

R Package **FME** : Inverse Modelling, Sensitivity, Monte Carlo – Applied to a Dynamic Simulation Model

Karline Soetaert
NIOO-CEME
The Netherlands

Abstract

Rpackage **FME** (Soetaert and Petzoldt 2010) contains functions for model calibration, sensitivity, identifiability, and Monte Carlo analysis of nonlinear models.

This vignette (`vignette("FMEdyna")`) applies the functions to a dynamic simulation model, solved with integration routines from package **deSolve**. A similar vignette, (`vignette("FMEsteady")`), applies **FME** to a partial differential equation, solved with a steady-state solver from package **rootSolve**. A third vignette (`vignette("FMEother")`), applies the functions to a simple nonlinear model. `vignette("FMEcmc")` tests the Markov chain Monte Carlo (MCMC) implementation.

Keywords: dynamic simulation models, differential equations, fitting, sensitivity, Monte Carlo, identifiability, R.

1. Introduction

R-package **FME** contains part of the functions present in the software environment **FEMME** (Soetaert, deClippele, and Herman 2002), a *Flexible Environment for Mathematically Modeling the Environment*. **FEMME** was written in FORTRAN. **FME** is – obviously – written in R.

Although **FME** can work with many types of functions, it is mainly meant to be used with models that are written as (a system of) differential equations (ordinary or partial), which are solved either with routines from package **deSolve** (Soetaert, Petzoldt, and Setzer 2010), which integrate the model in time, or from package **rootSolve** (Soetaert 2009) which estimate steady-state conditions. With **FME** it is possible to:

- perform local and global sensitivity analysis (Brun, Reichert, and Kunsch 2001; Soetaert and Herman 2009),
- perform parameter identifiability analysis (Brun *et al.* 2001),
- fit a model to data,
- run a Markov chain Monte Carlo (MCMC, Haario, Laine, Mira, and Saksman 2006).

Most of these functions have suitable methods for printing, visualising output etc. In addition, there are functions to generate parameter combinations corresponding to a certain distribution. In this document a – very quick – survey of the functionality is given, based on a simple model from (Soetaert and Herman 2009).

2. The example model

The example model describes growth of bacteria (BACT) on a substrate (SUB) in a closed vessel. The model equations are:

$$\begin{aligned}\frac{dBact}{dt} &= gmax \cdot eff \cdot \frac{Sub}{Sub + ks} \cdot Bact - d \cdot Bact - rB \cdot Bact \\ \frac{dSub}{dt} &= -gmax \cdot \frac{Sub}{Sub + ks} \cdot Bact + d \cdot Bact\end{aligned}$$

where the first, second and third term of the rate of change of **Bact** is growth of bacteria, death and respiration respectively. In R, this model is implemented and solved as follows (see help pages of **deSolve**). First the parameters are defined, as a list (a vector would also do)

```
> pars <- list(gmax = 0.5, eff = 0.5,
+             ks = 0.5, rB = 0.01, dB = 0.01)
```

The model function **solveBact** takes as input the parameters and the time sequence at which output is wanted. Within this function, **derivs** is defined, which is the *derivative* function, called at each time step by the solver. It takes as input the current time (**t**), the current values of the state variables (**state**) and the parameters (**pars**). It returns the rate of change of the state variables, packed as a list. Also within function **solveBact**, the state variables are given an initial condition (**state**) and the model is solved by integration, using function **ode** from package **deSolve**. The results of the integration are returned, packed as a data.frame.

```
> solveBact <- function(pars, times=seq(0,50,by=0.5)) {
+   derivs <- function(t, state, pars) { # returns rate of change
+     with(as.list(c(state, pars)), {
+
+       dBact <- gmax*eff*Sub/(Sub+ks)*Bact - dB*Bact - rB*Bact
+       dSub  <- -gmax      *Sub/(Sub+ks)*Bact + dB*Bact
+       return(list(c(dBact, dSub), TOC = Bact + Sub))
+     })
+   }
+   state <- c(Bact = 0.1, Sub = 100)
+   ## ode solves the model by integration...
+   return(ode(y = state, times = times, func = derivs, parms = pars))
+ }
```

The model is then solved by calling **solveBact** with the default parameters:

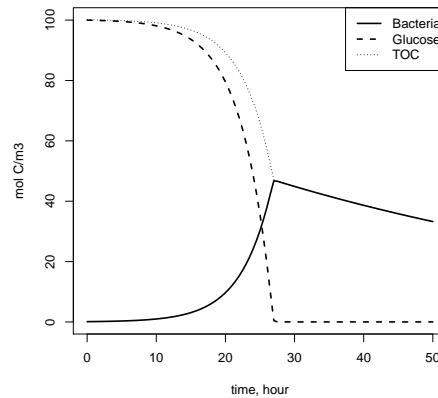


Figure 1: Solution of the simple bacterial growth model - see text for R-code

```
> out <- solveBact(pars)
```

and output plotted as:

```
> matplot(out[,1], out[,-1], type = "l", lty = 1:3, lwd = c(2, 2, 1),
+         col = "black", xlab = "time, hour", ylab = "mol C/m3")
> legend("topright", c("Bacteria", "Glucose", "TOC"),
+         lty = 1:3, lwd = c(2, 2, 1))
```

3. Global sensitivity

In global sensitivity analysis, certain parameters are changed over a large range, and the effect on certain model output variables assessed. In **FME** this is done via function **sensRange**.

First the sensitivity parameters are defined and a distribution is assigned; here we specify the minimum and maximum values of three parameters in a **data.frame**:

```
> parRanges <- data.frame(min = c(0.4, 0.4, 0.0), max = c(0.6, 0.6, 0.02))
> rownames(parRanges) <- c("gmax", "eff", "rB")
> parRanges

      min  max
gmax 0.4 0.60
eff   0.4 0.60
rB    0.0 0.02
```

Then we estimate the sensitivity to one parameter, **rB** (parameter 3), varying its values according to a regular grid (**dist=grid**). The effect of that on sensitivity variables **Bact** and **Sub** are estimated. To do this, the model is run 100 times (**num=100**). The **system.time** is printed (in seconds):

```

> tout      <- 0:50
> print(system.time(
+   sR <- sensRange(func = solveBact, parms = pars, dist = "grid",
+     sensvar = c("Bact", "Sub"), parRange = parRanges[3,], num = 50)
+ ))

      user system elapsed
      2.81    0.00     2.83

> head(summary(sR))

      x      Mean      Sd      Min      Max      q05      q25
Bact0  0.0 0.1000000 0.0000000000 0.1000000 0.1000000 0.1000000 0.1000000
Bact0.5 0.5 0.1121194 0.0003335405 0.1115597 0.1126809 0.1116155 0.1118390
Bact1   1.0 0.1257062 0.0007479668 0.1244532 0.1269674 0.1245777 0.1250770
Bact1.5 1.5 0.1409422 0.0012579439 0.1388384 0.1430668 0.1390468 0.1398836
Bact2   2.0 0.1580263 0.0018805072 0.1548863 0.1612075 0.1551964 0.1564430
Bact2.5 2.5 0.1771819 0.0026354918 0.1727886 0.1816476 0.1732211 0.1749620
      q50      q75      q95
Bact0  0.1000000 0.1000000 0.1000000
Bact0.5 0.1121189 0.1123996 0.1126246
Bact1   0.1257040 0.1263341 0.1268405
Bact1.5 0.1409367 0.1419978 0.1428524
Bact2   0.1580153 0.1596034 0.1608854
Bact2.5 0.1771627 0.1793911 0.1811941

```

The results are represented as a data.frame, containing summary information of the value of the sensitivity variable (*var*) at each time step (*x*). It is relatively simple to plot the ranges, either as $\text{min} \pm \text{sd}$ or using quantiles:

```

> summ.sR <- summary(sR)
> par(mfrow=c(2, 2))
> plot(summ.sR, xlab = "time, hour", ylab = "molC/m3",
+   legpos = "topright", mfrow = NULL)
> plot(summ.sR, xlab = "time, hour", ylab = "molC/m3", mfrow = NULL,
+   quant = TRUE, col = c("lightblue", "darkblue"), legpos = "topright")
> mtext(outer = TRUE, line = -1.5, side = 3, "Sensitivity to rB", cex = 1.25)
> par(mfrow = c(1, 1))

```

Sensitivity ranges can also be estimated for a combination of parameters. Here we use all 3 parameters, and select the latin hypercube sampling algorithm.

```

> Sens2 <- summary(sensRange(func = solveBact, parms = pars,
+   dist = "latin", sensvar = "Bact", parRange = parRanges, num = 100))

> plot(Sens2, main = "Sensitivity gmax,eff,rB", xlab = "time, hour",
+   ylab = "molC/m3")

```

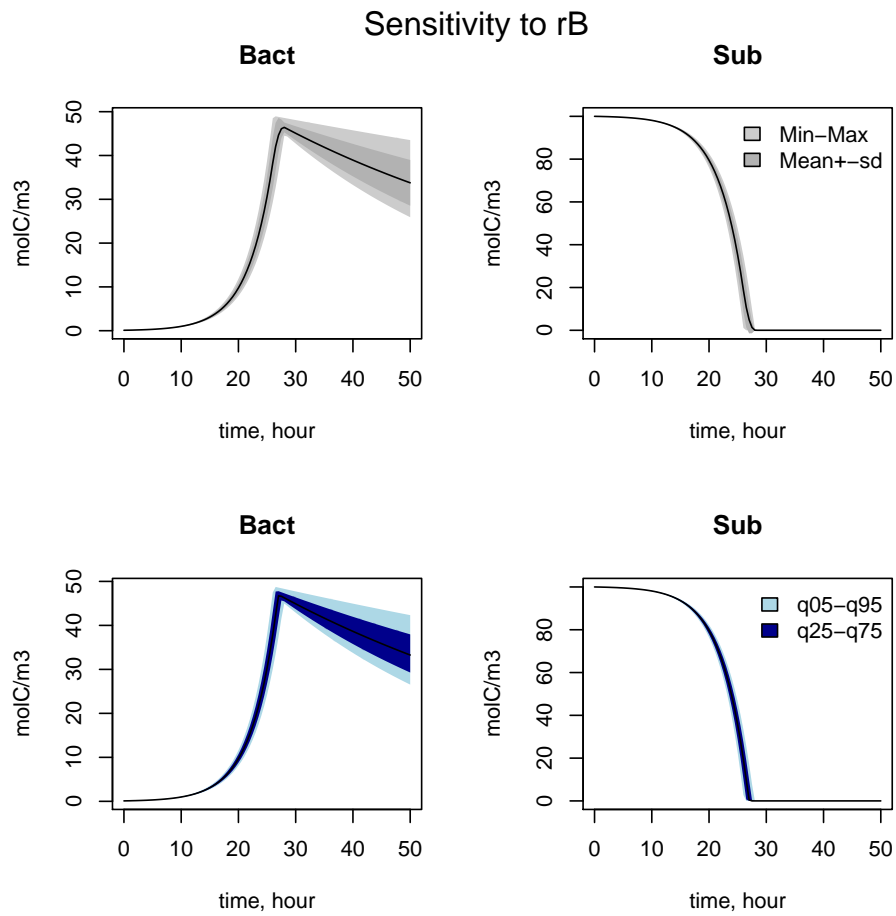


Figure 2: Sensitivity range for one parameter - see text for R-code

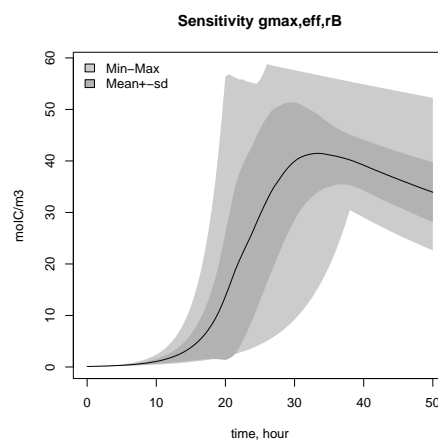


Figure 3: Sensitivity range for a combination of parameters - see text for R-code

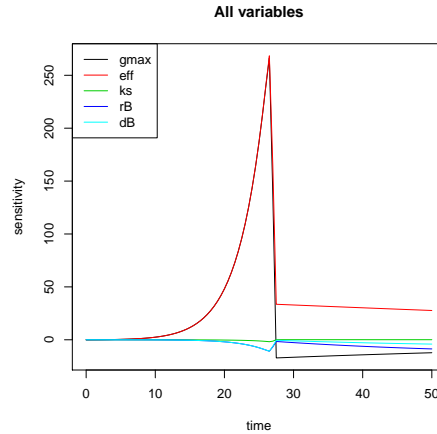


Figure 4: Sensitivity functions - see text for R-code

4. Local sensitivity

In local sensitivity, the effect of a parameter value in a very small region near its nominal value is estimated. The methods implemented in **FME** are based on [Brun *et al.* \(2001\)](#) which should be consulted for details. They are based on so-called “sensitivity functions”.

4.1. Sensitivity functions

Sensitivity functions are generated with `sensFun`, and estimate the effect of a selection of parameters (here all parameters are selected) on a selection of variables (here only `Bact`).

```
> SnsBact<- sensFun(func = solveBact, parms = pars,
+                  sensvar = "Bact", varscale = 1)
> head(SnsBact)
```

| | x | var | gmax | eff | ks | rB | dB |
|---|-----|------|------------|------------|---------------|--------------|---------------|
| 1 | 0.0 | Bact | 0.00000000 | 0.00000000 | 0.0000000000 | 0.000000000 | 0.0000000000 |
| 2 | 0.5 | Bact | 0.01394713 | 0.01394714 | -0.0000703354 | -0.000560559 | -0.0005605588 |
| 3 | 1.0 | Bact | 0.03127214 | 0.03127218 | -0.0001562084 | -0.001257071 | -0.0012570697 |
| 4 | 1.5 | Bact | 0.05259210 | 0.05259221 | -0.0002623735 | -0.002114135 | -0.0021141330 |
| 5 | 2.0 | Bact | 0.07861787 | 0.07861810 | -0.0003921211 | -0.003160362 | -0.0031603571 |
| 6 | 2.5 | Bact | 0.11017714 | 0.11017756 | -0.0005495271 | -0.004429033 | -0.0044290250 |

They can easily be plotted (Fig. 3):

```
> plot(SnsBact)
```

4.2. Univariate sensitivity

Based on the sensitivity functions, several summaries are generated, which allow to rank the parameters based on their influence on the selected variables.

```
> summary(SnsBact)
```

| | value | scale | L1 | L2 | Mean | Min | Max | N |
|------|-------|-------|-------|-------|------|-------|---------|-----|
| gmax | 0.50 | 0.50 | 29.51 | 5.859 | 16.2 | -17.1 | 266.361 | 101 |
| eff | 0.50 | 0.50 | 37.12 | 6.212 | 37.1 | 0.0 | 268.408 | 101 |
| ks | 0.50 | 0.50 | 0.17 | 0.037 | -0.1 | -1.8 | 0.097 | 101 |
| rB | 0.01 | 0.01 | 3.47 | 0.463 | -3.5 | -10.8 | 0.000 | 101 |
| dB | 0.01 | 0.01 | 2.06 | 0.297 | -2.1 | -10.8 | 0.000 | 101 |

Here

- L1 is the L1-norm, $\sum |S_{ij}|/n$
- L2 is the L2-norm, $\sqrt{\sum (S_{ij}^2)/n}$
- Mean: the mean of the sensitivity functions
- Min: the minimal value of the sensitivity functions
- Max: the maximal value of the sensitivity functions

Sensitivity analysis can also be performed on several variables:

```
> summary(sensFun(solveBact, pars, varscale = 1), var = TRUE)
```

| | value | scale | L1 | L2 | Mean | Min | Max | N | var |
|-------|-------|-------|-------|--------|--------|----------|---------|-----|------|
| gmax1 | 0.50 | 0.50 | 29.51 | 58.88 | 16.25 | -1.7e+01 | 2.7e+02 | 101 | Bact |
| gmax2 | 0.50 | 0.50 | 48.40 | 122.95 | -48.40 | -5.6e+02 | 0.0e+00 | 101 | Sub |
| gmax3 | 0.50 | 0.50 | 32.16 | 65.66 | -32.16 | -3.0e+02 | 0.0e+00 | 101 | TOC |
| eff1 | 0.50 | 0.50 | 37.12 | 62.43 | 37.12 | 0.0e+00 | 2.7e+02 | 101 | Bact |
| eff2 | 0.50 | 0.50 | 39.64 | 102.50 | -39.64 | -4.8e+02 | 6.8e-06 | 101 | Sub |
| eff3 | 0.50 | 0.50 | 30.39 | 48.31 | -2.52 | -2.1e+02 | 3.4e+01 | 101 | TOC |
| ks1 | 0.50 | 0.50 | 0.17 | 0.37 | -0.10 | -1.8e+00 | 9.7e-02 | 101 | Bact |
| ks2 | 0.50 | 0.50 | 0.29 | 0.77 | 0.29 | 0.0e+00 | 3.8e+00 | 101 | Sub |
| ks3 | 0.50 | 0.50 | 0.19 | 0.41 | 0.19 | 0.0e+00 | 2.0e+00 | 101 | TOC |
| rB1 | 0.01 | 0.01 | 3.47 | 4.65 | -3.47 | -1.1e+01 | 0.0e+00 | 101 | Bact |
| rB2 | 0.01 | 0.01 | 1.59 | 4.12 | 1.59 | -2.8e-07 | 1.9e+01 | 101 | Sub |
| rB3 | 0.01 | 0.01 | 3.19 | 4.37 | -1.87 | -8.6e+00 | 8.3e+00 | 101 | TOC |
| dB1 | 0.01 | 0.01 | 2.06 | 2.98 | -2.06 | -1.1e+01 | 0.0e+00 | 101 | Bact |
| dB2 | 0.01 | 0.01 | 1.78 | 4.54 | 1.78 | 0.0e+00 | 2.1e+01 | 101 | Sub |
| dB3 | 0.01 | 0.01 | 1.97 | 2.84 | -0.29 | -4.1e+00 | 1.0e+01 | 101 | TOC |

4.3. Bivariate sensitivity

The pairwise relationships in parameter sensitivity is easily assessed by plotting the sensitivity functions using R-function `pairs`, and by calculating the correlation.

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

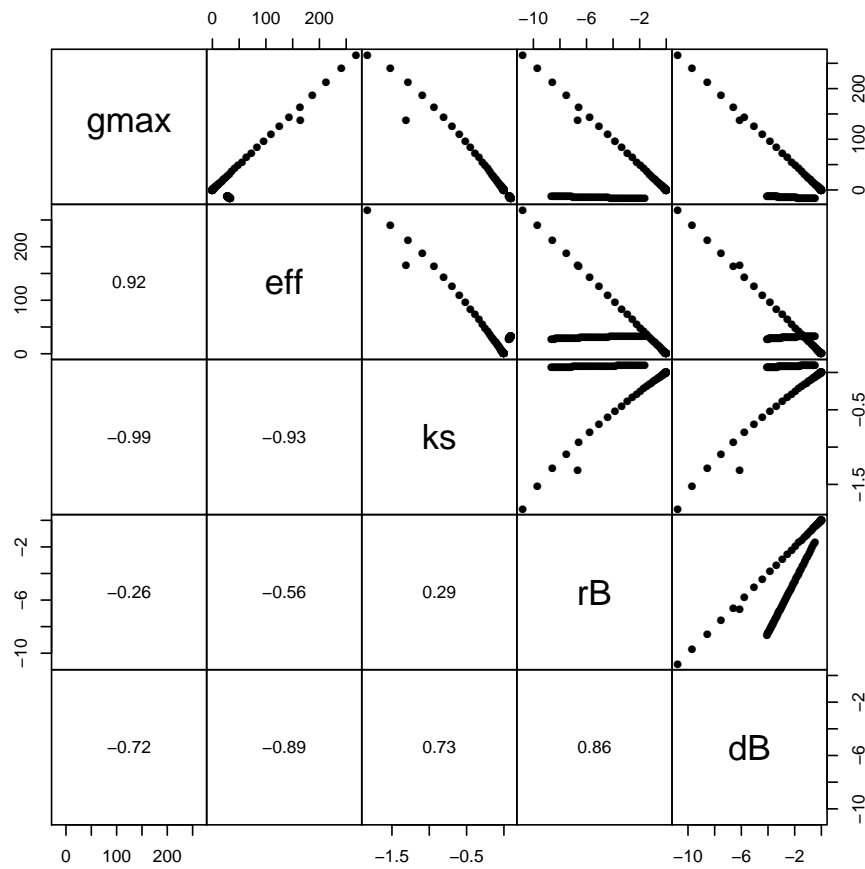


Figure 5: Pairs of sensitivity functions - see text for R-code

| | gmax | eff | ks | rB | dB |
|------|------------|------------|------------|------------|------------|
| gmax | 1.0000000 | 0.9184218 | -0.9879349 | -0.2602262 | -0.7165956 |
| eff | 0.9184218 | 1.0000000 | -0.9265098 | -0.5575106 | -0.8883636 |
| ks | -0.9879349 | -0.9265098 | 1.0000000 | 0.2878336 | 0.7302580 |
| rB | -0.2602262 | -0.5575106 | 0.2878336 | 1.0000000 | 0.8599353 |
| dB | -0.7165956 | -0.8883636 | 0.7302580 | 0.8599353 | 1.0000000 |

```
> pairs(SnsBact)
```

4.4. Monte Carlo runs

Function `modCRL` runs a Monte Carlo simulation, outputting single variables.

This is in contrast to `sensRange` which outputs vectors of variables, e.g. a time-sequence, or a spatially-dependent variable.

It can be used to test what-if scenarios. Here it is used to calculate the final concentration of bacteria and substrate as a function of the maximal growth rate.

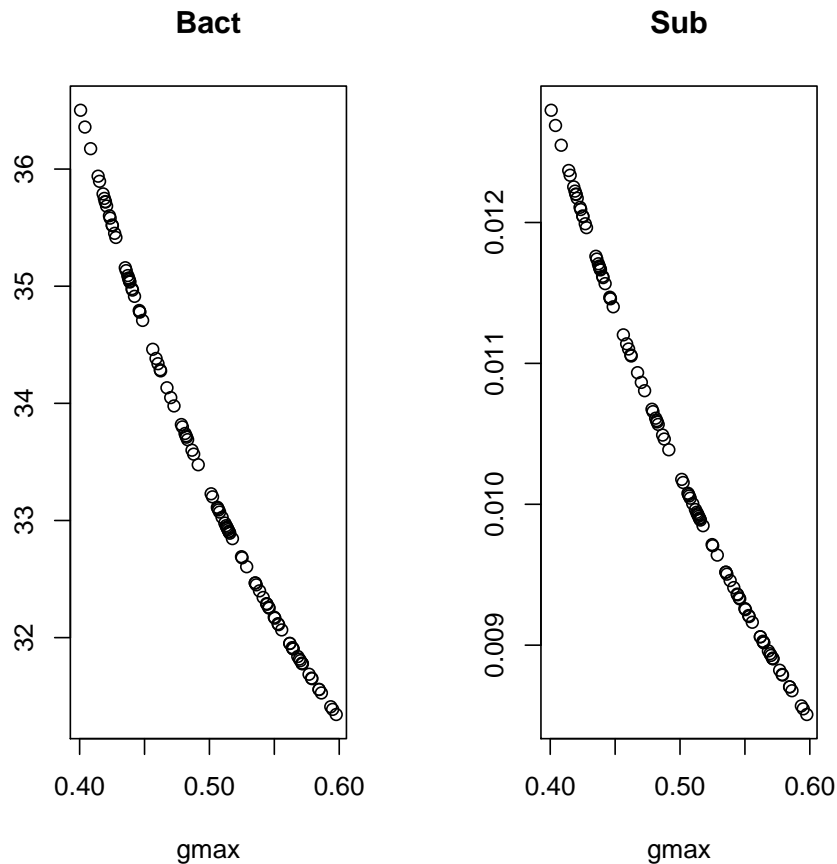


Figure 6: Monte carlo analysis - see text for R-code

```
> SF <- function (pars) {
+   out <- solveBact(pars)
+   return(out[nrow(out), 2:3])
+ }
> CRL <- modCRL(func = SF, parms = pars, parRange = parRanges[1,])
> plot(CRL)
```

Monte Carlo methods can also be used to see how parameter uncertainties propagate, i.e. to derive the distribution of output variables as a function of parameter distribution.

Here the effect of the parameters **gmax** and **eff** on final bacterial concentration is assessed. The parameter values are generated according to a multi-normal distribution; they are positively correlated (with a correlation = 0.63).

```
> CRL2 <- modCRL(func = SF, parms = pars, parMean = c(gmax = 0.5, eff = 0.7),
+               parCovar = matrix(nr = 2, data = c(0.02, 0.02, 0.02, 0.05)),
+               dist = "norm", sensvar = "Bact", num = 150)
```

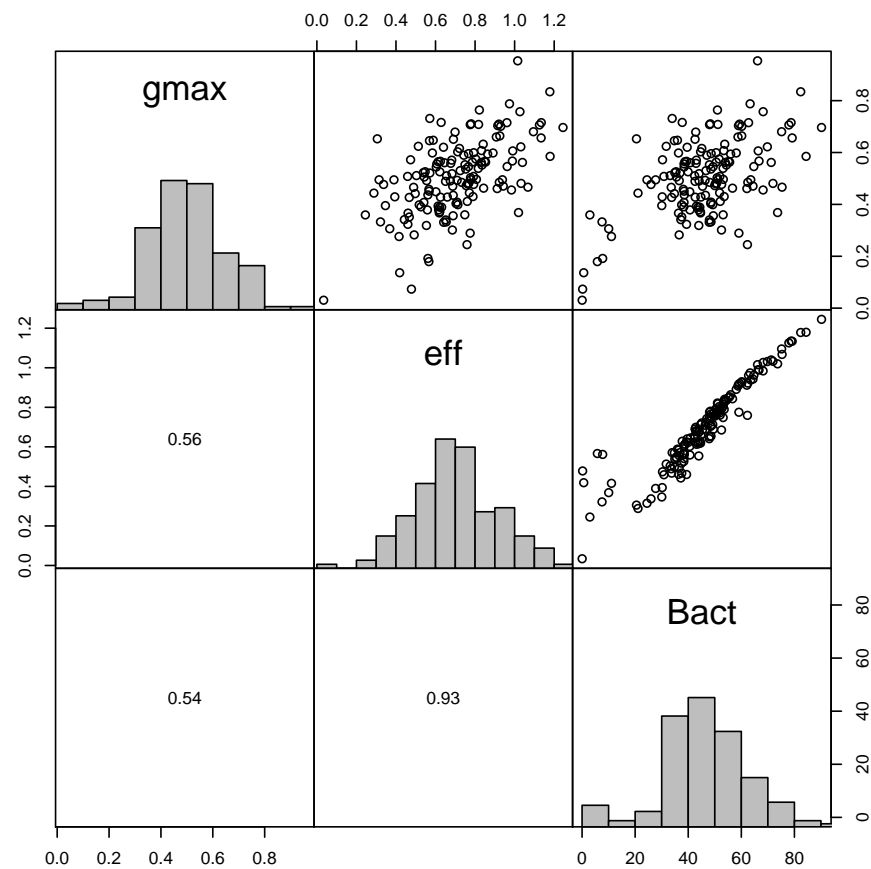


Figure 7: Multivariate Monte Carlo analysis - see text for R-code

```
> pairs(CRL2)
```

5. Multivariate sensitivity analysis

Based on the sensitivity functions of model variables to selection of parameters, function `collin` calculates the *collinearity* or *identifiability* of sets of parameters.

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

| | gmax | eff | ks | rB | dB | N | collinearity |
|---|------|-----|----|----|----|---|--------------|
| 1 | 1 | 1 | 0 | 0 | 0 | 2 | 2.8 |
| 2 | 1 | 0 | 1 | 0 | 0 | 2 | 9.5 |
| 3 | 1 | 0 | 0 | 1 | 0 | 2 | 1.3 |
| 4 | 1 | 0 | 0 | 0 | 1 | 2 | 1.8 |

| | | | | | | | |
|----|---|---|---|---|---|---|-----------|
| 5 | 0 | 1 | 1 | 0 | 0 | 2 | 2.9 |
| 6 | 0 | 1 | 0 | 1 | 0 | 2 | 2.0 |
| 7 | 0 | 1 | 0 | 0 | 1 | 2 | 3.7 |
| 8 | 0 | 0 | 1 | 1 | 0 | 2 | 1.3 |
| 9 | 0 | 0 | 1 | 0 | 1 | 2 | 1.8 |
| 10 | 0 | 0 | 0 | 1 | 1 | 2 | 3.8 |
| 11 | 1 | 1 | 1 | 0 | 0 | 3 | 9.5 |
| 12 | 1 | 1 | 0 | 1 | 0 | 3 | 7.0 |
| 13 | 1 | 1 | 0 | 0 | 1 | 3 | 6.8 |
| 14 | 1 | 0 | 1 | 1 | 0 | 3 | 9.5 |
| 15 | 1 | 0 | 1 | 0 | 1 | 3 | 9.5 |
| 16 | 1 | 0 | 0 | 1 | 1 | 3 | 2261.4 |
| 17 | 0 | 1 | 1 | 1 | 0 | 3 | 6.7 |
| 18 | 0 | 1 | 1 | 0 | 1 | 3 | 6.8 |
| 19 | 0 | 1 | 0 | 1 | 1 | 3 | 12.2 |
| 20 | 0 | 0 | 1 | 1 | 1 | 3 | 22.6 |
| 21 | 1 | 1 | 1 | 1 | 0 | 4 | 9.6 |
| 22 | 1 | 1 | 1 | 0 | 1 | 4 | 9.5 |
| 23 | 1 | 1 | 0 | 1 | 1 | 4 | 3631.3 |
| 24 | 1 | 0 | 1 | 1 | 1 | 4 | 3451.1 |
| 25 | 0 | 1 | 1 | 1 | 1 | 4 | 23.5 |
| 26 | 1 | 1 | 1 | 1 | 1 | 5 | 2421318.6 |

```
> Coll [Coll[, "collinearity"] < 20 & Coll[, "N"] == 4, ]
```

| | gmax | eff | ks | rB | dB | N | collinearity |
|---|------|-----|----|----|----|---|--------------|
| 1 | 1 | 1 | 1 | 1 | 0 | 4 | 9.6 |
| 2 | 1 | 1 | 1 | 0 | 1 | 4 | 9.5 |

```
> collin(SnsBact, parset = 1:5)
```

| | gmax | eff | ks | rB | dB | N | collinearity |
|---|------|-----|----|----|----|---|--------------|
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 2421319 |

The higher the value, the larger the (approximate) linear dependence. This function is mainly useful to derive suitable parameter sets that can be calibrated based on data (see next section).

6. Fitting the model to data

6.1. Data structures

There are two modes of data input:

- *data table (long) format*; this is a two to four column data.frame that contains the **name** of the observed variable (always the FIRST column), the (optional) **value of the independent variable** (default = "time"), the **value of the observation** and the (optional) **value of the error**.

- *crosstable format*; this is a matrix, where each column denotes one dependent (or independent) variable; the column name is the name of the observed variable.

As an example of both formats consider the data, called `Dat` consisting of two observed variables, called "Obs1" and "Obs2", both containing two observations, at time 1 and 2:

| name | time | val | err |
|------|------|-----|-----|
| Obs1 | 1 | 50 | 5 |
| Obs1 | 2 | 150 | 15 |
| Obs2 | 1 | 1 | 0.1 |
| Obs2 | 2 | 2 | 0.2 |

for the long format and

| time | Obs1 | Obs2 |
|------|------|------|
| 1 | 50 | 1 |
| 2 | 150 | 2 |

for the crosstable format. Note, that in the latter case it is not possible to provide separate errors per data point.

6.2. The model cost function

FME function `modCost` estimates the “model cost”, which is the sum of (weighted) squared residuals of the model versus the data. This function is central to parameter identifiability analysis, model fitting or running a Markov chain Monte Carlo.

Assume the following model output (in a matrix or `data.frame` called `Mod`):

| time | Obs1 | Obs2 |
|------|------|------|
| 0 | 4 | 1 |
| 1 | 4 | 2 |
| 2 | 4 | 3 |
| 3 | 4 | 4 |

Then the `modCost` will give:

```
> Dat<- data.frame(name = c("Obs1", "Obs1", "Obs2", "Obs2"),
+                   time = c(1, 2, 1, 2), val = c(50, 150, 1, 2),
+                   err = c(5, 15, 0.1, 0.2))
> Mod <- data.frame(time = 0:3, Obs1 = rep(4, 4), Obs2 = 1:4)
> modCost(mod = Mod, obs = Dat, y = "val")
```

```
$model
[1] 23434
```

```
$minlogp
```

```
[1] Inf
```

```
$var
```

| | name | scale | N | SSR.unweighted | SSR.unscaled | SSR |
|---|------|-------|---|----------------|--------------|-------|
| 1 | Obs1 | 1 | 2 | 23432 | 23432 | 23432 |
| 2 | Obs2 | 1 | 2 | 2 | 2 | 2 |

```
$residuals
```

| | name | x | obs | mod | weight | res.unweighted | res |
|---|------|---|-----|-----|--------|----------------|------|
| 1 | Obs1 | 1 | 50 | 4 | 1 | -46 | -46 |
| 2 | Obs1 | 2 | 150 | 4 | 1 | -146 | -146 |
| 3 | Obs2 | 1 | 1 | 2 | 1 | 1 | 1 |
| 4 | Obs2 | 2 | 2 | 3 | 1 | 1 | 1 |

```
attr("class")
```

```
[1] "modCost"
```

in case the residuals are not weighed and

```
> modCost(mod = Mod, obs = Dat, y = "val", err = "err")
```

```
$model
```

```
[1] 304.3778
```

```
$minlogp
```

```
[1] Inf
```

```
$var
```

| | name | scale | N | SSR.unweighted | SSR.unscaled | SSR |
|---|------|-------|---|----------------|--------------|----------|
| 1 | Obs1 | 1 | 2 | 23432 | 179.3778 | 179.3778 |
| 2 | Obs2 | 1 | 2 | 2 | 125.0000 | 125.0000 |

```
$residuals
```

| | name | x | obs | mod | weight | res.unweighted | res |
|---|------|---|-----|-----|-------------|----------------|-----------|
| 1 | Obs1 | 1 | 50 | 4 | 0.20000000 | -46 | -9.200000 |
| 2 | Obs1 | 2 | 150 | 4 | 0.06666667 | -146 | -9.733333 |
| 3 | Obs2 | 1 | 1 | 2 | 10.00000000 | 1 | 10.000000 |
| 4 | Obs2 | 2 | 2 | 3 | 5.00000000 | 1 | 5.000000 |

```
attr("class")
```

```
[1] "modCost"
```

in case the residuals are weighed by 1/error.

6.3. Model fitting

Assume the following data set (in crosstable (wide) format):

```
> Data <- matrix (nc=2,byrow=2,data=
+ c( 2, 0.14, 4, 0.21, 6, 0.31, 8, 0.40,
+ 10, 0.69, 12, 0.97, 14, 1.42, 16, 2.0,
+ 18, 3.0, 20, 4.5, 22, 6.5, 24, 9.5,
+ 26, 13.5, 28, 20.5, 30, 29, 35, 65, 40, 61)
+ )
> colnames(Data) <- c("time", "Bact")
> head(Data)
```

```
      time Bact
[1,]    2 0.14
[2,]    4 0.21
[3,]    6 0.31
[4,]    8 0.40
[5,]   10 0.69
[6,]   12 0.97
```

and assume that we want to fit the model parameters **gmax** and **eff** to these data.

We first define an objective function that returns the residuals of the model versus the data, as estimated by **modcost**. Input to the function are the current values of the parameters that need to be finetuned and their names (or position in **par**).

```
> Objective <- function(x, parset = names(x)) {
+   pars[parset] <- x
+   tout <- seq(0, 50, by = 0.5)
+   ## output times
+   out <- solveBact(pars, tout)
+   ## Model cost
+   return(modCost(obs = Data, model = out))
+ }
```

First it is instructive to establish which parameters can be identified based on the data set. We assess that by means of the identifiability function **collin**, selecting only the output variables at the instances when there is an observation.

```
> Coll <- collin(sF <- sensFun(func = Objective, parms = pars, varscale = 1))
> Coll
```

```
      gmax eff ks rB dB N collinearity
1      1   1  0  0  0  2           4.5
2      1   0  1  0  0  2          21.1
3      1   0  0  1  0  2           2.1
4      1   0  0  0  1  2           3.7
5      0   1  1  0  0  2           4.5
6      0   1  0  1  0  2           3.3
7      0   1  0  0  1  2           9.2
```

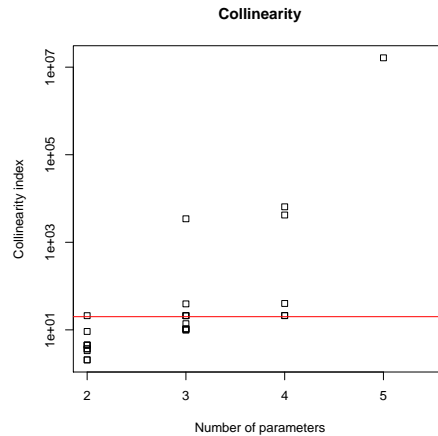


Figure 8: Collinearity analysis - see text for R-code

| | | | | | | | |
|----|---|---|---|---|---|---|------------|
| 8 | 0 | 0 | 1 | 1 | 0 | 2 | 2.1 |
| 9 | 0 | 0 | 1 | 0 | 1 | 2 | 3.7 |
| 10 | 0 | 0 | 0 | 1 | 1 | 2 | 4.5 |
| 11 | 1 | 1 | 1 | 0 | 0 | 3 | 21.1 |
| 12 | 1 | 1 | 0 | 1 | 0 | 3 | 10.2 |
| 13 | 1 | 1 | 0 | 0 | 1 | 3 | 10.5 |
| 14 | 1 | 0 | 1 | 1 | 0 | 3 | 21.1 |
| 15 | 1 | 0 | 1 | 0 | 1 | 3 | 21.1 |
| 16 | 1 | 0 | 0 | 1 | 1 | 3 | 3442.4 |
| 17 | 0 | 1 | 1 | 1 | 0 | 3 | 9.9 |
| 18 | 0 | 1 | 1 | 0 | 1 | 3 | 10.5 |
| 19 | 0 | 1 | 0 | 1 | 1 | 3 | 13.8 |
| 20 | 0 | 0 | 1 | 1 | 1 | 3 | 39.2 |
| 21 | 1 | 1 | 1 | 1 | 0 | 4 | 21.3 |
| 22 | 1 | 1 | 1 | 0 | 1 | 4 | 21.1 |
| 23 | 1 | 1 | 0 | 1 | 1 | 4 | 6465.5 |
| 24 | 1 | 0 | 1 | 1 | 1 | 4 | 4198.5 |
| 25 | 0 | 1 | 1 | 1 | 1 | 4 | 40.1 |
| 26 | 1 | 1 | 1 | 1 | 1 | 5 | 16346953.1 |

The larger the collinearity value, the less identifiable the parameter based on the data. In general a collinearity value less than about 20 is "identifiable". Below we plot the collinearity as a function of the number of parameters selected. We add a line at the height of 20, the critical value:

```
> plot(Coll, log = "y")
> abline(h = 20, col = "red")
```

The collinearity index for parameters `gmax` and `eff` is small enough to enable estimating both parameters.

```
> collin(sF,parset=1:2)
```

```
      gmax eff ks rB dB N collinearity
1      1    1  1  0  0  0  2          4.5
```

We now use function `modFit` to locate the minimum. It includes several fitting procedures; the default one is the Levenberg-Marquardt algorithm.

In the following example, parameters are constrained to be > 0

```
> print(system.time(Fit <- modFit(p = c(gmax = 0.5, eff = 0.5),
+                                f = Objective, lower = c(0.0, 0.0))))
```

```
      user  system elapsed
1.72      0.00      1.71
```

```
> summary(Fit)
```

Parameters:

```
      Estimate Std. Error t value Pr(>|t|)
gmax 0.3003277  0.0004744   633.1  <2e-16 ***
eff   0.7006292  0.0010819   647.6  <2e-16 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.1531 on 15 degrees of freedom
```

Parameter correlation:

```
      gmax    eff
gmax  1.0000 -0.9151
eff   -0.9151  1.0000
```

The model is run with the original and the best-fit parameters, the model cost function estimated and the model outcome compared to data.

```
> init <- solveBact(pars)
> pars[c("gmax", "eff")] <- Fit$par
> out   <- solveBact(pars)
> Cost  <- modCost(obs = Data, model = out)
> Cost
```

```
$model
[1] 0.3514637
```

```
$minlogp
[1] 15.79769
```



```

$var
  name scale  N SSR.unweighted SSR.unscaled      SSR
1 Bact      1 17      0.3514637      0.3514637 0.3514637

$residuals
  name x  obs      mod weight res.unweighted      res
1 Bact 2  0.14 0.1460459      1  0.006045881 0.006045881
2 Bact 4  0.21 0.2132921      1  0.003292129 0.003292129
3 Bact 6  0.31 0.3115005      1  0.001500479 0.001500479
4 Bact 8  0.40 0.4549261      1  0.054926094 0.054926094
5 Bact 10 0.69 0.6643861      1 -0.025613922 -0.025613922
6 Bact 12 0.97 0.9702790      1  0.000278983 0.000278983
7 Bact 14 1.42 1.4169922      1 -0.003007834 -0.003007834
8 Bact 16 2.00 2.0693334      1  0.069333357 0.069333357
9 Bact 18 3.00 3.0219120      1  0.021911987 0.021911987
10 Bact 20 4.50 4.4128138      1 -0.087186196 -0.087186196
11 Bact 22 6.50 6.4435104      1 -0.056489647 -0.056489647
12 Bact 24 9.50 9.4077851      1 -0.092214939 -0.092214939
13 Bact 26 13.50 13.7335832      1  0.233583172 0.233583172
14 Bact 28 20.50 20.0429738      1 -0.457026155 -0.457026155
15 Bact 30 29.00 29.2356342      1  0.235634179 0.235634179
16 Bact 35 65.00 65.0449093      1  0.044909319 0.044909319
17 Bact 40 61.00 60.9533705      1 -0.046629493 -0.046629493

attr(,"class")
[1] "modCost"

> plot(out, init, xlab = "time, hour", ylab = "molC/m3", lwd = 2,
+      obs = Data, obspar = list(cex = 2, pch = 18))
> legend("bottomright", lwd = 2, col = 1:2, lty = 1:2, c("fitted", "original"))

Finally, model residuals are plotted:

> plot(Cost, xlab = "time", ylab = "", main = "residuals")

```

7. Markov chain Monte Carlo

We can use the results of the fit to run a MCMC ([Gelman, Varlin, Stern, and Rubin 2004](#)). Function `modMCMC` implements the delayed rejection (DR) adaptive Metropolis (AM) algorithm ([Haario *et al.* 2006](#)).

The `summary` method of the best fit returns several useful values:

- The model variance `modVariance` is used as the initial model error variance (`var0`) in the MCMC. In each MCMC step, $1/\text{model variance}$ is drawn from a gamma function with parameters `rate` and `shape`, calculated as: `shape = 0.5*N * (1 + pvar0)`, and

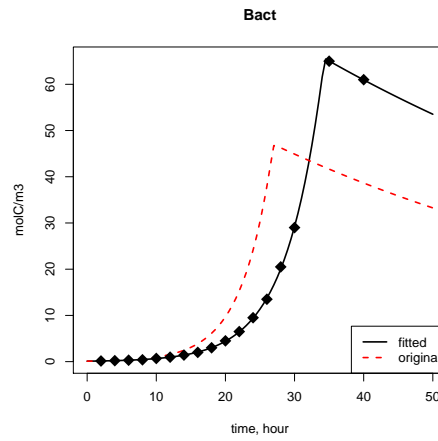


Figure 9: Fitting the model to data - see text for R-code

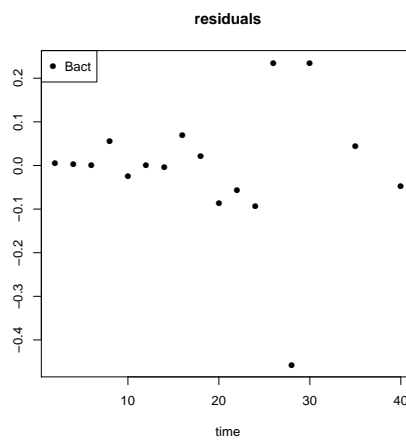


Figure 10: Model-data residuals - see text for R-code

$\text{rate} = 0.5 * (\text{pvar0} * N * \text{var0} + \text{SS})$ and where SS is the current sum of squared residuals, N is the number of data points and pVar0 is a weighing parameter, argument of function `modMCMC`.

- The best-fit parameters are used as initial parameter values for the MCMC (p).
- The parameter covariance returned by the `summary` method, scaled with $2.4^2/\text{length}(p)$, gives a suitable covariance matrix, for generating new parameter values (`jump`).

```
> SF<-summary(Fit)
> SF
```

Parameters:

| | Estimate | Std. Error | t value | Pr(> t) |
|------|-----------|------------|---------|------------|
| gmax | 0.3003277 | 0.0004744 | 633.1 | <2e-16 *** |
| eff | 0.7006292 | 0.0010819 | 647.6 | <2e-16 *** |

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1531 on 15 degrees of freedom

Parameter correlation:

| | gmax | eff |
|------|---------|---------|
| gmax | 1.0000 | -0.9151 |
| eff | -0.9151 | 1.0000 |

```
> SF[]
```

\$residuals

| | Bact | Bact | Bact | Bact | Bact | Bact |
|--------------|--------------|-------------|--------------|--------------|--------------|------|
| 0.006045881 | 0.003292129 | 0.001500479 | 0.054926094 | -0.025613922 | 0.000278983 | |
| | Bact | Bact | Bact | Bact | Bact | Bact |
| -0.003007834 | 0.069333357 | 0.021911987 | -0.087186196 | -0.056489647 | -0.092214939 | |
| | Bact | Bact | Bact | Bact | Bact | |
| 0.233583172 | -0.457026155 | 0.235634179 | 0.044909319 | -0.046629493 | | |

\$residualVariance

```
[1] 0.02343091
```

\$sigma

```
[1] 0.1530716
```

\$modVariance

```
[1] 0.02067434
```

\$df

```
[1] 2 15
```

```

$cov.unscaled
      gmax      eff
gmax 9.604592e-06 -2.004618e-05
eff  -2.004618e-05  4.995866e-05

$cov.scaled
      gmax      eff
gmax 2.250444e-07 -4.697003e-07
eff  -4.697003e-07  1.170577e-06

$info
[1] 3

$niter
[1] 7

$stopmess
[1] "ok"

$par
      Estimate Std. Error t value    Pr(>|t|)
gmax 0.3003277 0.0004743884 633.0839 1.274418e-34
eff  0.7006292 0.0010819321 647.5722 9.076384e-35

> Var0 <- SF$modVariance
> covIni <- SF$cov.scaled *2.4^2/2
> MCMC <- modMCMC(p = coef(Fit), f = Objective, jump = covIni,
+               var0 = Var0, wvar0 = 1)

number of accepted runs: 352 out of 1000 (35.2%)

The plot method shows the trace of the parameters and, in Full is TRUE, also the model
function.

> plot(MCMC, Full = TRUE)

The pairs method plots both parameters as a function of one another:

> pairs(MCMC)

The MCMC output can be used in the functions from the coda package:

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

> cumuplot(MC)

```

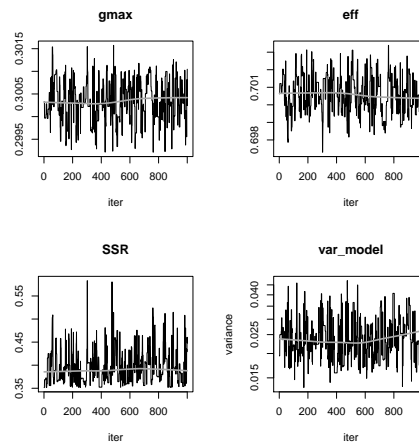


Figure 11: MCMC parameter values per iteration - see text for R-code

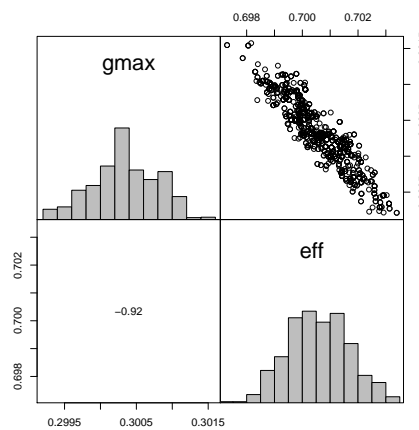


Figure 12: Pairs plot of MCMC results. See text for R-code

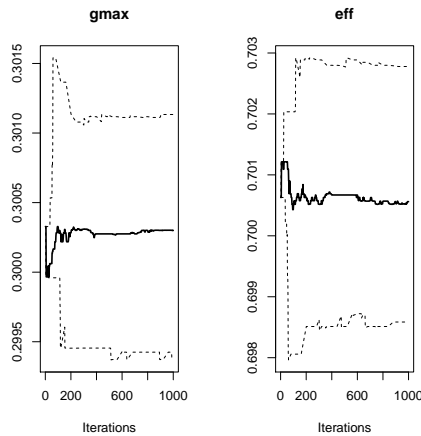


Figure 13: cumulative quantile plot from the MCMC run as from package **coda** - see text for R-code

Finally, we compare the covariances based on generated parameters with the ones from the fit:

```
> cov(MCMC$pars)

           gmax          eff
gmax  2.058520e-07 -4.492594e-07
eff   -4.492594e-07  1.159581e-06

> covIni

           gmax          eff
gmax  6.481277e-07 -1.352737e-06
eff   -1.352737e-06  3.371262e-06
```

8. Distributions

Parameter values can be generated according to 4 different distributions:

Grid, Uniform, Normal, Latinhyper:

```
> par(mfrow = c(2, 2))
> Minmax <- data.frame(min = c(1, 2), max = c(2, 3))
> rownames(Minmax) <- c("par1", "par2")
> Mean    <- c(par1 = 1.5, par2 = 2.5)
> Covar   <- matrix(nr = 2, data = c(2, 2, 2, 3))
> plot(Unif(Minmax, 100), main = "Unif", xlim = c(1, 2), ylim = c(2, 3))
> plot(Grid(Minmax, 100), main = "Grid", xlim = c(1, 2), ylim = c(2, 3))
```

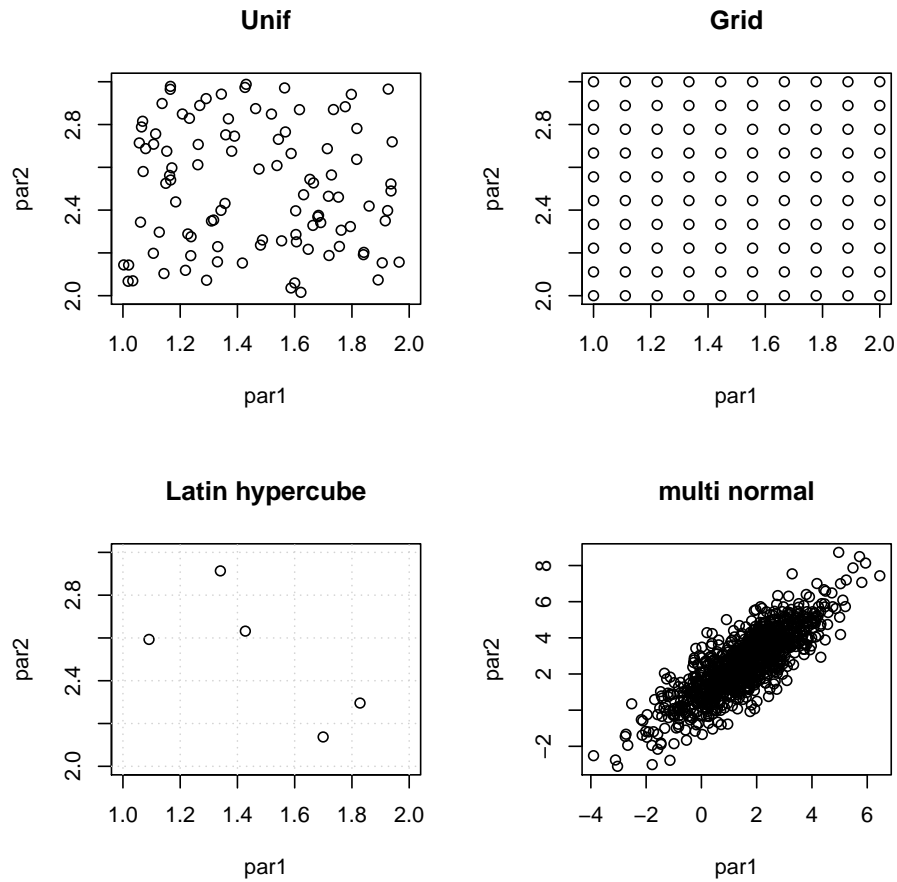


Figure 14: distributions

```
> plot(Latinhyper(Minmax, 5), main = "Latin hypercube", xlim = c(1, 2),
+      ylim = c(2, 3))
> grid()
> plot(Norm(parMean = Mean, parCovar = Covar, num = 1000),
+      main = "multi normal")
```

9. Examples

Several examples are present in subdirectory examples of the package. They include, a.o.:

- **BOD02_FME.R**, a 1-D model of oxygen dynamics in a river. This model consists of two coupled partial differential equations, which are solved to steady-state.
- **ccl4model_FME.R**. Here the functions are applied to "ccl4model", one of the models included in package **deSolve**. This is a model that has been written in **FORTRAN**.

- `Omexdia_FME.R`. Here the functions are applied to a model implemented in **simecol**, an object-oriented framework for ecological modeling ([Petzoldt and Rinke 2007](#)), more specifically in package **simecolModels** ([Petzoldt and Soetaert 2008](#)). The omexdia model is a 1-D diagenetic model.
- `O2profile_FME.R`. This contains a simple model of oxygen, diffusing along a spatial gradient, with imposed upper and lower boundary concentration

10. Finally

This vignette is made with Sweave ([Leisch 2002](#)).

Table 1: Summary of the functions in package FME

| Function | Description |
|---------------------------------|---|
| sensFun | Sensitivity functions |
| sensRange | Sensitivity ranges |
| modCost | Estimates cost functions |
| modFit | Fits a model to data |
| modMCMC | Runs a Markov chain Monte Carlo |
| collin | Estimates collinearity based on sensitivity functions |
| Grid, Norm, Unif, Latinhyper | Generates parameter sets based on grid, normal, uniform or latin hypercube design |

Table 2: Summary of the methods in package FME

| Method | Function | Description |
|---------------|-----------|---|
| summary | modFit | Summary statistics, including parameter std deviations, significance, parameter correlation |
| deviance | modFit | Model deviance (sum of squared residuals) |
| coef | modFit | Values of fitted parameters |
| residuals | modFit | Residuals of model and data |
| df.residual | modFit | Degrees of freedom |
| plot | modFit | Plots results of the fitting |
| print.summary | modFit | Printout of model summary |
| plot | modCost | Plots model-data residuals |
| summary | modMCMC | Summary statistics of sampled parameters |
| plot | modMCMC | Plots all sampled parameters |
| pairs | modMCMC | Pairwise plots all sampled parameters |
| hist | modMCMC | Histogram of all sampled parameters |
| summary | modCRL | Summary statistics of monte carlo variables |
| plot | modCRL | Plots Monte Carlo variables |
| pairs | modCRL | Pairwise plots of Monte Carlo variables |
| hist | modCRL | Histogram of Monte Carlo variables |
| summary | sensFun | Summary statistics of sensitivity functions |
| plot | sensFun | Plots sensitivity functions |
| pairs | sensFun | Pairwise plots of sensitivity functions |
| print.summary | sensFun | Prints summary of sensitivity functions |
| plot.summary | sensFun | Plots summary of sensitivity functions |
| summary | sensRange | Summary statistics of sensitivity range |
| plot | sensRange | Plots sensitivity ranges |
| plot.summary | sensRange | Plots summary of sensitivity ranges |
| print | collin | Prints collinearity results |
| plot | collin | Plots collinearity results |

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*. Chapman & Hall/CRC, Boca Raton, 2nd edition.
- Haario H, Laine M, Mira A, Saksman E (2006). “DRAM: Efficient Adaptive MCMC.” *Statistics and Computing*, **16**, 339–354.
- 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.
- Petzoldt T, Rinke K (2007). “**simecol**: An Object-Oriented Framework for Ecological Modeling in R.” *Journal of Statistical Software*, **22**(9), 1–31. URL <http://www.jstatsoft.org/v22/i09/>.
- Petzoldt T, Soetaert K (2008). *simecolModels: Model Collection for the simecol Package*. R package version 0.3, URL <http://www.simecol.de/>.
- Soetaert K (2009). *rootSolve: Nonlinear Root Finding, Equilibrium and Steady-State Analysis of Ordinary Differential Equations*. R package version 1.6, URL <http://CRAN.R-project.org/package=rootSolve>.
- Soetaert K, deClippele V, Herman PMJ (2002). “**FEMME**, A Flexible Environment for Mathematically Modelling the Environment.” *Ecological Modelling*, **151**, 177–193.
- Soetaert K, Herman PMJ (2009). *A Practical Guide to Ecological Modelling. Using R as a Simulation Platform*. Springer-Verlag, New York.
- 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/>.
- Soetaert K, Petzoldt T, Setzer RW (2010). *deSolve: General Solvers for Initial Value Problems of Ordinary Differential Equations (ODE), Partial Differential Equations (PDE), Differential Algebraic Equations (DAE), and Delay Differential Equations (DDE)*. R package version 1.7, URL <http://CRAN.R-project.org/package=deSolve>.

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>