R-package **FME** : inverse modelling, sensitivity, monte carlo - applied to a steady-state 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("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 similation 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 (?) for a description of reaction-transport models.

The constitutive equations are:

$$\frac{\partial O_2}{\partial t} = -\frac{\partial Flux}{\partial x} - cons \cdot \frac{O_2}{O_2 + k_s}$$
$$Flux = -D \cdot \frac{\partial O_2}{\partial x}$$
$$O_2(x=0) = upO2$$

```
> 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
+             ks=1,       # O2 half-saturation ct, mmolO2/m3
+             D=1)        # diffusion coefficient, cm2/d
```

Next the sediment is vertically subdivided into 100 grid cells, each 0.05 cm thick.

```
> 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.band` from package **rootSolve** ((?)) does this in a very efficient way (see (?)).

```
> 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.band(y=runif(n),func=derivs,parms=pars,nspec=1,positive=TRUE)
+  data.frame(X=X,O2=ox$y)
+ }
```

The model is run

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

and the results plotted...

```
> 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`).

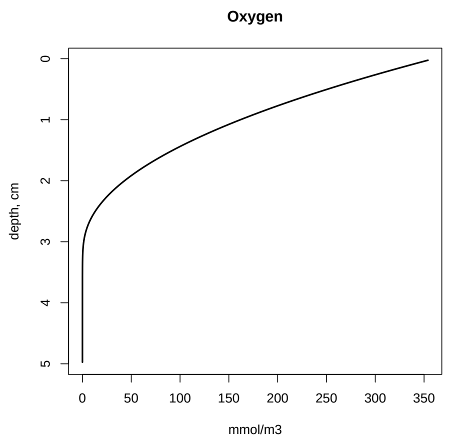

Figure 1: The modeled oxygen profile - see text for R-code

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

```
   user  system elapsed
   0.97    0.00    0.96
```

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 analsysis 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)
```

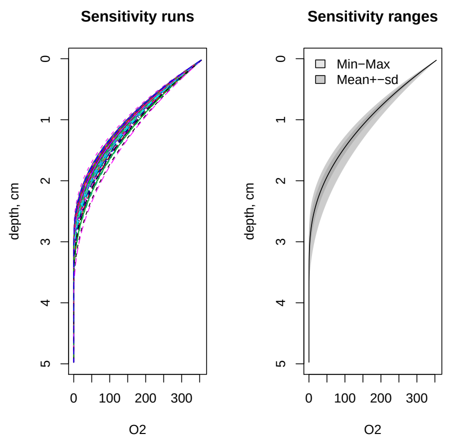

Figure 2: Results of the sensitivity run - left: all model runs, right: summary - see text for R-code

```
     value scale  L1   L2 Mean      Min     Max   N
upO2   360   360 7.0 0.88  7.0  1.0e+00 13.4176 100
cons    80    80 8.1 1.14 -8.1 -2.2e+01 -0.0084 100
ks       1     1 2.2 0.37  2.2  1.2e-04  9.6137 100
D        1     1 8.1 1.14  8.1  8.4e-03 22.0312 100
```

In bivariate sensitivity the pair-wise relationship and the correlation is estimated and/or plotted:

```
> pairs(O2sens)
```

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

```
           upO2       cons         ks          D
upO2  1.0000000 -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 (**?**).

```
> 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
3    1    0  0 1 2          7.7
```

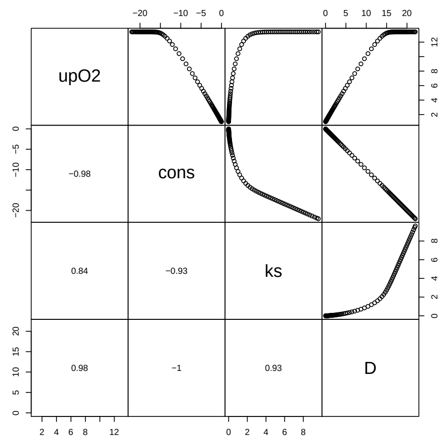

Figure 3: pairs plot - see text for R-code

```
4      0    1  1 0 2         4.4
5      0    1  0 1 2  23364307.5
6      0    0  1 1 2         4.4
7      1    1  1 0 3        25.5
8      1    1  0 1 3  39866094.1
9      1    0  1 1 3        25.5
10     0    1  1 1 3         NaN
11     1    1  1 1 4  83257280.5
```

```
> plot(Coll,log="y")
```

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:

```
> 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,
+           109,92,87,78,72,62,55,49,43,35,32,27,20,15,15,10,8,5,3,2,1,0))
> 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.

```
> O2fun2 <- function(pars)
+ {
+   derivs<-function(t,O2,pars)
```

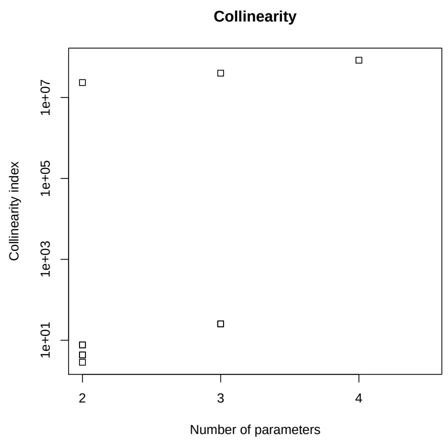

Figure 4: collinearity - see text for R-code

```
+   {
+   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.band(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.

```
> Objective <- function (P)
+ {
+  Pars <- pars
+  Pars[names(P)]<-P
+  modO2 <- O2fun2(Pars)
+
+  # Model cost: first the oxygen profile
+  Cost  <- modCost(obs=O2depth,model=modO2[[1]],x="x",y="y")
+
+  # then the flux
+  modFl <- c(UpFlux=modO2$UpFlux)
+  Cost  <- modCost(obs=O2flux,model=modFl,x=NULL,cost=Cost)
+
+  return(Cost)
+ }
```

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

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

   user  system elapsed
   0.11    0.00    0.12

> summary(sF)

     value scale   L1   L2  Mean      Min  Max  N
upO2   360   360 4.25 0.97  4.25   0.5069 13.3 36
cons    80    80 3.68 0.99 -3.65 -15.3722  0.5 36
ks       1     1 0.40 0.14  0.40  -0.0069  3.1 36
D        1     1 3.68 0.99  3.68   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"))

  upO2 cons ks D N collinearity
1    1    1  1 0 3           14
```

```
> print(system.time(
+ Fit<-modFit(p=c(upO2=360,cons=80,ks=1),
+                   f=Objective,lower=c(0,0,0))
+                   ))

   user  system elapsed 
   1.00    0.00    1.02 

> (SFit<-summary(Fit))

Parameters:
     Estimate Std. Error t value Pr(>|t|)    
upO2  292.937      2.104 139.241   <2e-16 ***
cons   49.686      2.367  20.990   <2e-16 ***
ks      1.296      1.363   0.951    0.349    
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Š.Š 0.1 Š Š 1 

Residual standard error: 4.401 on 33 degrees of freedom

Parameter correlation:
       upO2   cons     ks
upO2 1.0000 0.5791 0.2976
cons 0.5791 1.0000 0.9013
ks   0.2976 0.9013 1.0000
```

We next plot the residuals

```
> plot(Objective(Fit$par),xlab="depth",ylab="",main="residual",legpos="top")
```

and show the best-fit model

```
> Pars <- pars
> Pars[names(Fit$par)]<- Fit$par
> modO2 <- O2fun(Pars)

> plot(O2depth$y,O2depth$x,ylim=rev(range(O2depth$x)),pch=18,
+      main="Oxygen-fitted", xlab="mmol/m3",ylab="depth, cm")
> lines(modO2$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.

```
> Covar   <- SFit$cov.scaled * 2.4^2/3
> s2prior <- SFit$modVariance
```

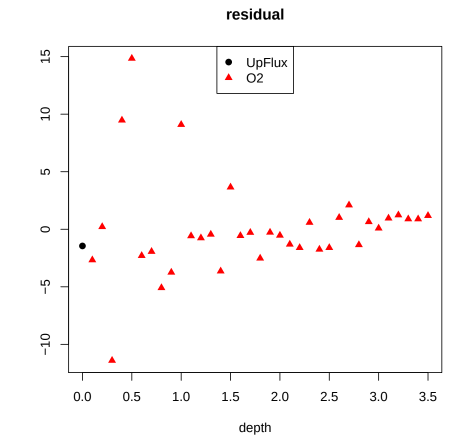

Figure 5: residuals - see text for R-code

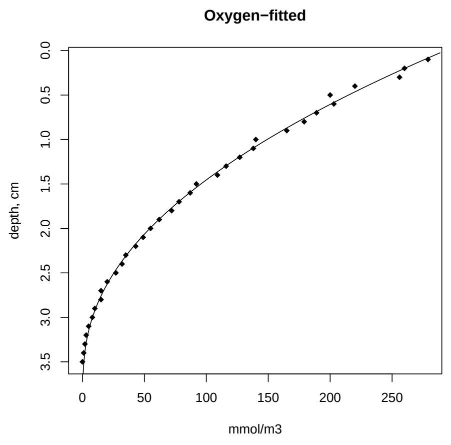

Figure 6: Best fit model - see text for R-code

We run an adaptive metropolis, making sure that ks does not become negative...

```
> print(system.time(
+ MCMC <- modMCMC(f=Objective,p=Fit$par,jump=Covar,niter=1000,ntrydr=2,
+                 var0=s2prior,wvar0=1,updatecov=100,lower=c(NA,NA,0))
+ ))

number of accepted runs: 731 out of 1000 (73.1%)
   user  system elapsed 
  29.95    0.01   30.35 

> MCMC$count

     dr_steps     Alfasteps  num_accepted num_covupdate 
          639          1917           731             9 
```

Plotting the results is similar to previous cases.

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

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

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

or summaries can be created:

```
> summary(MCMC)

           upO2      cons          ks    var_model
mean 294.316196 53.032356  3.67039123   247.906307
sd     5.196583  5.366047  3.50392126  1454.649189
min  280.062805 44.978733  0.01798573     1.706588
max  319.980637 91.210222 26.38313193 29764.048443
q025 291.368485 49.876739  1.46534978    11.364723
q050 293.371337 51.376391  2.58907940    27.633790
q075 295.637797 54.711478  4.64807462    90.684463

> cor(MCMC$pars)

          upO2      cons        ks
upO2 1.0000000 0.7422241 0.4345458
cons 0.7422241 1.0000000 0.8783082
ks   0.4345458 0.8783082 1.0000000
```

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

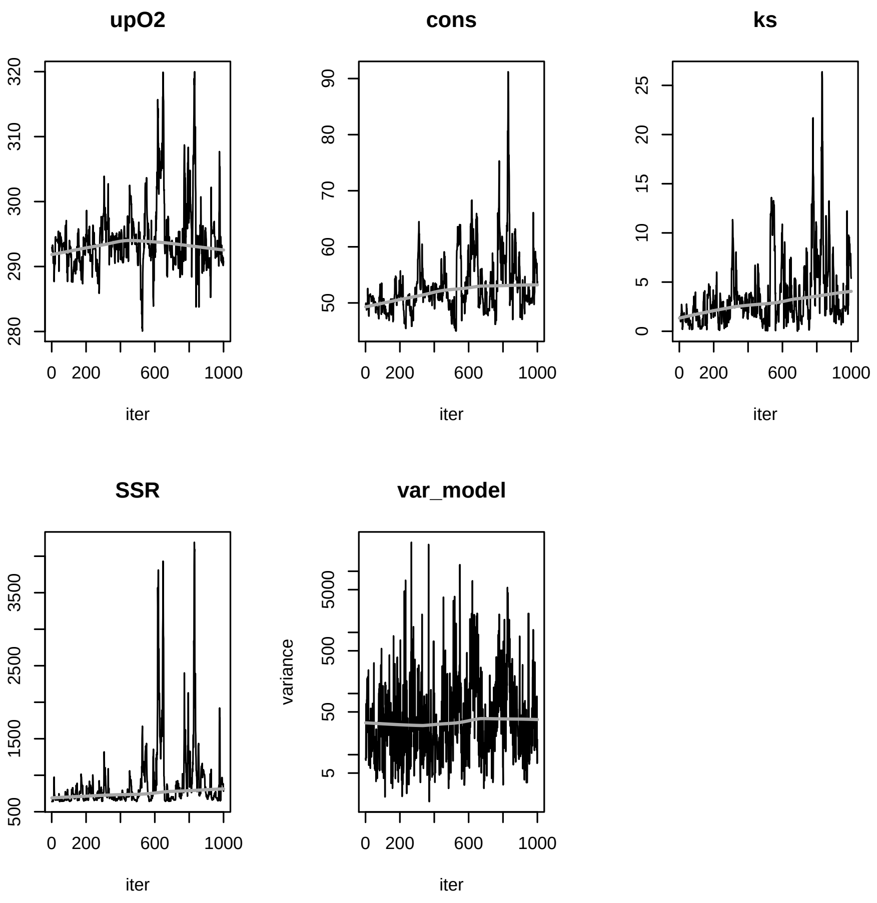

Figure 7: MCMC plot results - see text for R-code

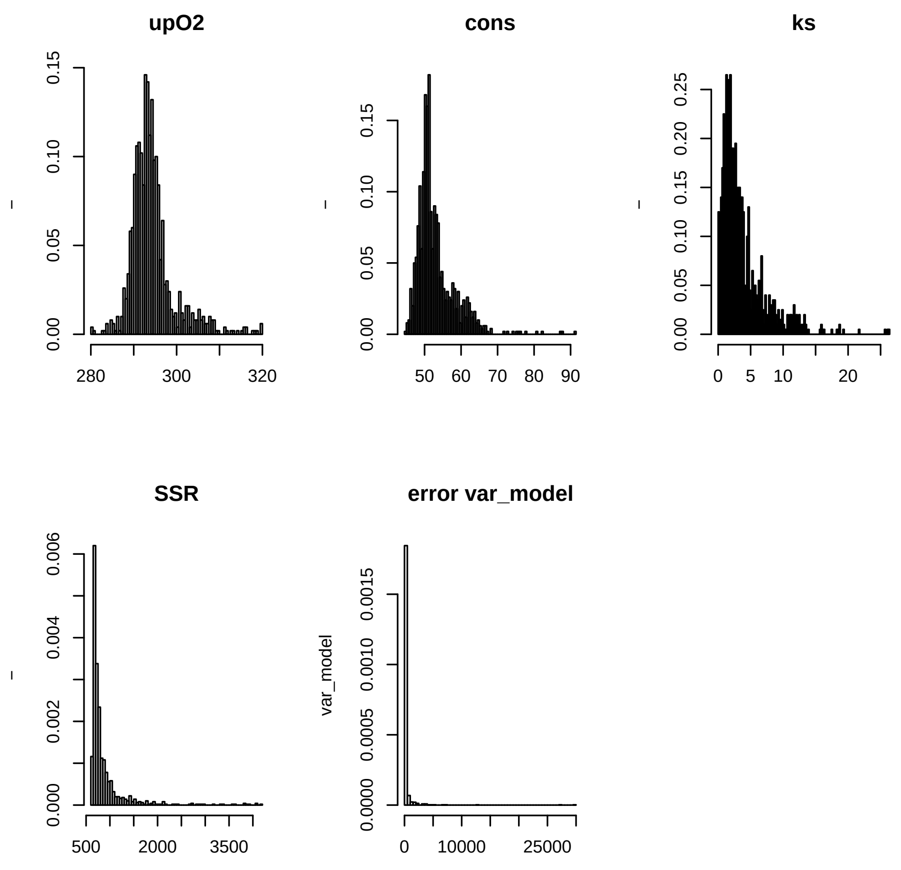

Figure 8: MCMC histogram results - see text for R-code

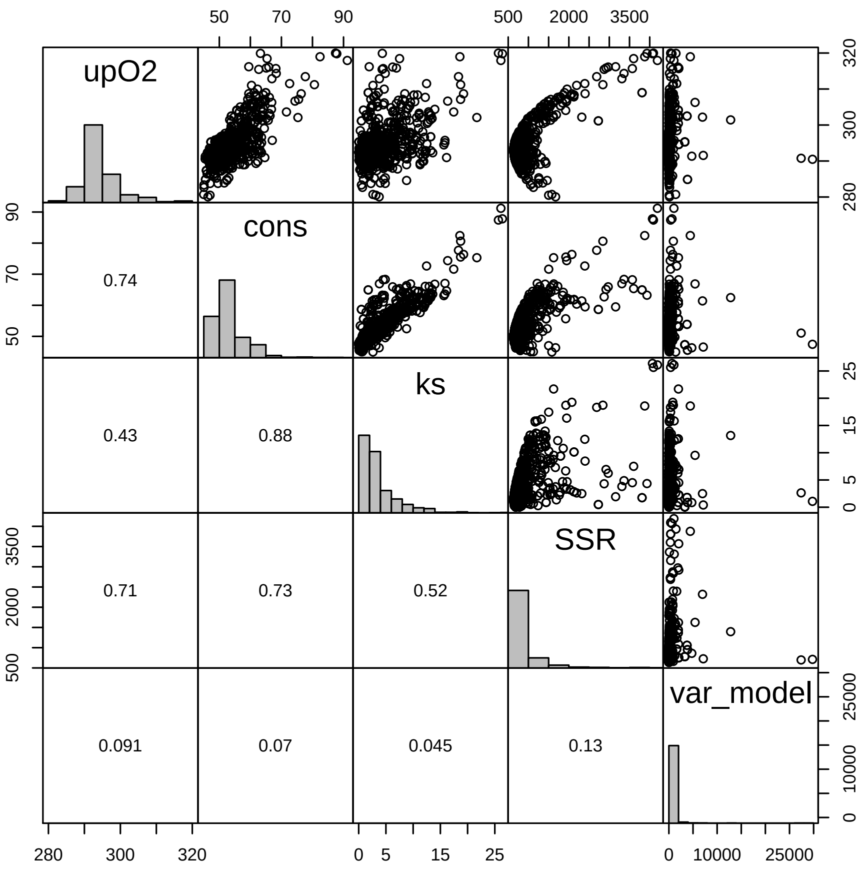

Figure 9: MCMC pairs plot - see text for R-code

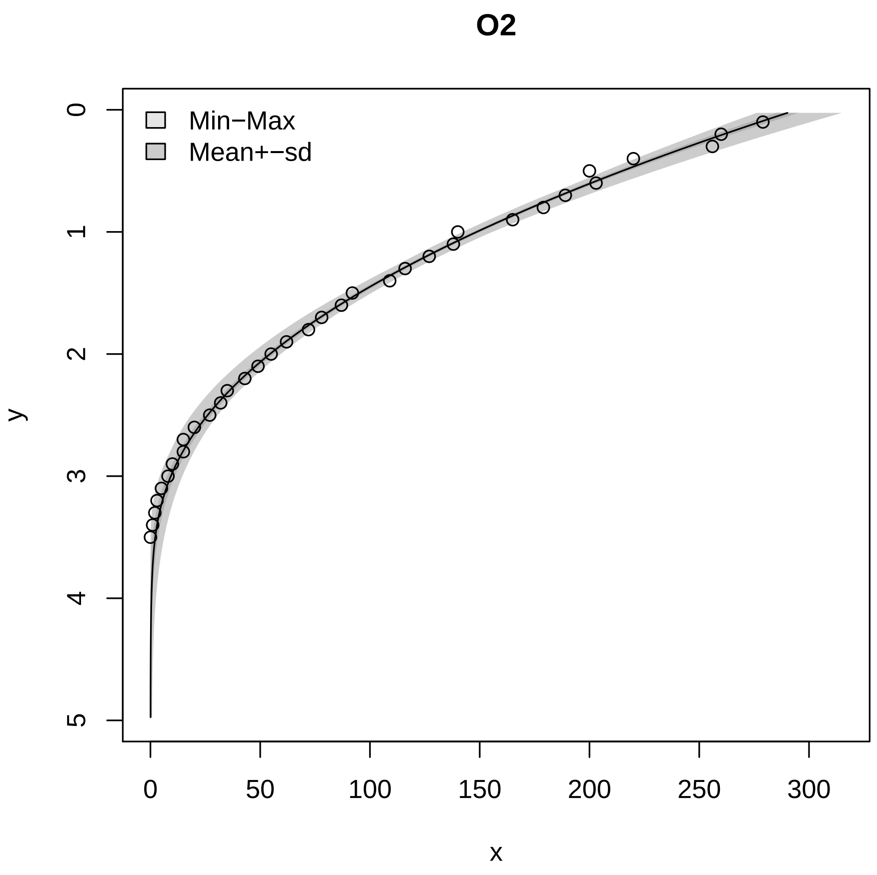

Figure 10: MCMC range plot - see text for R-code

```
> plot(summary(sensRange(parms=pars,parInput=MCMC$par,f=O2fun,num=500)),
+    xyswap=TRUE)
> points(O2depth$y,O2depth$x)
```

6. 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