539 World 0</td>
</tr>
</tbody>
</table>
To within numerical errors they are equal.

```python
# NBVAL_IGNORE_OUTPUT
fig = plt.figure(figsize=(16, 9))

ax = fig.add_subplot(121)
combined.filter(variable="*Temperature*`).lineplot(
    hue="variable", style="climate_model", alpha=0.7, linewidth=2, ax=ax
)
ax.legend(loc="upper left")

ax = fig.add_subplot(122)
combined.filter(variable="Heat Uptake`).lineplot(
    hue="climate_model", style="climate_model", alpha=0.7, linewidth=2, ax=ax
)
```

[11]: <AxesSubplot:xlabel='time', ylabel='W/m^2'>
Appendix A

We begin with the definitions of the $\alpha$ constants,

$$\alpha_1 = \frac{\phi_2 \tau_1}{C(\phi_2 - \phi_1)} \lambda_0$$  \hspace{1cm} (2.43)

$$\alpha_2 = -\frac{\phi_1 \tau_2}{C(\phi_2 - \phi_1)} \lambda_0$$  \hspace{1cm} (2.44)

We then have

$$\alpha_1 + \alpha_2 = \frac{\phi_2 \tau_1}{C(\phi_2 - \phi_1)} \lambda_0 - \frac{\phi_1 \tau_2}{C(\phi_2 - \phi_1)} \lambda_0$$  \hspace{1cm} (2.46)

$$= \frac{\lambda_0}{C(\phi_2 - \phi_1)} (\phi_2 \tau_1 - \phi_1 \tau_2)$$  \hspace{1cm} (2.47)

Recalling the definition of the $\phi$ parameters,

$$\phi_1 = \frac{C}{2e \eta} (b^* - \sqrt{\delta})$$  \hspace{1cm} (2.48)

$$\phi_2 = \frac{C}{2e \eta} (b^* + \sqrt{\delta})$$  \hspace{1cm} (2.49)

We have,

$$\phi_2 - \phi_1 = \frac{C \sqrt{\delta}}{e \eta}$$  \hspace{1cm} (2.50)
Recalling the definition of the \( \tau \) parameters,

\[
\tau_1 = \frac{CC_D}{2\lambda_0\eta} (b - \sqrt{\delta})
\]

(2.51)

\[
\tau_2 = \frac{CC_D}{2\lambda_0\eta} (b + \sqrt{\delta})
\]

(2.52)

We have,

\[
\phi_2 \tau_1 - \phi_1 \tau_2 = \frac{C}{2\epsilon\eta} (b^* + \sqrt{\delta}) \times \frac{CC_D}{2\lambda_0\eta} (b - \sqrt{\delta}) - \frac{C}{2\epsilon\eta} (b^* - \sqrt{\delta}) \times \frac{CC_D}{2\lambda_0\eta} (b + \sqrt{\delta})
\]

(2.53)

\[
= \frac{C^2 C_d}{4\epsilon\eta^2\lambda_0} [(b^* + \sqrt{\delta})(b - \sqrt{\delta}) - (b^* - \sqrt{\delta})(b + \sqrt{\delta})]
\]

(2.54)

\[
= \frac{C^2 C_d}{4\epsilon^2\lambda_0} [b^* b + b\sqrt{\delta} - b^* \sqrt{\delta} - \delta - bb^* + b\sqrt{\delta} - b^* \sqrt{\delta} + \delta]
\]

(2.55)

\[
= \frac{C^2 C_d \sqrt{\delta} - b^* \sqrt{\delta}}{2\epsilon\eta^2\lambda_0}
\]

(2.56)

\[
= \frac{C^2 C_d \sqrt{\delta} - b^* \sqrt{\delta}}{2\epsilon\eta^2 \lambda_0} [b - b^*]
\]

(2.57)

Recalling the definition of the \( b \) parameters,

\[
b = \frac{\lambda_0 + \epsilon \eta}{C} + \frac{\eta}{C_D}
\]

(2.58)

\[
b^* = \frac{\lambda_0 + \epsilon \eta}{C} - \frac{\eta}{C_D}
\]

(2.59)

We then have

\[
\phi_2 \tau_1 - \phi_1 \tau_2 = \frac{C^2 C_d \sqrt{\delta}}{2\epsilon\eta^2\lambda_0} \left[ \frac{2\eta}{C_D} \right]
\]

(2.60)

\[
= \frac{C^2 \sqrt{\delta}}{\epsilon \eta \lambda_0}
\]

(2.61)

Putting it all back together,

\[
a_1 + a_2 = \frac{\lambda_0}{C(\phi_2 - \phi_1)} (\phi_2 \tau_1 - \phi_1 \tau_2)
\]

(2.62)

\[
= \frac{\lambda_0 \epsilon \eta}{C \sqrt{\delta}} \frac{C^2 \sqrt{\delta}}{\epsilon \eta \lambda_0}
\]

(2.63)

\[
= 1
\]

(2.64)

### 2.2.2 One layer model

Here we show how to run our two-layer model as a single-layer model. There are two different ways to do this, which we present below.
Imports and loading data

```
[1]: # NBVAL_IGNORE_OUTPUT
    import os.path

    import numpy as np
    import pandas as pd
    from openscm_units import unit_registry as ur
    import tqdm.autonotebook as tqdm
    from scmdata import ScmRun, run_append

    from openscm_twolayermodel import TwoLayerModel

    import matplotlib.pyplot as plt

/home/docs/checkouts/readthedocs.org/user_builds/openscm-two-layer-model/envs/stable/lib/
...python3.7/site-packages/ipykernel_launcher.py:7: TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force console mode (e.g. in jupyter console)
    import sys
```

For this we use an idealised scenario which is a reasonable representation of the forcing which occurs in response to an abrupt doubling in atmospheric CO\(_2\) concentrations (often referred to as an abrupt-2xCO\(_2\) experiment).

```
[2]: run_length = 2000

    data = np.zeros(run_length)
    data[10 :] = 4.0

    driver = ScmRun(
        data=data,
        index=1850 + np.arange(run_length),
        columns={
            "unit": "W/m^2",
            "model": "idealised",
            "scenario": "1pctCO2",
            "region": "World",
            "variable": "Effective Radiative Forcing",
        },
    )
    driver
```

```
[2]: <scmdata.ScmRun (timeseries: 1, timepoints: 2000)>

    Time:
    Start: 1850-01-01T00:00:00
    End: 3849-01-01T00:00:00

    Meta:
    model region scenario unit variable
    0 idealised World lpctCO2 W/m^2 Effective Radiative Forcing
```

```
[3]: # NBVAL_IGNORE_OUTPUT
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111)
    driver.filter(variable="Effective Radiative Forcing").lineplot()
```
No second layer

The first, and arguably the simplest way, to make a single layer model is to simply remove the connection between the top and second layers. Recalling the equations which define the two-layer model below,

\[ C \frac{dT}{dt} = F - (\lambda_0 - aT)T - \epsilon \eta (T - T_D) \tag{2.65} \]
\[ C_D \frac{dT_D}{dt} = \eta (T - T_D) \tag{2.66} \]

We see that we can effectively remove the second layer by setting \( \eta = 0 \).
(50 250 500) meter

# NBVAL_IGNORE_OUTPUT
runner = TwoLayerModel()
output = []
equivalent_parameters = []
for eta in tqdm(eta_values, desc="eta values", leave=False):
    runner.eta = eta
    for du in tqdm(du_values, desc="du values", leave=False):
        runner.du = du
        output.append(runner.run_scenarios(driver))
output = run_append(output)
output.head()
eta values:  0% |  0/2  [00:00<?, ?it/s]
du values:  0% |  0/3  [00:00<?, ?it/s]
scenarios:  0it [00:00, ?it/s]
scenarios:  0it [00:00, ?it/s]
scenarios:  0it [00:00, ?it/s]
du values:  0% |  0/3  [00:00<?, ?it/s]
scenarios:  0it [00:00, ?it/s]
scenarios:  0it [00:00, ?it/s]
scenarios:  0it [00:00, ?it/s]

```
time
ger=1850-01-01 00:00:00 a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
  efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable
0.0 1200 50 1.0 two_layer idealised World 0 1pctCO2 W/m^2 Effective Radiative
  Forcing 0.0

  delta_degC Surface

  Temperature|Upper 0.0

  Temperature|Lower 0.0

  W/m^2 Heat Uptake 250 1.0

  idealised World 0 1pctCO2 W/m^2 Effective Radiative
  Forcing 0.0
```

2.2. More detail
(continued from previous page)

```plaintext
 time

1851-01-01 00:00:00 "a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter) efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable

0.0 two_layer 1200 50 1.0

0.0 0.0 1.246667 idealised World 0 1pctCO2 W/m^2 Effective Radiative

Forcing 0.0

delta_degC Surface

Temperature|Upper 0.0

Temperature|Lower 0.0

W/m^2 Heat Uptake

0.0

250 1.0

0.0 1.246667 idealised World 0 1pctCO2 W/m^2 Effective Radiative

Forcing 0.0

delta_degC Surface

Temperature|Upper 0.0

Temperature|Lower 0.0

W/m^2 Heat Uptake

0.0
```

(continues on next page)
2.2. More detail

```plaintext
(continued from previous page)

<table>
<thead>
<tr>
<th>Start Year</th>
<th>Time</th>
<th>Climate Model</th>
<th>Upper Layer Depth</th>
<th>Lower Layer Depth</th>
<th>Effective Radiative Forcing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1853-01-01</td>
<td>0.0</td>
<td>idealised World</td>
<td>0.0</td>
<td>1.246667</td>
<td>0.0</td>
</tr>
<tr>
<td>1854-01-01</td>
<td>0.0</td>
<td>idealised World</td>
<td>0.0</td>
<td>1.246667</td>
<td>0.0</td>
</tr>
</tbody>
</table>

250 1.0

---

(continues on next page)
```
\[ \text{Heat Uptake} = 250 \times 1. \]

\[ \text{Effective Radiative Forcing} = 0.0 \]

\[ \text{Idealised World} = 0 \]

\[ \text{1pctCO2} \]

\[ \text{Effective} \]

\[ \text{Radiative} \]

\[ \text{Forcing} = 0.0 \]

\[ \text{Temperature|Upper} = 0.0 \]

\[ \text{Temperature|Lower} = 0.0 \]

\[ \text{Heat Uptake} = 250 \times 1. \]

\[ \text{Effective Radiative Forcing} = 0.0 \]

\[ \text{Idealised World} = 0 \]

\[ \text{1pctCO2} \]

\[ \text{Effective} \]

\[ \text{Radiative} \]

\[ \text{Forcing} = 0.0 \]

\[ \text{Temperature|Upper} = 0.0 \]

\[ \text{Temperature|Lower} = 0.0 \]

\[ \text{Heat Uptake} = 250 \times 1. \]

\[ \text{Effective Radiative Forcing} = 0.0 \]

\[ \text{Idealised World} = 0 \]

\[ \text{1pctCO2} \]

\[ \text{Effective} \]

\[ \text{Radiative} \]

\[ \text{Forcing} = 0.0 \]

\[ \text{Temperature|Upper} = 0.0 \]
(continued from previous page)

Surface Temp | Lower 0.0

W/m^2 Heat Uptake 0.0

250 1.

idealised World

Forcing 0.0

time

1857-01-01 00:00:00

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)

efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable

0.0 two_layer 1200 50 1.

idealised World

Forcing 0.0

delta_degC Surface

Surface Temp | Upper 0.0

Surface Temp | Lower 0.0

W/m^2 Heat Uptake 0.0

250 1.

idealised World

Forcing 0.0

time

1858-01-01 00:00:00

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)

efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable

0.0 two_layer 1200 50 1.

idealised World

Forcing 0.0

(continues on next page)
delta_degC Surface
  Temperature|Upper  0.0

Surface
  Temperature|Lower  0.0

W/m^2 Heat Uptake
  0.0

250 1.0

0.0 0.0 1.0

idealised World 0 1pctCO2 W/m^2 Effective Radiative

Forcing 0.0

time

1859-01-01 00:00:00 \ a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
  efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable
0.0 two_layer 1200 50 1.0

0.0 0.0 1.0

idealised World 0 1pctCO2 W/m^2 Effective Radiative

Forcing 0.0

delta_degC Surface
  Temperature|Upper  0.0

Surface
  Temperature|Lower  0.0

W/m^2 Heat Uptake
  0.0

250 1.0

0.0 0.0 1.0

idealised World 0 1pctCO2 W/m^2 Effective Radiative

Forcing 0.0

time

... \ a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
  efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable

...
0.0 two_layer 1200 50 1.
→ 0 0.0 1.246667 ←
→ idealised World 0 1pctCO2 W/m^2 Effective Radiative ...
→ Forcing ...

delta_degC Surface ...

Temperature|Upper ...

Temperature|Lower ...

W/m^2 Heat Uptake ...

0 0.0 1.246667 ←
→ idealised World 0 1pctCO2 W/m^2 Effective Radiative ...
→ Forcing 4.000000

time ...

3840-01-01 00:00:00 \a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter) efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable 0.0 two_layer 1200 50 1.
→ 0 0.0 1.246667 ←
→ idealised World 0 1pctCO2 W/m^2 Effective Radiative ...
→ Forcing 4.000000

delta_degC Surface ...

Temperature|Upper 3.208556

Temperature|Lower 0.000000

W/m^2 Heat Uptake ...

0.000000 ←
→ 250 1. ←
→ idealised World 0 1pctCO2 W/m^2 Effective Radiative ...
→ Forcing 4.000000

time ...

3841-01-01 00:00:00 \a (continues on next page)
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable
0.0 two_layer 1200 50 1.
0 0.0 1.246667
idealised World 0 1pctCO2 W/m^2 Effective Radiative
Forcing 4.000000

delta_degC Surface
Temperature|Upper 3.208556

Surface
Temperature|Lower 0.000000

W/m^2 Heat Uptake
0.000000 250 1.
0 0.0 1.246667
idealised World 0 1pctCO2 W/m^2 Effective Radiative
Forcing 4.000000
time

3842-01-01 00:00:00 \n
(continues on next page)
(continued from previous page)

time

3843-01-01 00:00:00 \n
(a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable
0.0 two_layer 1200 50 1.

0 0.0 1.246667

idealised World 0 lpc CO2 W/m^2 Effective Radiative

Forcing 4.000000

delta_degC Surface

Temperature|Upper 3.208556

Temperature|Lower 0.000000

W/m^2 Heat Uptake

0.000000

250 1.

0 0.0 1.246667

idealised World 0 lpc CO2 W/m^2 Effective Radiative

Forcing 4.000000

delta_degC Surface

Temperature|Upper 3.208556

Temperature|Lower 0.000000

W/m^2 Heat Uptake

0.000000

250 1.

0 0.0 1.246667

idealised World 0 lpc CO2 W/m^2 Effective Radiative

Forcing 4.000000

2.2. More detail
(continued from previous page)

time

```
3845-01-01 00:00:00 \n a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter).
  efficacy (dimensionless) eta (watt / kelvin / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable
  0.0 two_layer 1200 50 1.
  0 0.0
  idealised World 0 lperc02 W/m^2 Effective Radiative
  Forcing
  4.000000
```

```
 delta_degC Surface
```

```
 Temperature|Upper  3.208556
```

```
 Temperature|Lower  0.000000
```

```
 W/m^2 Heat Uptake
```

```
  0.000000
```

```
  250 1.
```

```
  1.246667
```

```
 idealised World 0 lperc02 W/m^2 Effective Radiative
```

```
 Forcing
```

```
  4.000000
```

(continues on next page)
2.2. More detail

<table>
<thead>
<tr>
<th>time</th>
<th>dl (meter)</th>
<th>du (meter)</th>
<th>efficacy (dimensionless)</th>
<th>eta (watt / kelvin / meter ** 2)</th>
<th>lambda0 (watt / delta_degree_Celsius / meter ** 2)</th>
<th>model</th>
<th>region</th>
<th>run_idx</th>
<th>scenario</th>
<th>unit</th>
<th>variable</th>
<th>Heat Uptake</th>
</tr>
</thead>
<tbody>
<tr>
<td>3847-01-01 00:00:00</td>
<td>0.0</td>
<td>1.246667</td>
<td>0.0</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.0</td>
<td>1.246667</td>
<td>0.0</td>
<td>1.246667</td>
<td>4.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>Temperature</td>
<td>Upper</td>
<td>3.208556</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>Lower</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
As we can see in the plots below, the runs with $\eta = 0$ only have a single timescale in their response. In contrast, the runs with $\eta \neq 0$ have two clear, distinct timescales. Notably, because equilibrium warming is independent of ocean heat uptake, the equilibrium warming is the same in all cases.

As expected, we see that the depth of the mixed-layer affects the response time of the mixed-layer (the only response time in the case of $\eta = 0$) whilst having a much smaller effect on the response time of the deep ocean.

```python
# NBVAL_IGNORE_OUTPUT
scenario_to_plot = "1pctCO2"
xlim = [1850, 3500]
kwarg = dict(hue="du (meter)",
            style="eta (watt / kelvin / meter ** 2)",
            time_axis="year"
```

[5 rows x 2000 columns]
fig = plt.figure(figsize=(9, 9))

ax = fig.add_subplot(211)
output.filter(scenario=scenario_to_plot, variable="Surface Temperature|Upper").
    → lineplot(**pkwargs, ax=ax)
ax.set_title("Surface Temperature|Upper")

ax = fig.add_subplot(212, sharex=ax)
output.filter(scenario=scenario_to_plot, variable="Heat Uptake").lineplot(**pkwargs,
    → ax=ax)
ax.set_title("Heat Uptake")
ax.set_xlim(xlim)

plt.tight_layout()
Infinite reservoir second layer

If we make the deep ocean component of the two-layer model infinitely deep, then we also have a single layer model. The concept is described by Equation 4 of Geoffroy et al. 2013, Part 1.

\[
\frac{C}{\rho} \frac{dT}{dt} = F - (\lambda_0 - aT)T - \epsilon\eta(T - T_D)
\] 
\[
\frac{C_D}{\rho} \frac{dT_D}{dt} = \eta(T - T_D)
\]

In short, if \(C_D \to \infty\), then \(T_D = 0\) and the equation governing the mixed layer response becomes

\[
\frac{C}{\rho} \frac{dT}{dt} = F - (\lambda_0 - aT)T - \epsilon\eta T
\]
In effect, we alter the climate feedback factor from $\lambda_0 - aT$ to $\lambda_0 - aT + \epsilon\eta$ we increase the climate feedback factor and hence lower the equilibrium climate sensitivity.

[9]: `# NBVAL_IGNORE_OUTPUT
dl_values = np.array([10 ** 3, 10 ** 4, 10 ** 5, 10 ** 15]) * ur("m")
dl_values

[9]: (1000 10000 100000 1000000000000000) meter

[10]: `# NBVAL_IGNORE_OUTPUT
runner = TwoLayerModel()
output = []
equivalent_parameters = []
for dl in tqdmman.tqdm(dl_values, desc="dl values", leave=False):
    runner.dl = dl
    output.append(runner.run_scenarios(driver))
equivalent_parameters.append(("two-layer deep ocean depth": runner.dl}, runner.get_
impulse_response_parameters()))
output = run_append(output)
output.head()
dl values: 0% | 0/4 [00:00<?, ?it/s]
scenarios: 0it [00:00, ?it/s]
scenarios: 0it [00:00, ?it/s]
scenarios: 0it [00:00, ?it/s]
scenarios: 0it [00:00, ?it/s]
[10]: `time
delta_degC Surface
  0.0
---
  Temperature|Upper
  0.0
  Temperature|Lower
  0.0
---
  Uptake
  0.0
  W/m^2 Heat
---
  Effective Radiative Forcing
  0.0
  (continues on next page)
(continued from previous page)

```
time
   → 1851-01-01 00:00:00 \ 
  a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)→ efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt /→ delta_degree_Celsius / meter ** 2) model region run_idx scenario unit → variable
  0.0 two_layer 1000 50 1.0
  0 0.8 1.246667 → idealised World 0 1pctCO2 W/m^2 ␣ → Effective Radiative Forcing 0.0 ␣ → delta_degC Surface
  → Temperature|Upper 0.0 ␣ → Surface
  → Temperature|Lower 0.0 ␣ → W/m^2 Heat
  → Uptake 0.0 → Effective Radiative Forcing 0.0 ␣ → delta_degC Surface
  → Temperature|Upper 0.0 ␣ → Surface
  → Temperature|Lower 0.0 ␣ → W/m^2 Heat
  → Uptake 0.0 +1852-01-01 00:00:00 \ 
  a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)→ efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt /→ delta_degree_Celsius / meter ** 2) model region run_idx scenario unit → variable
  0.0 two_layer 1000 50 1.0
  0 0.8 1.246667 → idealised World 0 1pctCO2 W/m^2 ␣ → Effective Radiative Forcing 0.0 ␣ → delta_degC Surface
  → Temperature|Upper 0.0 ␣ → Surface
  → Temperature|Lower 0.0 ␣ → W/m^2 Heat
  → Uptake 0.0 ␣ (continued on next page)
```
OpenSCM Two Layer Model Documentation, Release 0.2.3+0.g6e5f9fe.dirty

(continued from previous page)

```
0 0.8 idealised World 0 1pctCO2 W/m²²
  Effective Radiative Forcing 0.0

time
  1853-01-01 00:00:00
  a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
  efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit
  variable
  0.0 two_layer 1000 50 1.
  0 0.8 idealised World 0 1pctCO2 W/m²²
  1854-01-01 00:00:00
  a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
  efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit
  variable
  0.0 two_layer 1000 50 1.
  0 0.8 idealised World 0 1pctCO2 W/m²²
```

(continues on next page)

2.2. More detail 59
W/m² Heat uptake 0.0
10000 50 1.
0.8
1.246667
idealised World 0 1pctCO2 W/m²
Effective Radiative Forcing 0.0
time

1855-01-01 00:00:00
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt /
delta_degree_Celsius / meter ** 2) model region run_idx scenario unit
variable
0.0 two_layer 1000 50 1.
0.8 1.246667
idealised World 0 1pctCO2 W/m²
Effective Radiative Forcing 0.0
delta_degC Surface

Temperature|Upper 0.0

Temperature|Lower 0.0
W/m² Heat uptake 0.0
10000 50 1.
0.8 1.246667
idealised World 0 1pctCO2 W/m²
Effective Radiative Forcing 0.0
time

1856-01-01 00:00:00
a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt /
delta_degree_Celsius / meter ** 2) model region run_idx scenario unit
variable
0.0 two_layer 1000 50 1.
0.8 1.246667
idealised World 0 1pctCO2 W/m²
Effective Radiative Forcing 0.0
delta_degC Surface
Temperature|Upper 0.0
Surface Temperature|Lower 0.0

W/m^2 Heat Uptake 0.0 10000 50 1.

0 0.8 1.246667 idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 0.0

time

1857-01-01 00:00:00 \\

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable 0.0 two_layer 1000 50 1.

0 0.8 1.246667 idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 0.0

delta_degC Surface

Temperature|Upper 0.0

Surface

Temperature|Lower 0.0

W/m^2 Heat Uptake 0.0 10000 50 1.

0 0.8 1.246667 idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 0.0

time

1858-01-01 00:00:00 \\

a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)
efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit variable 0.0 two_layer 1000 50 1.

0 0.8 1.246667 idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 0.0

(continues on next page)
delta_degC Surface

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Upper</th>
<th>0.0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Temperature</th>
<th>Lower</th>
<th>0.0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Uptake</th>
<th>0.0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>W/m^2 Heat</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>W/m^2 Heat</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Effective Radiative Forcing</th>
<th>0.0</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>time</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>1859-01-01 00:00:00</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>variable</th>
</tr>
</thead>
</table>

| 0.0 two_layer 1000 50 1. |
| --- | --- | --- | --- |

| 0 0.8 1.246667 |
| --- | --- | --- |

| idealised World 0 1pctCO2 W/m^2 |
| --- | --- |

| Effective Radiative Forcing 0.0 |
| --- | --- |

<table>
<thead>
<tr>
<th>time</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>...</th>
</tr>
</thead>
</table>
0.0   two_layer  1000 50 1.
0     0.8   1.246667
idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing ...

Temperature|Upper ...

Temperature|Lower ...

Uptake ...

W/m^2 Heat...

do(10000 50 1. 0 0.8 1.246667)
idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing ...

time

3840-01-01 00:00:00 \(a (watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)\)

efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt / delta_degree_Celsius / meter ** 2) model region run_idx scenario unit ...

variable

0.0   two_layer  1000 50 1.
0     0.8   1.246667
idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 4.000000

do(10000 50 1. 0 0.8 1.246667)
idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing ...

time

3841-01-01 00:00:00 \(continues on next page\)
variable
0.0  two_layer  1000  50  1.0
 0    0.8     1.246667
→  idealised World 0 1pctCO2 W/m^2
→  Effective Radiative Forcing 4.000000

... (continued on next page)
time

3843-01-01 00:00:00 \n
*(watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)*

efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt /

delta_degree_Celsius / meter ** 2) model region run_idx scenario unit

variable

0.0 two_layer 1000 50 1.
0 0.8 1.246667

idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 4.000000

Temperature|Upper 3.207645

Temperature|Lower 3.206252

Uptake 0.001140

W/m^2 Heat

3844-01-01 00:00:00 \n
*(watt / delta_degree_Celsius ** 2 / meter ** 2) climate_model dl (meter) du (meter)*

efficacy (dimensionless) eta (watt / delta_degree_Celsius / meter ** 2) lambda0 (watt /

delta_degree_Celsius / meter ** 2) model region run_idx scenario unit

variable

0.0 two_layer 1000 50 1.
0 0.8 1.246667

idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 4.000000

Temperature|Upper 3.207648

Temperature|Lower 3.206261

Uptake 0.001136

W/m^2 Heat

2.2. More detail

(continues on next page)
Effective Radiative Forcing

0.0 two_layer 1000 50 1.
0.0 0.8 idealised World 0 1pctCO2 W/m^2

Effective Radiative Forcing 4.000000

delta_degC Surface

temperature 3.207652

Surface Temperature

Temperature|Lower 3.206269

Uptake 0.001132

Effective Radiative Forcing 4.000000

delta_degC Surface

Temperature|Upper 3.207655

Surface Temperature

Temperature|Lower 3.206278
2.2. More detail
As we can see below, as the deep ocean becomes deeper and deeper, its equivalent timescale increases. This demonstrates that the deep ocean is becoming increasingly close to being an infinite reservoir.

[11]: for v in equivalent_parameters:
    v[1]["d1"] = v[1]["d1"].to("yr")
    v[1]["d2"] = v[1]["d2"].to("yr")
As shown in the plots below, as the deep ocean becomes bigger, it can uptake more heat and hence mixed-layer warming is reduced. However, whilst the deep ocean is finite, the mixed-layer warming does eventually reach the same equilibrium (independent of deep ocean depth), it just takes longer to do so. Once the deep ocean becomes infinite, as discussed above, we effectively have a single-layer model with an increased climate feedback factor (lower equilibrium climate sensitivity) which can uptake heat forever.

```python
scenario_to_plot = "1pctCO2"
xlim = [1850, 3500]
pkwargs = dict(
    hue="dl (meter)",
    style="variable",
    time_axis="year"
)
fig = plt.figure(figsize=(9, 9))
ax = fig.add_subplot(211)
output.filter(scenario=scenario_to_plot, variable="Surface Temperature|Upper").lineplot(**pkwargs, ax=ax)
ax = fig.add_subplot(212, sharex=ax)
output.filter(scenario=scenario_to_plot, variable="Heat Uptake").lineplot(**pkwargs, ax=ax)
ax.set_xlim(xlim)
```

```
(1850.0, 3500.0)
```
If you’re interested in contributing to OpenSCM Two Layer Model, we’d love to have you on board! This section of the docs details how to get setup to contribute and how best to communicate.

- **Contributing**
- **Getting setup**
  - Getting help
    - Development tools
    - Other tools
- **Formatting**
- **Building the docs**
  - Gotchas
  - Docstring style
- **Releasing**
  - First step
  - Push to repository
- **Why is there a Makefile in a pure Python repository?**

### 3.1 Contributing

All contributions are welcome, some possible suggestions include:

- tutorials (or support questions which, once solved, result in a new tutorial :D)
- blog posts
- improving the documentation
- bug reports
- feature requests
- pull requests

Please report issues or discuss feature requests in the OpenSCM Two Layer Model issue tracker. If your issue is a feature request or a bug, please use the templates available, otherwise, simply open a normal issue.
As a contributor, please follow a couple of conventions:

- Create issues in the OpenSCM Two Layer Model issue tracker for changes and enhancements, this ensures that everyone in the community has a chance to comment.
- Be welcoming to newcomers and encourage diverse new contributors from all backgrounds: see the Python Community Code of Conduct.
- Only push to your own branches, this allows people to force push to their own branches as they need without fear or causing others headaches.
- Start all pull requests as draft pull requests and only mark them as ready for review once they’ve been rebased onto master, this makes it much simpler for reviewers.
- Try and make lots of small pull requests, this makes it easier for reviewers and faster for everyone as review time grows exponentially with the number of lines in a pull request.

### 3.2 Getting setup

To get setup as a developer, we recommend the following steps (if any of these tools are unfamiliar, please see the resources we recommend in *Development tools*):

1. Install conda and make
2. Run `make virtual-environment`, if that fails you can try doing it manually
   1. Change your current directory to OpenSCM Two Layer Model’s root directory (i.e. the one which contains `README.rst`), `cd openscm-twolayermodel`
   2. Create a virtual environment to use with OpenSCM Two Layer Model `python3 -m venv venv`
   3. Activate your virtual environment `source ./venv/bin/activate`
   4. Upgrade pip `pip install --upgrade pip`
   5. Install the development dependencies (very important, make sure your virtual environment is active before doing this) `pip install -e .[dev]`
3. Make sure the tests pass by running `make checks`, if that fails the commands can be read out of the *Makefile*

### 3.2.1 Getting help

Whilst developing, unexpected things can go wrong (that’s why it’s called ‘developing’, if we knew what we were doing, it would already be ‘developed’). Normally, the fastest way to solve an issue is to contact us via the issue tracker. The other option is to debug yourself. For this purpose, we provide a list of the tools we use during our development as starting points for your search to find what has gone wrong.

#### Development tools

This list of development tools is what we rely on to develop OpenSCM Two Layer Model reliably and reproducibly. It gives you a few starting points in case things do go inexplicably wrong and you want to work out why. We include links with each of these tools to starting points that we think are useful, in case you want to learn more.

- Git
- Make
- Conda virtual environments
• Pip and pip virtual environments
• Tests
  – we use a blend of pytest and the inbuilt Python testing capabilities for our tests so checkout what we’ve already done in tests to get a feel for how it works
• Continuous integration (CI) (also brief intro blog post and a longer read)
  – we use GitHub CI for our CI but there are a number of good providers
• Jupyter Notebooks
  – Jupyter is automatically included in your virtual environment if you follow our Getting setup instructions
• Sphinx

Other tools

We also use some other tools which aren’t necessarily the most familiar. Here we provide a list of these along with useful resources.

• Regular expressions
  – we use regex101.com to help us write and check our regular expressions, make sure the language is set to Python to make your life easy!

3.3 Formatting

To help us focus on what the code does, not how it looks, we use a couple of automatic formatting tools. These automatically format the code for us and tell use where the errors are. To use them, after setting yourself up (see Getting setup), simply run make format. Note that make format can only be run if you have committed all your work i.e. your working directory is ‘clean’. This restriction is made to ensure that you don’t format code without being able to undo it, just in case something goes wrong.

3.4 Building the docs

After setting yourself up (see Getting setup), building the docs is as simple as running make docs (note, run make -B docs to force the docs to rebuild and ignore make when it says ‘... index.html is up to date’). This will build the docs for you. You can preview them by opening docs/build/html/index.html in a browser.

For documentation we use Sphinx. To get ourselves started with Sphinx, we started with this example then used Sphinx’s getting started guide.

3.4.1 Gotchas

To get Sphinx to generate pdfs (rarely worth the hassle), you require Latexmk. On a Mac this can be installed with sudo tlmgr install latexmk. You will most likely also need to install some other packages (if you don’t have the full distribution). You can check which package contains any missing files with tlmgr search --global --file [filename]. You can then install the packages with sudo tlmgr install [package].
3.4.2 Docstring style

For our docstrings we use numpy style docstrings. For more information on these, here is the full guide and the quick reference we also use.

3.5 Releasing

3.5.1 First step

1. Test installation with dependencies make test-install
2. Update CHANGELOG.rst
   - add a header for the new version between master and the latest bullet point
   - this should leave the section underneath the master header empty
3. git add .
4. git commit -m "Prepare for release of vX.Y.Z"
5. git tag vX.Y.Z
6. Test version updated as intended with make test-install

3.5.2 Push to repository

To do the release, push the tags and the repository state.

1. git push
2. git push --tags

Assuming all the checks pass, this automatically triggers a release on PyPI via the .github/workflows/ci-cd-workflow.yml action.

3.6 Why is there a Makefile in a pure Python repository?

Whilst it may not be standard practice, a Makefile is a simple way to automate general setup (environment setup in particular). Hence we have one here which basically acts as a notes file for how to do all those little jobs which we often forget e.g. setting up environments, running tests (and making sure we’re in the right environment), building docs, setting up auxillary bits and pieces.
Module containing the base for model implementations

```python
class openscm_twolayermodel.base.Model
    Bases: abc.ABC

    Base class for model implementations

    reset()
        Reset everything so that a new run can be performed.
        Called as late as possible before run().

    run()
        Run the model.

    abstract set_drivers(*args, **kwargs)
        Set the model’s drivers

    step()
        Do a single time step.
```

class openscm_twolayermodel.base.TwoLayerVariant
    Bases: openscm_twolayermodel.base.Model

    Base for variations of implementations of the two-layer model

    property delta_t
        pint.Quantity Time step for forward-differencing approximation

    property erf
        pint.Quantity Effective radiative forcing

    reset()
        Reset everything so that a new run can be performed.
        Called as late as possible before run().

    run()
        Run the model.

    run_scenarios(scenarios, driver_var='Effective Radiative Forcing', progress=True)
        Run scenarios.

        The model timestep is automatically adjusted based on the timestep used in scenarios. The timestep used in scenarios must be constant because this implementation has a constant timestep. Pull requests to upgrade the implementation to support variable timesteps are welcome https://github.com/openscm/openscm-twolayermodel/pulls.

        Parameters
- **scenarios** (ScmDataFrame or ScmRun or `pyam.IamDataFrame` or `pd.DataFrame` or `np.ndarray` or `str`) – Scenarios to run. The input will be converted to an ScmRun before the run takes place.

- **driver_var** (`str`) – The variable in scenarios to use as the driver of the model

- **progress** (`bool`) – Whether to display a progress bar

**Returns**  Results of the run (including drivers)

**Return type**  ScmRun

**Raises**  ValueError – No data is available for driver_var in the "World" region in scenarios.

**set_drivers**(
    erf
)

Set drivers for a model run

**Parameters**  erf (pint.Quantity) – Effective radiative forcing (W/m^2) to use to drive the model

**Raises**  AssertionError – erf is not one-dimensional

**step**()

Do a single time step.
Module containing the impulse response model

The 2-timescale impulse response model is mathematically equivalent to the two-layer model without state dependence.

```python
class openscm_twolayermodel.impulse_response_model.ImpulseResponseModel(q1=<Quantity(0.3, 'delta_degree_Celsius * meter ** 2 / watt')>, q2=<Quantity(0.4, 'delta_degree_Celsius * meter ** 2 / watt')>, d1=<Quantity(9.0, 'a'>, d2=<Quantity(400.0, 'a'>, efficacy=<Quantity(1.0, 'dimensionless'>, delta_t=<Quantity(0.0833333333, 'a'>)
```

**Bases:** openscm_twolayermodel.base.TwoLayerVariant

**TODO:** top line and paper references

This implementation uses a forward-differencing approach. This means that temperature and ocean heat uptake values are start of timestep values. For example, temperature[i] is only affected by drivers from the i-1 timestep. In practice, this means that the first temperature and ocean heat uptake values will always be zero and the last value in the input drivers has no effect on model output.

- **property d1**
  - pint.Quantity Response timescale of first box

- **property d2**
  - pint.Quantity Response timescale of second box

- **property delta_t**
  - pint.Quantity Time step for forward-differencing approximation

- **property efficacy**
  - pint.Quantity Efficacy factor

- **property erf**
  - pint.Quantity Effective radiative forcing

- **get_two_layer_parameters()**
  - Get equivalent two-layer model parameters
For details on how the equivalence is calculated, please see the notebook `impulse-response-equivalence.ipynb` in the OpenSCM Two Layer model repository.

**Returns** dict of str – Input arguments to initialise an `openscm_twolayermodel`. TwoLayerModel with the same temperature response as `self`

**Return type** pint.Quantity

**property q1**
pint.Quantity Sensitivity of first box response to radiative forcing

**property q2**
pint.Quantity Sensitivity of second box response to radiative forcing

**reset()**
Reset everything so that a new run can be performed.

Called as late as possible before `run()`.

**run()**
Run the model.

**run_scenarios(scenarios, driver_var='Effective Radiative Forcing', progress=True)**
Run scenarios.

The model timestep is automatically adjusted based on the timestep used in `scenarios`. The timestep used in `scenarios` must be constant because this implementation has a constant timestep. Pull requests to upgrade the implementation to support variable timesteps are welcome https://github.com/openscm/openscm-twolayermodel/pulls.

**Parameters**

- **scenarios** (ScmDataFrame or ScmRun or pyam.IamDataFrame or pd.DataFrame or np.ndarray or str) – Scenarios to run. The input will be converted to an ScmRun before the run takes place.

- **driver_var** (str) – The variable in `scenarios` to use as the driver of the model

- **progress** (bool) – Whether to display a progress bar

**Returns** Results of the run (including drivers)

**Return type** ScmRun

**Raises** ValueError – No data is available for `driver_var` in the "World" region in `scenarios`.

**set_drivers(erf)**
Set drivers for a model run

**Parameters** erf (pint.Quantity) – Effective radiative forcing (W/m^2) to use to drive the model

**Raises** AssertionError – `erf` is not one-dimensional

**step()**
Do a single time step.
TWO LAYER MODEL API

Module containing the two-layer model

class openscm_twolayermodel.two_layer_model.TwoLayerModel(
    du=Quantity(50, 'meter'),
    dl=Quantity(1200, 'meter'),
    lambda0=Quantity(1.24666667, 'watt / delta_degree_Celsius / meter ** 2'),
    a=Quantity(0.0, 'watt / delta_degree_Celsius ** 2 / meter ** 2'),
    efficacy=Quantity(1.0, 'dimensionless'),
    eta=Quantity(0.8, 'watt / delta_degree_Celsius / meter ** 2'),
    delta_t=Quantity(31557600.0, 'second'))

Bases: openscm_twolayermodel.base.TwoLayerVariant

TODO: top line and paper references

This implementation uses a forward-differencing approach. This means that temperature and ocean heat uptake values are start of timestep values. For example, temperature[i] is only affected by drivers from the i-1 timestep. In practice, this means that the first temperature and ocean heat uptake values will always be zero and the last value in the input drivers has no effect on model output.

property a
    pint.Quantity Dependence of climate feedback factor on temperature

property delta_t
    pint.Quantity Time step for forward-differencing approximation

property dl
    pint.Quantity Depth of lower layer

property du
    pint.Quantity Depth of upper layer

property efficacy
    pint.Quantity Efficacy factor

property erf
    pint.Quantity Effective radiative forcing

property eta
    pint.Quantity Heat transport efficiency

get_impulse_response_parameters()
    Get equivalent two-timescale impulse response model parameters
For details on how the equivalence is calculated, please see the notebook `impulse-response-equivalence.ipynb` in the OpenSCM Two Layer model repository.

**Returns**

- dict of str
  - Input arguments to initialise an `openscm_twolayermodel`. ImpulseResponseModel with the same temperature response as `self`

**Return type**

- pint.Quantity

**Raises**

- ValueError
  - `self.a` is non-zero, the two-timescale model does not support state-dependence.

**property heat_capacity_lower**

- pint.Quantity
  - Heat capacity of lower layer

**property heat_capacity_upper**

- pint.Quantity
  - Heat capacity of upper layer

**property lambda0**

- pint.Quantity
  - Initial climate feedback factor

**reset()**

- Reset everything so that a new run can be performed.

- Called as late as possible before `run()`.

**run()**

- Run the model.

**run_scenarios(scenarios, driver_var='Effective Radiative Forcing', progress=True)**

- Run scenarios.

  - The model timestep is automatically adjusted based on the timestep used in `scenarios`. The timestep used in `scenarios` must be constant because this implementation has a constant timestep. Pull requests to upgrade the implementation to support variable timesteps are welcome https://github.com/openscm/openscm-twolayermodel/pulls.

  **Parameters**

  - scenarios (ScmDataFrame or ScmRun or pyam.IamDataFrame or pd.DataFrame or np.ndarray or str) – Scenarios to run. The input will be converted to an ScmRun before the run takes place.

  - driver_var (str) – The variable in `scenarios` to use as the driver of the model

  - progress (bool) – Whether to display a progress bar

  **Returns**

  - Results of the run (including drivers)

  **Return type**

  - ScmRun

  **Raises**

  - ValueError
    - No data is available for `driver_var` in the "World" region in `scenarios`.

**set_drivers(erf)**

- Set drivers for a model run

  **Parameters**

  - erf (pint.Quantity) – Effective radiative forcing (W/m^2) to use to drive the model

  **Raises**

  - AssertionError
    - erf is not one-dimensional

**step()**

- Do a single time step.
Physical constants used in calculations

```python
openscm_twolayermodel.constants.DENSITY_WATER = <Quantity(1000.0, 'kilogram / meter ** 3')>

density of water

    Type pint.Quantity

openscm_twolayermodel.constants.HEAT_CAPACITY_WATER = <Quantity(4181.0, 'joule / delta_degree_Celsius / kilogram')>

heat capacity of water

    Type pint.Quantity
```
CHAPTER
EIGHT

ERRORS API

Exceptions raised within openscm_twolayermodel

```python
exception openscm_twolayermodel.errors.ModelStateError
    Bases: ValueError
    Exception raised if a model’s state is incompatible with the action
    args
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
```

```python
exception openscm_twolayermodel.errors.UnitError
    Bases: ValueError
    Exception raised if something has the wrong units
    args
    with_traceback()
        Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.
```
Utility functions

```python
gls: utilities
convert_lambda_to_ecs

Convert a lambda value to equilibrium climate sensitivity (ECS)

**Parameters**

- `lambda_val` (*pint.Quantity*) – Value of lambda to convert to ECS
- `f2x` (*pint.Quantity*) – Value of the forcing due to a doubling of atmospheric CO\textsubscript{2} to assume during the conversion

**Returns**

ECS value

**Return type**

`pint.Quantity`

**Raises**

`TypeError` – `lambda_val` or `f2x` is not a `pint.Quantity`.
```
All notable changes to this project will be documented in this file. The format is based on Keep a Changelog, and this project adheres to Semantic Versioning. The changes listed in this file are categorised as follows:

- Added: new features
- Changed: changes in existing functionality
- Deprecated: soon-to-be removed features
- Removed: now removed features
- Fixed: any bug fixes
- Security: in case of vulnerabilities.

### 10.1 v0.2.3 - 2021-04-27

**10.1.1 Fixed**

- (#34, #35, #36) Final tweaks to JOSS paper

### 10.2 v0.2.2 - 2021-04-27

**10.2.1 Added**

- (#33) Information in README and testing for conda install

**10.2.2 Changed**

- (#32) Include LICENSE, README.rst and CHANGELOG in package
- (#30) Require scmdata>=0.9
- (#27) Fixed the discussion (in the relevant notebook) of how a one-layer model can be made from the two-layer implementation here
10.2.3 Fixed

• (#30) Incorrect call to \texttt{scmdata.ScmRun()} in tests

10.3 v0.2.1 - 2020-12-23

10.3.1 Added

• (#20) Statement of need to the README following JOSS review (closes #18)

10.3.2 Changed

• (#26) Updated to new scmdata version (and hence new openscm-units API)
• (#25) JOSS paper following JOSS review 1
• (#23) Moved notebooks into full documentation following JOSS review (closes #17)
• (#21) Quoted pip install instructions to ensure cross-shell compatibility following JOSS review (closes #16)
• (#20) Option to remove tqdm progress bar by passing \texttt{progress=False}

10.4 v0.2.0 - 2020-10-09

10.4.1 Added

• (#7) JOSS paper draft

10.4.2 Changed

• (#7) Require \texttt{scmdata>=0.7}

10.5 v0.1.2 - 2020-31-07

10.5.1 Changed

• (#12) Upgrade to \texttt{scmdata>=0.6.2} so that package can be installed
10.6 v0.1.1 - 2020-06-29

10.6.1 Added

• (#8) Add notebook showing how to run a single-layer model

10.6.2 Changed

• (#11) Re-wrote the getting started notebook
• (#10) Re-wrote CHANGELOG
• (#9) Update to scmdata 0.5.Y

10.7 v0.1.0 - 2020-05-15

10.7.1 Added

• (#3) Add first implementation of the models
• (#1) Setup repository
INDEX

- genindex
- modindex
- search
PYTHON MODULE INDEX

O
openscm_twolayermodel.base, 75
openscm_twolayermodel.constants, 81
openscm_twolayermodel.errors, 83
openscm_twolayermodel.impulse_response_model, 77
openscm_twolayermodel.two_layer_model, 79
openscm_twolayermodel.utils, 85
INDEX

A
a (openscm_twolayermodel.two_layer_model.TwoLayerModel property), 79
args (openscm_twolayermodel.errors.ModelStateError attribute), 83
args (openscm_twolayermodel.errors.UnitError attribute), 83

C
convert_lambda_to_ecs() (in module openscm_twolayermodel.utils), 85

D
d1 (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77
d2 (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77
delta_t (openscm_twolayermodel.base.TwoLayerVariant property), 75
delta_t (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77
delta_t (openscm_twolayermodel.two_layer_model.TwoLayerModel property), 79
DENSITY_WATER (in module openscm_twolayermodel.constants), 81
dl (openscm_twolayermodel.two_layer_model.TwoLayerModel property), 79
du (openscm_twolayermodel.two_layer_model.TwoLayerModel property), 79

e (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77
efficacy (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77
efficacy (openscm_twolayermodel.two_layer_model.TwoLayerModel property), 79
erf (openscm_twolayermodel.base.TwoLayerVariant property), 75
erf (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77
erf (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77

G
get_impulse_response_parameters() (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel method), 77
get_two_layer_parameters() (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel method), 77

H
heat_capacity_lower (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 80
heat_capacity_upper (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 80
HEAT_CAPACITY_WATER (in module openscm_twolayermodel.constants), 81

I
ImpulseResponseModel (class in openscm_twolayermodel.impulse_response_model), 77

L
lambda0 (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 77
lambda0 (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 80

M
Model (class in openscm_twolayermodel.impulse_response_model), 75
ModelStateError, 83
ModelStateError, 83

eta (openscm_twolayermodel.impulse_response_model.ImpulseResponseModel property), 79