R-package FME : MCMC tests

Karline Soetaert
NIOO-CEME
The Netherlands

Marko Laine
Finnish Meteorological Institute
Finland

Abstract

This vignette tests the Markov chain Monte Carlo (MCMC) implementation of Rpackage **FME** (**?**).

It includes the delayed rejection and adaptive Metropolis algorithm (**?**)

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

1. Introduction

Function `modMCMC` from package **FME** (**?**) implements a Markov chain Monte Carlo (MCMC) algorithm using a delayed rejection and adaptive Metropolis procedure (**?**).

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 (**?**)
- A simple chemical model, fitted to a data series (**?**)
- 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 θ and independent variables x. Assuming additive, independent Gaussian errors with an unknown variance σ^2:

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

where

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

For simplicity we assume that the prior distribution for θ 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 σ^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 (`f`, see below) should be specified.

Similarly, to obtain a general non-Gaussian prior for the parameters θ (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 θ 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 numer 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: 471 out of 2000 (23.55%)
```

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: 1377 out of 2000 (68.85%)

> summary(MCMC)

             p1
mean  9.9296799
sd    0.9933976
min   6.0960400
max  13.3548492
q025  9.2636771
q050  9.9554884
q075 10.6099787
```

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: 1407 out of 2000 (70.35%)

> summary(MCMC2)

             p1
mean 10.3827213
sd    0.8018337
min   9.0128753
max  13.0797108
q025  9.7353610
q050 10.2854426
q075 10.9020663
```

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 devation 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: 1366 out of 2000 (68.3%)

> summary(MCMC3)

             p1
mean  9.0386549
sd    0.7287828
min   6.0700128
max  10.9711474
q025  8.5389519
q050  9.0312314
q075  9.5519412
```

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: 1920 out of 2000 (96%)
            p1
mean  9.017530
sd    0.668744
min   6.982116
max  10.975786
q025  8.558114
q050  9.029679
q075  9.510606
```

```
> MCMC4$count
```

```
     dr_steps     Alfasteps  num_accepted num_covupdate
          530          1590          1920           180
```

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

```
> 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)
```

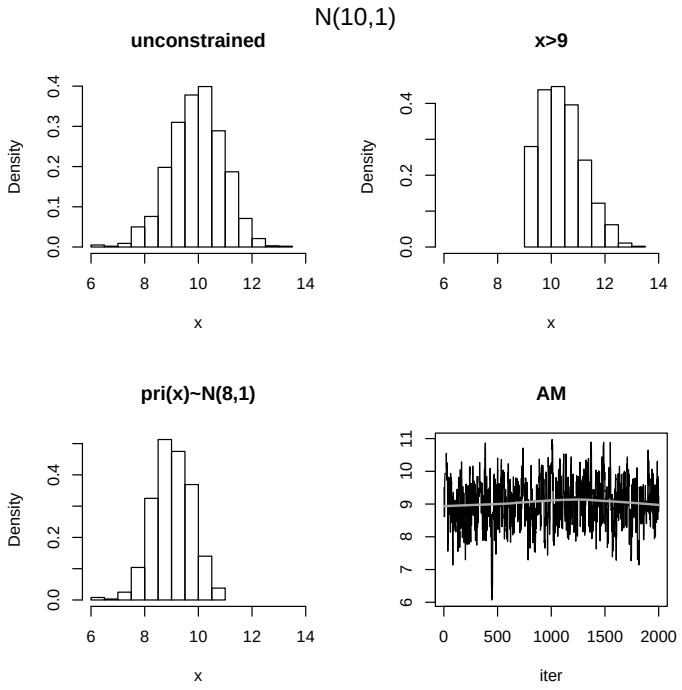

Figure 1: Simulated draws of a normal distribution (N(10,1)) with different prior parameter distributions - see text for R-code

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: 3126 out of 10000 (31.26%)
```

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

```
> 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: 2861 out of 5000 (57.22%)
```

Convergence of the chain is checked (figure **??**).

```
> plot(MCMC1)
```

The histograms show the posterior densities (figure **??**).

```
> hist(MCMC1)


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

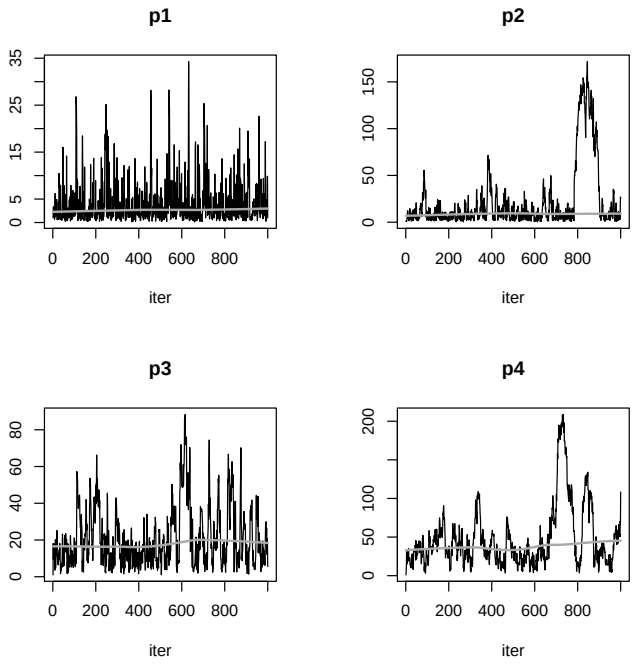

Figure 2: The trace of the log normal distribution -Metropolis algorithm - see text for R-code

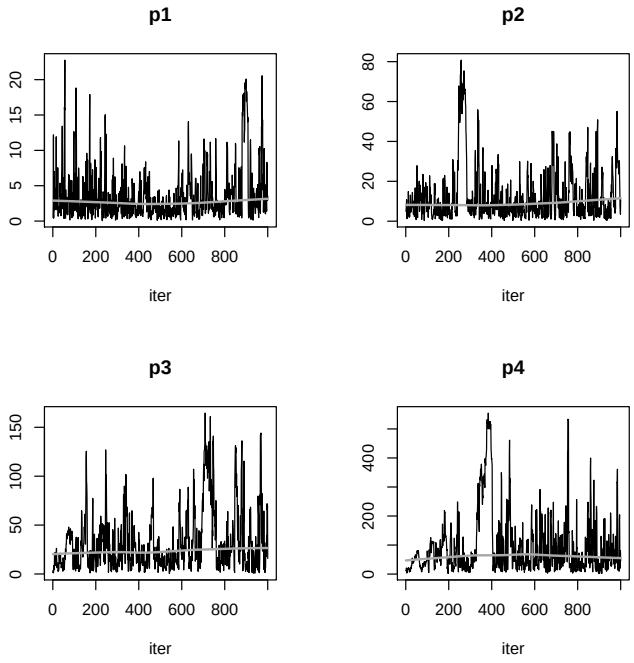

Figure 3: The trace of the log normal distribution - adaptive Metropolis - see text for R-code

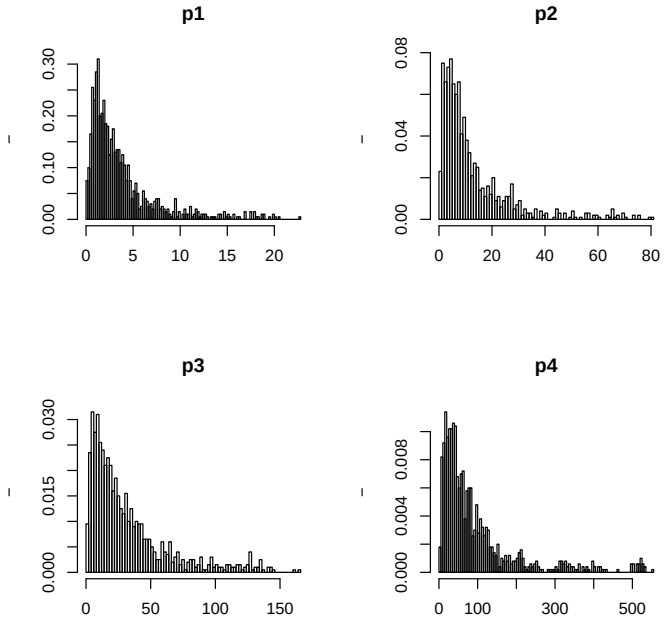

Figure 4: The histograms of the log normal distributed samples - see text for R-code

```
             p1         p2         p3        p4
mean  0.8860375  2.070532  3.0063663 4.0595008
sd    0.9615712  1.012748  1.0225095 1.0347525
min  -2.6365805 -1.149994 -0.7841334 0.6390234
max   3.1251828  4.391652  5.1030729 6.3182357
q025  0.2704375  1.425467  2.3247867 3.3969559
q050  0.9322627  2.071262  3.0765368 4.0754985
q075  1.4972211  2.771071  3.7219690 4.7389060
```

5. The banana

5.1. The model

This example is from **?**.

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:

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

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 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 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: 207 out of 2000 (10.35%)

> MCMC$count

     dr_steps     Alfasteps  num_accepted num_covupdate
            0             0           207             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: 302 out of 2000 (15.1%)

> MCMC2$count

     dr_steps     Alfasteps  num_accepted num_covupdate
            0             0           302            19
```

5.5. Delayed Rejection algorithm

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=2000)

number of accepted runs: 1095 out of 2000 (54.75%)

> MCMC3$count

     dr_steps     Alfasteps  num_accepted num_covupdate
         1830          5490          1095             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: 1153 out of 2000 (57.65%)
   user  system elapsed
   1.81    0.00    1.82

> MCMC4$count

     dr_steps     Alfasteps  num_accepted num_covupdate
         1760          5280          1153            19
```

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

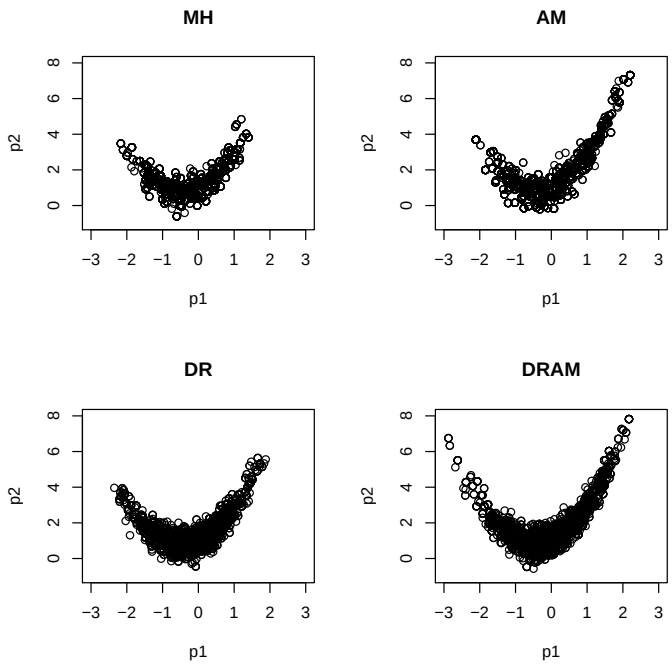

Figure 5: The bananas - see text for R-code

```
[1] -0.06951994 -0.04240827

> sd(trans)          # was:1

[1] 0.9564239 0.9614441

> cov(trans)         # 0.9 off-diagonal

          [,1]      [,2]
[1,] 0.9147467 0.8201834
[2,] 0.8201834 0.9243748
```

6. A simple chemical model

This is an example from (**?**). 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 (**?**).

6.1. Implementation in R

First a function is defined that takes as input the parameters (`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 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
```

Here the observations have additive independent Gaussian errors with unknown variance σ^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$modfVariance`) is used as initial model variance (argument `var0`), the scaled covariance matrix of the fit (`sF$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=5000)
+ ))

number of accepted runs: 4814 out of 5000 (96.28%)
   user  system elapsed
   3.05    0.00    3.06
```

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 **??**)

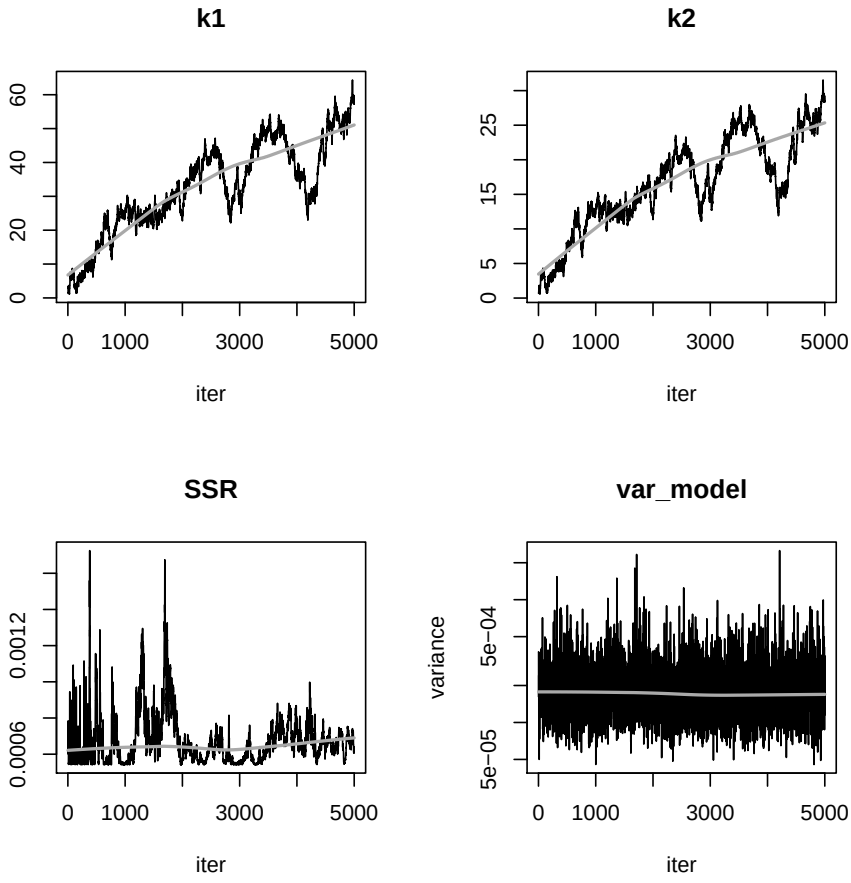

Figure 6: Metropolis-Hastings MCMC of the chemical model - see text for R-code

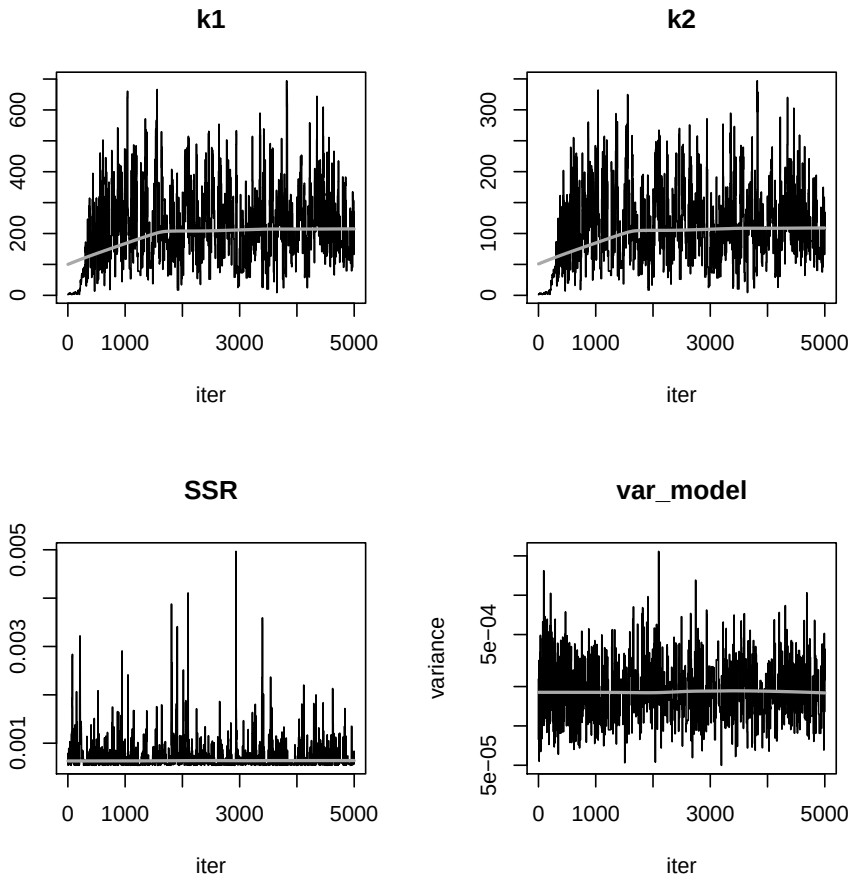

Figure 7: Adaptive Metropolis MCMC of the chemical model - see text for R-code

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

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

```
> 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: 1553 out of 5000 (31.06%)

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

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

```
> pairs(MCMC2)
```

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

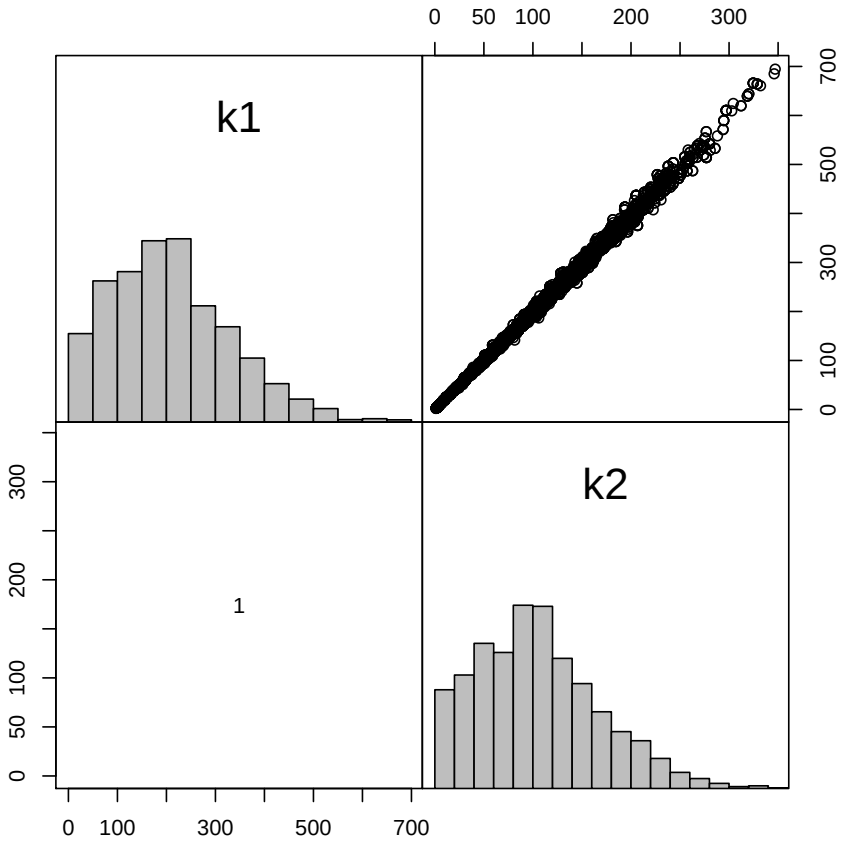

Figure 8: Pairs plot of the Adaptive Metropolis MCMC of the chemical model - see text for R-code

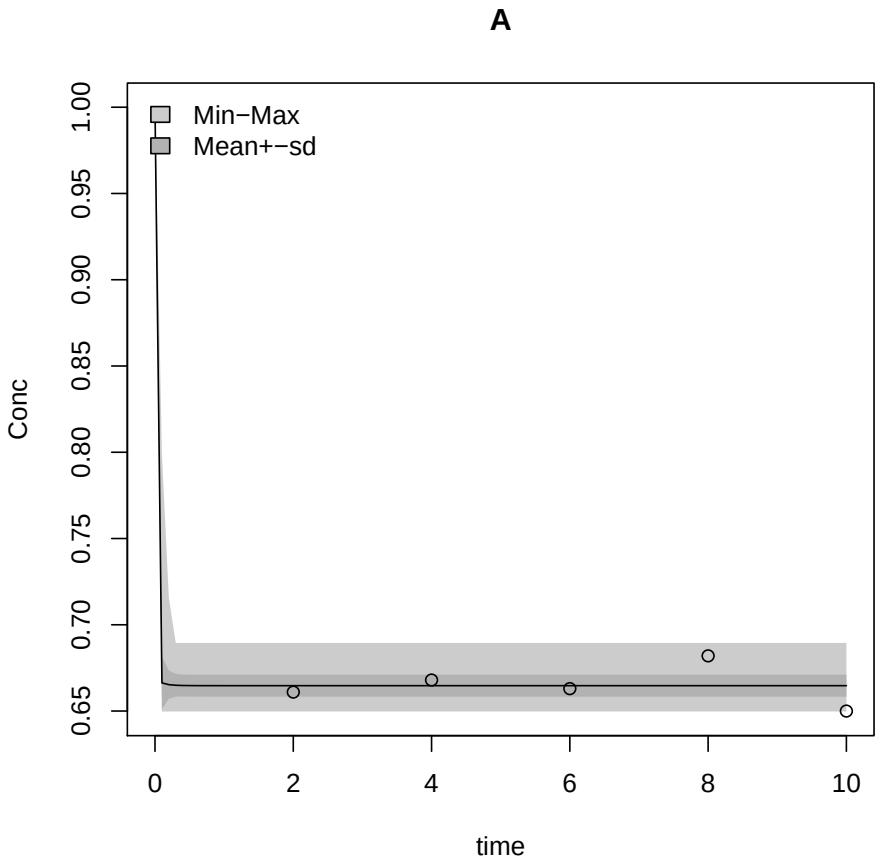

Figure 9: Output ranges induced by parameter uncertainty of the chemical model - see text for R-code

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

> 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 [1].

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
   0.27    0.00    0.27
```

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

```
> 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))
```

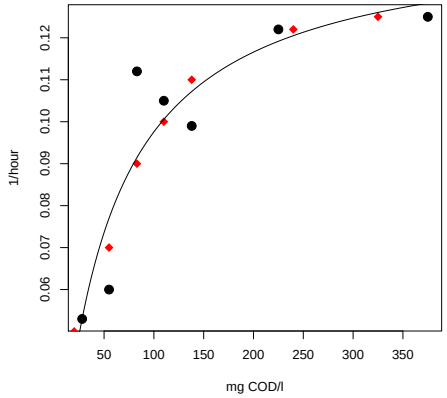

Figure 10: The two sets of Monod observations with best-fit line - see text for R-code

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 (**?**).

```
> 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 σ^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: 215 out of 1000 (21.5%)
   user  system elapsed
  13.41    0.00   13.42
```

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

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

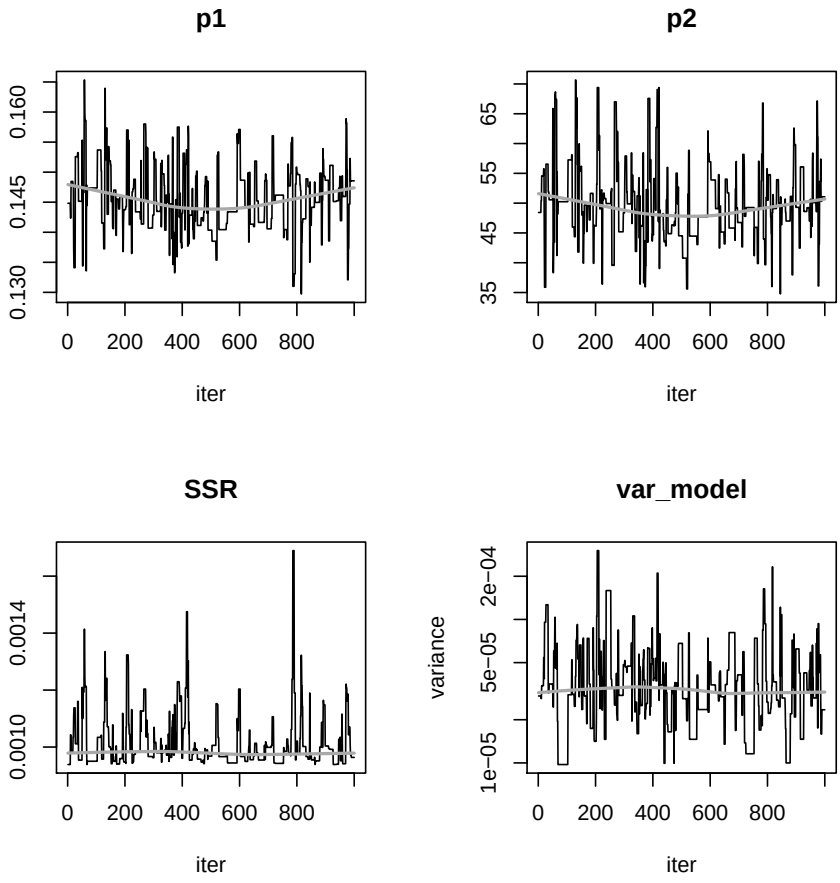

Figure 11: The mcmc, same error variance - see text for R-code

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: 251 out of 1000 (25.1%)
   user  system elapsed
  13.53    0.02   13.69
```

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

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

The summaries for both Markov chains show only small differences.

```
> summary(MCMC)

              p1        p2    var_model
mean 0.145548616 49.816887 4.258169e-05
sd   0.005432869  6.301157 3.796334e-05
min  0.129741633 34.767995 9.741822e-06
max  0.165332532 70.664839 3.018050e-04
q025 0.141627282 45.938158 2.335194e-05
q050 0.145107374 49.255773 3.075867e-05
q075 0.148653777 52.271446 4.835129e-05

> summary(MCMC2)

              p1        p2    sig.var_y  sig.var_y.1
mean 0.144732155 48.542877 3.932502e-05 1.764884e-04
sd   0.005891667  6.597159 4.376019e-05 1.263556e-04
min  0.127431746 31.924846 4.837780e-06 4.075925e-05
max  0.163722752 68.471266 5.598325e-04 1.045515e-03
q025 0.140615623 44.171034 1.860142e-05 9.314645e-05
q050 0.144337007 48.285839 2.612625e-05 1.400486e-04
q075 0.148646730 52.431260 4.706605e-05 2.264053e-04
```

[1] A similar example is also discussed in vignette ("FMEother"). Here the emphasis is on the MCMC method

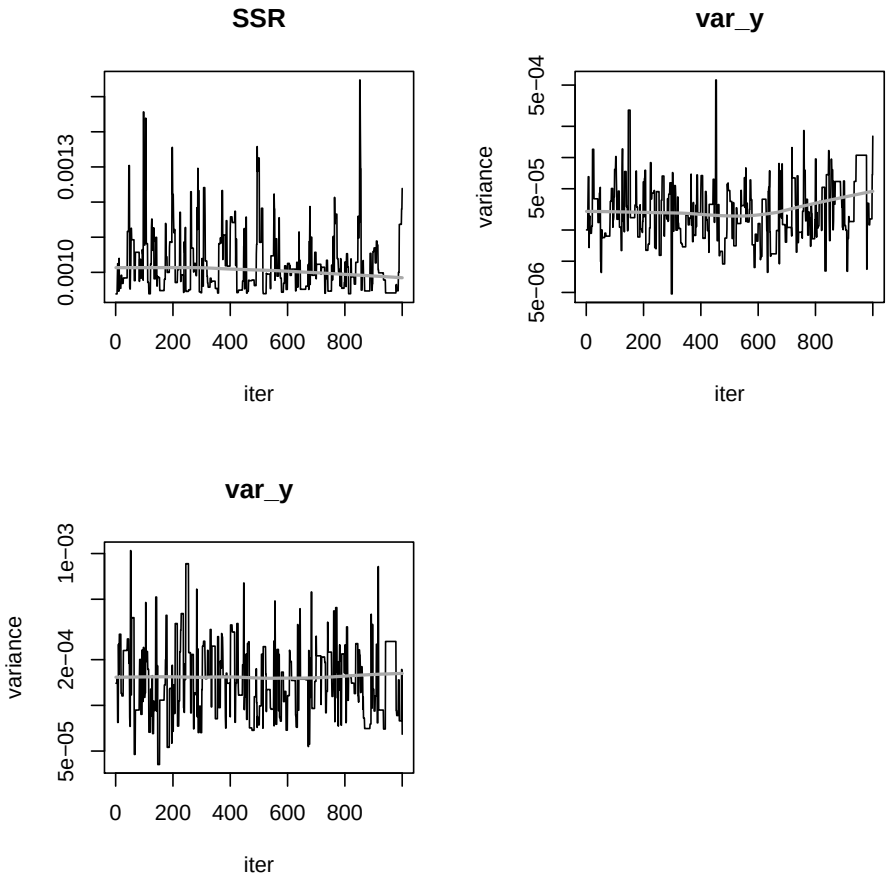

Figure 12: The mcmc chain, separate error variance per data set - see text for R-code

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 (**?**).

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/users/ksoetaert

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