```r
# differential evolution
# enrico schumann, version 2010-03-27
# ------------------------------------------------------------------



# ------------------------------------------------------------------
# optimisation function
# ------------------------------------------------------------------

DE <- function(de,dataList,OF)
{
  # auxiliary functions
  # ----------------------------------------------------------------
  # random numbers: like rand(m,n)/randn(m,n) in Matlab
  mRU <- function(m,n){
    return(array(runif(m*n), dim = c(m,n)))
  }
  mRN <- function(m,n){
    return(array(rnorm(m*n), dim = c(m,n)))
  }
  shift <- function(x)
  {
    rr <- length(x)
    return(c(x[rr],x[1:(rr-1)]))
  }
  # penalty
  pen <- function(mP,pso,vF)
  {
    minV <- pso$min
    maxV <- pso$max
    ww <- pso$ww

    # max constraint: if larger than maxV, element in A is positiv
    A <- mP - as.vector(maxV)
    A <- A + abs(A)

    # max constraint: if smaller than minV, element in B is positiv
    B <- as.vector(minV) - mP
    B <- B + abs(B)

    # beta 1 + beta2 > 0
    C <- ww*((mP[1,]+mP[2,])-abs(mP[1,]+mP[2,]))
    A <- ww * colSums(A + B)*vF - C
    return(A)
  }

  # main algorithm
  # ----------------------------------------------------------------
  # set up initial population
  mP <- de$min + diag(de$max - de$min) %*% mRU(de$d,de$nP)
  # include extremes
  mP[,1:de$d] <- diag(de$max)
  mP[,(de$d+1):(2*de$d)] <- diag(de$min)

  # evaluate initial population
  vF <- apply(mP,2,OF,data = dataList)

  # constraints
  vP <- pen(mP,de,vF)
  vF <- vF + vP

  # keep track of OF
  Fmat <- array(NaN,c(de$nG,de$nP))

  for (g in 1:de$nG){

    # update population
    vI <- sample(1:de$nP,de$nP)
```

```
    R1 <- shift(vI)
    R2 <- shift(R1)
    R3 <- shift(R2)

    # prelim. update
    mPv = mP[,R1] + de$F * (mP[,R2] - mP[,R3])
    if(de$R > 0){mPv <- mPv + de$R * mRN(de$d,de$nP)}

    mI <- mRU(de$d,de$nP) > de$CR
    mPv[mI] <- mP[mI]

    # evaluate updated population
    vFv <- apply(mPv,2,OF,data = dataList)
    # constraints
    vPv <- pen(mPv,de,vF)
    vFv <- vFv + vPv
    vFv[!(is.finite(vFv))] <- 1000000


    # find improvements
    logik <- vFv < vF
    mP[,logik] <- mPv[,logik]
    vF[logik] <- vFv[logik]
    Fmat[g,] <- vF

  } # g in 1:nG
  sGbest <- min(vF)
  sgbest <- which.min(vF)[1]

  # return best solution
  return(list(beta = mP[,sgbest], OFvalue = sGbest, popF = vF, Fmat = Fmat))
}



# ---------------------------------------------------------------------
# define functions
# ---------------------------------------------------------------------

# nelson--siegel
NS <- function(betaV,mats)
{
  # betaV = beta1-3, lambda1
  gam <- mats / betaV[4]
  y = betaV[1] + betaV[2] * ((1 - exp(-gam)) / (gam)) + betaV[3] * (((1 - exp(-
gam)) / (gam)) - exp(-gam))
  return(y)
}
# nelson--siegel--svensson 1
NSS <- function(betaV,mats)
{
  # betaV = beta1-4, lambda1-2
  gam1 <- mats / betaV[5]
  gam2 <- mats / betaV[6]
  y <- betaV[1] + betaV[2] * ((1 - exp(-gam1)) / (gam1)) +
    betaV[3] * (((1 - exp(-gam1)) / (gam1)) - exp(-gam1)) +
    betaV[4] * (((1 - exp(-gam2)) / (gam2)) - exp(-gam2))
  return(y)
}
# nelson--siegel--svensson 2 (a bit faster)
NSS2 <- function(betaV,mats)
{
  # betaV = beta1-4, lambda1-2
  gam1 <- mats / betaV[5]
  gam2 <- mats / betaV[6]
  aux1 <- 1 - exp(-gam1)
  aux2 <- 1 - exp(-gam2)
  y     <- betaV[1] + betaV[2] * (aux1 / gam1) +
    betaV[3] * (aux1 / gam1 + aux1 - 1) +
```

```r
    betaV[4] * (aux2 / gam2 + aux2 - 1)
  return(y)
}
# generic objective function
OF <- function(betaV,data)
{
  mats          <- data$mats
  yM            <- data$yM
  model         <- data$model
  y             <- model(betaV,mats)
  aux           <- y - yM
  aux           <- crossprod(aux)
  return(aux)
}

# ------------------------------------------------------------------
# example 2: NSS
# ------------------------------------------------------------------
# source('de.r')
# set up yield curve (in this example: artificial data), and plot it
mats <- 1:1:40
betaTRUE <- c(5,-2,5,-5,1,3); yM <- NSS2(betaTRUE,mats)
plot(mats,yM,xlab="maturities in years",ylab="yields in %")

# collect all in dataList
dataList <- list(yM = yM, mats = mats, model = NSS2)

dlist <- function(yM){
  return(list(yM = yM, mats = 1:1:40, model = NSS2))}

df_densite = read.csv('df_densite.csv')[,-1]

# set parameters for de
de <- list(
  min   = c( 0,-15,-30,-30,0  ,2.5),
  max   = c(15, 30, 30, 30,2.5,5  ),
  d     = 6,
  nP    = 200,
  nG    = 600,
  ww    = 0.1,
  F     = 0.50,
  CR    = 0.99,
  R     = 0.00 # random term (added to change)
)

# import des données
df =  read.csv('df_densite.csv')[,-1]
# creation de la list de datalist
df_den = apply(df,2,dlist)
# fonction qui retourne les beta
solsf <- function(de,dataList,OF){
  return(DE(de = de,dataList = dataList,OF = OF)$beta)}

system.time(sol <- solsf(de = de,dataList = dataList,OF = OF))
# application de nss sur les densités
system.time(sols <- lapply(df_den,solsf,de = de, OF=OF))
# maximum error
max(abs(dataList$model(sol$beta,mats)-dataList$model(betaTRUE,mats)))
# value of objective function
sqrt(sol$OFvalue)
lines(mats,dataList$model(sol$beta,mats), col="blue")

legend(x  = "bottom", legend = c("true yields","DE"),
       col = c("black","blue"),
       pch = c(1,NA), lty=c(0,1))

sols = sol$beta
# > sols
# [1]  5 -2  5 -5  1  3
```

```r
library(ggplot2)

# ggplot
df = data.frame(mats = mats,ym = yM+0.025,model=dataList$model(sols,mats))
p = ggplot(data = df, aes(mats,ym,color='données')) + geom_point() + ggtitle
("Série chronologique normaliée ") + ylab("Valeurs en %") + xlab("Maturités")
#p = p + geom_line(data = df, aes(x=mats,y=model,color='NSS'))
p

# Create a grid to plot the different values of "a"
#par(mfrow=c(3,2))

df = data.frame(mats = mats,ym = yM+0.025,model=dataList$model(c
(6,-2,5,-5,1,3),mats))
# p1 = ggplot(data = df, aes(mats,ym)) + geom_point() + ggtitle("Original vs
variation de beta 1") + ylab("Valeurs en %") + xlab("Maturités")
p1 = p + geom_line(data = df, aes(x=mats,y=model,color='beta1'))+ ggtitle("Effet
de la variation de beta 1") + ylab("Valeurs en %") + xlab("Maturités")
p1


df = data.frame(mats = mats,ym = yM+0.025,model=dataList$model(c
(5,-1,5,-5,1,3),mats))
#p2 = ggplot(data = df, aes(mats,ym)) + geom_point() + ggtitle("Graphique d'une
courbe 2") + ylab("Valeurs en %") + xlab("Maturités")
p2 = p + geom_line(data = df, aes(x=mats,y=model,color='beta2')) + ggtitle("Effet
de la variation de beta 2") + ylab("Valeurs en %") + xlab("Maturités")
p2

df = data.frame(mats = mats,ym = yM+0.025,model=dataList$model(c
(5,-2,6,-5,1,3),mats))
#p3 = ggplot(data = df, aes(mats,ym)) + geom_point() + ggtitle("Graphique d'une
courbe 3") + ylab("Valeurs en %") + xlab("Maturités")
p3 = p + geom_line(data = df, aes(x=mats,y=model,color='beta3')) + ggtitle("Effet
de la variation de beta 3") + ylab("Valeurs en %") + xlab("Maturités")
p3

df = data.frame(mats = mats,ym = yM+0.025,model=dataList$model(c
(5,-2,5,-4,1,3),mats))
#p4 = ggplot(data = df, aes(mats,ym)) + geom_point() + ggtitle("Graphique d'une
courbe 4") + ylab("Valeurs en %") + xlab("Maturités")
p4 = p + geom_line(data = df, aes(x=mats,y=model,color='beta4')) + ggtitle("Effet
de la variation de beta 4") + ylab("Valeurs en %") + xlab("Maturités")
p4

df = data.frame(mats = mats,ym = yM+0.025,model=dataList$model(c
(5,-2,5,-5,2,3),mats))
#p5 = ggplot(data = df, aes(mats,ym)) + geom_point() + ggtitle("Graphique d'une
courbe 5") + ylab("Valeurs en %") + xlab("Maturités")
p5 = p + geom_line(data = df, aes(x=mats,y=model,color='beta5')) + ggtitle("Effet
de la variation de beta 5") + ylab("Valeurs en %") + xlab("Maturités")
p5

df = data.frame(mats = mats,ym = yM+0.025,model=dataList$model(c
(5,-2,5,-5,1,4),mats))
#p6 = ggplot(data = df, aes(mats,ym)) + geom_point() + ggtitle("Graphique d'une
courbe 6") + ylab("Valeurs en %") + xlab("Maturités")
p6 = p + geom_line(data = df, aes(x=mats,y=model,color='beta6')) + ggtitle("Effet
de la variation de beta 6") + ylab("Valeurs en %") + xlab("Maturités")
p6

z = multiplot(p1, p2, p3, p4, p5, p6, cols=2)


# Multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
```

```
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)

  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}
```