

# R-package **FME** : inverse modelling, sensitivity, Monte Carlo - applied to a nonlinear model

Karline Soetaert  
NIOO-CEME  
The Netherlands

---

## Abstract

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

This vignette (`vignette("FMEother")`), applies the **FME** functions to a simple nonlinear model.

A similar vignette (`vignette("FMEdyna")`), applies the functions to a dynamic simulation model, solved with integration routines from package **deSolve**

A third vignette, (`vignette("FMEsteady")`), applies **FME** to a partial differential equation, solved with a steady-state solver from package **rootSolve**

`vignette("FMEcmc")` tests the Markov chain Monte Carlo (MCMC) implementation

*Keywords:* steady-state models, differential equations, fitting, sensitivity, Monte Carlo, identifiability, R.

---

## 1. Fitting a Monod function

### 1.1. the model

This example is discussed in (?) (who quotes Berthouex and Brown, 2002. Statistics for environmental engineers, CRC Press).

The following model:

$$y = \theta_1 \cdot \frac{x}{x + \theta_2} + \epsilon$$
$$\epsilon \sim N(0, I\sigma^2)$$

is fitted to data.

### 1.2. implementation in R

```
> require(FME)
```

First we input the observations

2 R-package **FME** : inverse modelling, sensitivity, Monte Carlo - applied to a nonlinear model

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

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

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

### 1.3. Fitting the model to data

We first fit the model to the data.

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

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

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

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

```
user  system elapsed
0.03   0.00   0.37
```

We can estimate and print the summary of fit

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

Parameters:

```
      Estimate Std. Error t value Pr(>|t|)
[1,]  0.14542    0.01564   9.296 0.000242 ***
[2,] 49.05292   17.91196   2.739 0.040862 *
```

---

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

Residual standard error: 0.01278 on 5 degrees of freedom

Parameter correlation:

```
      [,1] [,2]
[1,] 1.0000 0.8926
[2,] 0.8926 1.0000
```

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

```
> x      <-0:375
```

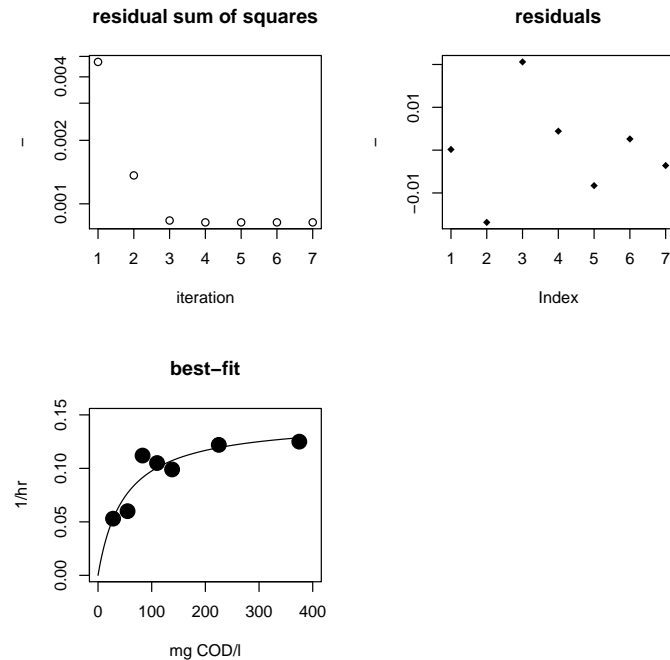


Figure 1: Fit diagnostics of the Monod function - see text for R-code

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

#### 1.4. MCMC analysis

We then run an MCMC analysis. The -scaled- parameter covariances returned from the `summary` function are used as estimate of the proposal covariances (`jump`). Scaling is as in (?).

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

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

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

4 *R*-package **FME** : inverse modelling, sensitivity, Monte Carlo - applied to a nonlinear model

```
+               var0=s2prior,wvar0=1,lower=c(0,0))
+ ))
```

```
number of accepted runs: 983 out of 3000 (32.76667%)
  user  system elapsed
 1.68   0.00   1.75
```

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

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

```
number of accepted runs: 2562 out of 3000 (85.4%)
  user  system elapsed
 4.21   0.00   4.29
```

```
> MCMC$count
```

dr_steps	Alfasteps	num_accepted	num_covupdate
2566	10752	2562	29

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

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

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

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

The pairs plot shows the relationship between the two parameters

```
> pairs(MCMC)
```

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

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

	p1	p2
p1	1.0000000	0.9201083
p2	0.9201083	1.0000000

```
> cov(MCMC$pars)
```

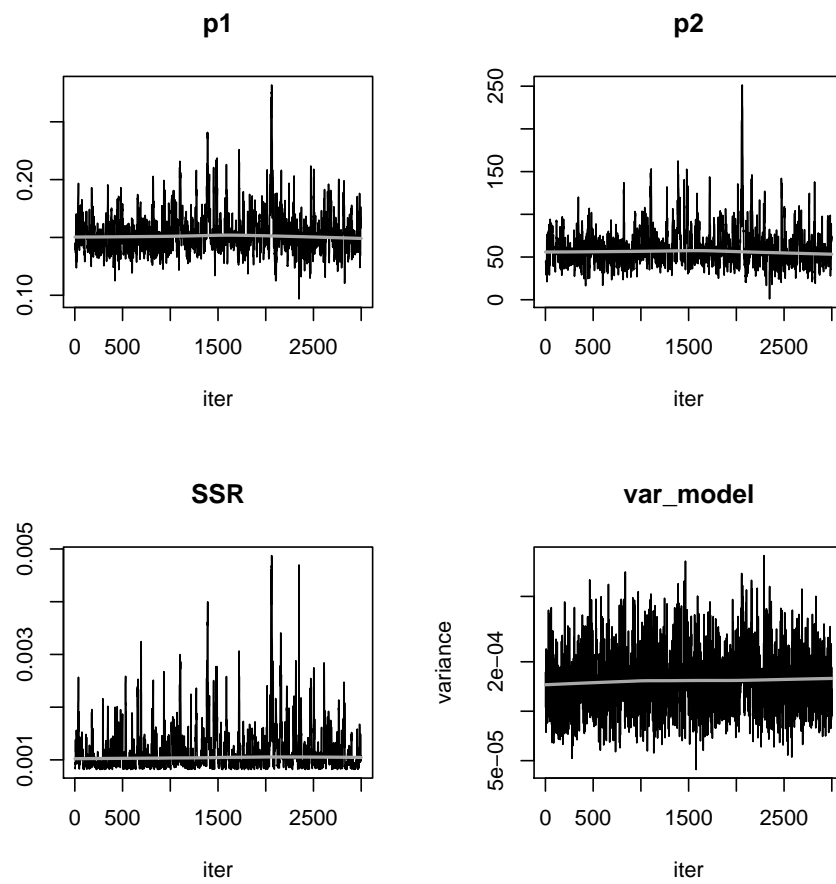


Figure 2: The mcmc - see text for R-code

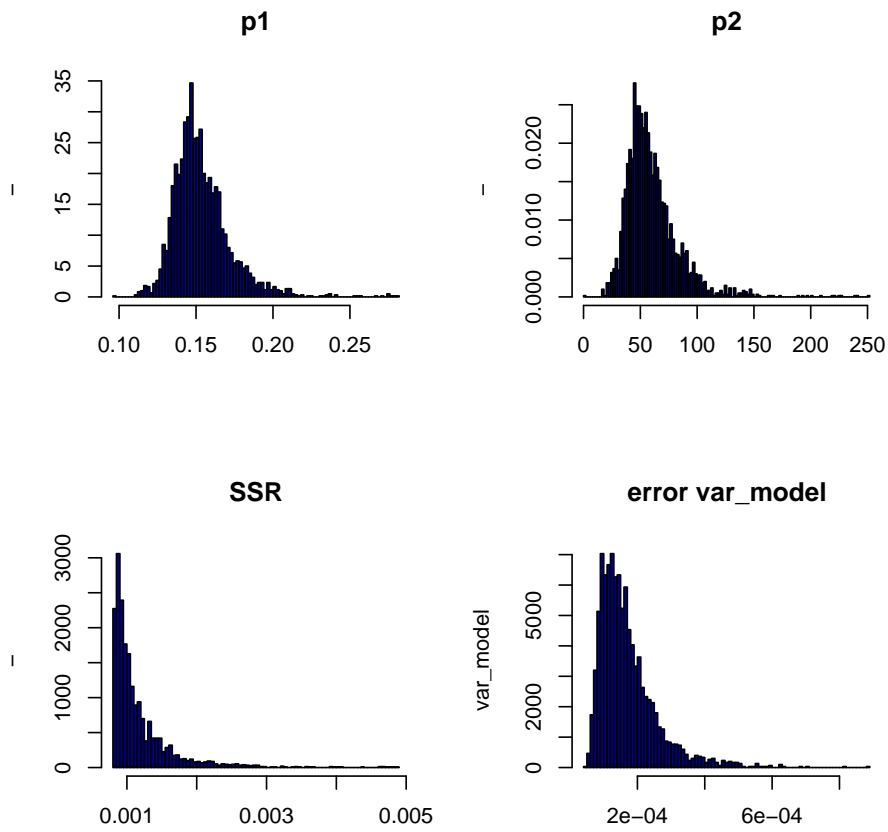


Figure 3: Hist plot - see text for R-code

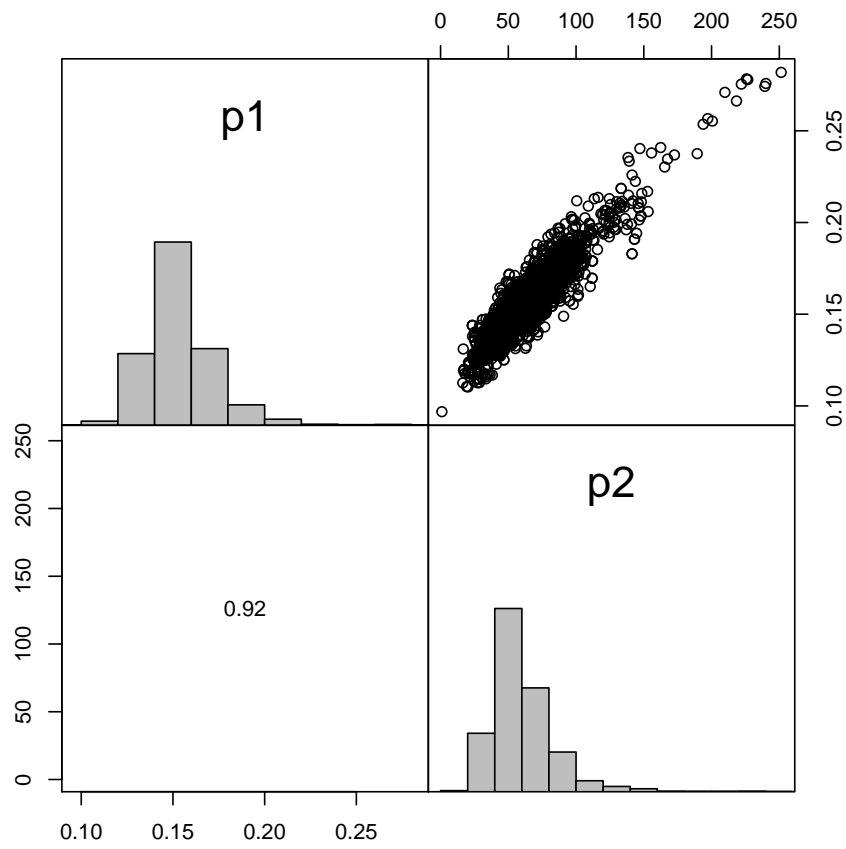


Figure 4: Pairs plot - see text for R-code

```

          p1          p2
p1 0.0003538242  0.4116905
p2 0.4116905389 565.8174574

> sP$cov.scaled

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

```

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

```

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

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

```

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

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

```

> cumuplot(MC)

```

### 1.5. Predictive inference including only parameter uncertainty

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

```

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

```

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

```

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

```

### 1.6. Predictive inference including also measurement error

There is an other source of error, which is not captured by the **senRange** method, i.e. the one corresponding to the measurement error, as represented by the sampled values of  $\sigma^2$ .

This can be estimated by adding normally distribution noise,  $\xi \sim N(0, I\sigma^2)$  to the model predictions produced by the parameters from the MCMC chain. Of course, the  $\sigma$  and parameter sets used must be compatible.

First we need to extract the parameter sets that were effectively used to produce the output in **sR**. This information is kept as an attribute in the output:



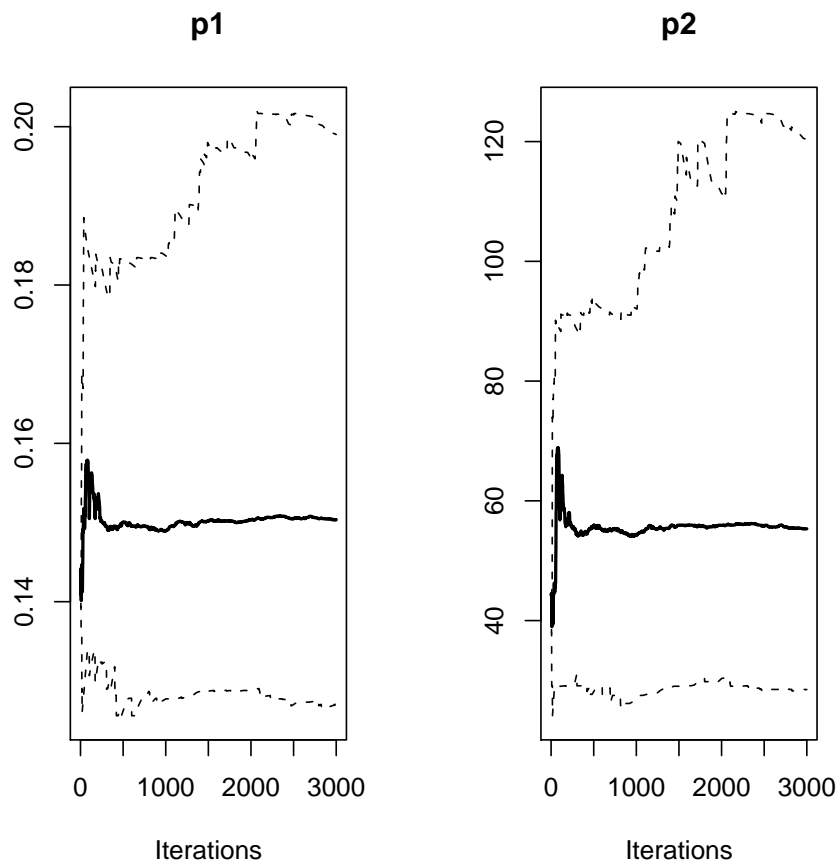


Figure 5: Cumulative quantile plot - see text for R-code

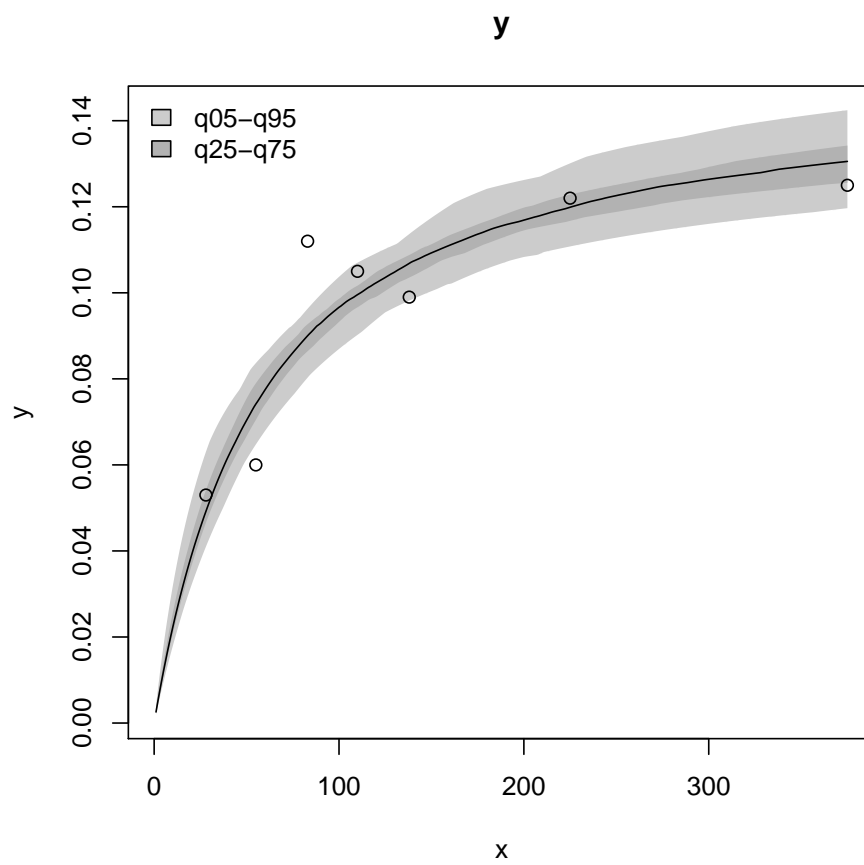


Figure 6: Predictive envelopes of the model, only assuming parameter noise - see text for R-code

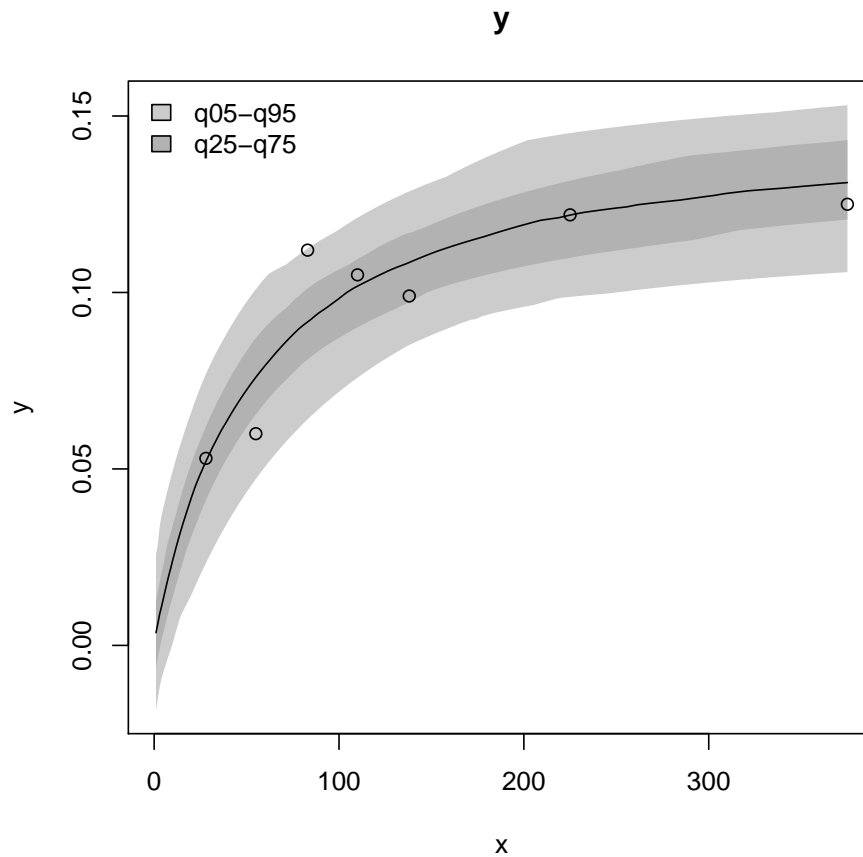


Figure 7: Predictive envelopes of the model, including parameter and measurement noise - see text for R-code

```
> pset <- attributes(sR)$pset
```

Then randomly distributed noise is added; note that the first two columns are parameters; `ivar` points only to the variables.

```
> nout <- nrow(sR)
> sR2 <- sR
> ivar <- 3:ncol(sR)
> error <- rnorm(nout, mean=0, sd=sqrt(MCMC$sig[pset]))
> sR2[,ivar] <- sR2[,ivar] + error
```

```
> plot(summary(sR2), quant=TRUE)
> points(Obs)
```

## 2. finally

This vignette was 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](mailto:k.soetaert@nioo.knaw.nl)

URL: <http://www.nioo.knaw.nl/users/ksoetaert>