Fitting a 1D Reaction Transport Model to Data in R

Reader Accompanying the Course Reaction Transport Modeling in the Hydrosphere

Karline Soetaert and Lubos Polerecky

08–April–2021

Abstract

Here we show how to fit observed data with a 1D reaction-transport model. We use the phosphorus diagenesis model described in another Reader as an example. The fitting methods are quite advanced, and include the use of a Markov Chain Monte Carlo (MCMC) method to estimate the probability distribution of the model parameters, and a sensitivity analysis to quantify the uncertainty surrounding the modelled variables. This Reader only describes how these methods are *applied*. The mathematical and numerical details are described in Soetaert and Petzoldt (2010).

The model

First, we implement the 1D reaction-transport model of phosphorus diagenesis in sediments. Details of this model are presented in a different Reader (*Pdynamics*).

```
require(deSolve)
require(marelac)
require(ReacTran)
require(rootSolve)

# units: distance=m, concentrations=mol/m3, time=d

Length <- 2.0   # [m]
N      <- 200
Grid   <- setup.grid.1D(L = Length, N = N, dx.1 = 0.1e-2)

# porosity and solid volume fraction
porFun.L <- function(x, por.SWI, por.deep, porcoef)
  return(por.deep + (por.SWI-por.deep)*exp(-x*porcoef))
```

```
porFun.S <- function(x, por.SWI, por.deep, porcoef)
  return(1-porFun.L(x, por.SWI, por.deep, porcoef))

porLiquid <- setup.prop.1D(func = porFun.L, grid = Grid,
  por.SWI = 0.9, por.deep = 0.6, porcoef = 50)

porSolid <- setup.prop.1D(func = porFun.S, grid = Grid,
  por.SWI = 0.9, por.deep = 0.6, porcoef = 50)

# Sediment diffusion coefficient (m2/d)
diffHCO3  <- diffcoeff(S=35, t=20)$HCO3  * 3600*24
diffH2PO4 <- diffcoeff(S=35, t=20)$H2PO4 * 3600*24
porInt    <- porLiquid$int
diffDIC   <- diffHCO3  / (1-log(porInt^2))
diffPO4   <- diffH2PO4 / (1-log(porInt^2))
```

```
# model function
PDiamodel <- function (t, Conc, pars)
{
  with (as.list(pars),{

    POC  <- Conc[     1 :   N ]
    DIC  <- Conc[(  N+1):(2*N)]
    PO4  <- Conc[(2*N+1):(3*N)]
    Pads <- Conc[(3*N+1):(4*N)]

    tran.POC <- tran.1D(C = POC, flux.up = depoPOC,
                        dx = Grid, VF = porSolid,
                        D = biot, v = v)
    tran.Pads <- tran.1D(C = Pads, flux.up = 0,
                        dx = Grid, VF = porSolid,
                        D = biot, v = v)
    tran.DIC <- tran.1D(C = DIC, C.up = bwDIC,
                        dx = Grid, VF = porLiquid,
                        D = diffDIC, v = v)
    tran.PO4 <- tran.1D(C = PO4, C.up = bwPO4,
                        dx = Grid, VF = porLiquid,
                        D = diffPO4, v = v)

    Mineralisation <- rMin  * POC
    Adsorption     <- rPads * PO4
    Desorption     <- rPdes * Pads

    poro    <- porLiquid$mid
```

```
    dPOC.dt   <- tran.POC$dC - Mineralisation
    dPads.dt  <- ( tran.Pads$dC
                   + Adsorption*poro/(1-poro)
                   - Desorption )

    dDIC.dt <- ( tran.DIC$dC +
                 Mineralisation * (1-poro)/poro )
    dPO4.dt <- ( tran.PO4$dC
                 + PCratio*Mineralisation*(1-poro)/poro
                 + Desorption*(1-poro)/poro
                 - Adsorption)

    TotalMin     <- sum(Mineralisation*Grid$dx * (1-poro))
    TotalPmin    <- PCratio*TotalMin
    TotalPdesorp <- sum(Desorption    *Grid$dx * (1-poro))
    TotalPadsorp <- sum(Adsorption    *Grid$dx * poro)

    return(list(c(dPOC.dt, dDIC.dt, dPO4.dt, dPads.dt),

          Mineralisation_B = PCratio*Mineralisation*(1-poro),
          Adsorption_B     = Adsorption*poro,
          Desorption_B     = Desorption*(1-poro),

          Quotient         = PO4*poro / (Pads*(1-poro)),

          TotalMin     = TotalMin,
          TotalPmin    = TotalPmin,
          TotalPdesorp = TotalPdesorp,
          TotalPadsorp = TotalPadsorp,

          DIC.SWI.Flux  = tran.DIC$flux.up,
          DIC.Deep.Flux = tran.DIC$flux.down,
          POC.SWI.Flux  = tran.POC$flux.up,
          POC.Deep.Flux = tran.POC$flux.down,
          PO4.SWI.Flux  = tran.PO4$flux.up,
          PO4.Deep.Flux = tran.PO4$flux.down,
          Pads.SWI.Flux = tran.Pads$flux.up,
          Pads.Deep.Flux= tran.Pads$flux.down,
          POC.SWI.Flux  = tran.POC$flux.up,
          POC.Deep.Flux = tran.POC$flux.down,
          POP.SWI.Flux  = PCratio*tran.POC$flux.up,
          POP.Deep.Flux = PCratio*tran.POC$flux.down))
 })
}
```

```
names <- c("POC", "DIC", "PO4", "Pads")
nspec <- length(names)
Conc  <- runif(nspec*N)

default.parms <- c(
 biot      = 5e-4/365,      # [m2/d]      bioturbation mixing coefficient
 v         = 0.1e-2/365,    # [m/d]       sediment advection velocity
 rMin      = 0.01,          # [/d]        POC mineralisation rate constant
 depoPOC   = 1e-3,          # [mol/m2/d]  POC deposition rate (flux at SWI)
 bwDIC     = 2,             # [mol/m3]    DIC bottom water concentration
 bwPO4     = 0.5e-3,        # [mol/m3]    PO4 bottom water concentration
 PCratio   = 1/106,         # [molP/molC] P:C ratio in Redfield organic matter
 rPads     = 5e-4,          # [/d]        rate constant for P adsorption to CaCO3
 rPdes     = 1e-5           # [/d]        P desorption rate constant
   )
```

Experimental data

Suppose that we have the following experimental data for the concentrations of DIC and PO4 (mol/m3) at several depths (m) of the sediment:

```
knitr::kable(DATA, digits = c(4,4,5))
```

x	DIC	PO4
0.01	2.1184	0.00180
0.02	2.1530	0.00207
0.04	2.1494	0.00205
0.06	2.1563	0.00193
0.08	2.1475	0.00182
0.10	2.1584	0.00171
0.15	2.1508	0.00147
0.20	2.1488	0.00127

Fitting the model to data

Methods used for fitting a mechanistic model to data are implemented in the R-package FME. Details of this package are described by Soetaert and Thomas Petzoldt (2010). Here we only illustrate how the methods are *applied*.

The overall approach has several steps: (1) selection of model parameters that can be fitted

to the data; (2) finding the best estimates of the model parameters; (3) calculating the data-dependent probability distribution of the model parameters; and (4) calculating the effect of the parameter uncertainty on the model output.

Functions for fitting

To fit the above model to the data, we need to define a "cost function". This function takes as input the model parameters to fit (`fitpar`), runs the model, and returns the *deviation* of the model output from the observed data. The observed data (`OBS`) are passed as input to the function as well.

In the first step, it is handy to create a function that runs the model for a given set of model parameters (input argument `pars`) and returns the profiles of the model variables as a function of the sediment depth. Here this function is called *std.fun*. We will also use it when performing the sensitivity analysis (see below).

```
std.fun <- function(pars){
  std <- steady.1D (y=Conc, func=PDiamodel, parms=pars, nspec=nspec, dimens=N,
                    names=names, positive=TRUE)
  return(data.frame(x = Grid$x.mid, std$y))
}
```

The simplest way to do the model-data comparison is by using the FME function *modCost*. Using the detailed profiles generated by the model, this function extracts the values that correspond to the data. Note that because the values of the data are quite different (e.g., the DIC concentrations are three orders of magnitude greater than the PO4 concentrations), we rescale the model-data deviations by the mean values of each observed data set (weight="mean"). Alternative rescaling approaches include "std" or "none" (default).

Typically, not all model parameters are tuned to fit the model.[1] Therefore, in the first two lines of the function *modCost.fun*, we create the parameter vector (p) that contains the default parameter values "updated" by those that need to be fitted (passed in the input variable `fitpar`). When calling *modCost*, we also pass the *name* of the independent variable in the model and the data (here this name is "x").

```
modCost.fun <- function(fitpar, OBS){
  p               <- default.parms
  p[names(fitpar)] <- fitpar

  return(modCost(model = std.fun(p),
                 obs = OBS, x = "x", weight = "mean"))
}
```

[1]The FME package offers the possibility to find out which model parameters *can* or *cannot* be constrained by fitting the model to the available data. Explaining this functionality here would, however, go far beyond the scope of this Reader. If you are interested in learning more about this, consult the paper by Soetaert and Petzoldt (2010) and the examples provided with the FME package.

The function *modCost.fun* returns a list with a lot of output that defines the deviation of the model results from the data. One of them is called *model*, which corresponds to the sum of squared residuals (SSR).

```
require(FME)
mC <- modCost.fun(fitpar=default.parms, OBS=DATA)
mC$model   # SSR
```

```
## [1] 8.211812
```

Fitting by a simple approach

Now we perform the actual fitting using a simple approach. In this first example, we assume that we can use the available data to constrain three model parameters: `depoPOC`, `biot` and `rPdes`. Therefore, we first define a vector containing the *names* and *initial guesses* of these parameters (`pIni`) and the corresponding minimum (`pMin`) and maximum values (`pMax`). Then, we use the FME function *modFit* to find the best model fit.

```
pIni <- c(depoPOC = 1e-3, biot = 5e-4/365, rPdes = 1e-5)  # initial guesses
pMin <- c(          0,           0,                0   )  # minimum allowed
pMax <- c(          5e-3,       10e-4/365,         1e-4)  # maximum allowed

# perform the fitting of observations stored in DATA
PFit <- modFit(f=modCost.fun, OBS=DATA, p=pIni, lower=pMin, upper=pMax)
```

We use the function *summary* to display the results of the fitting:

```
SFit <- summary(PFit)
SFit
```

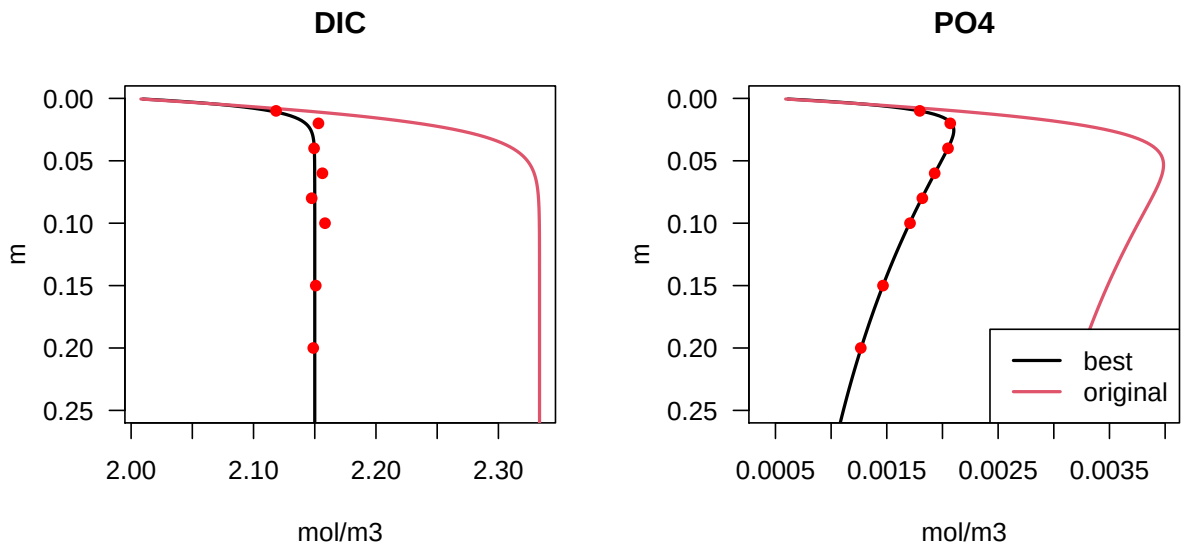
```
##
## Parameters:
##          Estimate Std. Error t value Pr(>|t|)
## depoPOC 1.203e-03  1.117e-05  107.74  < 2e-16 ***
## biot    2.727e-07  4.825e-09   56.52  < 2e-16 ***
## rPdes   1.997e-06  7.719e-08   25.87 1.45e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.002919 on 13 degrees of freedom
##
## Parameter correlation:
##         depoPOC    biot   rPdes
## depoPOC  1.0000 -0.9801  0.3460
## biot    -0.9801  1.0000 -0.4773
## rPdes    0.3460 -0.4773  1.0000
```

On the one hand, the results show the best estimates of the (fitted) model parameters, their estimated standard errors, and the p-value.[2] On the other hand, the results also show that there is a high negative correlation between the model parameters `depoPOC` and `biot`. This indicates that it may be difficult, if possible at all, to estimate both of these parameters *independently* based on the available experimental data. We will discuss this issue further below.

In the last step of this simple fitting approach, we run the model with the original (`default.parms`) and best-fit (`PFit$par`) parameters and compare the results with the experimental data.

```
p <- default.parms
p[names(PFit$par)] <- PFit$par

Original <- steady.1D(y=Conc, func=PDiamodel, parms=default.parms,
                      nspec=nspec, dimens=N, names=names,
                      positive=TRUE)
BestFit  <- steady.1D(y=Conc, func=PDiamodel, parms=p,
                      nspec=nspec, dimens=N, names=names,
                      positive=TRUE)
plot(BestFit, Original, xyswap=TRUE, grid=Grid$x.mid, ylim=c(0.25,0),
     lty=1, lwd=2, las=1, ylab="m", xlab="mol/m3",
     obs=DATA, obspar=list(pch=16, col="red"))

legend("bottomright", col=1:2, lty=1, lwd=2, legend=c("best","original"),
       bg="white")
```

[2]The p-value describes the chance (probability) that, given the data, the predicted model parameter is *different from zero* while, in reality, the model parameter *is zero*. For example, the best estimate of the parameter `rPdes` is about 2×10^{-6}, which is a value different from zero. However, IF true value of this paremeter *was zero*, then the probability that the experimental data are the way they are is about 1.4×10^{-12}, which is a very small number. In other words, it is very *unlikely* that the experimental data would be as measured while, at the same time, the true value of `rPdes` was zero. Thus, based on the available experimental data, we reject the hypothesis that the value of `rPdes` is zero, and instead conclude that the value is somewhere in the range predicted by the fit.



As expected, the data are fitted well by the model parameterized by the values obtained from the fit.

Advanced fitting techniques

Markov Chain

Now we use a Bayesian technique, the Markov Chain Monte Carlo (MCMC), to derive the *probability density* function of model parameters, given the data. The specifics of this technique are beyond the scope of this Reader (see Soetaert and Petzoldt (2010) for details). The application of this technique goes as follows:

Among other things, the MCMC makes use of the cost function (*modCost.fun*) and the best parameter estimates (`PFit$par`) from above. Here, we let the *modMCMC* function run 10000 iterations.

```
Covar   <- SFit$cov.scaled * 2.4^2/3
s2prior <- SFit$modVariance
N.iter  <- 10000

MCMC <- modMCMC(f=modCost.fun, p=PFit$par, lower=pMin, upper=pMax,
    OBS=DATA, var0=s2prior, jump=Covar, wvar0=1, updatecov=100, niter=N.iter)
```

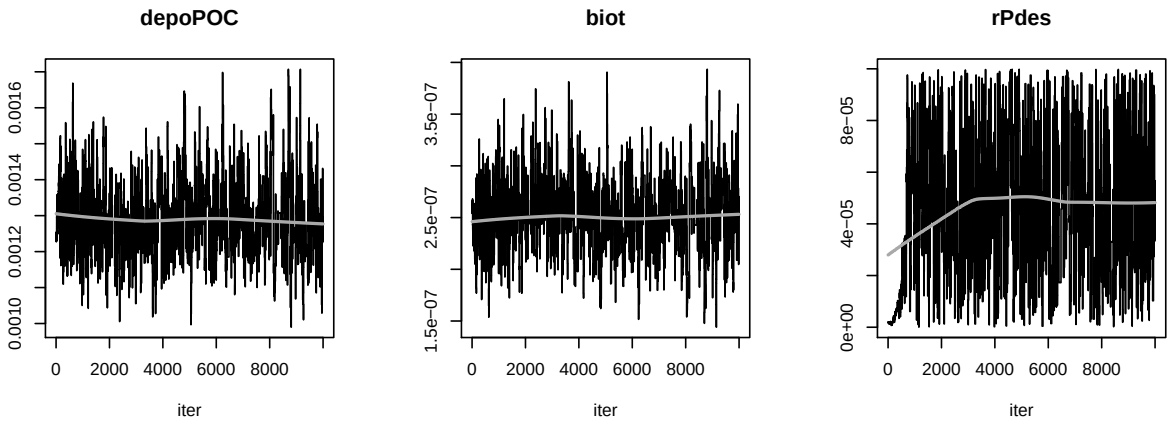
```
summary(MCMC)
```

```
##           depoPOC         biot        rPdes    var_model
## mean 0.0012966985 2.509325e-07 4.694812e-05 2.043384e-05
## sd   0.0001154603 3.811474e-08 2.971177e-05 8.736931e-06
```

```
## min  0.0009897627 1.440226e-07 5.989185e-09 5.989530e-06
## max  0.0017070475 3.933474e-07 9.981678e-05 7.091619e-05
## q025 0.0012135999 2.246069e-07 2.089672e-05 1.437639e-05
## q050 0.0012828345 2.511829e-07 4.518703e-05 1.855041e-05
## q075 0.0013689159 2.764478e-07 7.217742e-05 2.399055e-05
```

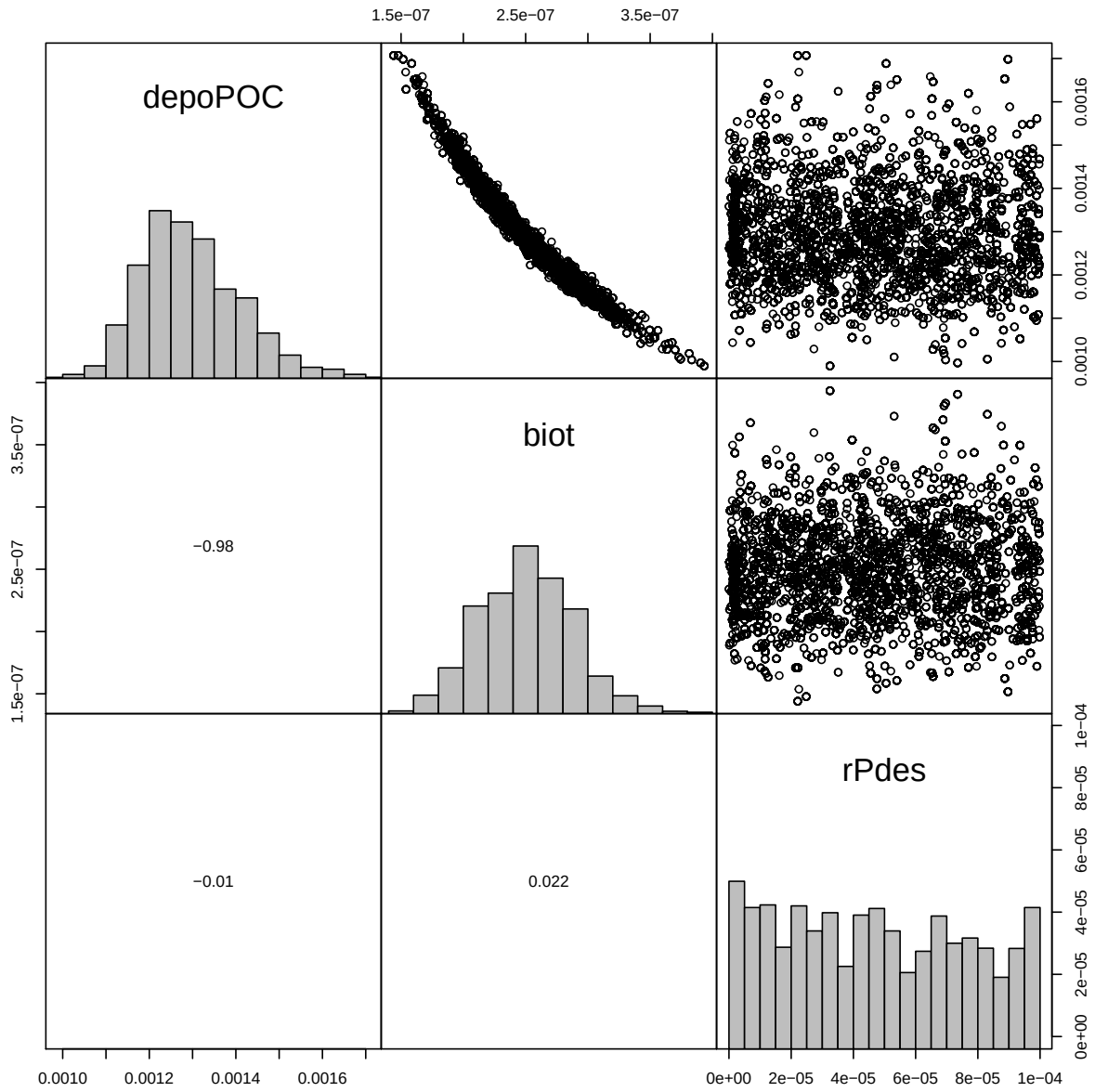
The first figure shows the parameter values that were selected; if the values appear random, then the MCMC has converged.

```
plot(MCMC, mfrow=c(1,3))
```



The next two figures show the selected values of the model parameters and their pair-wise relationships.

```
pairs(MCMC)
```



We see that the values of `depoPOC` and `biot` are constrained within a reasonably well-defined interval, as indicated by the corresponding histograms. However, there is a non-linear relationship between them,[3] indicating that they *cannot* be constrained *independently* by the available data. Additionally, the lower-right histogram shows that the parameter `rPdes` *cannot* be fitted from the data! Thus, the MCMC analysis suggests that the available data is *insufficient* to constrain (estimate) the P desorption rate constant `rPdes`.

[3]Note that the summary of the fit assumes a linear relationship (see Soetaert and Petzoldt (2010) for more details).

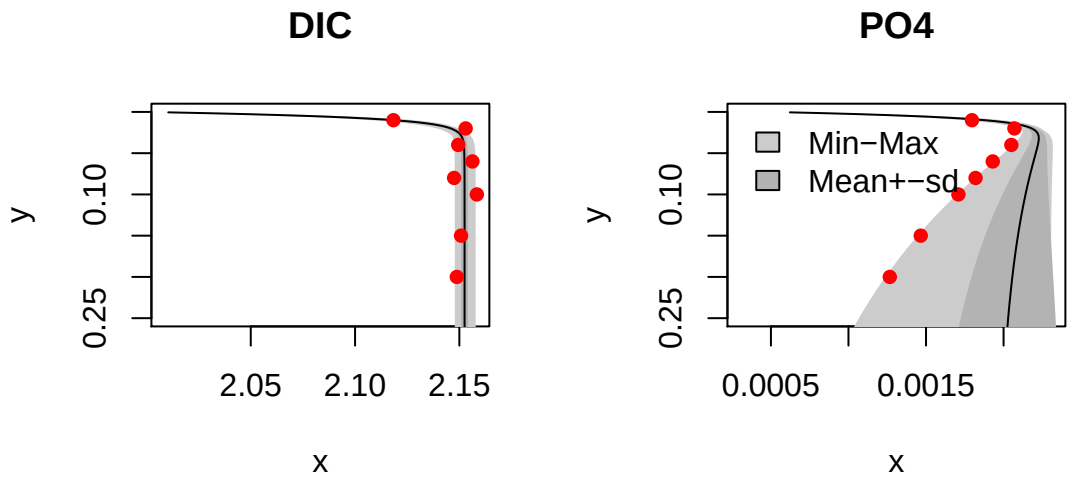
Sensitivity range

To illustrate the implications of the conclusion reached above, we perform a sensitivity analysis using the funcion *sensRange*. This function calculates the effect of the parameter uncertainty on the model output. First, we use 1000 values of model parameters randomly selected from the distributions determined by the *modMCMC* function and calculate the range of model variables predicted by the model.

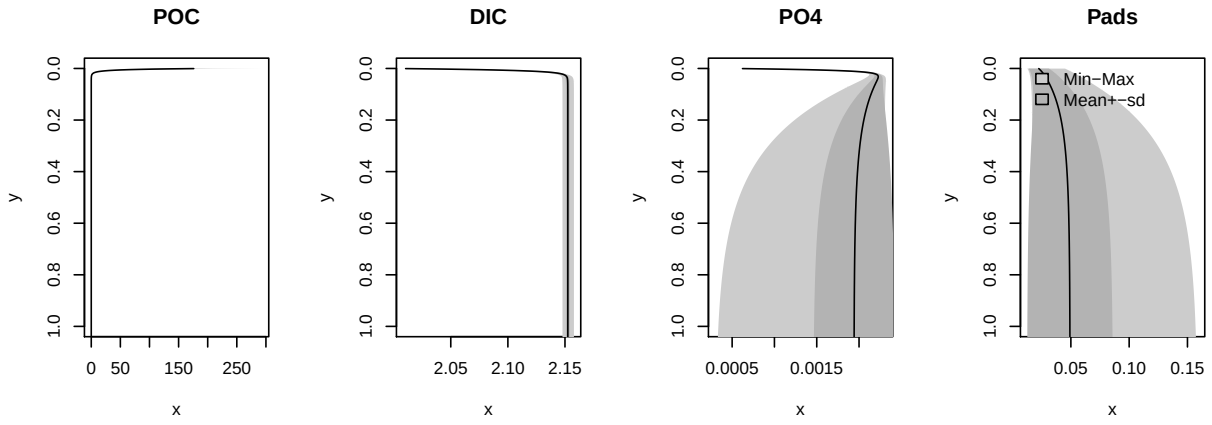
```
sR <- sensRange(parms=default.parms, parInput=MCMC$par, f=std.fun,
                num=100)
```

As shown by the graphs below, sediment concentrations of the dissolved PO_4 and adsorbed P predicted by the model based on the available experimental data are *very poorly* constrained!

```
plot(summary(sR), xyswap=TRUE, ylim=c(0.25,0), mfrow=c(1,2),
     obs=DATA, obspar=list(pch=16, col="red"))
```




```
plot(summary(sR), xyswap=TRUE, ylim=c(1,0), mfrow=c(1,4))
```



Experimental data revisited

One possible aim of gathering data is to constrain parameters of a model and then use the model to make predictions about processes that cannot be measured directly. For example, we could decide that we want to estimate the rate of P burial in sediments in the form of adsorbed phosphate. The analysis above revealed that our estimate would not be very useful if the available data only consisted of DIC and PO_4 concentrations shown above. This illustrates how modeling can provide valuable insights when designing a sampling strategy.

Suppose that the available experimental data is more extensive, consisting of the concentrations of DIC, PO_4 and P_{ads} over a greater depth range, as shown in the following table:

```
knitr::kable(DATA2, digits = c(4,3,5,4))
```

x	DIC	PO4	Pads
0.01	2.104	0.00159	0.0430
0.02	2.118	0.00180	0.0467
0.04	2.120	0.00175	0.0535
0.06	2.120	0.00166	0.0592
0.08	2.120	0.00161	0.0651
0.10	2.132	0.00150	0.0698
0.15	2.131	0.00129	0.0801
0.20	2.134	0.00112	0.0907
0.25	2.134	0.00099	0.0985
0.30	2.135	0.00086	0.1052
0.35	2.129	0.00075	0.1117
0.40	2.118	0.00068	0.1158

Fitting of the new experimental data

Here, we additionally assume that the POC flux at the sediment-water interface can be constrained through an independent measurement (1 $mmolC\ m^{-2}\ d^{-1}$, which corresponds to the POP flux of $0.001/106 = 9.434\ \mu molP\ m^{-2}\ d^{-1}$). Thus, our aim now is to test whether we can better constrain the parameters `biot` and `rPdes` by fitting the model to the new data.

To do this, we define the initial guesses (`pIni2`) and minimum (`pMin2`) and maximum values (`pMax2`) of the model parameters and use the *modFit* function to perform the initial fitting of the new data (`DATA2`):

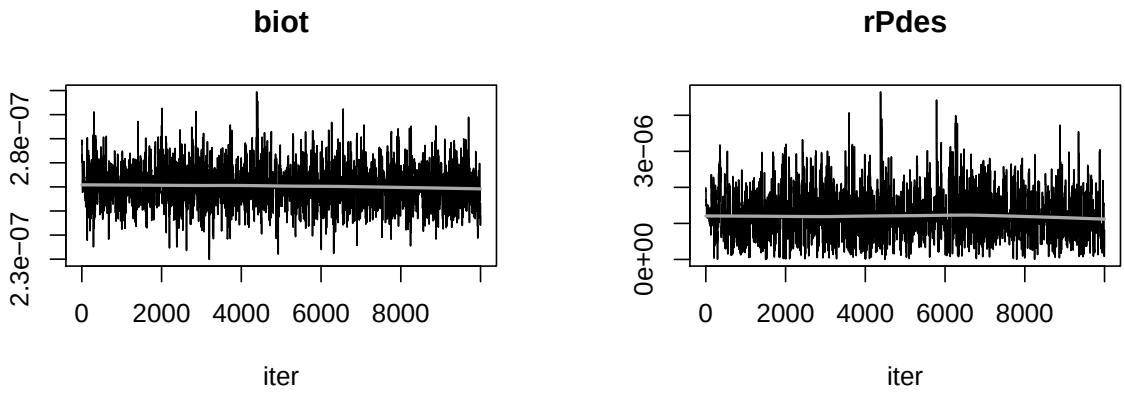
```
pIni2 <- c(biot =  5e-4/365, rPdes = 1e-5)
pMin2 <- c(         0,                0   )
pMax2 <- c(       10e-4/365,          1e-4)

PFit2 <- modFit(f = modCost.fun, OBS=DATA2, p=pIni2, lower=pMin2, upper=pMax2)
```

Here we skip the steps showing the results of this initial fitting step, and continue directly with the MCMC fitting following the same steps as described above. Note that now we use the output variable `PFit2` and the data in the variable `DATA2`.

```
SFit <- summary(PFit2)
Covar <- SFit$cov.scaled * 2.4^2/3
s2prior <- SFit$modVariance
MCMC2 <- modMCMC(f=modCost.fun, p=PFit2$par, lower=pMin2, upper=pMax2,
                 OBS=DATA2, var0=s2prior, jump=Covar, wvar0=1, updatecov=100,
                 niter=N.iter)
```

The following figures show that the values of the selected parameters appear random, thus the MCMC has converged.

```
plot(MCMC2, mfrow=c(1,2))
```



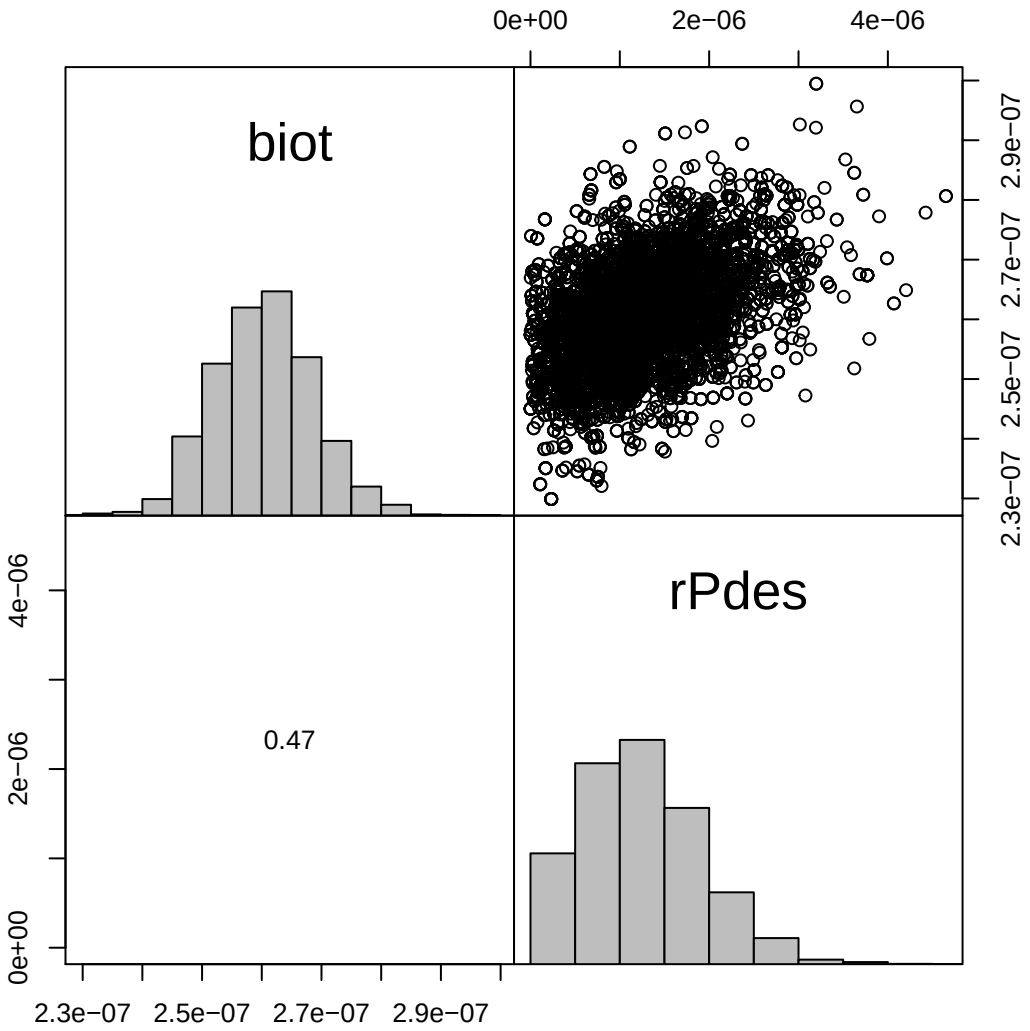
Using the new data, the estimates of the fitted model parameters are as follows:

```
summary(MCMC2)
```

```
##              biot        rPdes    var_model
## mean 2.603863e-07 1.240090e-06 6.898364e-05
## sd   8.368137e-09 6.675976e-07 2.125837e-05
## min  2.299070e-07 1.676848e-09 2.940359e-05
## max  2.994644e-07 4.650308e-06 1.979941e-04
## q025 2.546326e-07 7.418070e-07 5.334639e-05
## q050 2.603966e-07 1.178140e-06 6.580682e-05
## q075 2.657954e-07 1.668331e-06 8.099440e-05
```

The next two figures show that the values of `biot` and `rPdes` are constrained within a rather narrow interval. The scatter plot indicates, however, that there is a slight colinearity between these two model parameters, suggesting that their estimates based on the new data may not be totally independent.

```
pairs(MCMC2)
```



Predictions of the model based on the new experimental data

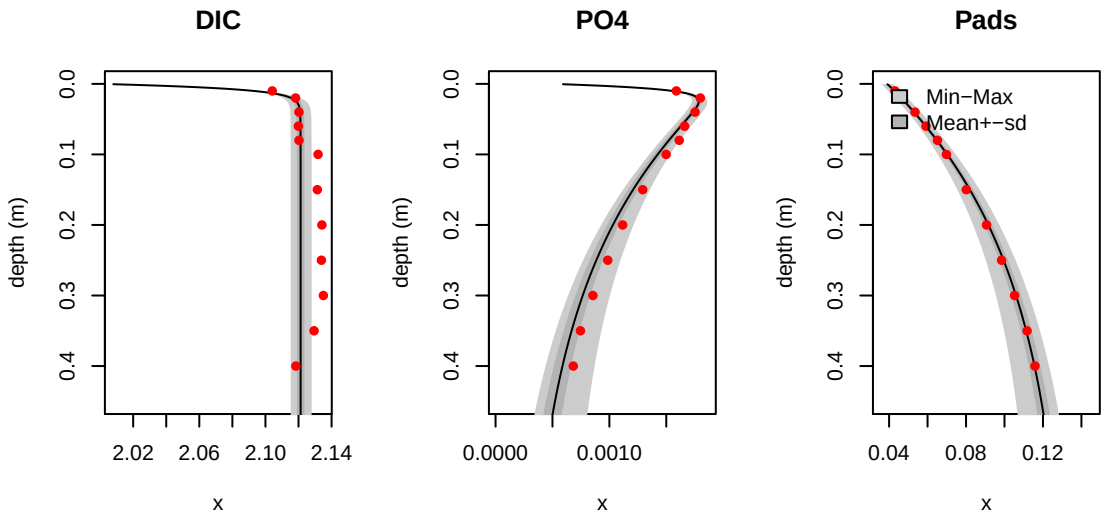
In the last step, we calculate model predictions based on the (uncertain) model parameters determined in the previous step. First, we predict *depth profiles* of the state variables, then we predict *fluxes* of the state variables *PO4* and *Pads*, and of the output variable *POP*, at the model boundary. The fluxes will be used to reconstruct the P budget in the sediment column predicted by the model given the (new) data.

To predict depth profiles, we run the function *sensRange* with the input argument `f` set to the function *std.fun*, which was defined above.

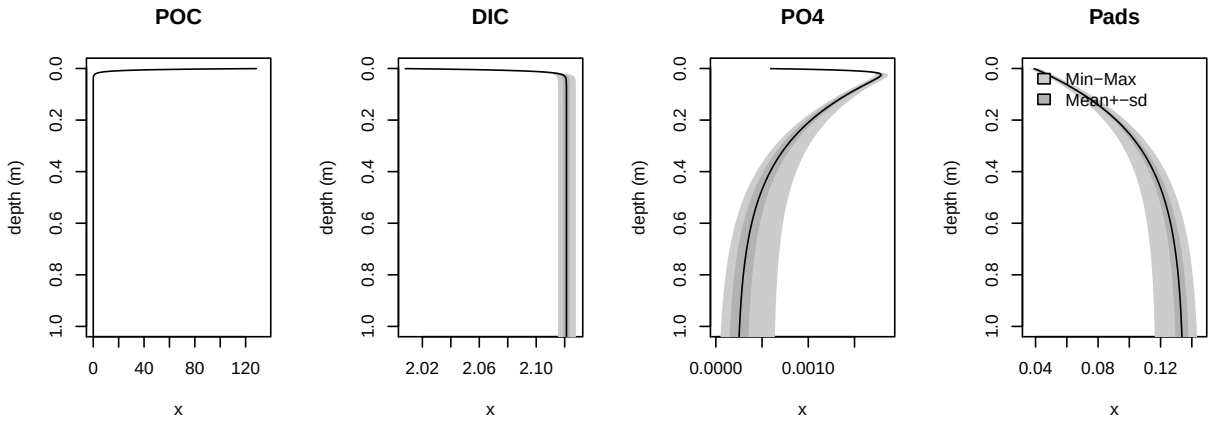
```
sR2 <- sensRange(parms=default.parms, parInput = MCMC2$par, f = std.fun,
                 num = 100)
```

By plotting the results, we see that, based on the new data, the sediment concentrations of the DIC, dissolved PO_4 and adsorbed P predicted by the model are much better constrained.

```
plot(summary(sR2), xyswap = TRUE, ylim = c(0.45,0), mfrow = c(1,3),
     obs = DATA2, obspar = list(pch = 16, col = "red"), ylab="depth (m)")
```




```
plot(summary(sR2), xyswap=TRUE, ylim=c(1,0), ylab="depth (m)", mfrow=c(1,4))
```



To predict phosphorus fluxes at the model boundaries, we must first define a new function that is similar to *std.fun* but returns fluxes rather than profiles, and then use it when calling the function *sensRange* (input argument `f`).

```
std.fun2 <- function(pars){
  std <- steady.1D (y=Conc, func=PDiamodel, parms=pars, nspec=nspec,
                    dimens=N, names=names, positive=TRUE)
```

```
  toselect <- c("PO4.SWI.Flux",  "PO4.Deep.Flux",
                "Pads.SWI.Flux", "Pads.Deep.Flux",
                "POP.SWI.Flux",  "POP.Deep.Flux")
  return(data.frame(std[toselect]))
}
```

We compare the predicted P budgets using the fitted model parameters obtained from the old (`MCMC$par`) and new data (`MCMC2$par`). Note that because the *std.fun2* function does not output the (independent) mapping variable (e.g., the sediment depth, x), we need to set `map=NULL` when calling the *sensRange* function (see `?sensRange` for more explanation).

```
sR.old <- sensRange(parms=default.parms, parInput=MCMC$par,  f=std.fun2,
                    num=100, map=NULL)
sR.new <- sensRange(parms=default.parms, parInput=MCMC2$par, f=std.fun2,
                    num=100, map=NULL)
```

```
out.old <- summary(sR.old)
out.new <- summary(sR.new)

# final P budget, values in umol P m-2 d-1
BUDGET.estim <- data.frame(
  OldData = data.frame(out.old["Mean"], out.old["Sd"]),
  NewData = data.frame(out.new["Mean"], out.new["Sd"])
  ) * 1e6
knitr::kable(BUDGET.estim, digits=4)
```

	OldData.Mean	OldData.Sd	NewData.Mean	NewData.Sd
PO4.SWI.Flux	-12.1958	1.0671	-9.2850	0.0056
PO4.Deep.Flux	0.0032	0.0007	0.0004	0.0002
Pads.SWI.Flux	0.0000	0.0000	0.0000	0.0000
Pads.Deep.Flux	0.0562	0.0389	0.1486	0.0057
POP.SWI.Flux	12.2551	1.0691	9.4340	0.0000
POP.Deep.Flux	0.0000	0.0000	0.0000	0.0000

The table above shows that the P fluxes (in $\mu mol\ P\ m^{-2}\ d^{-1}$) are in a much narrower interval if predicted by the model constrained by the new data (Sd is much smaller). For example, out of the P flux that enters the sediment due to the sedimentation of organic matter (9.434 $\mu mol\ P\ m^{-2}\ d^{-1}$), about $9.285 \pm 0.0056\ \mu mol\ P\ m^{-2}\ d^{-1}$ leaves the sediment at the SWI in the form of PO_4, while about $0.1486 \pm 0.0057\ \mu mol\ P\ m^{-2}\ d^{-1}$ is buried in the deeper sediment in the form of P_{ads}.

References

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Soetaert Karline (2009). rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations. R-package version 1.6. https://CRAN.R-project.org/package=rootSolve

Soetaert, Karline and Meysman, Filip (2012). Reactive transport in aquatic ecosystems: Rapid model prototyping in the open source software R. Environmental Modelling & Software, 32, 49-60.

Soetaert, Karline and Thomas Petzoldt (2020). marelac: Tools for Aquatic Sciences. R package version 2.1.10. https://CRAN.R-project.org/package=marelac

Soetaert, Karline and Thomas Petzoldt, 2010. Inverse Modelling, Sensitivity and Monte Carlo Analysis in R Using Package FME. Journal of Statistical Software, 33(3), 1-28. DOI 10.18637/jss.v033.i03 URL http://www.jstatsoft.org/v33/i03/.