# R package rodeoFABM: Basic Use and Sample Applications

johannes.feldbauer@tu-dresden.de

2020-04-14

## Contents

**1 Main features of `rodeoFABM`** **1**

**2 Installation and requirements** **1**

**3 Basic use** **2**
3.1 First example (how it works) . . . 2
3.2 Create a model step by step . . . 5
3.2.1 Getting dependencies from the host model . . . 9
3.3 Sedimentation . . . 12
3.3.1 Processes at the upper and lower boundaries (surface and sediment) . . . 15
3.3.2 Sediment or surface attached state variables . . . 18
3.4 Additional features . . . 23
3.4.1 Additional state variable arguments . . . 23
3.4.2 Profile initial values for the state variables . . . 23
3.4.3 Defining own functions . . . 24
3.4.4 Automatic model documentation . . . 24

**References** **25**

# 1 Main features of `rodeoFABM`

The `R` package `rodeoFABM` is a colletion of functions to help create water quality models that can be coupled to physical host models using the `FABM` interface (Bruggeman and Bolding (2014)). As the name suggests it is heaviely influenced by the `R` package `rodeo` (Kneis, Petzoldt, and Berendonk (2017)). The principle idea is to creat tools that:

- Decouple the "code writing" from the "model developement" part of creating a model
- Make model adaptation, communikation, and maintenace easy

The main idea is to store the model equations in the standard Peterson matrix notation and saved in text files or spread sheets. The package `rodeoFABM` then can automatically generate `FABM` specific FORTRAN code from these files, create `.yaml` control files for the water quality model and, automatically compile `GOTM` (Burchard et al. (2006)) coupled with the model.

# 2 Installation and requirements

In order to use `rodeoFABM` and run the examples some tools are neede:

- The GNU compilers

- GNU Make
- GNU CMake
- Rdevtools
- git
- the netcdf libraries
- R packages: `readODs`, `plot3D`, `ncdf4`, and `ColorBrewer`

The package `rodeoFABM` can be installed from github using:

```
library("devtools")
install_github("JFeldbauer/rodeoFABM")
```

# 3 Basic use

## 3.1 First example (how it works)

To demonstrate the workflow we will create and compile a simple model. The files used in the example are contained in the package and can be copied to the current working directory using:

```
# copy example ods file
example_model <- system.file("extdata/simple_model.ods", package = "rodeoFABM")
file.copy(from = example_model, to = ".", recursive = TRUE)
```

This will copy the Libre Office spread sheet *simple_model.ods* to your current working directory. Now we can read in the tables with the declarations of state variables, model parameters, used functions and external dependencies, process rate descriptions, and stoichiometry matrix.

```
library(readODS)

# read in example ods file
odf_file <- "simple_model.ods"
vars <- read_ods(odf_file, sheet = 1)
pars <- read_ods(odf_file, sheet = 2)
funs <- read_ods(odf_file, sheet = 3)
pros <- read_ods(odf_file, sheet = 4)
stoi <- read_ods(odf_file, sheet = 5)
```

We store the declarations in the five data.frames *vars*, *pars*, *funs*, *pros*, and *stoi*. Using these we can now generate FORTRAN files using the function `gen_fabm_code()`

```
library(rodeoFABM)

# generate fabm code
gen_fabm_code(vars,pars,funs,pros,stoi,"simple_model.f90",diags = TRUE)
```

```
## Checking model..

## Warning in chk_units(pars, "parameter"): Units of parameter mu_max, k_death,
## k_O2_exch, k_O2_cons seem not to be in x per second. FABM demands that the rate
## of change in the processes is in per second. Please change the unit (and value)

## Warning in chk_units(pros, "process"): Units of process growth, death, O2_exch,
## O2_cons, sed_C seem not to be in x per second. FABM demands that the rate of
## change in the processes is in per second. Please change the unit (and value)

## Model input OK
## Writing simple_model.f90 fortran90 file
```

```
## Writin fabm.yaml file
##
## finished
```

This will create two new files: the *FABM* specific FORTRAN source code *simple_model.f90* and the control file *fabm.yaml* which can be used to change model parameters and initial conditions. The function also checks if all parameter, functions, and state variables used are also decalred and issues a warning because the units decalred for the parameters are not in seconds, which is required by *FABM*.

With the source code file *simple_model.f90* we can compile *GOTM-FABM*. Therfore first we need to clone the lake branche of *GOTM* from github and prepare the build process using cmake. This needs only to be done once, and can automatically be done by running the function `clone_GOTM()`:

```
# clone github repo
clone_GOTM(build_dir = "build", src_dir = "gotm_src")
```

This will take a moment and download the source code for *GOTM* and *FABM* as well as prepare the compilation using *CMake*. You can see that there are now two new folders in the working directory called *gotm_src* and *build*. Now we can build *GOTM-FABM* with our simple model using the *simple_model.f90* file and the `build_gotm()` function:

```
# build GOTM
build_GOTM(build_dir = "build",fabm_file = "simple_model.f90",
           src_dir = "gotm_src")
```

This will copy the *simple_model.f90* file we just created to the correct folder within the *gotm_src* folder and then compile *GOMT-FABM* using *Make*. As a last step it will copy the created executable to the current working directory. You can see that there is now a *gotm* executable file in the working directory. In order to run our created model we will need a *gotm.yaml* file (the GOTM controll fille), an hypsograph file, and the meteorological forcing data. We will copy the example files provides in thos package by using:

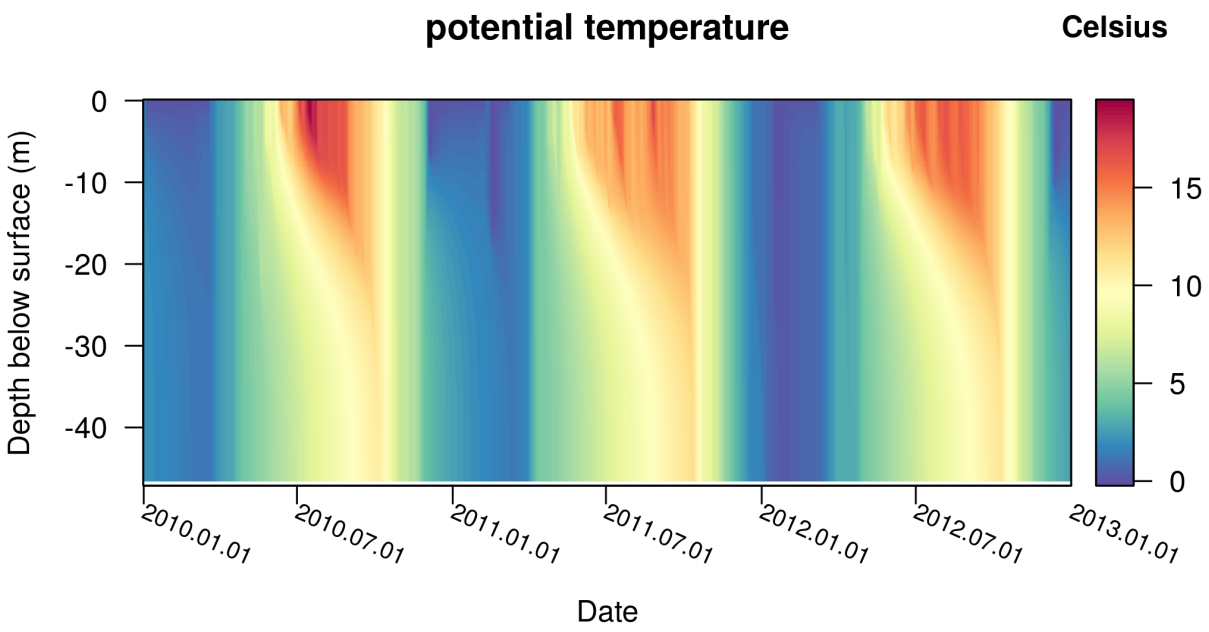
```
# copy example gotm.yaml
yaml <- system.file("extdata/gotm.yaml", package = "rodeoFABM")
file.copy(from = yaml, to = ".", recursive = TRUE)
# write hypsograph
write.table(hypsograph, "hypsograph.dat", sep = "\t", row.names = FALSE,
            quote = FALSE)
# write meteo data
write.table(meteo_file, "meteo_file.dat", sep = "\t", row.names = FALSE,
            quote = FALSE)
```

Now that we have the files *gotm.yaml*, *hypsograph.dat*, and *meteo_file.dat* in our working directory, we can run *GOTM-FABM* using:

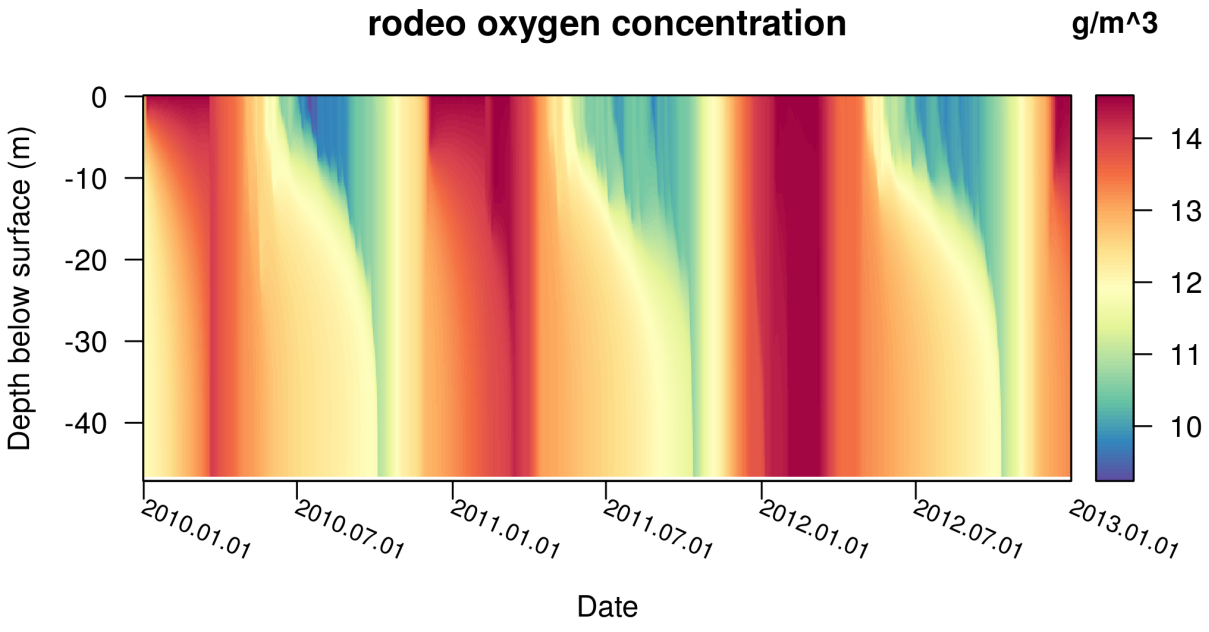
```
# run gotm
system2("./gotm")
```

After successfully running there are two new files: *output.nc* and *restart.nc*, both are netcdf files. *output.nc* is the output of the model run and *restart.nc* is a netcdf file that can be used to initialize a simulation with states stored from the previous run. We can plot the results e.g. by using the `plot_var()` function:

```
# plot temperature
plot_var("output.nc", "temp")
```




```
# plot oxygen
plot_var("output.nc", "rodeo_C_O2")
```




These are the essential steps used to build and run a *GOTM-FABM* model. In the next section we will step by step go a little bit more into the details of building a own model using the `rodeoFABM` package.

## 3.2 Create a model step by step

In order to demonstrate the necessary steps and functionalities we will create a simple phytoplankton-nutrients model and step by step add more processes and state variables.

All used Libre Office spread sheets containing the model information, the GOTM controll file, and the forcing data are contained in the `rodeoFABM` package. You can copy all necessary files to run GOTM useing the same method described before. We will use the same meteorological forcing (*meteo_file.dat*) and hypsographic curve (*hypsograph.dat*) as in the first example. Additionally we now add one inflow and one outflow (files `inflow_m.dat`, `outflow.dat`, and `inflow_wq_m.dat` containing the nutrient concentrations of the inflow)

```
# GOTM controll fille
yaml <- system.file("extdata/examples/gotm.yaml",
                    package = "rodeoFABM")
file.copy(from = yaml, to = ".", recursive = TRUE)
# inflow hydrological data
infl <- system.file("extdata/examples/inflow_m.dat",
                    package = "rodeoFABM")
file.copy(from = infl, to = ".", recursive = TRUE)
# inflow nutient data
nut <- system.file("extdata/examples/inflow_wq_m.dat",
                    package = "rodeoFABM")
file.copy(from = nut, to = ".", recursive = TRUE)
# outflow data
out <- system.file("extdata/examples/outflow.dat",
                    package = "rodeoFABM")
file.copy(from = out, to = ".", recursive = TRUE)
```

The inflows and especially the inflow of state variables to a *FABM* model are defined in the `streams` section of the *GOTM* control file (*gotm.yaml*). The section looks like this:

```
streams:
   inflow:                                  # stream configuration
      method: 4                               # inflow method, default=1
      zu: 0.0                                 # upper limit m
      zl: 0.0                                 # lower limit m
      flow:                                   # water flow
         method: 2                            # 0=constant, 2=from file, default = 0
         constant_value: 1.0                  # constant value( m^3/s)
         file: inflow_m.dat                   # path to file with time series
         column: 1                            # index of column to read from
      temp:                                   # flow temperature
         method: 2                            # 0=constant, 2=from file; default=0
         constant_value: 10.0                 # constant value (°C)
         file: inflow_m.dat                   # path to file with time series
         column: 2                            # index of column to read from
      salt:                                   # flow salinity
         method: 0                            # 0=constant, 2=from file; default=0
         constant_value: -1.0                 # constant value (PSU)
         file: inflow.dat                     # path to file with time series
         column: 3                            # index of column to read from
      rodeo_HPO4:                             # rodeo phosphprus
         method: 0                            # 0=constant, 2=from file; default=0
         constant_value: 0.5                  # constant value (gP/m^3)
         file: inflow_wq_m.dat                # path to file with time series
         column: 4                            # index of column to read from
```

Within the `streams` section several in- and outflows can be defined with any desired name (here "inflow"). The inflow/outflow depth is defined by `streams/method`, whereas `1` means surface, `2` means bottom, `3` mean a specified range of deapths defined by `streams/zu` (upper) and `streams/zl` (l), and `4` means inflow to the depth with same temperature as the inflow temperature. Every in- or outflow needs the `streams/flow` section defining the flow rate in $m^3/s$ and can have additional entries like `streams/temp` for temperature or inflowing state variables of the *FABM* model (like `streams/rodeo_HPO4`). The *FABM* sate variables need to start with *rodeo_* followd by the defined state variable name. The values can either be constant (`streams/rodeo_HPO4/method` = 0) or a time series given by a tab separated file (`streams/rodeo_HPO4/method` = 2) with first column datetime (as `YYYY-mm-dd HH:MM:ss`). The name of the file is supplied by `streams/rodeo_HPO4/file` and the column the variable is in by `streams/rodeo_HPO4/column`, take care: the first column with datetime is not counted and if the columns have a headder it needs to start with an excalamation mark "!".

We can create the source code of the phytoplankton nutients model in the same way as we we created the source code in the first example:

```
# copy the spread sheet
ods <- system.file("extdata/examples/simple_alg.ods",
                   package = "rodeoFABM")
file.copy(from = ods, to = ".", recursive = TRUE)

# declare data frames for vars, pars, funs, pros, and stoi
vars <- read_ods("simple_alg.ods", sheet = "vars")
pars <- read_ods("simple_alg.ods", sheet = "pars")
funs <- NULL
pros <- read_ods("simple_alg.ods", sheet = "pros")
stoi <- read_ods("simple_alg.ods", sheet = "stoi")
```

This first model is a simple model with two state variables, which are declared in the `vars` data frame. The table needs to have at least three columns *name* giving the identifier of the state variable, *unit* giving the used unit, and *description* giving a short description of the state variable. If additionally the column *default* is supplied the initial value will be included in the *FABM* control file (`fabm.yaml`), which is automatically generated by `gen_fabm_code()`.

Table 1: Data set `vars`: Declaration of state variables.

| name | unit | description | default |
|---|---|---|---|
| $C$ | gDM/mˆ3 | algae concentration | 0.0 |
| $HPO4$ | gP/mˆ3 | phosphorus concentration | 0.1 |

The models parameters are defined in the `pars` data frame in a similar fashion. They need the same three columns *name*, *unit*, and *description* and can have the aditional column *default* as well. Take care that *FABM* requires all parameters with relation to time to be in units of second.

Table 2: Data set `pars`: Declaration of model parameters.

| name | unit | description | default |
|---|---|---|---|
| $mu\_max$ | 1/s | maximum growth rate | 1e-05 |
| $K\_P$ | W/mˆ2 | half saturation concentration of HPO4 limitation | 2e-02 |
| $k\_death$ | 1/s | death rate | 2e-06 |
| $a\_P$ | gP/gDM | phosphorus content of phytoplankton | 5e-02 |

External functions, or forcing data that needs to be obtained from the physical host model ( e.g. water temperature) are defined in `funs`. As the first simpole model has no such things this is explaine in the later steps. As in this example the data frame is not needed it hast to be set to `NULL`.

The declaration of the processes and process rates is done in the `pros` data frame. It has four requred columns: *name* giving the name of the proicess, *unit* giving the unit of the process rate (again in seconds!), *description* giving a short description of the process, and *expression* giving the mathematical expression of the process. There can be additional columns to define the saptial domain of the process, or to declare sinking processes, but they will be explained later.

Table 3: Data set `pros`: Declaration of processes.

| name | unit | description | expression |
|---|---|---|---|
| growth | g/m^3/d | growth of algae | $C \cdot mu\_max \cdot HPO4/(HPO4 + K\_P)$ |
| death | g/m^3/d | death of algae | $C \cdot k\_death$ |

The phytoplankton have a simple linear growth term with a Monod like limitation for the limiting nutrient Phosphorus and a linear decay/death term.

The last data frame `stoi` gives the stoichiometry table (in long format) connecting the process rates with the state variables. It has three required columns: *variable* giving the variable affected by the *process*, and *expression* giving a factor to multiply the process rate by:

Table 4: Data set `stoi`: Declaration of stoichiometry matrix in long format.

| variable | process | expression |
|---|---|---|
| $C$ | growth | $1$ |
| $C$ | death | $-1$ |
| $HPO4$ | growth | $-1 \cdot a\_P$ |

The growth of phytoplankton is increasing its concentration $C$ and decreasing the nutrient $HPO4$ by the fraction of $a_P$, which is the Phosphorus conten of the phytoplankton. Decay/death is just decreasing phytoplankton concentration $C$.

Having declared all five data frames we can now generate the fortran code using `gen_fabm_code()`. This will also perfome some automated checks e.g. if all used parameters and state variables are also declared, and will issue a warning if the used units are not using seconds for time. It will also create the *FABM* control file `fabm.yaml` and insert the default values for parameters and initial values (if declared). If the argument `diags` is set to TRUE the process rates are stored as diagnostic variables in the output netcdf file.

```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_1.f90", diags = TRUE)
```

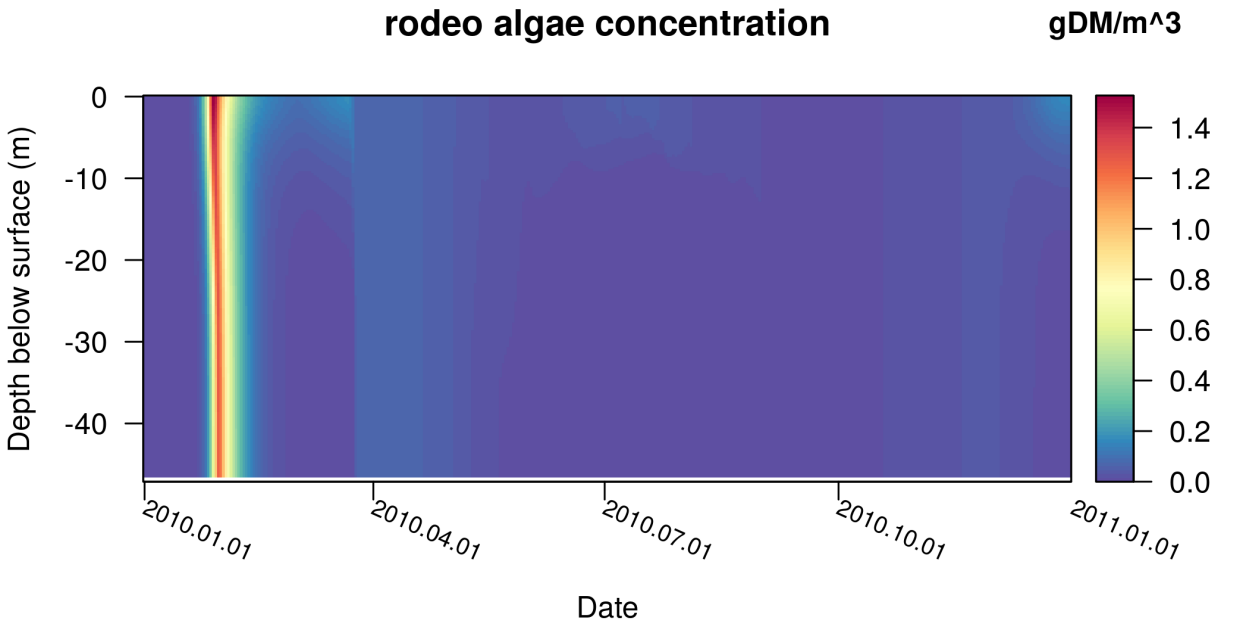
After creating the fortran source code, *GOTM-FABM* can be automatically compiled unsing the function `build_GOTM()` (assuming the source code was allready fetched and prepared for compilation using `clone_GOTM()`), this will also copy the compiled executable to the current working directory, which then can be ran using e.g. `system2("./gotm")`.

```
# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src",
           fabm_file = "model_1.f90")

# run the model
```

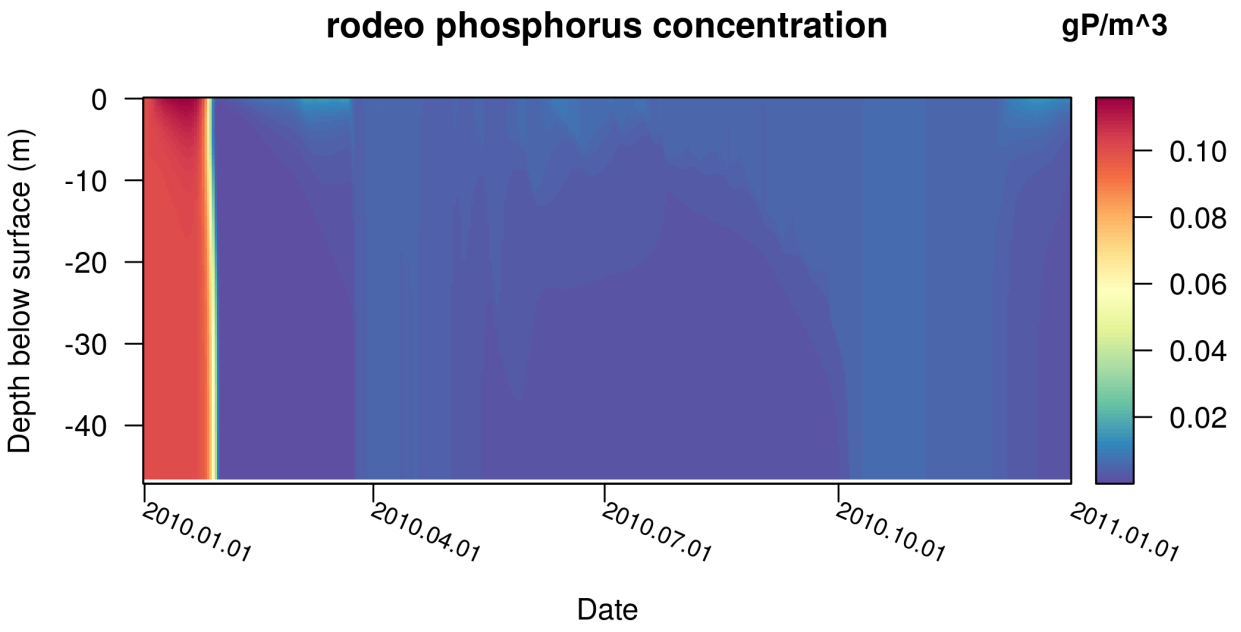
```
system2("./gotm")
```

We can now plot the model results e.g. using `gotmtools::plot_vari()`.

```
# plot the variables
plot_var("output.nc", "rodeo_C")
```




```
plot_var("output.nc", "rodeo_HPO4")
```



### 3.2.1 Getting dependencies from the host model

As many biogeochemical processes depend on external forcing, such as temperature or available iradiation, these values can be obtained from the physical host model. In the next step we want to add the dependency of phytoplankton growth on available iradiation. We first copy the prepared spread sheet and declare the data frames:

```
ods <- system.file("extdata/examples/simple_alg_par.ods",
                   package = "rodeoFABM")
file.copy(from = ods, to = ".", recursive = TRUE)

# declare data frames for vars, pars, funs, pros, and stoi
vars <- read_ods("simple_alg_par.ods", sheet = "vars")
pars <- read_ods("simple_alg_par.ods", sheet = "pars")
funs <- read_ods("simple_alg_par.ods", sheet = "funs")
pros <- read_ods("simple_alg_par.ods", sheet = "pros")
stoi <- read_ods("simple_alg_par.ods", sheet = "stoi")
```

Now we need to get values for the photosynthetic active radiation (PAR) from GOTM. *FABM* has so called "standard-variables" with defined names (stored in the `std_names_FABM` data). If you want to access these variables you need to defind them as a function in the `funs` data frame and add the additional column *dependency* wich contains the full standard-variable name. The data frame `funs` has three required columns that are the same as in `vars`, and `pars`: *name*, *unit*, and *description*, additionally the column *dependency*. If you declare several functions of whome some are not dependencies the corresponding entry in column *dependency* needs to be `NA` for these and the corresponding standard-name for the ones that are dependencies.

Table 5: Data set `funs`: Declaration of model functions and dependencies from the host model.

| name | unit | description | dependency |
|---|---|---|---|
| *par* | W/m^2 | Downwelling photosynthetic radiative flux | downwelling_photosynthetic_radiative_flux |

The declared functions/dependencies can now be used in the process expression, same as parameters and state variables. We added a Monod Term for light limitation in the *growth* process:

Table 6: Data set `pros`: Declaration of processes.

| name | unit | description | expression |
|---|---|---|---|
| growth | g/m^3/d | growth of algae | $C \cdot mu\_max \cdot HPO4/(HPO4 + K\_P) \cdot par/(par + K\_par)$ |
| death | g/m^3/d | death of algae | $C \cdot k\_death$ |

For this we need to declare the additional parameter for the half-saturation irradiation in the `pars` data frame:

Table 7: Data set `pars`: Declaration of model parameters.

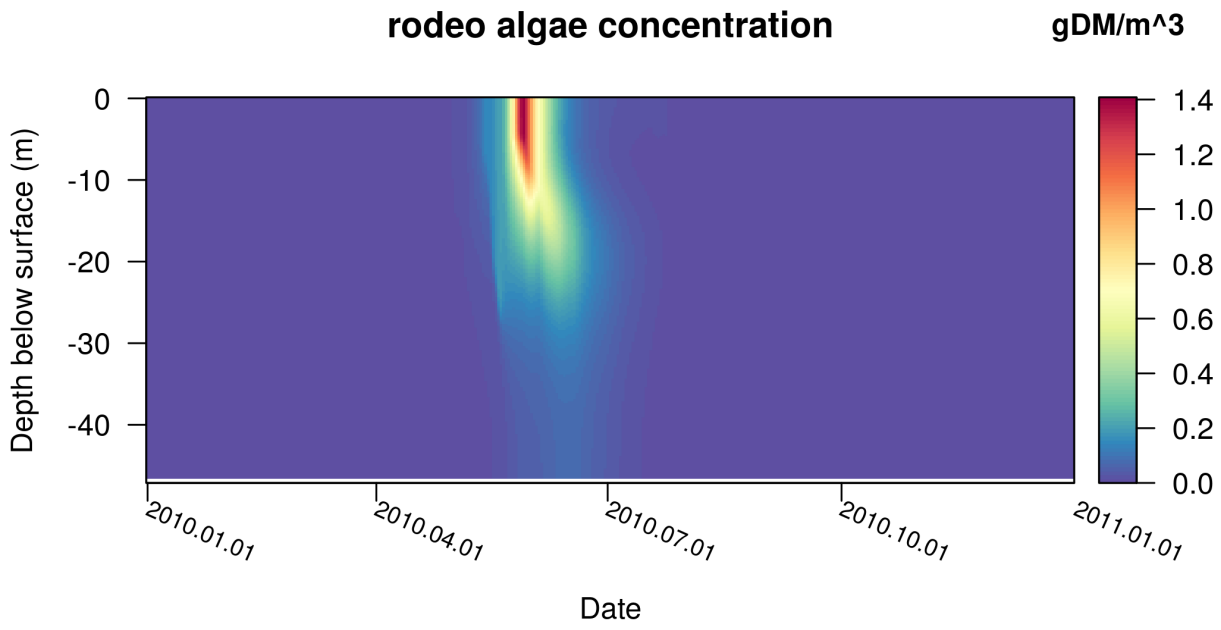
| name | unit | description | default |
|---|---|---|---|
| $mu\_max$ | 1/s | maximum growth rate | 1.0e-05 |
| $K\_P$ | W/m^2 | half saturation of photosyntetic flux | 2.0e-02 |
| $k\_death$ | 1/s | death rate | 2.0e-06 |
| $a\_P$ | gP/gDM | phosphorus content of phytoplankton | 5.0e-02 |
| $K\_par$ | W/m^2 | half saturation of photosyntetic flux | 2.7e+01 |

Now we can generate the fortran code, compile *GOTM-FABM*, and run the adapted model.

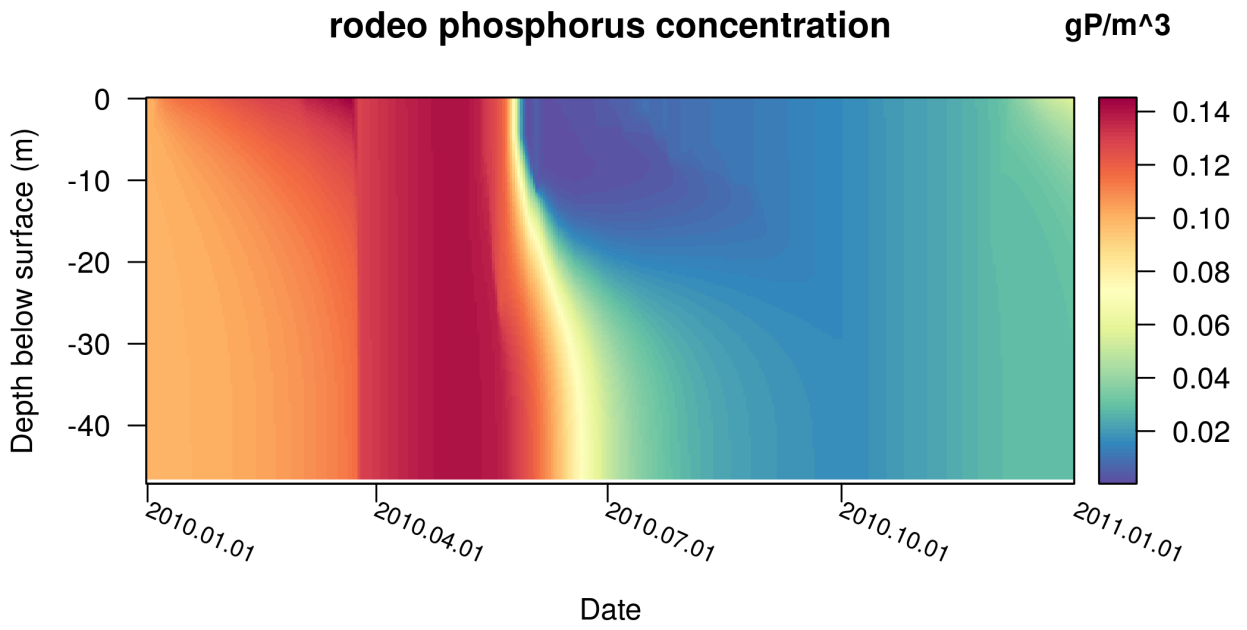
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_2.f90", diags = TRUE)

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src",
           fabm_file = "model_2.f90")

# run the model
system2("./gotm")
```

And plot some of the simulated state variables:

```
# plot the variables
plot_var("output.nc", "rodeo_C")
```

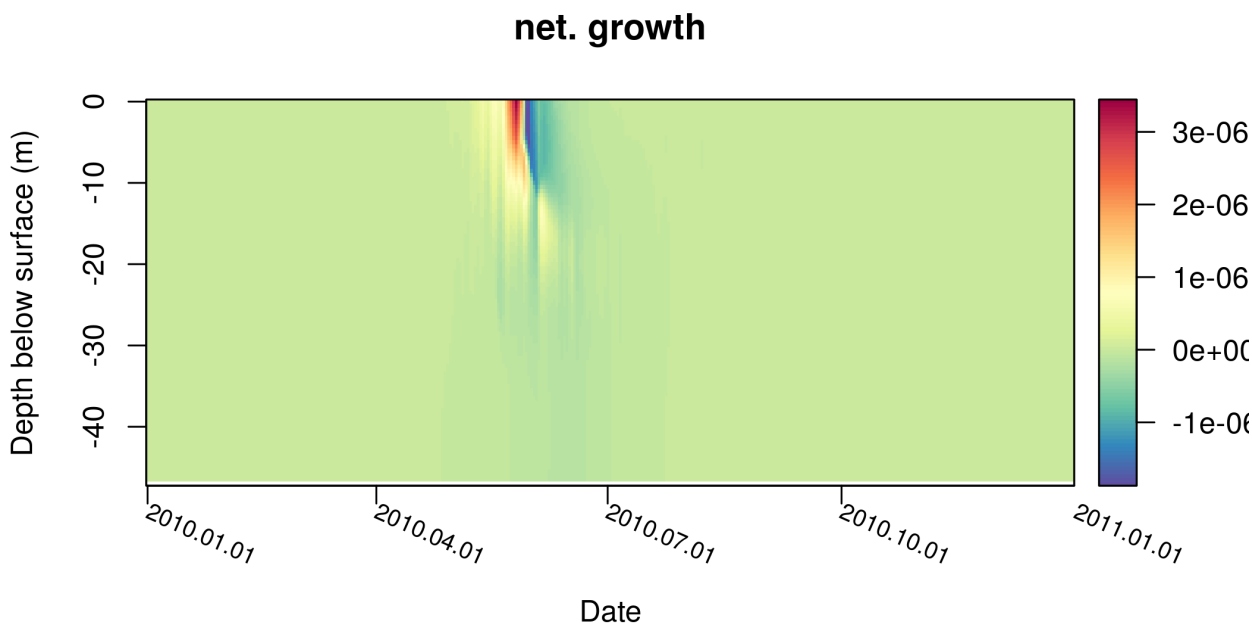



```
plot_var("output.nc", "rodeo_HPO4")
```




From the saved process rates (the diagnostic variables) we can plot e.g. the net. growth rate. We can acces the values stored in the netcdf file e.g. using the function `gotmtools::get_vari()` and plot them using `gotmtools::long_heatmap()`:

```
library(plot3D)

# also plot net. growth
growth <- get_var("output.nc", "rodeo_growth")
death <- get_var("output.nc", "rodeo_death")
net_growth <- growth$var - death$var
# nice colors
mycol <- colorRampPalette(rev(RColorBrewer::brewer.pal(11, 'Spectral')))
# plot the net. growth
image2D(net_growth, growth$time, growth$z, main = "net. growth", col = mycol(100), xaxt = "n",
        xlab = "Date", ylab = "Depth below surface (m)")
axis(1, at = pretty(growth$time), labels = FALSE)
text(pretty(growth$time), par("usr")[3] - 3, labels = format(pretty(growth$time), "%Y.%m.%d"),
     xpd = NA, srt = 336, adj = 0.0, cex = 0.8)
```




### 3.3 Sedimentation

Often in biogeochemical models some state variables are sinking in the water body ( e.g. phytoplankton or particular organic matter). In the next adaptation of the model we want to include a constant sinking velocuty for the phytoplankton. Therefore, again we first copy the spread sheets from the package data and declare the data frames:

```
ods <- system.file("extdata/examples/simple_alg_par_sed.ods",
                   package = "rodeoFABM")
file.copy(from = ods, to = ".", recursive = TRUE)

# declare data frames for vars, pars, funs, pros, and stoi
vars <- read_ods("simple_alg_par_sed.ods", sheet = "vars")
pars <- read_ods("simple_alg_par_sed.ods", sheet = "pars")
funs <- read_ods("simple_alg_par_sed.ods", sheet = "funs")
pros <- read_ods("simple_alg_par_sed.ods", sheet = "pros")
stoi <- read_ods("simple_alg_par_sed.ods", sheet = "stoi")
```

*FABM* allows for time varying sinking of state variables. This in implemented in *rodeoFABM* as a process declared in the `pros` data frame that has a logical flag set in an additional column called *sedi*. The expression for this can also be a function of external dependencies (e.g. water density) or internal state variables (e.g. nutrient concentration), in this simple case we choose a constant sinking velocity:

Table 8: Data set `pros`: Declaration of processes.

| name | unit | description | expression | sedi |
|---|---|---|---|---|
| growth | g/m^3/d | growth of algae | $C \cdot mu\_max \cdot HPO4/(HPO4 + K\_P) \cdot par/(par + K\_par)$ | NA |
| death | g/m^3/d | death of algae | $C \cdot k\_death$ | NA |
| sed | g/m^3/s | sinking | $v\_sed$ | TRUE |

For this to work we need to declare the additional parameter for the sinking velocity:

Table 9: Data set `pars`: Declaration of model parameters.

| name | unit | description | default |
|---|---|---|---|
| *mu_max* | 1/s | maximum growth rate | 1.0e-05 |
| *K_P* | W/m^2 | half saturation of photosyntetic flux | 2.0e-02 |
| *k_death* | 1/s | death rate | 2.0e-06 |
| *a_P* | gP/gDM | phosphorus content of phytoplankton | 5.0e-02 |
| *K_par* | W/m^2 | half saturation of photosyntetic flux | 2.7e+01 |
| *v_sed* | m/s | sedimentation velocity | 1.0e-06 |

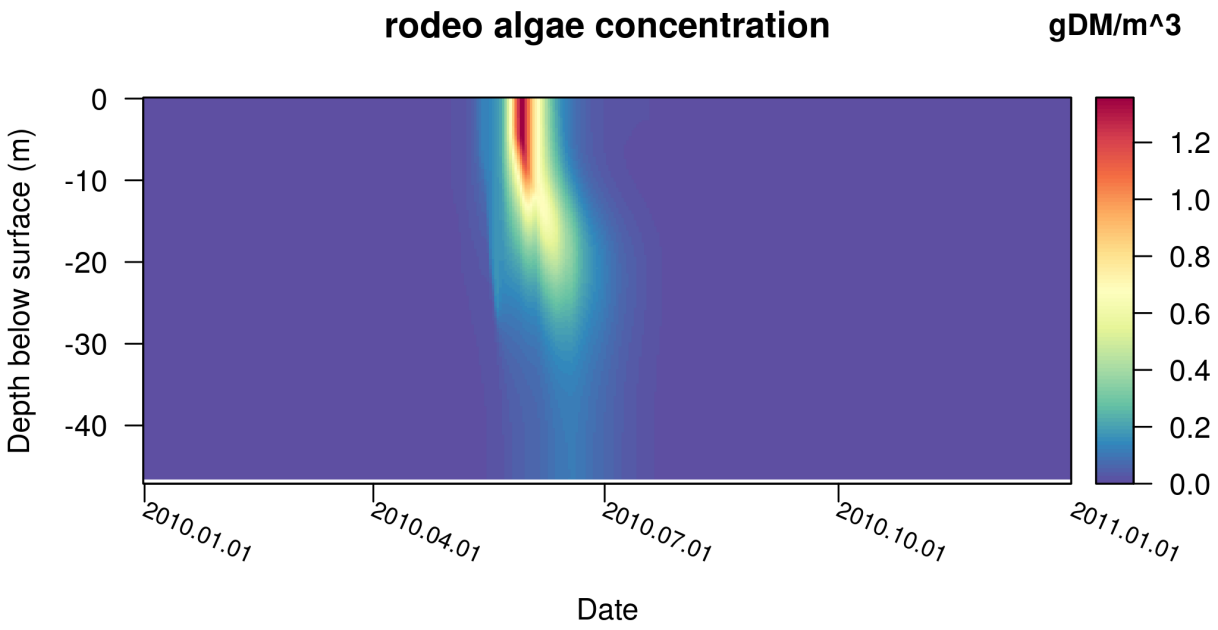
And add the process to the stopichiometry table:

Table 10: Data set `stoi`: Declaration of stoichiometry matrix in long format.

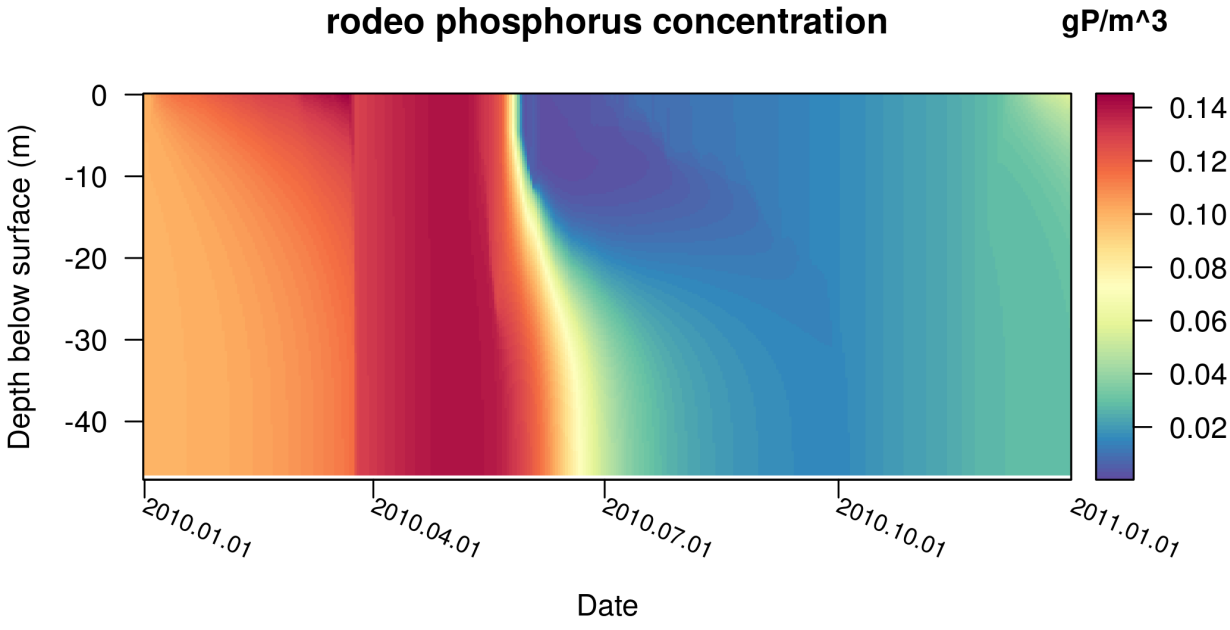
| variable | process | expression |
|---|---|---|
| $C$ | growth | 1 |
| $C$ | death | $-1$ |
| $C$ | sed | $-1$ |
| $HPO4$ | growth | $-1 \cdot a\_P$ |

Now we can create the fortran source file, compile *GOTM-FABM*, run the model and plot some of the results:

```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_3.f90")

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src",
           fabm_file = "model_3.f90")

# run the model
system2("./gotm")

# plot the variables
plot_var("output.nc", "rodeo_C")
```

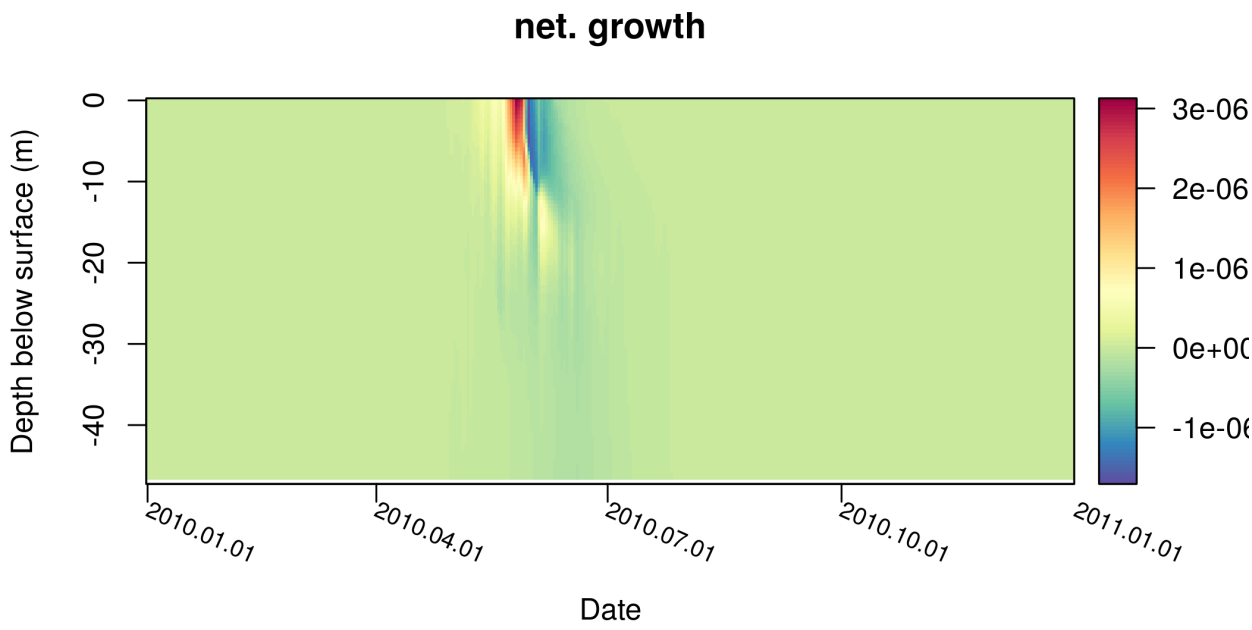



```
plot_var("output.nc", "rodeo_HPO4")
```




```
# also plot net. growth
growth <- get_var("output.nc", "rodeo_growth")
death <- get_var("output.nc", "rodeo_death")
net_growth <- growth$var - death$var

image2D(net_growth, growth$time, growth$z, main = "net. growth", col = mycol(100), xaxt = "n",
```

```
        xlab = "Date", ylab = "Depth below surface (m)")
axis(1, at = pretty(growth$time), labels = FALSE)
text(pretty(growth$time), par("usr")[3] - 3, labels = format(pretty(growth$time), "%Y.%m.%d"),
     xpd = NA, srt = 336, adj = 0.0, cex = 0.8)
```




### 3.3.1 Processes at the upper and lower boundaries (surface and sediment)

There are some processes that only take place at the surface or bottom (sediment) of lakes. *FABM* knows three spatial domains: open water (pelagial), surface, and bottom (sediment) and processes can be declared to only take place at one of these domaines. To demonstrate this we add oxygen along with surface exchange and a constant oxygen consumption in the sediment to the model. We again start by copying the spread sheet from the package data:

```
ods <- system.file("extdata/examples/simple_alg_O2.ods",
                   package = "rodeoFABM")
file.copy(from = ods, to = ".", recursive = TRUE)
# read in first simple model
vars <- read_ods("simple_alg_O2.ods", sheet = "vars")
pars <- read_ods("simple_alg_O2.ods", sheet = "pars")
funs <- read_ods("simple_alg_O2.ods", sheet = "funs")
pros <- read_ods("simple_alg_O2.ods", sheet = "pros")
stoi <- read_ods("simple_alg_O2.ods", sheet = "stoi")
```

Now we need to add the new state variable $O2$ to the `vars` data frame:

Table 11: Data set `vars`: Declaration of state variables.

| name | unit | description | default |
|---|---|---|---|
| $C$ | gDM/m^3 | algae concentration | 0.0 |
| $HPO4$ | gP/m^3 | phosphorus concentration | 0.1 |

| name | unit | description | default |
|---|---|---|---|
| $O2$ | gO/m^3 | oxygen concentration | 10.0 |

If processes occure only at the surface or bottom interface we can declare this by setting a logical flag in additional columns in the `pros` data frame called *bot* and *surf*. We add two new processes *O2_exch*, and *O2_cons* and set the flag in the corresponding columns to TRUE:

Table 12: Data set 'pros': Declaration of processes.

| name | unit | description | expression | surf | bot | sedi |
|---|---|---|---|---|---|---|
| growth | g/m^3/d | growth of algae | $C \cdot mu\_max \cdot HPO4/(HPO4 + K\_P) \cdot par/(par + K\_par)$ | | | |
| death | g/m^3/d | death of algae | $C \cdot k\_death$ | | | |
| sed | g/m^3/s | sinking | $v\_sed$ | | | TRUE |
| O2_exch | g/m^3/d | exchange of Oxygen at the surface | $v\_O2 \cdot (exp(7.7117 - 1.31403 \cdot log(Temp + 45.93)) \cdot (p/101325) - O2)$ | TRUE | | |
| O2_cons | g/m^3/d | consumption of Oxygen in the pelagial | $O2/(O2 + K\_O2) \cdot k\_O2\_cons$ | | TRUE | |

We need to declare the additional parameter for the oxygen exchange velocity, the constant consumption in the sediment, and the half-saturation concentration of oxygen limiting the oxygen consumption in the sediment:

Table 13: Data set `pars`: Declaration of model parameters.

| name | unit | description | default |
|---|---|---|---|
| *mu_max* | 1/s | maximum growth rate | 1.0e-05 |
| *K_P* | W/m^2 | half saturation of photosyntetic flux | 2.0e-02 |
| *k_death* | 1/s | death rate | 2.0e-06 |
| *a_P* | gP/gDM | phosphorus content of phytoplankton | 5.0e-02 |
| *K_par* | W/m^2 | half saturation of photosyntetic flux | 2.7e+01 |
| *v_sed* | m/s | sedimentation velocity | 1.0e-06 |
| *v_O2* | 1/s | speed of oxygen transfer | 1.0e-05 |
| *k_O2_cons* | 1/s/m^2 | Oxygen consumption rate in sediment | 5.0e-07 |
| *K_O2* | gO/m^3 | half saturation concentration of oxygen consumption | 5.0e+00 |
| *a_O* | gO/gDM | oxygen production per growth of algae | 1.0e+00 |

We need to declare the used functions *log*, and *exp*, as well as the external dependencies $p$ (the barometric pressure at the surface), and $Temp$ (water temperature) which are needed to calculate the oxygen saturation concentration:

Table 14: Data set `funs`: Declaration of model functions and dependencies from the host model.

| name | unit | description | dependency |
|---|---|---|---|
| *par* | W/m^2 | Downwelling photosynthetic radiative flux | downwelling_photosynthetic_radiative_flux |
| *p* | Pa | Atmospheric Pressure | surface_air_pressure |
| *Temp* | celsius | Water temperature | temperature |
| *exp* | - | exponential function | NA |
| *log* | - | logarithmic function | NA |

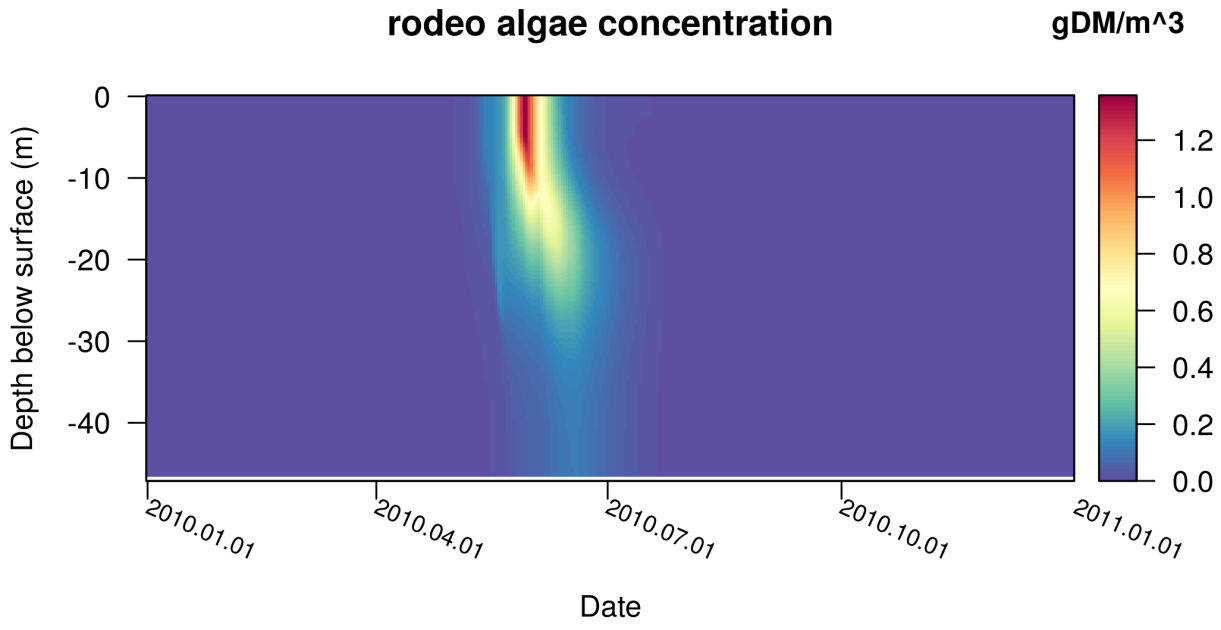
After adding the new processes to the stoichiometry table:

Table 15: Data set `stoi`: Declaration of stoichiometry matrix in long format.

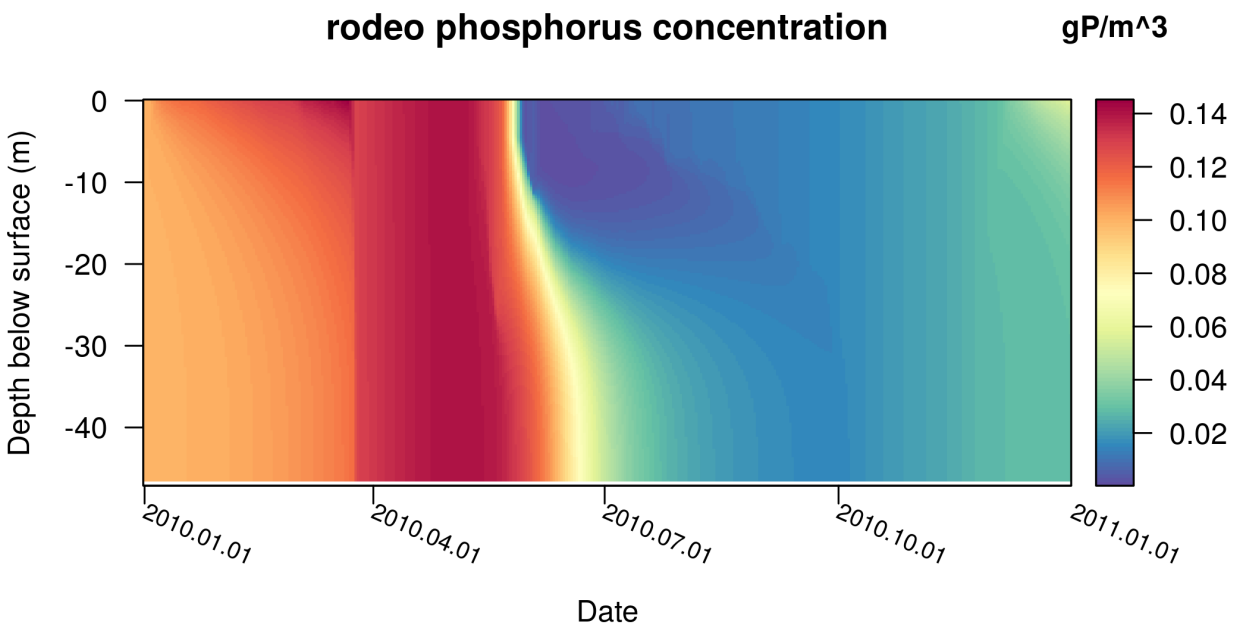
| variable | process | expression |
|---|---|---|
| $C$ | growth | 1 |
| $C$ | death | $-1$ |
| $C$ | sed | $-1$ |
| $HPO4$ | growth | $-1 \cdot a\_P$ |
| $O2$ | O2_exch | 1 |
| $O2$ | O2_cons | $-1$ |
| $O2$ | growth | $a\_O$ |

We can generate the source code, compile *GOTM-FABM*, run the model, and plot some of the results:

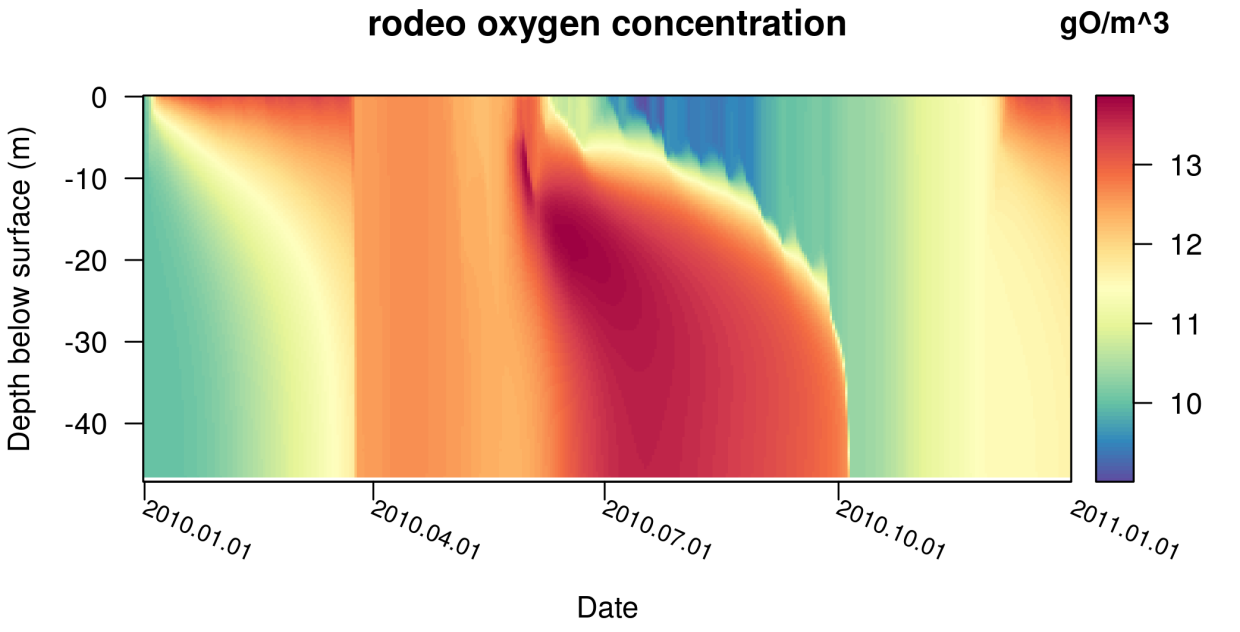
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_4.f90")

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src",
           fabm_file = "model_4.f90")

# run the model
system2("./gotm")

# plot the variables
plot_var("output.nc", "rodeo_C")
```




```
plot_var("output.nc", "rodeo_HPO4")
```




```
plot_var("output.nc", "rodeo_O2")
```




### 3.3.2 Sediment or surface attached state variables

As mentioned before *FABM* recognizes three spatial domains: open water, surface, and sediment. Like Processes, also state variables can be attaced to one of these domaines (e.g. sedimented particulated organic matter). To demonstrate this feature we will include two mroe state variables in our model; particulated

organic matter ($POM$) and sedimented particulated organic matter ($SPOM$). Again we first need to copy the spread sheet from the package:

```
ods <- system.file("extdata/examples/simple_alg_O2_POM.ods",
                   package = "rodeoFABM")
file.copy(from = ods, to = ".", recursive = TRUE)

# read in first simple model
vars <- read_ods("simple_alg_O2_POM.ods", sheet = "vars")
pars <- read_ods("simple_alg_O2_POM.ods", sheet = "pars")
funs <- read_ods("simple_alg_O2_POM.ods", sheet = "funs")
pros <- read_ods("simple_alg_O2_POM.ods", sheet = "pros")
stoi <- read_ods("simple_alg_O2_POM.ods", sheet = "stoi")
```

Then we add the two new state variables $POM$ and $SPOM$. We declare $SPOM$ as bottom bound state variable by adding another column to the `vars` data frame called *bot* and set it to TRUE for all bottom bound state variables and to NA or FALSE for all others. Surface bound state variables can be declared in a column named *surf* in the same manner.

Table 16: Data set `vars`: Declaration of state variables.

| name | unit | description | default | bot |
|---|---|---|---|---|
| $C$ | gDM/m^3 | algae concentration | 0.0 | NA |
| $HPO4$ | gP/m^3 | phosphorus concentration | 0.1 | NA |
| $O2$ | gO/m^3 | oxygen concentration | 10.0 | NA |
| $POM$ | gDM/m^3 | particulated organic matter | 0.0 | NA |
| $SPOM$ | gDM/m^2 | sedimented particulated organic matter | 0.0 | TRUE |

The death of algae generates $POM$, which settles down, and sediments to the ground to become $SPOM$. Both $POM$ and $SPOM$ are mineralized, releasing $HPO4$ but the mineralization is faster in the sediment. We add the new sinking, sedimentation, and mineralization processes to the `pros` data frame:

Table 17: Data set 'pros': Declaration of processes.

| name | unit | description | expression | surf | bot | sedi |
|---|---|---|---|---|---|---|
| growth | gDW/m^3/d | growth of algae | $C \cdot mu\_max \cdot (HPO4)/(HPO4 + K\_P) \cdot (par)/(par + K\_par)$ | | | |
| death | gDW/m^3/d | death of algae | $C \cdot k\_death$ | | | |
| sed_ALG | gDW/m^3/s | sinking of algae | $v\_sed\_ALG$ | | | TRUE |
| O2_exch | gO/m^3/d | exchange of Oxygen at the surface | $v\_O2 \cdot (exp(7.7117 - 1.31403 \cdot log(Temp + 45.93)) \cdot (p)/(101325) - O2)$ | TRUE | | |
| O2_cons | gO/m^3/d | consumption of Oxygen in the pelagial | $(O2)/(O2 + K\_O2) \cdot k\_O2\_cons$ | | TRUE | |
| sed_POM | gDW/m^3/s | sinking of POM | $v\_sed\_POM$ | | | TRUE |
| miner_POM | gDW/m^3/s | mineralization of POM | $POM \cdot k\_miner\_POM \cdot (O2)/(O2 + K\_miner\_O2)$ | | | |
| miner_SPOM | gDW/m^3/s | mineralization of SPOM | $SPOM \cdot k\_miner\_SPOM \cdot (O2)/(O2 + K\_miner\_O2)$ | | TRUE | |
| set_POM | gDW/m^3/s | settling of POM | $v\_sed\_POM \cdot POM$ | | TRUE | |

Then we need to adapt the stoichiometry table:

Table 18: Data set `stoi`: Declaration of stoichiometry matrix in long format.

| variable | process | expression |
|---|---|---|
| $C$ | growth | 1 |
| $C$ | death | $-1$ |
| $C$ | sed_ALG | $-1$ |
| $HPO4$ | growth | $-1 \cdot a\_P$ |
| $HPO4$ | miner_POM | $a\_P$ |
| $HPO4$ | miner_SPOM | $a\_P$ |
| $O2$ | O2_exch | 1 |
| $O2$ | O2_cons | $-1$ |

| variable | process | expression |
|---|---|---|
| $O2$ | growth | $a\_O$ |
| $O2$ | miner_POM | $-1 \cdot a\_miner$ |
| $O2$ | miner_SPOM | $-1 \cdot a\_miner$ |
| $POM$ | sed_POM | $-1$ |
| $POM$ | set_POM | $-1$ |
| $POM$ | death | 1 |
| $POM$ | miner_POM | $-1$ |
| $SPOM$ | set_POM | 1 |
| $SPOM$ | miner_SPOM | $-1$ |

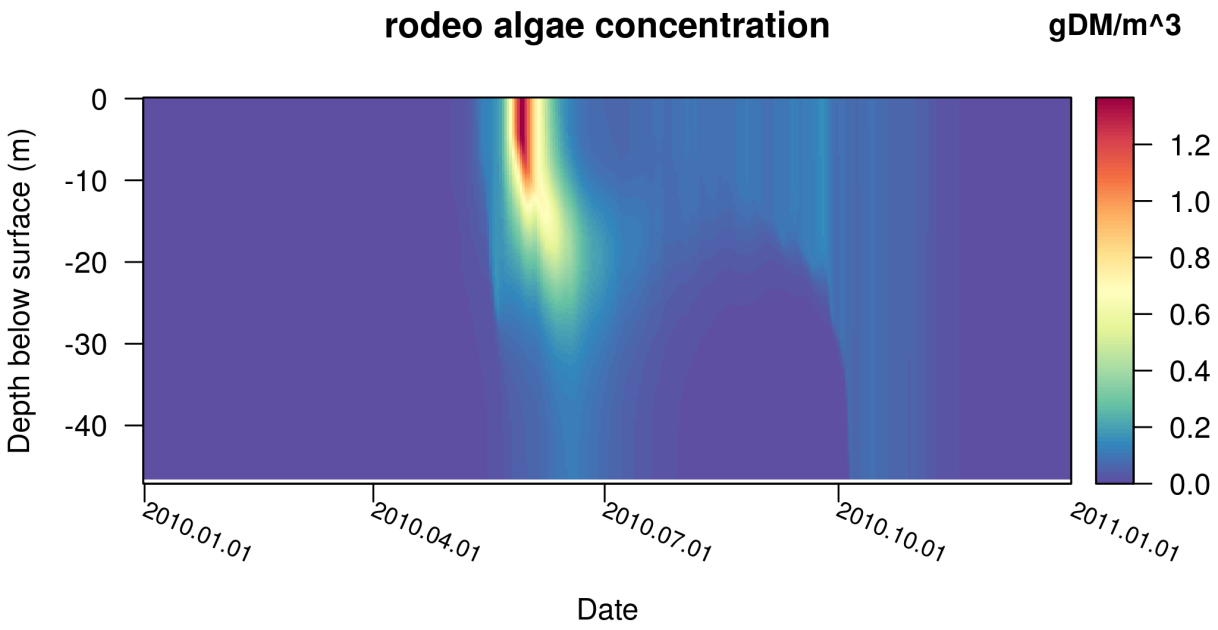
And declare the new parameters for the sinking velocity, the mineralization kinetic, and the half-saturation concentration limiting the mineralization:

Table 19: Data set `pars`: Declaration of model parameters.

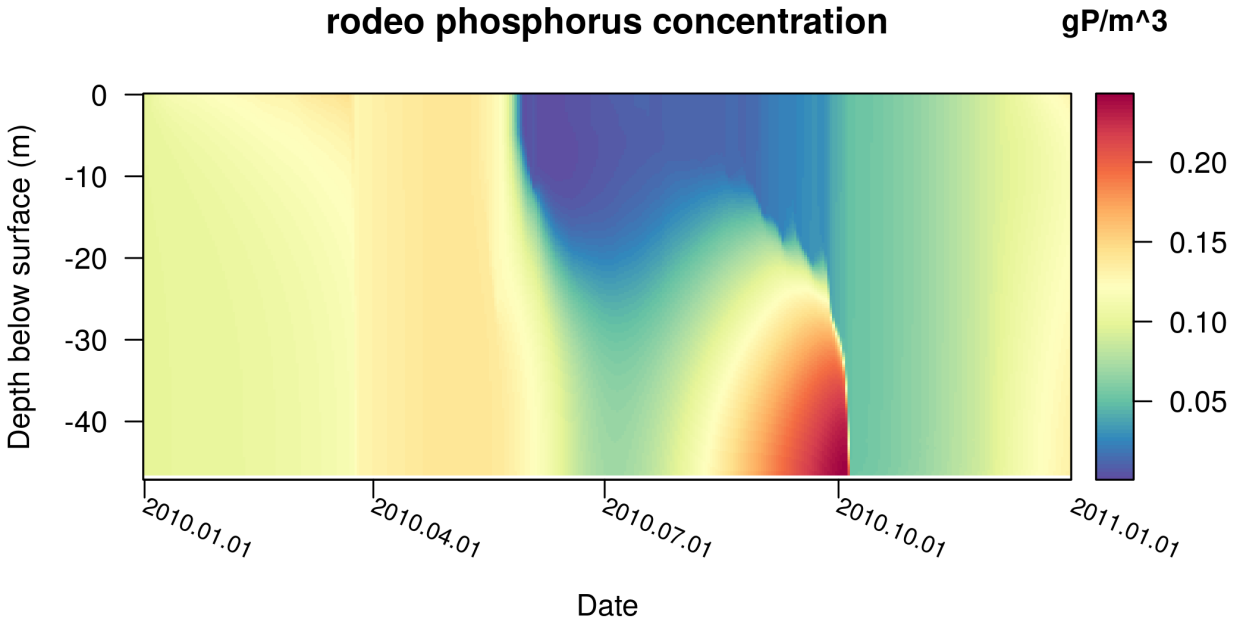
| name | unit | description | default |
|---|---|---|---|
| $mu\_max$ | 1/s | maximum growth rate | 1.0e-05 |
| $K\_P$ | W/m^2 | half saturation of photosyntetic flux | 2.0e-02 |
| $k\_death$ | 1/s | death rate | 2.0e-06 |
| $a\_P$ | gP/gDM | phosphorus content of phytoplankton | 5.0e-02 |
| $K\_par$ | W/m^2 | half saturation of photosyntetic flux | 2.7e+01 |
| $v\_sed\_ALG$ | m/s | sedimentation velocity of algae | 1.0e-06 |
| $v\_O2$ | 1/s | speed of oxygen transfer | 1.0e-04 |
| $k\_O2\_cons$ | 1/s/m^2 | Oxygen consumption rate in sediment | 5.0e-07 |
| $K\_O2$ | gO/m^3 | half saturation concentration of oxygen consumption | 5.0e+00 |
| $a\_O$ | gO/gDM | oxygen production per growth of algae | 1.0e+00 |
| $v\_sed\_POM$ | m/s | sedimentation velocity of POM | 2.0e-06 |
| $K\_miner\_O2$ | gO/m^3 | half saturation concentration of oxygen for mineralization | 3.0e+00 |
| $k\_miner\_POM$ | 1/s | maximum mineralization rate of POM | 0.0e+00 |
| $k\_miner\_SPOM$ | 1/s | maximum mineralization rate of SPOM | 3.0e-07 |
| $a\_miner$ | gO/gDM | oxygen consumption per oxygenation of POM/SPOM | 1.0e+00 |

Now we can create fortran source code, compile *GOTM-FABM*, run the model, and plot the results:

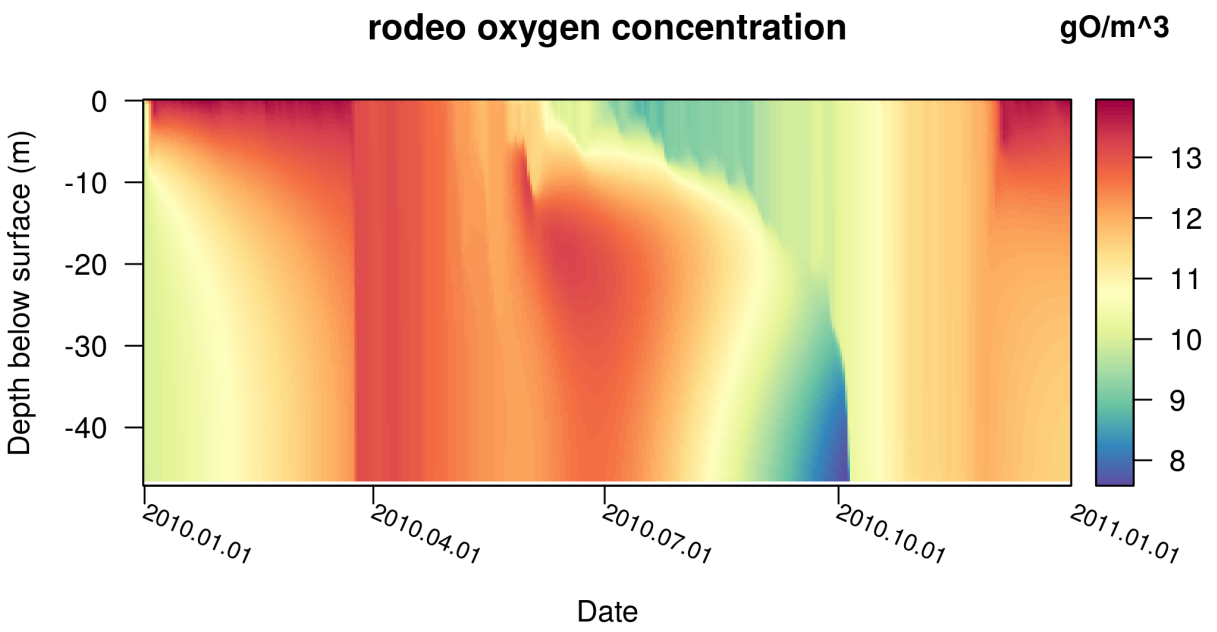
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_5.f90")

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src",
           fabm_file = "model_5.f90")

# run the model
system2("./gotm")

# plot the variables
plot_var("output.nc", "rodeo_C")
```



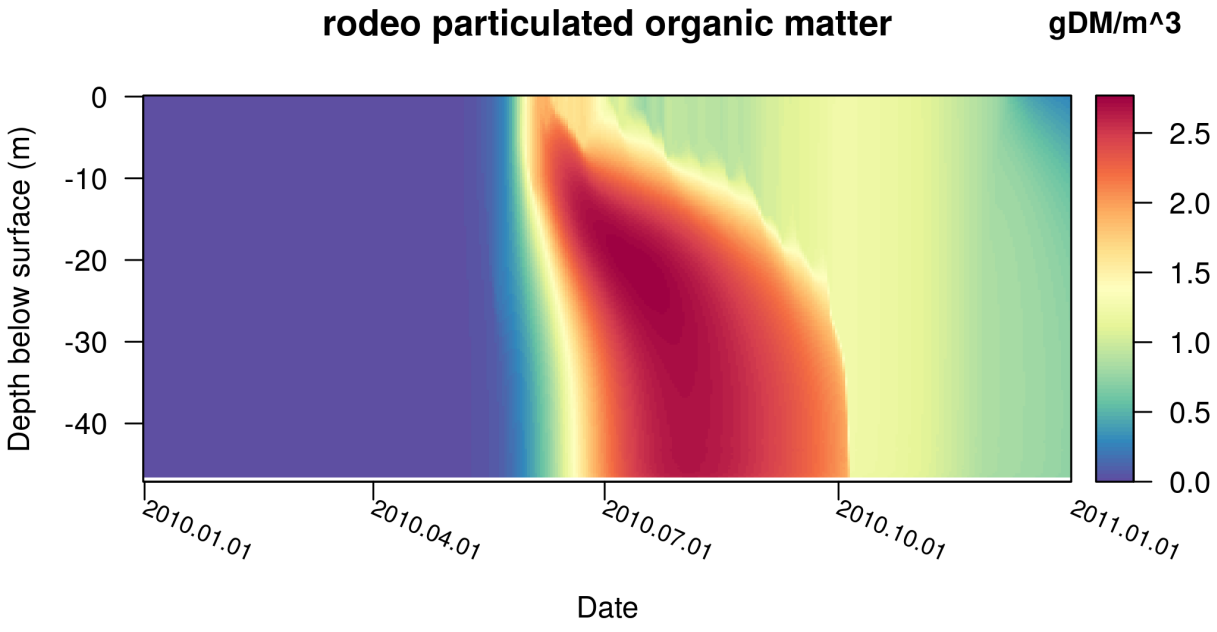

```
plot_var("output.nc", "rodeo_HPO4")
```



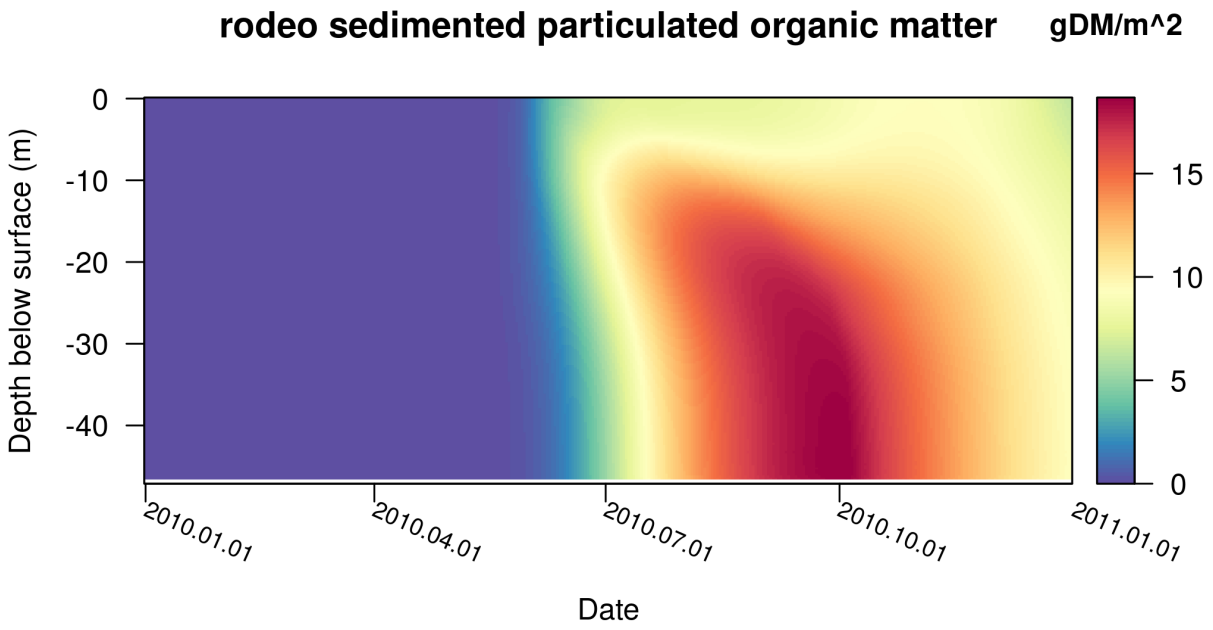

```
plot_var("output.nc", "rodeo_O2")
```




```
plot_var("output.nc", "rodeo_POM")
```




```
plot_var("output.nc", "rodeo_SPOM")
```




## 3.4 Additional features

### 3.4.1 Additional state variable arguments

There are a few additional arguments for state variables that can be defined in *FABM*. In order to use them a new column in the state variable data frame needs to be added with exactly the name.

- *minimum*: minnimum allowed value for the state variable
- *maximum*: maximum allowed value for the state variable
- *specific_light_extinction*: specific light extinction coefficient of this variable
- *no_precipitation_dilution*: the variable is not diluted by precipitation
- *no_river_dilution*: the variable is not diluted by river inflows

### 3.4.2 Profile initial values for the state variables

By default the initial values for the *FABM* sate variables are constant throughout the whole profile. With some tinkering we can profide profiles of initial values. Therefore, you need to have (a) text file(s) with (the) profiles that you want to initialise and install the R packages `ncdf4`, `data.table`, `gotmtools` from the AEMON-J github. The approach is to first run GOTM with 0 time steps (stop date = start date). After a run with GOTM, a `restart.nc` file is created, that can be use to restart a simulation with the same settings that ended the previous simulation. By running with 0 time steps, this file contains the "standard" initial values, including the ones in your biogeochemical model. Then the you need to replaces the homogeneous initial profiles by your specified profiles in the `restart.nc` file. Lastly, you need to set the `restart` option to "true". If you run gotm now, it will run with the initial profiles for you biogeochemical model (note that you'll have to rerun this approach every time before you want to run GOTM, because every new GOTM run overwrites the `restart.nc` file). Alternatively you can save the created `restart.nc` file e.g. as `restart_init_profiles.nc` and use this file to override the `restart.nc` before you run GOTM.

### 3.4.3 Defining own functions

#### 3.4.3.1 Defining functions in the *funs* data.frame

It is possible to define own functions and call them. If the function is simple i.e. can be calculated in one line of code (e.g. limiting functions for algae growth) it can be defined from the *expressions* column of the *pros* data.frame. In order to define own functions two additional columns are needed in the *funs* data.frame: `expression` and `arguments`. Similar to the *pros* data.frame the `expression` column gives the mathematical expression to calculate. The `arguments` column gives all input arguments that the functions uses, in the same order they are supplied during function calls (e.g. in the *expression* column of the *pros* data.frame), separated by commas (","). 

#### 3.4.3.2 Defining functions in external fortran files

More complex functions can be supplied as external fortran code. Two additional columns are needed in the *funs* data.frame: `file` and `module`. They give the name of the source code file and the name of the module, which is then loaded in the main source code. This feature is still experimental and might lead to errors!

### 3.4.4 Automatic model documentation

If wanted `rodeoFABM` can automatically generate LaTeX documentation of the state variables, parameter, processes and stoichometry. To do so the function `document_model()` can be used. Lets create a documentation of the final phytoplankton nutrients model from our example. In order to work we need an additional column named *tex* (you can also use another name for this column and supply the name to `document_model()` using the `tex` argument) in the data frames *vars*, *pars*, *funs*, and *pros* giving the corresponding LaTeX symbols to be used. The documentation function automatically generates LaTeX fraction, but in order for this to work all used fractions in the *expression* column of the *pros* data frame need to be in a specified format. The numerator and denominator need to be in brakets, even if they are just one single variable, number, or parameter: e.g. `(O2)/(O2 + K_O2)`. In the example spread sheet file they are allready added:

```
# see column "tex"
head(vars)
```

```
##   name    unit                          description default  bot   tex
## 1    C gDM/m^3                   algae concentration     0.0   NA     C
## 2 HPO4  gP/m^3              phosphorus concentration     0.1   NA HPO_4
## 3   O2  gO/m^3                  oxygen concentration    10.0   NA   O_2
## 4  POM gDM/m^3           particulated organic matter     0.0   NA   POM
## 5 SPOM gDM/m^2 sedimented particulated organic matter     0.0 TRUE  SPOM
```

```
# create LaTeX documentation for our model
document_model(vars, pars, pros, funs, stoi, landscape = FALSE)
```

```
##
##  finished
```

```
## [1] TRUE
```

We can see that now there are seven additional file in our worling directory:

```
grep(".*\\.tex", list.files(), value = TRUE)
```

```
## [1] "document_model.tex" "preamble-latex.tex" "pros_expr.tex"
## [4] "tab_funs.tex"       "tab_pars.tex"       "tab_pros.tex"
## [7] "tab_stoi.tex"       "tab_vars.tex"
```

They are LaTeX tables of the models state variables (*tab_vars.tex*), used model parameters (*tab_pars.tex*), used functions (*tab_funs.tex*), declaration of the models processes (*tab_pros.tex*), description of the process

equations (*pros_expr.tex*), the stoichiometry table (*tab_stoi.tex*), and a simple latex document that can be used to compile all of the before (*document_model.tex*).

The created expressions of the processes now look like this:

```
head(readLines("pros_expr.tex"))
```

```
## [1] "\\begin{align}"
## [2] " \\rho_{growth} =&~ C \\cdot \\mu_{max} \\cdot \\frac{HPO_4 }{ HPO_4 + K_P} \\cdot \\frac{par }{
## [3] " \\rho_{death} =&~ C \\cdot k_{death}\\\\"
## [4] " \\rho_{Sed,ALG} =&~ v_{sed,ALG}\\\\"
## [5] " \\rho_{O2,exch} =&~ v_{exch,O2} \\cdot \\left( \\exp \\left( 7.7117 - 1.31403 \\cdot \\log \\le
## [6] " \\rho_{O2,cons} =&~ \\frac{O_2 }{ O_2 + K_{O2}} \\cdot k_{O2,cons}\\\\"
```

and compiled they look like this:

$$\rho_{growth} = C \cdot \mu_{max} \cdot \frac{HPO_4}{HPO_4 + K_P} \cdot \frac{par}{par + K_{par}} \tag{1}$$

$$\rho_{death} = C \cdot k_{death} \tag{2}$$

$$\rho_{Sed,ALG} = v_{sed,ALG} \tag{3}$$

$$\rho_{O2,exch} = v_{exch,O2} \cdot \left(\exp\left(7.7117 - 1.31403 \cdot \log\left(\vartheta_z + 45.93\right)\right) \cdot \frac{p}{101325} - O_2\right) \tag{4}$$

$$\rho_{O2,cons} = \frac{O_2}{O_2 + K_{O2}} \cdot k_{O2,cons} \tag{5}$$

$$\rho_{Sed,POM} = v_{sed,POM} \tag{6}$$

$$\rho_{Miner,POM} = POM \cdot k_{miner,POM} \cdot \frac{O_2}{O_2 + K_{miner,O2}} \tag{7}$$

$$\rho_{Miner,SPOM} = SPOM \cdot k_{miner,SPOM} \cdot \frac{O_2}{O_2 + K_{miner,O2}} \tag{8}$$

$$\rho_{Set,POM} = v_{sed,POM} \cdot POM \tag{9}$$

# References


Bruggeman, Jorn, and Karsten Bolding. 2014. "A General Framework for Aquatic Biogeochemical Models." *Environmental Modelling & Software* 61 (November): 249–65. https://doi.org/10.1016/j.envsoft.2014.04.002.

Burchard, Hans, Karsten Bolding, Wilfried Kühn, Andreas Meister, Thomas Neumann, and Lars Umlauf. 2006. "Description of a Flexible and Extendable Physical–Biogeochemical Model System for the Water Column." *Journal of Marine Systems* 61 (3): 180–211. https://doi.org/https://doi.org/10.1016/j.jmarsys.2005.04.011.

Kneis, David, Thomas Petzoldt, and Thomas U. Berendonk. 2017. "An R-Package to Boost Fitness and Life Expectancy of Environmental Models." *Environmental Modelling & Software* 96 (October): 123–27. https://doi.org/10.1016/j.envsoft.2017.06.036.