**R Package FME : Inverse Modelling, Sensitivity, Monte Carlo – Applied to a Steady-State Model**

Karline Soetaert  
NIOZ Yerseke  
The Netherlands

**Abstract**

R package **FME** ([Soetaert and Petzoldt 2010](#)) contains functions for model calibration, sensitivity, identifiability, and Monte Carlo analysis of nonlinear models.

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

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

A third vignette *(vignette("FMEother"))* applies the functions to a simple nonlinear model.

*vignette("FMEmcmc")* tests the Markov chain Monte Carlo (MCMC) implementation.

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

1. **A steady-state model of oxygen in a marine sediment**

This is a simple model of oxygen in a marine (submersed) sediment, diffusing along a spatial gradient, with imposed upper boundary concentration oxygen is consumed at maximal fixed rate, and including a monod limitation.

See ([Soetaert and Herman 2009](#)) for a description of reaction-transport models.

The constitutive equations are:

\[
\frac{\partial O_2}{\partial t} = -\frac{\partial Flux}{\partial x} - \text{cons} \cdot \frac{O_2}{O_2 + k_s}
\]

\[
\text{Flux} = -D \frac{\partial O_2}{\partial x}
\]

\[
O_2(x = 0) = \text{upO2}
\]

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

First the model parameters are defined...

> `pars <- c(upO2 = 360, # concentration at upper boundary, mmolO2/m3`
> `+ cons = 80, # consumption rate, mmolO2/m3/day`
Next the sediment is vertically subdivided into 100 grid cells, each 0.05 cm thick.

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

The model function takes as input the parameter values and returns the steady-state condition of oxygen. Function `steady.1D` from package `rootSolve` (Soetaert 2009) does this in a very efficient way (see Soetaert and Herman 2009).

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

The model is run

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

and the results plotted...

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

## 2. Global sensitivity analysis: Sensitivity ranges

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

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

    user  system elapsed
    1.7     0.0     1.7

The results can be plotted in two ways:

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

3. Local sensitivity analysis: Sensitivity functions

Local sensitivity analysis starts by calculating the sensitivity functions

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

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

> summary(O2sens)
In bivariate sensitivity the pair-wise relationship and the correlation is estimated and/or plotted:

```r
> pairs(O2sens)
```

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

```
        upO2 cons ks D
    upO2 1.000000 -0.9787945 0.8375806 0.9787945
    cons -0.9787945 1.0000000 -0.9317287 -1.0000000
    ks  0.8375806 -0.9317287 1.0000000 0.9317287
    D    0.9787945 -1.0000000 0.9317287 1.0000000
```

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

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

```
          upO2 cons ks D N collinearity
         1 1 1 0 0 2  7.7
         2 1 0 1 0 2  2.9
```
4. Fitting the model to the data

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

These are the data:

```r
> O2dat <- data.frame(x = seq(0.1, 3.5, by = 0.1),
+                    y = c(279,260,256,220,200,203,189,179,165,140,138,127,116,
+                        109,92,87,78,72,62,55,49,43,32,27,20,15,15,10,8,5,3,2,1,0))
> O2depth <- cbind(name = "O2", O2dat) # oxygen versus depth
> O2flux <- c(UpFlux = 170) # measured flux
```

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

```r
> O2fun2 <- function(pars)
+ {
```
Figure 4: collinearity - see text for R-code

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

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

```r
> Objective <- function (P)
+ {  
+ Pars <- pars
+ Pars[names(P)]<-P
+ mod02 <- O2fun2(Pars)
+ + # Model cost: first the oxygen profile
+ + Cost <- modCost(obs = O2depth, model = mod02[[1]],
```

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

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

user  system elapsed
 0.14   0.00   0.14

> summary(sF)

 value scale L1 L2 Mean   Min   Max N
upO2 360 360 4.3  0.97  4.3 0.5069 13.3 36
cons 80 80  3.7  0.99 -3.6 -15.3722 0.5 36
ks   1  1  0.4  0.14  0.4 -0.0069 3.1 36
D    1  1  3.7  0.99  3.7  0.0342 15.4 36

> collin(sF)

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

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

> collin(sF, parset = c("upO2", "cons", "ks"))
FME – Inverse Modelling, Sensitivity, Monte Carlo with a Steady-State Model

\[
\begin{array}{cccccc}
\text{upO2} & \text{cons} & \text{ks} & \text{D} & \text{N} & \text{collinearity} \\
1 & 1 & 1 & 0 & 3 & 14 \\
\end{array}
\]

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

user  system elapsed 
1.25  0.00  1.26 

\[
> (\text{SFit}<-\text{summary(Fit))}
\]

Parameters:

| Estimate | Std. Error | t value | Pr(>|t|) |
|----------|------------|---------|----------|
| upO2     | 292.937    | 2.104   | <2e-16 *** |
| cons     | 49.687     | 2.366   | <2e-16 *** |
| ks       | 1.297      | 1.362   | 0.348    |

---

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

Residual standard error: 4.401 on 33 degrees of freedom

Parameter correlation:

<table>
<thead>
<tr>
<th></th>
<th>upO2</th>
<th>cons</th>
<th>ks</th>
</tr>
</thead>
<tbody>
<tr>
<td>upO2</td>
<td>1.0000</td>
<td>0.5791</td>
<td>0.2975</td>
</tr>
<tr>
<td>cons</td>
<td>0.5791</td>
<td>1.0000</td>
<td>0.9011</td>
</tr>
<tr>
<td>ks</td>
<td>0.2975</td>
<td>0.9011</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

We next plot the residuals

\[
> \text{plot(Objective(Fit$par), xlab = "depth", ylab = "",} \\
+ \text{main = "residual", legpos = "top")}
\]

and show the best-fit model

\[
> \text{Pars <- pars} \\
> \text{Pars[names(Fit$par)] <- Fit$par} \\
> \text{modO2 <- O2fun(Pars)}
\]

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

5. Running a Markov chain Monte Carlo

We use the parameter covariances of previous fit to update parameters, while the mean squared residual of the fit is use as prior fo the model variance.
Figure 5: residuals - see text for R-code

Figure 6: Best fit model - see text for R-code
We run an adaptive Metropolis, making sure that ks does not become negative...

number of accepted runs: 699 out of 1000 (69.9%)

Plotting the results is similar to previous cases.

or summaries can be created:

> summary(MCMC)

> cor(MCMC$pars)
Figure 7: MCMC plot results - see text for R-code
Figure 8: MCMC histogram results - see text for R-code
Figure 9: MCMC pairs plot - see text for R-code
Figure 10: MCMC range plot - see text for R-code

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

\begin{verbatim}
> plot(summary(sensRange(parms = pars, parInput = MCMC$par, f = O2fun, num = 500)),
+     xyswap = TRUE)
> points(O2depth$y, O2depth$x)
\end{verbatim}

6. Finally

This vignette is made with Sweave (Leisch 2002).

References


Affiliation:
Karline Soetaert
Royal Netherlands Institute of Sea Research (NIOZ)
4401 NT Yerseke, Netherlands
E-mail: k.soetaert@nioz.nl
URL: http://www.nioz.nl