Coupling BFM with FABM

E. Álvarez, A. Teruzzi, P. Lazzari, G. Occhipinti, G. Cossarini, J. Bruggeman

Release 1.0, April 2023
—— BFM Report series N. 6 ——

http://bfm-community.eu
bfm_st@cmcc.it

This document should be cited as:
Álvarez, E., Teruzzi, A., Lazzari, P., Occhipinti, G., Cossarini, G., Bruggeman, J. (2023). Coupling BFM with FABM. BFM Report series N. 6, Release 1.0, April 2023, Trieste, Italy, http://bfm-community.eu

Contents

1 Introduction

The open source, Fortran-based Framework for Aquatic Biogeochemical Models (FABM, Bruggeman and Bolding 2014) provides a coupling layer that enables flexible coupling of ecosystem processes with GOTM and other physical drivers. FABM enables the development of complex BGC models in the form of stand-alone, process-specific modules. These are combined at runtime through coupling links to form a customized BGC/ecosystem model. At each simulation time step, the BGC equations are applied to each layer and the rates of sink and source terms at the current time and space are calculated using local variables (e.g., light, temperature, and concentrations of state variables) provided by GOTM. The rates are passed through FABM to the hydrodynamic model or time integrator in the case of a 0D configuration, which performs numerical integration of BGC processes and transport of BGC substances (e.g., nutrients, dissolved and particulate organic matter) between layers. The updated states are then fed back to the BGC model via FABM.

The full documentation of FABM can be consulted at https://github.com/fabm-model/fabm/wiki.

The FABM version used in the present report is 1.0.4.

The development of BFM for FABM is based on a modular approach leveraging of the object oriented tools available in fortran 2003. In particular, different subroutines and FABM models are associated with the different functional plankton species (PFTs, e.g., phytoplankton, zooplankton) and chemical processes (e.g., light, denitrification) included in the ecosystem model. Since the models in FABM are classified by reference institutes, a new folder named OGS was created to contain the BFM code adapted for FABM. In FABM, each modular component is defined per se as a model. The models included in the framework are ogs/Phyto, ogs/PelBac, ogs/MicroZoo, ogs/MesoZoo, ogs/PelChem, ogs/CalciteDissolution, ogs/PelOxygen, ogs/PelagicCSYS. These modules map the one present in BFM Pel folder. The additional general routines ogs/bfm_pelagic_base and shared.F90. The relationships between the modular components (FABM models) are defined in the `fabm.yaml` file. Therefore, a special `fabm.yaml` file for BFM was configured to run the model with 4 phytoplankton groups, 1 heterotrophic bacteria, 2 microzooplankton groups, and 2 mesozooplankton groups. The `fabm.yaml` file is equivalent to the `layout` contains both the information of the layout and namelist contained in the presets of the standard BFM code.

The coupled FABM-BFM is developed starting from the original style and structure of the official BFM code. The version of BFM coupled with FABM is the one described in Álvarez et al. (2023); Cossarini et al. (2015); Lazzari et al. (2021, 2016, 2012). The coupling of FABM with BFM is mainly related to the inherent structure required by FABM models design. This structure applies to each component (phytoplankton, meso- and micro-zooplankton) and is composed of header, initialize, do, do_surface structure section. There are two extension header based on type_ogs_bfm_pelagic_base defined in ogs/bfm_pelagic_base.

For each model component reported in Tab.1.1 there are 6 main components to be specified and identified:

1. identification of state variables;

2. identification of state variables of other models required ad dependencies;

Models	Module name	Type extension
Phyto.F90	bfm_Phyto	type_ogs_bfm_primary_producer
PelBac.F90	bfm_PelBac	type_ogs_bfm_pelagic_bacteria
MicroZoo.F90	bfm_MicroZoo	type_ogs_bfm_microzoo
MesoZoo.F90	bfm_Mesozoo	type_ogs_bfm_mesozoo
PelChem.F90	bfm_PelChem	type_ogs_bfm_PelChem
CalciteDissolution.F90	bfm_CalciteDissolution	type_ogs_bfm_CalciteDissolution
PelOxygen.F90	bfm_PelOxygen	type_ogs_bfm_PelOxygen
PelagicCSYS.F90	bfm_PelagicCSYS	type_ogs_bfm_PelagicCSYS

Table 1.1: New BFM biogeochemical model components introduced in the FABM folder

3. environmental dependencies;
4. definition of biogeochemical equations acting on model state variables and dependencies;
5. identifiers of diagnostic variables;
6. parameters (described in subroutines initialize).

In the module "initialize procedure" all the parameters are included using the same formalism of BFM (i.e., parameters have prefix “p_”). We set up the register state variables by adding the specific constituents for each model (e.g.: for phytoplankton carbon ‘c’, nitrogen ‘n’, phosphorus ‘p’, chlorophyll ‘chl’ and silicon ‘s’). Moreover, we included a diagnostic variable for each biogeochemical process to be able to check if the FABM-BFM prototype was consistent with respect to the official BFM version. In the do procedure we replicated the structure of BFM (i.e., the original code opportunely adapted and comments are included) and we set a diagnostic for each process included.

Few changes with respect to the original BFM structure have been included:

1. nutrient stress sinking process in phytoplankton was added to the Phyto model through two specific functions get_sinking_rate and get_vertical_movement;
2. the terms of alkalinity sink/source have been directly included in Phyto, PelChem, PelBac, MicroZoo and MesoZoo;

2 Compiling FABM-BFM as a python module

Current prerequisite packages to install FABM-BFM in python are

- CMake, version 3.0 or later
- Python 2.7 or Python 3.4 or later
- pip (installed by default with Python 2 $\geq$ 2.7.9 or Python 3 $\geq$ 3.4)
- wheel (install with python -m pip install --user wheel)

The usage of conda is encouraged to install the software above.

On most Linux-like systems (UNIX/Linux/Mac OS X), a Git client is already installed. You can then obtain the FABM source code systems by executing

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

and then creating a folder named `extern`:

```
cd fabm
mkdir extern
```

to compile the FABM-BFM code it is further necessary to download the BFM code from the BFM website (www.bfm-community.eu and follow "Get the code" instructions).

After this operation it is necessary to copy the fabm folder contained in BFM "src" directory and put this in the extern folder in the FABM code. You can rename the folder according to your institute name `<institute_name>` to customize the code according to your project activity.

Copy the bash instruction below in a file editing the `<institute_name>` according to your needs:

```
# This script is intended to be source'd, not executed
set -e
REPO_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
# Add additional FABM "institutes" (bfm, ecosmo, etc.)
# and their base directory on the line below.

FABM_ARGS="-DFABM_INSTITUTES=<institute_name>
           -DFABM_<INSTITUTE_NAME>_BASE=${REPO_DIR}/extern/<institute_name>
           -DCMAKE_Fortran_COMPILER=gfortran" # e.g. gfortran or ifort
# Build pyfabm

WORK_DIR=`mktemp -d`
cd ${WORK_DIR}
cmake ${REPO_DIR}/src/drivers/python $FABM_ARGS
make -j4 install
cd -
rm -rf ${WORK_DIR}
```

by sourcing the script above it will create the python code.

To verify correct finalization of the installation you can type from python console

```
import pyfabm
```

and this should not raise any error.

3 The fabm.yaml configuration file

FABM uses a single configuration file, `fabm.yaml`, describing which biogeochemical models to use and how to setup the related parameterizations. The `fabm.yaml` is a text file in YAML format. It contains a hierarchy of information represented by a spatial indentation. Each active model has its own block in the configuration file.

In the case of BFM model coupling, there are separate instances for nutrients, pelagic chemistry (pelchem), bacteria, and 4 phytoplankton, 2 microzooplankton, 2 mesozooplankton groups.

Below it is reported an example of a part of the block related to diatoms in the `fabm_monospectral.yaml` used for BFM:

```
#   ---------------------- P1 Diatom ----------------------
    P1:
     long_name: diatoms
     model: ogs/Phyto
     parameters:
# --------- Physiological parameters -----------------
        p_q10: 2.0        # [-] Characteristic Q10 coefficient
        p_temp: 0.0       # [-] Cut-off threshold for temperature factor
        p_sum: 2.5        # [1/d] maximum specific productivity
                          #  at reference temperature (1/d)
        p_srs: 0.1        # [1/d] Respiration rate at 10 degrees C
        p_sdmo: 0.0       # [1/d] Max.specific nutrient-stress lysis rate
        p_thdo: 0.0       # [-] Half saturation constant for
                          # nutrient stress lysis
        p_seo: 0.0        # [1/d] Extra lysis rate (biomass density-dependent)
        p_sheo: 0.0       # [mgC/3] Half saturation constant for extra lysis
        p_pu_ea: 0.05     # [-] Excreted fraction of primary production
        p_pu_ra: 0.1      # [-] Activity respiration fraction
        p_switchDOC: 2    # [1-3] Switch for the type of DOC excretion
```

and for the coupling with other models invoked in the same `fabm.yaml`:

```
coupling:
     N1p: N1/p # phosphate (mmol P/m^3)
     N3n: N3/n # nitrate (mmol N/m^3)
     N4n: N4/n # ammonium (mmol N/m^3)
     N5s: N5/s # silicate (mmol Si/m^3)
     O3c: O3/c # dissolved inorganic carbon (mg C/m^3)
     O2o: O2/o # Oxygen (mmol O/m^3)
```

4 Running FABM-BFM as a python package

To perform a test simulation first create a folder for the setup from the root directory of FABM.

```
mkdir -p setup/0D
```

then copy the configuration file into the setup folder.

```
cp extern/<institute_name>/yaml-cfgs/fabm_monospectral.yaml setups/0D/fabm.yaml
```

Now it is possible to perform a simulation using a python script, to initialize the biogeochemical model and to perform a time integration of the equations.

If `pyfabm` is installed using a conda environment it is necessary to activate the associated environment in order to import the `pyfabm` package.

As a first operation we import the required python packages

```
import numpy as np
import pyfabm
import netCDF4 as nc
```

the following commands set the time interval, in seconds, to perform the simulation, in the case below we consider a 10 years time window. The state variable "y" is also initialized, in this case BFM has 54 state variables

```
t_eval = np.linspace(0, 3650.*86400, 50000)
y = np.zeros((len(t_eval),54))
```

The time series accounting for temperature variability accounts for daily and seasonal variations

```
#Sinusoidal temperature fluctuation
# parameters
MeanTemp = 15           # Average temperature in the country  Deg Celsius
DailyAmpl = 5           # Amplitude of the daily cycle        Deg Celsius
YearlyAmpl =  5         # Amplitude of the yearly cycle       Deg Celsius


# Total seconds in year
TotalHours = 24*365*60*60 #year period
tau        = 24*60*60     #day period

# Generate the frequency components of the data
DailyCycle = -DailyAmpl*np.cos( (2*np.pi)*t_eval/tau )
YearlyCycle = -YearlyAmpl*np.cos( (2*np.pi)*t_eval/TotalHours )

# Final series
T = MeanTemp + DailyCycle + YearlyCycle
```

Similarly to temperature also solar irradiance is time dependent

```
#Sinusoidal light fluctuation
# parameters
MeanTemp =  0           # Average temperature in the country W/m2
```

```
DailyAmpl = 100          # Amplitude of the daily cycle      W/m2
YearlyAmpl = 50         # Amplitude of the yearly cycle      W/m2

# Total seconds in year
TotalHours = 24*365*60*60 #year period
tau        = 24*60*60     #day period

# Generate the frequency components of the data
DailyCycle = -DailyAmpl*np.cos( (2*np.pi)*t_eval/tau )
YearlyCycle = -YearlyAmpl*np.cos( (2*np.pi)*t_eval/TotalHours )
#Noise = np.random.normal(0, NoiseStd, TotalHours)

# Final series
L = np.where((MeanTemp + DailyCycle + YearlyCycle) > 0, MeanTemp + DailyCycle + YearlyCycle, 0.)
```

The biogeochemical model object is created basing on the yaml configuration file

```
# Create model (loads fabm.yaml)
model = pyfabm.Model('fabm.yaml')
```

The following commands set the environmental regulating factors providing realistic values and check the parameters consistency

```
# Configure the environment
# Note: the set of environmental dependencies depends on the loaded biogeochemical model.
model.dependencies['cell_thickness'].value = 1.
model.dependencies['temperature'].value = T[0]
model.dependencies['practical_salinity'].value = 30.
model.dependencies['density'].value = 1000.
model.dependencies['depth'].value = 1.
model.dependencies['pressure'].value = 1.
model.dependencies['longitude'].value = 0.
model.dependencies['latitude'].value = 0.
model.dependencies['surface_downwelling_shortwave_flux'].value = L[0]
model.dependencies['surface_air_pressure'].value = 1.
model.dependencies['wind_speed'].value = 5.
model.dependencies['mole_fraction_of_carbon_dioxide_in_air'].value = 390.
model.dependencies['number_of_days_since_start_of_the_year'].value = 1.
model.cell_thickness=1.
# Verify the model is ready to be used

assert model.checkReady(), 'One␣or␣more␣model␣dependencies␣have␣not␣been␣fulfilled.'
```

The following blocks performs a temporal integration of the state variables using first order Euler method

```
# Time-integrate over 10 years (note: FABM's internal time unit is seconds!)
dt = (t_eval[-1]-t_eval[0])/len(t_eval)
y[0,:]=model.state[:]

for i in range(len(t_eval)):
    if i!=0:
        model.dependencies['temperature'].value = T[i]
        model.dependencies['surface_downwelling_shortwave_flux'].value = L[i]
        dy = model.getRates()
        for j in range(len(model.state)):
            y[i,j]=y[i-1,j]+dy[j]*dt
        model.state[:]=y[i,:]
```

The last part of the script save the state vector on a netcdf file, specific diagnostics can be computed directly in the script or offline loading the NetCDF output file.

```
t = t_eval/86400
Nt=t.shape[0]
deltaT=t[1]-t[0]
# Save results
fileoutput = 'result.nc'
f = nc.Dataset(fileoutput, mode='w')
lat_dim = f.createDimension('lat', 1)
lon_dim = f.createDimension('lon', 1)
dep_dim = f.createDimension('z', 1)
time_dim = f.createDimension('time', Nt)
lat = f.createVariable('lat', np.float32, ('lat',))
lat.units = 'degrees_north'
lat.long_name = 'latitude'
f.variables['lat'][:]=0
lon = f.createVariable('lon', np.float32, ('lon',))
lon.units = 'degrees_east'
lon.long_name = 'longitude'
f.variables['lon'][:]=0
time = f.createVariable('time', np.float64, ('time',))
time.units = 'days'
time.long_name = 'time'
f.variables['time'][:]=t
depth = f.createVariable('z', np.float32, ('z',))
depth.units = 'meters'
depth.long_name = 'depth'
f.variables['z'][:]=1
temp = f.createVariable('temp', np.float64, ('time',))
temp.units = 'C'
temp.long_name = 'temperature␣in␣celsius'
f.variables['temp'][:]=T
light = f.createVariable('light', np.float64, ('time',))
light.units = 'W/m^2'
light.long_name = 'surface_downwelling_shortwave_flux'
f.variables['light'][:]=L
for v,variable in enumerate(model.state_variables):
    ncvar = variable.name.replace("/","_")
    var = f.createVariable(ncvar, np.float64, ('time', 'z','lat','lon'))
    var.units = variable.units
    var.long_name = variable.long_name
    f.variables[ncvar][:]=y[:,v]
f.close()
```

5 Testing the BFM coupling with FABM

In order to test the accuracy of the BFM porting we used both the python coupling (https://github.com/fabm-model/fabm/wiki/python) and the 0D FORTRAN setup (fabm/src/-drivers/0d/) and a 2-phase check was implemented. In the first phase we performed a test checking that all the diagnostics were identical up to machine precision between the two FABM configurations. In the second phase, we performed a check of the integration in time by setting up a 10 day simulation using the original BFM in a standalone model v5.3 (http://bfm-community.eu/bfm-quick-guide/) and the BFM-FABM 0D configuration. The two configurations had the same initial conditions, the same constant environmental forcing (i.e., constant light, T, atmospheric CO_2 and wind speed) and the time step of 864 s. Output were saved at each time step to avoid differences arising from interpolation in time. Results show an extremely satisfactorily agreement between the two configurations for all state variables, see examples shown in Fig.5.1.

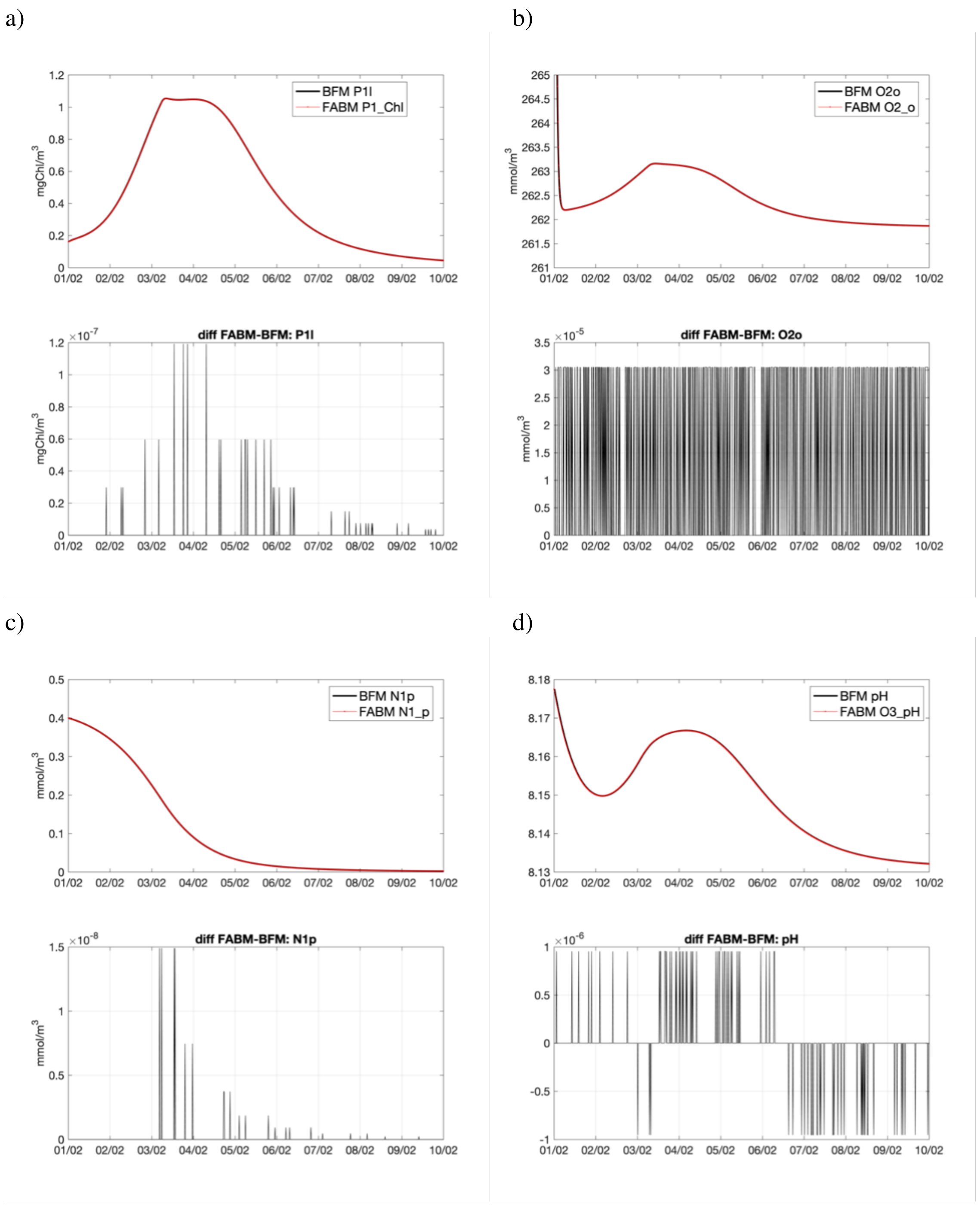

Figure 5.1: Results for the comparison between BFM official configuration and FABM coupled BFM. All state variables have been checked and here a subset is proposed, chlorophyll content in Diatoms a), Oxygen b), PO4 c) and pH d). Each panel shows a 10 days time-series of BFM standalone (black line) with super imposed FABM-BFM (red line). Each timeseries is accompanied by the differences between the two simulations (diff FABM-BFM).

6 Output and diagnostics

Below is reported a python script that illustrate how to plot results saved as netCDF file following the example model script reported in Sec. 4. The results of the simulation for population biomasses are shown in Fig.6.1.

```
import numpy as np
import netCDF4 as nc
import matplotlib.pyplot as plt
ncname = 'result.nc'
f_det = nc.Dataset(ncname)
varnames=['B1_c','P1_c','P2_c','P3_c','P4_c',
          'Z5_c','Z6_c','Z3_c','Z4_c']
t = np.linspace(0, 3650., 50000)
fig,axs = plt.subplots(3,3,sharex=True)#,constrained_layout=True)
fig.tight_layout()
plt.subplots_adjust(top=0.90)
plt.subplots_adjust(left=0.15)
plt.subplots_adjust(bottom=0.10)
axs = axs.ravel()
titles = ['Bacteria␣B1', 'Diatoms␣P1', 'Nanoflagellates␣P2',
          'Picophytoplankton␣P3','Dinoflagellates␣P4',
          'Microzooplankton␣Z5', 'het.␣Nanoflagellates␣Z6',
          'carn.␣Mesozooplankton␣Z3', 'omn.␣Mesozooplankton␣Z4']
for iax,ax in enumerate(axs):
    lns2 = ax.plot(t[:],f_det.variables[varnames[iax]][:,0,0,0]
                  ,c='k',label=r'$D=0$')
    ax.set_title(titles[iax],fontsize=8)
    ax.tick_params(axis='both', which='major', labelsize=7)
fig.text(0.5, 0.04, 'time␣[days]', ha='center')
fig.text(0.04, 0.5, 'Mean␣Concentration␣[$mg␣C/m^3$]'
         , va='center', rotation='vertical')
fig.savefig('results.png', format='png',dpi=250)
```

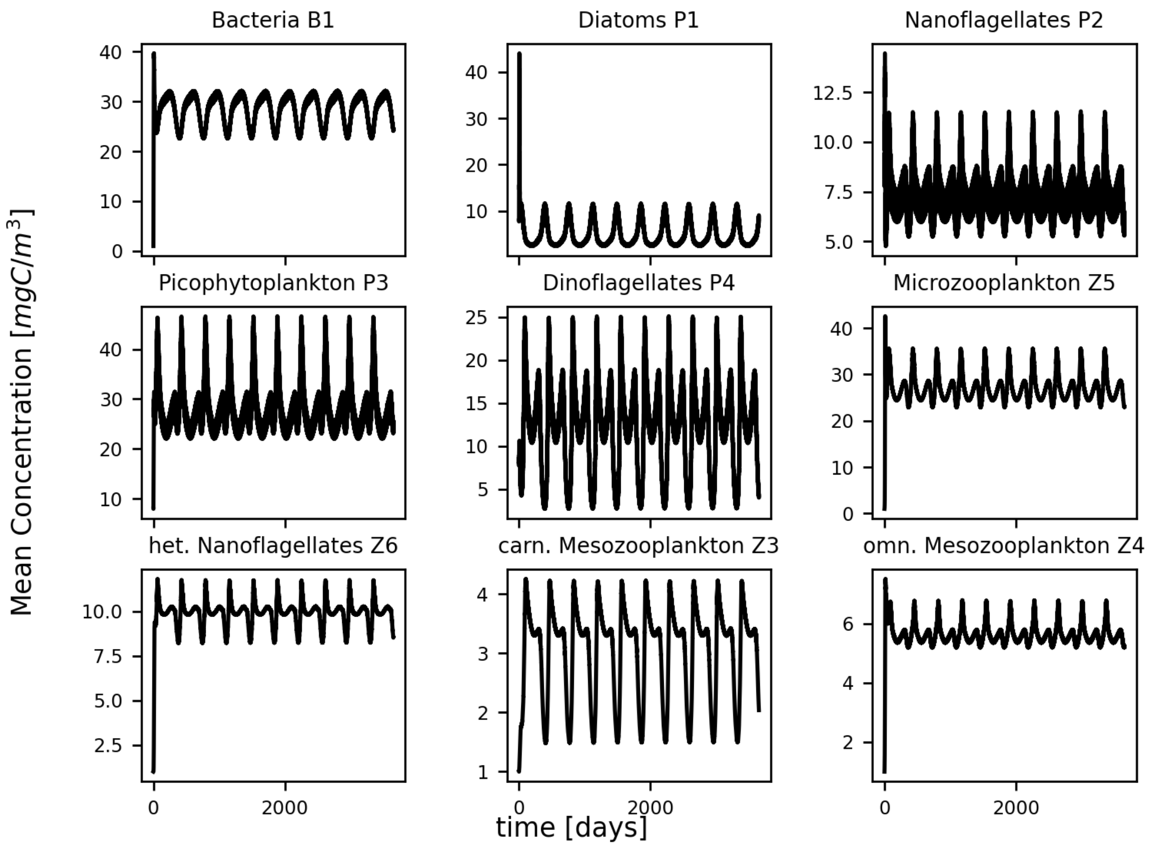

Figure 6.1: Example of output simulation using the python wrapper

Bibliography

Álvarez, E., Cossarini, G., Teruzzi, A., Bruggeman, J., Bolding, K., Ciavatta, S., Vellucci, V., D'Ortenzio, F., Antoine, D., Lazzari, P., 2023. Chromophoric dissolved organic matter dynamics revealed through the optimization of an optical-biogeochemical model in the NW mediterranean sea.
URL `https://bg.copernicus.org/preprints/bg-2023-48/`

Bruggeman, J., Bolding, K., 2014. A general framework for aquatic biogeochemical models. Environmental modelling & software 61, 249–265.

Cossarini, G., Lazzari, P., Solidoro, C., 2015. Spatiotemporal variability of alkalinity in the mediterranean sea. Biogeosciences 12 (6), 1647–1658.

Lazzari, P., Álvarez, E., Terzic, E., Cossarini, G., Chernov, I., D'Ortenzio, F., Organelli, E., 2021. CDOM spatiotemporal variability in the mediterranean sea: A modelling study 9 (2), 176.
URL `https://www.mdpi.com/2077-1312/9/2/176`

Lazzari, P., Solidoro, C., Ibello, V., Salon, S., Teruzzi, A., Béranger, K., Colella, S., Crise, A., 2012. Seasonal and inter-annual variability of plankton chlorophyll and primary production in the mediterranean sea: a modelling approach 9 (1), 217–233.
URL `https://bg.copernicus.org/articles/9/217/2012/`

Lazzari, P., Solidoro, C., Salon, S., Bolzon, G., 2016. Spatial variability of phosphate and nitrate in the mediterranean sea: A modeling approach 108, 39–52.
URL `https://linkinghub.elsevier.com/retrieve/pii/S0967063715301473`