Solving Differential Equations in R (book) - BVP examples

Karline Soetaert
Royal Netherlands Institute of Sea Research (NIOZ)
Yerseke, The Netherlands

Abstract

This vignette contains the R-examples of chapter 12 from the book:

Soetaert, K., Cash, J.R. and Mazzia, F. (2012). Solving Differential Equations in R. UseR series, Springer, 248 pp.

www.springer.com/statistics/computational+statistics/book/978-3-642-28069-6

Chapter 12. Solving Boundary Value Problems in R.

Here the code is given without documentation. Of course, much more information about each problem can be found in the book.

Keywords: partial differential equations, initial value problems, examples, R.

1. A simple BVP Example

```
prob7 <- function(x, y, pars) {
 list(c( y[2],
        1/eps * (-x*y[2] + y[1] - (1+eps*pi*pi)*
             cos(pi*x) - pi*x*sin(pi*x))))
}
eps  <- 0.1
sol <- bvptwp(yini = c(y = -1, y1 = NA),
             yend = c(1, NA), func = prob7,
             x = seq(-1, 1, by = 0.01))
prob7_2 <- function(x, y, pars) {
 list(1/eps * (-x*y[2] + y[1] - (1+eps*pi*pi)*
             cos(pi*x) - pi*x*sin(pi*x)))
}
sol1 <- bvptwp(yini = c(y = -1, y1 = NA),
             yend = c(1, NA), func = prob7_2,
             order = 2, x = seq(-1, 1, by = 0.01))
head(sol, n=3)
```

```
        x          y          y1
[1,] -1.00 -1.0000000 0.001699844
```

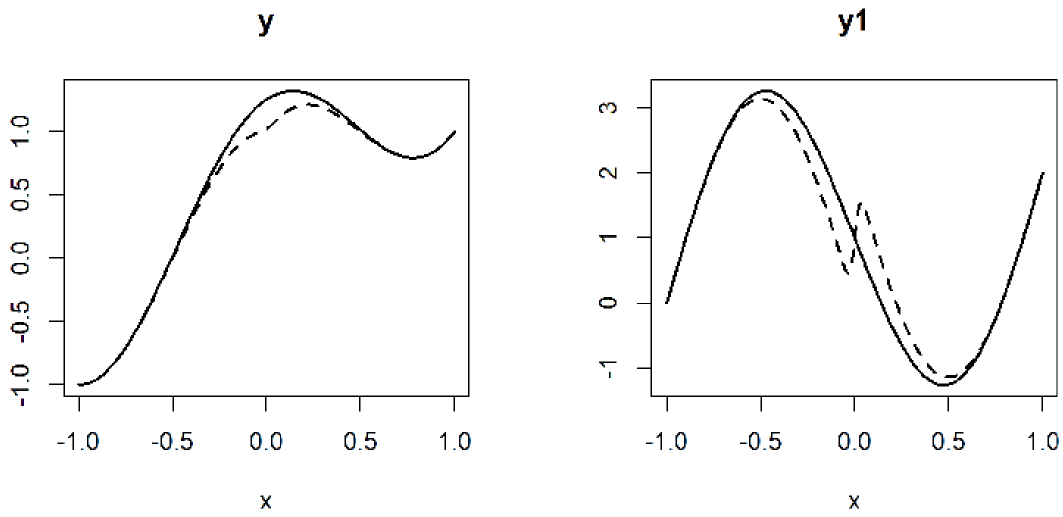

Figure 1: Solution of the test problem 7. See book for more information.

```
[2,] -0.99 -0.9994887 0.100558398
[3,] -0.98 -0.9979891 0.199338166
```

```
eps <-0.0005
sol2 <- bvptwp(yini = c(y = -1, y1 = NA),
              yend = c(1, NA), func = prob7,
              x =  seq(-1, 1, by=0.01))

plot(sol, sol2, col = "black", lty = c("solid", "dashed"),
     lwd = 2)
```

2. A More Complex BVP Example

```
swirl <- function (t, Y, eps)  {
  with(as.list(Y),
    list(c((g*f1 - f*g1)/eps,
           (-f*f3 - g*g1)/eps))
      )
}
eps  <- 0.001
x    <- seq(from = 0, to = 1, length = 100)
yini <- c(g = -1, g1 = NA, f = 0, f1 = 0, f2 = NA, f3 = NA)
yend <- c(1, NA, 0, 0, NA, NA)
Soltwp <- bvptwp(x = x, func = swirl, order = c(2, 4),
                 par = eps, yini = yini, yend = yend)

pairs(Soltwp, main = "swirling flow III, eps=0.001")


diagnostics(Soltwp)
```

```
--------------------
solved with  bvptwp
--------------------
  Integration was successful.

  1 The return code                                  : 0
  2 The number of function evaluations               : 28507
  3 The number of jacobian evaluations               : 3179
  4 The number of boundary evaluations               : 84
  5 The number of boundary jacobian evaluations      : 66
  6 The number of steps                              : 18
  7 The number of mesh resets                        : 1
  8 The maximal number of mesh points                : 1000
  9 The actual number of mesh points                 : 199
 10 The size of the real work array                  : 280660
 11 The size of the integer work array               : 14018


--------------------
conditioning pars
--------------------

  1 kappa1  : 12601.34
  2 gamma1  : 818.5373
  3 sigma   : 36.8086
  4 kappa   : 13175.8
  5 kappa2  : 574.46
```

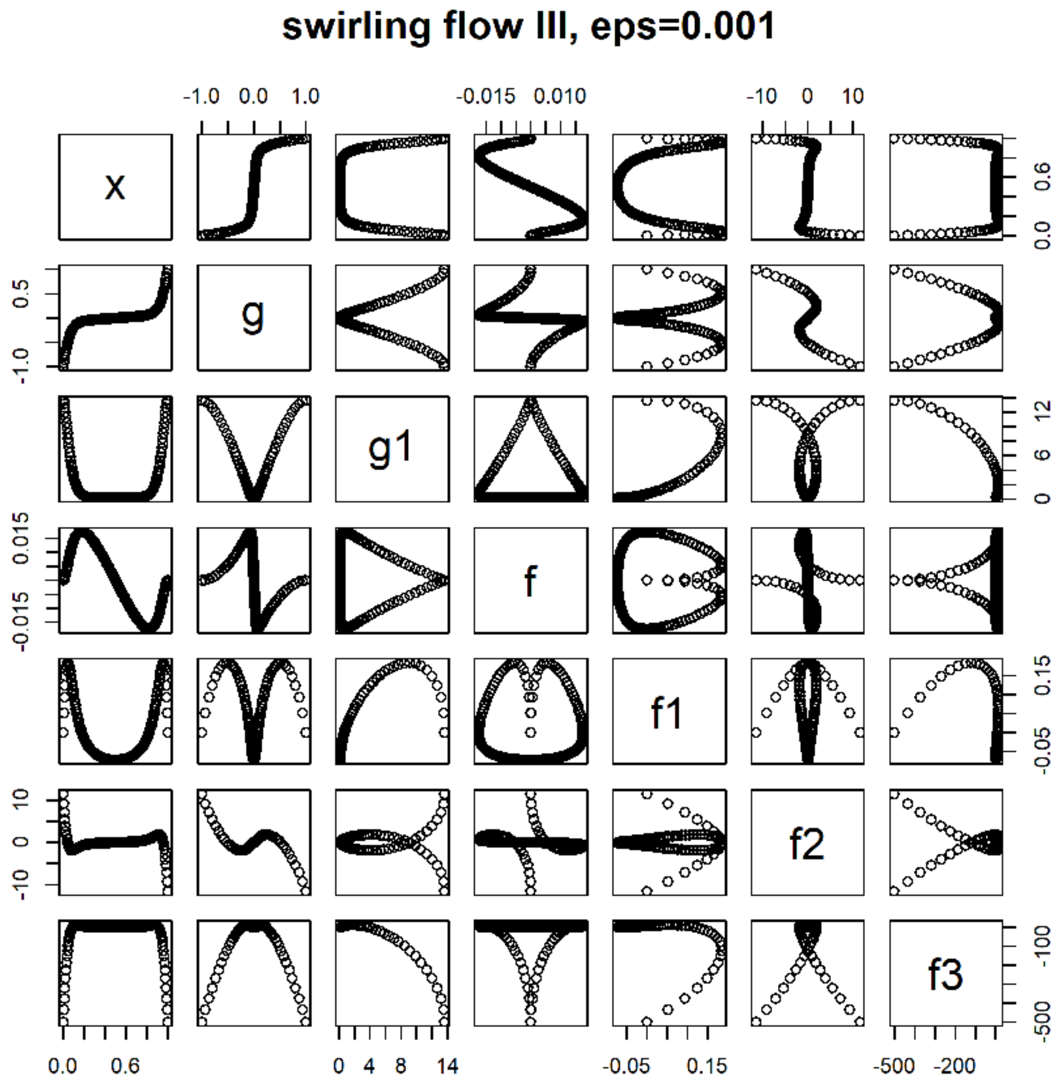

Figure 2: The swirling flow III problem. See book for explanation.

3. Complex Initial or End Conditions

```
musn <- function(x, Y, pars) {
  with (as.list(Y),  {
    du <- 0.5 * u * (w - u) / v
    dv <- -0.5 * (w - u)
    dw <- (0.9 - 1000 * (w - y) - 0.5 * w * (w - u))/z
    dz <- 0.5 * (w - u)
    dy <- -100 * (y - w)
    return(list(c(du, dv, dw, dz, dy)))
  })
}
bound <- function(i, Y, pars) {
  with (as.list(Y), {
    if (i == 1) return (u - 1)
    if (i == 2) return (v - 1)
    if (i == 3) return (w - 1)
    if (i == 4) return (z + 10)
    if (i == 5) return (w - y)
 })
}
xguess <- seq(0, 1, length.out = 5)
yguess <- matrix(ncol = 5,
                data = (rep(c(1, 1, 1, -10, 0.91), 5)))
rownames(yguess) <- c("u", "v", "w", "z", "y")
xguess
```

```
[1] 0.00 0.25 0.50 0.75 1.00
```

```
yguess
```

```
    [,1]   [,2]   [,3]   [,4]   [,5]
u   1.00   1.00   1.00   1.00   1.00
v   1.00   1.00   1.00   1.00   1.00
w   1.00   1.00   1.00   1.00   1.00
z -10.00 -10.00 -10.00 -10.00 -10.00
y   0.91   0.91   0.91   0.91   0.91
```

```
Sol <- bvptwp(x = x, func = musn, bound = bound,
              xguess = xguess, yguess = yguess,
              leftbc = 4, atol = 1e-10)
yguess <- matrix(ncol = 5, data = (rep(c(1,1,1, 10, 0.91), 5)))
rownames(yguess) <- c("u", "v", "w", "z", "y")
Sol2<- bvpcol(x = x, func = musn, bound = bound,
              xguess = xguess, yguess = yguess,
              leftbc = 4, atol = 1e-10)
```

```
plot(Sol, Sol2, which = "y", lwd = 2)
```

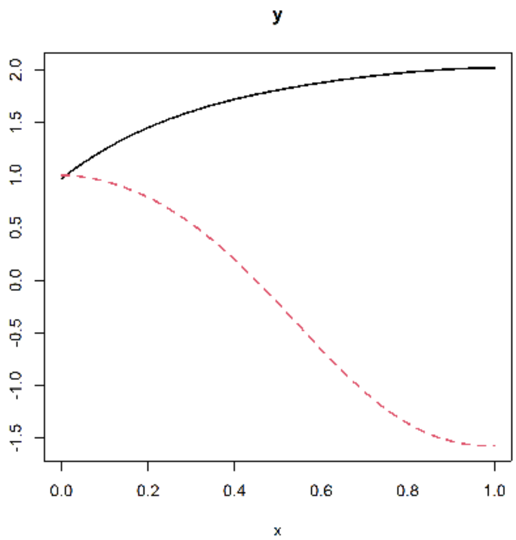

Figure 3: The musn problem. See book for explanation.

4. Solving a Boundary Value Problem using Continuation

```
Prob19 <- function(x, y, eps) {
  pix = pi*x
  list(c(y[2],
      (pi/2*sin(pix/2)*exp(2*y[1])-exp(y[1])*y[2])/eps))
}
x <- seq(0, 1, by = 0.01)
eps <- 1e-2
mod1 <- bvptwp(func = Prob19, yini = c(0, NA), yend = c(0, NA),
               x = x,            par = eps)
diagnostics(mod1)
```

```
--------------------
solved with  bvptwp
--------------------
  Integration was successful.

  1 The return code                                : 0
  2 The number of function evaluations             : 18057
  3 The number of jacobian evaluations             : 3091
  4 The number of boundary evaluations             : 40
  5 The number of boundary jacobian evaluations    : 26
  6 The number of steps                            : 29
  7 The number of mesh resets                      : 1
  8 The maximal number of mesh points              : 1000
  9 The actual number of mesh points               : 150
 10 The size of the real work array                : 56108
 11 The size of the integer work array             : 6006


--------------------
conditioning pars
--------------------

  1 kappa1  : 125.6702
  2 gamma1  : 2.236132
  3 sigma   : 93.77898
  4 kappa   : 168.4309
  5 kappa2  : 42.7607
```

```
plot(mod1,  lwd = 2)
```

```
xguess <- mod1[,1]
yguess <- t(mod1[,2:3])
```

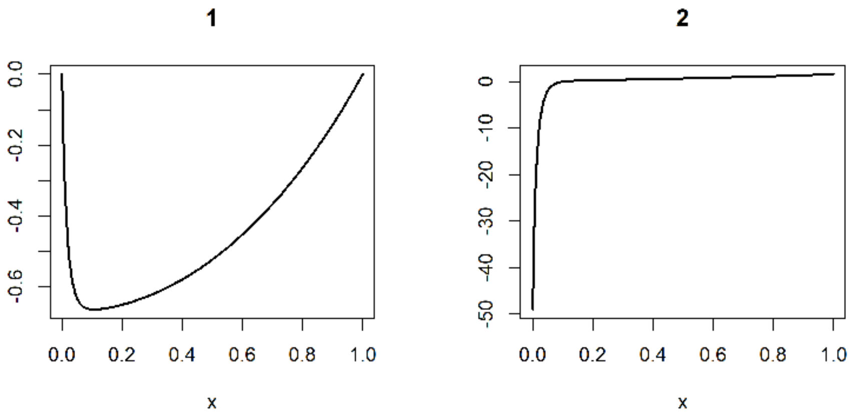

Figure 4: Solution of the test problem 19. See book for explanation.

```
eps <- 1e-3
mod2 <- bvpcol(func = Prob19, yini = c(0, NA), yend = c(0, NA),
              x = x, par = eps,
              xguess = xguess, yguess = yguess)
eps <- 1e-7
mod2 <- bvpcol(func = Prob19, yini = c(0, NA), yend = c(0, NA),
              x = x, par = eps, eps = eps, atol = 1e-4)
diagnostics(mod2)
```

```
--------------------
solved with  bvpcol
--------------------
  Integration was successful.

  1 The return code                                   : 1
  2 The number of function evaluations                : 22537
  3 The number of jacobian evaluations                : 4568
  4 The number of boundary evaluations                : 170
  5 The number of boundary jacobian evaluations       : 96
  6 The number of continuation steps                  : 10
  7 The number of succesfull continuation steps       : 10
  8 The actual number of mesh points                  : 50
  9 The number of collocation points per subinterval  : 4
 10 The number of equations                           : 2
 11 The number of components (variables)              : 2

    The problem was solved for final eps equal to     : 1e-07
```

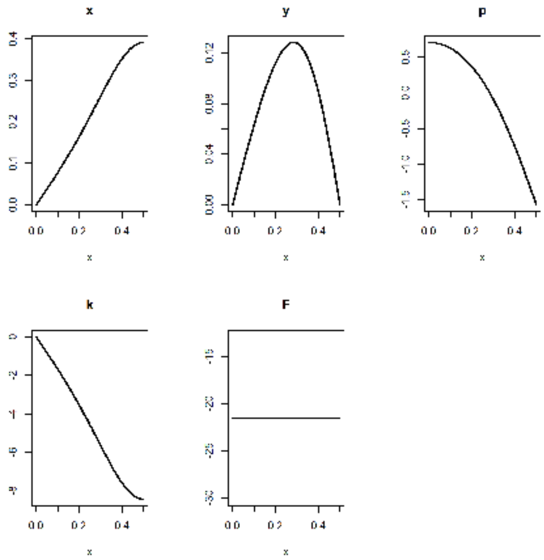

Figure 5: Solution of the elastica problem. See book for explanation.

5. BVP with Unknown Constants

5.1. Elastica Problem

```
Elastica <- function (x, y, pars) {
  list( c(cos(y[3]),
          sin(y[3]),
          y[4],
          y[5] * cos(y[3]),
          0))
}
bvpsol <- bvpcol(func = Elastica,
      yini = c(x = 0,  y = 0, p = NA,    k = 0,  F = NA),
      yend = c(x = NA, y = 0, p = -pi/2, k = NA, F = NA),
      x = seq(from = 0, to = 0.5, by = 0.01))
bvpsol[1,"F"]
```

```
       F
-21.54909
```

```
plot(bvpsol, lwd = 2)
```

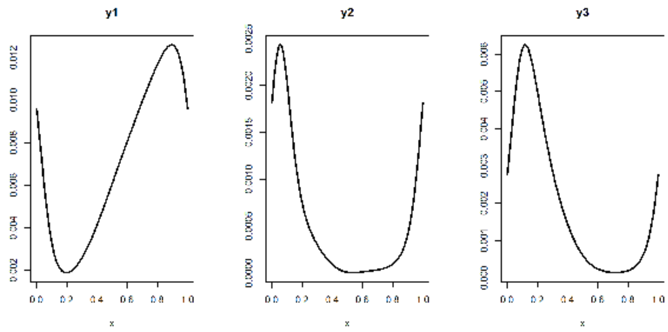

Figure 6: The measel problem. See book for explanation

5.2. Non-separated Boundary Conditions

```
measel <- function(t, y, pars)  {
  bet <- 1575 * (1 + cos(2 * pi * t))
  dy1 <- mu - bet * y[1] * y[3]
  dy2 <- bet * y[1] * y[3] - y[2] / lam
  dy3 <- y[2] / lam - y[3] / eta
  dy4 <- 0
  dy5 <- 0
  dy6 <- 0
  list(c(dy1, dy2, dy3, dy4, dy5, dy6))
}
bound <- function(i, y, pars) {
  if ( i == 1 | i == 4) return( y[1] - y[4])
  if ( i == 2 | i == 5) return( y[2] - y[5])
  if ( i == 3 | i == 6) return( y[3] - y[6])
}
mu  <- 0.02 ; lam <- 0.0279 ; eta <- 0.1
x <- seq(from = 0, to = 1, by = 0.01)
Sola <- bvpshoot(func = measel, bound = bound,
    x = x, leftbc = 3, atol = 1e-12, rtol = 1e-12,
    guess = c(y1 = 1, y2 = 1, y3 = 1, y4 = 1, y5 = 1, y6 = 1))
yguess <- matrix(ncol = length(x), nrow = 6, data = 1)
rownames(yguess) <- paste("y", 1:6, sep="")
Sol  <- bvptwp  (func = measel, bound = bound,
          x = x, leftbc = 3, xguess = x, yguess = yguess)
max(abs(Sol[1,-1] - Sol[nrow(Sol),-1]))
```

```
[1] 0
```

```
plot(Sol, lwd = 2, which = 1:3, mfrow = c(1, 3))
```

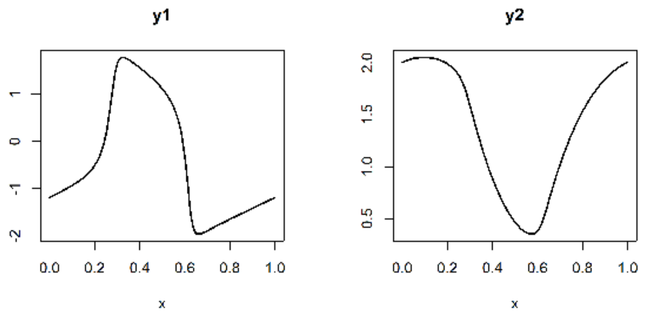

Figure 7: The nerve impulse problem. See book for explanation.

5.3. Unknown Integration Interval

```
nerve <- function (x, y, p)
  list(c(3 * y[3] * (y[1] + y[2] - 1/3 * (y[1]^3) -1.3),
        (-1/3) * y[3] * (y[1] - 0.7 + 0.8 * y[2]) ,
        0,
        0,
        0)
  )
bound <- function(i, y, p) {
 if (i ==1) return (-y[3]* (y[1] - 0.7 + 0.8*y[2])/3 -1)
 if (i ==2) return (y[1] - y[4] )
 if (i ==3) return (y[2] - y[5] )
 if (i ==4) return (y[1] - y[4] )
 if (i ==5) return (y[2] - y[5] )
}
xguess <- seq(0, 1, by = 0.1)
yguess <- matrix(nrow = 5, ncol = length(xguess), data = 5.)
yguess[1,] <- sin(2 * pi * xguess)
yguess[2,] <- cos(2 * pi * xguess)
rownames(yguess) <- c("y1", "y2", "T", "y1ini", "y2ini")
Sol  <- bvptwp(func = nerve, bound = bound,
              x = seq(0, 1, by = 0.01),leftbc = 3,
              xguess = xguess, yguess = yguess)
Sol[1,]
```

```
       x        y1        y2         T     y1ini     y2ini
0.000000 -1.183453  2.004203 10.710808 -1.183453  2.004203
```

```
plot(Sol, lwd = 2, which = c("y1", "y2"))
```

6. Integral Constraints

```
integro <- function (t, u, p)
  list(c(u[2], -p*exp(u[1]), u[2]))
yini <- c(u1 = 1, u2 = NA, I = 0)
yend <- c(NA, NA, 1)
x    <- seq(from = 0, to = 1, by = 0.01)
out <- bvpcol (yini = yini, yend = yend, func = integro,
               x = x, parms = 0.5)
```

7. Sturm-Liouville Problems

```
Sturm <- function(x, y, p) {
   dy1 <- y[2]
   dy2 <- -y[3] * y[1]
   dy3 <- 0.
   list( c(dy1, dy2, dy3))
}
yini <- c(y = 0, dy = 1,  lambda = NA)
yend <- c(y = 0, dy = NA, lambda = NA)
x    <- seq(from = 0, to = pi,  by = pi/10)
S1 <- bvpshoot(yini = yini, yend = yend, func = Sturm,
               parms = 0, x = x)
(lambda1 <- S1[1, "lambda"])
```

```
lambda
     1
```

```
ana <- function(x, lambda) sin(x*sqrt(lambda))/sqrt(lambda)
max (abs(S1[,2]-ana(S1[,1],lambda1)))
```

```
[1] 8.987612e-08
```

8. A Reaction Transport Problem

```
N    <- 1000
Grid <- setup.grid.1D(N = N, L = 100000)
v <- 1000; D <- 1e7; O2s <- 300;
NH3in <- 500; O2in <- 100; NO3in <- 50
r <- 0.1; k <- 1.; p <- 0.1
Estuary <- function(t, y, parms)  {
 NH3 <- y[1:N]
 NO3 <- y[(N+1):(2*N)]
 O2  <- y[(2*N+1):(3*N)]
 tranNH3<- tran.1D (C = NH3, D = D, v = v,
             C.up = NH3in, C.down = 10,  dx = Grid)$dC
 tranNO3<- tran.1D (C = NO3, D = D, v = v,
             C.up = NO3in,  C.down = 30,  dx = Grid)$dC
 tranO2 <- tran.1D (C = O2 , D = D, v = v,
             C.up = O2in, C.down = 250, dx = Grid)$dC

 reaeration <- p * (O2s - O2)
 r_nit      <- r * O2 / (O2 + k) * NH3

 dNH3    <- tranNH3 - r_nit
 dNO3    <- tranNO3 + r_nit
 dO2     <- tranO2  - 2 * r_nit + reaeration

 list(c(  dNH3, dNO3, dO2 ))
}
print(system.time(
std <- steady.1D(y = runif(3 * N), parms = NULL,
             names=c("NH3", "NO3", "O2"),
             func = Estuary, dimens = N,
             positive = TRUE)
))

  user  system elapsed
  0.11    0.01    0.13

NH3in <- 100
std2 <- steady.1D(y = runif(3 * N), parms = NULL,
             names=c("NH3", "NO3", "O2"),
             func = Estuary, dimens = N,
             positive = TRUE)

plot(std, std2, grid = Grid$x.mid, ylab = "mmol/m3",
     xlab = "m", mfrow = c(1,3), col = "black")
legend("bottomright", lty = 1:2, title = "NH3in",
      legend = c(500, 100))
```

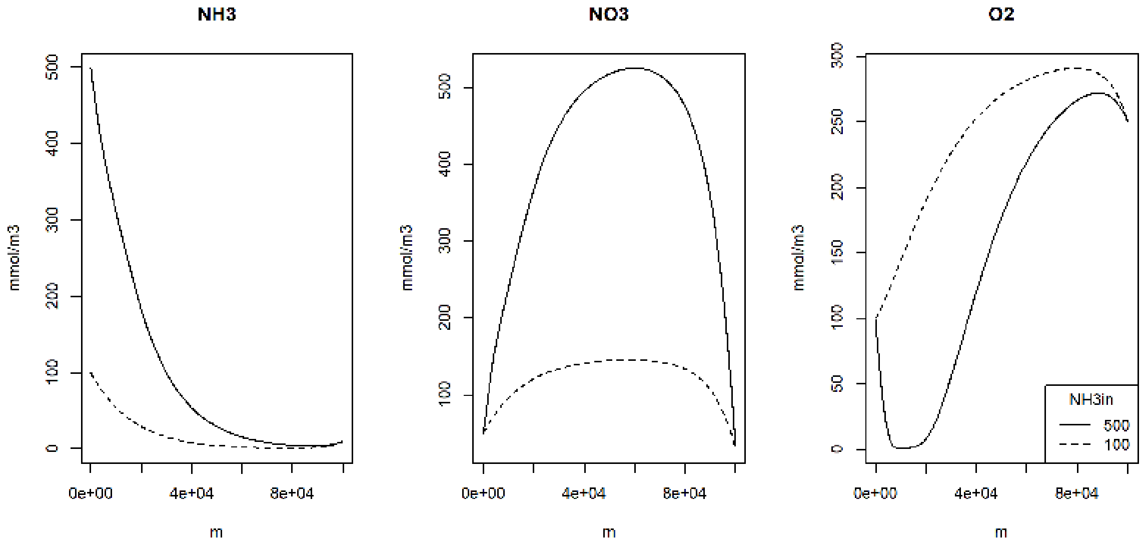

Figure 8: The estuarine problem. See book for explanation.

Affiliation:

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