Fitting the Nelson–Siegel–Svensson model with Differential Evolution

Enrico Schumann
es@enricoschumann.net

1 Introduction

In this tutorial we look into fitting the Nelson–Siegel–Svensson (NSS) model to data; for more details, please see [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. Since we will use a stochastic technique for optimisation, we should be running several restarts (see Gilli et al., 2011, Chapter 12, for a discussion). The variable nRuns sets the number of restarts for the examples to come. We set it to only two here to keep the build-time of the package acceptable; increase it to check the stochastics of the solutions. We set a seed to make the computations reproducible.

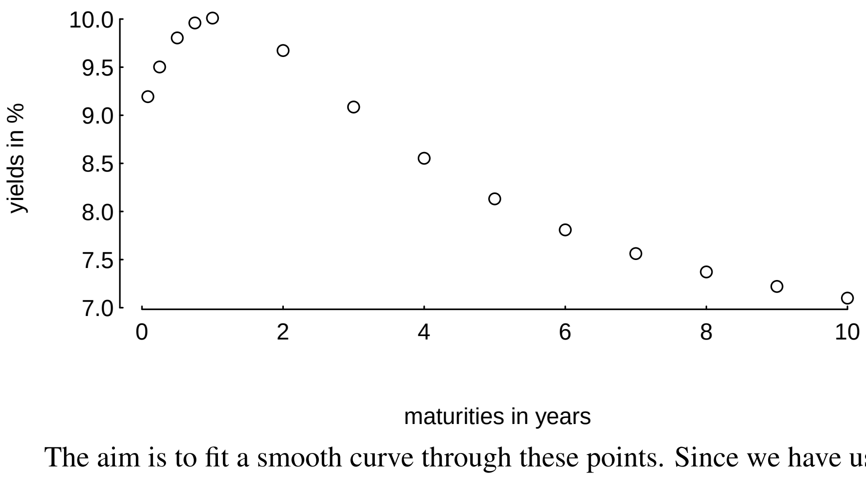
```
> require("NMOF")
> nRuns <- 2L
> set.seed(112233)
```

2 Fitting the NS model to given zero rates

The NS model

We create 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. It returns the maximum absolute difference between a vector of observed ('market') yields `yM`, and the model's yields for parameters `param`.

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

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         ## the solution...
> OF(param1, data)           ## ...gives 0
```

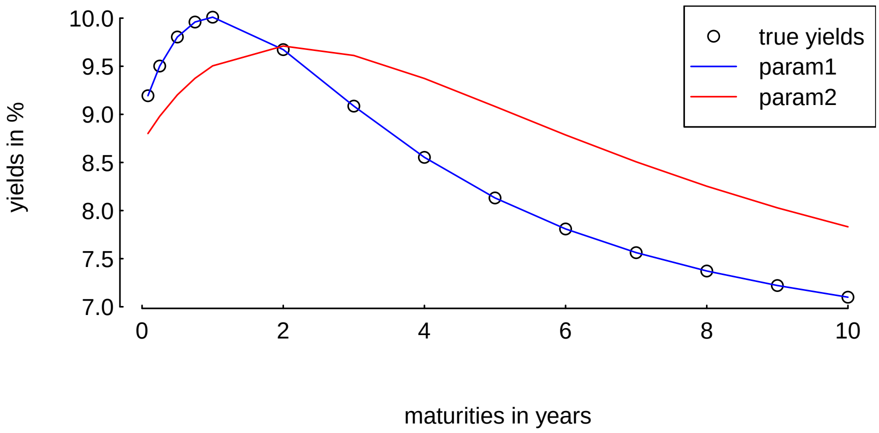
```
[1] 0
```

```
> param2 <- c(5.7, 3, 8, 2)  ## anything else
> OF(param2, data)           ## ... gives a postive number
```

```
[1] 0.97686
```

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[1L, ] + mP[2L, ]) - abs(mP[1L, ] + mP[2L, ]))
      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
 [ reached getOption("max.print") -- omitted 1 row ]
```

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,   ## population size
               nG = 500L,   ## number of generations
                F = 0.50,   ## step size
               CR = 0.99,   ## prob. of crossover
              min = c( 0,-15,-30, 0),
              max = c(15, 30, 30,10),
              pen = penalty,
           repair = NULL,
           loopOF = TRUE,   ## loop over popuation? yes
          loopPen = FALSE,  ## loop over popuation? no
       loopRepair = TRUE,   ## loop over popuation? yes
         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.
Best solution has objective function value 0 ;
standard deviation of OF in final population is 4.2399e-16 .
```

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 performed 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] 0.6107
```

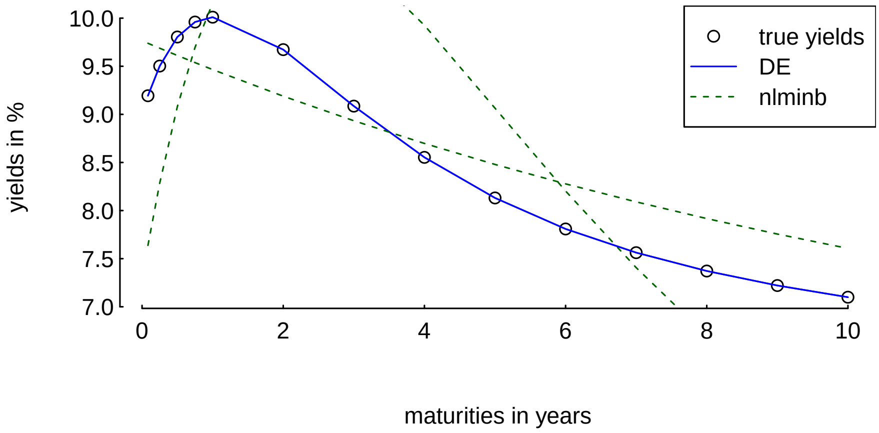
```
> sol2$objective
```

```
[1] 0.6107
```

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. To keep the build-time of the vignette down, we only run both methods once. But increase `nRuns` for more restarts.

```
> 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))
```

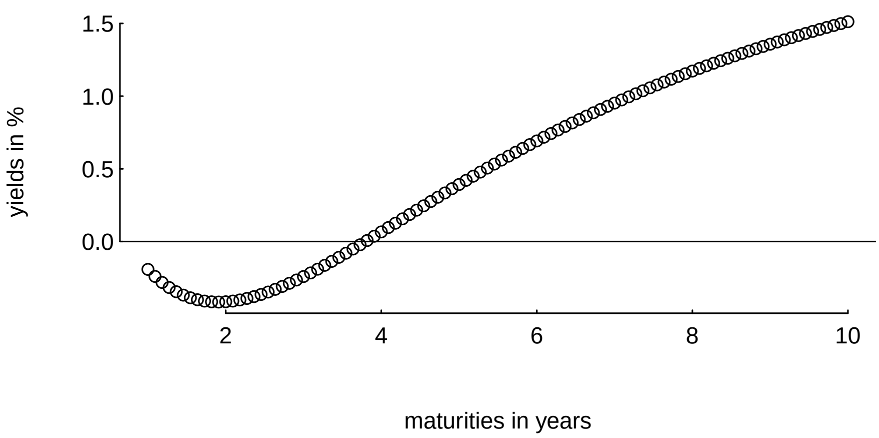


It is no error that there tpyically appears to be only one curve for DE: there are, in fact, `nRuns` lines, but they are printed on top of each other.

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)
      maxdiff <- abs(y - abs(y))
      sum(maxdiff) * data$ww
  }
```

Check:

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

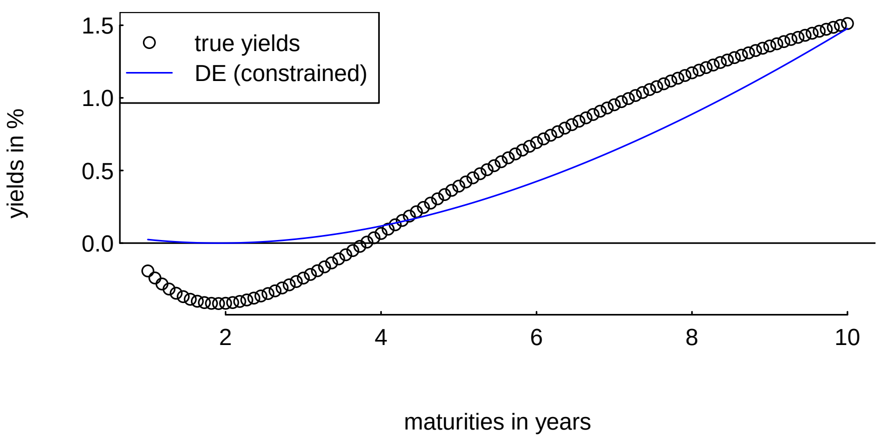
```
[1] 0.86343
```

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. (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] 7.5495e-15
```

```
> sol$OFvalue
```

```
[1] 7.5495e-15
```

We compare the results with nlminb.

```
> 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] 0.068345
```

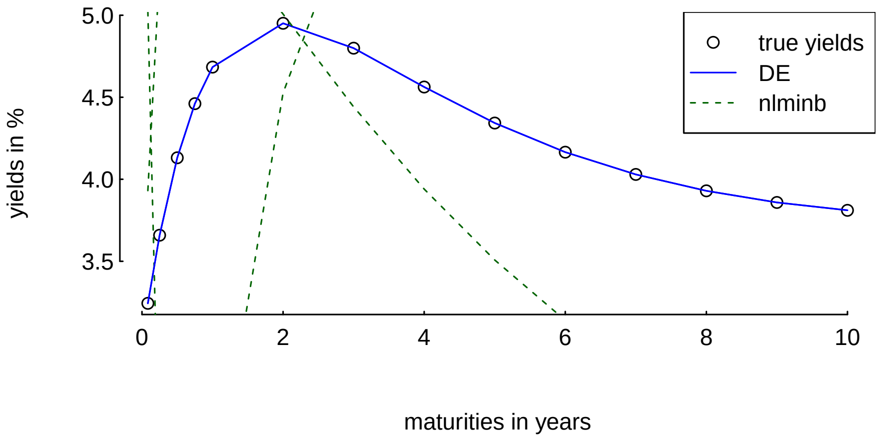
```
> sol2$objective
```

```
[1] 0.068345
```

Finally, we compare the yield curves resulting from several runs. (Recall that the number of runs is controlled by nRuns, which we have set initially.)

```
> 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

The section was removed to reduce the build-time of the package. The examples were moved to the ‘NMOF manual’ (see `http://enricoschumann.net/NMOF.htm`). The code is in the subdirectory `NMOFex`; to show the code in R, you can use the function `system.file`.

```
> whereToLook <- system.file("NMOFex/NMOFman.R", package = "NMOF")
> file.show(whereToLook, title = "NMOF examples")
```

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

The section was removed to reduce the build-time of the package. The examples were moved to the ‘NMOF manual’ (see `http://enricoschumann.net/NMOF.htm`). The code is in the subdirectory `NMOFex`; to show the code in R, you can use the function `system.file`.

```
> whereToLook <- system.file("NMOFex/NMOFman.R", package = "NMOF")
> file.show(whereToLook, title = "NMOF examples")
```

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.