R package rodeoFABM: Basic Use and Sample Applications

johannes.feldbauer@tu-dresden.de

2020-03-18

Contents

1 Main features of `rodeoFABM`

The package `rodeoFABM` is a colletion of small tools 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 have a system that:

- Helps users that don't have the technical know how
- make model adaptation, communikation, and maintanace easy

Therfore the water quality model is written in the standard Peterson matrix notation and stored in text files or spread sheets. The package `rodeoFABM` automatically generates `FABM` specific FORTRAN code from these files and can automatically compile `GOTM` coupled with the newly created model, as well as .yaml control files for the water quality model.

2 Installation and requirements

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

- The GNU compilers
- GNU Make

- GNU CMake
- Rdevtools
- R packages: `readODs`, `gotmtools`

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

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

3 Basic use

A simple example that is extended along the way

3.1 first example (how it works)

To demonstrate the workflow we will use a very simple model that is provided in the package. The files are contained in the package and can be loaded using:

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

Now we can read in the tables with the declaration of the state variables, model parameters, used functions and external dependencies, process rate descriptions, and the stoichiometry matrix.

```
library(readODS)

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

From these we can now generate FORTRAN files

```
library(rodeoFABM)

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

And compile GOTM-FABM with them. Therfore first we need to clone the lake branche of GOTM-FABM from github and prepare the build process using cmake. This needs only to be done once, using the function `clone_GOTM()`:

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

Now we can build GOTM-FABM with our own model using:

```
# build GOTM
build_GOTM(build_dir = "build",fabm_file = "simple_model.f90",
           src_dir = "gotm_src/extern/fabm/src/models/tuddhyb/rodeo")
```

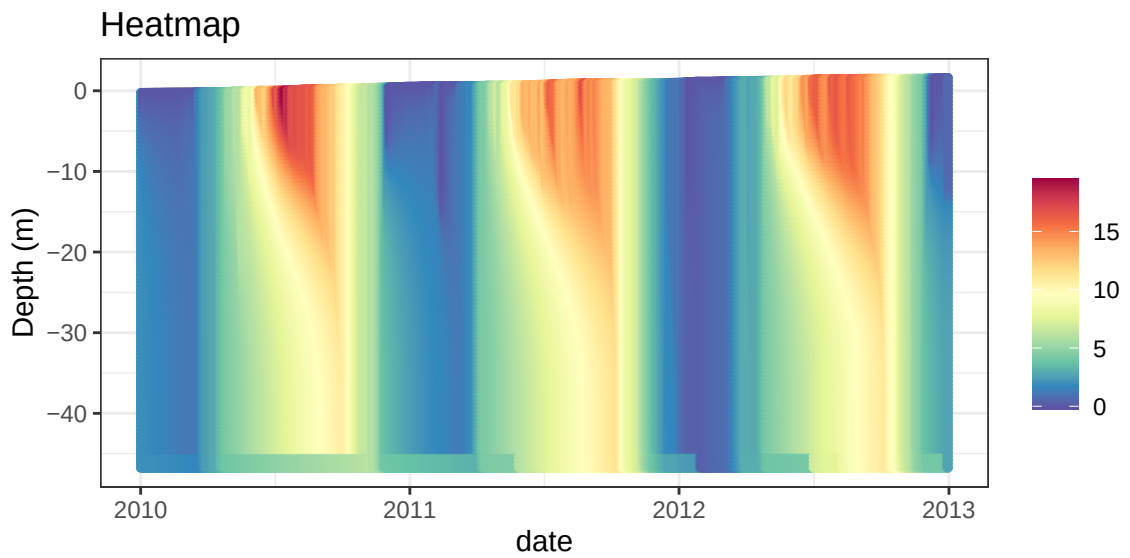
After copying the example gotm.yaml (the GOTM controll fille), the exymple hypsograph, and the example forcing data, we finally can run GOTM-FABM with our own small model using:

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

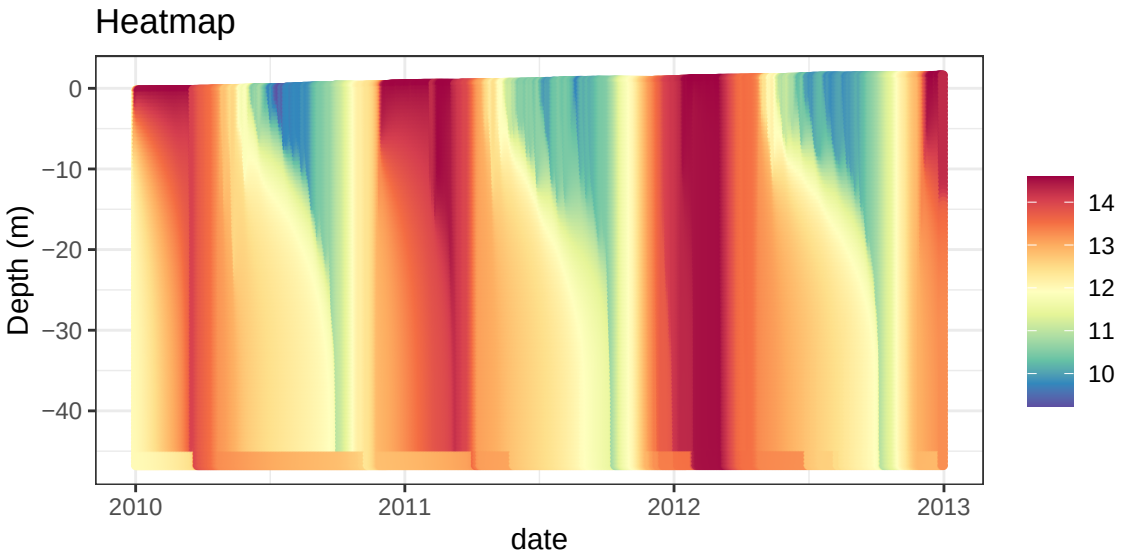
We can plot the results e.g. using the `gotmtools` library, which can be installed from the AEMON-J github using:

```
devtools:::install_github("aemon-j/gotmtools")
```

```
library(gotmtools)
# plot temperature
plot_vari("output.nc", "temp")
```




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



These are the essential steps used. In the next section we will go into the details of building a model using the `rodeoFABM` package step by step.

3.2 create own model

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

The libre office spread sheets with the model information, GOTM controll file, and forcing data are contained in the `rodeoFABM` package. You can copy all necessary files to run GOTM useing the same method as before. We will use the same meteorological forcing (`meteo_file.dat`) and hypsographic curve (`hypsograph.dat`) as in the first example. Additionally we now have 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 `stream` 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 creat the firs phytoplankton nutients model similar to how we created the first example:

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

# read in first simple model
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 very 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/m^3	algae concentration	0.0
$HPO4$	gP/m^3	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/m^2	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).

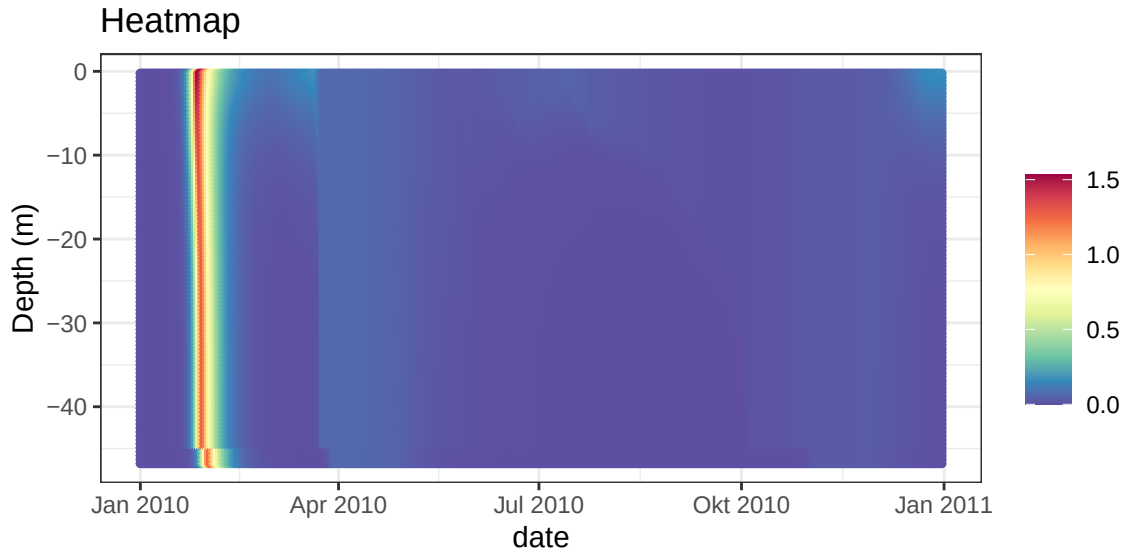
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_1.f90")
```

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

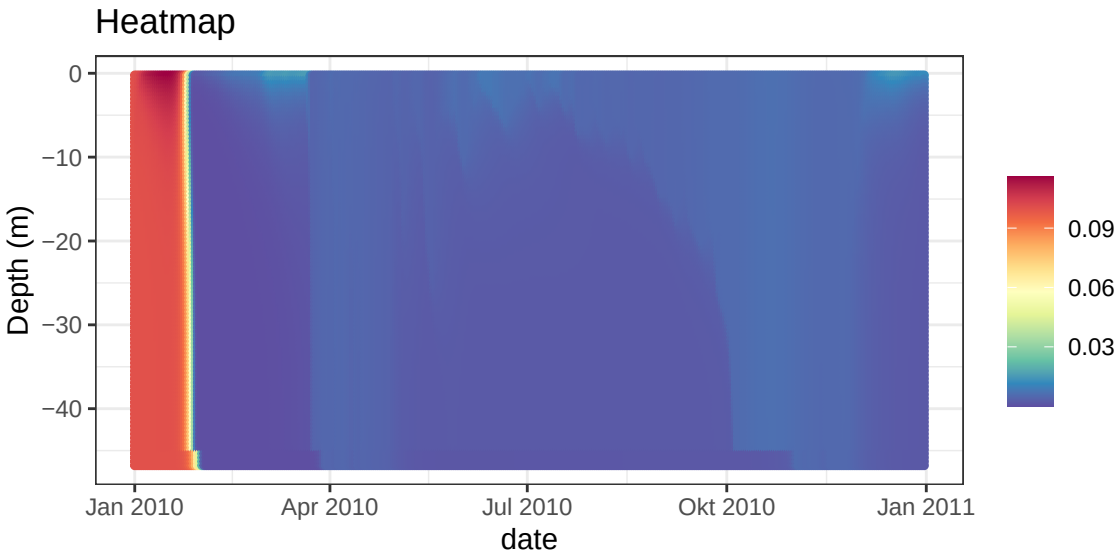
```
# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src/extern/fabm/src/models/tuddhyb/rodeo/",
           fabm_file = "model_1.f90")

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

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

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




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



3.2.1 Getting forcing data from the host model

In the next step we want to add the dependency of growth on available iradiation. Therefore we need to get the photosynthetic active radiation (PAR) from GOTM.

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

# read in first simple model
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")
```

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* is requred when you want to access forcing data from the coupled physical host model.

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

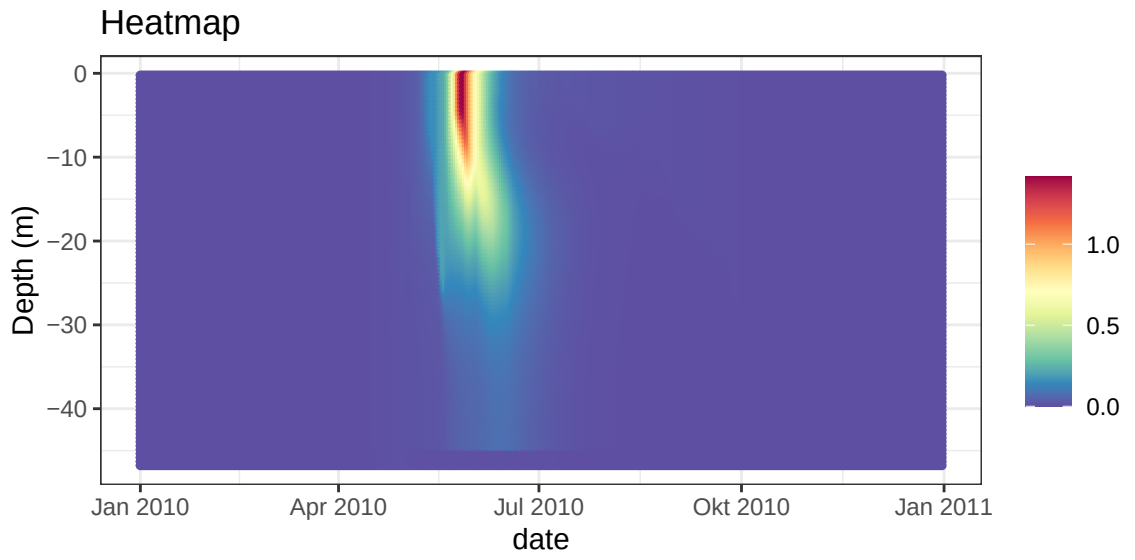
Table 6: Data set `pros`: Declaration of processes.

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

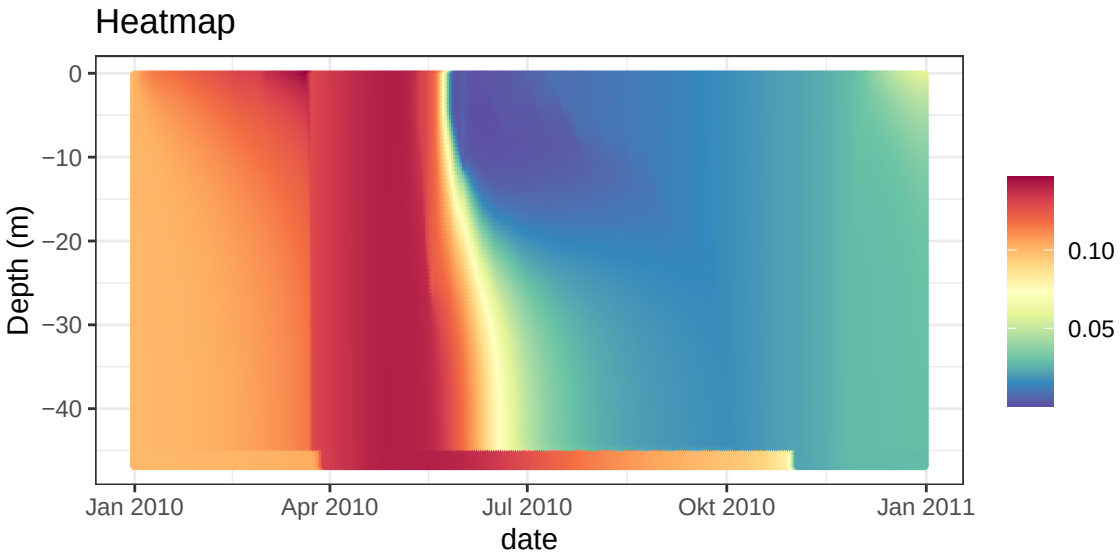
name	unit	description	expression
death	g/m^3/d	death of algae	$C \cdot k\_death$

Now we can run the adapted model.

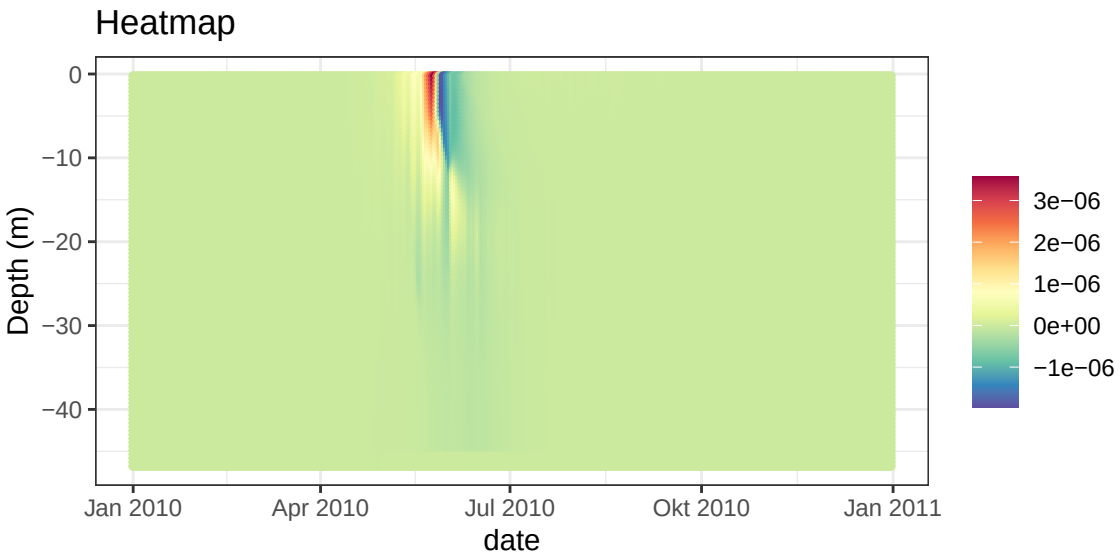
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_2.f90")

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src/extern/fabm/src/models/tuddhyb/rodeo/",
           fabm_file = "model_2.f90")

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

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




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




```
# also plot net. growth
growth <- get_vari("output.nc", "rodeo_growth")
death <- get_vari("output.nc", "rodeo_death")
net_growth <- cbind(growth$Datetime, growth[, -1] - death[, -1])
z  <-  get_vari("output.nc", var = "z", incl_time = TRUE)
long_heatmap(wide2long(net_growth, z))
```



3.3 sedimentation

algae are settling. copy files from package

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

# read in first simple model
```

```
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")
```

we add a process `sed` and declare the pocess as settling process by using the additional column `sedi` in the `pros` data frame.

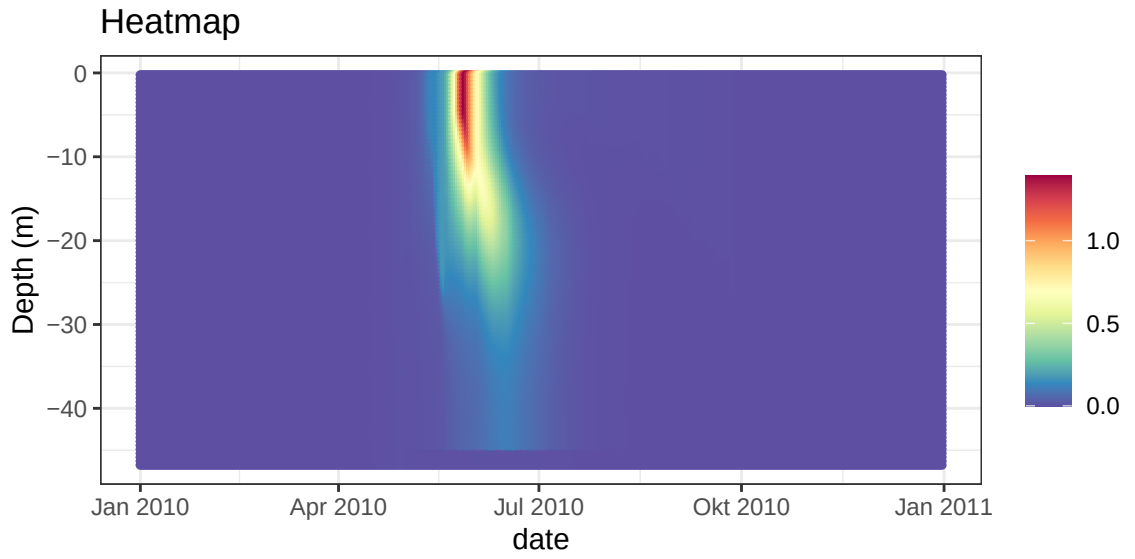
Table 7: Data set `pros`: Declaration of processes.

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

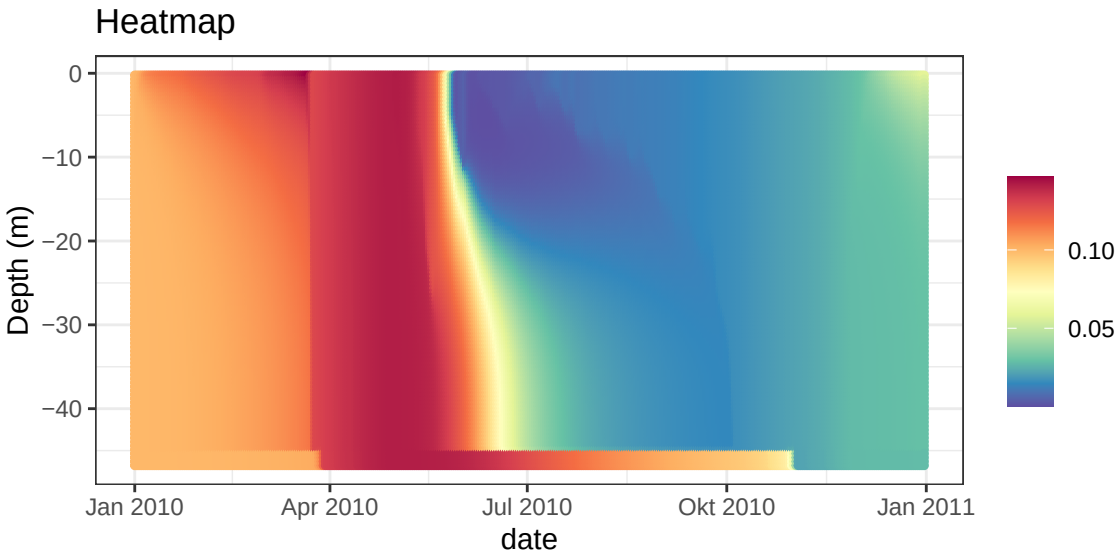
Add to stopicheometry:

compile, run model and plot results

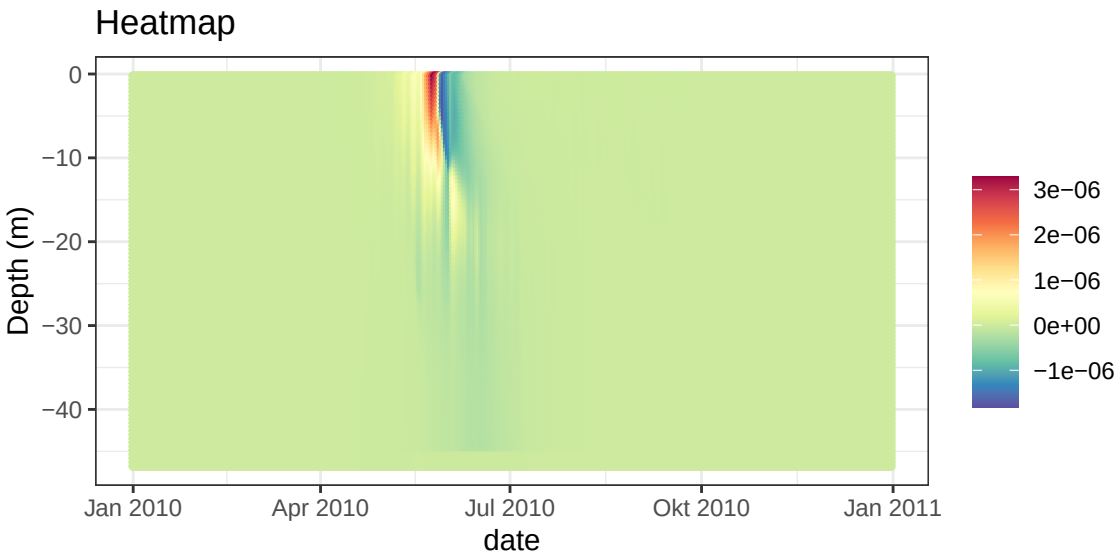
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_3.f90")

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src/extern/fabm/src/models/tuddhyb/rodeo/",
           fabm_file = "model_3.f90")

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

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




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




```
# also plot net. growth
growth <- get_vari("output.nc", "rodeo_growth")
death <- get_vari("output.nc", "rodeo_death")
net_growth <- cbind(growth$Datetime, growth[, -1] - death[, -1])
z  <-  get_vari("output.nc", var = "z", incl_time = TRUE)
long_heatmap(wide2long(net_growth, z))
```



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

Add oxygen along with surface exchange and a constant oxygen consumption

```
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")
```

Add new state variable `O2` oxygen consumption

Table 8: 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
$O2$	gO/m^3	oxygen concentration	10.0

Add new processes with exchange at the surface and consumption in the sediment:

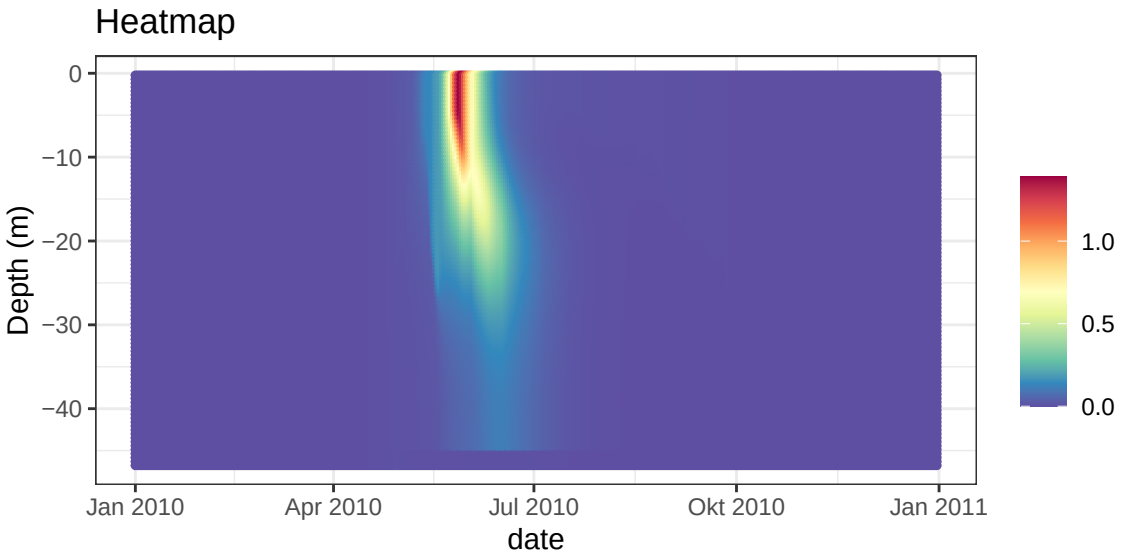
Table 9: Data set `pros`: Declaration of processes.

name	unit	description	expression
growth	g/m^3/d	growth of algae	$C \cdot mu\_max \cdot HPO4/(HPO4 + K\_P) \cdot par/(par + K\_pa$
death	g/m^3/d	death of algae	$C \cdot k\_death$
sed	g/m^3/s	sinking	v_sed
O2_exch	g/m^3/d	exchange of Oxygen at the surface	$v\_O2 \cdot (exp(7.7117 - 1.31403 \cdot log(Temp + 45.93)) \cdot (p/101$
O2_cons	g/m^3/d	consumption of Oxygen in the pelagial	$O2/(O2 + K\_O2) \cdot k\_O2\_cons$

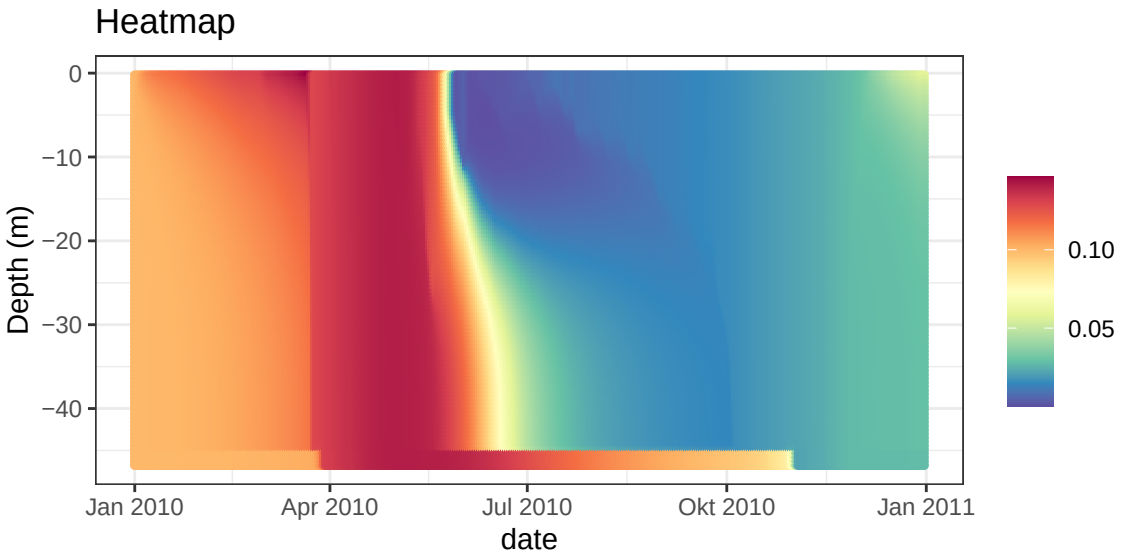
Add to stoicheometry:

And compile, run model, and plot results:

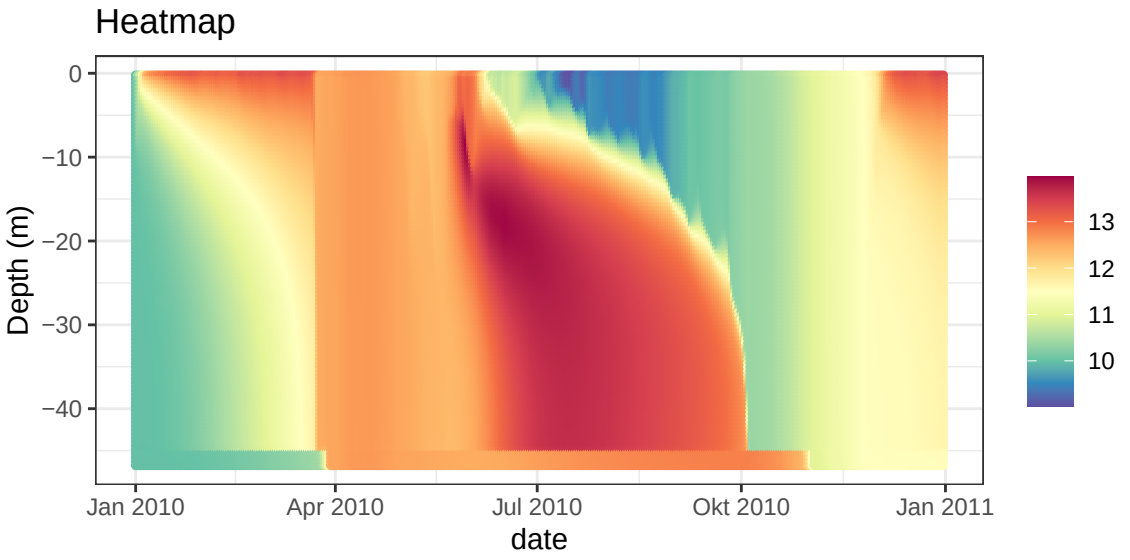
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_4.f90")

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src/extern/fabm/src/models/tuddhyb/rodeo/",
           fabm_file = "model_4.f90")

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

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




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




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



3.3.2 Sediment or Surface attached state variables

death of algae generates POM which sediments faster and can become SPOM and be mineralized at the sediment faster

```
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")
```

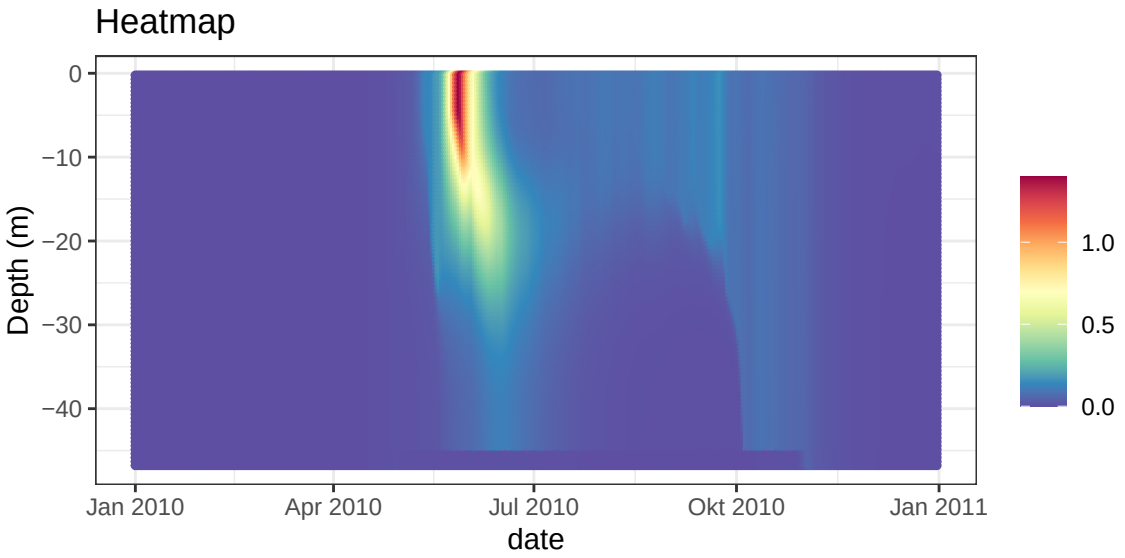
Add two new state variables `POM` and `SPOM`, declare `SPOM` as bottom bound state variable:

Add new process:

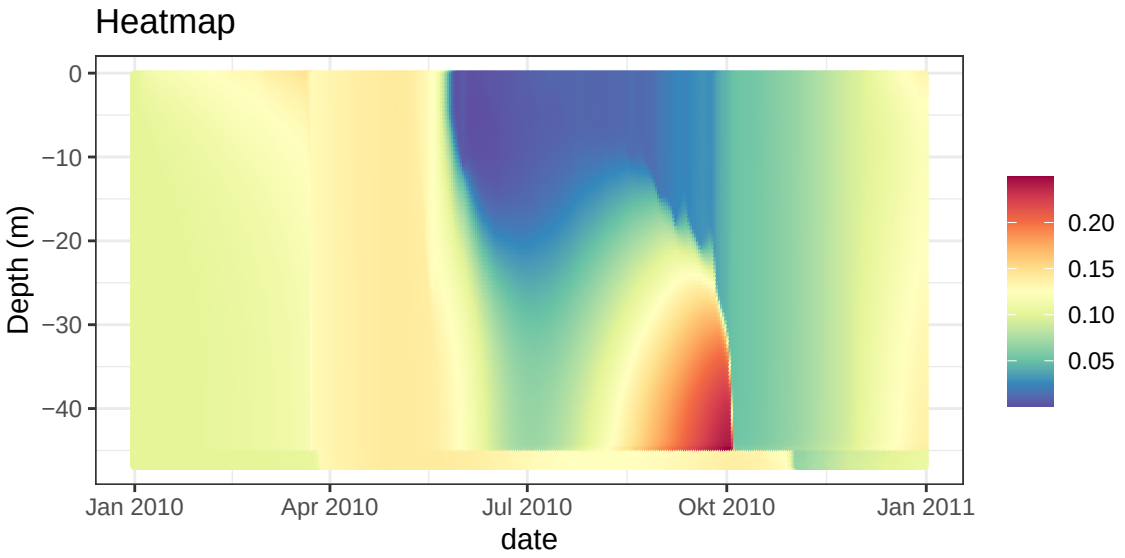
Change Stoicheometry:

compile, run model, and plot results:

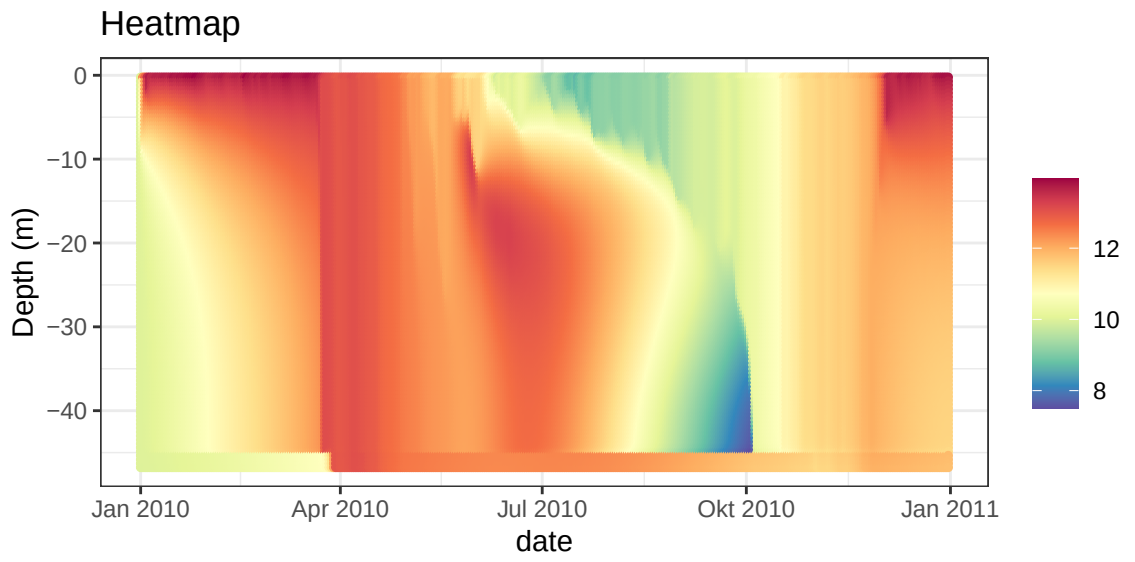
```
# create the fabm code
gen_fabm_code(vars, pars, funs, pros, stoi, "model_5.f90")

# build GOTM with the model
build_GOTM(build_dir = "build", src_dir = "gotm_src/extern/fabm/src/models/tuddhyb/rodeo/",
           fabm_file = "model_5.f90")

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

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



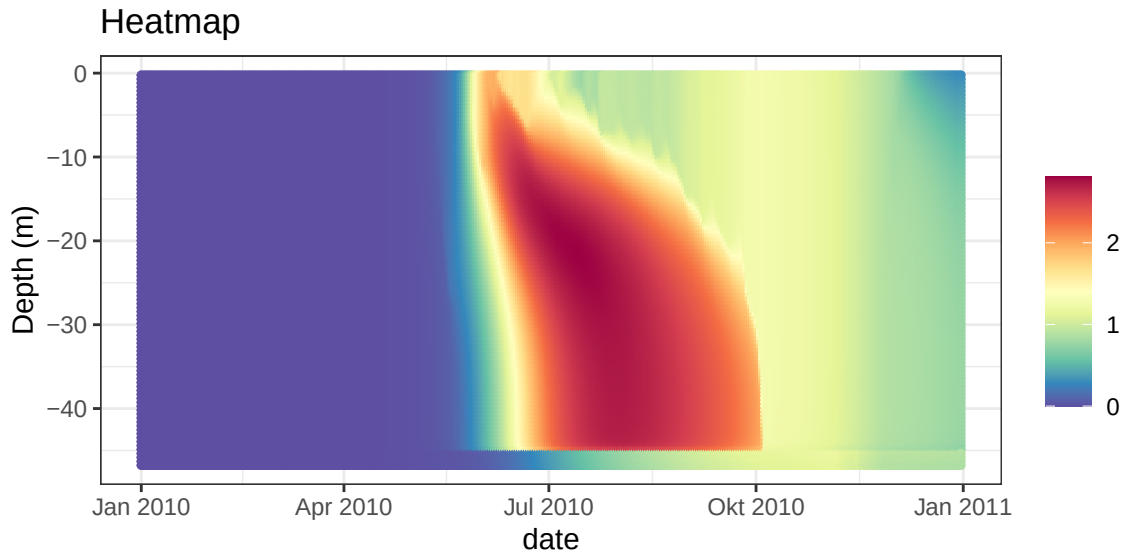

```
plot_vari("output.nc", "rodeo_HPO4")
```



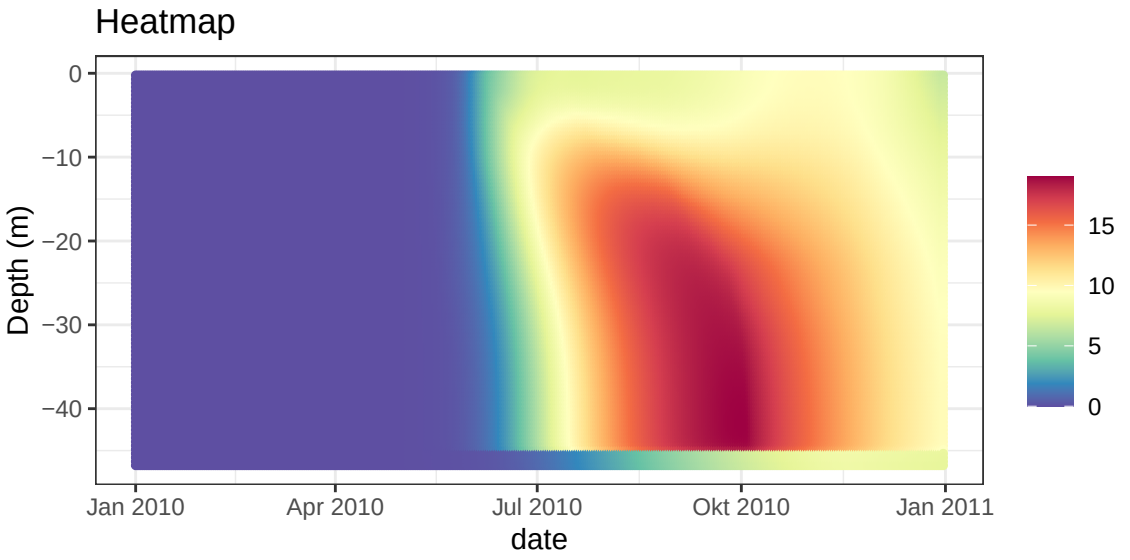

```
plot_vari("output.nc", "rodeo_O2")
```




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




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



3.4 Additional features

3.4.1 Additional state variable argumenhts

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 automatic model documentation

if wanted `rodeoFABM` can automatically generate LaTeX documentation of the state variables, parameter, processes and stoichometry. still under developement!

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.

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.