R-package **FME** : tests

Karline Soetaert
NIOO-CEME
The Netherlands

Abstract

This vignette tests several applications of Rpackage **FME** .

Keywords: simulation models, differential equations, fitting, sensitivity, monte carlo, identifiability R.

1. Introduction

Here some functions from R-package **FME** are tested. As it is a test of the function implementation, this does not mean that the results obtained are meaningful.

It includes:

- Three tests of the implemented MCMC method:
  - The "banana" function, sampling from a curvilinear function ((Laine 2008))
  - A monod function, fitted to a data-series ((Laine 2008))
  - A simple chemical model, fitted to a data series ((Haario, Laine, Mira, and Saksman 2006))
- sensitivity analysis, fitting, MCMC, ... of a simple model of sedimentary oxygen

2. The banana

2.1. the model

This example is from Laine (2008).

A banana-shaped function is created by distorting a two-dimensional Gaussian distribution, with mean = 0 and a covariance matrix τ) with unity variances and covariance of 0.9.

$$\tau = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.9 \end{bmatrix}$$

The distortion is along the second-axis only and given by:

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_2 + x_1^2 + 1 \end{aligned}$$

2.2. R-implementation

First the banana function is defined.

```
> Banana <- function (x1,x2)
+ {
+   return(x2 - (x1^2+1))
+ }
```

We also need a function that estimates the probability of a multinormally distributed vector

```
> pmultinorm <- function(vec,mean,Cov)
+
+ {
+  diff <- vec - mean
+  ex   <- -0.5*t(diff) %*% solve(Cov) %*% diff
+  rdet   <- sqrt(det(Cov))
+  power  <- -length(diff)*0.5
+  return((2.*pi)^power / rdet * exp(ex))
+ }
```

The target function returns -2 *log (probability) of the value

```
> BananaSS <- function (p)
+ {
+   P <- c(p[1],Banana(p[1],p[2]))
+   Cov <- matrix(nr=2,data=c(1,0.9,0.9,1))
+  -2*sum(log(pmultinorm(P,mean=0,Cov=Cov)))
+ }
```

The initial proposal covariance (`jump`) is the identity matrix with a variance of 5. The simulated chain is of length 1000 (`niter`). The `modMCMC` function prints the % of accepted runs. More information is in item `count` of its return element.

The First Markov chain is generated with the simple metropolis hastings (MH) algorithm

```
> MCMC <- modMCMC(f=BananaSS, p=c(0,0.5), jump=diag(nrow=2,x=5),
+                 niter=1000)

number of accepted runs: 102 out of 1000 (10.2%)

> MCMC$count
```

```
    dr_steps     Alfasteps  num_accepted num_covupdate
           0             0           102             1
```

Next we use the adaptive metropolis (AM) algorithm and update the proposal every 100 runs (updatecov)

```
> MCMC2 <- modMCMC(f=BananaSS, p=c(0,0.5), jump=diag(nrow=2,x=5),
+                    updatecov=100,niter=1000)

number of accepted runs: 211 out of 1000 (21.1%)

> MCMC2$count

    dr_steps     Alfasteps  num_accepted num_covupdate
           0             0           211            10
```

Then the metropolis algorithm with delayed rejection (DR) is applied; upon rejection one next parameter cadidate is tried (`ntrydr`). (note `ntrydr=1` means no delayed rejection steps).

```
> MCMC3 <- modMCMC(f=BananaSS, p=c(0,0.5), jump=diag(nrow=2,x=5),
+                    ntrydr=2,niter=1000)

number of accepted runs: 451 out of 1000 (45.1%)

> MCMC3$count

    dr_steps     Alfasteps  num_accepted num_covupdate
         906          2718           451             1
```

Finally the adaptive metropolis with delayed rejection (DRAM) is used. (Here we also estimate the elapsed CPU time - `print(system.time())` does this)

```
> print(system.time(
+ MCMC4 <- modMCMC(f=BananaSS, p=c(0,0.5), jump=diag(nrow=2,x=5),
+                    updatecov=100,ntrydr=2,niter=1000)
+ ))

number of accepted runs: 706 out of 1000 (70.6%)
   user  system elapsed 
   0.83    0.00    0.83 

> MCMC4$count

    dr_steps     Alfasteps  num_accepted num_covupdate
         802          2406           706            10
```

We plot the generated chains for both parameters and for the four runs in one plot. Calling `plot` with `mfrow=NULL` prevents the plotting function to overrule these settings.

```
> par(mfrow=c(4,2))
> par(mar=c(2,2,4,2))
> plot(MCMC ,mfrow=NULL,main="MH")
> plot(MCMC2,mfrow=NULL,main="AM")
> plot(MCMC3,mfrow=NULL,main="DR")
> plot(MCMC4,mfrow=NULL,main="DRAM")
> mtext(outer=TRUE,side=3,line=-2,at=c(0.05,0.95),c("y1","y2"),cex=1.25)
> par(mar=c(5.1,4.1,4.1,2.1))
```

The 2-D plots show the banana shape:

```
> par(mfrow=c(2,2))
> xl <- c(-3,3)
> yl <- c(-1,8)
> plot(MCMC$pars,main="MH",xlim=xl,ylim=yl)
> plot(MCMC2$pars,main="AM",xlim=xl,ylim=yl)
> plot(MCMC3$pars,main="DR",xlim=xl,ylim=yl)
> plot(MCMC4$pars,main="DRAM",xlim=xl,ylim=yl)
```

Finally, we test convergence to the original distribution. This can best be done by estimating means and covariances of the transformed parameter values.

```
> trans <- cbind(MCMC4$pars[,1],Banana(MCMC4$pars[,1],MCMC4$pars[,2]))
> colMeans(trans)    # was:c(0,0)

[1] -0.3757241 -0.3615950

> sd(trans)          # was:1

[1] 0.8130517 0.8356294

> cor(trans)         # 0.9 off-diagonal

          [,1]      [,2]
[1,] 1.0000000 0.8526828
[2,] 0.8526828 1.0000000
```

3. Fitting a Monod function

3.1. the model

The second example is also discussed in (Laine 2008) (who quotes Berthoux and Brown, 2002. Statistics for environmental engineers, CRC Press).

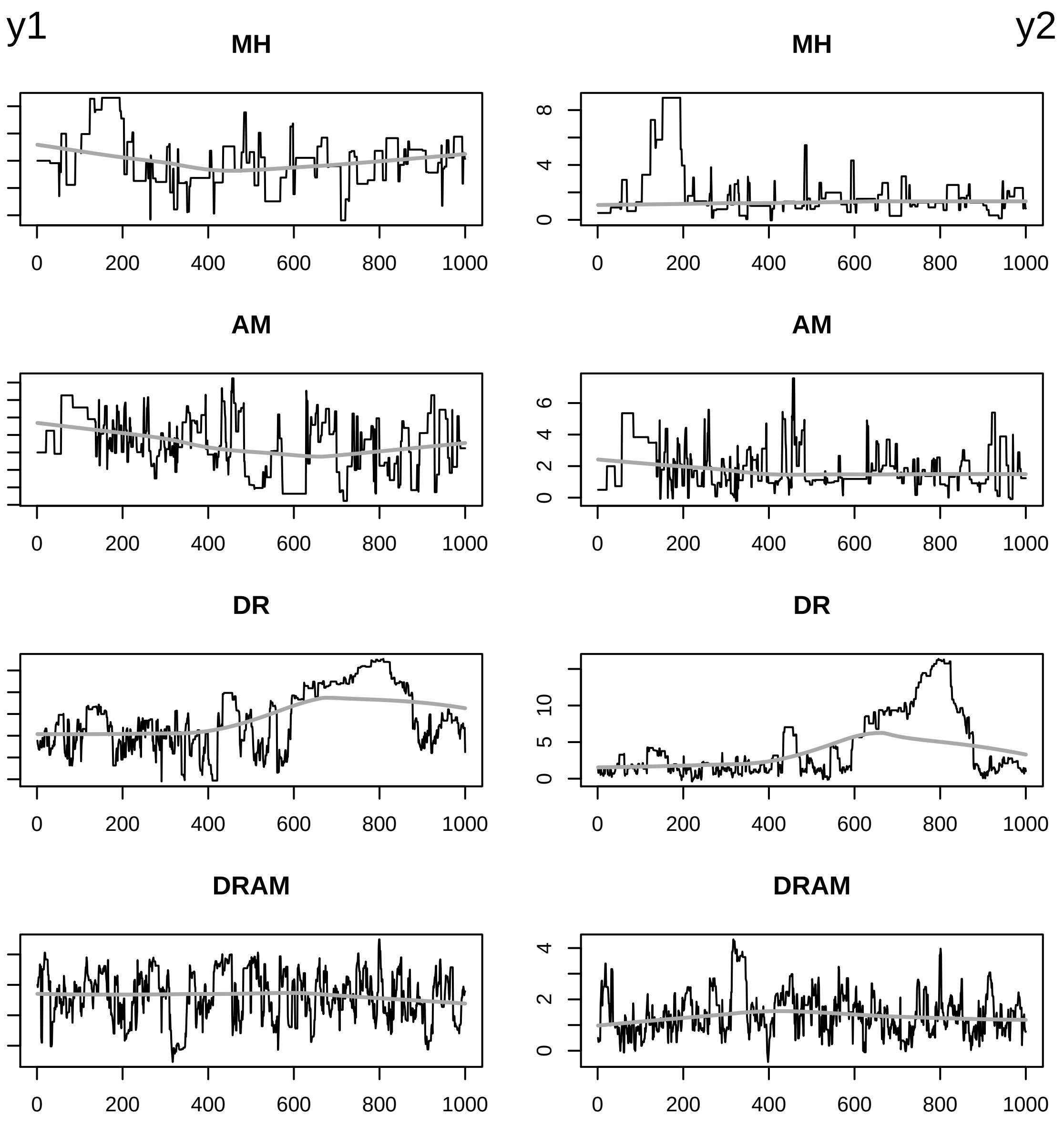

Figure 1: The MCMC chains for the four methods - see text for R-code

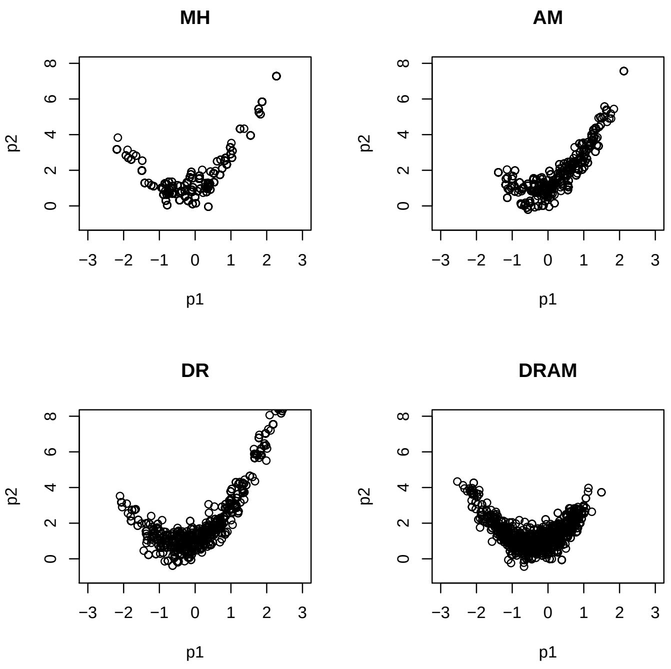

Figure 2: The bananas - see text for R-code

The following model:

$$y = \theta_1 \cdot \frac{x}{x + \theta_2} + \epsilon$$
$$\epsilon \sim N(\theta, I\sigma^2)$$

is fitted to data.

3.2. implementation in R

```
> require(FME)
```

First we input the observations

```
> Obs <- data.frame(x=c(   28,  55,   83,  110,  138,  225,  375),   # mg COD/l
+                   y=c(0.053,0.06,0.112,0.105,0.099,0.122,0.125))   # 1/hour
```

The Monod model returns a data.frame, with elements x and y :

```
> Model <- function(p,x) return(data.frame(x=x,y=p[1]*x/(x+p[2])))
```

We first fit the model to the data.

Function `Residuals` estimates the deviances of model versus the data.

```
> Residuals  <- function(p) (Obs$y-Model(p,Obs$x)$y)
```

This function is input to `modFit` which fits the model to the observations.

```
> print(system.time(
+ P      <- modFit(f=Residuals,p=c(0.1,1))
+ ))

   user  system elapsed 
   0.01    0.00    0.02 
```

We can estimate and print the summary of fit

```
> sP    <- summary(P)
> sP

Parameters:
     Estimate Std. Error t value Pr(>|t|)    
[1,]  0.14542    0.01564   9.296 0.000242 ***
[2,] 49.05292   17.91196   2.739 0.040862 *  
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1 

Residual standard error: 0.01278 on 5 degrees of freedom

Parameter correlation:
       [,1]   [,2]
[1,] 1.0000 0.8926
[2,] 0.8926 1.0000
```

We also plot the residual sum of squares, the residuals and the best-fit model

```
> x      <-0:375

> par(mfrow=c(2,2))
> plot(P,mfrow=NULL)
> plot(Obs,pch=16,cex=2,xlim=c(0,400),ylim=c(0,0.15),
+      xlab="mg COD/l",ylab="1/hr",main="best-fit")
> lines(Model(P$par,x))
> par(mfrow=c(1,1))
```

Finally, we run an MCMC analysis. The -scaled- parameter covariances returned from the `summary` function are used as estimate of the proposal covariances (`jump`). Scaling is as in (Gelman, Varlin, Stern, and Rubin 2004).

For the initial model variance (`var0`) we use the residual mean squares also returned by the `summary` function. We give equal weight to prior and modeled mean squares (`wvar0=1`)

The MCMC method adopted here is the metropolis-hastings algorithm; the MCMC is run for 3000 steps; we use the best-fit parameter set (`P$par`) to initiate the chain (`p`). A lower bound (0) is imposed on the parameters (`lower`).

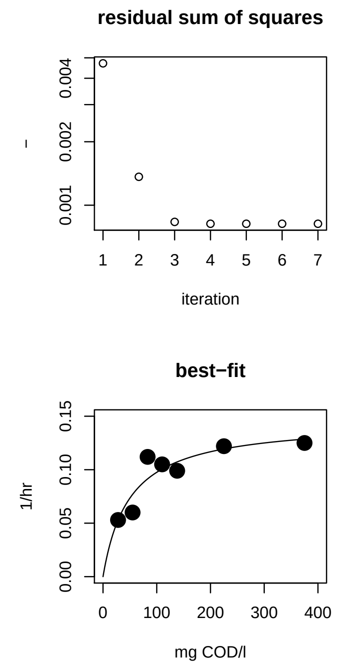

Figure 3: Fit diagnostics of the Monod function - see text for R-code

```
> Covar   <- sP$cov.scaled * 2.4^2/2
> s2prior <- sP$modVariance
> print(system.time(
+ MCMC <- modMCMC(f=Residuals,p=P$par,jump=Covar,niter=3000,
+                 var0=s2prior,wvar0=1,lower=c(0,0))
+ ))

number of accepted runs: 1037 out of 3000 (34.56667%)
   user  system elapsed 
   1.67    0.00    1.67 
```

The plotted results demonstrate (near-) convergence of the chain.

```
> plot(MCMC,Full=TRUE)
```

The posterior distribution of the parameters, the sum of squares and the model's error standard deviation.

```
> hist(MCMC,Full=TRUE,col="darkblue")
```

The pairs plot shows the relationship between the two parameters

```
> pairs(MCMC)
```

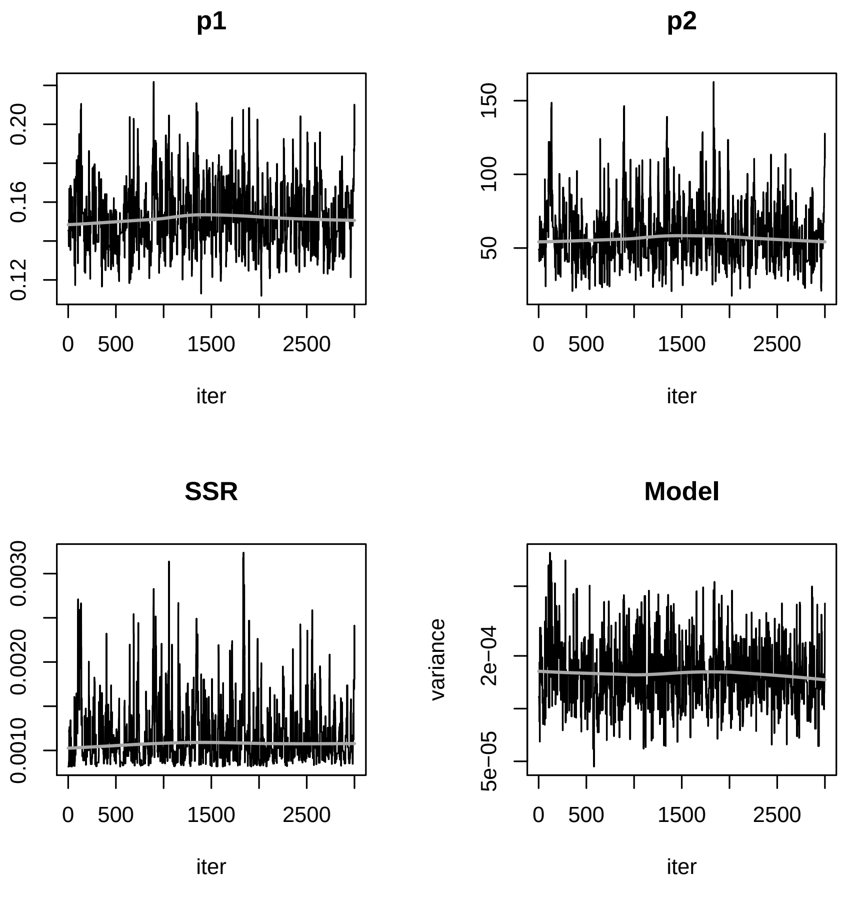

Figure 4: The mcmc - see text for R-code

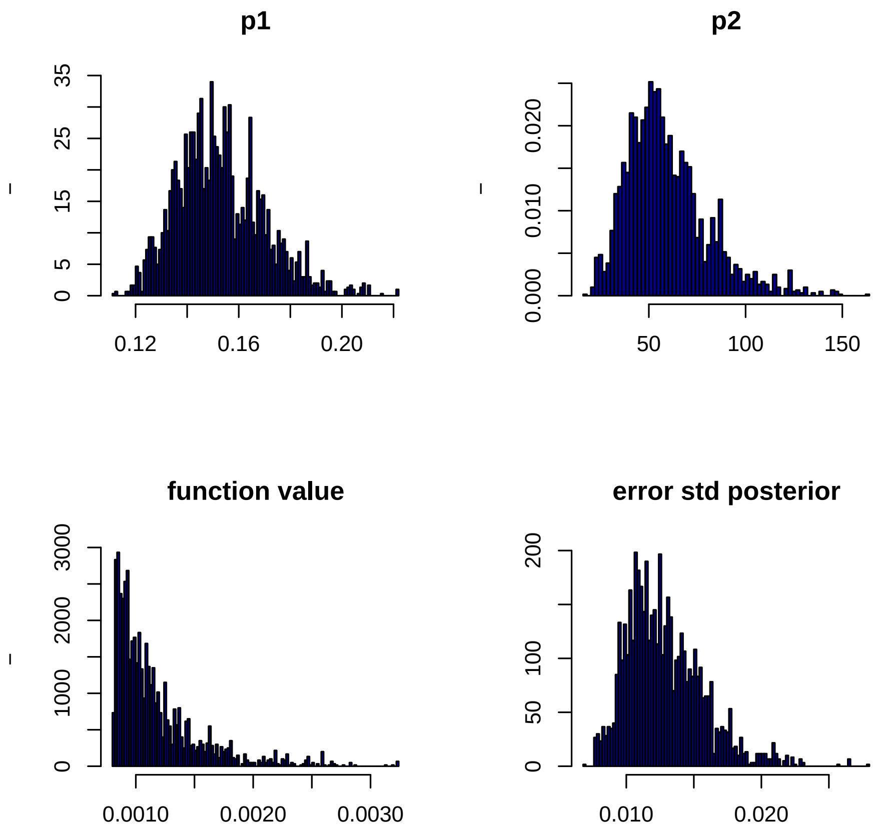

Figure 5: Hist plot - see text for R-code

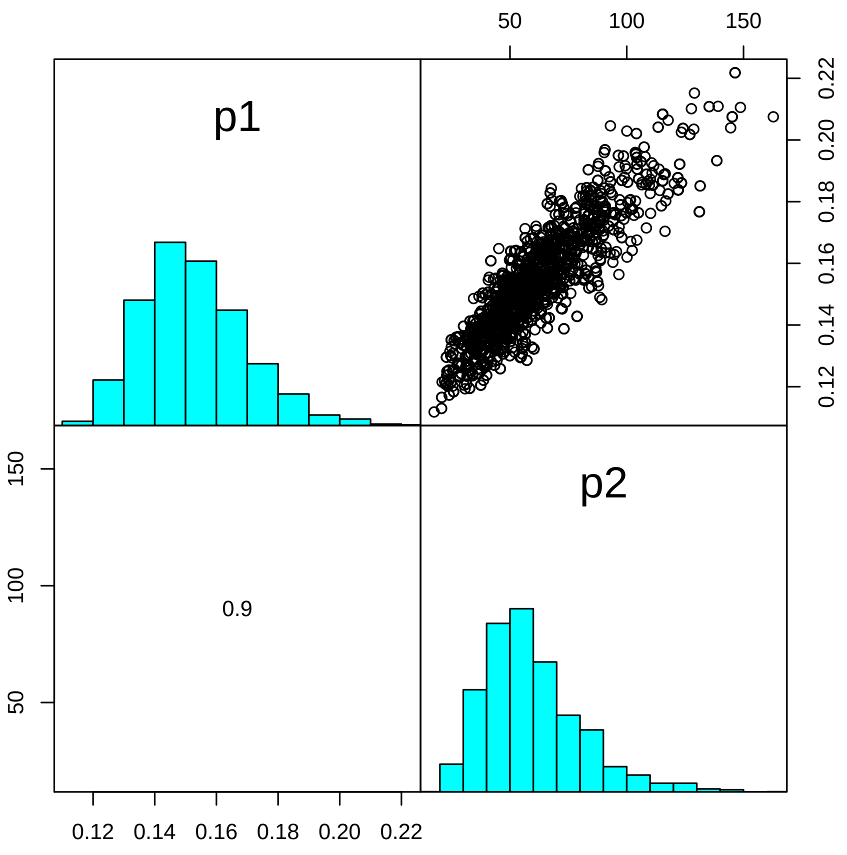

Figure 6: Pairs plot - see text for R-code

The parameter correlation and covariances from the MCMC results can be calculated and compared with the results obtained by the fitting algorithm.

```
> cor(MCMC$pars)

          p1        p2
p1 1.0000000 0.8914583
p2 0.8914583 1.0000000

> cov(MCMC$pars)

             p1         p2
p1 0.0002874929   0.317049
p2 0.3170489981 439.970548

> sP$cov.scaled

             [,1]         [,2]
[1,] 0.0002447075   0.2501157
[2,] 0.2501157147 320.8381590
```

The Raftery and Lewis's diagnostic from package **coda** gives more information on the number of runs that is actually needed. First the MCMC results need to be converted to an object of type `mcmc`, as used in **coda**.

```
> MC <- as.mcmc(MCMC$pars)
> raftery.diag(MC)

Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

You need a sample size of at least 3746 with these values of q, r and s
```

Also interesting is function `cumuplot` from **coda**:

```
> cumuplot(MC)
```

The predictive posterior distribution of the model is easily estimated by running function `sensRange`, using a randomly selected subset of the parameters in the chain (`MCMC$pars`; we use the default of 100 parameter combinations.

```
> sR<-sensRange(parInput=MCMC$pars,func=Model,x=1:375)
```

The distribution is plotted and the data added to the plot:

```
> plot(summary(sR),quant=TRUE)
> points(Obs)
```

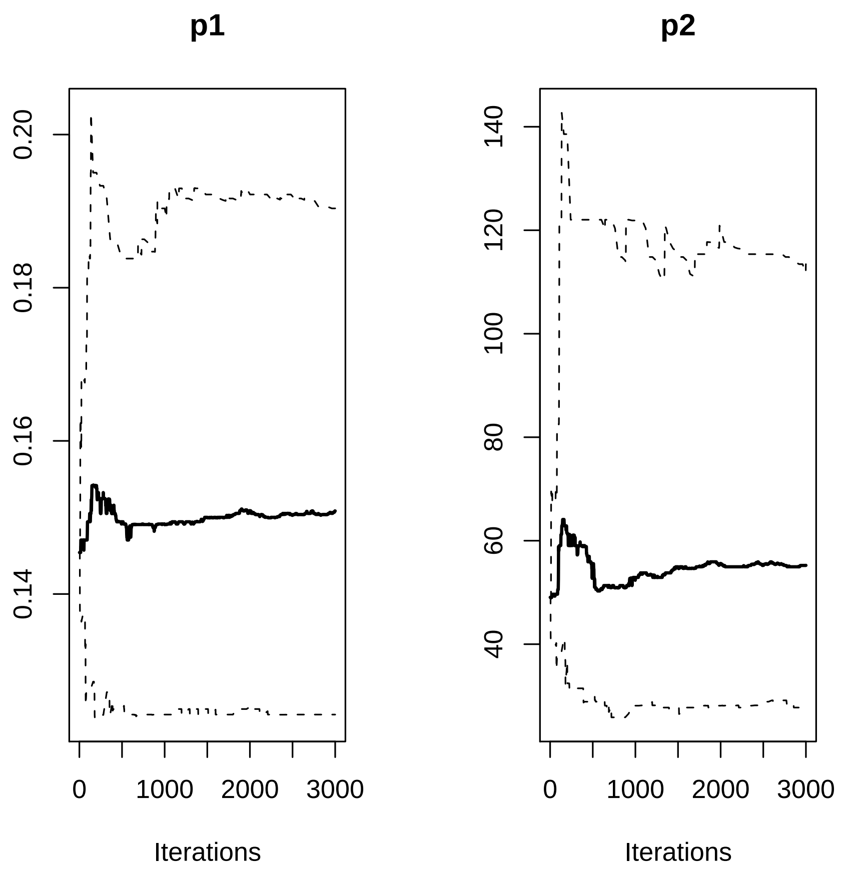

Figure 7: Cumulative quantile plot - see text for R-code

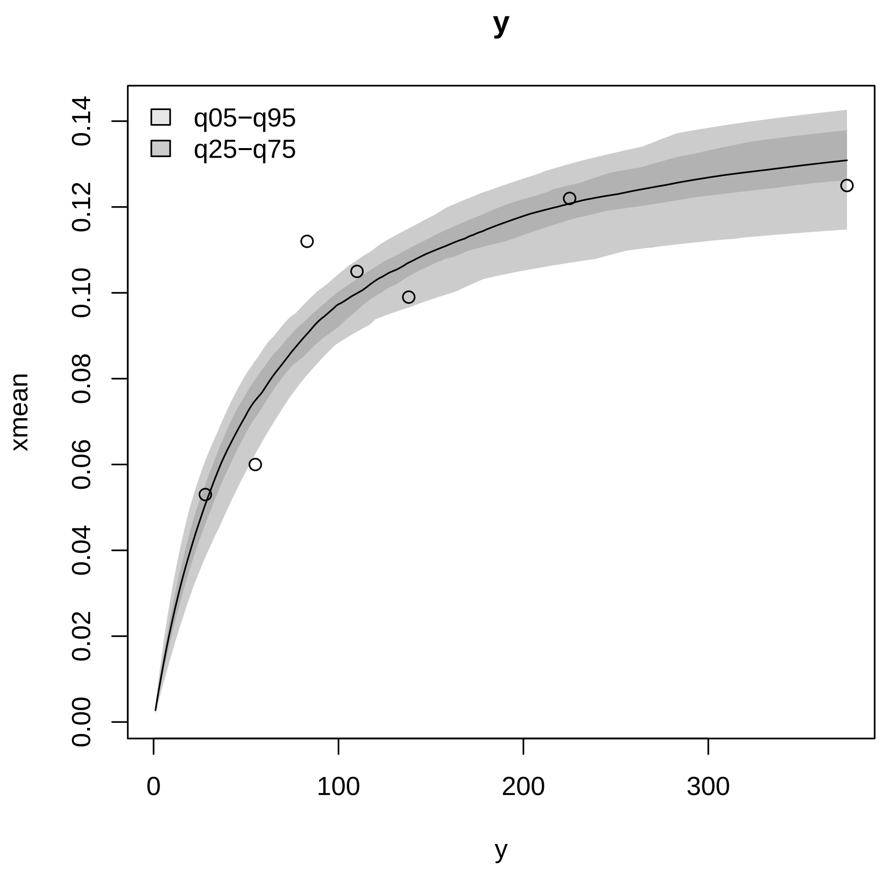

Figure 8: Predictive envelopes of the model - see text for R-code

By toggling on covariance adaptation (`updatecov` and delayed rejection (`ntrydr`), the acceptance rate is increased:

```
> print(system.time(
+ MCMC2 <- modMCMC(f=Residuals,p=P$par,jump=Covar,niter=3000, ntrydr=3,
+                  var0=s2prior,wvar0=1,updatecov=100,lower=c(0,0))
+ ))

number of accepted runs: 2555 out of 3000 (85.16667%)
   user  system elapsed 
   3.99    0.02    4.06 

> MCMC2$count

     dr_steps     Alfasteps  num_accepted num_covupdate 
         2564         10644          2555            30 
```

4. A simple chemical model

This is an example from (Haario *et al.* 2006). We fit two parameters that describe the dynamics in the following reversible chemical reaction:

$$\mathrm{A} \rightleftharpoons [k_2]k_1\mathrm{B}$$

Here k_1 is the forward, k_2 the backward rate coefficient.

The ODE system is written as:

$$\frac{dA}{dt} = -k_1 \cdot A + k_2 \cdot B$$
$$\frac{dB}{dt} = +k_1 \cdot A - k_2 \cdot B$$

with initial values $A_0 = 1$, $B_0 = 0$.

The analytical solution for this system of differential equations is given in (Haario *et al.* 2006).

First a function is defined that takes as input the parameters and that returns the values of the concentrations A and B, at selected output times.

```
> Reaction <- function (k, times)
+ {
+   fac <- k[1]/(k[1]+k[2])
+   A   <- fac + (1-fac)*exp(-(k[1]+k[2])*times)
+   return(data.frame(t=times,A=A))
+ }
```

All the concentrations were measured at the time the equilibrium was already reached. The data are the following:

```
> Data     <- data.frame(
+   times = c(2,     4,     6,     8,     10   ),
+   A     = c(0.661, 0.668, 0.663, 0.682, 0.650))
> Data

  times     A
1     2 0.661
2     4 0.668
3     6 0.663
4     8 0.682
5    10 0.650
```

We need parameter priors to prevent the model parameters from drifting to infinite values. The prior is taken to be a broad Gaussian distribution with mean (2,4) and standard deviation = 200 for both.

The prior function returns the weighted sum of squared residuals of the parameter values with the expected value.

```
> Prior <- function(p)
+     return( sum(((p-c(2,4))/200)^2 ))
```

First the model is fitted to the data; we restrict the parameter values to be in the interval [0,1].

```
> residual <- function(k) return(Data$A - Reaction(k,Data$times)$A)
> Fit <- modFit(p=c(k1=0.5,k2=0.5),f=residual,lower=c(0,0),upper=c(1,1))
> (sF <- summary(Fit))

Parameters:
   Estimate Std. Error t value Pr(>|t|)
k1   1.0000     0.3944   2.536   0.0850 .
k2   0.5123     0.1928   2.657   0.0765 .
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Residual standard error: 0.01707 on 3 degrees of freedom

Parameter correlation:
      k1    k2
k1 1.000 0.996
k2 0.996 1.000
```

The residual error of the fit is used as initial model variance, the scaled covariance matrix of the fit is used as the proposal distribution (to generate new parameter values). As the covariance matrix is nearly singular this is not a very good approximation. The initial MCMC method, using the Metropolis-Hastings method does not converge. The MCMC is initiated with the best-fit parameters; the parameters are restricted to be positive numbers (`lower`).

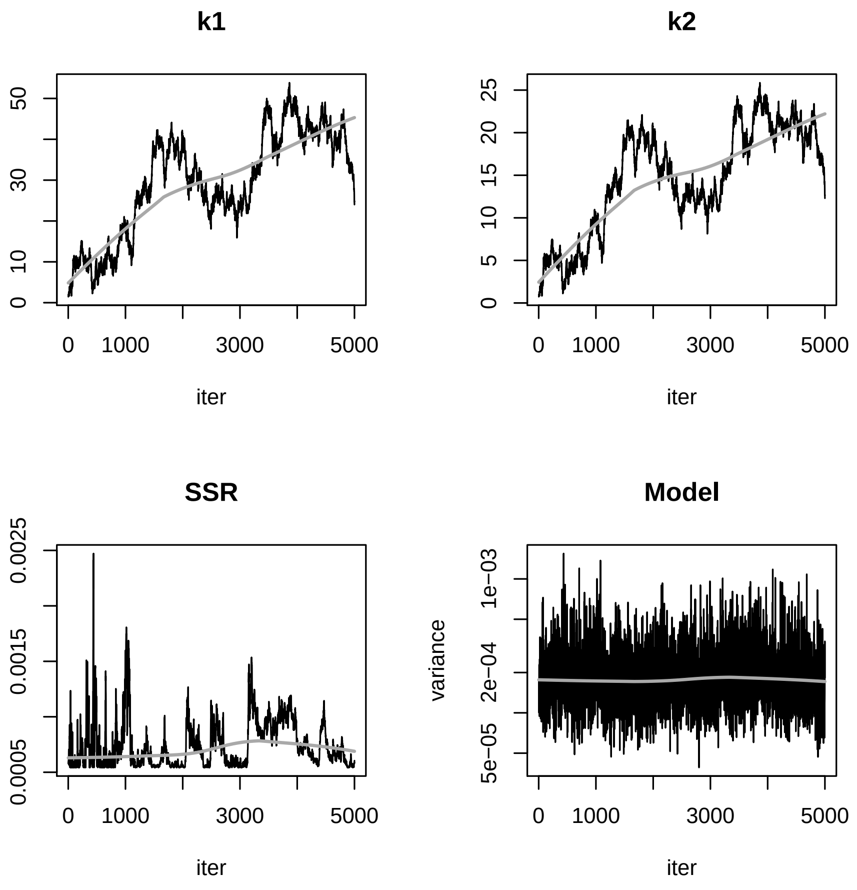

Figure 9: Metropolis-Hastings MCMC of the chemical model - see text for R-code

```
> mse <- sF$modVariance
> Cov <- sF$cov.scaled * 2.4^2/2
> print(system.time(
+ MCMC <- modMCMC(f=residual, p=Fit$par, jump=Cov, lower=c(0,0),
+                 var0=mse, wvar0=1, prior=Prior, niter=5000)
+ ))
```

```
number of accepted runs: 4796 out of 5000 (95.92%)
   user  system elapsed
      3       0       3
```

The number of accepted runs is much too high, and indeed the MCMC has not at all converged...

```
> plot(MCMC,Full=TRUE)
```

Better convergence is achieved by the adaptive metropolis, updating the proposal every 100 runs

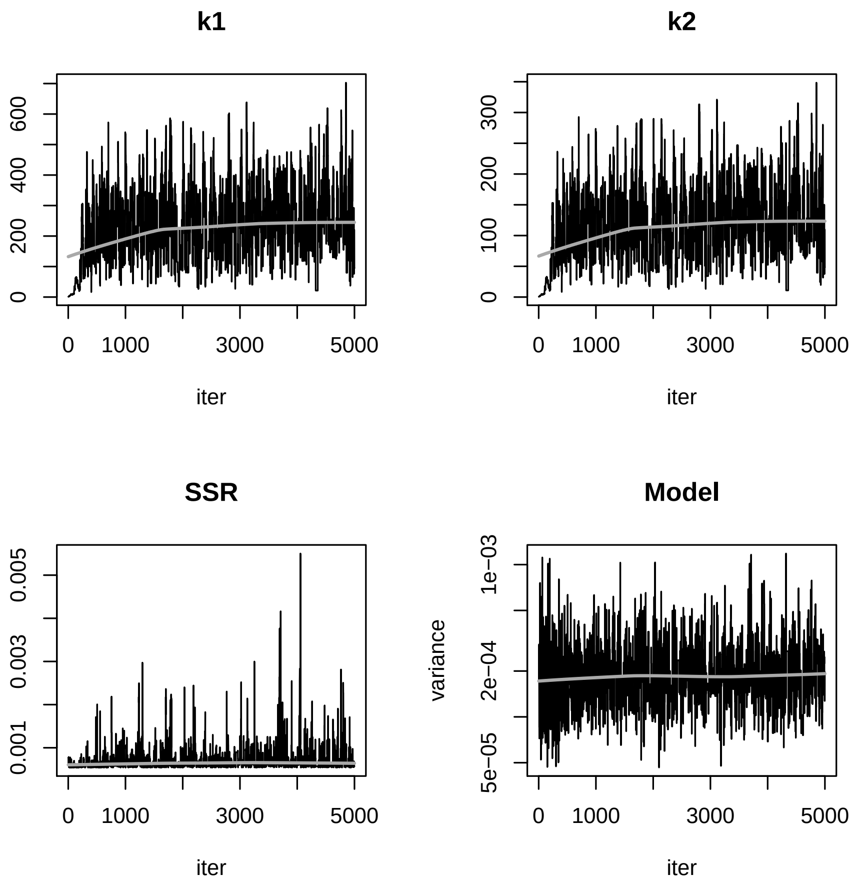

Figure 10: Adaptive Metropolis MCMC of the chemical model - see text for R-code

```
> MCMC2<- modMCMC(f=residual, p=Fit$par, jump=Cov, updatecov=100, lower=c(0,0),
+                   var0=mse, wvar0=1, prior=Prior,niter=5000)    #
```

```
number of accepted runs: 1750 out of 5000 (35%)
```

```
> plot(MCMC2,Full=TRUE)
```

The correlation between the two parameters is clear:

```
> pairs(MCMC2)
```

5. Oxygen in the sediment

5.1. the model

This is a simple model of oxygen, diffusing along a spatial gradient, with imposed upper

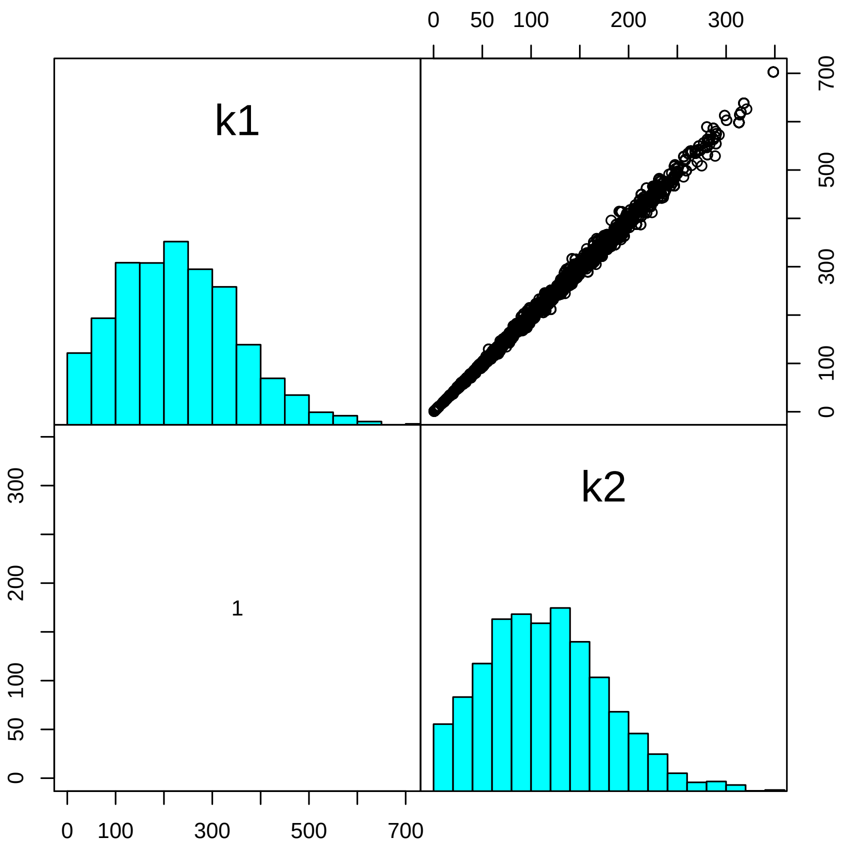

Figure 11: Pairs plot of the Adaptive Metropolis MCMC of the chemical model - see text for R-code

boundary concentration oxygen is consumed at maximal fixed rate, and including a monod limitation.

The constitutive equations are:

$$\frac{\partial O_2}{\partial t} = -\frac{\partial Flux}{\partial x} - cons \cdot \frac{O_2}{O_2 + k_s}$$
$$Flux = -D \cdot \frac{\partial O_2}{\partial x}$$
$$O_2(x = 0) = upO2$$

```
> par(mfrow=c(2,2))
> require(FME)
```

First the model parameters are defined...

```
> pars <- c(upO2=360,  # concentration at upper boundary, mmolO2/m3
+           cons=80,   # consumption rate, mmolO2/m3/day
+           ks=1,      # O2 half-saturation ct, mmolO2/m3
+           D=1)       # diffusion coefficient, cm2/d
```

Next the sediment is vertically subdivided into 100 grid cells, each 0.05 cm thick.

```
> n  <- 100                        # nr grid points
> dx <- 0.05   #cm
> dX <- c(dx/2,rep(dx,n-1),dx/2)   # dispersion distances; half dx near boundaries
> X  <- seq(dx/2,len=n,by=dx)      # distance from upper interface at middle of box
```

The model function takes as input the parameter values and returns the steady-state condition of oxygen. Function `steady.band` from package **rootSolve** ((Soetaert 2008)) does this in a very efficient way (see (Soetaert and Herman 2009)).

```
> O2fun <- function(pars)
+ {
+   derivs<-function(t,O2,pars)
+   {
+   with (as.list(pars),{
+
+     Flux <- -D* diff(c(upO2,O2,O2[n]))/dX
+     dO2  <- -diff(Flux)/dx-cons*O2/(O2+ks)
+
+     return(list(dO2,UpFlux = Flux[1],LowFlux = Flux[n+1]))
+   })
+  }
+
+  # Solve the steady-state conditions of the model
+  ox <- steady.band(y=runif(n),func=derivs,parms=pars,nspec=1,positive=TRUE)
+  data.frame(X=X,O2=ox$y)
+ }
```

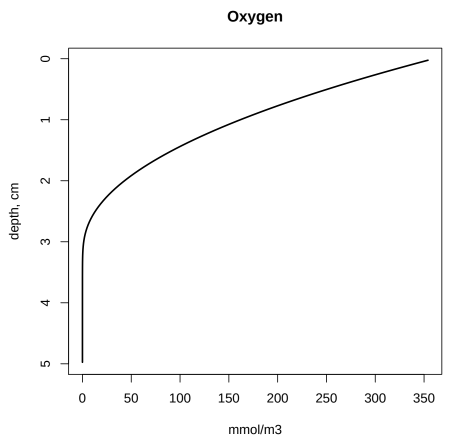

Figure 12: The modeled oxygen profile - see text for R-code

The model is run

```
> ox<-O2fun(pars)
```

and the results plotted...

```
> plot(ox$O2,ox$X,ylim=rev(range(X)),xlab="mmol/m3",
+      main="Oxygen", ylab="depth, cm",type="l",lwd=2)
```

5.2. Global sensitivity analysis : Sensitivity ranges

The sensitivity of the oxygen profile to parameter `cons`, the consumption rate is estimated. We assume a normally distributed parameter, with mean = 80 (`parMean`), and a variance=100 (`parCovar`). The model is run 100 times (`num`).

```
> print(system.time(
+ Sens2 <- sensRange(parms=pars,func=O2fun,dist="norm",
+              num=100,parMean=c(cons=80),parCovar=100)
+ ))

   user  system elapsed 
   1.01    0.00    1.02 
```

The results can be plotted in two ways:

```
> par(mfrow=c(1,2))
> plot(Sens2,xyswap=TRUE,xlab= "O2",
+      ylab="depth, cm",main="Sensitivity runs")
> plot(summary(Sens2),xyswap=TRUE,xlab= "O2",
+      ylab="depth, cm",main="Sensitivity ranges")
> par(mfrow=c(1,1))
```

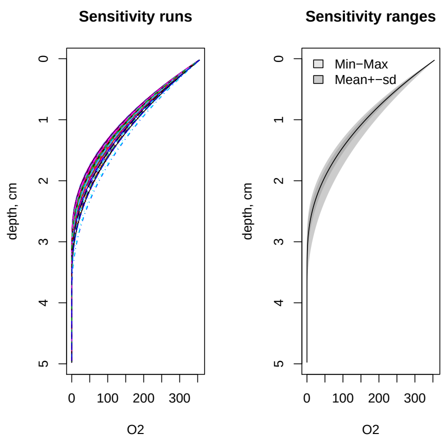

Figure 13: Results of the sensitivity run - left: all model runs, right: summary - see text for R-code

5.3. Local sensitivity analysis : Sensitivity functions

Local sensitivity analsysis starts by calculating the sensitivity functions

```
> O2sens <- sensFun(func=O2fun,parms=pars)
```

The summary of these functions gives information about which parameters have the largest effect (univariate sensitivity):

```
> summary(O2sens)
```

```
     value scale  L1   L2 Mean     Min     Max   N
upO2   360   360 6.3 0.77  6.3  1.0e+00 11.5242 100
cons    80    80 8.6 1.21 -8.6 -2.3e+01 -0.0084 100
ks       1     1 1.4 0.26  1.4  1.1e-04  7.5730 100
D        1     1 8.1 1.14  8.1  8.4e-03 22.0312 100
```

In bivariate sensitivity the pair-wise relationship and the correlation is estimated and/or plotted:

```
> pairs(O2sens)
```

```
> cor(O2sens[,-(1:2)])
```

```
           upO2       cons         ks          D
upO2  1.0000000 -0.9648628  0.7272077  0.9666205
cons -0.9648628  1.0000000 -0.8705570 -0.9998597
ks    0.7272077 -0.8705570  1.0000000  0.8709830
D     0.9666205 -0.9998597  0.8709830  1.0000000
```

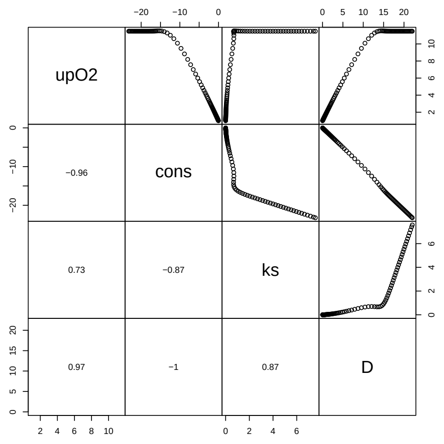

Figure 14: pairs plot - see text for R-code

Multivariate sensitivity is done by estimating the collinearity between parameter sets (Brun, Reichert, and Kunsch 2001).

```
> Coll <- collin(O2sens)
> Coll
```

```
   upO2 cons ks D N collinearity
1     1    1  0 0 2          6.1
2     1    0  1 0 2          2.2
3     1    0  0 1 2          6.3
4     0    1  1 0 2          3.2
5     0    1  0 1 2        107.6
6     0    0  1 1 2          3.2
7     1    1  1 0 3         12.1
8     1    1  0 1 3        131.1
9     1    0  1 1 3         13.4
10    0    1  1 1 3        108.0
11    1    1  1 1 4        191.5
```

```
> plot(Coll,log="y")
```

5.4. Fitting the model to the data

Assume both the oxygen flux at the upper interface and a vertical profile of oxygen has been measured.

These are the data:

```
> O2dat <- data.frame(x=seq(0.1,3.5,by=0.1),
+     y = c(279,260,256,220,200,203,189,179,165,140,138,127,116,
```

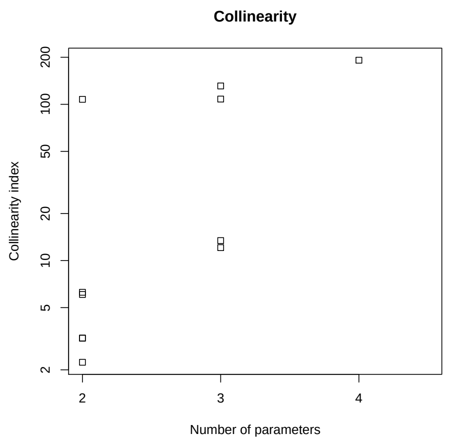

Figure 15: collinearity - see text for R-code

```
+           109,92,87,78,72,62,55,49,43,35,32,27,20,15,15,10,8,5,3,2,1,0))
> O2depth <- cbind(name="O2",O2dat)        # oxygen versus depth
> O2flux  <- c(UpFlux=170)                 # measured flux
```

First a function is defined that returns only the required model output.

```
> O2fun2 <- function(pars)
+ {
+   derivs<-function(t,O2,pars)
+   {
+   with (as.list(pars),{
+
+     Flux <- -D*diff(c(upO2,O2,O2[n]))/dX
+     dO2  <- -diff(Flux)/dx-cons*O2/(O2+ks)
+
+     return(list(dO2,UpFlux = Flux[1],LowFlux = Flux[n+1]))
+     })
+   }
+
+  ox <- steady.band(y=runif(n),func=derivs,parms=pars,nspec=1,
+                    positive=TRUE,rtol=1e-8,atol=1e-10)
+
+  list(data.frame(x=X,O2=ox$y),
+       UpFlux=ox$UpFlux)
+ }
```

The function used in the fitting algorithm returns an instance of type `modCost`. This is created by calling function `modCost` twice. First with the modeled oxygen profile, then with the modeled flux.

```
> Objective <- function (P)
```

```
+ {
+  Pars <- pars
+  Pars[names(P)]<-P
+  modO2 <- O2fun2(Pars)
+
+  # Model cost: first the oxygen profile
+  Cost  <- modCost(obs=O2depth,model=modO2[[1]],x="x",y="y")
+
+  # then the flux
+  modFl <- c(UpFlux=modO2$UpFlux)
+  Cost  <- modCost(obs=O2flux,model=modFl,x=NULL,cost=Cost)
+
+  return(Cost)
+ }
```

We first estimate the identifiability of the parameters, given the data:

```
> print(system.time(
+ sF<-sensFun(Objective,parms=pars)
+ ))

   user  system elapsed
   0.13    0.00    0.12

> summary(sF)

     value scale   L1   L2  Mean      Min  Max  N
upO2   360   360 4.25 0.97  4.25   0.5069 13.3 36
cons    80    80 3.68 0.99 -3.65 -15.3722  0.5 36
ks       1     1 0.40 0.14  0.40  -0.0069  3.1 36
D        1     1 3.68 0.99  3.68   0.0342 15.4 36

> collin(sF)

   upO2 cons ks D N collinearity
1     1    1  0 0 2          8.6
2     1    0  1 0 2          3.1
3     1    0  0 1 2          8.7
4     0    1  1 0 2          4.2
5     0    1  0 1 2         50.6
6     0    0  1 1 2          4.2
7     1    1  1 0 3         14.2
8     1    1  0 1 3         50.8
9     1    0  1 1 3         14.7
10    0    1  1 1 3         50.6
11    1    1  1 1 4         51.0
```

The collinearity of the full set is too high, but as the oxygen diffusion coefficient is well known, it is left out of the fitting. The combination of the three remaining parameters has a low enough collinearity to enable automatic fitting. The parameters are constrained to be >0

```
> collin(sF,parset=c("upO2","cons","ks"))

  upO2 cons ks D N collinearity
1    1    1  1 0 3           14

> print(system.time(
+ Fit<-modFit(p=c(upO2=360,cons=80,ks=1),
+                   f=Objective,lower=c(0,0,0))
+                   ))

   user  system elapsed
   0.92    0.00    0.92

> (SFit<-summary(Fit))

Parameters:
     Estimate Std. Error t value Pr(>|t|)
upO2  292.937      2.104 139.242   <2e-16 ***
cons   49.687      2.367  20.991   <2e-16 ***
ks      1.297      1.363   0.951    0.348
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Residual standard error: 4.401 on 33 degrees of freedom

Parameter correlation:
       upO2   cons     ks
upO2 1.0000 0.5791 0.2976
cons 0.5791 1.0000 0.9012
ks   0.2976 0.9012 1.0000
```

We next plot the residuals

```
> plot(Objective(Fit$par),xlab="depth",ylab="",main="residual",legpos="top")
```

and show the best-fit model

```
> Pars <- pars
> Pars[names(Fit$par)]<- Fit$par
> modO2 <- O2fun(Pars)

> plot(O2depth$y,O2depth$x,ylim=rev(range(O2depth$x)),pch=18,
+      main="Oxygen-fitted", xlab="mmol/m3",ylab="depth, cm")
> lines(modO2$O2,modO2$X)
```

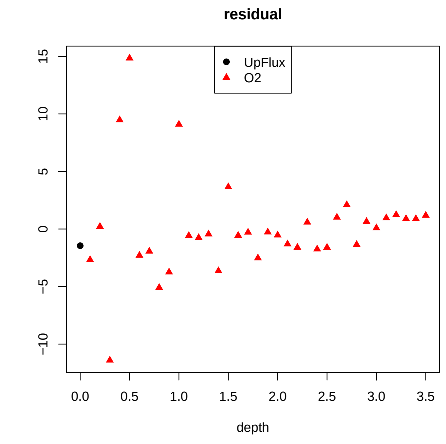

Figure 16: residuals - see text for R-code

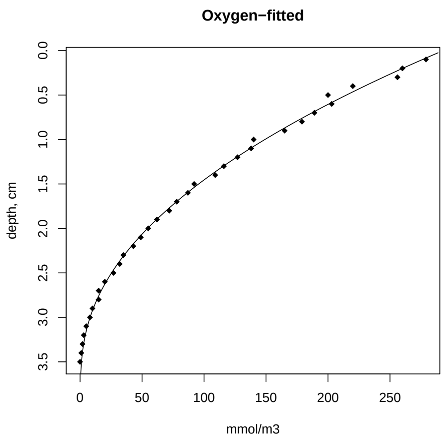

Figure 17: Best fit model - see text for R-code

5.5. Run MCMC

We use the parameter covariances of previous fit to update parameters, while the mean squared residual of the fit is use as prior fo the model variance.

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

We run an adaptive metropolis, making sure that ks does not become negative...

```
> print(system.time(
+ MCMC <- modMCMC(f=Objective,p=Fit$par,jump=Covar,niter=1000,ntrydr=2,
+                 var0=s2prior,wvar0=1,updatecov=100,lower=c(NA,NA,0))
+ ))

number of accepted runs: 613 out of 1000 (61.3%)
   user  system elapsed 
  27.41    0.00   27.47 
```

```
> MCMC$count

     dr_steps     Alfasteps  num_accepted num_covupdate 
          611          1833           613            10 
```

Plotting the results is similar to previous cases.

```
> plot(MCMC,Full=TRUE)
```

```
> hist(MCMC,Full=TRUE)
```

```
> pairs(MCMC,Full=TRUE)
```

or summaries can be created:

```
> summary(MCMC)

           upO2      cons           ks          sig
mean 294.985847  57.98132   9.34952533   655.891733
sd     7.853813  15.45276  16.06166699  3542.181482
min  276.917004  42.80188   0.01689210     1.613269
max  335.063812 176.91964 116.79736104 74434.659496
q025 290.944190  49.30220   1.39707636    16.070616
q050 293.171259  51.91314   2.97249258    44.098827
q075 295.690293  59.67832   7.28115214   221.496223
```

```
> cor(MCMC$pars)
```

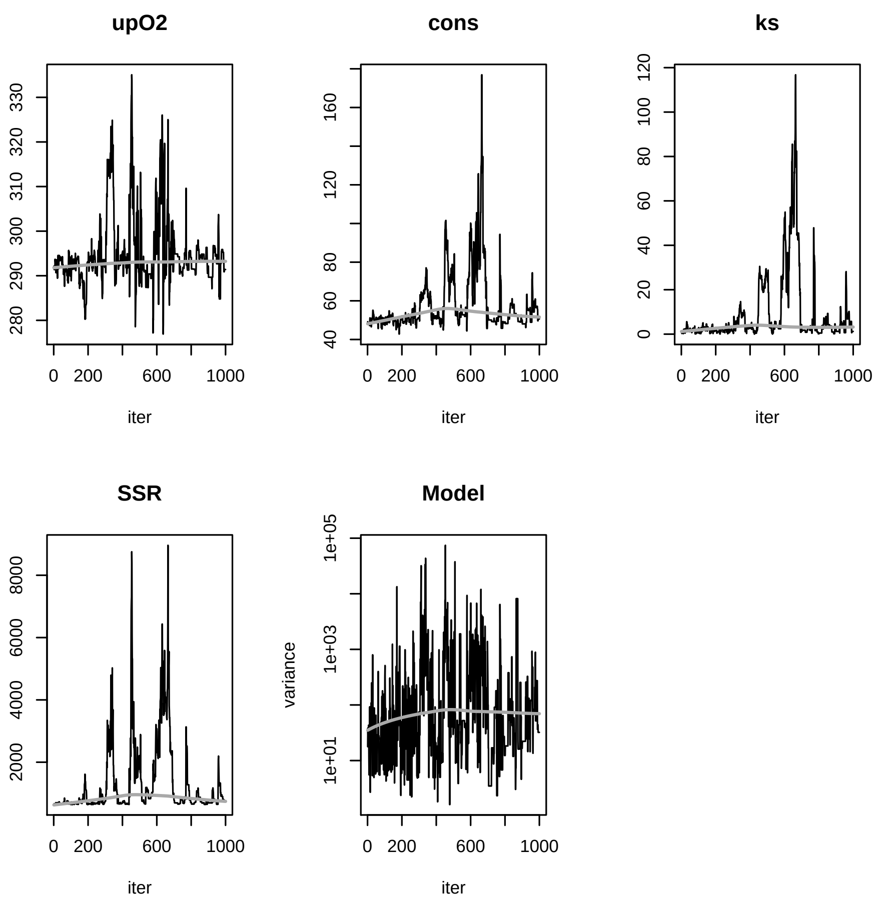

Figure 18: MCMC plot results - see text for R-code

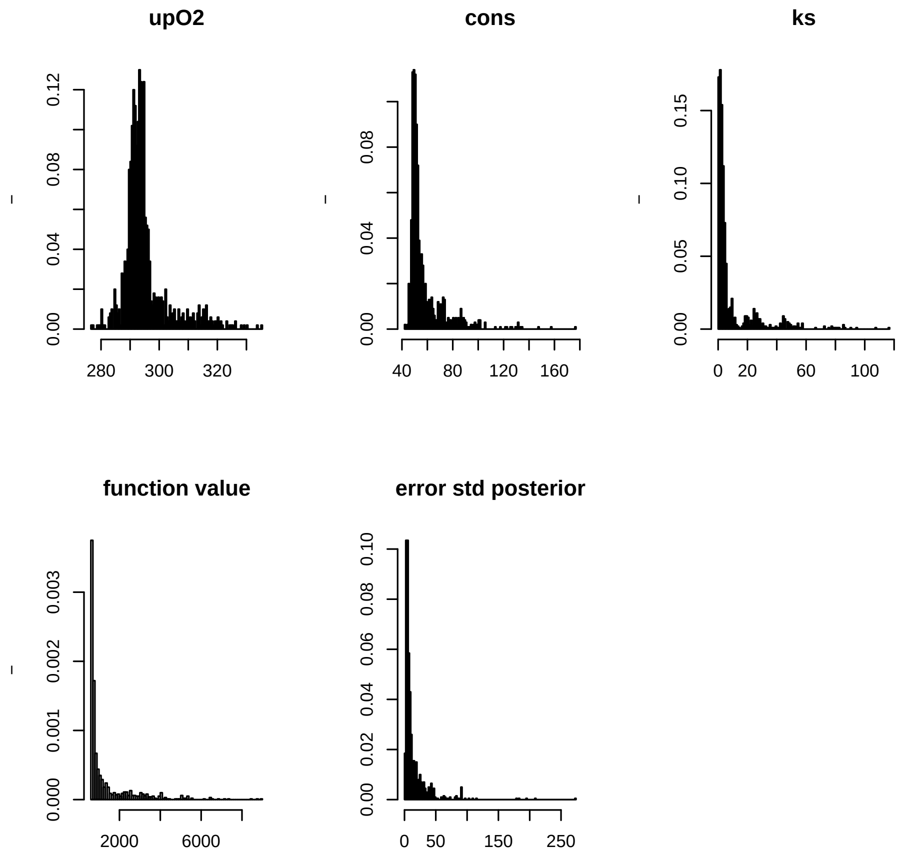

Figure 19: MCMC histogram results - see text for R-code

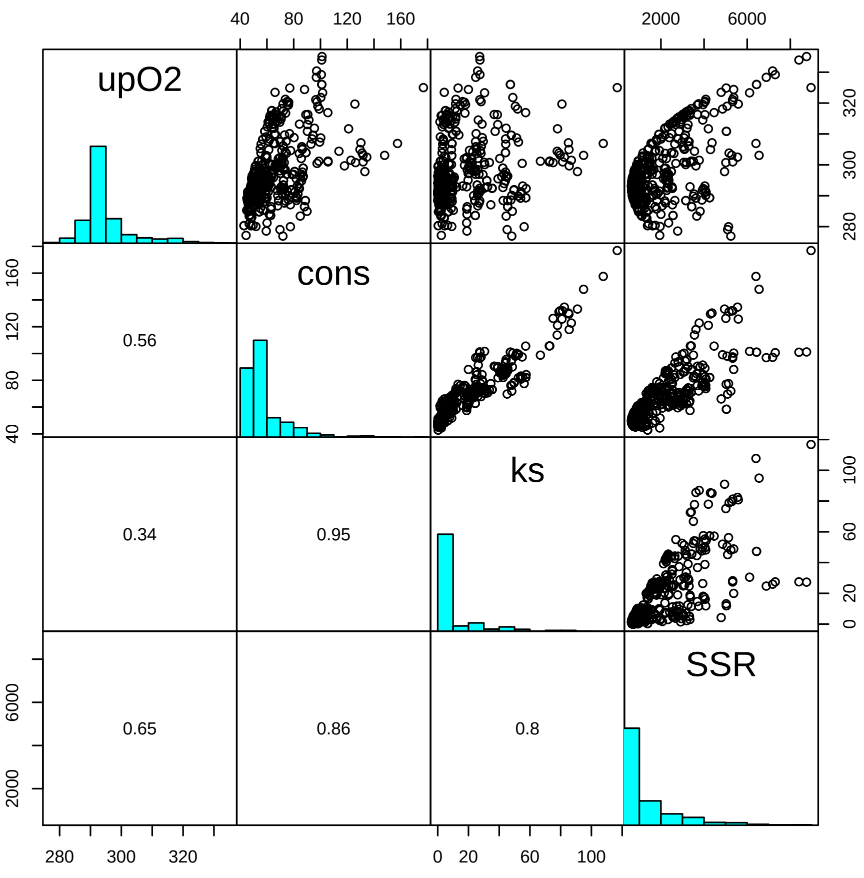

Figure 20: MCMC pairs plot - see text for R-code

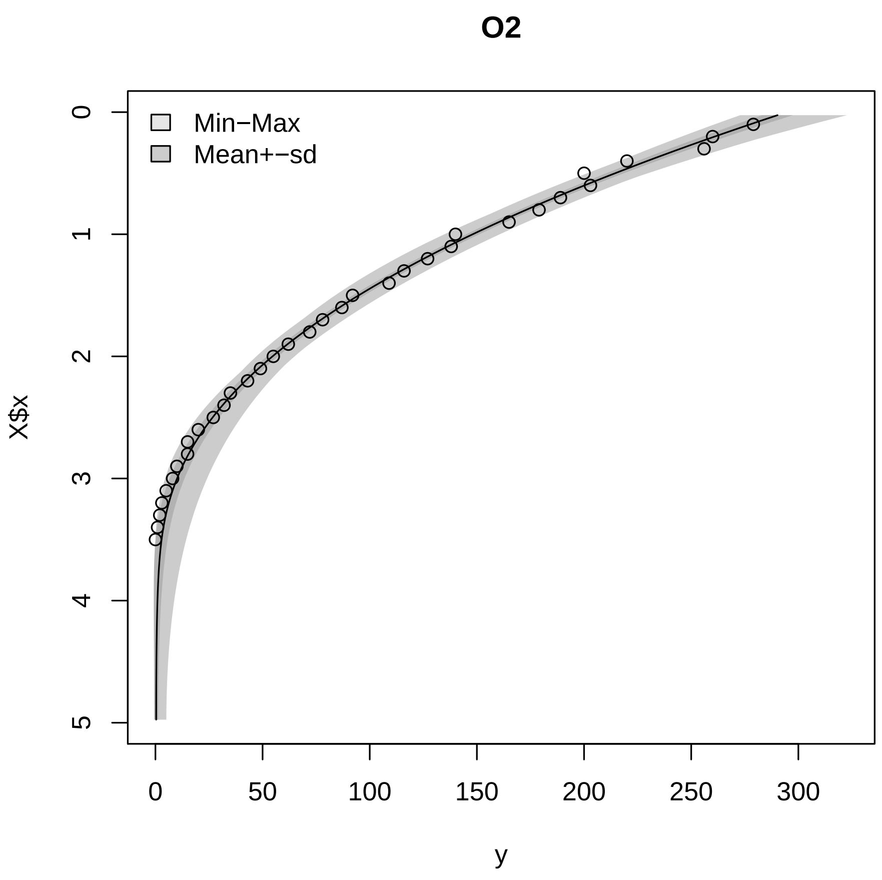

Figure 21: MCMC range plot - see text for R-code

```
          upO2      cons        ks
upO2 1.0000000 0.5638086 0.3388768
cons 0.5638086 1.0000000 0.9456164
ks   0.3388768 0.9456164 1.0000000
```

Note: we pass to sensRange the full parameter vector (`parms`) and the parameters sampled during the MCMC (`parInput`).

```
> plot(summary(sensRange(parms=pars,parInput=MCMC$par,f=O2fun,num=500)),xyswap=TRUE)
> points(O2depth$y,O2depth$x)
```

6. finally

This vignette is a Sweave (Leisch 2002) translation of part of the **FME** examples.

References

Brun R, Reichert P, Kunsch H (2001). "Practical identifiability analysis of large environmental simulation models." *WATER RESOURCES RESEARCH*, **37**(4), 1015–1030.

Gelman A, Varlin JB, Stern HS, Rubin DB (2004). *Bayesian Data Analysis, second edition.* Chapman and Hall / CRC, Boca Raton.

Haario H, Laine M, Mira A, Saksman E (2006). "DRAM: efficient adaptive MCMC." *Statistical Computing*, **16**, 339–354.

Laine M (2008). *Adaptive MCMC methods with applications in environmental and geophysical models.* Finnish meteorological institute contributions n0 69 -ISBN 978-951-697-662-7.

Leisch F (2002). "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), "Compstat 2002 - Proceedings in Computational Statistics," pp. 575–580. Physica Verlag, Heidelberg. ISBN 3-7908-1517-9, URL `http://www.stat.uni-muenchen.de/~leisch/Sweave`.

Soetaert K (2008). *rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations.* R package version 1.2.

Soetaert K, Herman PMJ (2009). *A Practical Guide to Ecological Modelling. Using R as a Simulation Platform.* Springer. ISBN 978-1-4020-8623-6.

Affiliation:
Karline Soetaert
Centre for Estuarine and Marine Ecology (CEME)
Netherlands Institute of Ecology (NIOO)
4401 NT Yerseke, Netherlands
E-mail: `k.soetaert@nioo.knaw.nl`
URL: `http://www.nioo.knaw.nl/ppages/ksoetaert`