An Alternative Majorization for Multidimensional Scaling

Jan de Leeuw

Version April 09, 2018

Abstract

The Cauchy-Schwartz majorization of the distance function in SMACOF is replaced by a majorization of the squared distance function. This leads to an interesting SMACOF alternative, which we call SMOCAF.

Contents

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The directory gifi.stat.ucla.edu/cmds has a pdf version, the bib file, the complete Rmd file with the code chunks, and the R source code.

1 Introduction

The (Euclidean, metric, least squares) multidimensional scaling or MDS problem is a special case of a nonconvex optimization problem in which we minimize a function of the form

$$f(x) \triangleq 1 + \frac{1}{2}x'x - \sum_{i=1}^{n}\sqrt{x'A_ix},$$

where the A_i are given positive semi-definite matrices. Because the third term on the right, the one with the square root, is convex, the problem is a DC-problem (Le Thi and Tao (2018)),

i.e. the minimization of the difference of two convex functions, one of which is quadratic and one of which is not differentiable at some points. We do not really have to worry about non-differentiability, because of the result in De Leeuw (1984).

2 SMACOF

The SMACOF algorithm (De Leeuw (1977), De Leeuw and Heiser (1977), De Leeuw and Mair (2009)) to minimize the MDS objective function f is a majorization algorithm (De Leeuw (1994)), currently more commonly known as an MM algorithm (Lange (2016)). It is based on the Cauchy-Schwarz inequality

$$\sqrt{x'A_ix\ y'A_iy} \geq x'A_iy.$$

Define

$$e_i(x) \triangleq \begin{cases} (x'A_ix)^{-\frac{1}{2}} & \text{if } x'A_ix \neq 0, \\ 0 & \text{otherwise .} \end{cases}$$

and

$$g(x,y) \triangleq 1 + \frac{1}{2}x'x - \sum_{i=1}^n e_i(y)x'A_iy,$$

then we have the basic majorization relations $f(x) \leq g(x,y)$ and $f(x) = g(x,x)$. Thus step k in the iterative algorithm minimizes $g(x, x^{(k)})$ over x, which leads to the convergent algorithm

$$x^{(k+1)} = \sum_{i=1}^n e_i(x^{(k)})A_ix^{(k)}.$$

3 SMOCAF

For our alternative majorization we use the inequality

$$x'A_ix \geq 2x'A_iy - y'A_iy.$$

Define the polyhedral convex set $\mathcal{C}(y)$ of all x such that $2x'A_iy - y'A_iy \geq 0$ for all i. Clearly $y \in \mathcal{C}(y)$, and thus $\mathcal{C}(y)$ is non-empty. Also define

$$h(x,y) \triangleq 1 + \frac{1}{2}x'x - \sum_{i=1}^n \sqrt{2x'A_iy - y'A_iy},$$

for all $x \in \mathcal{C}(y)$. Because the square root of a non-negative linear form is concave, minimizing $h(x, x^{(k)})$ over $x \in \mathcal{C}(x^{(k)})$ is a convex problem with linear constraints and leads to a convergent MM algorithm, which we call SMOCAF.

Of course a SMOCAF step is inherently more complicated than a SMACOF step. The idea is interesting, at least to me, because the SMOCAF algorithm solves a sequence of convex programming problems in which the constraint set changes over iterations.

4 Example

We try out both SMACOF and SMOCAF on a simple example, where there are 10 matrices A_i, all covariance matrices of a 20 by 5 matrix filled with random standard normals. The SMOCAF iterations are implemented using `constrOptim()` from base R, which uses a logarithmic barrier method for the convex programming step. Since we have very good initial estimates, and we do not really have to iterate until convergence within each step, I assume the SMOCAF implementation can be made much more efficient. In addition, acceleration techniques specific to DC programming, such as the one discussed in Artacho, Fleming, and Vuong (2017), could be applied. Or general acceleration techniques, discussed recently in great detail in Sidi (2017). But optimal speed is not what interests us here.

```
system.time(print(smacof(a, rep(1,5), verbose = FALSE)))
```

```
## $itel
## [1] 116
##
## $stress
## [1] -61.77740087
##
## $coef
## [1]  0.7270085698  6.3676564611  8.5480388387 -2.1757084262  2.1625300759
##
## $grad
## [1] -7.316141844e-07 -4.745316849e-06  4.648248849e-06  5.576051584e-06
## [5]  1.455168160e-06
```

```
##    user  system elapsed
##   0.034   0.001   0.035
```

```
system.time(print(smocaf(a, rep(1,5), verbose = FALSE)))
```

```
## $itel
## [1] 120
##
## $stress
## [1] -61.77740087
##
## $coef
## [1]  0.727008322  6.367654855  8.548040412 -2.175706538  2.162530569
##
## $grad
## [1] -7.554716913e-07 -4.899925776e-06  4.799694909e-06  5.757723873e-06
## [5]  1.502571069e-06
```

```
##    user  system elapsed
##   0.310   0.007   0.318
```

For both majorizations convergence is nice and smooth, and both converge to the same solution. The number of iterations is about the same, but since SMOCAF iterations are considerably more time-consuming the overall SMOCAF time is about 10 times the SMACOF time.

5 Appendix: Code

5.1 smacof.R

```
smacof <-
  function (a,
            xold,
            itmax = 1000,
            eps = 1e-10,
            verbose = TRUE) {
    n <- dim(a)[3]
    m <- dim(a)[1]
    dold <- rep(0, n)
    itel <- 1
    for (i in 1:n)
      dold[i] <- sqrt (sum (a[, , i] * outer (xold, xold)))
    sold <- sum (xold ^ 2) / 2 - sum(dold)
    repeat {
      xnew <- rep (0, m)
      dnew <- rep (0, n)
      for (i in 1:n)
        xnew <- xnew + drop(a[, , i] %*% xold) / dold [i]
      for (i in 1:n)
        dnew[i] <- sqrt (sum (a[, , i] * outer (xnew, xnew)))
      snew <- sum (xnew ^ 2) / 2 - sum (dnew)
      if (verbose)
        cat(
          "Iteration: ",
          formatC (itel, width = 3, format = "d"),
          "sold: ",
          formatC (
            sold,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "snew: ",
```

```
          formatC (
            snew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "\n"
        )
      if ((sold - snew) < eps || (itel == itmax))
        break
      sold <- snew
      xold <- xnew
      dold <- dnew
      itel <- itel + 1
    }
    g <- xnew
    for (i in 1:n) {
      g <- g - drop(a[, , i] %*% xnew) / dnew[i]
    }
    return(list(
      itel = itel,
      stress = snew,
      coef = xnew,
      grad = g
    ))
  }
```

5.2 smocaf.R

```
obj <- function (x, y, a) {
  f <- sum (x ^ 2) / 2
  for (i in 1:10) {
    f <- f -  sqrt (abs (sum (a[, , i] * outer (2 * x - y, y))))
  }
  return (f)
}

grad <- function (x, y, a) {
  g <- x
  for (i in 1:10) {
    g <-
      g -  (1 / sqrt (abs (sum (
```

```
        a[, , i] * outer (2 * x - y, y)
      )))) * a[, , i] %*% y
  }
  return (g)
}

smocaf <-
  function (a,
            xold,
            itmax = 1000,
            eps = 1e-10,
            verbose = TRUE) {
    n <- dim(a)[3]
    m <- dim(a)[1]
    dold <- rep(0, n)
    itel <- 1
    for (i in 1:n)
      dold[i] <- sqrt (sum (a[, , i] * outer (xold, xold)))
    sold <- sum (xold ^ 2) / 2 - sum(dold)
    repeat {
      uold <- matrix (0, n, m)
      cold <- rep (0, n)
      dnew <- rep (0, n)
      for (i in 1:n) {
        uold[i, ] <- 2 * a[, , i] %*% xold
        cold[i] <- sum (a[, , i] * outer (xold, xold))
      }
      h <-
        constrOptim (xold, obj, grad, uold, cold, y = xold, a = a)
      xnew <- h$par
      for (i in 1:n)
        dnew[i] <- sqrt (sum (a[, , i] * outer (xnew, xnew)))
      snew <- sum (xnew ^ 2) / 2 - sum (dnew)
      if (verbose)
        cat(
          "Iteration: ",
          formatC (itel, width = 3, format = "d"),
          "sold: ",
          formatC (
            sold,
            digits = 8,
            width = 12,
            format = "f"
          ),
```

```
          "snew: ",
          formatC (
            snew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "\n"
        )
      if ((sold - snew) < eps || (itel == itmax))
        break
      sold <- snew
      xold <- xnew
      dold <- dnew
      itel <- itel + 1
    }
    g <- xnew
    for (i in 1:n) {
      g <- g - drop(a[, , i] %*% xnew) / dnew[i]
    }
    return(list(
      itel = itel,
      stress = snew,
      coef = xnew,
      grad = g
    ))
  }
```

References

Artacho, F. J. A., R.M. T. Fleming, and P. T. Vuong. 2017. "Accelerating the DC Algorithm for Smooth Functions." *Mathematical Programming, Series B*. https://link.springer.com/content/pdf/10.1007%2Fs10107-017-1180-1.pdf.

De Leeuw, J. 1977. "Applications of Convex Analysis to Multidimensional Scaling." In *Recent Developments in Statistics*, edited by J.R. Barra, F. Brodeau, G. Romier, and B. Van Cutsem, 133–45. Amsterdam, The Netherlands: North Holland Publishing Company. http://www.stat.ucla.edu/~deleeuw/janspubs/1977/chapters/deleeuw_C_77.pdf.

———. 1984. "Differentiability of Kruskal's Stress at a Local Minimum." *Psychometrika* 49: 111–13. http://www.stat.ucla.edu/~deleeuw/janspubs/1984/articles/deleeuw_A_84f.pdf.

———. 1994. "Block Relaxation Algorithms in Statistics." In *Information Systems and Data Analysis*, edited by H.H. Bock, W. Lenski, and M.M. Richter, 308–24. Berlin: Springer Verlag.

http://www.stat.ucla.edu/~deleeuw/janspubs/1994/chapters/deleeuw_C_94c.pdf.

De Leeuw, J., and W.J. Heiser. 1977. "Convergence of Correction Matrix Algorithms for Multidimensional Scaling." In *Geometric Representations of Relational Data*, edited by J.C. Lingoes, 735–53. Ann Arbor, Michigan: Mathesis Press. http://www.stat.ucla.edu/~deleeuw/janspubs/1977/chapters/deleeuw_heiser_C_77.pdf.

De Leeuw, J., and P. Mair. 2009. "Multidimensional Scaling Using Majorization: SMACOF in R." *Journal of Statistical Software* 31 (3): 1–30. http://www.stat.ucla.edu/~deleeuw/janspubs/2009/articles/deleeuw_mair_A_09c.pdf.

Lange, K. 2016. *MM Optimization Algorithms*. SIAM.

Le Thi, H. A., and P. D. Tao. 2018. "DC Programming and DCA: Thirty Years of Developments." *Mathematical Programming, Series B*. https://link.springer.com/content/pdf/10.1007%2Fs10107-018-1235-y.pdf.

Sidi, A. 2017. *Vector Extrapolation Methods with Applicatons*. SIAM.