Fitting the Nelson–Siegel–Svensson model with Differential Evolution

Enrico Schumann

December 8, 2011

Abstract

A brief tutorial on how to use Differential Evolution (DE) to fit the Nelson–Siegel model.

1 Introduction

In this tutorial we look into fitting the Nelson–Siegel–Svensson (NSS) model to data. The purpose of this vignette is to provide the code in a convenient way; for more details, please see the book [Gilli et~al., 2011]. Further information can be found in Gilli et~al. [2010] and Gilli and Schumann [2010].

We start by attaching the package; the variable `nRuns` sets the number of restarts for the examples to come. It is set to only two here to keep the build-time of the package acceptable; feel free to increase it. We set a seed to make the computations reproducible.

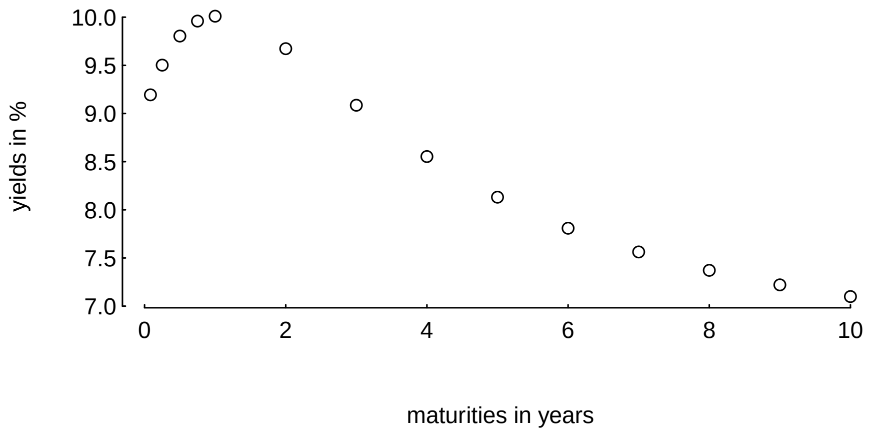
```
> require("NMOF")
> nRuns <- 2L
> set.seed(112233)
```

2 Fitting the NS model to given zero rates

The NS model

and creating a 'true' yield curve `yM` with given parameters `betaTRUE`. The times-to-payment, measured in years, are collected in the vector `tm`.

```
> tm <- c(c(1,3,6,9)/12,1:10)
> betaTRUE <- c(6, 3, 8, 1)
> yM <- NS(betaTRUE, tm)
> par(ps = 11, bty = "n", las = 1, tck = 0.01,
+     mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years", ylab = "yields in %")
```



The aim is to fit a smooth curve through these points. Since we have used the model to create the points, we should be able to obtain a perfect fit. We start with the objective function OF. It takes two arguments: param, which is a candidate solution (a numeric vector), and the list data, which holds all other variables.

```
> OF <- function(param, data) {
+     y <- data$model(param, data$tm)
+     aux <- y - data$yM
+     aux <- max(abs(aux))
+     if (is.na(aux)) aux <- 1e10
+     aux
+ }
```

We have a added a crude but effective safeguard against ‘strange’ parameter values that lead to NA values: the objective function returns a large positive value. We minimise, and hence parameters that produce NA values are marked as bad.

In this first example, we set up data as follows:

```
> data <- list(yM = yM,
+              tm = tm,
+           model = NS,
+              ww = 0.1,
+             min = c( 0,-15,-30, 0),
+             max = c(15, 30, 30,10))
```

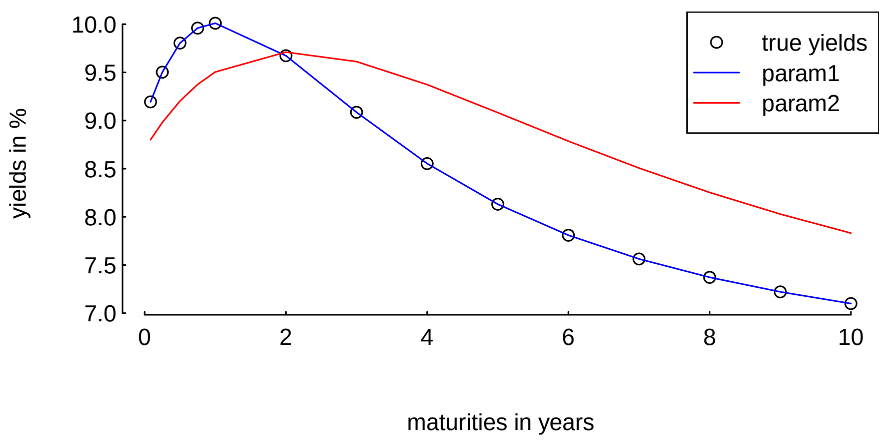
We add a model (a function; in this case NS) that describes the mapping from parameters to a yield curve, and vectors min and max that we will later use as constraints. ww is a penalty weight, explained below.

OF will take a candidate solution param, transform this solution via data$model into yields, and compare these yields with yM, which here means to compute the maximum absolute difference.

```
> param1 <- betaTRUE
> OF(param1, data)

[1] 0

> param2 <- c(5.7, 3, 8, 2)
> OF(param2, data)

[1] 0.9768586
```

We can also compare the solutions in terms of yield curves.

```
> par(ps = 11, bty = "n", las = 1, tck = 0.01, mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years", ylab = "yields in %")
> lines(tm, NS(param1, tm), col = "blue")
> lines(tm, NS(param2, tm), col = "red")
> legend(x = "topright",
+        legend = c("true yields", "param1", "param2"),
+        col = c("black", "blue", "red"),
+        pch = c(1, NA, NA), lty = c(0, 1, 1))
```



We generally want to obtain parameters such that certain constraints are met. We include these through a penalty function.

```
> penalty <- function(mP, data) {
+     minV <- data$min
+     maxV <- data$max
+     ww <- data$ww
+     # if larger than maxV, element in A is positiv
+     A <- mP - as.vector(maxV); A <- A + abs(A)
+     # 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) - C
+     A
+ }
```

We already have `data`, so let us see what the function does to solutions that violate a constraint. Suppose we have a population `mP` of three solutions (the `m` in `mP` is to remind us that we deal with a matrix).

```
> param1 <- c( 6,3,8,-1)
> param2 <- c( 6,3,8, 1)
> param3 <- c(-1,3,8, 1)
> mP <- cbind(param1,param2,param3)
> rownames(mP) <- c("b1","b2","b3","lambda")
> mP
```

```
       param1 param2 param3
b1          6      6     -1
b2          3      3      3
b3          8      8      8
lambda     -1      1      1
```

The first and the third solution violate the constraints: in the first solution, λ is negative; in the third solution, β_1 is negative.

```
> penalty(mP,data)
```

```
param1 param2 param3
   0.2    0.0    0.2
```

The parameter ww controls how heavily we penalise.

```
> data$ww <- 0.5
> penalty(mP,data)
```

```
param1 param2 param3
     1      0      1
```

For valid solutions, the penalty should be zero.

```
> param1 <- c( 6,3,8, 1)
> param2 <- c( 6,3,8, 1)
> param3 <- c( 2,3,8, 1)
> mP <- cbind(param1,param2,param3)
> rownames(mP) <- c("b1","b2","b3","lambda")
> penalty(mP, data)
```

```
param1 param2 param3
     0      0      0
```

Note that penalty works on the complete population at once; there is no need to loop over the solutions.

So we can run a test. We start by defining the parameters of DE. Note in particular that we pass the penalty function, and that we set loopPen to FALSE.

```
> algo <- list(nP = 100L,
+              nG = 500L,
+               F = 0.50,
+              CR = 0.99,
+             min = c( 0,-15,-30, 0),
+             max = c(15, 30, 30,10),
+             pen = penalty,
+          repair = NULL,
+          loopOF = TRUE,
+         loopPen = FALSE,
+      loopRepair = TRUE,
+        printBar = FALSE)
```

DEopt is then called with the objective function OF, the list data, and the list algo.

```
> sol <- DEopt(OF = OF, algo = algo, data = data)
```

```
Differential Evolution.

Standard deviation of OF in final population is 4.239866e-16.
```

Just to check whether the objective function works properly, we compare the maximum error with the returned objective function value – they should be the same.

```
> max( abs(data$model(sol$xbest,tm) - data$model(betaTRUE,tm)) )

[1] 0

> sol$OFvalue

[1] 0
```

As a benchmark, we run the function `nlminb` from the stats package. This is not a fair test: `nlminb` is not appropriate for such problems. (But then, if we found that it performs better than DE, we would have a strong indication that something is wrong with our implementation of DE.) We use a random starting value `s0`.

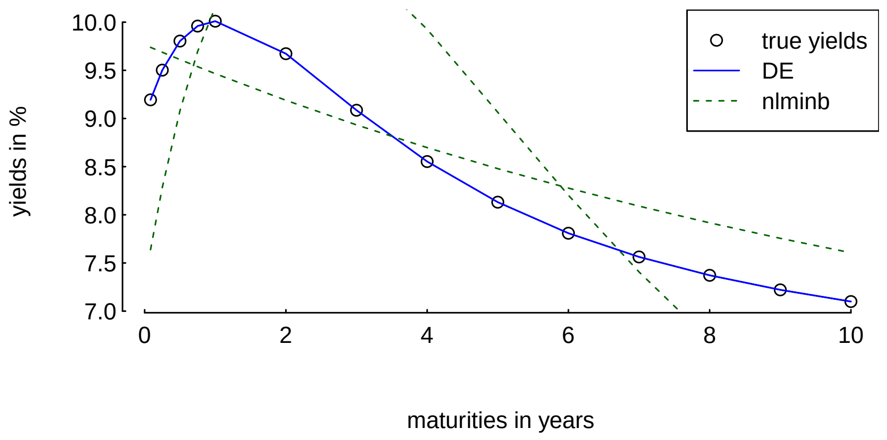
```
> s0 <- algo$min + (algo$max-algo$min) * runif(length(algo$min))
> sol2 <- nlminb(s0, OF, data = data,
+                            lower = data$min,
+                            upper = data$max,
+                            control = list(eval.max = 50000L,
+                                           iter.max = 50000L))
```

Again, we compare the returned objective function value and the maximum error.

```
> max(abs(data$model(sol2$par,tm)-data$model(betaTRUE,tm)))

[1] 0.6107039

> sol2$objective

[1] 0.6107039
```

To compare our two solutions (DE and `nlminb`), we can plot them together with the true yields curve. But it is important to stress that the results of both algorithms are stochastic: in the case of DE because it deliberately uses randomness; in the case of `nlminb` because we set the starting value randomly. To get more meaningful results we should run both algorithms several times.

```
> par(ps = 11, bty = "n", las = 1, tck = 0.01, mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years",
+              ylab = "yields in %")
> algo$printDetail <- FALSE
> for (i in seq_len(nRuns)) {
+     sol <- DEopt(OF = OF, algo = algo, data = data)
+     lines(tm, data$model(sol$xbest,tm), col = "blue")
+     s0 <- algo$min + (algo$max-algo$min) * runif(length(algo$min))
+     sol2 <- nlminb(s0, OF, data = data,
+                            lower = data$min,
+                            upper = data$max,
+                            control = list(eval.max = 50000L,
+                                           iter.max = 50000L))
+
+     lines(tm,data$model(sol2$par,tm), col = "darkgreen", lty = 2)
+ }
> legend(x = "topright", legend = c("true yields", "DE", "nlminb"),
+        col = c("black","blue","darkgreen"),
+        pch = c(1, NA, NA), lty = c(0, 1, 2))
```

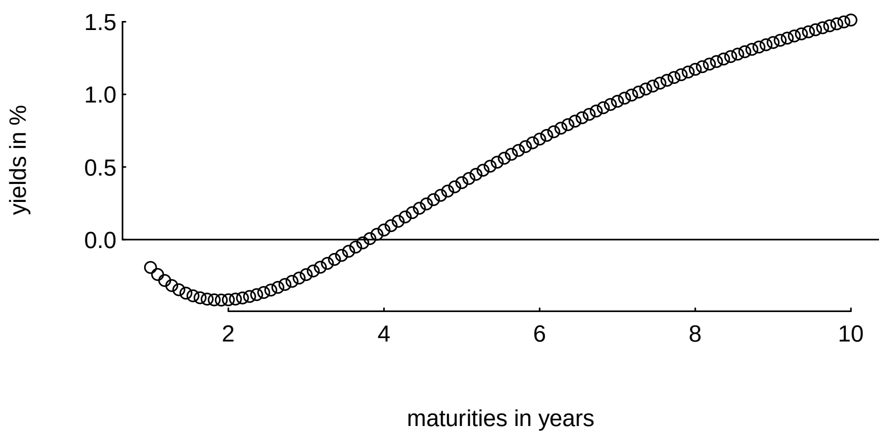


It is no error that there appears to be only one curve for DE: there are, in fact, `nRuns` lines, but they are printed on top of each other.

Other constraints

The parameter constraints on the NS (and NSS) model are to make sure that the resulting zero rates are nonnegative. But in fact, they do not guarantee positive rates.

```
> tm <- seq(1, 10, length.out = 100)    ## 1 to 10 years
> betaTRUE <- c(3, -2, -8, 1.5)         ## 'true' parameters
> yM <- NS(betaTRUE, tm)
> par(ps = 11, bty = "n", las = 1, tck = 0.01, mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years", ylab = "yields in %")
> abline(h = 0)
```



This is really a made-up example, but nevertheless we may want to include safeguards against such parameter vectors: we could include just one constraint that all rates are greater than zero. This can be done, again, with a penalty function.

```
> penalty2 <- function(param,data) {
+     y <- data$model(param,data$tm)
+     aux <- abs(y - abs(y))
+     sum(aux) * data$ww
+ }
```

Check:

```
> penalty2(c(3, -2, -8, 1.5),data)
```

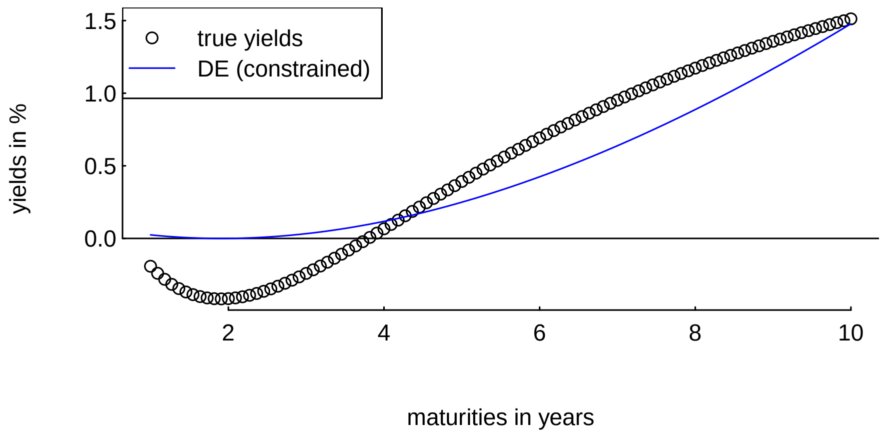
```
[1] 0.8634335
```

This penalty function only works for a single solution, so it is actually simplest to write it directly into the objective function.

```
> OFa <- function(param,data) {
+     y <- data$model(param,data$tm)
+     aux <- y - data$yM
+     res <- max(abs(aux))
+     # compute the penalty
+     aux <- y - abs(y) # aux == zero for nonnegative y
+     aux <- -sum(aux) * data$ww
+     res <- res + aux
+     if (is.na(res)) res <- 1e10
+     res
+ }
```

So just as a numerical test: suppose the above parameters were true, and interest rates were negative.

```
> algo$pen <- NULL; data$yM <- yM; data$tm <- tm
> par(ps = 11, bty = "n", las = 1, tck = 0.01, mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years", ylab = "yields in %")
> abline(h = 0)
> sol <- DEopt(OF = OFa, algo = algo, data = data)
> lines(tm,data$model(sol$xbest,tm), col = "blue")
> legend(x = "topleft", legend = c("true yields", "DE (constrained)"),
+        col = c("black", "blue"),
+        pch = c(1, NA, NA), lty = c(0, 1, 2))
```



3 Fitting the NSS model to given zero rates

There is little that we need to change if we want to use the NSS model instead. We just have to pass a different `model` to the objective function (and change the `min`/`max`-vectors). An example follows. Again, we fix true parameters and try to recover them.

```
> tm <- c(c(1,3,6,9)/12, 1:10)
> betaTRUE <- c(5,-2,5,-5,1,6)
> yM <- NSS(betaTRUE, tm)
```

The lists data and algo are almost the same as before; the objective function stays exactly the same.

```
> data <- list(yM = yM,
+              tm = tm,
+           model = NSS,
+             min = c( 0,-15,-30,-30,  0,5),
+             max = c(15, 30, 30, 30,  5,  10),
+              ww = 1)
> algo <- list(nP = 100L,
+              nG = 500L,
+               F = 0.50,
+              CR = 0.99,
+             min = c( 0,-15,-30,-30,  0,5),
+             max = c(15, 30, 30, 30,  5,  10),
+             pen = penalty,
+          repair = NULL,
+          loopOF = TRUE,
+         loopPen = FALSE,
+      loopRepair = TRUE,
+        printBar = FALSE,
+     printDetail = FALSE)
```

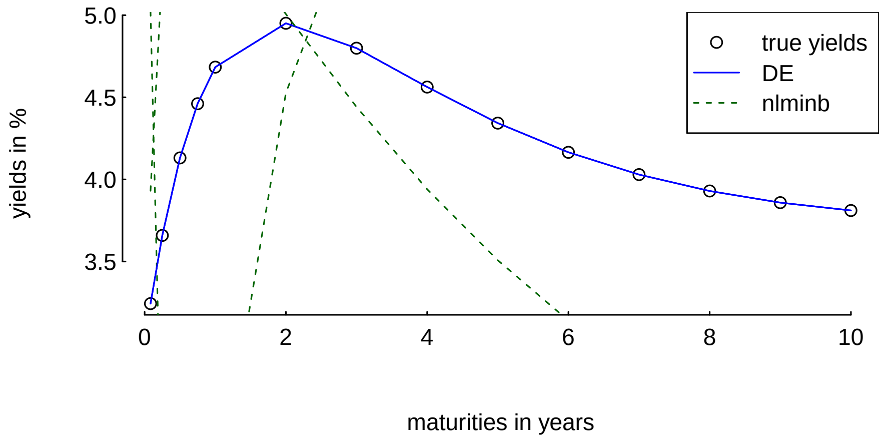
It remains to run the algorithm. We compare the results with nlminb. (Again, we check the returned objective function value.)

```
> sol <- DEopt(OF = OF, algo = algo, data = data)
> max(abs(data$model(sol$xbest,tm) - data$model(betaTRUE,tm)))

[1] 7.549517e-15

> sol$OFvalue

[1] 7.549517e-15

> s0 <- algo$min + (algo$max - algo$min) * runif(length(algo$min))
> sol2 <- nlminb(s0,OF,data = data,
+                          lower = data$min,
+                          upper = data$max,
+                        control = list(eval.max = 50000L,
+                                       iter.max = 50000L))
> max(abs(data$model(sol2$par,tm) - data$model(betaTRUE,tm)))

[1] 0.5279637

> sol2$objective

[1] 0.5279637
```

Finally, we compare the yield curves resulting from three runs. (Again, the low number of restarts is chosen only to keep the time to build the vignette reasonable. A better number would 100.)

```
> par(ps = 11, bty = "n", las = 1, tck = 0.01, mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years", ylab = "yields in %")
> for (i in seq_len(nRuns)) {
```

```
+     sol <- DEopt(OF = OF, algo = algo, data = data)
+     lines(tm, data$model(sol$xbest,tm), col = "blue")
+     s0 <- algo$min + (algo$max - algo$min) * runif(length(algo$min))
+     sol2 <- nlminb(s0, OF, data = data,
+                           lower = data$min,
+                           upper = data$max,
+                           control = list(eval.max = 50000L,
+                                          iter.max = 50000L))
+
+     lines(tm, data$model(sol2$par,tm), col = "darkgreen", lty = 2)
+ }
> legend(x = "topright", legend = c("true yields", "DE", "nlminb"),
+        col = c("black","blue","darkgreen"),
+        pch = c(1,NA,NA), lty = c(0,1,2), bg = "white")
```



4 Fitting the NSS model to given bond prices

A bond is a list of payment dates (given a valuation date, we can translate them into times-to-payment) and associated payments. Suppose we are given the following set of bonds.

```
> cf1 <- c(rep(5.75,  8), 105.75); tm1 <- 0:8 + 0.5
> cf2 <- c(rep(4.25, 17), 104.25); tm2 <- 1:18
> cf3 <- c(3.5, 103.5); tm3 <- 0:1 + 0.5
> cf4 <- c(rep(3.00, 15), 103.00); tm4 <- 1:16
> cf5 <- c(rep(3.25, 11), 103.25); tm5 <- 0:11 + 0.5
> cf6 <- c(rep(5.75, 17), 105.75); tm6 <- 0:17 + 0.5
> cf7 <- c(rep(3.50, 14), 103.50); tm7 <- 1:15
> cf8 <- c(rep(5.00,  8), 105.00); tm8 <- 0:8 + 0.5
> cf9 <- 105; tm9 <- 1
> cf10 <- c(rep(3.00, 12), 103.00); tm10 <- 0:12 + 0.5
> cf11 <- c(rep(2.50,  7), 102.50); tm11 <- 1:8
> cf12 <- c(rep(4.00, 10), 104.00); tm12 <- 1:11
> cf13 <- c(rep(3.75, 18), 103.75); tm13 <- 0:18 + 0.5
> cf14 <- c(rep(4.00, 17), 104.00); tm14 <- 1:18
> cf15 <- c(rep(2.25,  8), 102.25); tm15 <- 0:8 + 0.5
> cf16 <- c(rep(4.00,  6), 104.00); tm16 <- 1:7
> cf17 <- c(rep(2.25, 12), 102.25); tm17 <- 1:13
> cf18 <- c(rep(4.50, 19), 104.50); tm18 <- 0:19 + 0.5
> cf19 <- c(rep(2.25,  7), 102.25); tm19 <- 1:8
> cf20 <- c(rep(3.00, 14), 103.00); tm20 <- 1:15
```

We put all cash flows into a matrix `cfMatrix`, such that one bond is one column, and one row corresponds to one payment date.

```
> cfList <- list(cf1,cf2,cf3,cf4,cf5,cf6,cf7,cf8,cf9,cf10,cf11,cf12,cf13,cf14,cf15,cf16,cf17
> tmList <- list(tm1,tm2,tm3,tm4,tm5,tm6,tm7,tm8,tm9,tm10,tm11,tm12,tm13,tm14,tm15,tm16,tm17
> tm <- unlist(tmList, use.names = FALSE)
> tm <- sort(unique(tm))
> ## set up cashflow matrix
> nR <- length(tm)
> nC <- length(cfList)
> cfMatrix <- array(0, dim = c(nR, nC))
> for(j in seq(nC))
+     cfMatrix[tm %in% tmList[[j]], j] <- cfList[[j]]
> rownames(cfMatrix) <- tm
> cfMatrix[1:10, 1:10]
```

```
    [,1] [,2]  [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
0.5 5.75 0.00   3.5    0 3.25 5.75  0.0    5    0     3
1   0.00 4.25   0.0    3 0.00 0.00  3.5    0  105     0
1.5 5.75 0.00 103.5    0 3.25 5.75  0.0    5    0     3
2   0.00 4.25   0.0    3 0.00 0.00  3.5    0    0     0
2.5 5.75 0.00   0.0    0 3.25 5.75  0.0    5    0     3
3   0.00 4.25   0.0    3 0.00 0.00  3.5    0    0     0
3.5 5.75 0.00   0.0    0 3.25 5.75  0.0    5    0     3
4   0.00 4.25   0.0    3 0.00 0.00  3.5    0    0     0
4.5 5.75 0.00   0.0    0 3.25 5.75  0.0    5    0     3
5   0.00 4.25   0.0    3 0.00 0.00  3.5    0    0     0
```

Suppose we have zero rates for all maturities (ie, one for each row of `cfMatrix`), then we can transform this vector of rates into discount factors. Premultiplying `cfMatrix` by the row vector of discount factors then gives us a row vector of bond prices.

```
> betaTRUE <- c(5,-2,1,10,1,3)
> yM <- NSS(betaTRUE,tm)
> diFa <- 1 / ( (1 + yM/100)^tm )
> bM <- diFa %*% cfMatrix
```

So, with a vector of ‘true’ bond prices `bm`, we can set up DE.

```
> data <- list(bM = bM, tm = tm, cfMatrix = cfMatrix, model = NSS,
+               ww = 1,
+              min = c( 0,-15,-30,-30,0  ,2.5),
+              max = c(15, 30, 30, 30,2.5,5  ))
```

The objective function takes the path that we just saw: given parameters for the NSS model, it computes zero rates, and transforms these into discount factors. Given the matrix `cfMatrix`, it then computes theoretical bond prices, and compares these with the given prices `bm`. As the optimisation criterion, we use the maximum absolute difference.

```
> OF2 <- function(param, data) {
+     tm <- data$tm; bM <- data$bM
+     model <- data$model; cfMatrix <- data$cfMatrix
+     diFa  <- 1 / ((1 + model(param,tm)/100)^tm)
+     b <- diFa %*% cfMatrix
+     aux <- b - bM; aux <- max(abs(aux))
+     if (is.na(aux)) aux <- 1e10
+     aux
+ }
```

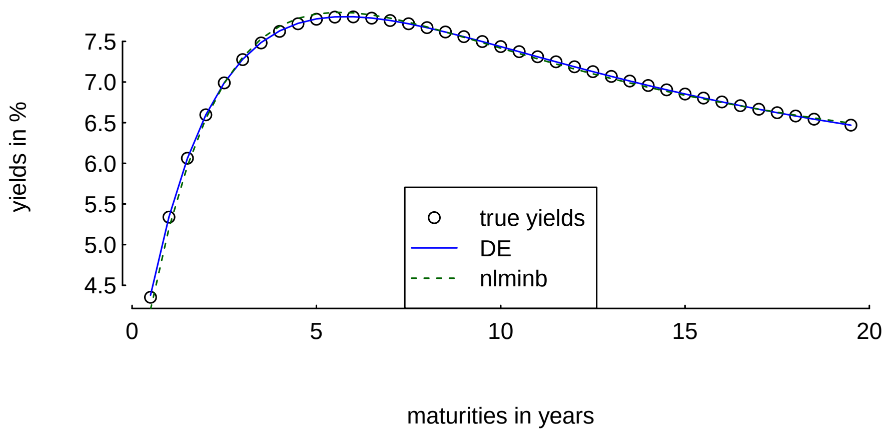
We set up the parameters and run DE.

```
> algo <- list(nP  = 200L,
+              nG  = 600L,
+              F   = 0.50,
+              CR  = 0.99,
+              min = c( 0,-15,-30,-30,0  ,2.5),
+              max = c(15, 30, 30, 30,2.5,5  ),
+              pen = penalty,
+              repair = NULL,
+              loopOF = TRUE,
+              loopPen = FALSE,
+              loopRepair = FALSE,
+              printBar = FALSE,
+              printDetail = FALSE)
> sol <- DEopt(OF = OF2, algo = algo, data = data)
> ## maximum yield error and value of OF
> max(abs(data$model(sol$xbest,tm) - data$model(betaTRUE,tm)))

[1] 0.0249778

> sol$OFvalue

[1] 0.002598282
```

Note that now the objective function value (the difference in bond prices) does not correspond to the yield difference anymore. It is instructive to compare them nevertheless.

```
> s0 <- algo$min + (algo$max - algo$min) * runif(length(algo$min))
> system.time(sol2 <- nlminb(s0,OF2,data = data,
+                                  lower = data$min,
+                                  upper = data$max,
+                                control = list(eval.max = 50000,
+                                               iter.max = 50000)))

   user  system elapsed
  0.048   0.000   0.049

> # maximum error yield and value of OF
> max(abs(data$model(sol2$par,tm) - data$model(betaTRUE,tm)))

[1] 0.1511508

> sol2$objective

[1] 0.1318322
```

```
> par(ps = 11, bty = "n", las = 1, tck = 0.01,
+     mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years", ylab = "yields in %")
> lines(tm,data$model(sol$xbest,tm), col = "blue")
> lines(tm,data$model(sol2$par,tm), col = "darkgreen", lty = 2)
> legend(x = "bottom", legend = c("true yields", "DE", "nlminb"),
+        col = c("black", "blue", "darkgreen"),
+        pch = c(1, NA, NA), lty = c(0, 1, 2))
```

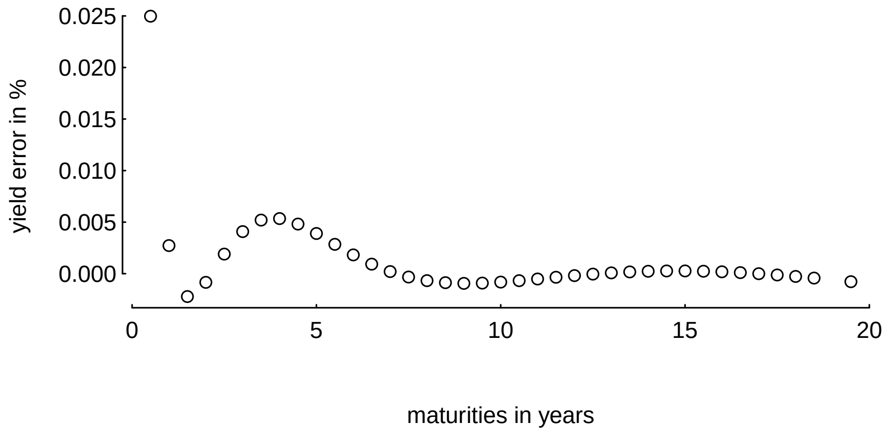


We can check the price errors.

```
> diFa <- 1 / ((1 + NSS(sol$xbest,tm)/100)^tm)
> b <- diFa %*% cfMatrix
> b - bM

             [,1]         [,2]        [,3]         [,4]         [,5]
[1,] 0.0009042739 -5.91043e-06 0.002558904 -0.002028559 0.0003693585
             [,6]         [,7]        [,8]         [,9]         [,10]
[1,] -0.001795275 -0.002589598 0.001266119 -0.002582812 -0.0009820629
           [,11]       [,12]       [,13]       [,14]       [,15]        [,16]
[1,] 0.001717525 0.001140375 0.000758916 7.80463e-05 0.002592883 -0.002598282
            [,17]       [,18]       [,19]        [,20]
[1,] -0.001111361 0.002555857 0.001822755 -0.002419504
```

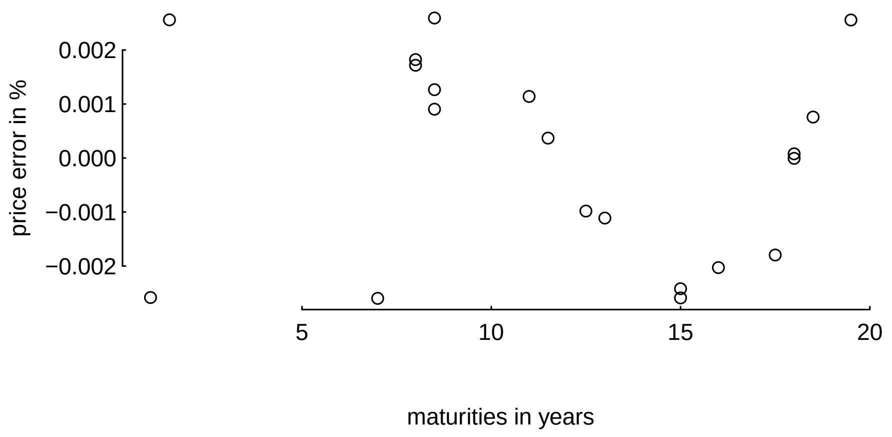
We can also plot the rate errors against time-to-payment.

```
> par(ps = 11, bty = "n", las = 1, tck = 0.01,
+     mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, NSS(sol$xbest,tm) - NSS(betaTRUE,tm),
+      xlab = "maturities in years", ylab = "yield error in %")
```



These apparently systematic (albeit small) errors are less visible when we plot price errors against time-to-maturity (see the book for a discussion).

```
> par(ps = 11, bty = "n", las = 1, tck = 0.01,
+     mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(as.numeric(unlist(lapply(tmList, max))), as.vector(b - bM),
+      xlab = "maturities in years", ylab = "price error in %")
```



5 Fitting the NSS model to given yields-to-maturity

We will need the following function; it converts cash flows and times-to-payment into present values, and those present values into yields-to-maturities.

```
> compYield <- function(cf,tm, guess = NULL) {
+     fy <- function(ytm,cf,tm) sum( cf / ( (1+ytm)^tm ) )
+     logik <- cf != 0
+     cf <- cf[logik]
+     tm <- tm[logik]
+     if (is.null(guess)) {ytm <- 0.05} else {ytm <- guess}
+     h <- 1e-8;        dF <- 1; ci <- 0
+     while (abs(dF) > 1e-5) {
+         ci <- ci + 1; if (ci > 5) break
+         FF <- fy(ytm,cf,tm)
+         dFF <- (fy(ytm+h, cf, tm)-FF) / h
+         dF <- FF / dFF
+         ytm <- ytm - dF
+     }
+     if (ytm < 0) ytm <- 0.99
+     return(ytm)
+ }
```

The objective function, `OF3`, looks as follows.

```
> OF3 <- function(param,data) {
+     tm <- data$tm; rM <- data$rM
+     model <- data$model; cfMatrix<- data$cfMatrix
+     nB <- dim(cfMatrix)[2L]
+     zrates <- model(param,tm); aux <- 1e10
+     if ( all(zrates > 0,
+              !is.na(zrates))
+         ) {
+         diFa <- 1 / ((1 + zrates/100)^tm)
+         b <- diFa %*% cfMatrix
+         r <- numeric(nB)
+         if ( all(!is.na(b),
+                  diFa < 1,
+                  diFa > 0,
+                  b > 1)
+             ) {
+             for (bb in 1:nB) {
+                 r[bb] <- compYield(c(-b[bb], cfMatrix[ ,bb]), c(0,tm))
+             }
+             aux <- abs(r - rM)
+             aux <- sum(aux)
+         }
+     }
+     aux
+ }
```

So the game plan is as follows: we compute prices `b` as in the last section, but then we convert them into yields-to-maturity `r` with the function `compYield`. The objective function evaluates the discrepancy between the market yields-to-maturity `rM` and our model yields `r`. We start by defining the 'true' `rM`.

```
> betaTRUE <- c(5,-2,1,10,1,3)
> yM <- NSS(betaTRUE, tm)
> diFa <- 1 / ( (1 + yM/100)^tm )
> bM <- diFa %*% cfMatrix
> rM <- apply(rbind(-bM, cfMatrix), 2, compYield, c(0, tm))
```

We set up `data` and `algo`.

```
> data <- list(rM = rM, tm = tm,
+              cfMatrix = cfMatrix,
+              model = NSS,
+              min = c( 0,-15,-30,-30,0  ,2.5),
+              max = c(15, 30, 30, 30,2.5,5  ), ww = 0.1)
> algo <- list(nP = 50L,
+              nG = 500L,
+              F  = 0.50,
+               CR = 0.99,
+               min = c( 0,-15,-30,-30,0  ,2.5),
+               max = c(15, 30, 30, 30,2.5,5  ),
+               pen = penalty,
+               repair = NULL,
+               loopOF = TRUE,
+               loopPen = FALSE,
+               loopRepair = FALSE,
+               printBar = FALSE,
+               printDetail = FALSE)
```

```
> system.time(sol <- DEopt(OF = OF3, algo = algo, data = data))
```

```
   user  system elapsed
 82.534   0.000  82.535
```

```
> max(abs(data$model(sol$xbest,tm) - data$model(betaTRUE,tm)))
```

```
[1] 0.0319574
```

```
> sol$OFvalue
```

```
[1] 0.0002154321
```

With nlminb:

```
> s0 <- algo$min + (algo$max - algo$min) * runif(length(algo$min))
> system.time(sol2 <- nlminb(s0, OF3, data = data,
+                                  lower = algo$min,
+                                  upper = algo$max,
+                                control = list(eval.max = 50000L,
+                                               iter.max = 50000L)))
```

```
   user  system elapsed
  2.144   0.000   2.141
```

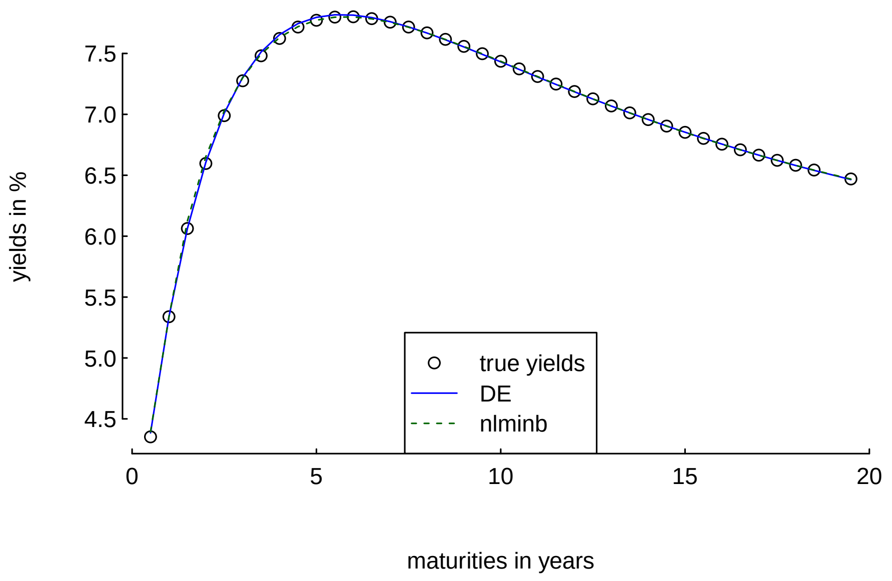
```
> max(abs(data$model(sol2$par,tm) - data$model(betaTRUE,tm)))
```

```
[1] 0.06121153
```

```
> sol2$objective
```

```
[1] 0.0006661621
```

```
> par(ps = 11, bty = "n", las = 1, tck = 0.01,
+     mgp = c(3, 0.2, 0), mar = c(4, 4, 1, 1))
> plot(tm, yM, xlab = "maturities in years", ylab = "yields in %")
> lines(tm,data$model(sol$xbest,tm), col = "blue")
> lines(tm,data$model(sol2$par,tm), col = "darkgreen", lty = 2)
> legend(x = "bottom", legend = c("true yields","DE","nlminb"),
+        col = c("black", "blue", "darkgreen"),
+        pch = c(1,NA,NA), lty = c(0,1,2))
```



Compare the recovered parameters.

```
> betaTRUE

[1]  5 -2  1 10  1  3

> round(sol$xbest,3)

[1]  4.825 -1.689  7.509  5.120  2.076  4.084
```

While the returned `OF` value will typically be acceptable, we need many more iterations to have the parameters converge. But compare the fitted yield curve: the fitted yields are generally fine. If you need more precision, just increase the number of generations (and possibly adjust the tolerance in the `while` condition in function `compYield`).

References

Manfred Gilli and Enrico Schumann. A Note on 'Good Starting Values' in Numerical Optimisation. *COMISEF Working Paper Series No. 44*, 2010. available from `http://comisef.eu/?q=working_papers`.

Manfred Gilli, Stefan Große, and Enrico Schumann. Calibrating the Nelson–Siegel–Svensson model. *COMISEF Working Paper Series No. 31*, 2010. available from `http://comisef.eu/?q=working_papers`.

Manfred Gilli, Dietmar Maringer, and Enrico Schumann. *Numerical Methods and Optimization in Finance*. Elsevier, 2011.