

# R Package **FME** : Tests of the Markov Chain Monte Carlo Implementation

**Karline Soetaert**  
NIOZ Yerseke  
The Netherlands

**Marko Laine**  
Finnish Meteorological Institute  
Finland

---

## Abstract

This vignette tests the Markov chain Monte Carlo (MCMC) implementation of Rpackage **FME** (Soetaert and Petzoldt 2010).

It includes the delayed rejection and adaptive Metropolis algorithm (Haario, Laine, Mira, and Saksman 2006)

*Keywords:* Markov chain Monte Carlo, delayed rejection, adaptive Metropolis, MCMC, DRAM, R.

---

## 1. Introduction

Function `modMCMC` from package **FME** (Soetaert and Petzoldt 2010) implements a Markov chain Monte Carlo (MCMC) algorithm using a delayed rejection and adaptive Metropolis procedure (Haario *et al.* 2006).

In this vignette , the DRAM MCMC function is tested on several functions.

- Sampling from a normal distribution, using different priors
- Sampling from a log-normal distribution
- Sampling from a curvilinear ("banana") function (Laine 2008)
- A simple chemical model, fitted to a data series (Haario *et al.* 2006)
- A nonlinear monod function, fitted to a data series.

Other examples of **FME** functions (including `modMCMC`) are in the following vignettes:

- "FMEdyna", FMEs functions applied to a dynamic ordinary differential equation model
- "FMEsteady", applied to a steady-state solution of a partial differential equation

## 2. Function modMCMC

### 2.1. The Markov chain Monte Carlo method

The implemented MCMC method is designed to be applied to nonlinear models, and taking into account both the uncertainty in the model parameters and in the model output.

Consider observations  $y$  and a model  $f$ , that depend on parameters  $\theta$  and independent variables  $x$ . Assuming additive, independent Gaussian errors with an unknown variance  $\sigma^2$ :

$$y = f(x, \theta) + \xi$$

where

$$\xi \sim N(0, I\sigma^2)$$

For simplicity we assume that the prior distribution for  $\theta$  is Gaussian:

$$\theta_i \sim N(v_i, \mu_i)$$

while for the reciprocal of the error variance  $1/\sigma^2$ , a Gamma distribution is used as a prior:

$$p(\sigma^{-2}) \sim \Gamma\left(\frac{n_0}{2}, \frac{n_0}{2} S_0^2\right).$$

The posterior for the parameters will be estimated as:

$$p(\theta|y, \sigma^2) \propto \exp\left(-0.5\left(\frac{SS(\theta)}{\sigma^2} + SS_{pri}(\theta)\right)\right)$$

and where  $\sigma^2$  is the error variance, SS is the sum of squares function

$$SS(\theta) = \sum (y_i - f(\theta)_i)^2$$

and

$$SS_{pri}(\theta) = \sum_i \left(\frac{\theta_i - v_i}{\mu_i}\right)^2.$$

In the above, the sum of squares functions ( $SS(\theta)$ ) are defined for Gaussian likelihoods. For a general likelihood function the sum-of-squares corresponds to twice the log likelihood,

$$SS(\theta) = -2\log(p(y|\theta))$$

. This is how the function value ( $\mathbf{f}$ , see below) should be specified.

Similarly, to obtain a general non-Gaussian prior for the parameters  $\theta$  (i.e.  $SS_{pri}(\theta)$ ) minus twice the log of the prior density needs to be calculated.

If non-informative priors are used, then  $SS_{pri}(\theta)=0$ .

### 2.2. Arguments to function modMCMC

The default input to modMCMC is:

```
modMCMC(f, p, ..., jump=NULL, lower=-Inf, upper=+Inf, prior=NULL,
        var0 = NULL, wvar0 = NULL, n0= NULL, niter=1000, outputlength = niter,
        burninlength=0, updatecov=niter, covscale = 2.4^2/length(p),
        ntrydr=1, drscale=NULL, verbose=TRUE)
```

with the following arguments (see help page for more information):

- `f`, the sum-of-squares function to be evaluated,  $SS(\theta)$
- `p`, the initial values of the parameters  $\theta$  to be sampled
- `...`, additional arguments passed to function `f`
- `jump`, the proposal distribution (this generates new parameter values)
- `prior`, the parameter prior,  $SS_{pri}(\theta)$
- `var0`, `wvar0`, `n0`, the initial model variance and weight of the initial model variance, where `n0=wvar0*n`, `n`=number of observations.
- `lower`, `upper`, lower and upper bounds of the parameters
- `niter`, `outputlength`, `burninlength`, the total number of iterations, the number of iterations kept in output, and the number of initial iterations removed from the output.
- `updatecov`, `covscale`, arguments for the adaptation of the proposal covariance matrix (AM-part of the algorithm).
- `ntrydr`, `drscale`, delayed rejection (DR) arguments.

### 3. Sampling from a normal distribution

In the first example, function `modMCMC` is used to sample from a normal distribution, with mean = 10 and standard deviation = 1. We use this simple example mainly for testing the algorithm, and to show various ways of defining parameter priors.

In this example, the error variance of the model is 0 (the default).

We write a function, `Nfun` that takes as input the parameter value and that returns 2 times the log of the normal likelihood.

```
> mu <- 10
> std <- 1
> Nfun <- function(p)
+   -2*log(dnorm(p, mean = mu, sd = std))
```

The proposal covariance is assumed to be 5.

#### 3.1. Noninformative prior

In the first run, a noninformative prior parameter distribution is used. 2000 iterations are produced; the initial parameter value is taken as 9.5.

```
> MCMC <- modMCMC (f = Nfun, p = 9.5, niter = 2000, jump = 5)
```

```
number of accepted runs: 466 out of 2000 (23.3%)
```

It is more efficient to update the proposal distribution, e.g. every 10 iterations:

```
> MCMC <- modMCMC (f = Nfun, p = 9.5, niter = 2000, jump = 5, updatecov = 10)
```

```
number of accepted runs: 1396 out of 2000 (69.8%)
```

```
> summary(MCMC)
```

```

              p1
mean 10.070124
sd   1.007911
min  7.518412
max  13.196895
q025 9.351091
q050 10.034233
q075 10.760927
```

#### 3.2. Noninformative prior, lower bound imposed

In the second run, the sampled parameters are restricted to be  $> 9$  (`lower=9`):

```
> MCMC2 <- modMCMC (f = Nfun, p = 9.5, lower = 9, niter = 2000, jump = 5,
+   updatecov = 10)
```

number of accepted runs: 1430 out of 2000 (71.5%)

```
> summary(MCMC2)
```

```

                p1
mean 10.2756619
sd   0.8004843
min  9.0010938
max  12.9663015
q025 9.6333582
q050 10.1917549
q075 10.7735331
```

### 3.3. A normally distributed prior

Finally, it is assumed that the prior for the model parameter is itself a normal distribution, with mean 8 and standard deviation 1:  $pri(\theta) \sim N(8, 1)$ .

The posterior for this problem is a normal distribution with mean = 9, standard deviation of 0.707.

```
> pri <- function(p) -2*log(dnorm(p, 8, 1))
> MCMC3 <- modMCMC (f = Nfun, p = 9.5, niter = 2000, jump = 5,
+   updatecov = 10, prior = pri)
```

number of accepted runs: 1399 out of 2000 (69.95%)

```
> summary(MCMC3)
```

```

                p1
mean  9.0055525
sd    0.6934714
min   6.4201943
max   11.4132288
q025  8.5756074
q050  9.0358937
q075  9.4460343
```

The number of accepted runs is increased by toggling on delayed rejection; at most 2 delayed rejections steps are tried (`ntrydr=2`):

```
> summary(MCMC4 <- modMCMC(f = Nfun, p = 1, niter = 2000, jump = 5,
+   updatecov = 10, prior = pri, ntrydr = 2))
```

```
number of accepted runs: 1927 out of 2000 (96.35%)
```

```
      p1
mean  8.9962693
sd    0.6986521
min   4.7873685
max   11.2930606
q025  8.5359626
q050  8.9857000
q075  9.4311402
```

```
> MCMC4$count
```

dr_steps	Alfasteps	num_accepted	num_covupdate
627	1881	1927	181

Finally, we plot a histogram of the three MCMC runs, and end by plotting the trace of the last run (figure 1).

```
> par(mfrow = c(2,2))
> hist(MCMC$pars, xlab="x", freq = FALSE, main = "unconstrained", xlim = c(6, 14))
> hist(MCMC2$pars, xlab="x", freq = FALSE, main = "x>9", xlim = c(6, 14))
> hist(MCMC3$pars, xlab="x", freq = FALSE, main = "pri(x)~N(8,1)", xlim = c(6, 14))
> plot(MCMC3, mfrow = NULL, main = "AM")
> mtext(outer = TRUE, line = -1.5, "N(10,1)", cex = 1.25)
```

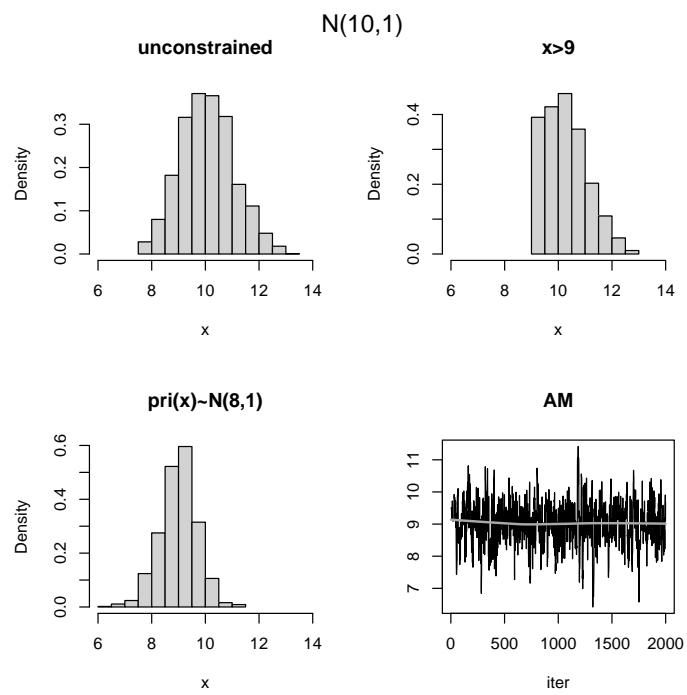


Figure 1: Simulated draws of a normal distribution ( $N(10,1)$ ) with different prior parameter distributions - see text for R-code

## 4. Sampling from a lognormal distribution

In the second example, function `modMCMC` is used to sample from a 3-variate log-normal distribution, with mean = 1,2,3, and standard deviation = 0.1.

We write a function that has as input the parameter values (a 3-valued vector) and that returns 2 times the lognormal likelihood.

```
> mu <- 1:4
> std <- 1
> NL <- function(p) {
+   -2*sum(log(dlnorm(p, mean = mu, sd = std)))
+ }
```

The proposal covariance is assumed to be the identity matrix with a variance of 5. The simulated chain is of length 10000 (`niter`), but only 1000 are kept in the output (`outputlength`).

```
> MCMC1 <- modMCMC (f = NL, p = rep(1, 4), niter = 10000,
+   outputlength = 1000, jump = 5)
```

number of accepted runs: 3519 out of 10000 (35.19%)

Convergence is tested by plotting the trace; in the first run convergence is not good (figure 2)

```
> plot(MCMC1)
```

The number of accepted runs is increased by updating the jump covariance matrix every 100 runs and toggling on delayed rejection.

```
> MCMC1 <- modMCMC (f = NL, p = rep(1, 4), niter = 5000,
+   outputlength = 1000, jump = 5, updatecov = 100, ntrydr = 2)
```

number of accepted runs: 2889 out of 5000 (57.78%)

Convergence of the chain is checked (figure 3).

```
> plot(MCMC1)
```

The histograms show the posterior densities (figure 4).

```
> hist(MCMC1)
```

```
> MCMC1$pars <- log(MCMC1$pars)
> summary(MCMC1)
```

	p1	p2	p3	p4
mean	0.9431005	1.9532548	3.2036788	4.0976091
sd	0.9170056	0.9699925	1.0623289	1.0238266
min	-2.0315258	-1.3889940	-0.3464631	0.2111235
max	3.3563106	4.1178055	5.7962178	6.7109035
q025	0.3058258	1.3379801	2.4872130	3.3954041
q050	0.9331448	1.9814548	3.1746040	4.1432607
q075	1.5805549	2.6419000	3.9442604	4.8022651



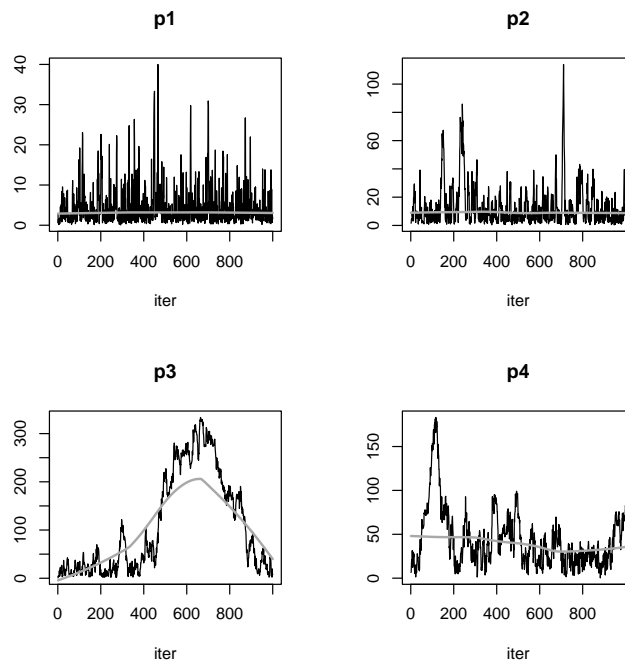


Figure 2: The trace of the log normal distribution -Metropolis algorithm - see text for R-code

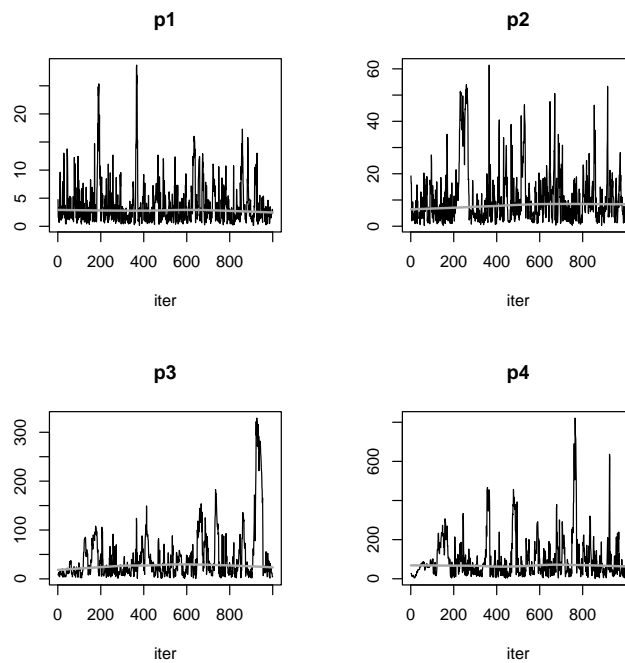


Figure 3: The trace of the log normal distribution - adaptive Metropolis - see text for R-code

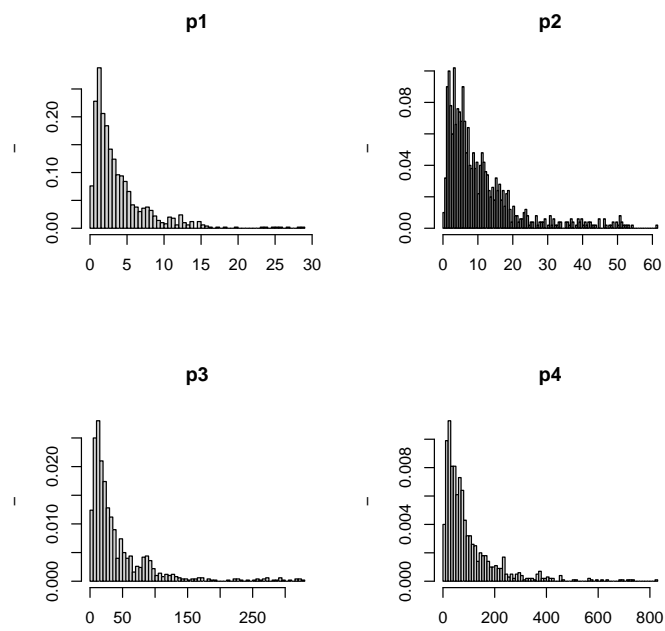


Figure 4: The histograms of the log normal distributed samples - see text for R-code

## 5. The banana

### 5.1. 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  $\tau$  with unity variances and covariance of 0.9:

$$\tau = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}.$$

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

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

### 5.2. R-implementation

First the banana function is defined.

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

We need a function that estimates the probability of a multivariately 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  $-2 \cdot \log$  (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 2000 (`niter`). The `modMCMC` function prints the % of accepted runs. More information is in item `count` of its return element.

### 5.3. Metropolis Hastings algorithm

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 = 2000)
```

number of accepted runs: 196 out of 2000 (9.8%)

```
> MCMC$count
```

dr_steps	Alfasteps	num_accepted	num_covupdate
0	0	196	0

### 5.4. Adaptive Metropolis algorithm

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 = 2000)
```

number of accepted runs: 369 out of 2000 (18.45%)

```
> MCMC2$count
```

dr_steps	Alfasteps	num_accepted	num_covupdate
0	0	369	19

### 5.5. Delayed Rejection algorithm

Then the Metropolis algorithm with delayed rejection (DR) is applied; upon rejection one next parameter candidate 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 = 2000)
```

number of accepted runs: 1054 out of 2000 (52.7%)

```
> MCMC3$count
```

dr_steps	Alfasteps	num_accepted	num_covupdate
1799	5397	1054	0

`dr_steps` denotes the number of delayed rejection steps; `Alfasteps` is the number of times the algorithm has entered the acceptance function for delayed rejection.

## 5.6. Delayed Rejection Adaptive Metropolis algorithm

Finally the adaptive Metropolis with delayed rejection (DRAM) is used. (Here we also estimate the elapsed CPU time, in seconds - `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 = 2000)
+ ))
```

```
number of accepted runs: 1251 out of 2000 (62.55%)
```

```
  user  system elapsed
 0.41   0.01   0.43
```

```
> MCMC4$count
```

```
      dr_steps      Alfasteps  num_accepted  num_covupdate
      1700           5100         1251           19
```

We plot the generated chains for both parameters and for the four runs in one plot (figure 5). 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.

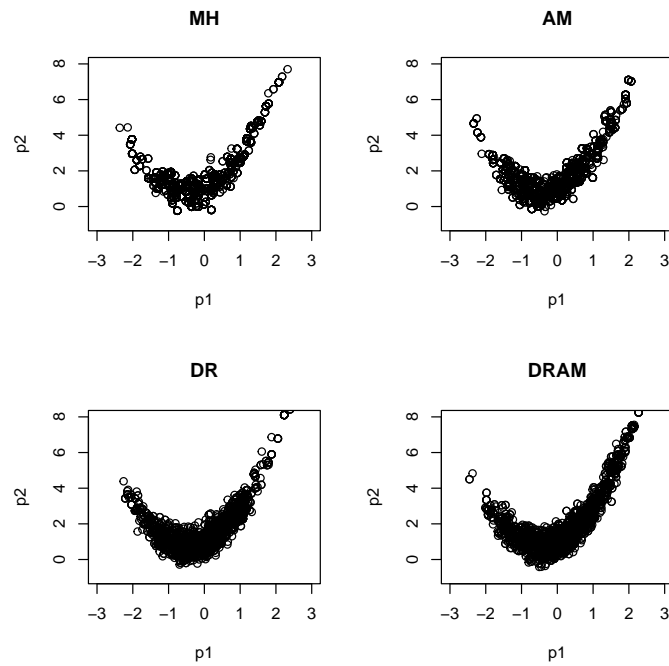


Figure 5: The bananas - see text for R-code

```

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

[1] 0.08365834 0.07091574

> apply(trans, 2, sd) # was:1

[1] 0.9403218 0.9798653

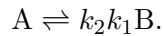
> cov(trans)          # 0.9 off-diagonal

      [,1]      [,2]
[1,] 0.8842051 0.8199264
[2,] 0.8199264 0.9601359

```

## 6. 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:



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

The ODE system is written as:

$$\begin{aligned} \frac{dA}{dt} &= -k_1 \cdot A + k_2 \cdot B \\ \frac{dB}{dt} &= +k_1 \cdot A - k_2 \cdot B, \end{aligned}$$

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).

### 6.1. Implementation in R

First a function is defined that takes as input the parameters ( $\mathbf{k}$ ) 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 impose 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 sum of squares function (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 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01707 on 3 degrees of freedom

Parameter correlation:

	k1	k2
k1	1.0000	0.9959
k2	0.9959	1.0000

Here the observations have additive independent Gaussian errors with unknown variance  $\sigma^2$ . As explained above, the error variance is treated as a 'nuisance' parameter, and a prior distribution should be specified as a Gamma-type distribution for its inverse.

The residual error of the fit (`sF$modVariance`) is used as initial model variance (argument `var0`), the scaled covariance matrix of the fit (`sF$cov.scaled`) 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 MCMC is initiated with the best-fit parameters (`Fit$par`); the parameters are restricted to be positive numbers (`lower`).

```
> 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 = 2000)
+ ))
```

number of accepted runs: 1864 out of 2000 (93.2%)

user	system	elapsed
0.5	0.0	0.5



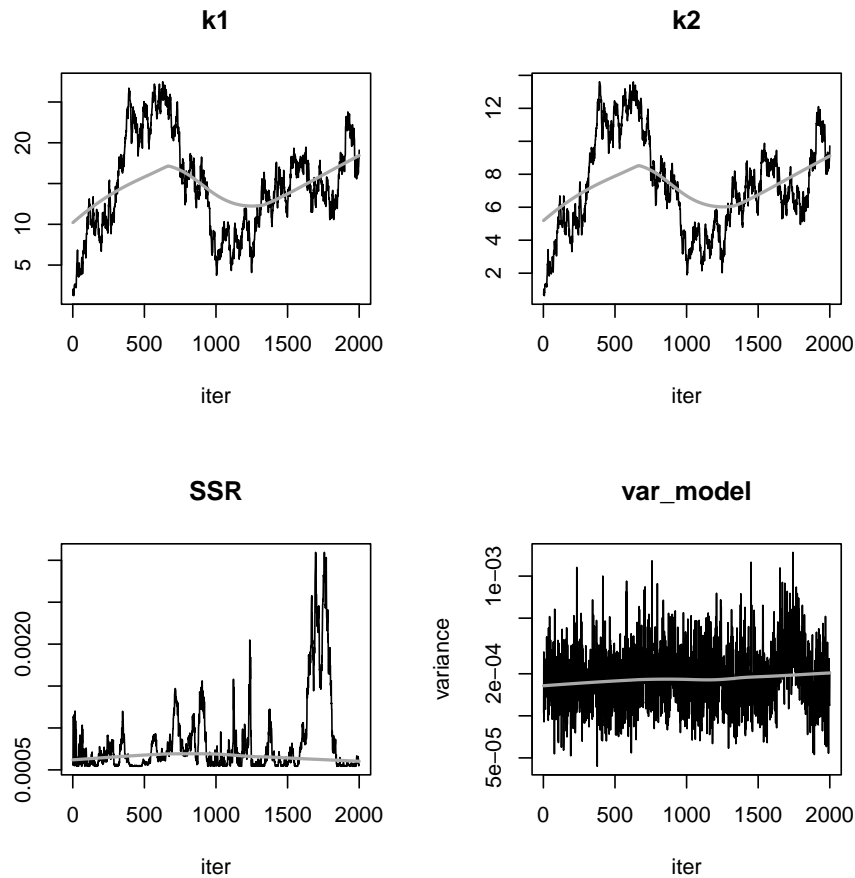


Figure 6: Metropolis-Hastings MCMC of the chemical model - see text for R-code

The initial MCMC method, using the Metropolis-Hastings, has very high acceptance rate, indicating that it has not at all converged; this is confirmed by plotting the chain (figure 6)

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

Better convergence is achieved by the adaptive Metropolis, updating the proposal every 100 runs (figure 7)

```
> MCMC2<- modMCMC(f = residual, p = Fit$par, jump = Cov, updatecov = 100,
+               lower = c(0, 0), var0 = mse, wvar0 = 1, prior = Prior, niter = 2000)
```

number of accepted runs: 720 out of 2000 (36%)

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

The correlation between the two parameters is clear (figure 8):

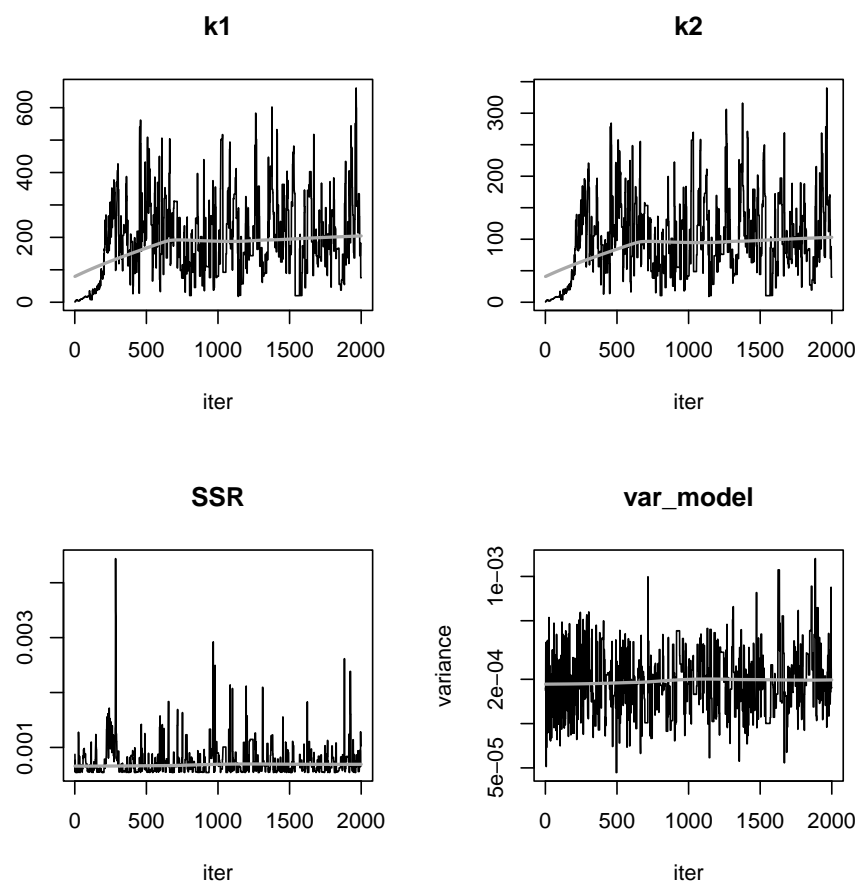


Figure 7: Adaptive Metropolis MCMC of the chemical model - see text for R-code

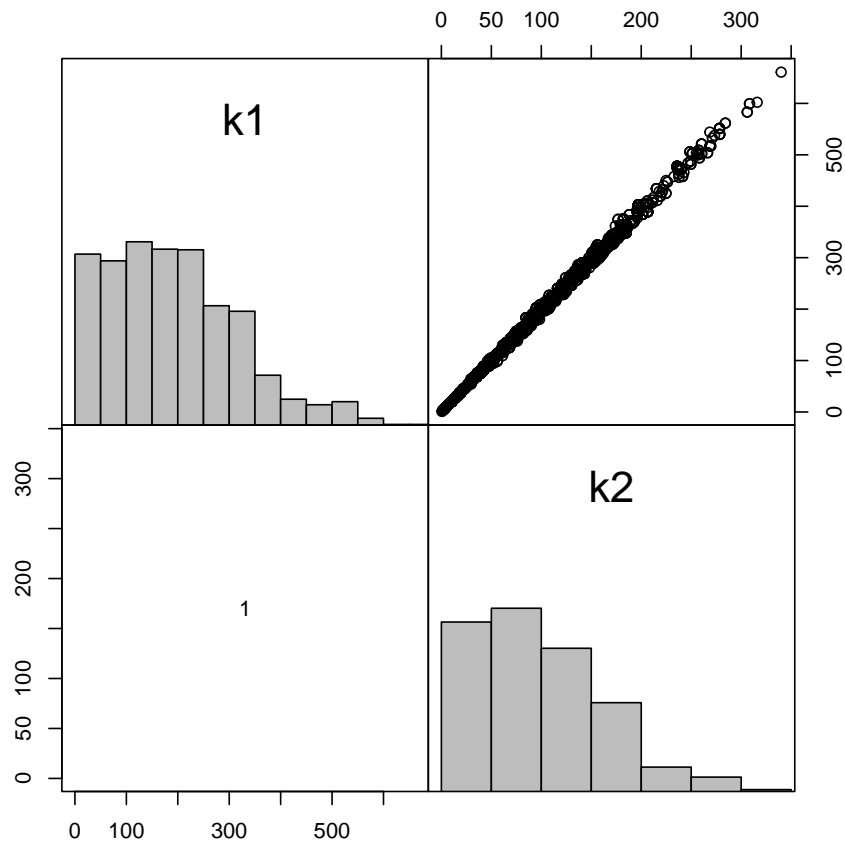


Figure 8: Pairs plot of the Adaptive Metropolis MCMC of the chemical model - see text for R-code

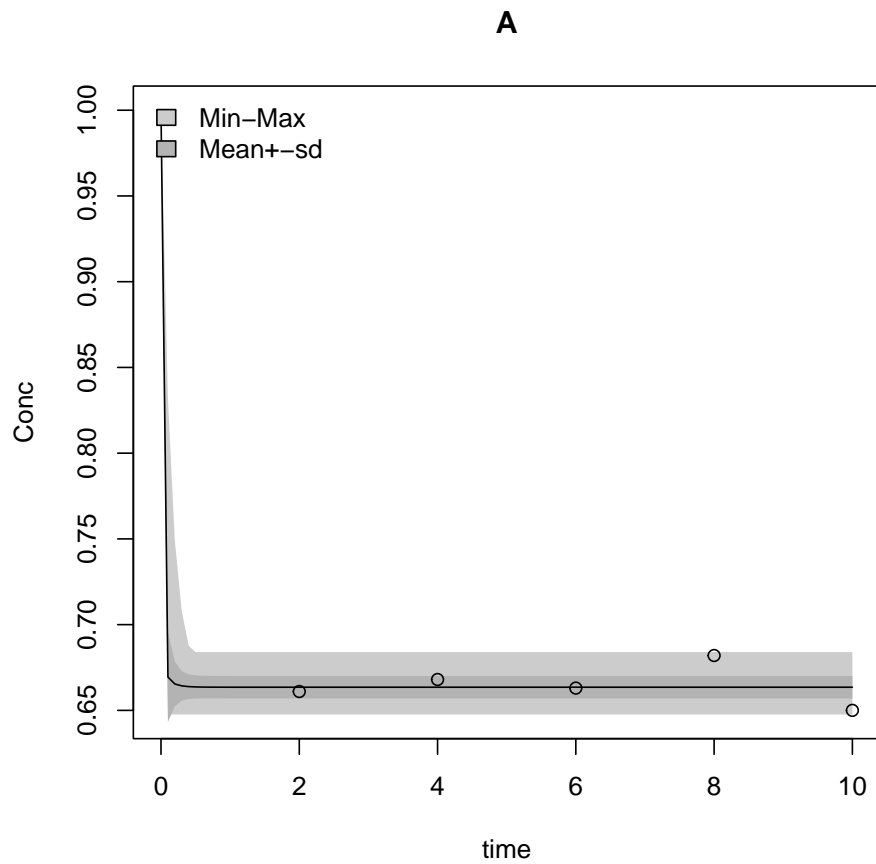


Figure 9: Output ranges induced by parameter uncertainty of the chemical model - see text for R-code

```
> pairs(MCMC2)
```

Finally, we estimate and plot the effects of the estimated parameters on the model output (figure 9)

```
> sR <- sensRange(f=Reaction, times=seq(0,10,0.1), parInput=MCMC2$params)
```

```
> plot(summary(sR), xlab = "time", ylab = "Conc")
```

```
> points(Data)
```

## 7. Fitting a nonlinear model

The following model:

$$y = \theta_1 \cdot \frac{x}{x + \theta_2} + \epsilon$$

$$\epsilon \sim N(0, I\sigma^2)$$

is fitted to two data sets <sup>1</sup>.

### 7.1. Implementation in R

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
> Obs2<- data.frame(x=c( 20, 55, 83, 110, 138, 240, 325), # mg COD/l
+                   y=c(0.05,0.07,0.09,0.10,0.11,0.122,0.125)) # 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])))
```

In function `Residuals`, the model residuals and sum of squares are estimated. In this function, `modCost` is called twice; first with data set "Obs", after which the cost function is updated with data set "Obs2".

```
> Residuals <- function(p) {
+   cost <- modCost(model = Model(p, Obs$x), obs = Obs, x = "x")
+   modCost(model = Model(p, Obs2$x), obs = Obs2, cost = cost, x = "x")
+ }
```

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
0.06 0.00 0.07
```

We plot the observations and best-fit line (figure 10)

```
> plot(Obs, xlab = "mg COD/l", ylab = "1/hour", pch = 16, cex = 1.5)
> points(Obs2, pch = 18, cex = 1.5, col = "red")
> lines(Model(p = P$par, x = 0:375))
```

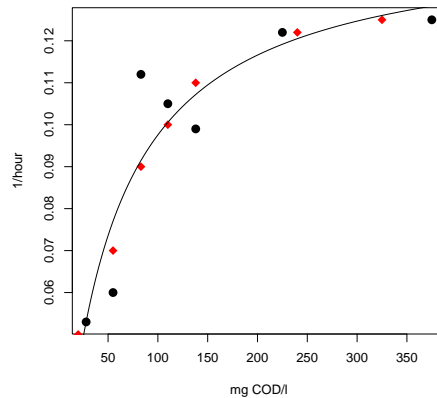


Figure 10: The two sets of Monod observations with best-fit line - see text for R-code

Starting from the best fit, we run several MCMC analyses.

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).

```
> Covar <- summary(P)$cov.scaled * 2.4^2/2
```

## 7.2. Equal model variance

In the first run, we assume that both data sets have equal model variance  $\sigma^2$ .

For the initial model variance (`var0`) we use the residual mean squares `P$ms`, returned by the `modFit` function. We give low weight to the prior (`wvar0=0.1`)

The adoptive Metropolis MCMC is run for 1000 steps; the best-fit parameter set (`P$par`) is used to initiate the chain (`p`). A lower bound (`0`) is imposed on the parameters (`lower`).

```
> s2prior <- P$ms
> print(system.time(
+ MCMC <- modMCMC(f = Residuals, p = P$par, jump = Covar, niter = 1000,
+                 var0 = s2prior, wvar0 = 0.1, lower = c(0, 0))
+ ))
```

```
number of accepted runs: 224 out of 1000 (22.4%)
```

```
  user  system elapsed
3.58   0.04   3.65
```

The plotted results demonstrate (near-) convergence of the chain, and the sampled error variance (`Model`)(figure 11).

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

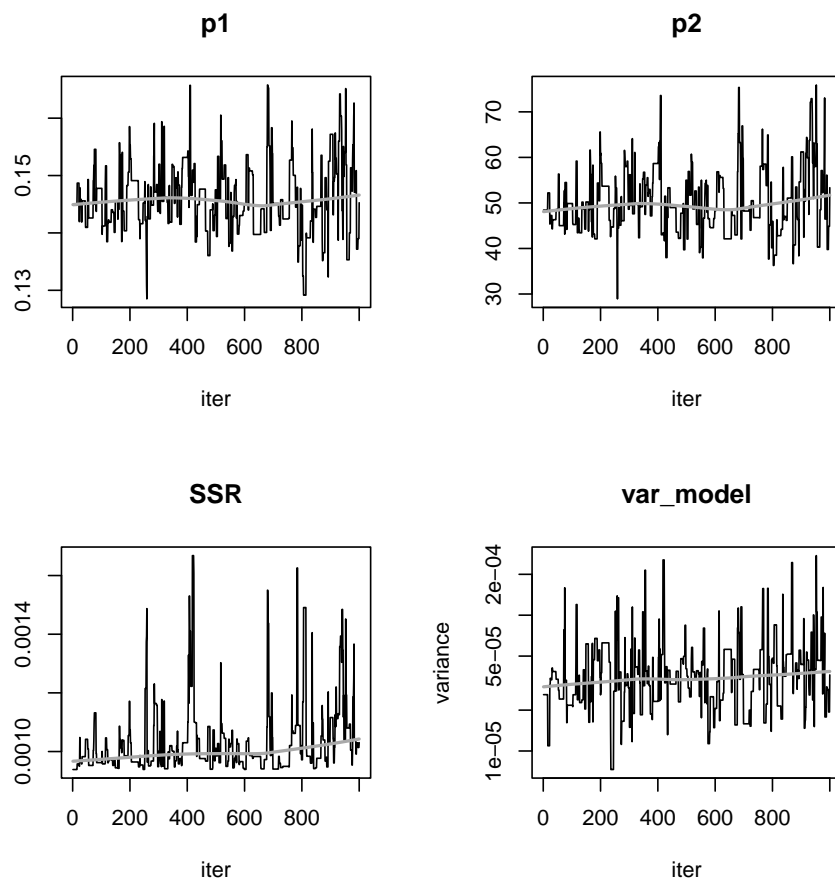


Figure 11: The mcmc, same error variance - see text for R-code

### 7.3. Dataset-specific model variance

In the second run, a different error variance for the two data sets is used.

This is simply done by using, for the initial model variance the variables mean squares, before they are weighted (`P$var_ms_unweighted`).

```
> varprior <- P$var_ms_unweighted
> print(system.time(
+ MCMC2 <- modMCMC(f = Residuals, p = P$par, jump = Covar, niter = 1000,
+                 var0 = varprior, wvar0 = 0.1, lower = c(0, 0))
+ ))
```

```
number of accepted runs: 258 out of 1000 (25.8%)
  user  system elapsed
 3.69   0.03   3.78
```

We plot only the residual sum of squares and the error variances; `which=NULL` does that (figure 12).

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

The summaries for both Markov chains show only small differences.

```
> summary(MCMC)
```

	p1	p2	var_model
mean	0.14585561	49.984356	4.213936e-05
sd	0.00596486	6.927227	3.352869e-05
min	0.12850065	28.926491	7.303041e-06
max	0.16578426	75.905880	2.746997e-04
q025	0.14195455	45.016571	2.368401e-05
q050	0.14512915	48.678591	3.436602e-05
q075	0.14936779	54.308151	4.742310e-05

```
> summary(MCMC2)
```

	p1	p2	sig.var_y	sig.var_y.1
mean	0.145822650	49.668926	3.206104e-05	1.755564e-04
sd	0.005440936	7.005069	2.431168e-05	1.184691e-04
min	0.132350822	36.534122	8.849562e-06	2.449403e-05
max	0.173729210	90.986110	2.380429e-04	9.371530e-04
q025	0.142030486	44.587417	1.704036e-05	9.874067e-05
q050	0.145943571	49.849032	2.431048e-05	1.327817e-04
q075	0.149363981	53.466464	3.708073e-05	2.080904e-04

---

<sup>1</sup>A similar example is also discussed in vignette ("FMEother"). Here the emphasis is on the MCMC method



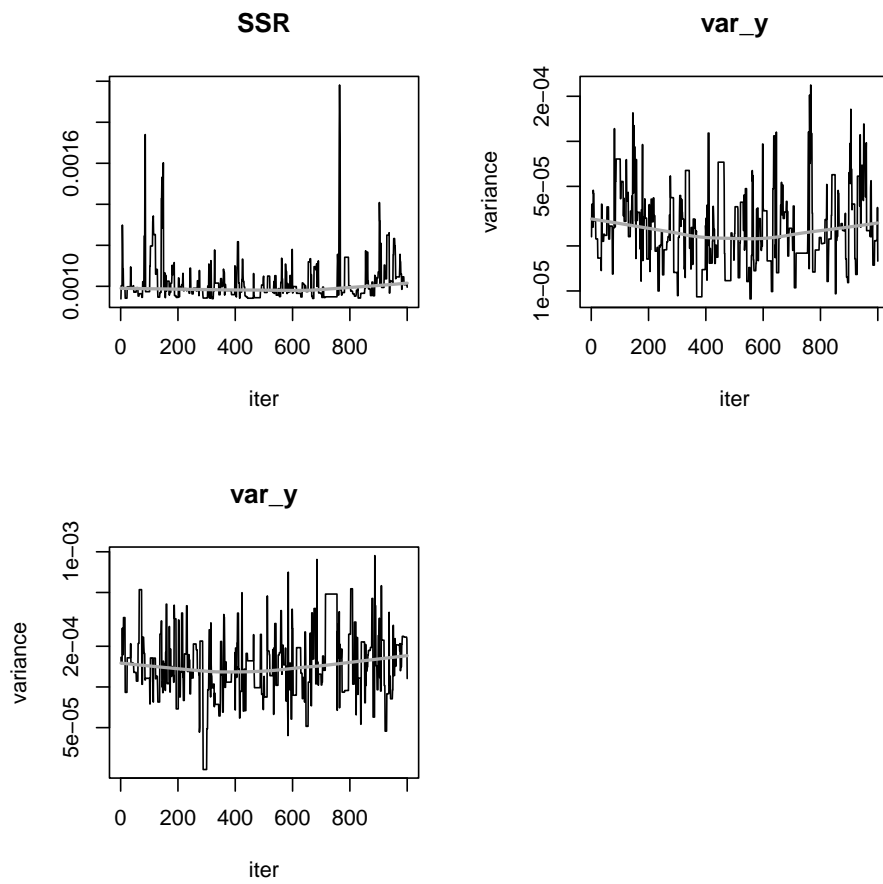


Figure 12: The mcmc chain, separate error variance per data set - see text for R-code

If `var0` has the same number of elements as the number of data points, then distinct error variances for each data point will be estimated.

## 8. Finally

This vignette is made with Sweave (Leisch 2002).

## References

- Gelman A, Varlin JB, Stern HS, Rubin DB (2004). *Bayesian Data Analysis*. 2nd edition. Chapman & Hall/CRC, Boca Raton.
- Haario H, Laine M, Mira A, Saksman E (2006). “DRAM: Efficient Adaptive MCMC.” *Statistics and Computing*, **16**, 339–354.
- Laine M (2008). *Adaptive MCMC Methods with Applications in Environmental and Models*. Finnish Meteorological Institute Contributions 69. ISBN 978-951-697-662-7.
- Leisch F (2002). “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.
- Soetaert K, Petzoldt T (2010). “Inverse Modelling, Sensitivity and Monte Carlo Analysis in R Using Package **FME**.” *Journal of Statistical Software*, **33**(3), 1–28. URL <http://www.jstatsoft.org/v33/i03/>.

### Affiliation:

Karline Soetaert  
Royal Netherlands Institute of Sea Research (NIOZ)  
4401 NT Yerseke, Netherlands  
E-mail: [karline.soetaert@nioz.nl](mailto:karline.soetaert@nioz.nl)  
URL: <http://www.nioz.nl>

Marko Laine  
Finnish Meteorological Institute  
P.O. Box 503  
FI-00101 Helsinki  
Finland  
E-mail: [marko.laine@fmi.fi](mailto:marko.laine@fmi.fi)