Solving Differential Equations in R (book) - ODE examples

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

Abstract

This vignette contains the R-examples of chapter 4 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 4. Solving Ordinary Differential Equations in R.

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

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

1. A differential Equation Comprising One Variable

```
r <- 1 ; K <- 10
yini <- c(y = 2)
derivs <- function(t, y, parms)
   list(r * y * (1-y/K))
times <- seq(from = 0, to = 20, by = 0.2)
out   <- ode(y = yini, times = times, func = derivs,
             parms = NULL)
head(out, n = 3)
```

```
     time        y
[1,]  0.0 2.000000
[2,]  0.2 2.339222
[3,]  0.4 2.716436
```

```
#
yini <- c(y = 12)
out2 <- ode(y = yini, times = times, func = derivs,
            parms = NULL)

plot(out, out2, main = "logistic growth", lwd = 2)
```

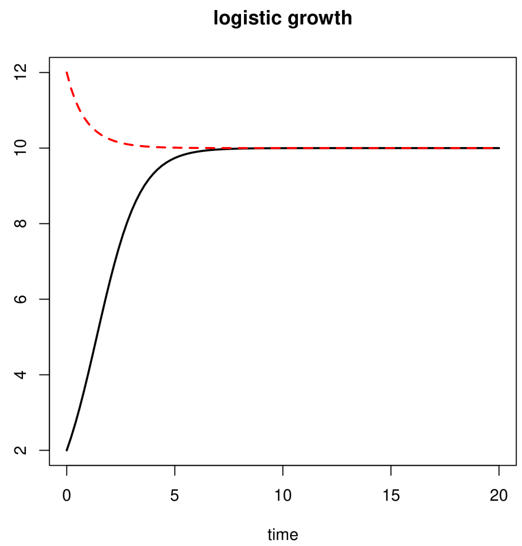

Figure 1: A simple initial value problem, solved twice with different initial conditions. See book for more information.

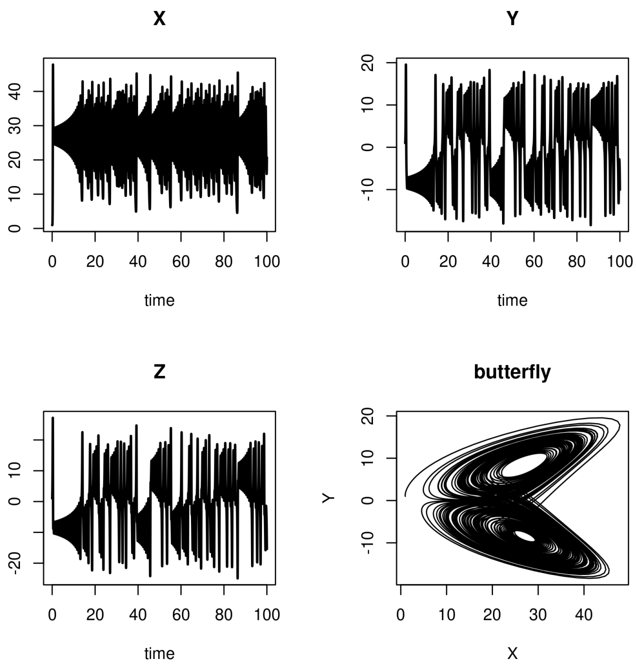

Figure 2: Solution of the lorenz equation. See book for more information.

2. Multiple Variables: the Lorenz Model

```
a    <- -8/3 ; b <- -10; c <- 28
yini <- c(X = 1, Y = 1, Z = 1)
Lorenz <- function (t, y, parms) {
 with(as.list(y), {
    dX <- a * X + Y * Z
    dY <- b * (Y - Z)
    dZ <- -X * Y + c * Y - Z
    list(c(dX, dY, dZ))
 })
}
times <- seq(from = 0, to = 100, by = 0.01)
out   <- ode(y = yini, times = times, func = Lorenz, parms = NULL)

plot(out, lwd = 2)
plot(out[,"X"], out[,"Y"], type = "l", xlab = "X",
  ylab = "Y", main = "butterfly")
```

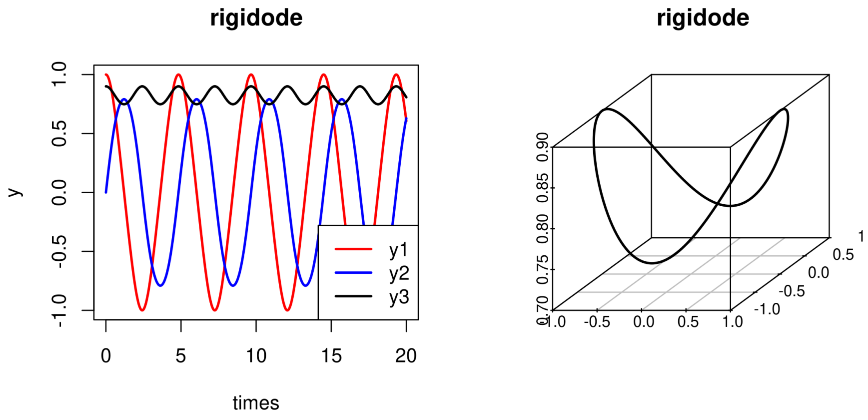

Figure 3: The rigid body equations. See book for more information.

3. Rigid Body Equations

```
yini  <- c(1, 0, 0.9)
rigidode <- function(t, y, parms) {
  dy1 <- -2   * y[2] * y[3]
  dy2 <- 1.25* y[1] * y[3]
  dy3 <- -0.5* y[1] * y[2]
  list(c(dy1, dy2, dy3))
}
times <- seq(from = 0, to = 20, by = 0.01)
out   <- ode (times = times, y = yini, func = rigidode,
              parms = NULL, method = rkMethod("rk45ck"))
head (out, n = 3)
```

```
     time         1          2         3
[1,] 0.00 1.0000000 0.00000000 0.9000000
[2,] 0.01 0.9998988 0.01124950 0.8999719
[3,] 0.02 0.9995951 0.02249603 0.8998875
```

4. Arenstorff Orbits

```
Arenstorff <- function(t, y, p) {
  D1 <- ((y[1] + mu1)^2 + y[2]^2)^(3/2)
  D2 <- ((y[1] - mu2)^2 + y[2]^2)^(3/2)
  dy1 <- y[3]
  dy2 <- y[4]
  dy3 <- y[1] + 2*y[4] - mu2*(y[1]+mu1)/D1 - mu1*(y[1]-mu2)/D2
  dy4 <- y[2] - 2*y[3] - mu2*y[2]/D1 - mu1*y[2]/D2
  return(list( c(dy1, dy2, dy3, dy4) ))
}
mu1   <- 0.012277471
mu2   <- 1 - mu1
yini  <- c(y1 = 0.994, y2 = 0,
          dy1 = 0, dy2 = -2.00158510637908252240537862224)
times <- seq(from = 0, to = 18, by = 0.01)
out <- ode(func = Arenstorff, y = yini, times = times,
           parms = 0, method = "ode45")
yini2 <- c(y1 = 0.994, y2 = 0,
            dy1 = 0, dy2 = -2.0317326295573368357302057924)
out2 <- ode(func = Arenstorff, y = yini2, times = times,
            parms = 0, method = "ode45")
yini3 <- c(y1 = 1.2, y2 = 0,
            dy1 = 0, dy2 = -1.049357510)
out3 <- ode(func = Arenstorff, y = yini3, times = times,
            parms = 0, method = "ode45")

plot(out, out2, out3, which = c("y1", "y2"),
     mfrow = c(2, 2), col = "black", lwd = 2)
plot(out[  ,c("y1", "y2")], type = "l", lwd = 2,
     xlab = "y1", ylab = "y2", main = "solutions 1,2")
lines(out2[ ,c("y1", "y2")], lwd = 2, lty = 2)
plot(out3[ ,c("y1", "y2")], type = "l", lwd = 2, lty = 3,
     xlab = "y1", ylab = "y2", main = "solution 3")
```

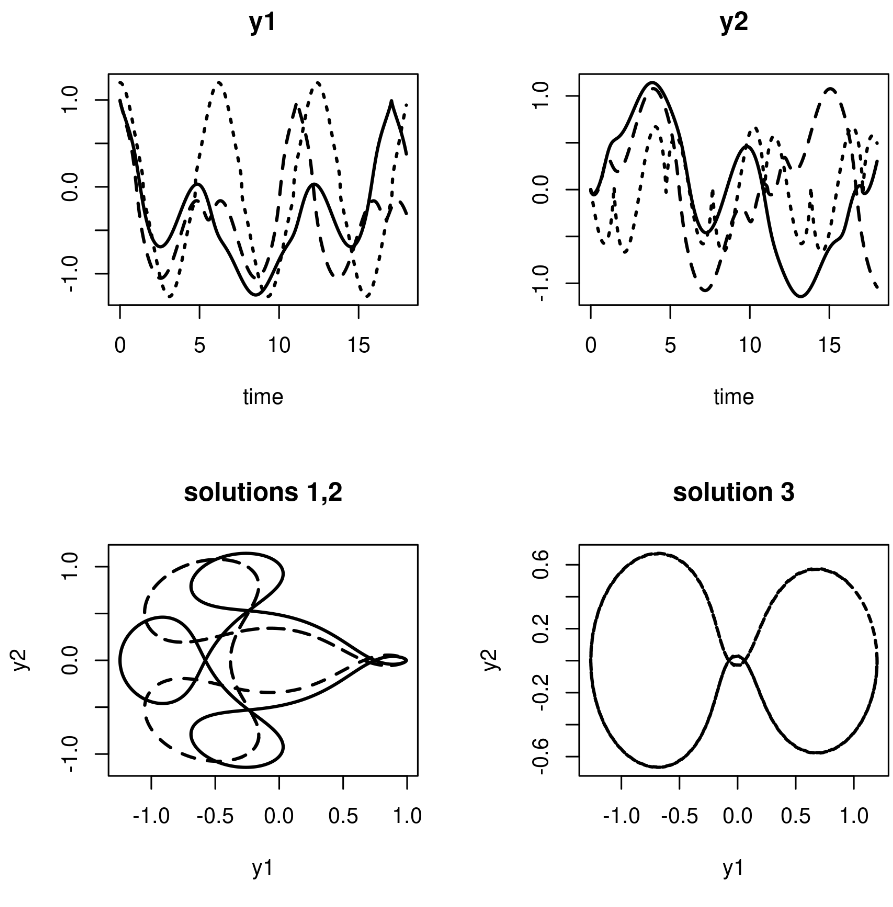

Figure 4: The Arenstorff problem. See book for more information.

5. Seven Moving Stars

```
pleiade <- function (t, Y, pars) {
   x <- Y[1:7]
   y <- Y[8:14]
   u <- Y[15:21]
   v <- Y[22:28]

   distx <- outer(x, x, FUN = function(x, y) x - y)
   disty <- outer(y, y, FUN = function(x, y) x - y)

   rij3 <- (distx^2 + disty^2)^(3/2)

   fx <- starMass * distx / rij3
   fy <- starMass * disty / rij3

   list(c(dx = u,
          dy = v,
          du = colSums(fx, na.rm = TRUE),
          dv = colSums(fy, na.rm = TRUE)))
}
starMass <- 1:7
yini<- c(x1= 3, x2= 3, x3=-1, x4=-3,    x5= 2, x6=-2,   x7= 2,
        y1= 3, y2=-3, y3= 2, y4= 0,     y5= 0, y6=-4,   y7= 4,
        u1= 0, u2= 0, u3= 0, u4= 0,     u5= 0, u6=1.75, u7=-1.5,
        v1= 0, v2= 0, v3= 0, v4=-1.25, v5= 1, v6= 0,   v7= 0)
print(system.time(
out <- ode(func = pleiade, parms = NULL, y = yini,
          method = "adams", times = seq(0, 3, 0.01))))
```

```
  user  system elapsed
 0.104   0.000   0.104
```

```
par(mfrow = c(3, 3))
for (i in 1:7) {
  plot(out[,i+1], out[,i+8], type = "l",
      main = paste("star ",i), xlab = "x", ylab = "y")
  points (yini[i], yini[i+7])
}
plot(out[, 2:8], out[, 9:15], type = "p", cex = 0.5,
     main = "ALL", xlab = "x", ylab = "y")
text(yini[1:7], yini[8:14], 1:7)
matplot(out[,"time"], out[, c("u1", "u7")], type = "l",
    lwd = 2, col = c("black", "grey"), lty = 1,
    xlab = "time", ylab = "velocity", main = "stars 1, 7")
abline(v = c(1.23, 1.68), lty = 2)
```

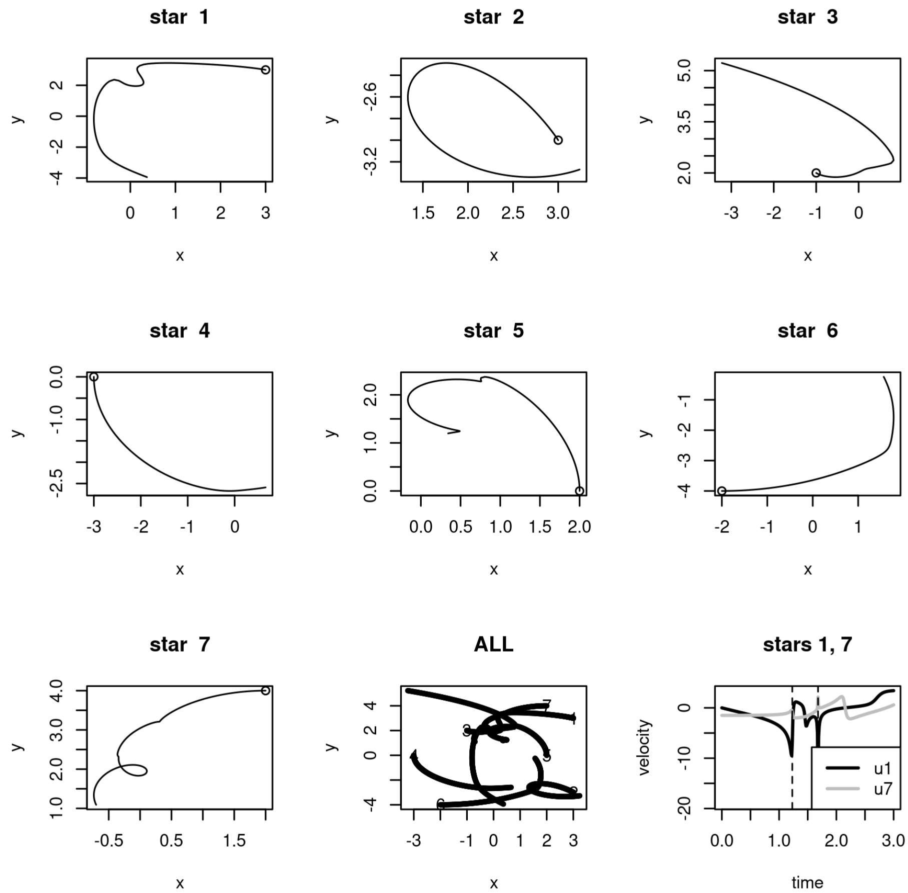

Figure 5: The pleiades problem. See book for more information.

```
legend("bottomright", col = c("black", "grey"), lwd = 2,
       legend = c("u1", "u7"))
```

5.1. A stiff Chemical Example

```
load(file = "light.rda")
head(Light, n = 4)
```

```
        day    irrad
1 0.0000000   0.0000
2 0.3333333   0.0000
3 0.3541667 164.2443
4 0.3750000 204.7486
```

```
irradiance <- approxfun(Light)
irradiance(seq(from = 0, to = 1, by = 0.25))
```

```
[1]   0.0000   0.0000 698.8911 490.4644   0.0000
```

```
k3 <- 1e-11; k2 <- 1e10; k1a <- 1e-30
k1b <- 1; sigma <- 1e11
yini <- c(O = 0, NO = 1.3e8, NO2 = 5e11, O3 = 8e11)
chemistry <- function(t, y, parms) {
  with(as.list(y), {

    radiation <- irradiance(t)
    k1  <- k1a + k1b*radiation

    dO   <-  k1*NO2 - k2*O
    dNO  <-  k1*NO2        - k3*NO*O3 + sigma
    dNO2 <- -k1*NO2        + k3*NO*O3
    dO3  <-           k2*O - k3*NO*O3
    list(c(dO, dNO, dNO2, dO3), radiation = radiation)
  })
}
times <- seq(from = 8/24, to = 5, by = 0.01)
out <- ode(func = chemistry, parms = NULL, y = yini,
          times = times, method = "bdf")

plot(out, type = "l", lwd = 2 )
```

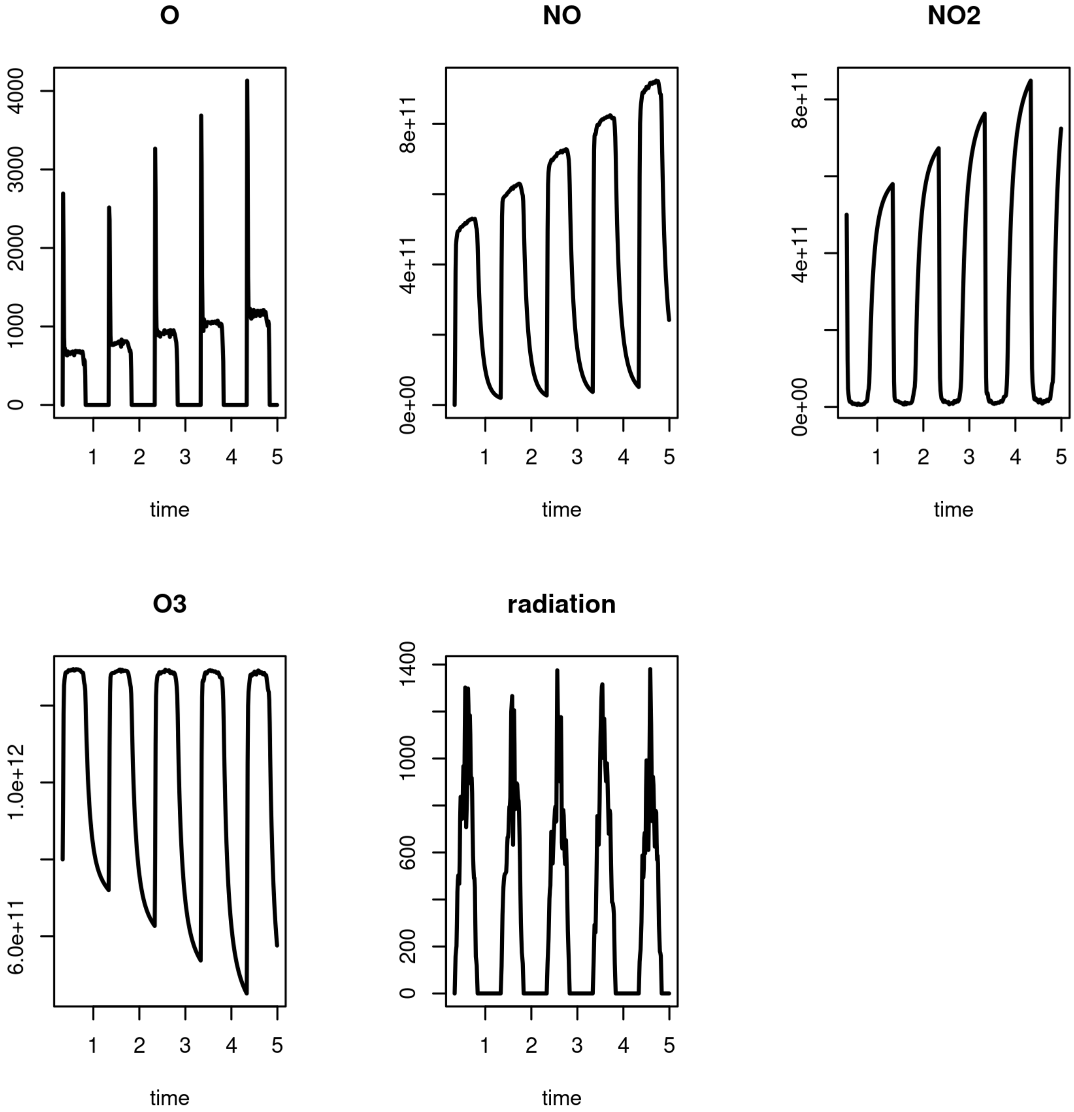

Figure 6: The atmospheric chemistry model. See book for more information.

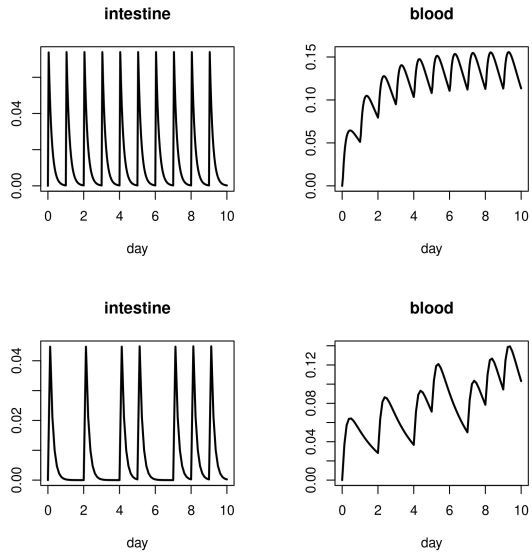

Figure 7: The 2-compartment pharmacokinetic model. See book for more information.

6. Pharmacokinetic Models

6.1. first example

```
a <- 6; b <- 0.6
yini <- c(intestine = 0, blood = 0)
pharmacokinetics <- function(t, y, p) {
  if ( (24*t) %% 24 <= 1)
    uptake <- 2
  else
    uptake <- 0
  dy1 <- - a* y[1] + uptake
  dy2 <-  a* y[1] - b *y[2]
  list(c(dy1, dy2))
}
times <- seq(from = 0, to = 10, by = 1/24)
out <- ode(func = pharmacokinetics, times = times,
           y = yini, parms = NULL)
times <- seq(0, 10,  by = 3/24)
out2 <- ode(func = pharmacokinetics, times = times,
           y = yini, parms = NULL, method = "impAdams")
```

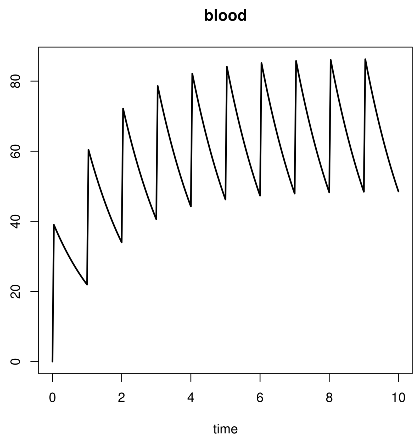

Figure 8: The 1-compartment pharmacokinetic model. See book for more information.

6.2. second example

```
b    <- 0.6
yini <- c(blood = 0)
pharmaco2 <- function(t, blood, p) {
  dblood <-  - b * blood
  list(dblood)
}
injectevents <- data.frame(var = "blood",
                          time =  0:20,
                          value = 40,
                          method = "add")
head(injectevents, n=3)
```

```
    var time value method
1 blood    0    40    add
2 blood    1    40    add
3 blood    2    40    add
```

```
times <- seq(from = 0, to = 10, by = 1/24)
out2 <- ode(func = pharmaco2, times = times, y = yini,
           parms = NULL, method = "impAdams",
           events = list(data = injectevents))

plot(out2, lwd = 2)
```

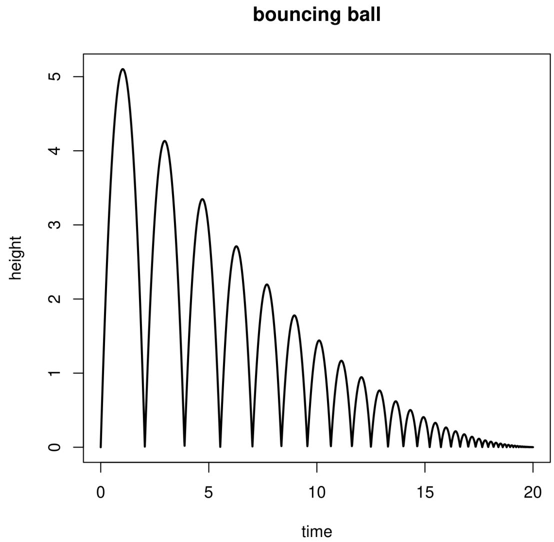

Figure 9: The bouncing ball model. See book for more information.

7. A Bouncing Ball

```
yini  <- c(height = 0, velocity = 10)
ball <- function(t, y, parms) {
  dy1 <- y[2]
  dy2 <- -9.8

  list(c(dy1, dy2))
}
rootfunc <- function(t, y, parms) y[1]
eventfunc <- function(t, y, parms) {
 y[1] <- 0
 y[2] <- -0.9*y[2]
 return(y)
}
times <- seq(from = 0, to = 20, by = 0.01)
out <- ode(times = times, y = yini, func = ball,
             parms = NULL, rootfun = rootfunc,
             events = list(func = eventfunc, root = TRUE))

plot(out, which = "height", lwd = 2,
    main = "bouncing ball", ylab = "height")
```

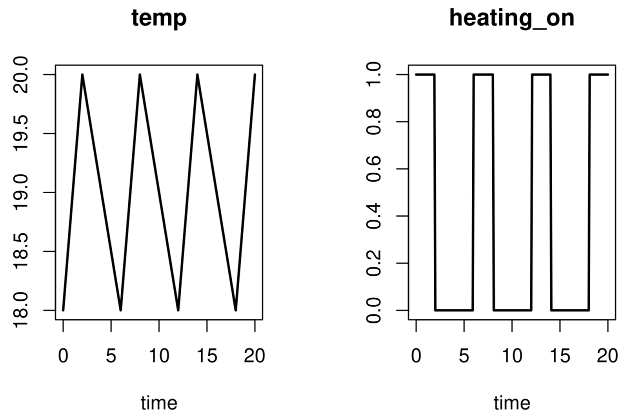

Figure 10: The temperature model. See book for more information.

8. Temperature in a Climate-controlled Room

```
yini  <- c(temp = 18, heating_on = 1)
temp <- function(t, y, parms) {
  dy1 <- ifelse(y[2] == 1, 1.0, -0.5)
  dy2 <- 0
  list(c(dy1, dy2))
}
rootfunc <- function(t, y, parms) c(y[1]-18, y[1]-20)
eventfunc <- function(t, y, parms) {
 y[1] <- y[1]
 y[2] <- ! y[2]
 return(y)
}
times <- seq(from = 0, to = 20, by = 0.1)
out <- lsode(times = times, y = yini, func = temp,
            parms = NULL, rootfun = rootfunc,
            events = list(func = eventfunc, root = TRUE))

attributes(out)$troot

[1]  2  6  6  6  8 12 14 18 18 18

plot(out, lwd = 2)
```

9. Method Selection

```
yini  <- c(y = 2, dy = 0)
Vdpol <- function(t, y, mu)
   list(c(y[2],
          mu * (1 - y[1]^2) * y[2] - y[1]))
times <- seq(from = 0, to = 30, by = 0.01)
nonstiff <- ode(func = Vdpol, parms = 1,    y = yini,
                times = times)
diagnostics(nonstiff)
```

```
--------------------
lsoda return code
--------------------

  return code (idid) =  2
  Integration was successful.

--------------------
INTEGER values
--------------------

  1 The return code : 2
  2 The number of steps taken for the problem so far: 3004
  3 The number of function evaluations for the problem so far: 6009
  5 The method order last used (successfully): 7
  6 The order of the method to be attempted on the next step: 7
  7 If return flag =-4,-5: the largest component in error vector 0
  8 The length of the real work array actually required: 52
  9 The length of the integer work array actually required: 22
 14 The number of Jacobian evaluations and LU decompositions so far: 0
 15 The method indicator for the last succesful step,
          1=adams (nonstiff), 2= bdf (stiff): 1
 16 The current method indicator to be attempted on the next step,
          1=adams (nonstiff), 2= bdf (stiff): 1

--------------------
RSTATE values
--------------------

  1 The step size in t last used (successfully): 0.01
  2 The step size to be attempted on the next step: 0.01
  3 The current value of the independent variable which the solver has reached: 30.00947
  4 Tolerance scale factor > 1.0 computed when requesting too much accuracy: 0
  5 The value of t at the time of the last method switch, if any: 0
```

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