ERSEM

Release 2021

The PML Modelling Group

Jun 30, 2023

CONTENTS:

ERSEM is a marine biogeochemical and ecosystem model. It describes the cycling of carbon, nitrogen, phosphorus, silicon, oxygen and iron through the lower trophic level pelagic and benthic ecosystems.

- **Installation Instructions:** conda: https://ersem.readthedocs.io/en/latest/tutorials/index.html#conda-installation, source: https://ersem.readthedocs.io/en/latest/developers/index.html
- **User Documentation & Example Usage:** https://ersem.readthedocs.io/en/latest/
- **Automated Tests:** Run via GitHub Actions https://github.com/pmlmodelling/ersem/actions
- **Acknowledgements:** https://ersem.readthedocs.io/en/latest/acknowledgements.html
- **License:** https://ersem.readthedocs.io/en/latest/license.html

CHAPTER

ONE

SUPPORT

We strongly encourage everyone using the ERSEM code to register as a user by filling a short registration form. We'd love to get a better understanding of who is using the ERSEM code and for what applications, as it will help us to provide the best support to the user community. It will also allow you to receive information and news on the latest model developments.

CHAPTER

TWO

HOW TO CITE

To refer to ERSEM in publications, please cite:

Butenschön, M., Clark, J., Aldridge, J.N., Allen, J.I., Artioli, Y., Blackford, J., Bruggeman, J., Cazenave, P., Ciavatta, S., Kay, S., Lessin, G., van Leeuwen, S., van der Molen, J., de Mora, L., Polimene, L., Sailley, S., Stephens, N., Torres, R. (2016). ERSEM 15.06: a generic model for marine biogeochemistry and the ecosystem dynamics of the lower trophic levels. Geoscientific Model Development, 9(4), 1293–1339. doi: 10.5194/gmd-9-1293-2016.

To refer specifically to the ERSEM source code, you may use its Zenodo DOI:

2.1 The ERSEM Model

Here we outline the features of the ERSEM model.

2.1.1 What is ERSEM?

Background

> PML is a charity, undertaking innovative, cutting-edge marine research, allowing society to benefit from clean, productive, biologically diverse seas, now and for future generations.

Ecosystem

The ecosystem in ERSEM is divided into functional types, which are further subdivided by traits such as size. In the pelagic, ERSEM by default distinguishes:

- 4 types of phytoplankton: diatoms, picophytoplankton, nanophytoplankton, microphytoplankton
- 3 types of zooplankton: nanoflagellates, microzooplankton, mesozooplankon
- bacteria

The benthic system includes:

- 3 types of infauna: meiofauna, suspension feeders, deposit feeders
- 2 types of bacteria: aerobic and anaerobic

In addition, ERSEM tracks the concentrations of phosphate, nitrate, ammonium, silicate, iron, oxygen, dissolved inorganic carbon and alkalinity in the pelagic and in sediment porewaters. It includes several classes of particulate and dissolved organic matter in pelagic and sediment. A carbonate system module calculates pH and calcium carbonate saturation.

FABM also makes it easy to customise the default set of functional types described above, and to combine ERSEM with modules representing other parts of the ecosystem, including fish communities, shellfish, seagrass meadows and spectraly resolved irradiance.

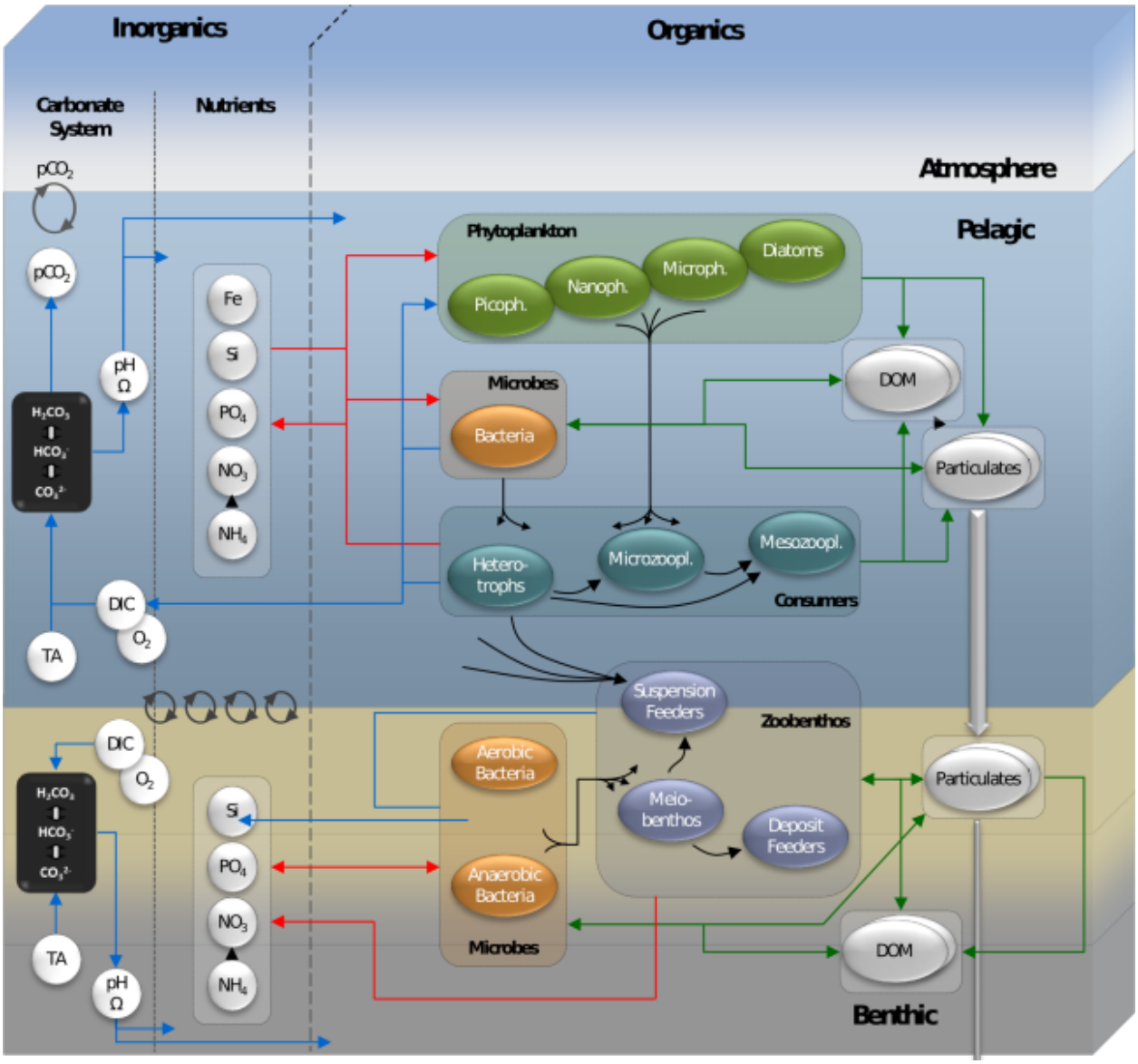


Fig. 1: ERSEM schematic

2.1.2 The History of ERSEM

The European Regional Seas Ecosystem Model, ERSEM, was initially developed in two projects, funded by the EU Marine Science and Technology programme (ERSEM, 1990-1993; ERSEM II, 1993-1996) both under the leadership of Job Baretta at NIOZ (NL). In addition to NIOZ, these initial projects involved Plymouth Marine Laboratory (UK), the Universities of Oldenburg (DE), Hamburg (DE), Strathclyde (UK) and Aberdeen (UK), Marine Laboratory Aberdeen, Ecological Modelling Centre (DK) and CEAB (ES).

The earlier ambitions of the program were significant, not only coupling benthic and pelagic processes in a physical context and developing a prognostic model, but extending up the food chain to age-structured mesozooplankton models, pelagic and demersal fish and even seabirds. As a general rule, the higher the trophic level the less successful the models as 20+ years ago understanding, data and computational power were far inferior to that which we currently enjoy.

Although ERSEM itself had roots in earlier modelling initiatives (the Ems-Dollard and Gembase models) it pioneered a community approach to model development. At the heart of this was a set of scripts (SESAME) which controlled compilation and output generation, data transfer by COMMON BLOCK, a system of code nomenclature ([1], if you are wondering why variables and parameters have strange names) and a structure of almost independent modules. This meant that for the majority of developers, with minimal practice, the code was accessible with limited computational expertise. Development of modules could occur in parallel without worrying about interfaces between functional groups. This accessibility did not lead to the most efficient coding but did allow a range of computationally naive experts the unfettered ability to develop the model. A six monthly development cycle, whereby new modules were submitted for assessment and incorporation ahead of a project meeting to evaluate results ensured rapid model development. Computational An early schematic of ERSEM incompatibility was avoided by providing each partner with the same UNIX computer system, the then eye-wateringly expensive SUN SPARC desktops retailing at £20K apiece. Just a few years previously, before the internet, email and processing capability exceeding 100 MHz ERSEM would have been inconceivable.

Each of the initial ERSEM projects produced a special issue describing model systems and results, the Netherlands Journal of Sea Research V33 (3/4) in 1995 and the by then renamed Journal of Sea Research V38 (3/4) in 1997. After cessation of the ERSEM projects model development occurred more independently in three main centres, NIOZ and PML producing variants using the ERSEM name and at Bologna where ERSEM formed the basis of the Biogeochemical Flux Model (BFM). We suspect that in excess of 200 papers have been published using ERSEM since 1995.

The ERSEM model now aspires to reach beyond its name, addressing biogeochemical and ecological systems in many applications in global regional seas and more recently the global ocean, engaging in a range of heuristic, predictive and impact studies.

2.1.3 ERSEM model domains

ERSEM is used in a variety of geographical regions, coupled to different physical (hydrodynamic) models. The table bellow outlines several of these setups.

Area		Spatial Scale		Quality (data used)
		Domain	Res.(km)	
Water column (GOTM-ERSEM)		Various locations	N/A	4,6,8,14,16,22
Global (NEMO)		Global	111	8,11,19,20,21
North Atlantic (NEMO)		20N-8ON	28	8,11,19,20,21
North Atlantic (NEMO shelf)		20N-80N	7	8,11,19,20,21
Irish and Celtic Seas (FVCOM)		41.8-56.8N, 9.6-2.5W	3.5	7,23,24,25
NW European shelf (NEMO)		40N-65N, 2OW-13E	7	3,8,11,12,13,14,19
NW European shelf (POLCOMS)		40N-65N, 20W-13E	10	3,8,9,10,11,12, 13,14,15,16,17,19
NW European shelf (POLCOMS + data assimilation)		40N-65N, 20w-13E	10	2,3,8,9,10,12,13, 14,15,17
AMM7NW European-Baltic-GCOMS		46.4-63N, 17.5W-13E	10	3,8,11,14,17,18
Western European Channel (WEC)		48.5-50.9N, 7.6-1.25W	1.9	2,4,8,12,13,14
WEC (data assimilation)		48.5-50.9N, 7.6-1.25W	7	1,2,4,8,12,13,14

Key

1. GlobColour ocean colour
2. ESA CCI ocean colour
3. North Sea Project cruises
4. Western Channel Observatory
5. Tide gauges
6. Smart Buoy data
7. NCEP reanalysis
8. World Ocean Atlas
9. ICES temperature and salinity
10. ICES nutrients and chlorophyll
11. IPCC climate forcing (HADGEM, IPSL, ECHAM)
12. EA riverine nutrients data
13. European river data
14. ECMWF reanalysis meteorology
15. DMI reanalysis meteorology
16. ESSC ocean reanalysis
17. GLORYS reanalysis
18. GLOBAL NEWS river data
19. GLODAP
20. DFS atmospheric reanalysis
21. GEMS-GLORI river data
22. BATS data
23. HF Radar and CTD Data
24. NTSLF data
25. CEH river discharge and temperature

Note: The table of regional setups is not yet complete, please stay tuned for updates.

Existing uses:

- **Natural resources:** understanding biogeochemical cycles, biodiversity and valuation of ecosystem services.
- **Resilience to environmental hazards:** eutrophication, microplastics, fishing pressure, invasive species and harmful algal blooms. Impacts of offshore renewable energy and ecological risks of carbon capture and storage.
- **Environmental change:** climate change impacts, ocean acidification, multiple stressors, and sustainable fisheries.
- Run operationally by the UK Met Office to predict water quality.
- Estimation of the carbon and nutrient budgets of the UK shelf.

Potential new uses:

- Understanding shelf seas carbon ("blue carbon") and nutrient budgets (past and future climate).
- Expansion to represent biodiversity-relevant processes over a range of spatial and temporal scales, and simulate changes in function in the context of ecosystem services.
- Implementation and testing scalable models of differing complexity.

Key modelling issues:

- Setting inputs (parameterisation} and testing outputs against real data (calibration) is an essential, but resource-intensive and on-going process to ensure quality and improve predictions. Understanding the impact of changing inputs on the outputs from the models (sensitivity) and the effect of uncertainty in model parameters on robustness of model predictions.
- Challenge to assess model capability with respect to seasonal variability, long-term changes, regime shift and tipping points due to limitations of the data available.
- Complexity of model leads to a need for significant interpretation and explanation for stakeholders.
- Potential mismatch between scales of model output and data sets.
- Significant expertise needed to operate system and high performance parallel computing facility required for three-dimensional full scale simulations.

2.2 ERSEM tutorials

ERSEM is mainly run with 3D hydrodynamic models such as NEMO and FVCOM. However, due to its flexiblilty, it can also be run as a 0D box model or within a 1D water column, via GOTM. Parts of the library can also be called directly from Python through the `pyfabm` interface. Here we describe several tutorials for the common use cases for ERSEM.

2.2.1 Python front-end (`pyfabm`)

`FABM` comes with a `pyfabm` package for the `python` programming language. This package enables you to access FABM's biogeochemical models directly from `python`. For instance, you can enumerate a model's parameters and variables, or obtain temporal derivatives and diagnostics for any given environment and model state. In combination with a `python`-based time integration scheme (e.g., `scipy.integrate.odeint`), this also allows you to perform model simulations. More information about `pyfabm` can be found on the FABM wiki page. Below you can find brief instructions on how to use `pyfabm` with ERSEM.

Running `pyfabm` tutorial

To demonstrate how `pyfabm` can be used with ERSEM, we will calculate the oxygen saturation concentration as a function of temperature and salinity using the method implemented in ERSEM, which is taken from [3].

Note: The following script requires `matplotlib` to be installed. This can easily be done via `pip` in the following way:

```
python -m pip install matplotlib
```

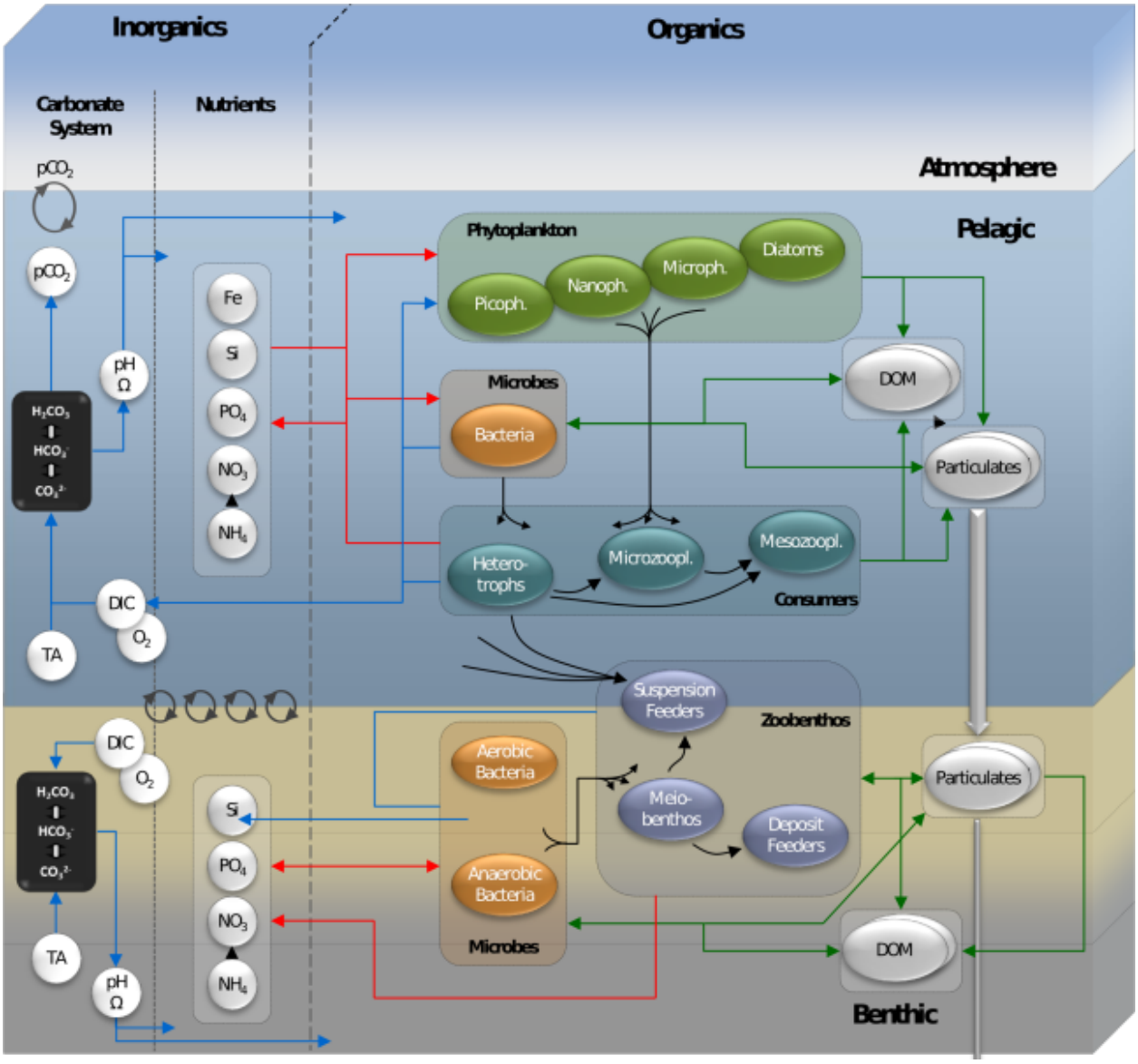


Fig. 2: ERSEM schematic

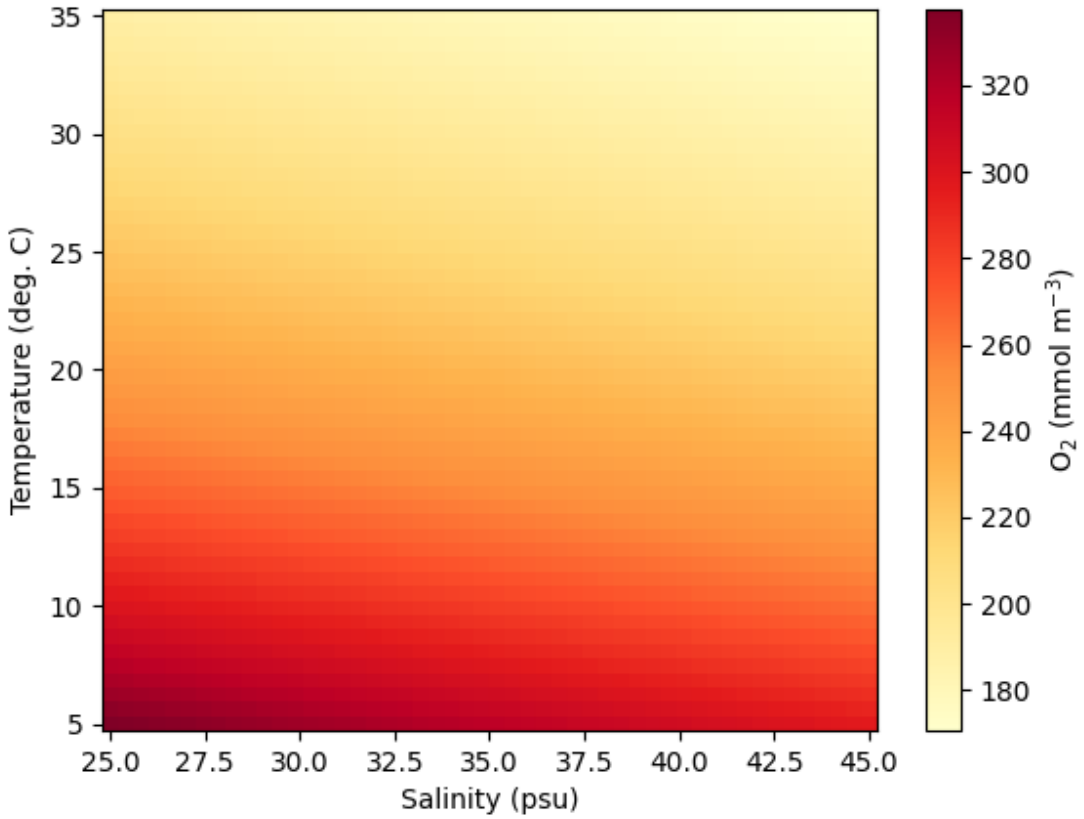
With pyfabm installed, this can be achieved using the following code:

```
from matplotlib import cm
import matplotlib.pylab as plt
import numpy as np
import os
import pyfabm

def main():
    """
    Run pyfabm-ERSEM tutorial

    :param model_path: Full path to netCDF model output
    :type model_path: str

    :return: list of oxygen saturation concentrations
    :rtype: list
    """
    dir_path = os.path.dirname(os.path.realpath(__file__))
    ersem_dir = os.path.dirname(os.path.dirname(dir_path))

    # Path to ERSEM yaml file
    ersem_yaml_file = os.path.join(ersem_dir,
                                   'testcases',
                                   'fabm-ersem-15.06-L4-noben-docdyn-iop.yaml')

    if not os.path.isfile(ersem_yaml_file):
        raise RuntimeError("Could not find Ersem yaml file with the "
                           "{}".format(ersem_yaml_file))

    # Create model
    model = pyfabm.Model(ersem_yaml_file)

    # Configure the environment
    model.findDependency('longitude').value = -4.15
    model.findDependency('latitude').value = 50.25
    model.findDependency('number_of_days_since_start_of_the_year').value = 0.
    model.findDependency('temperature').value = 10.
    model.findDependency('wind_speed').value = 1.
    model.findDependency('surface_downwelling_shortwave_flux').value = 50.
    model.findDependency('practical_salinity').value = 35.
    model.findDependency('pressure').value = 10.
    model.findDependency('density').value = 1035.
    model.findDependency('mole_fraction_of_carbon_dioxide_in_air').value = 280.
    model.findDependency('absorption_of_silt').value = 0.07
    model.findDependency('bottom_stress').value = 0.
    model.findDependency('cell_thickness').value = 1.
    model.setCellThickness(1)

    # Verify the model is ready to be used
    assert model.checkReady(), 'One or more model dependencies have not been fulfilled.'

```

(continues on next page)

(continued from previous page)

```
    # Define ranges over which temperature and salinity will be varied
    n_points = 50
    temperature_array = np.linspace(5, 35, n_points)
    salinity_array = np.linspace(25, 45, n_points)

    # Create an array in which to store oxygen_saturation_concentrations
    oxygen_saturation_concentration = np.empty((n_points, n_points), dtype=float)

    # Calculate oxygen saturation concentrations using ERSEM
    for t_idx, t in enumerate(temperature_array):
        model.findDependency('temperature').value = t
        for s_idx, s in enumerate(salinity_array):
            model.findDependency('practical_salinity').value = s
            _ = model.getRates()
            oxygen_saturation_concentration[t_idx, s_idx] = \
                    model.findDiagnosticVariable('O2/osat').value

    # Create the figure
    figure = plt.figure()
    axes = plt.gca()

    # Set color map
    cmap = cm.get_cmap('YlOrRd')

    # Plot
    plot = axes.pcolormesh(salinity_array,
                           temperature_array,
                           oxygen_saturation_concentration,
                           shading='auto',
                           cmap=cmap)

    axes.set_xlabel('Salinity (psu)')
    axes.set_ylabel('Temperature (deg. C)')



    # Add colour bar
    cbar = figure.colorbar(plot)
    cbar.set_label('O$_{2}$ (mmol m$^{-3}$)')

    plt.show()

    return oxygen_saturation_concentration

if __name__ == "__main__":
    main()
```



In additions, example Jupyter notebooks that use the Python front-end can be found in `<FABMDIR>/testcases/python` and more examples can be found on the FABM wiki.

2.2.2 fabm0d: ERSEM in an aquarium

FABM's 0d driver allows you to run biogeochemical models in a "well-mixed box" under arbitrary (time-varying) environmental forcing.

Running FABM0d tutorial

To run the model, you must first obtain or generate a set of input or forcing files. Setups for the L4 site are stored under version control and can be accessed here.

To run the tutorial, you will first need to clone the configuration files and then run *FABM0d*. This is done with the following:

```
#!/bin/bash

echo "Cloning config repo"
git clone https://github.com/pmlmodelling/ersem-setups.git

cd ersem-setups/0d-aquarium
```

(continues on next page)

(continued from previous page)

```

echo "Running FABM with repo configuration"
~/local/fabm/0d/bin/fabm0d -y ../L4/fabm.yaml
```

Note: The following script requires `matplotlib`, `netCDF` and `numpy` to be installed. This can easily be done via `pip` in the following way:

```
python -m pip install matplotlib numpy netCDF
```

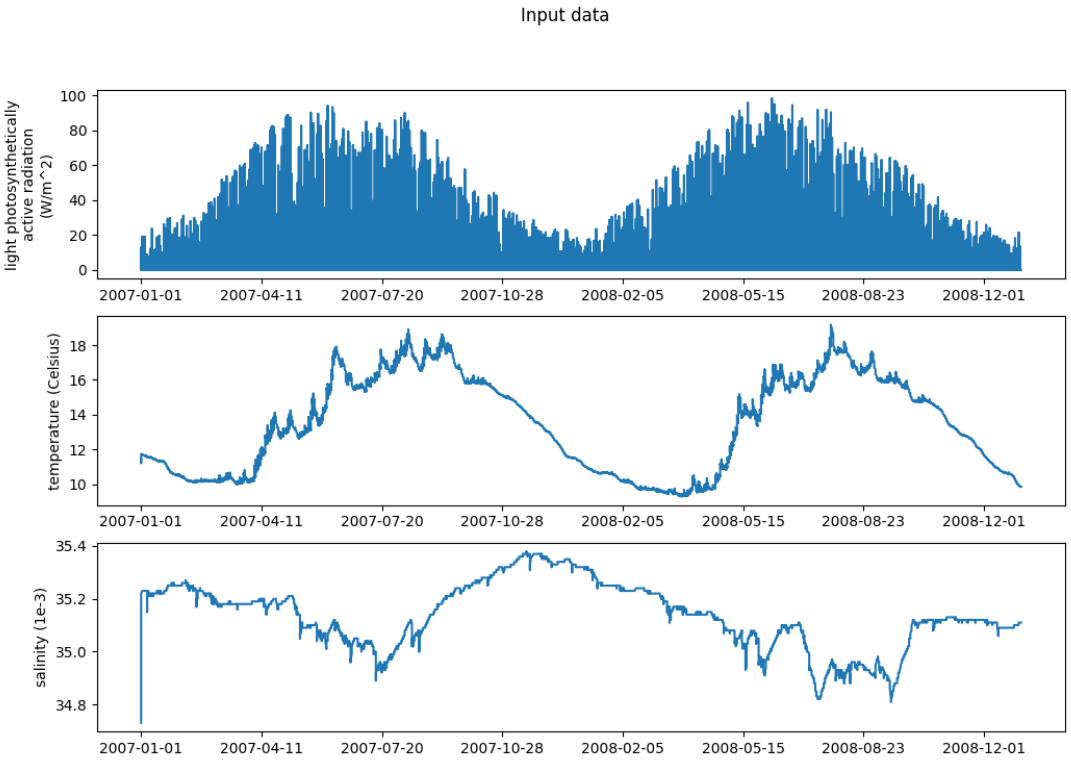
To visualise the results, we again use a python script. Running the script you will need to add a commandline argument `--model-path` which is the path to the output from the FABM0d run.

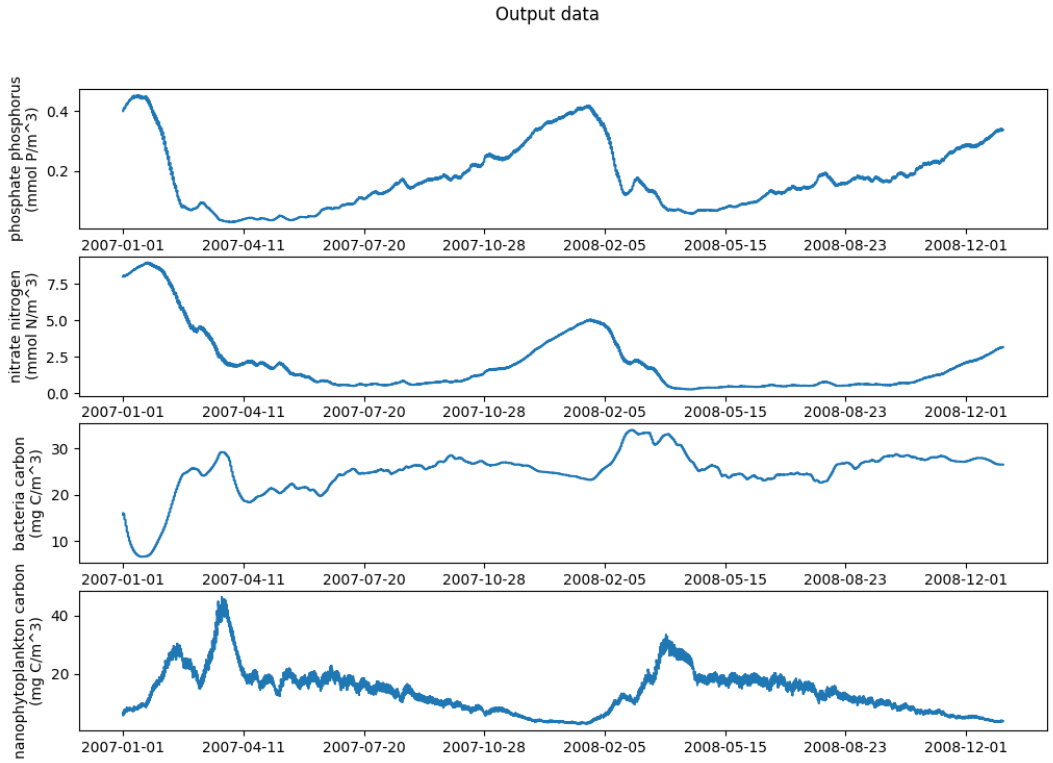
```
import argparse
import matplotlib.pylab as plt
from matplotlib import ticker
import numpy as np
import datetime
import re
import sys


try:
    import netCDF4 as nc
except ImportError:
    print("Please install the Python interface to netCDF")
    sys.exit()


def main(model_path):
    """
    Run FABM0D-ERSEM tutorial

    :param model_path: Full path to netCDF model output
    :type model_path: str

    :return: Dictionary containing output model variables for systests
    :rtype: dict
    """
    # Dictionary used to save input and output variables for testing
    test_vars = {}
    def plot_var(axes, x, y, label):
        # sets the maximum number of labels on the x axis to be 10
        xticks = ticker.MaxNLocator(10)

        var = data.variables[label]
        y = var[:].squeeze()

        long_label = '{} ({})'.format(var.long_name, var.units)
        y_label = re.sub(r'(\s\S*?)\s', r'\1\n',long_label)

```

(continues on next page)

(continued from previous page)

```
        axes.plot(x, y)
        axes.set_ylabel(y_label)
        axes.xaxis.set_major_locator(xticks)
        test_vars[label] = y


    data = nc.Dataset(model_path, 'r')

    times = data.variables['time']
    dates = nc.num2date(times[:],
                        units=times.units,
                        calendar=times.calendar)
    dates = [str(d).split(" ")[0] for d in dates]
    test_vars["dates"] = dates

    # Plotting input data
    input_data = ["light_parEIR", "temp", "salt"]
    fig, ax_arr = plt.subplots(len(input_data), 1)
    DPI = fig.get_dpi()
    fig.set_size_inches(1200.0/float(DPI),800.0/float(DPI))
    for ax, label in zip(ax_arr, input_data):
        plot_var(ax, dates, data, label)
    fig.suptitle("Input data")

    # Plotting output data
    output_data = ["N1_p", "N3_n", "B1_c", "P2_c"]
    fig, ax_arr = plt.subplots(len(input_data), 1)
    DPI = fig.get_dpi()
    fig.set_size_inches(1200.0/float(DPI),800.0/float(DPI))
    for ax, label in zip(ax_arr, output_data):
        plot_var(ax, dates, data, label)
    fig.suptitle("Output data")
    plt.show()

    return test_vars

if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--model-path', type=str, required=True,
                        help='Path to FABM0D-ERSEM output')
    args, _ = parser.parse_known_args()
    model_path = args.model_path

    main(model_path)
```


Input data
light photosynthetically active radiation (W/m^2)
temperature (Celsius)
salinity (1e-3)
2007-01-01
2007-04-11
2007-07-20
2007-10-28
2008-02-05
2008-05-15
2008-08-23
2008-12-01


Output data
phosphate phosphorus (mmol P/m^3)
nitrate nitrogen (mmol N/m^3)
bacteria carbon (mg C/m^3)
nanophytoplankton carbon (mg C/m^3)
2007-01-01
2007-04-11
2007-07-20
2007-10-28
2008-02-05
2008-05-15
2008-08-23
2008-12-01

2.2.3 GOTM: ERSEM in a water column

To run the model, you must first obtain or generate a set of input or forcing files. Setups for the L4 and BATS sites which were used in [2] are stored under version control and can be accessed here. Forcing data for the SSB sampling sites can be accessed here. The following example uses the L4 setup described in [2].

Running GOTM-FABM-ERSEM

The L4 sampling site is one part of the Western Channel Observatory (WCO), and is located at (50°15.0'N; 4°13.0'W).

To run the tutorial, you will first need to clone the configuration files and then run *gotm*. This is done with the following:

```
#!/bin/bash

echo "Cloning config repo"
git clone https://github.com/pmlmodelling/ersem-setups.git

cd ersem-setups/L4

echo "Running GOTM with repo configuration"
~/local/gotm/bin/gotm
```

Note: The following script requires `matplotlib`, `netCDF4` and `numpy` to be installed. This can easily be done via `pip` in the following way:

```
python -m pip install matplotlib numpy netCDF4
```

To visualise the results, we again use a python script. Running the script you will need to add a commandline argument `--model-path` which is the path to the output from the GOTM-FABM-ERSEM run.

```
import argparse
import matplotlib.pyplot as plt
from matplotlib import ticker
import numpy as np
import os
import sys
import re

try:
    import netCDF4 as nc
except ImportError:
    print("Please install the Python interface to netCDF")
    sys.exit()


def main(model_path):
    """
    Run GOTM-ERSEM tutorial

    :param model_path: Full path to netCDF model output
    :type model_path: str

```

(continues on next page)

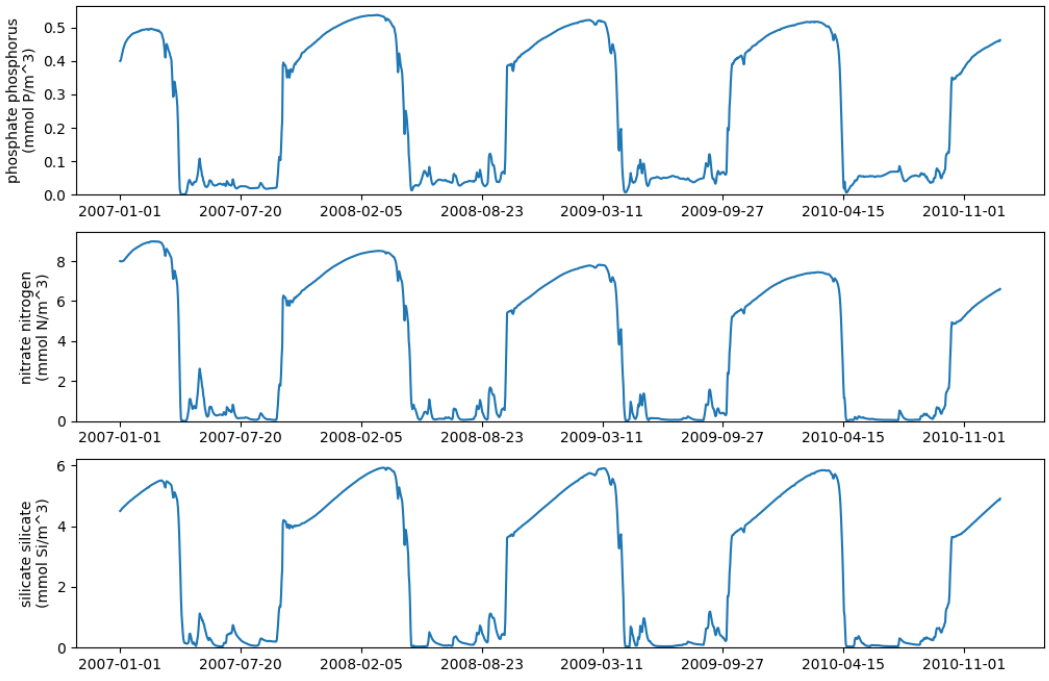
(continued from previous page)

```
    :return: Dictionary containing output model variables for systests
    :rtype: dict
    """

    # Dictionary used to save output variables for testing
    output_vars = {}
    def plot_time_series(axes, data, depth, model_var_name):
        # sets the maximum number of labels on the x axis to be 10
        xticks = ticker.MaxNLocator(10)

        times = data.variables['time']
        dates = nc.num2date(times[:],
                            units=times.units,
                            calendar=times.calendar)

        z = data.variables['z'][:].squeeze()
        zi = data.variables['zi'][:].squeeze()
        var = data.variables[model_var_name]

        long_label = '{} ({})'.format(var.long_name, var.units)
        y_label = re.sub(r'(\s\S*?)\s', r'\1\n',long_label)

        # Interpolate variable data to the given depth below the moving free surface
        var_time_series = []
        for i in range(var.shape[0]):
            # Remove offset introduced by the moving free surface
            depth_offset = depth + zi[i, -1]

            var_time_series.append(np.interp(depth_offset, z[i, :], var[i, :].squeeze()))

        dates = [str(d).split(" ")[0] for d in dates]
        axes.plot(dates, var_time_series)
        axes.set_ylabel(y_label)
        axes.xaxis.set_major_locator(xticks)
        output_vars[model_var_name] = var_time_series
        output_vars['dates'] = dates

    data = nc.Dataset(model_path, 'r')
    data.set_auto_maskandscale(True)

    # Depth at which to compare the model and data, given as relative to the moving free␣
↪surface.
    depth = 0.0

    fig, ax_arr = plt.subplots(3,1)
    DPI = fig.get_dpi()
    fig.set_size_inches(1200.0/float(DPI),800.0/float(DPI))

    names = ['N1_p', 'N3_n', 'N5_s']
    for ax, name in zip(ax_arr, names):
        plot_time_series(ax, data, depth, name)

```

(continues on next page)

(continued from previous page)

```
        ax.tick_params(axis='both', which='major')
        ax.tick_params(axis='both', which='minor')
        ax.set_ylim(0)

    plt.show()

    return output_vars

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-p', '--model-path', type=str, required=True,
                        help='Path to GOTM-FABM-ERSEM output')
    args, _ = parser.parse_known_args()
    model_path = args.model_path

    main(model_path)
```



2.2.4 FVCOM: ideal estuary model

This tutorial gives an end to end example how to install and then use FVCOM-ERSEM with an ideal estuary model on a high performance computing (HPC) machine. We have used PML's in house machine, CETO. Unlike the other tutorials in this section we go through setting up FVCOM-ERSEM on the HPC machine and then running and plotting the results.

This tutorial is based on the scripts found in the ERSEM's setups repository. The individual scripts can be found in the *ideal_estuary* folder.

Note: You will need to get access to the UK-FVCOM GitLab repo.

Building and running FVCOM-ERSEM

The three key packages required to run this tutorial are:

- FVCOM (UK version – see note)
- FABM
- ERSEM

Both ERSEM and FABM are freely available on GitHub, however, for UK-FVCOM you will have to register for the code – see note above.

An example build script is as follows:

```
#!/usr/bin/env bash

module load intel-mpi/5.1.2 netcdf-intelmpi/default intel/intel-2016 hdf5-intelmpi/default

SCRIPT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" &> /dev/null && pwd )"
CODE_DIR=$SCRIPT_DIR/code
INSTALL_DIR=$SCRIPT_DIR/model

num_cpu=$(nproc)

website=("git@github.com:UK-FVCOM-Usergroup/uk-fvcom.git" "git@github.com:pmlmodelling/ersem.git" "git@github.com:fabm-model/fabm.git")
name=("uk-fvcom" "ersem" "fabm")
branch=("FVCOM-FABM" "master" "master")

mkdir $CODE_DIR
cd $CODE_DIR

echo "Obtaining source code"
for i in 0 1 2
do
    git clone ${website[i]} ${name[i]}
    cd ${name[i]}
    git checkout ${branch[i]}
    cd $CODE_DIR
done
```

(continues on next page)

(continued from previous page)

```

cd $SCRIPT_DIR

FABM=$CODE_DIR/fabm/src
ERSEM=$CODE_DIR/ersem
FABM_INSTALL=$INSTALL_DIR/FABM-ERSEM
FC=$(which mpiifort)

mkdir -p $FABM_INSTALL

cd $FABM
mkdir build
cd build
# Production config:
cmake $FABM -DFABM_HOST=fvcom -DFABM_ERSEM_BASE=$ERSEM -DCMAKE_Fortran_COMPILER=$FC -
↪DCMAKE_INSTALL_PREFIX=$FABM_INSTALL
make install -j $num_cpu

cd $SCRIPT_DIR

sed -i 's|BASE_SETUP_DIR|'"$SCRIPT_DIR"'|g' make_ideal_estuary.inc
ln -s $SCRIPT_DIR/make_ideal_estuary.inc $SCRIPT_DIR/code/uk-fvcom/FVCOM_source/make.inc

# Installing FVCOM additional packages (METIS, Proj, etc)
cd $SCRIPT_DIR/code/uk-fvcom/FVCOM_source/libs
mv makefile makefile_
ln -s makefile.CETO makefile
make -j $num_cpu

# Building FVCOM
cd ..
make -j $num_cpu
```

Here you will have to adapt the script to ensure you are using the right HPC modules and the corresponding compilers. The important compiler is the fortran one which is set with the variable *FC*.

Another key file to change is the `make.inc` file, here again you will need to change the compilers to reflect the modules you are using on the HPC machine. For example, on lines 74 and 75 `IOLIBS` and `IOINCS` are set, these would need to be change to reflect the modules on the HPC machine you are using.

`make.inc` file

```
#/==========================================================================/
#
# PML ceto makeinc for the ideal estuary model
#
#/==========================================================================/

#==============================================================================
#
#    WELCOME TO FVCOM
#
#    TO BUILD FVCOM, PLEASE SET THE FOLLOWING IN THIS FILE:
```

(continues on next page)

(continued from previous page)

```
#         TOPDIR - the directory this make.inc file is in
#         LIBDIR - the directroy where libraries are installed
#         INCIDR - the directroy where include files are installed
#
#         CHOOSE YOUR MODEL OPTIONAS - The Def Flags!
#
#         CHOOSE A COMPILER FROM THE LIST OR CREATE YOUR OWN
#
#    You can also use the makefile to build and install some of the libraries
#    used by fvcom. Set the INSTALLDIR and choose the 'LOCAL INSTALL' option
#    below. Select an non-mpi compiler from the list and run 'make libs_ext'
#
#    Good Luck!
#

#========== TOPDIR ===========================================================
# TOPDIR is the directory in which this make file and the fvcom source reside

            TOPDIR        =  BASE_SETUP_DIR/code/uk-fvcom/FVCOM_source
# TOPDIR must be set!

#========== INSTALLDIR =======================================================
# INSTALLDIR is the directory where you wish to install external libraries
# The default is in the $(TOPDIR)/libs/install, but you can choose...
             INSTALLDIR    = $(TOPDIR)/libs/install
#==============================================================================

# PREPROCESSOR OPTIONS FOR CPP
             DEF_FLAGS     = -P -traditional
#==============================================================================

######################### MEDM ENVIRONMENT ####################################
# Use the environmental variables, LIBPATH and INCLUDEPATH, set by MODULE
# to find the packages used by this build.
colon=:
empty=
dashI= $(empty) -I
dashL= $(empty) -L
# ### UNCOMMENT HERE!
#              LIBDIR       =  -L$(subst $(colon),$(dashL),$(LIBPATH))
#              INCDIR       =  -I$(subst $(colon),$(dashI),$(INCLUDEPATH))

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

# LOCAL INSTAL
              LIBDIR       =  -L$(INSTALLDIR)/lib
              INCDIR       =  -I$(INSTALLDIR)/include

#------------------------------------------------------------------------------
#  STANDARD LIBRARIES FOR DATA AND TIME IN fVCOM:
#
              DTLIBS       = -ljulian
```

(continues on next page)

(continued from previous page)

```
             DTINCS       =
#
#----------------------------------------------------------------------------
#        NETCDF OUTPUT      NETCDF IS NOW REQUIRED TO COMPILE FVCOM
#                           DUMP OUTPUT INTO NETCDF FILES (yes/no)
#                           REQUIRES SYSTEM DEPENDENT NETCDF LIBRARIES
#                           COMPILED WITH SAME F90 COMPILER
#                           SET PATH TO LIBRARIES WITH IOLIBS
#                           SET PATH TO INCLUDE FILES (netcdf.mod) WITH IOINCS
#----------------------------------------------------------------------------
             IOLIBS       = -L/gpfs1/apps/netcdf/intelmpi/lib -lnetcdff -lnetcdf -lhdf5_
↪hl -lhdf5 -lz -lm
             IOINCS       = -I/gpfs1/apps/netcdf/intelmpi/include -I/gpfs1/apps/hdf5/
↪intelmpi/include
#----------------------------------------------------------------------------


#============================================================================
#  BEGIN USER DEFINITION SECTION
#============================================================================
#        SELECT MODEL OPTIONS
#          SELECT FROM THE FOLLOWING OPTIONS BEFORE COMPILING CODE
#          SELECT/UNSELECT BY COMMENTING/UNCOMMENTING LINE (#)
#          CODE MUST BE CLEANED (with "make clean") AND
#          RECOMPILED IF NEW SET OF OPTIONS IS DESIRED
#----------------------------------------------------------------------------


#----------------------------------------------------------------------------
#        PRECISION          DEFAULT PRECISION: SINGLE
#                           UNCOMMENT TO SELECT DOUBLE PRECISION
#----------------------------------------------------------------------------

#            FLAG_1 =  -DDOUBLE_PRECISION

## SINGLE PRECISION OUTPUT FOR VISIT
            FLAG_1 =  -DDOUBLE_PRECISION -DSINGLE_OUTPUT -DNETCDF4_COMPRESSION
#            FLAG_1 =  -DSINGLE_OUTPUT -DNETCDF4_COMPRESSION

#----------------------------------------------------------------------------
#        SPHERICAL          SELECT SPHERICAL COORDINATES FOR INTEGRATION
#                           DEFAULT: CARTESIAN
#                           UNCOMMENT TO SELECT SPHERICAL COORDINATES
#----------------------------------------------------------------------------

#            FLAG_2 = -DSPHERICAL

#----------------------------------------------------------------------------
#        FLOODYING/DRYING   INCLUDE WET/DRY TREATMENT OF DOMAIN
#                           CAN BE ACTIVATED/DEACTIVATED AT RUN TIME WITH
#                           INPUT FILE CONTROL.  (SEE exa_run.dat) FILE
#                           DEFAULT: NO FLOODYING/DRYING INCLUDED
```

(continues on next page)

(continued from previous page)

```
#                           UNCOMMENT TO INCLUDE FLOODYING/DRYING
#-------------------------------------------------------------------------

             FLAG_3 = -DWET_DRY

#-------------------------------------------------------------------------
#        MULTI_PROCESSOR    INCLUDES PARALLELIZATION WITH MPI
#                           REQUIRES LINKING MPI LIBRARIES OR COMPILING
#                           WITH A PRELINKED SCRIPT (mpif90/mpf90/etc)
#                           DEFAULT: NO PARALLEL CAPABILITY
#                           UNCOMMENT TO INCLUDE MPI PARALLEL CAPABILITY
#-------------------------------------------------------------------------

             FLAG_4 = -DMULTIPROCESSOR
             PARLIB = -lmetis -L$(LIBDIR)

#-------------------------------------------------------------------------
#        WATER_QUALITY      INCLUDE EPA WATER QUALITY MOD
#                           CAN BE ACTIVATED/DEACTIVATED AT RUN TIME WITH
#                           VARIABLE WQM_ON IN INPUT FILE
#                           DEFAULT: NO WATER QUALITY MODEL
#                           UNCOMMENT TO INCLUDE WATER QUALITY MODEL
#-------------------------------------------------------------------------

#             FLAG_5 = -DWATER_QUALITY

#-------------------------------------------------------------------------
#        PROJECTION         A Fortran90 wrapper for the Cartographic projection
#                           Software, proj4.
#                           Proj can be obtained from:
#                           http://www.remotesensing.org/proj/
#                           Thanks to: USGS
#
#                           The Proj fortran bindings can be obtained from:
#                           http://forge.nesc.ac.uk/projects/glimmer/
#                           Thanks to: Magnus Hagdorn (Magnus.Hagdorn@ed.ac.uk)
#
#                           !! NOTE THAT THE PROJ 4 LIBRARY MUST BE IN YOUR
#                           LD_LIBRARY_PATH FOR DYNAMIC LOADING!!
#
#-------------------------------------------------------------------------

             FLAG_6 = -DPROJ

             PROJLIBS       = -L$(LIBDIR) -lfproj4 -lproj -lm
             PROJINCS       = -I$(INCDIR)

#-------------------------------------------------------------------------
#        DATA_ASSIMILATION  INCLUDE NUDGING BASED DATA ASSIMILATION FOR
#                           CURRENT/TEMP/SALINITY/SST
#                           CAN BE ACTIVATED/DEACTIVATED AT RUN TIME WITH
#                           INPUT FILE CONTROL.  (SEE exa_run.dat) FILE
```

(continues on next page)

(continued from previous page)

```
#                           DEFAULT: NO DATA ASSIMILATION INCLUDED
#                           UNCOMMENT TO INCLUDE DATA ASSIMILATION
#-------------------------------------------------------------------------

#             FLAG_7 = -DDATA_ASSIM
#             include ${PETSC_DIR}/bmake/common/variables
#             OILIB  = -lmkl_lapack -lmkl_em64t -lguide -lpthread

#             OILIB  = -L/gpfs1/apps/intel/mkl/lib/em64t/ -lmkl_lapack95_ilp64 -lmkl_
↪intel_ilp64 -lmkl_core -lmkl_sequential

# For Build on em64t computer (Guppy)
#             LIBDIR = $(LIBDIR) -L/usr/local/lib64
# For Build on Cluster (Typhoeus and Hydra)
#             LIBDIR = $(LIBDIR) -L/usr/local/lib/em64t
# For i386 computers at SMAST (salmon and minke)
#   NO NEED TO ADD ANYTHING LIBS ARE IN THE DEFAULT PATH

#-------------------------------------------------------------------------
#        IN UPWIND LEAST SQUARE SCHEME:
#        LIMITED_NO: NO LIMITATION
#        LIMITED_1 : FIRST ORDER LIMITATION
#        LIMITED_2 : SECOND ORDER LIMITATION(  )
#   !!!!!! ONLY ONE OF THE FLAGS BELOW CAN BE AND MUST BE CHOSEN
#-------------------------------------------------------------------------

            FLAG_8 =  -DLIMITED_NO

#-------------------------------------------------------------------------
#  Semi-Implicit time stepping method
#-------------------------------------------------------------------------

#             FLAG_9 = -DSEMI_IMPLICIT
#             include ${PETSC_DIR}/bmake/common/variables


#-------------------------------------------------------------------------
#       SOLID BOUNDARY     IF GCN, NO GHOST CELL
#                          IF GCY1, GHOST CELL IS SYMMETRIC RELATIVE TO BOUNDARY
#                          CELL EDGE
#                          IF GCY2, GHOST CELL IS SYMMETRIC RELATIVE TO MIDDLE
#                          POINT OF THE BOUNDARY CELL EDGE
#-------------------------------------------------------------------------

             FLAG_10  = -DGCN

#-------------------------------------------------------------------------
#        TURBULENCE MODEL   USE GOTM TURBULENCE MODEL INSTEAD OF THE ORIGINAL
#                           FVCOM MELLOR-YAMADA 2.5 IMPLEMENTATION
#                           UNCOMMENT TO USE GOTM TURBULENCE MODEL
#        NOTE: You Must Build GOTM 4.x, GOTM 3.x used a different
#              do_turbulence interface and will not work.
```

(continues on next page)

(continued from previous page)

```
#-----------------------------------------------------------------------

#            FLAG_11 = -DGOTM
#            GOTMLIB       = -L../GOTM_source/ -lturbulence -lutil -lmeanflow
#            GOTMINCS      = -I../GOTM_source/

#-----------------------------------------------------------------------
#       EQUILIBRIUM TIDE
#-----------------------------------------------------------------------

#            FLAG_12 = -DEQUI_TIDE

#-----------------------------------------------------------------------
#       ATMOSPHERIC TIDE
#-----------------------------------------------------------------------

#            FLAG_13 = -DATMO_TIDE

#-----------------------------------------------------------------------
#       RIVER DISTRIBUTION OPTION:
#       THE STANDARD NAME LIST USES A CHARACTER STRING TO SET A FUNCION
#       DISTROBUTION. YOU CAN OPTIONALLY SPECIFY TO USE THE OLD STYLE,
#       FLOATING POINT DISTROBUTION HERE. USE THIS WHEN CONVERTING OLD-STYLE
#       RIVER INPUT FILES!
#-----------------------------------------------------------------------

#            FLAG_14 = -DRIVER_FLOAT

#-----------------------------------------------------------------------
#        Using A fully multidimensional positive definite advection
#        transport algorithm with small implicit diffusion.
#        Based on Smolarkiewicz, P. K; Journal of Computational
#        Physics, 54, 325-362, 1984
#-----------------------------------------------------------------------

            FLAG_15 =  -DMPDATA -DTVD

#-----------------------------------------------------------------------
#         Run Two-D Barotropic Mode Only
#-----------------------------------------------------------------------

#             FLAG_16 = -DTWO_D_MODEL

#-----------------------------------------------------------------------
#         Output 2-D Momentum Balance Checking
#-----------------------------------------------------------------------

#            FLAG_17 = -DBALANCE_2D

#-----------------------------------------------------------------------
#           OPEN BOUNDARY FORCING TYPE
#           DEFAULT: OPEN BOUNDARY NODE WATER ELEVATION FORCING
```

(continues on next page)

(continued from previous page)

```
#           UNCOMMENT TO SELECT BOTH OPEN BOUNDARY NODE WATER ELEVATION
#           FORCING AND OPEN BOUNDARY VOLUME TRANSPORT FORCING
#-----------------------------------------------------------------------

#            FLAG_18 = -DMEAN_FLOW

#-----------------------------------------------------------------------
#           OUTPUT TIDAL INFORMATION AT NTIDENODE and NTIDECELL
#           FOR MEANFLOW CALCULATION.
#-----------------------------------------------------------------------

#            FLAG_19 = -DTIDE_OUTPUT

#-----------------------------------------------------------------------
#           dye release
#-----------------------------------------------------------------------

#            FLAG_20 = -DDYE_RELEASE

#-----------------------------------------------------------------------
#        SUSPENDED SEDIMENT MODEL:   UNCOMMENT TO INCLUDE MODEL
# ORIG : the sediment transport model developed by Geoffey Cowles in v3.1-v3.2
# CSTMS: Community Sediment Transport Modeling System with cohesive model
# DELFT: Sediment modeling system as Delft Flow (not included)
#
# note: only one model should be chosen for modeling.
#
#  Utilities:
#    OFFLINE_SEDIMENT : run sediment with offline hydrodynamic forcing
#    FLUID_MUD : activate the 2-D fluid mud at bed-water interface
#-----------------------------------------------------------------------

            FLAG_21 =  -DSEDIMENT
            FLAG_211 = -DORIG_SED
#            FLAG_211 = -DCSTMS_SED
#
#            FLAG_22 =  -DOFFLINE_SEDIMENT
#            FLAG_43 =  -DFLUID_MUD
#
#-----------------------------------------------------------------------
#         KALMAN FILTERS
#-----------------------------------------------------------------------

#              FLAG_23 = -DRRKF
#              FLAG_23 = -DENKF
#             include ${PETSC_DIR}/bmake/common/variables
#              KFLIB  = -lmkl_lapack -lmkl_em64t -lguide -llapack -lblas

# For Build on em64t computer (Guppy)
#             LIBDIR = $(LIBDIR) -L/usr/local/lib64
# For Build on Cluster (Typhoeus and Hydra)
#             LIBDIR = $(LIBDIR) -L/usr/local/lib/em64t
```

(continues on next page)

(continued from previous page)

```
# For i386 computers at SMAST (salmon and minke)
#   NO NEED TO ADD ANYTHING LIBS ARE IN THE DEFAULT PATH

#-------------------------------------------------------------------------
#         Run One-D Mode with Biological Model
#-------------------------------------------------------------------------

#             FLAG_24 = -DONE_D_MODEL

#-------------------------------------------------------------------------
#       GENERAL BIOLOGICAL MODEL:    UNCOMMENT TO INCLUDE MODEL
#-------------------------------------------------------------------------
            FLAG_25 = -DFABM  #-DOFFLINE_FABM  # -DFABM_DIAG_OUT
            BIOLIB       = -LBASE_SETUP_DIR/model/FABM-ERSEM/lib -lfabm
            BIOINCS      = -IBASE_SETUP_DIR/model/FABM-ERSEM/include -IBASE_SETUP_DIR/
↪model/FABM-ERSEM/include/yaml

#-------------------------------------------------------------------------
#         Dynamic/Thermodynamic Ice
#-------------------------------------------------------------------------

# NOTE: Must use -DSPHERICAL  and -DHEAT_FLUX ----- this note only for old version v2.7
#       ICE_EMBEDDING must with SEMI_IMPLICIT
#            FLAG_26 = -DICE
#            FLAG_261 = -DICE_EMBEDDING

#-------------------------------------------------------------------------
#         CALCULATE THE NET HEAT FLUX IN MODEL (THREE CHOICES):
#         1. CALCULATE THE NET HEAT FLUX USING COARE26Z
#         2. CALCULATE THE NET HEAT FLUX USING COARE26Z for Great Lake
#         3. CALCULATE THE NET HEAT FLUX USING SOLAR HEATING MODULE
↪
#-------------------------------------------------------------------------

#            FLAG_27 = -DHEATING_CALCULATED
#            FLAG_27 = -DHEATING_CALCULATED_GL
#            FLAG_27 = -DHEATING_SOLAR

#-------------------------------------------------------------------------
#         AIR_PRESSURE FROM SURFACE FORCING
#-------------------------------------------------------------------------

#             FLAG_28 = -DAIR_PRESSURE

#-------------------------------------------------------------------------
# Visit online simulation mode
#-------------------------------------------------------------------------

#            FLAG_29 = -DVISIT

#      VISITLIB     = -lm -ldl -lsimf -lsim -lpthread
#      VISITLIBPATH = $(LIBDIR)
```

(continues on next page)

(continued from previous page)

```
#       VISITINCPATH = $(INCDIR)


# USE DEVELOPER INSTALL VISIT
#       VISITLIBPATH =
#       VISITLIB     =  -lm -ldl -lsimf -lsim -lpthread
#       VISITINC     =

#-------------------------------------------------------------------------
#         NON-HYDROSTATIC MODEL:
#-------------------------------------------------------------------------

#             FLAG_30 =  -DNH
#             include ${PETSC_DIR}/bmake/common/variables

#-------------------------------------------------------------------------
#         PARTICLE TRACKING
#-------------------------------------------------------------------------

#             FLAG_31 =  -DLAG_PARTICLE

#-------------------------------------------------------------------------
#         WAVE-CURRENT INTERACTION
#-------------------------------------------------------------------------
#             FLAG_32 =  -DWAVE_CURRENT_INTERACTION
#             FLAG_33 =  -DPLBC
#           NOTE! THis option is for wave code
#             FLAG_34 =  -DEXPLICIT
#           WAVE ONLY
#             FLAG_35 = -DWAVE_ONLY
# Svendsen Roller contribution
#             FLAG_36 = -DWAVE_ROLLER
#             FLAG_37 = -DWAVE_OFFLINE
#             include ${PETSC_DIR}/bmake/common/variables
#-------------------------------------------------------------------------
#         THIN-DAM MODEL
#-------------------------------------------------------------------------
#             FLAG_38 = -DTHIN_DAM

#-------------------------------------------------------------------------
#         PWP MIXED LAYER MODEL:
#-------------------------------------------------------------------------

#            FLAG_39 =  -DPWP

#-------------------------------------------------------------------------
#         VERTICAL ADVECTION LIMITER:
#         FOR S-COORDINATES, DON'T USE THIS FLAG
#-------------------------------------------------------------------------

#           FLAG_40 =  -DLIMITER_VER_ADV

```

(continues on next page)

(continued from previous page)

```
#-----------------------------------------------------------------------
#        PETSC Version
#        If your PETSc is 2.3.2 or older, uncomment this flag
#-----------------------------------------------------------------------
#             FLAG_41 = -DOLD_PETSC

#-----------------------------------------------------------------------
#        SPECIAL PARTITION
#        This flag can make sure the identical repeat run for same amount of CPUs
#-----------------------------------------------------------------------
            FLAG_42 = -DPARTITION_SPECIAL

#-----------------------------------------------------------------------
#        DEVELOPMENT FLAGS
#            FOR BETA WORK ONLY
#-----------------------------------------------------------------------

#             FLAG_101 = -DDEVELOP1
#             FLAG_102 = -DDEVELOP2
#             FLAG_103 = -DDEVELOP3
#             FLAG_104 = -DDEVELOP4
#             FLAG_105 = -DDEVELOP5

#-----------------------------------------------------------------------
#        SELECT COMPILER/PLATFORM SPECIFIC DEFINITIONS
#          SELECT FROM THE FOLLOWING PLATFORMS OR USE "OTHER" TO DEFINE
#          THE FOLLOWING VARIABLES:
#          CPP:  PATH TO C PREPROCESSOR
#           FC:  PATH TO FORTRAN COMPILER (OR MPI COMPILE SCRIPT)
#          OPT:  COMPILER OPTIONS
#       MPILIB:  PATH TO MPI LIBRARIES (IF NOT LINKED THROUGH COMPILE SCRIPT)
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
#  Intel/MPI Compiler Definitions (PML)
#-----------------------------------------------------------------------
         CPP      = mpiicc -E
         COMPILER = -DINTEL
         FC       = mpiifort
         DEBFLGS  =  #-check all
         OPT      = -O3 -L/gpfs1/apps/intel/compilers_and_libraries/linux/mpi/intel64/
↪lib -I/gpfs1/apps/intel/compilers_and_libraries/linux/mpi/intel64/include/ -xHost #-
↪init=zero -init=arrays -ftrapuv
         CLIB     =
         CC       = mpiicc
         CFLAGS   =
#-----------------------------------------------------------------------
#  Intel/MPI Compiler Definitions (PML) Debugging
#-----------------------------------------------------------------------
#         COMPILER = -DIFORT
#         CPP      = /lib/cpp
#         CPPFLAGS = $(DEF_FLAGS) -P -traditional -DINTEL CPPMACH=-DNOGUI -I/opt/mpi/
↪mpibull2-current/include
```

(continues on next page)

(continued from previous page)

```
#         FC       = ifort
#         DEBFLGS  = #-check all
#         OPT      =  -I/opt/mpi/mpibull2-current/include -g -traceback -warn -nofor_
↪main -fp-model precise -traceback -fpe0 -keep
#         OILIB    = -L/opt/intel/cmkl/10.0.1.014/lib/em64t -Wl,-rpath=/opt/intel/cmkl/10.
↪0.1.014/lib/em64t -i-static -L/opt/mpi/mpibull2-current/lib -lmpi -L/usr/lib64 -
↪libverbs -L/opt/mpi/mpibull2-current/lib/pmi -lpmi
#         CC       = icc
#         CFLAGS   = -g -I/opt/mpi/mpibull2-current/include -traceback
#-----------------------------------------------------------------------
#        COMPAQ/ALPHA Definitions
#-----------------------------------------------------------------------
#        COMPILER   = -DCOMPAQ
#        CPP        = /bin/cpp
#        FC         = f90
#        DEBFLGS    =  # -check bounds -check overflow -g
#        OPT        =  -fast -arch ev6 -fpe1
#-----------------------------------------------------------------------
#  CRAY Definitions
#-----------------------------------------------------------------------
#         COMPILER = -DCRAY
#         CPP      = /opt/ctl/bin/cpp
#         FC       = f90
#         DEBFLGS  =
#         OPT      =
#-----------------------------------------------------------------------
#   Linux/Portland Group Definitions
#-----------------------------------------------------------------------
#         CPP      = /usr/bin/cpp
#         COMPILER =
#         FC       = pgf90
#         DEBFLGS  = -Mbounds -g -Mprof=func
#         OPT      = #-fast  -Mvect=assoc,cachesize:512000,sse
#-----------------------------------------------------------------------
#  Intel Compiler Definitions
#-----------------------------------------------------------------------
#         CPP      = /usr/bin/cpp
#         COMPILER = -DIFORT
#         FC       = ifort
#         CC       = icc
#         CXX      = icc
#         CFLAGS   = -O3
#         DEBFLGS  = #-check all
# Use 'OPT = -O0 -g' for fast compile toBASE_SETUP_DIRthe make
# Use 'OPT = -xP' for fast run on em64t (Hydra and Guppy)
# Use 'OPT = -xN' for fast run on ia32 (Salmon and Minke)
#         OPT      = -O0 -g
#         OPT      = -xP
# Do not set static for use with visit!
#         VISOPT   = -Wl,--export-dynamic
#         LDFLAGS  = $(VISITLIBPATH)
#-----------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
#  Intel/MPI Compiler Definitions (SMAST)
#---------------------------------------------------------------------
#         CPP      = /usr/bin/cpp
#         COMPILER = -DIFORT
#         CC       = mpicc
#         CXX      = mpicxx
#         CFLAGS   = -O3
#         FC       = mpif90
#         DEBFLGS  = -check all -traceback
# Use 'OPT = -O0 -g'  for fast compile toBASE_SETUP_DIRthe make
# Use 'OPT = -xP' for fast run on em64t (Hydra and Guppy)
# Use 'OPT = -xN' for fast run on ia32 (Salmon and Minke)
#         OPT      = -O0 -g
#         OPT      = -axN -xN
#         OPT      = -O3
# Do not set static for use with visit!
#         VISOPT   = -Wl,--export-dynamic
#         LDFLAGS  =  $(VISITLIBPATH)
#---------------------------------------------------------------------
#  gfortran defs
#---------------------------------------------------------------------
#         CPP      = /usr/bin/cpp
#         COMPILER = -DGFORTRAN
#         FC       = gfortran  -O3
#         DEBFLGS  =
#         OPT      =
#         CLIB     =
#---------------------------------------------------------------------
#  absoft / mac os x defs
#---------------------------------------------------------------------
#         CPP      = /usr/bin/cpp
#         COMPILER = -DABSOFT
#         FC       = f90  -O3  -lU77
#         DEBFLGS  =
#         OPT      =
#         CLIB     =
#---------------------------------------------------------------------
#  IBM/AIX Definitions
#---------------------------------------------------------------------
#         COMPILER = -DAIX
#         CPP      = /usr/local/bin/cpp
#         FC       = mpxlf90 -qsuffix=f=f90
#         DEBFLGS  =  # -qcheck -C -g
#         OPT      = -O -qarch=pwr4 -qtune=pwr4  -bmaxdata:0x80000000 -qhot -qmaxmem=8096
#---------------------------------------------------------------------
#  APPLE OS X/XLF Definitions (G5)
#---------------------------------------------------------------------
#         COMPILER = -DAIX
#         CPP      = /usr/bin/cpp
#         FC       =  /opt/ibmcmp/xlf/8.1/bin/xlf90 -qsuffix=f=f90
#         DEBFLGS  =  # -qcheck -C -g
#         OPT      = -O5 -qarch=g5 -qtune=g5  -qhot -qmaxmem=8096 -qunroll=yes -Wl,-
↪stack_size,10000000
```

(continues on next page)

(continued from previous page)

```
#-----------------------------------------------------------------------
#  ARCHER Intel/MPI Compiler
#-----------------------------------------------------------------------
#         CPP      = /usr/bin/cpp
#         COMPILER = -DIFORT
#         CC       = cc
#         CXX      = CC
#         CFLAGS   = -O3
#         FC       = ftn
##         DEBFLGS  = -check all
#         OPT      = -O3
#         COPTIONS = -c89
#-----------------------------------------------------------------------
#=======================================================================
#  END USER DEFINITION SECTION
#=======================================================================
         CPPFLAGS = $(DEF_FLAGS) $(COMPILER)
         FFLAGS = $(DEBFLGS) $(OPT)
         MDEPFLAGS = --cpp --fext=f90 --file=-
         RANLIB = ranlib
         AR = ar rc
#-----------------------------------------------------------------------
#  CAT Preprocessing Flags
#-----------------------------------------------------------------------
           CPPARGS =  $(CPPFLAGS) $(DEF_FLAGS) $(FLAG_1) $(FLAG_2) \
   $(FLAG_3) $(FLAG_4) $(FLAG_5) $(FLAG_6)  \
   $(FLAG_7) $(FLAG_8) $(FLAG_9) $(FLAG_10) \
   $(FLAG_11) $(FLAG_12) $(FLAG_13) $(FLAG_14) \
   $(FLAG_15) $(FLAG_16) $(FLAG_17) $(FLAG_18) \
   $(FLAG_19) $(FLAG_20) $(FLAG_21) $(FLAG_22) \
   $(FLAG_23) $(FLAG_24) $(FLAG_25) $(FLAG_26) \
   $(FLAG_27) $(FLAG_28) $(FLAG_29) $(FLAG_30) \
   $(FLAG_31) $(FLAG_32) $(FLAG_33) $(FLAG_34) \
   $(FLAG_35) $(FLAG_36) $(FLAG_37) $(FLAG_38) \
   $(FLAG_39) $(FLAG_40) $(FLAG_41) $(FLAG_42) \
   $(FLAG_43) $(FLAG_LAM)\
   $(FLAG_101) $(FLAG_102) $(FLAG_103) $(FLAG_104) $(FLAG_105)\
   $(FLAG_211) $(FLAG_212) $(FLAG_213) $(FLAG_251) $(FLAG_261)
#-----------------------------------------------------------------------
#  Libraries
#-----------------------------------------------------------------------

           LIBS  =        $(LIBDIR) $(CLIB)  $(PARLIB) $(IOLIBS)        $(DTLIBS)\
                       $(MPILIB) $(GOTMLIB) $(KFLIB) $(BIOLIB)        \
                       $(OILIB) $(VISITLIB) $(PROJLIBS) $(PETSC_LIB)

           INCS  =         $(INCDIR) $(IOINCS) $(GOTMINCS) $(BIOINCS)\
                        $(VISITINCPATH) $(PROJINCS) $(DTINCS) \
                        $(PETSC_FC_INCLUDES)
```

The key lines to change are:

- 74–75

- 174
- 458–465

After building FVCOM-ERSEM, we suggest you use a HPC scheduler, for example, SLURM to run example. An example of the SLURM script used here is given below:

```
#!/bin/bash --login

#SBATCH --nodes=4
#SBATCH --ntasks-per-node=20
#SBATCH --threads-per-core=1
#SBATCH --job-name=estuary
#SBATCH --partition=all
#SBATCH --time=48:00:00
##SBATCH --mail-type=ALL
##SBATCH --mail-user=your_mail@pml.ac.uk

# Set the number of processes based on the number of nodes we have `select'ed.
np=$SLURM_NTASKS

# Export the libraries to LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$(readlink -f $WORKDIR/install/lib):$LD_LIBRARY_PATH

set -eu
ulimit -s unlimited

# Number of months to skip
skip=0

# d53f5083 = FVCOM v3.2.2, M-Y, HEATING_ON, ceto
binary=bin/fvcom
grid=${grid:-"estuary"}
casename="${grid}"

# Make sure any symbolic links are resolved to absolute path
export WORKDIR=$(readlink -f $(pwd))

# Set the number of threads to 1
#   This prevents any system libraries from automatically
#   using threading.
export OMP_NUM_THREADS=1

# Magic stuff from the Atos scripts.
export I_MPI_PIN_PROCS=0-19
export I_MPI_EXTRA_FILESYSTEM=on
export I_MPI_EXTRA_FILESYSTEM_LIST=gpfs
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so

# Change to the directory from which the job was submitted. This should be the
# project "run" directory as all the paths are assumed relative to that.
cd $WORKDIR

if [ ! -d ./logs/slurm ]; then
```

(continues on next page)

(continued from previous page)

```
    mkdir -p ./logs/slurm
fi
mv *.out logs/slurm || true
if [ -f ./core ]; then
    rm core
fi

if [ ! -d ./output ]; then
    mkdir -p ./output
fi

# Iterate over the months in the year

    # Launch the parallel job
    srun -n $np $binary --casename=$casename --dbg=0 > logs/${casename}-$SLURM_JOBID.log


    # Check if we crashed and if so, exit the script, bypassing the restart
    # file creation.
    if grep -q NaN logs/${casename}-$SLURM_JOBID.log; then
        echo "NaNs in the output. Halting run."
        exit 2
    fi
```

Example output from FVCOM-ERSEM

We provide two python scripts to demonstrate how to visualise both the input files and the output files. The plotting uses PyFVCOM, we suggest you ask for access here, however, a version of the code is available on GitHub as well as it being pip installable.

python input plot script

```
import os
import numpy as np
from netCDF4 import Dataset, num2date
from PyFVCOM.read import MFileReader
import matplotlib.pylab as plt

# simple grid
dir_path = os.path.dirname(os.path.realpath(__file__))
estuary_path = os.path.join('model',
                            'estuary')
figure_path = os.path.join(estuary_path,
                           "figures")
fvcom_files = os.path.join(estuary_path,
                           'output',
                           'estuary_avg_0001.nc')


if not os.path.isdir(figure_path):
    os.makedirs(figure_path)
```

(continues on next page)

(continued from previous page)

```

print("Plotting mesh")
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111)
dims = {'siglay': [0]}
fvcom = MFileReader(fvcom_files, dims=dims)
DATA = fvcom.grid.h
p1 = ax.tripcolor(fvcom.grid.x, fvcom.grid.y,
                  fvcom.grid.triangles, DATA, shading='flat', edgecolors='k')
plt.colorbar(p1)
ax.set_aspect('equal', 'box')
name = os.path.join(figure_path, 'bathymetry')
fig.savefig(name, bbox_inches='tight', pad_inches=0.2, dpi=300)
plt.close()

print("Plotting wind forcing")
# check forcing
idtime = 100
fvcom_files = os.path.join(estuary_path,
                           'input',
                           '2D_forcing_50_years_20180927_new_grd.nc')
fvcom_frc = MFileReader(fvcom_files, variables=['uwind_speed',
                                                'vwind_speed',
                                                'short_wave',
                                                'net_heat_flux'])
uwind = np.squeeze(fvcom_frc.data.uwind_speed[idtime, :])
vwind = np.squeeze(fvcom_frc.data.vwind_speed[idtime, :])

# wind speed
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111)
scale = 75
p1 = ax.quiver(fvcom.grid.xc[0:-1:2],
               fvcom.grid.yc[0:-1:2],
               uwind[0:-1:2],
               vwind[0:-1:2],
               scale=scale,
               headlength=8,
               headaxislength=8,
               width=0.0015)

arrow_legend = 2
plt.text(5e4, 6e4, '{} m/s'.format(str(arrow_legend)), fontsize=10)
q = ax.quiver(5e4, 5e4, arrow_legend, 0, scale=scale,
              headlength=8, headaxislength=8, width=0.0015)
ax.set_aspect('equal', 'box')
name = os.path.join(figure_path, 'wind_forcing')
fig.savefig(name, bbox_inches='tight', pad_inches=0.2, dpi=300)
plt.close()

print("Plotting short wave")
# short wave
```

(continues on next page)

(continued from previous page)

```
fvcom = fvcom_frc
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111)
idnode = 500
shortwave = getattr(fvcom_frc.data, 'short_wave')
ax.plot(fvcom.time.datetime[0:8*365*1], shortwave[0:8*365*1, idnode])
ax.set_xlabel('Time (DD-MM HH)', fontsize='x-large')
ax.set_ylabel('W/m2')
name = os.path.join(figure_path, 'short_wave')
fig.savefig(name, bbox_inches='tight', pad_inches=0.2, dpi=300)
plt.close()


print("Plotting net heat flux")
# net heat flux
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111)
idnode = 500
netheat = getattr(fvcom_frc.data, 'net_heat_flux')
ax.plot(fvcom.time.datetime, netheat[:, idnode])
ax.set_xlabel('Time (year)', fontsize='x-large')
ax.set_ylabel('W/m2')
name = os.path.join(figure_path, 'net_heat_flux')
fig.savefig(name, bbox_inches='tight', pad_inches=0.2, dpi=300)
plt.close()

print("Plotting OBC ts")
# OBC T S
obc_node = 40
fvcom_files = os.path.join(estuary_path,
                           'output',
                           'estuary_avg_0001.nc')
fvcom = MFileReader(fvcom_files, dims={'time': range(1)})
fvcom_files_obc = os.path.join(estuary_path,
                               'input',
                               '2D_obc_ts_forcing.nc')
nc = Dataset(fvcom_files_obc).variables
fig = plt.figure(figsize=(10, 5))

ax = fig.add_subplot(121)
month = np.arange(1, 5)
for month in month:
    obc_temp = nc['obc_temp'][month+12*0, :, obc_node]
    depth = fvcom.grid.siglay_z[:, obc_node]
    if month in np.arange(1, 7):
        ax.plot(obc_temp, -depth, label=month)
    else:
        ax.plot(obc_temp, -depth, '--', label=month)
    ax.legend(loc='upper right')
    ax.set_xlim(4.5, 33)
    ax.set_xlabel('Temp (degC)')
    ax.set_ylabel('Depth (m)')
```

(continues on next page)

(continued from previous page)

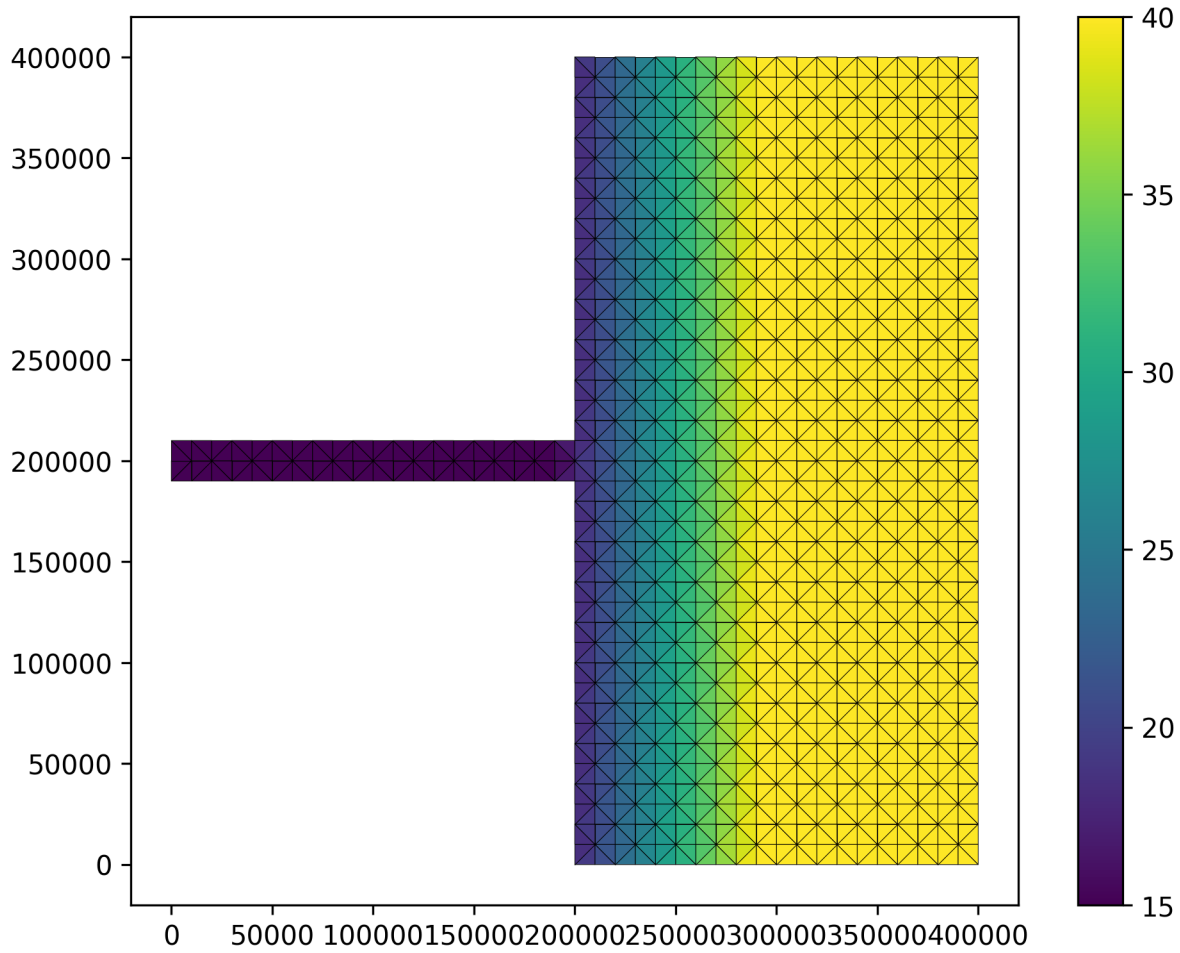
```
    ax.set_title('obc node {}'.format(str(obc_node)))

ax = fig.add_subplot(122)
month = np.arange(1, 5)
for month in month:
    obc_salt = nc['obc_salinity'][month+12*0, :, obc_node]
    depth = fvcom.grid.siglay_z[:, obc_node]
    if month in np.arange(1, 7):
        ax.plot(obc_salt, -depth, label=month)
    else:
        ax.plot(obc_salt, -depth, '--', label=month)
    ax.legend(loc='upper right')
    ax.set_xlim(28, 37)
    ax.set_xlabel('Salinity (psu)')
    ax.set_ylabel('Depth (m)')
    ax.set_title('obc node {}'.format(str(obc_node)))
name = os.path.join(figure_path, 'obc_TS_node_{}'.format(str(obc_node)))
fig.savefig(name, bbox_inches='tight', pad_inches=0.2, dpi=300)
plt.close()

print("Plotting river forcing")
# River forcing
tracer_f = "2D_bio_River_50_years_20180821_monthly_river_2_tracers_T1T2.nc"
fvcom_file_river = \
    os.path.join(estuary_path,
                 'input',
                 tracer_f)
nc = Dataset(fvcom_file_river)
nc.variables.keys()
varlist = ['river_flux', 'river_temp', 'river_salt', 'river_sed',
           'N4_n', 'N3_n']
fig = plt.figure(figsize=(12, 8))
time_var = nc['time']
dtime = num2date(time_var[:], time_var.units)
for var, num in zip(varlist, np.arange(1, 3)):
    ax = fig.add_subplot(3, 2, num)
    if var in ('river_flux'):
        river_var = np.sum(nc[var][:, :], 1)
    else:
        river_var = nc[var][:, 0]
    ax.plot(dtime, river_var)
    ax.set_title('{} ({})'.format(nc[var].long_name, nc[var].units))
    ax.set_xlabel('Time (month)')
    plt.tight_layout()
name = os.path.join(figure_path, 'river_flux1')
fig.savefig(name, bbox_inches='tight', pad_inches=0.2, dpi=300)
plt.close()

varlist = ['N1_p', 'N5_s', 'O3_c', 'O3_TA', 'O3_bioalk', 'O2_o']
fig = plt.figure(figsize=(12, 8))
for var, num in zip(varlist, np.arange(1, 3)):
    ax = fig.add_subplot(3, 2, num)
```

(continues on next page)

(continued from previous page)

```
    river_var = nc[var][:, 0]
    ax.plot(dtime, river_var)
    ax.set_title('{} ({})'.format(nc[var].long_name, nc[var].units))
    ax.set_xlabel('Time (month)')
    plt.tight_layout()
name = os.path.join(figure_path, 'river_flux2')
fig.savefig(name, bbox_inches='tight', pad_inches=0.2, dpi=300)
plt.close()
```

Using the input python plot script, we generate the domain as follows:



python output plot script

```
import os
import glob
import numpy as np

from PyFVCOM.read import MFileReader
from pythontools.visual.utils import colourmap
import PyFVCOM as pf
```

(continues on next page)

(continued from previous page)

```
from PyFVCOM.plot import Plotter, Depth
import matplotlib.pyplot as plt
import matplotlib

project0 = 'estuary'
casename = 'ideal'
base = os.path.join('model', 'estuary')
experiment = ''
suffix = ''
level_label = 'vertical_transect'
level_label2 = 'surface'
fname = 'estuary_avg_0001.nc'
fpath1 = os.path.join(base, 'output', fname)
fvcom_files = sorted(glob.glob(fpath1))

noisy = True
if noisy:
    print(fpath1, flush=True)
    print(fvcom_files, flush=True)

if not fvcom_files:
    raise Exception('Cannot find FVCOM output: {}'.format(fpath1))

MyFile = fvcom_files

plot_map = True  # to plot horizontal map
save_plot = True  # to save the plot

fvcom = MFileReader(MyFile)

varlist = ['GPP', 'temp', 'salinity',
           'P1_Chl', 'O3_c', 'R6_c', 'N1_p', 'N3_n',
           'tracer1_c', 'total_chl']

clims = {
    'temp': [5, 25],
    'O3_c': [1900, 2300],
    'R6_c': [0, 200],
    'salinity': [0, 31]
}

# plot all time series, maybe too much
time_indices = range(0, fvcom.dims.time)
level = 0  # select vertical layer (0: surface, -1: bottom) if plot_map = True

plt.rcParams['axes.facecolor'] = '0.6'
for var in varlist:
    baseDir = os.path.join(base, 'figures', var)
    baseName = os.path.join(baseDir, var)

    if var in ('uv'):
        plotvars = ['u', 'v']
```

(continues on next page)

(continued from previous page)

```
    elif var == 'total_chl':
        plotvars = ['P1_Chl', 'P2_Chl', 'P3_Chl', 'P4_Chl']
    elif var == 'netPP':
        plotvars = ['P1_Chl', 'P2_Chl', 'P3_Chl', 'P4_Chl']
    elif var == 'GPP':
        plotvars = ["P1_fO3PIc", "P2_fO3PIc", "P3_fO3PIc", "P4_fO3PIc"]
    else:
        plotvars = [var]

    if noisy:
        print('Loading {} data from netCDF... '.format(var),
              end='',
              flush=True)
        if save_plot:
            print('Saving files to {}'.format(baseDir), flush=True)
            if not os.path.isdir(baseDir):
                os.makedirs(baseDir)
        print('Working on {}...'.format(var), flush=True),

    fvcom = MFileReader(MyFile, variables=plotvars)
    fvcom.grid.lon = fvcom.grid.x
    fvcom.grid.lat = fvcom.grid.y
    positions = np.array(((0, 2e5), (4e5, 2e5)))
    indices, distance = fvcom.horizontal_transect_nodes(positions)
    cmap = colourmap(var)

    if var in ('total_chl'):
        setattr(
            fvcom.data, var, fvcom.data.P1_Chl + fvcom.data.P2_Chl +
            fvcom.data.P3_Chl + fvcom.data.P4_Chl)
        attributes = type('attributes', (object, ), {})()
        setattr(attributes, 'long_name', 'Total chlorophyll')
        setattr(attributes, 'units', fvcom.atts.P1_Chl.units)
        setattr(fvcom.atts, var, attributes)
    elif var in ('total_food'):
        setattr(fvcom.data, var, fvcom.data.P1_c + fvcom.data.P2_c
                + fvcom.data.P3_c + fvcom.data.P4_c + fvcom.data.Z5_c
                + fvcom.data.Z5_c + fvcom.data.R8_c + fvcom.data.R6_c
                + fvcom.data.R4_c)
        attributes = pf.utilities.general.PassiveStore()
        setattr(attributes, 'long_name', 'Total mussel food ')
        setattr(attributes, 'units', fvcom.atts.P1_c.units)
        setattr(fvcom.atts, var, attributes)
        attributes = pf.utilities.general.PassiveStore()
        setattr(fvcom.data, var, getattr(fvcom.data, var) / 1000000)
        setattr(attributes, 'long_name', 'Integrated Total mussel food')
        setattr(attributes, 'units', 'Kg\C')
        setattr(fvcom.atts, var, attributes)
    elif var in ('GPP'):
        setattr(fvcom.data, var, fvcom.data.P1_fO3PIc +
                fvcom.data.P2_fO3PIc + fvcom.data.P3_fO3PIc +
                fvcom.data.P4_fO3PIc)
```

(continues on next page)

(continued from previous page)

```
        attributes = pf.utilities.general.PassiveStore()
        setattr(attributes, 'long_name', 'Gross Primary Productivity')
        setattr(attributes, 'units', fvcom.atts.P1_fO3PIc.units)
        setattr(fvcom.atts, var, attributes)

    if var in clims:
        clim = clims[var]
    else:
        clim = [
            np.nanpercentile(getattr(fvcom.data, var), 3),
            np.nanpercentile(getattr(fvcom.data, var), 97)
        ]
    plot = Plotter(fvcom,
                   figsize=(20, 20),
                   res='i',
                   cb_label='{} ({})'.format(
                       getattr(fvcom.atts, var).long_name,
                       getattr(fvcom.atts, var).units),
                   cmap=cmap,
                   vmin=clim[0],
                   vmax=clim[1],
                   cartesian=True)



    # Plot a temperature transect between two locations.
    depth_plot = Depth(fvcom,
                       figsize=(20, 9),
                       cb_label='{} ({})'.format(
                            getattr(fvcom.atts, var).long_name,
                            getattr(fvcom.atts, var).units),
                       cmap=cmap)
    depth_plot.axes.set_xlabel('Distance (km)')
    depth_plot.axes.set_ylabel('Depth (m)')
    for ntime in time_indices:
        print("Time step index: {}".format(ntime))

        # Make a plot of the surface temperature.
        plot.plot_field(np.squeeze(getattr(fvcom.data, var))[ntime, level, :])
        plot.axes.set_title(
            fvcom.time.datetime[ntime].strftime('%Y-%m-%d %H:%M:%S'))

        if save_plot:
            plot.figure.savefig('{}{}_{}_{}_{:04d}.png'.format(
                baseName, suffix, experiment, level_label2, ntime),
                bbox_inches='tight',
                pad_inches=0.2,
                dpi=120)

        # fill_seabed makes the part of the plot below the seabed grey.
        # plot.plot_slice(distance / 1000,  # to kilometres from metres
        depth_plot.plot_slice(
```

(continues on next page)

(continued from previous page)

```
            fvcom.grid.lon[indices] / 1000,  # to kilometres from metres
            fvcom.grid.siglay_z[:, indices],
            getattr(fvcom.data, var)[ntime, :, indices],
            fill_seabed=True,
            vmin=clim[0],
            vmax=clim[1])
        depth_plot.axes.set_title(
            fvcom.time.datetime[ntime].strftime('%Y-%m-%d %H:%M:%S'))
        depth_plot.axes.set_xlim(
            right=(fvcom.grid.lon[indices] /
                   1000).max())  # set the x-axis to the data range

        if save_plot:
            depth_plot.figure.savefig('{}{}_{}_{}_{:04d}.png'.format(
                baseName, suffix, experiment, level_label, ntime),
                bbox_inches='tight',
                pad_inches=0.2,
                dpi=120)
```

The following plots videos are produced from the plots produced by the `python` output plot script.

Gros primary production

Phosphate phosphorus

Nitrate nitrogen

Carbonate total dissolved inorganic carbon

Diatoms chlorophyll

Medium-sized POM carbon

Salinity

Temperature

Total chorophyll

Tracer1 concentration

2.2.5 NEMO: Atlantic Margin Model

This tutorial gives an end-to-end example on how to install and then run NEMO-ERSEM on the 7 km Atlantic Margin Model 7 (AMM7) domain using a high performance computing (HPC) machine. Here we have used UK National Supercomputing Service ARCHER2 with a singularity container with NEMO-ERSEM installed on it.

This tutorial is based on utilising a singularity container with a NEMO-ERSEM install within. The configuration scripts used in the installation can be found here. You will also need access to the shared folder within the ARCHER2 project id *n01* to obtain the forcing files.

Obtaining NEMO-ERSEM container

The key packages that are installed into NEMO-ERSEM container are:

- NEMO (physical ocean model)
- XIOS (input/output server)
- FABM (biogeochemical framework)
- ERSEM (biogeochemical model)

NEMO, ERSEM and FABM are freely available on GitHub and XIOS is available through the NEMO consortium svn server. Full instructions how to build the container can be found here. We note these instructions are based on the "Containerisation of NEMO Employing Singularity" (CoNES) repository.

Running the container on ARCHER2

The instructions to generate the slurm HPC scheduling scripts can be found here. These are used to run the model on ARCHER2.

The scripts generated via mkslurm require two executables, namely *xios_server.exe* and *nemo*. In a native installation, one would simply link the *XIOS* and *NEMO* executables to *xios_server.exe* and *nemo*. However, when using the singularity container, you are required to create *xios_server.exe* and *nemo* executables based on the following bash script:

Listing 1: Example script to run container. User is required to change *[RUNING-DIR]* and *[INPUT-DIR]* to the corresponding directories, and *[NEMO-XIOS]* to either *nemo* or *xios*.

```
#! /bin/bash
BIND_OPTS="-B [RUNING-DIR],"
BIND_OPTS="${BIND_OPTS}[INPUT-DIR],"
BIND_OPTS="${BIND_OPTS}/usr/lib64/liblustreapi.so:/opt/lib64/liblustreapi.so,"
BIND_OPTS="${BIND_OPTS}/usr/lib64/liblustreapi.so.1:/opt/lib64/liblustreapi.so.1,"
BIND_OPTS="${BIND_OPTS}/usr/lib64/liblustreapi.so.1.0.0:/opt/lib64/liblustreapi.so.1.0.0,
↪"
BIND_OPTS="${BIND_OPTS}/opt/cray/libfabric/1.11.0.4.71/lib64/:/opt/fabric,"
BIND_OPTS="${BIND_OPTS}/usr/lib64/libpals.a:/opt/lib64/libpals.a,"
BIND_OPTS="${BIND_OPTS}/usr/lib64/libpals.so:/opt/lib64/libpals.so,"
BIND_OPTS="${BIND_OPTS}/usr/lib64/libpals.so.0:/opt/lib64/libpals.so.0,"
BIND_OPTS="${BIND_OPTS}/usr/lib64/libpals.so.0.0.0:/opt/lib64/libpals.so.0.0.0,"
BIND_OPTS="${BIND_OPTS}/opt/cray/pe/mpich/8.1.9/ofi/gnu/9.1:/opt/mpi/install/,"
BIND_OPTS="${BIND_OPTS}/opt/cray/pe/pmi/6.0.13/lib,"
BIND_OPTS="${BIND_OPTS}/var/spool/slurmd/mpi_cray_shasta"
```

(continues on next page)

(continued from previous page)

```
export SINGULARITYENV_LD_LIBRARY_PATH=/opt/hdf5/install/lib:$SINGULARITYENV_LD_LIBRARY_
↪PATH
export SINGULARITYENV_LD_LIBRARY_PATH=/opt/mpi/install/lib-abi-mpich:$SINGULARITYENV_LD_
↪LIBRARY_PATH
export SINGULARITYENV_LD_LIBRARY_PATH=/.singularity.d/libs:$SINGULARITYENV_LD_LIBRARY_
↪PATH
export SINGULARITYENV_LD_LIBRARY_PATH=/opt/cray/pe/pmi/6.0.13/lib:$SINGULARITYENV_LD_
↪LIBRARY_PATH
export SINGULARITYENV_LD_LIBRARY_PATH=/opt/lib64:$SINGULARITYENV_LD_LIBRARY_PATH
export SINGULARITYENV_LD_LIBRARY_PATH=/opt/fabric:$SINGULARITYENV_LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/opt/hdf5/install/lib
export OMP_NUM_THREADS=1

singularity run ${BIND_OPTS} --home [RUNING-DIR] nemo.sif [NEMO-XIOS]
```

The CoNES documentation gives additional information on how to run NEMO singularity containers.

Example output from NEMO-ERSEM

To visualise NEMO output, we recommend using nctoolkit. A basic example of how to use *nctoolkit* is given below. Within the documentation of *nctoolkit* one will find additional example uses.

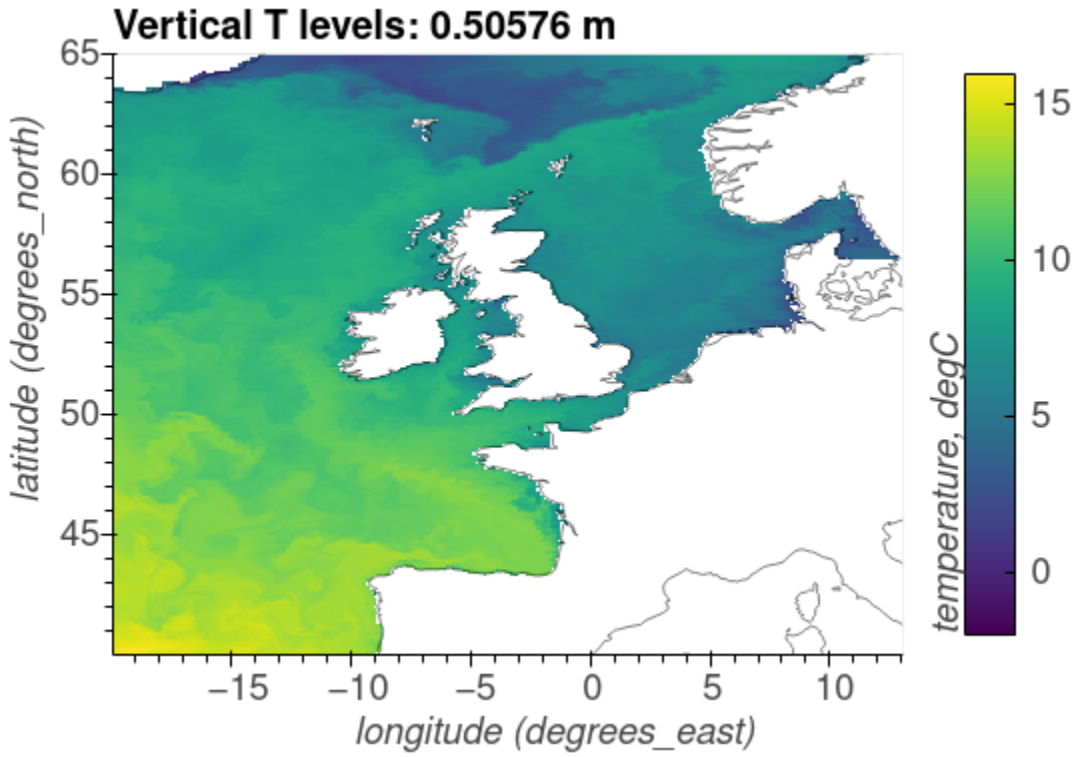
Listing 2: Example plotting script using *nctoolkit*, user is required to change *[PATH_TO_NETCDF_FILE]* and *[VARIABLE]* to the exact location of the NEMO output and the variable to be plotted, respectively.

```
import nctoolkit as nc

# path to netCDF file
ff = [PATH_TO_NETCDF_FILE]
ds = nc.open_data(ff)

# removes colours from land values
ds.set_missing(0)

# draws the outline of the land
ds.fix_nemo_ersem_grid()

# selects the final timestep for plotting
ds.select(time=[len(ds.times)-1])

# plots the variable. Note, the colour bar will change for each plot since autoscale
# is set to False
plot = ds.plot([VARIABLE], autoscale=False)
```

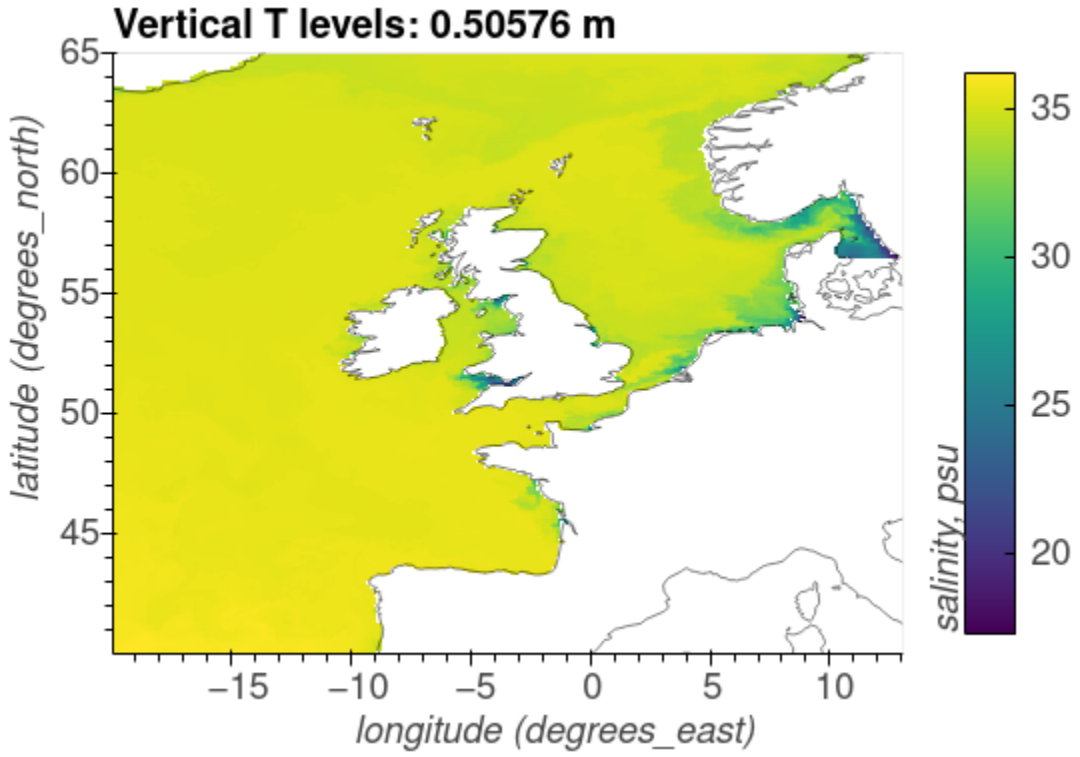
The following plots show the surface distributions of a subset of variables from the NEMO-ERSEM simulation on the AMM7 domain.

Note: The plots below are snapshots take from the model output at *30/01/2005*.

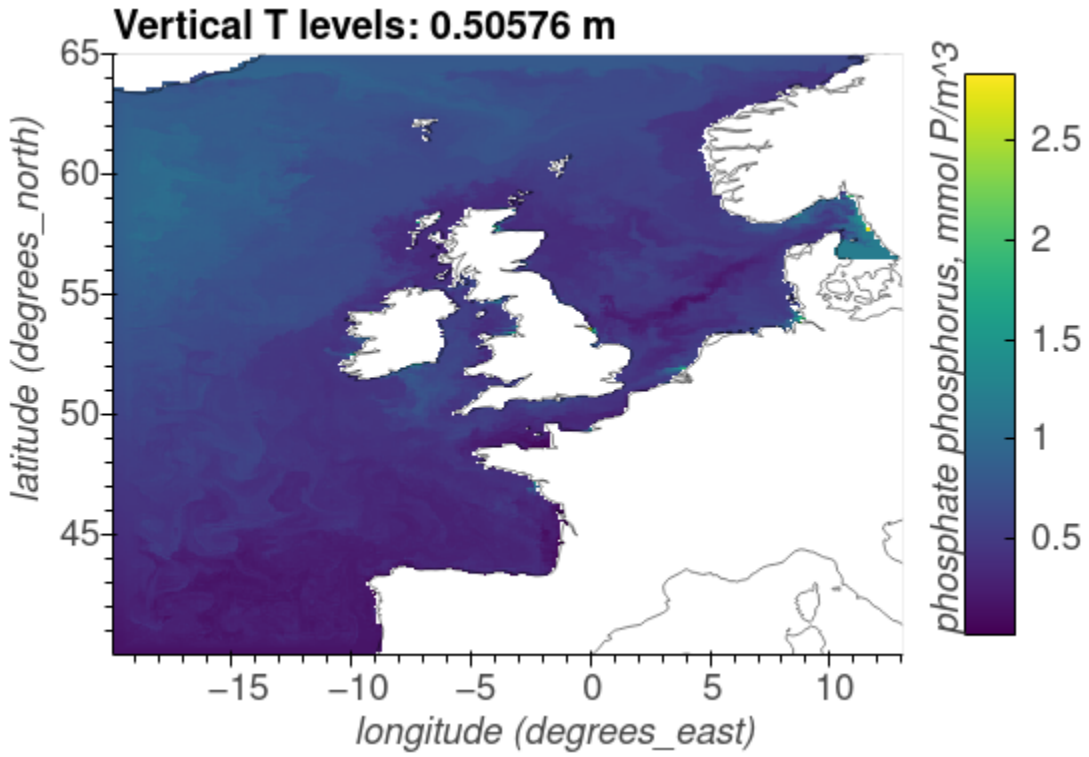
Potential temperature, `degC`



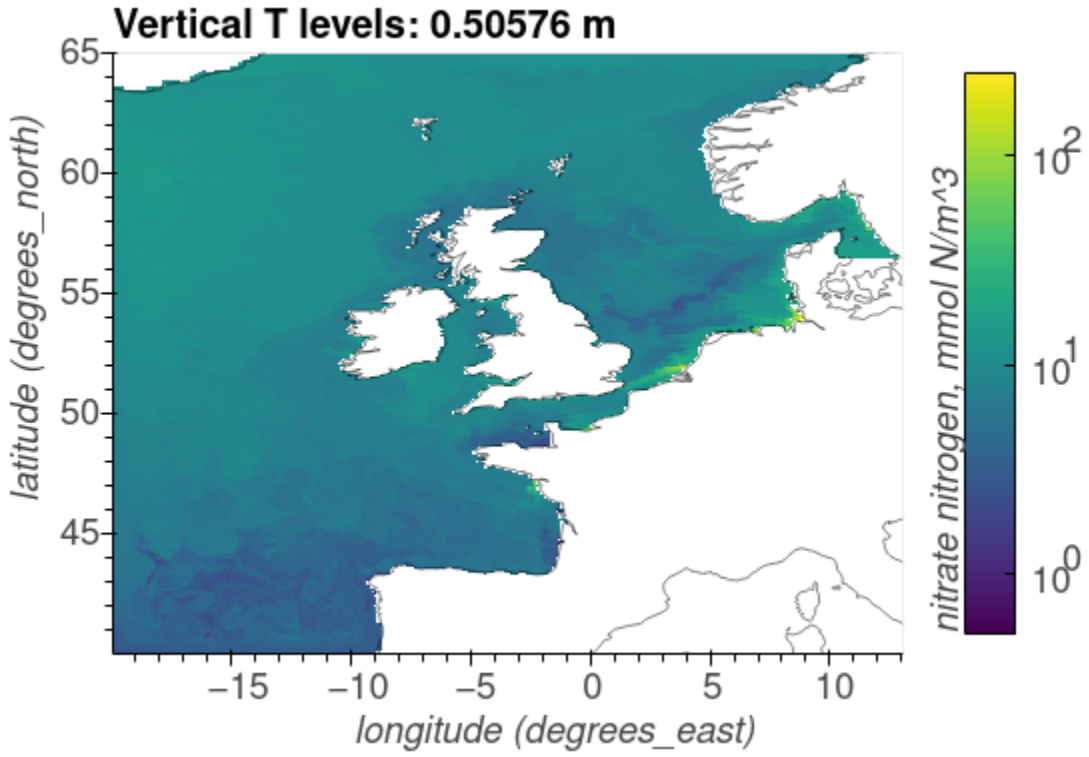
Salinity, `psu`



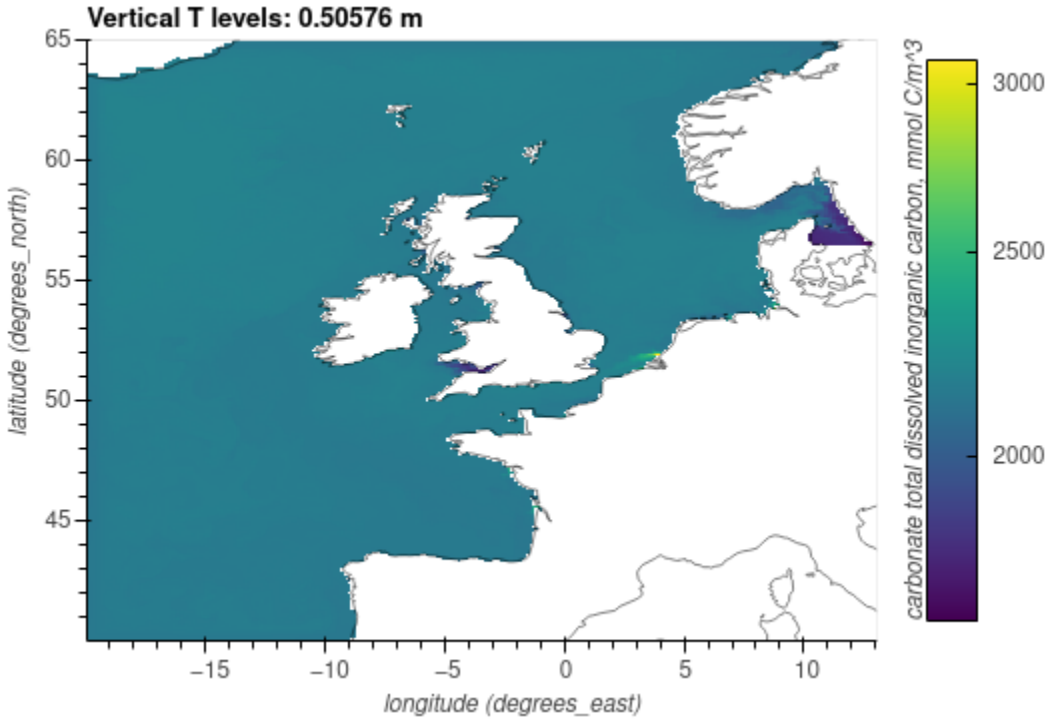
Phosphate phosphorus, `mmol P/m^3`



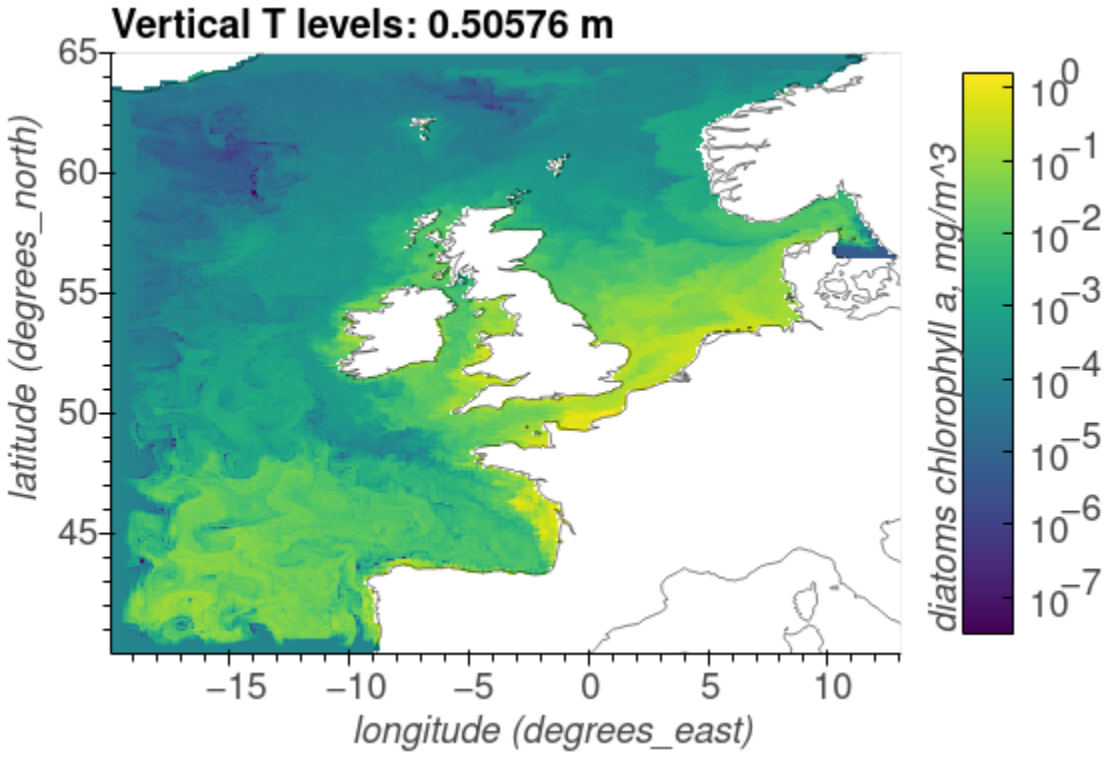
Nitrate nitrogen, `mmol N/m^3`



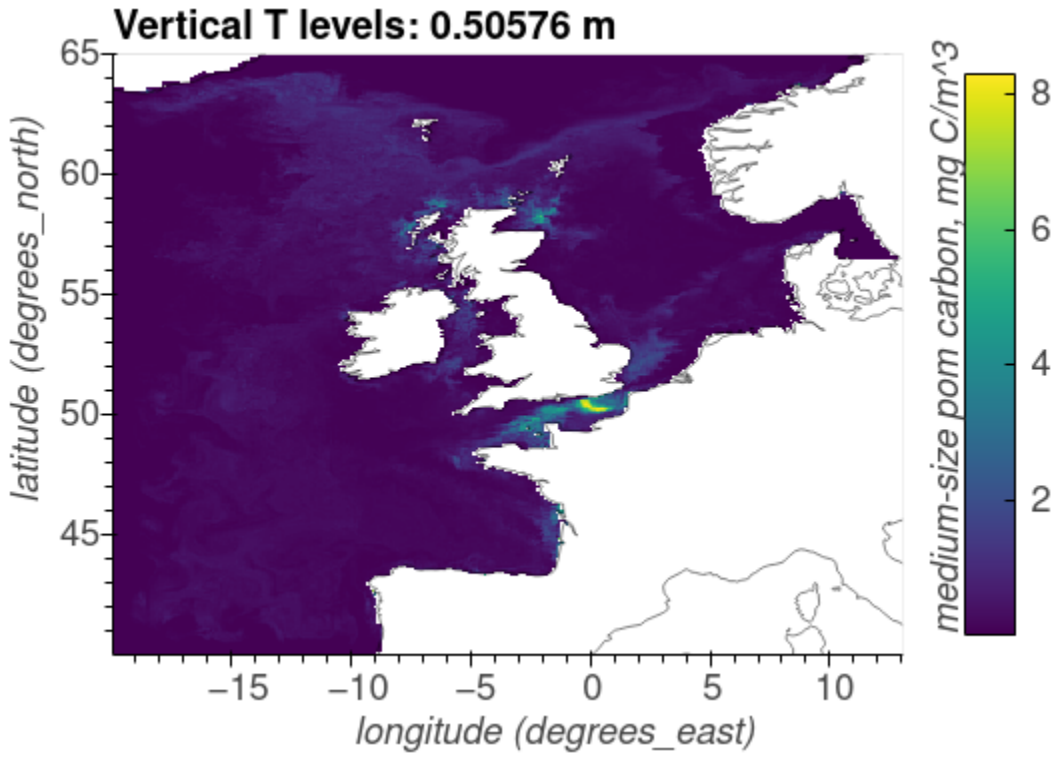
Carbonate total dissolved inorganic carbon, `mmol C/m^3`



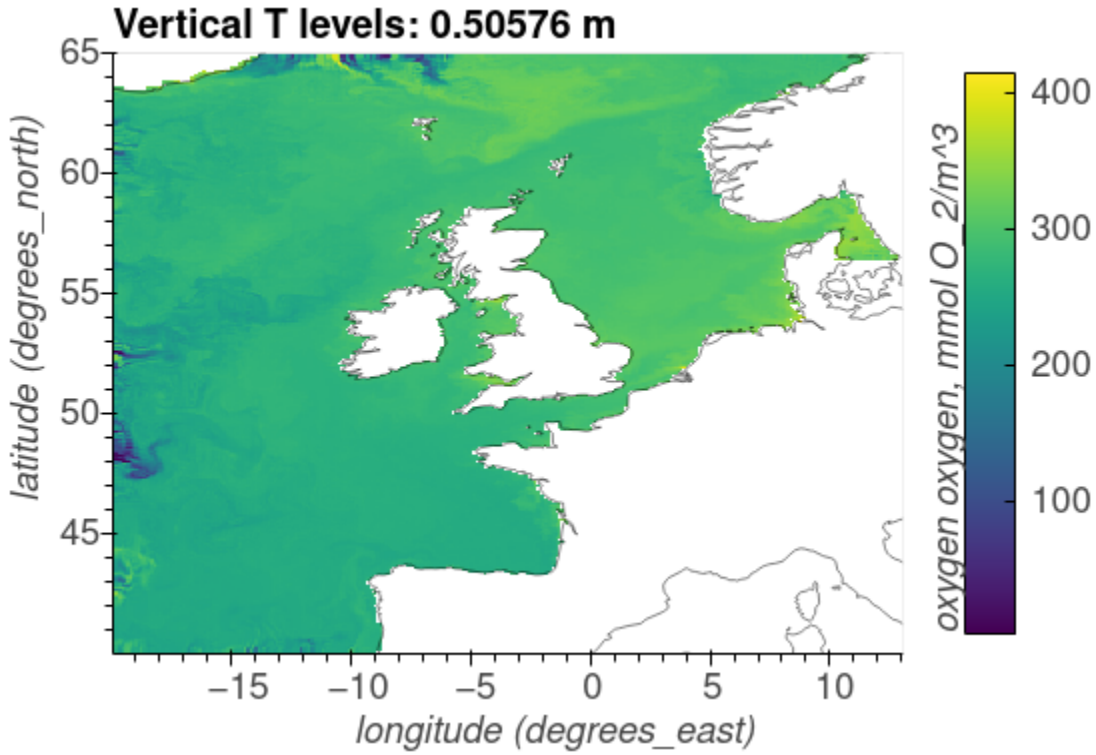
Diatoms chlorophyll, `mg/m^3`



Medium-sized POM carbon, `mg C/m^3`


Vertical T levels: 0.50576 m
latitude (degrees_north)
65
60
55
50
45
−15
−10
−5
0
5
10
longitude (degrees_east)
medium-size pom carbon, mg C/m^3
8
6
4
2

Oxygen, `O_2/m^3`


Vertical T levels: 0.50576 m
latitude (degrees_north)
65
60
55
50
45
−15
−10
−5
0
5
10
longitude (degrees_east)
oxygen oxygen, mmol O_2/m^3
400
300
200
100

2.2.6 NEMO-ERSEM in a water column

This tutorial demonstrates how to configure and run NEMO 1D (water-column) reference configuration C1D_PAPA with ERSEM biogeochemistry. This example is a demonstration of the concept, as it combines hydrodynamics of ocean station PAPA in the North Pacific Ocean with biogeochemistry of L4 station in the Western English Channel. Users are encouraged to modify this example to fit their own purposes.

We recommend to read the brief description of the NEMO C1D_PAPA configuration before continuing with this tutorial.

Step 1: Obtaining the code

The input data contained in `INPUTS_C1D_PAPA_v4.0.tar` file must be obtained from NEMO Reference configurations inputs repository on Zenodo and unpacked into the working directory.

NEMO4 code base with FABM support can be obtained in the corresponding repository. The next step is to download FABM and ERSEM. Finally, I/O server XIOS-2.5 must be downloaded and installed. The compiled *xios_server.exe* executable should be copied into the working directory.

Step 2: Compiling the code

First, FABM must be compiled with ERSEM support, specifying nemo as a physical host model. The following commands can be run in a command line or wrapped into an executable file for easy recompilation at any future point:

```
old=$(pwd)                          #remember current (working) directory
mkdir -p ~/build/nemo-fabm-ersem # create directory for the build
cd ~/build/nemo-fabm-ersem       # go to the build directory
cmake <FABM_DIR> -DFABM_HOST=nemo -DFABM_ERSEM_BASE=<ERSEM_DIR> -DCMAKE_
↪INSTALL_PREFIX=~/local/fabm/nemo-fabm-ersem
#replace <FABM_DIR> and <ERSEM_DIR> with the corresponding directories the␣
↪FABM and ERSEM code bases were downloaded to.
make -j4
make install
cd $old                             # return to the working directory
```

Thereafter, it is time to compile the NEMO executable. Users may refer to the *C1D_PAPA_FABM_ERSEM* configuration provided with the NEMO4.0-FABM in *cfgs* directory. The critical point is to specify all the necessary compilation keys in `cpp_X.fcm` file, i.e. key_c1d for compilation in 1D, and key_fabm for FABM support:

```
bld::tool::fppkeys   key_c1d key_mpp_mpi key_iomput key_nosignedzero key_top␣
↪key_fabm
```

Next, compile the model by executing the following lines:

```
#!/bin/bash

NEMO_BUILD_DIR=$<NEMO_DIR>
RUN_DIR=$<RUN_DIR>
export XIOS_HOME=$<XIOS_DIR>
export FABM_HOME=$HOME/local/fabm/nemo-fabm-ersem
# replace <NEMO_DIR> and <XIOS_DIR> with the path to the corresponding code␣
↪bases and <RUNDIR> with the working directory. FABM_HOME in this example␣
↪corresponds to the directory where FABM-ERSEM was installed.
```

(continues on next page)

(continued from previous page)

```
ARCH=GCC_PMPC # specify build architecture

cd $NEMO_BUILD_DIR
./makenemo -m $ARCH -r C1D_PAPA_FABM_ERSEM -n C1D_PAPA_FABM_ERSEM_BLD_SCRATCH␣
↪| tee compile.log
mv $NEMO_BUILD_DIR/cfgs/C1D_PAPA_FABM_ERSEM_BLD_SCRATCH/BLD/bin/nemo.exe $RUN_
↪DIR/
echo "Done."
```

The script above will compile the model and move the nemo executable into the working directory.

Note that the architecture file might require some editing depending on the machine the model is compiled and run on. This will include compiler version, compiler flags and links to netCDF libraries. Here, we are pointing our compilation to arch-GCC_PMPC.fcm file (available within NEMO4.0-FABM repository) to compile on a typical PML workstation running Fedora Linux distribution and using a GNU Fortran compiler.

Step 3: Getting ready to run the model

Within the working directory, create a link to the desired model configuration file. For this tutorial, we will link fabm.yaml provided with the cfgs/C1D_PAPA_FABM_ERSEM configuration:

```
ln -sf <NEMO_DIR>/cgfs/C1D_PAPA_FABM_ERSEM/fabm.yaml.ersem_c1d fabm.yaml
# replace <NEMO_DIR> with the directory containing the ERSEM code.
```

ERSEM requires some external inputs, which must be provided. The following lines have been appended to the fabm.yaml file we are using (for simplicity we are using constant parameter values here). Depending on the configuration, the list of required external inputs will vary.

```
pco2a:
  model: horizontal_constant
  parameters:
    value: 400.
    standard_name: mole_fraction_of_carbon_dioxide_in_air
ADY_0:
  model: horizontal_constant
  parameters:
    value: 1.0e-10
    standard_name: gelbstoff_absorption_satellite
```

Create links to, or copy namelist files from NEMO cfgs/C1D_PAPA_FABM_ERSEM folder into the working directory. Repeat the same procedure for *.xml files. file_def_nemo.xml defines which outputs will be saved, and at what frequency. For the purpose of this example, we will save a range of daily averaged pelagic and benthic state and diagnostic variables. This file can be used as a template to specify the desired range of model outputs.

Step 4: Running the model and visualising the outputs

The model is deployed by running the executable file in the working directory:

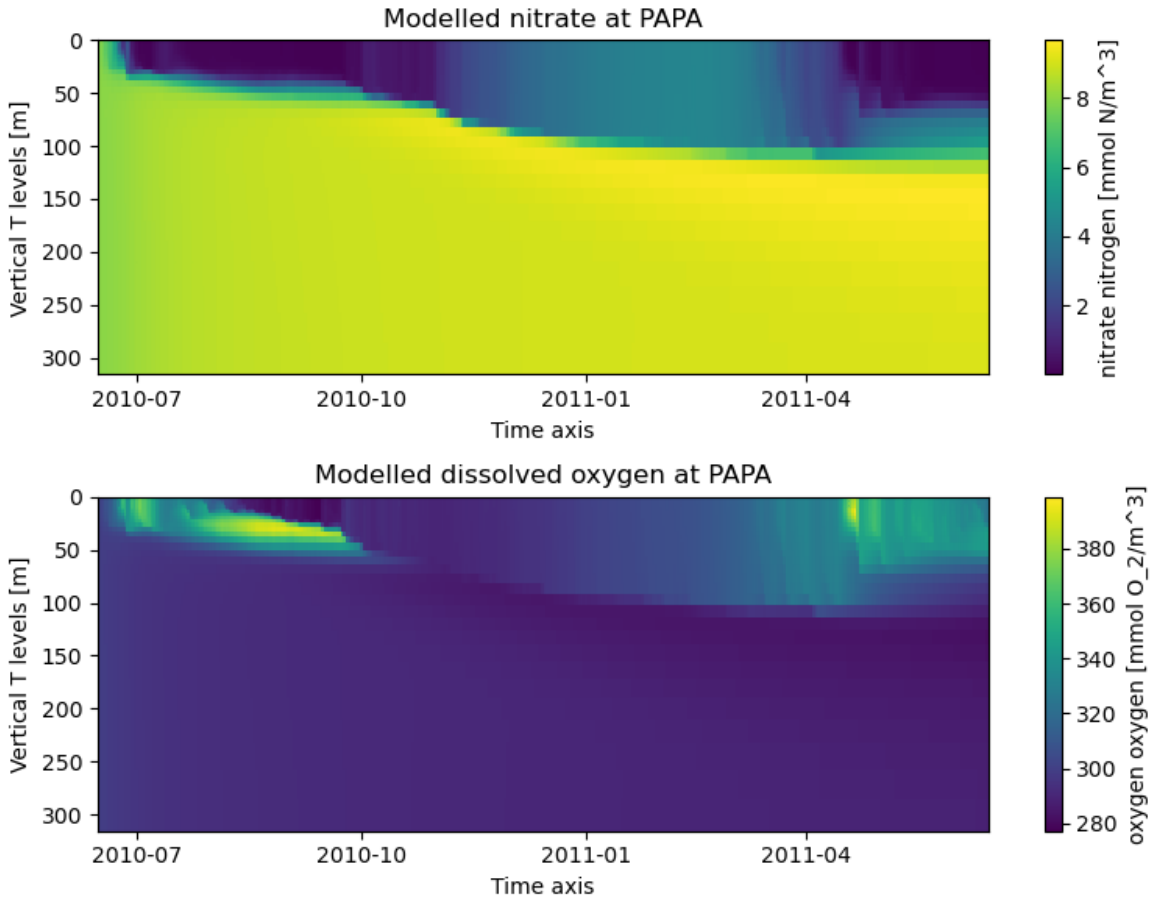
```
./nemo.exe
```

As a result, several output files will be generated according to specifications in file_def_nemo.xml file. Additionally, NEMO will generate restart files. The outputs can be visualised using netCDF viewer (e.g. ncview or PyNcView), or in Python using xarray:

```
import xarray as xr                                                    # import␣
↪xarray
import matplotlib.pyplot as plt                                        # import␣
↪matplotlib
dat = xr.open_dataset('C1D_PAPA_1d_20100615_20110614_ptrc_T.nc')        # open␣
↪dataset with ERSEM variables
fig,axes=plt.subplots(nrows=2,figsize=(8,6))                           # create␣
↪figure with 2 subplots
dat.N3_n[:,0:35,1,1].plot(x='time_counter',yincrease=False,ax=axes[0])  # plot␣
↪Hovmöller diagram for nitrate
dat.O2_o[:,0:35,1,1].plot(x='time_counter',yincrease=False,ax=axes[1])  # plot␣
↪Hovmöller diagram for oxygen
axes[0].set_title('Modelled nitrate at PAPA')                          # add title
axes[1].set_title('Modelled dissolved oxygen at PAPA')                 # add␣
↪another title
fig.tight_layout()                                                     # make it␣
↪look nice
```

In the example above we plot the variables over the entire modelled period, but only in the upper 35 vertical layers (from surface down to ~300 m depth). We also have to specify spatial dimensions of the output, as 1D configuration in NEMO actually has horizontal resolution of 3x3 grid points.

The resulting figure should look like this:



2.2.7 conda installation

We recommend using `conda` to obtain the latest version of `ersem`. The conda package includes interfaces for *Python front-end (pyfabm)*, *fabm0d: ERSEM in an aquarium* and *GOTM: ERSEM in a water column* tutorials, and can be installed as follows:

```
conda create -n ersem-tut -y
conda activate ersem-tut
conda install -c pmlmodelling ersem -y
```

To install `ersem` from source please look at the *developers documentation*.

2.3 Tips and tricks/troubleshooting

2.3.1 General:

- cmake autodetects a suitable Fortran compiler. If a Fortran compiler is not found or you are unhappy with the compiler cmake selected, you can override this by providing cmake with `-DCMAKE_Fortran_COMPILER=<COMPILER_EXECUTABLE>`. Note: if you change `CMAKE_Fortran_COMPILER` (or provide it for the first time, while you previously ran cmake without), you need to empty your build directory first!

- When building GOTM or FABM's 0d driver, cmake will try to auto-detect NetCDF using `nf-config`. If nf-config is not present on your system, you'll need to provide cmake with the path(s) to the NetCDF include directories (`-DNetCDF_INCLUDE_DIRS=<PATH>`) and the path(s) to the NetCDF libraries (`-DNetCDF_LIBRARIES=<PATH>`). If you need to provide multiple paths to these variables, the individual paths should be separated by semi-colons. A common reason to use this: On some systems, `nf-config` does not detect where `netcdf.mod` is installed, which means you have to tell cmake by adding `-DNetCDF_INCLUDE_DIRS=<netcdf.mod_DIR>`. For instance, when using gfortran on Fedora, `<netcdf.mod_DIR>` can be `/usr/lib64/gfortran/modules`. In that case you usually do not need to provide `-DNetCDF_LIBRARIES=<PATH>`. Also, on some systems (like in the current LTS release of Ubuntu), `nf-config` is not included in the NetCDF packages. In this case, you can use `nc-config`: specify `-DNetCDF_CONFIG_EXECUTABLE=<PATH_TO_nc-config>` when calling cmake, or create a link to `nc-config` somewhere in your default path, to get auto-detection working.
- By default, cmake will select the "release" build type, which creates an executable without debugging information. If you want to compile in debug mode, specify `-DCMAKE_BUILD_TYPE=debug` in the call to cmake. When using gfortran, you may also want to add `-DCMAKE_Fortran_FLAGS_DEBUG=-fcheck=bounds` to catch out-of-bounds array access.
- To see the settings you specified when you ran cmake, and to selectively make changes to these setttings, you can run `ccmake .` in your build directory.
- To speed up compilation, you can perform a parallel build by providing the `-j N` switch to `make` (not `cmake`!), with `N` being the number of cores you want to use.

Specific platforms:

- ARCHER: make sure to first load the cmake and cray-netcdf modules (`module load cmake cray-netcdf`). These enable cmake and autodetection of NetCDF paths, respectively. Provide `-DCMAKE_Fortran_COMPILER=ftn` to cmake to make sure cmake uses Cray's compiler wrapper script for Fortran compilation. Compilation should succeed with all three programming environments (intel, gnu, cray).
- Ubuntu LTS: `nf-config` is not included in the netcdf packages, but `nc-config` is. Specify `-DNetCDF_CONFIG_EXECUTABLE=<PATH_TO_nc-config>` when calling cmake, or create a link from `nf-config` to `nc-config` somewhere in your default path to get auto-detection of NetCDF paths working.
- Fedora: `nf-config` does not detect where `netcdf.mod` is installed (typically in `/usr/lib64/gfortran/modules`). This means you have to tell cmake by adding `-DNetCDF_INCLUDE_DIRS=/usr/lib64/gfortran/modules`.

2.3.2 Migrating from an earlier ERSEM version

If you were using an earlier release of ERSEM, you can update your old ERSEM configuration (`fabm.yaml`) to the latest by running the Python script `<SOURCEDIR>/ersem/testcases/update.py` with one argument: the path to your old fabm.yaml file.

This script requires a recent version of Python 2 or 3 and the PyYAML package. Ideally, you also have the latest version of *the Python front end to FABM-ERSEM* installed; this enables the update script to clean up the yaml file and add documentation to it.

2.4 Developers: installation from source

2.4.1 pyfabm-ersem

To build the `python` driver and use it with `ERSEM`, you must first obtain copies of the `FABM` and `ERSEM` codes. The latest version of each can be obtained by cloning their respective code repositories using git. The instructions of how to do this are found *here*.

To run the install script you will need to have `wheel` and `numpy` installed. This can be done via:

```
#!/bin/bash

echo "Installing numpy and wheel"
python -m pip install wheel numpy
```

To install PyFABM-ERSEM, we suggest you use the following script below

```
#!/bin/bash
BRANCH=${1:-master}

echo "Cloning ERSEM"
git clone https://github.com/pmlmodelling/ersem.git

echo "Cloning FABM"
git clone https://github.com/fabm-model/fabm.git

echo "Checking out branch: $BRANCH"
cd ersem && git checkout $BRANCH && cd ..

echo "Building PyFABM-ERSEM"
mkdir build && cd build
cmake ../fabm/src/drivers/python -DFABM_ERSEM_BASE=../ersem
make install
```

2.4.2 fabm0d-ersem

To run the install script, you will need to have `netCDF` installed. An example of how to do this is here:

```
#!/bin/bash

echo "Installing netCDF"
sudo apt update
sudo apt install libnetcdff-dev
```

To install FABM0d-GOTM-ERSEM we suggest you use the following script below

```
#!/bin/bash

BRANCH="master"
CMAKE_FLAG=""
while getopts ":b:f:" flag; do
    case "${flag}" in
```

(continues on next page)

(continued from previous page)

```
        b) BRANCH=${OPTARG};;
        f) CMAKE_FLAG=${OPTARG};;
    esac
done

CPU="$(nproc)"

echo "Cloning ERSEM"
git clone https://github.com/pmlmodelling/ersem.git

echo "Cloning FABM"
git clone https://github.com/fabm-model/fabm.git

echo "Cloning GOTM"
git clone https://github.com/gotm-model/code.git gotm
cd gotm && git checkout v6.0 && git submodule update --init --recursive && cd ..

echo "Checking out branch: $BRANCH"
cd ersem && git checkout $BRANCH && cd ..

echo "Building FABM-GOTM-ERSEM"
mkdir build && cd build
cmake ../fabm/src/drivers/0d -DGOTM_BASE=../gotm -DFABM_ERSEM_BASE=../ersem $CMAKE_FLAG
make install -j $CPU
```

2.4.3 gotm-ersem

To run the install script, you will need to have `netCDF` installed. An example of how to do this is here:

```
#!/bin/bash

echo "Installing netCDF"
sudo apt update
sudo apt install libnetcdff-dev
```

To install GOTM-FABM-ERSEM, we suggest you use the following script below

```
#!/bin/bash
BRANCH="master"
CMAKE_FLAG=""
while getopts ":b:f:" flag; do
    case "${flag}" in
        b) BRANCH=${OPTARG};;
        f) CMAKE_FLAG=${OPTARG};;
    esac
done

CPU="$(nproc)"

echo "Cloning ERSEM"
git clone https://github.com/pmlmodelling/ersem.git
```

(continues on next page)

(continued from previous page)

```

echo "Cloning FABM"
git clone https://github.com/fabm-model/fabm.git

echo "Cloning GOTM"
git clone https://github.com/gotm-model/code.git gotm
cd gotm && git checkout v6.0 && git submodule update --init --recursive && cd ..

echo "Checking out branch: $BRANCH"
cd ersem && git checkout $BRANCH && cd ..

echo "Building GOTM-FABM-ERSEM"
mkdir build && cd build
cmake ../gotm -DFABM_BASE=../fabm -DFABM_ERSEM_BASE=../ersem $CMAKE_FLAG
make install -j $CPU
```

If you experience NetCDF issues when running `make install`, see *Tips and tricks/troubleshooting*.

Now you should have a GOTM executable with FABM and ERSEM support at `~/local/gotm/bin/gotm`.

2.4.4 Obtaining source code

To get the ERSEM and FABM source codes:

```
git clone https://github.com/pmlmodelling/ersem.git
git clone https://github.com/fabm-model/fabm.git
```

This locally creates `ersem` and `fabm` directories with source code.

For water column models, you will need the GOTM source code:

```
git clone --recurse-submodules https://github.com/gotm-model/code.git gotm
```

To build the code, you will need: * a recent Fortran compiler * cmake 3.0 or higher. First check whether you have that installed: run `cmake --version` on the command line.

Note: It is not necessary to use `pip` or `apt` to install dependences. We suggest that conda or brew would also work well.

2.5 ERSEM Workflow

As a rule of thumb, ERSEM developments should be made in the public ERSEM with the exception of specific project developments.

2.5.1 Issue tracking

All developments, either private or public, are required to create an issue/issues describing the problem or development required.

For larger project developments that require multiple issues, it is often preferable to create a milestone which you can then assign multiple issues too.

2.5.2 Private developments

Private developments are those generated within new and ongoing projects that requires significant scientific changes to the code that will be initially isolated from the public version of the code

Workflow for private developments

The workflow for private developments is as follows:

1. Create an issue or issues in the public ERSEM repo based on the development required.
2. Create a branch from the private ERSEM repo. The naming convention of the branch is as follows *XX-[description]* where *XX* is the issue ID (this can be found after creation of the issue) and *[description]* is a one- or two- word description to identify the branch with the corresponding issue.
3. Commit all changes outlined in the issue to this branch and push branch to the private ERSEM repo.
4. Create pull request to merge the branch you have made the changes on into the *master* branch. In the creation of the pull request you will need to assign specific people to review the changes you have made before it can be merged into the master branch.
5. Once the updates have been approved by the reviewer/reviewers they will merge the development into the master branch.
6. When the private master branch is updated you will need to talk to the ERSEM repo manager to merge those changes back to the public repo.

Note: When creating the pull request it is important to add *Fixes #[issue-id-number]*, this will insure that when the pull request is merged into master the corresponding issue will be closed at the same time.

2.5.3 Public developments

The following changes are considered public developments and would need to follow a slightly different workflow (see below):

1. Documentation - new descriptions and updates to current documentation, these include new ERSEM tutorials or fixes to current docs
2. Bug fixes - these require immediate attention when identified within the code base
3. Public project developments - if possible, small developments and additions to the code shall be added publicly, these could include different options or parameterisations of the model.

Workflow for public developments

For public developments, the workflow is pretty simple:

1. Create an issue or issues in the public ERSEM repo based on the development required.
2. Create a branch from the public ERSEM repo. The naming convention of the branch is as follows *XX-[description]* where *XX* is the issue ID (this can be found after creation of the issue) and *[description]* is a one- or two- word description to identify the branch with the corresponding issue.
3. Commit all changes outlined in the issue to this branch and push branch to the ERSEM repo.
4. Create pull request to merge the branch you have made the changes on into the *master* branch. In the creation of the pull request you will need to assign specific people to review the changes you have made before it can be merged into the master branch.
5. Once the updates have been approved by the reviewer/reviewers they will merge the development into the master branch.
6. When the public master branch is updated you will need to talk to the ERSEM repo manager to merge those changes back to the private repo.

Note: When creating the pull request it is important to add *Fixes #[issue-id-number]*, this will insure that when the pull request is merged into master the corresponding issue will be closed at the same time.

2.5.4 Notes for reviewers

When reviewing pull requests there are a few things to check:

1. Ensure that the new changes do not adversely affect ERSEM. **Note** some modifications will change the results produced by the model, however, it is the author of the changes and reviewer's responsibility to decide whether these changes go into ERSEM. If unsure, please contact the ERSEM repository manager.
2. Ensure that the changes made to the code correctly address the issue created and that *Fixes #[issue-id-number]* is added to the pull request description to ensure the relevant issue is closed upon merging with master.
3. For the public ERSEM, make sure that the tests continue to pass.

Note: Any questions about the workflow, please contact the ERSEM repository manager.

2.6 ERSEM module index

2.6.1 Module name: ersem_primary_producer

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description: Silicate use (P1) is optional - it is activated by setting the "use_Si" flag in the run-time configuration.

Calcification (P2) is optional - it is activated by setting the "calcify" flag in the run-time configuration. Preprocessor symbol CALC is no longer used.

The model can be switched from a constant ratio between labile and semi- labile dissolved organic matter production to a dynamic ratio by setting the "docdyn" flag in the run-time configuration. Preprocessor symbol DOCDYN is no longer used.

Iron use is controlled by use_iron defined in ersem/shared.F90.

Functions:

```
function get_sinking_rate(self,e__arguments_local_e) result (sd)
    real(kind=rk) :: sd
```

2.6.2 Module name: ersem_benthic_bacteria

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_benthic_base
```

Description:

2.6.3 Module name: ersem_carbonate

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description:

Functions:

```
function approximate_alkalinity(iswtalk,t,s) result (ta)
    integer, intent(in) :: iswtalk
    real(kind=rk), intent(in) :: t
    real(kind=rk), intent(in) :: s
    real(kind=rk) :: ta
```

2.6.4 Module name: ersem_light

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description:

2.6.5 Module name: ersem_mesozooplankton

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description:

2.6.6 Module name: ersem_benthic_carbonate

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description:

2.6.7 Module name: ersem_benthic_erosion

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description: type_ersem_benthic_erosion

This model estimates sediment erosion in m/d from the bottom shear stress.

2.6.8 Module name: ersem_bacteria_docdyn

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description:

2.6.9 Module name: ersem_benthic_calcite

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_benthic_base
```

Description: Benthic variable that supports resuspension and remineralization. Both processes return material to the pelagic.

default: all is private.

2.6.10 Module name: ersem_benthic_column

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description: type_ersem_benthic_column

This model specifies the structure of the three-layer sediment column.

This model also computes the diffusivity of solutes in the different layers, and a "particulate diffusivity" that represents bioturbation. These variables account for variable bioturbation and bioirrigation activity, respectively.

2.6.11 Module name: ersem_light_iop

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description:

2.6.12 Module name: ersem_calcification

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description:

2.6.13 Module name: ersem_microzooplankton

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description:

2.6.14 Module name: ersem_benthic_nitrogen_cycle

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description: Model for nitrogen cycle

2.6.15 Module name: ersem_benthic_base

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_particle_model
```

Description: Benthic variable that supports resuspension and remineralization. Both processes return material to the pelagic.

default: all is private.

2.6.16 Module name: ersem_pelagic_base

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_particle_model
```

Description: default: all is private.

Functions:

```
function get_sinking_rate(self,e__arguments_local_e) result (rm)
    real(kind=rk) :: rm
```

2.6.17 Module name: ersem_light_iop_ady

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description:

2.6.18 Module name: ersem_benthic_column_particulate_matter

Include:

```
"fabm_driver.h"
 surface flux $-D C0 z/z_mean d/dz exp(-z/z_mean) = 0$, as well as non-
↪zero bottom flux
```

Class inheritance:

```
type_ersem_benthic_base
```

Description: Particulate organic matter with idealized [exponential] profile.

The concentration of matter, in mass per unit sediment volume, is assumed to be an exponential function of depth z:

C(z) = C0*exp(-b*z)

What value do constants $C0$ and b take?

Let's define the mean depth of matter within entire sediment column (depth range 0 to $infty$):

z_mean = int_0^infty z C(z) dz / int_0^infty C(z) dz

When we insert the exponential distribution of $C(z)$, $C0$ drops out and we obtain

z_mean = int_0^infty z exp(-b*z) dz / int_0^infty exp(-b*z) dz

For the denominator of z_mean (concentration integrated from surface to infinite depth) we find through standard integration:

int_0^infty exp(-b*z) dz = [-1/b exp(-b*z)]_0^infty = 1/b

Numerator $int_0^infty z exp(-b*z) dz$ can be found through integration by parts: General rule: $int u v dz = u int v dz - int (u' int v dz) dz$ We can apply this to find the antiderivative of $z exp(-b*z)$:

int z exp(-b*z) dz = -z/b exp(-b*z) - 1/b^2 exp(-b*z) = -(z+1/b)/b exp(-b*z)

Verify by differentiation (apply chain rule):

-1/b exp(-b*z) + z exp(-b*z) + 1/b exp(-b*z) = z exp(-b*z) [OK]

Integrating from 0 to $intfy$ while assuming $b>0$ we obtain

int_0^infty z exp(-b*z) dz = [-(z+1/b)/b exp(-b*z)]_0^infty = 1/b^2

Combining the expressions for the numerator and denominator of z_mean:

z_mean = (1/b^2)/(1/b) = 1/b

Thus, the exponential decay constant b is equal to $1/z_mean$, and

C(z) = C0*exp(-z/z_mean)

NB this is a standard result (mean of the exponential distribution); the above derivation is given simply for completeness.

The integral of C(z) from 0 to the bottom of the modelled column, z_bot, should equal the modelled density of mass:

int_0^z_bot C(z) dz = [-z_mean*C0*exp(-z/z_mean)]_0^z_bot = z_mean*C0*(1-exp(-z_bot/z_mean)) = C_int

Thus, surface concentration $C0$ can be rewritten in terms of the depth-integrated concentration C_int, (integrated up to z_bot, not $infty$

C0 = C_int/z_mean/(1-exp(-z_bot/z_mean))

Note: Ebenhoeh et al. 1995 define the penetration depth as mean depth within the model domain, i.e., with lower boundary z_bot rather than $infty$:

z_mean' = int_0^z_bot z C(z) dz / int_0^z_bot C(z) dz = [-(z+1/b)/b exp(-b*z)]_0^z_bot / [-1/b exp(-b*z)]_0^z_bot = [1/b^2-(z_bot+1/b)/b exp(-b*z_bot)] / [1/b-1/b exp(-b*z_bot)] = 1/b [1-(z_bot/b+1) exp(-b*z_bot)] / [1-exp(-b*z_bot)] = 1/b [1-z_bot/b exp(-b*z_bot)/[1-exp(-b*z_bot)]

This will NOT produce an explicit expression for b as a function of z_mean'. Instead, Ebenhoeh et al. implicitly assume z_bot>>z_mean', which leads to $b approx 1/z_mean'$. However, this approximation becomes invalid when z_mean becomes large, and that can happen in the model. To avoid this problem, we define penetration depth as the mean depth of mass between 0 and infty. Since our exponential function remains the same, expressions based on that (e.g., impact of bioturbation) are the same as in the Oldenburg model. Expressions based on the interpretation of penetration depth (the mean depth of mass between 0 and infty in our case, and between 0 and z_bot for the Oldenburg model) will differ slightly, as e.g. in the next section.

Sources and sinks change $C(z)$, and therefore also penetration depth

z_mean = int_0^infty z C(z) dz / int_0^infty C(z) dz

The time derivative of this expression is found by applying the chain rule

d/dt z_mean = [int_0^infty z d/dt C(z) dz - int_0^infty d/dt C(z) dz int_0^infty z C(z) dz / int_0^infty C(z) dz] / int_0^infty C(z) dz

Introducing depth-integrated concentration

C_int_infty = int_0^infty C(z) dz,

depth-integrated sources minus sinks

sms = int_0^infty d/dt C(z) dz,

and the mean depth of the sources minus sinks

z_sms = int_0^infty z d/dt C(z) dz / int_0^infty d/dt C(z) dz,

we can simplify this to

d/dt z_mean = (z_sms - z_mean) sms/**C_int**_infty

It is worth noting that the final division is by the concentration integrated from surface to *infinite depth*, C_int_infty:

$C_int_infty = z_mean C0

That is not the same as C_int, i.e., the concentration integrated over the modelled depth interval (0 to z_bot), which equals

C_int = z_mean C0 (1-exp(-z_bot/z_mean))

In the original Oldenburg implementation, penetration depth is defined as the mean depth of mass between 0 and z_bot (not 0 to infty). As a result, C_int is substituted for C_int_infty. One consequence of this difference is that penetration depth in the Oldenburg model can "run away" (tend to infinity), while that is prevented in our expression due to the additional multiplication of the rate of change with 1-exp(-z_bot/z_mean).

It is worth noting that the resulting expression for d/dt z_mean does NOT depend on distribution $C(z)$. That is, it is valid for ANY vertical distribution, exponential or otherwise.

Modelled as a diffusion process, bioturbation changes $C(z)$, and therefore also penetration depth

z_mean = int_0^infty z C(z) dz / int_0^infty C(z) dz

Let us try this first for the denominator:

d/dt int_0^infty C(z) dz = int_0^infty d/dt C(z) dz

For d/dt C(z) we have the normal diffusion equation:

d/dt C(z) = d/dz(D d/dz C(z)) = d/dz(-D/z_mean C0 exp(-z/z_mean)) = D C0/z_mean^2 exp(-z/z_mean)

Inserting this expression we obtain for the change in depth-integrated mass:

d/dt int_0^infty C(z) dz = int_0^infty D C0/z_mean^2 exp(-z/z_mean) dz = [-D C0/z_mean exp(-z/z_mean)]_0^infty = D C0/z_mean

However, we KNOW that diffusion witin the column should not affect the mass integral. Why is this then non-zero? The reason for this is that we have not accounted for the no-flux boundary conditions. As a result, we are implicity using a non-zero inward flux at the surface that is determined by the gradient:

-D d/dz C(0) = D C0/z_mean exp(-z/z_mean) = D C0/z_mean

In other words, to close the column for mass, we need to subtract this surface flux. (NB the gradient is zero at infinite depth, i.e., the bottom flux is zero)

With this knowledge, we can revisit the numerator in the change in penetration depth z_mean. Its time derivative equals

d/dt int_0^infty z C(z) dz = int_0^infty z d/dt C(z) dz

From the diffusion equation for $C(z)$ we derived $d/dt C(z) = D C0/z_mean^2 exp(-z/z_mean)$. Inserting this in $d/dt z_mean$, while limiting bioturbation to maximum depth z_tur, we obtain

d/dt int_0^z_tur z C(z) dz = D C0/z_mean^2 int_0^z_tur z exp(-z/z_mean) dz

For the expression within the integral we previously found antiderivative $-(z+z_mean)z_mean exp(-z/z_mean)$. Thus,

int_0^z_tur z exp(-z/z_mean) dz = [-(z+z_mean)z_mean exp(-z/z_mean)]_0^z_tur = -(z_tur+z_mean)z_mean exp(-z_tur/z_mean) + z_mean^2

Using this antiderivative to solve the integral from 0 to z_tur:

d/dt int_0^infty z C(z) dz = D C0 [1-(z_tur/z_mean+1)] exp(-z_tur/z_mean)

$-D C0 z_tur/z_mean exp(-z_tur/z_mean)$. Subtracting the latter we obtain:

d/dt int_0^infty z C(z) dz = D C0 [1 - exp(-z_tur/z_mean)]

Finally, $d/dt z_mean$ is given by the ratio of this expression to $int_0^infty C(z) dz = C0 z_mean$:

d/dt z_mean = D/z_mean [1 - exp(-z_tur/z_mean)]

This describes how the penetration depth z_mean changes due to bioturbation up to depth z_tur, with the intensity of bioturbation described by diffusivity D. It is worth noting that $d/dt z_mean to infty$ when $z_mean to 0$. Thus, in the analytical solution of the model penetration depth cannot become negative as long as $D>0$.

A change in penetration depth without an associated change in total mass C_int_infty will cause a change in the mass in the depth interval described by the model, i.e., int_0^z_bot C0 exp(-z/z_mean). Inserting $C0 = **C_int_**infty/z_mean$:

int_0^z_bot C0 exp(-z/z_mean) = **C_int_**infty/z_mean int_0^z_bot exp(-z/z_mean)

Taking the time derivative, with only z_mean depending on time (**C_int_**infty is constant

d/dt int_0^z_bot C0 exp(-z/z_mean) = d/dt z_mean -C_int_infty/z_mean^2 int_0^z_bot exp(-z/z_mean)

- C_int_infty/z_mean int_0^z_bot z/z_mean^2 d/dt z_mean exp(-z/z_mean) dz

For the first term (first line), we have:

d/dt z_mean -C_int_infty/z_mean^2 int_0^z_bot exp(-z/z_mean) = - 1/z_mean d/dt z_mean C_int

with C_int being the concentration integrated from surface to model bottom [not infinite depth

We can rewrite the second term by moving constants outside the integral:

C_int_infty/z_mean^3 d/dt z_mean int_0^z_bot z exp(-z/z_mean) dz

Now we can solve the integral with our previously found antiderivative:

int_0^z_bot z exp(-z/z_mean) dz = [-(z+z_mean)z_mean exp(-z/z_mean)]_0^z_bot = -(z_bot+z_mean)z_mean exp(-z_bot/z_mean) + z_mean^2

Inserting this

C_int_infty/z_mean d/dt z_mean [-(z_bot/z_mean+1) exp(-z_bot/z_mean) + 1] = C_int_infty/z_mean d/dt z_mean [1 - exp(-z_bot/z_mean) - z_bot/z_mean exp(-z_bot/z_mean)]

Since $C_int = C_int_infty [1 - exp(-z_bot/z_mean)]$, we can rewrite this as

1/z_mean d/dt z_mean C_int - C_int_infty/z_mean d/dt z_mean z_bot/z_mean exp(-z_bot/z_mean)

Combining the two contributions to d/dt int_0^z_bot C0 exp(-z/z_mean), we are left with

d/dt int_0^z_bot C0 exp(-z/z_mean) = - C_int_infty z_bot/z_mean^2 exp(-z_bot/z_mean) d/dt z_mean

JB 28/11/2014: this solution was validated in Python with sympy, using the following code

C_int_infty,z_bot,z,t = sympy.symbols('C_int_infty,z_bot,z,t',positive=True,real=True) z_mean = sympy.Function('z_mean',positive=True,real=True)(t) C = C_int_infty/z_mean*sympy.exp(-z/z_mean) dC_dt = sympy.diff(C,t) print sympy.integrate(dC_dt,(z,0,z_bot),conds='none')

Inserting $C_int_infty = C_int/[1 - exp(-z_bot/z_mean)]$

d/dt int_0^z_bot C0 exp(-z/z_mean) = - C_int/[1 - exp(-z_bot/z_mean)] z_bot/z_mean^2 exp(-z_bot/z_mean) d/dt z_mean

Since $C0 = C_int/[1 - exp(-z_bot/z_mean)]/z_mean$, this is equivalent to

d/dt int_0^z_bot C0 exp(-z/z_mean) = C(z_bot) z_bot/z_mean d/dt z_mean

In the Oldenburg code, the final expression for the change in mass within model domain due to burial is

- C_int/[1-exp(-z_bot/z_mean)] [exp(-(z_bot-d/dt z_mean)/z_mean-exp(-z_bot/z_mean)]

Both our expression and the Oldenburg one contain -C_int/[1-exp(-z_bot/z_mean)] exp(-z_bot/z_mean). A discrepancy remains in the scale factor applied to it:

our derivation: z_bot/z_mean^2 d/dt z_mean Oldenburg: exp(d/dt z_mean/z_mean) - 1

It's worth noting that the Oldenburg model here has a unit problem: the factor in the exponential is not dimensionless, but has unit time^-1.

This file contains two modules that can be instantiated by the user (from fabm.yaml): type_ersem_benthic_column_particulate_matter: describes particulate organic matter in terms of column-integrated mass and penetration depth type_ersem_benthic_pom_layer: describes particulate organic matter within a user-specified depth interval

The idealized profile is defined only in terms of its density (quantity/m^2) and its penetration depth. By assuming an exponential distribution of matter (constant positive concentration at sediment surface, tending to zero at infinite depth), these two variables suffice to specify the concentration profile.

From the idealized concentration profile, we first compute densities (quantity/m^2) per layer. These layer-specific densities act as state variable: other modules can retrieve their value, but also provide their rate of change. We finally retrieve these rates of change, and convert them into a change in total depth-integrated matter *and* a change in penetration depth.

The "within a single depth interval" functionality is partitioned over two modules:

type_ersem_benthic_pom_layer computes the density of all constituents within the specified depth interval. This is the module that can be instantiated from fabm.yaml.

type_constituent_for_single_layer_change collects interval-specific sink-source terms for a single constituent and translates those into the change in column-integrated matter and penetration depth. It is created automatically as a submodel of type_ersem_benthic_pom_layer and does not interact with the user.

The split of functionality across two modules is needed because the mass within the interval must be known early, so all other modules can use it, while the change in column-integrated mass and penetration depth must be computed late, after all other individual modules have computed interval-specific rates of change. Thus we have the dependency chain:

1 mass within the desired depth interval (type_ersem_benthic_pom_layer)

-> 2 interval-specific sink-source terms (all other modules) -> 3 change in column-integrated mass and penetration depth (type_constituent_for_single_layer_change)

Putting 1 and 3 in the same module creates a circular dependency [that's BAD]. default: all is private.

Module for particulate organic matter class with idealized profile (e.g., Q6, Q7) Also handles the change in average penetration depth due to bioturbation, as well as burial in the form of organic matter moving beyond the bottom of the modelled column due to an increase in penetration depth. This module will always create a "surface" submodel for organic matter at the very surface of the sediment; this is typically used as target for sedimention/resuspension.

Functions:

```
function partq(d_pen,d_top,d_bot,d_max)
    real(kind=rk), intent(in) :: d_pen
    real(kind=rk), intent(in) :: d_top
    real(kind=rk), intent(in) :: d_bot
    real(kind=rk), intent(in) :: d_max
```

2.6.19 Module name: ersem_benthic_column_dissolved_matter

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description: depth-integrated mass in the benthic column depth-integrated [negative] mass below the zero-concentration isocline concentration in bottom-most pelagic cell sources-sinks specified for layer-integrated variables (pore water and adsorbed) sources-sinks specified for layer-integrated pore water concentration pelagic-benthic flux Flag specifying whether constituent is non-negative Model for dissolved matter in sediment, using idealized equilibrium profiles to determine pelagic-benthic diffusive flux, and to determine equilibrium depths of all but the last layer (NB layer depth will be relaxed to equilibrium value).

2.6.20 Module name: ersem_nitrification

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description:

2.6.21 Module name: ersem_oxygen

Include:

```
"fabm_driver.h"
 as part of utilization.
```

Class inheritance:

```
type_base_model
```

Description: Computes oxygen saturation, apparent oxygen utilization and handles exchange of oxygen across the water surface. Note: negative oxygen concentrations are permitted. These reflect an oygen debt (e.g., presence of H2S) In this case, oxygen saturation will be zero (not negative while apparent oxygen utilization will still be the difference between saturation concentration and [negative] oxygen concentration.

Functions:

```
function oxygen_saturation_concentration(self,etw,x1x) result (osat)
    real(kind=rk), intent(in) :: etw
    real(kind=rk), intent(in) :: x1x
    real(kind=rk) :: osat
```

2.6.22 Module name: ersem_zenith_angle

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_base_model
```

Description: This module calculates the solar zenith angle as a function of longitude, latitude, day of year, and time of day. The code that performs this calculation has been adapted from the General Ocean Turbulence Model (GOTM, http://www.gotm.net). Original GOTM source file: src/airsea/solar_zenith_angle.F90.

2.6.23 Module name: ersem_model_library

Include:

```

```

Class inheritance:

```
type_base_model_factory
```

Description:

2.6.24 Module name: ersem_nitrous_oxide

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description: This module calculates saturation concentrations and air-sea exchange of nitrous oxide. Production of nitrous oxide is implemented in nitrification module (nitrification.F90). For implementation and validation of nitrous oxide on the North-West European Shelf see Lessin et al. (2020): doi.org/10.1029/2019JG005613 ———————————————————————————

Functions:

```
function n2o_transfer_coefficient(self,etw,x1x) result (kon2o)
    real(kind=rk), intent(in) :: etw
    real(kind=rk), intent(in) :: x1x
    real(kind=rk) :: kon2o
```

2.6.25 Module name: ersem_bacteria

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_pelagic_base
```

Description:

2.6.26 Module name: ersem_benthic_fauna

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_benthic_base
```

Description: To achieve compatibility with legacy ERSEM, we need to be able to decouple the variable from which food availability is derived from the variable that absorbs the loss due to gross food uptake. The following variables absorb the loss due to food uptake - by default they are coupled to the same variable from which available food is derived.

2.6.27 Module name: ersem_fluff

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_benthic_base
```

Description:

2.6.28 Module name: ersem_fluff

Include:

```
"fabm_driver.h"
```

Class inheritance:

```
type_ersem_benthic_base
```

Description:

2.7 License

2.8 Acknowledgements

Over the years, ERSEM has been developed using funding from a wide variety of sources and projects. The current codebase was developed in the NERC/Defra Shelf Seas Biogeochemistry Grant: [NE/K001876/1] and Marine Ecosystems Research Programmes Grant: [NE/L003066/1]. Today, ERSEM maintenance and development continues to be funded through the NERC Climate Linked Atlantic Sector Science programme Grant [NE/R015953/1] and a combination of UK Research and Innovation (UKRI) and European Research Council (ERC) funded research projects.

2.9 Bibliography

BIBLIOGRAPHY

[1] J.C. Blackford and P.J. Radford. A structure and methodology for marine ecosystem modelling. *Netherlands Journal of Sea Research*, 33(3-4):247–260, 1995. doi:10.1016/0077-7579(95)90048-9.

[2] M. Butenschön, J. Clark, J. N. Aldridge, J. I. Allen, Y. Artioli, J. Blackford, J. Bruggeman, P. Cazenave, S. Ciavatta, S. Kay, G. Lessin, S. van Leeuwen, J. van der Molen, L. de Mora, L. Polimene, S. Sailley, N. Stephens, and R. Torres. Ersem 15.06: a generic model for marine biogeochemistry and the ecosystem dynamics of the lower trophic levels. *Geoscientific Model Development*, 9(4):1293–1339, 2016. doi:10.5194/gmd-9-1293-2016.

[3] R.F. Weiss. The solubility of nitrogen, oxygen and argon in water and seawater. *Deep Sea Research and Oceanographic Abstracts*, 17(4):721–735, 1970. doi:10.1016/0011-7471(70)90037-9.