Discrete Minimax by Quadratic Majorization

Jan de Leeuw

Version 16, December 14, 2016

Abstract

We construct piecewise quadratic majorizers for minimax problems. This is appled to finding roots of cubics. An application to a Chebyshev versions of MDS loss is also outlined.

Contents

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The directory deleeuwpdx.net/pubfolders/minimax has a pdf version, the complete Rmd file with all code chunks, the bib file, and the R source code.

1 Introduction

Suppose $\mathcal{I}$ is the closed interval $[L, U]$, and $f : \mathcal{I} \to \mathbb{R}$. A function $g : \mathcal{I} \otimes \mathcal{I} \to \mathbb{R}$ is a *majorization scheme* for f on $\mathcal{I}$ if

- $g(x, x) = f(x)$ for all $x \in \mathcal{I}$,
- $g(x, y) \geq f(x)$ for all $x, y \in \mathcal{I}$.

In other words for each y the global minimum of $g(x, y) - f(x)$ over $x \in \mathcal{I}$ is zero, and it is attained at y. If the functions f and g are differentiable and the minimum is attained at an

interior point of the interval, we have $\mathcal{D}_1 g(x,y) = \mathcal{D}f(x)$. If the functions are in addition twice differentiable we have $\mathcal{D}_{11} g(x,y) \geq \mathcal{D}^2 f(x)$.

The majorization conditions are not symmetric in x and y, and consequently it sometimes is more clear to write $g_y(x)$ for $g(x,y)$, so that $g_y : \mathcal{I} \to \mathbb{R}$. We say that g_y majorizes f on $\mathcal{I}$ at y, or with *support point* y.

A *majorization algorithm* is of the form

$$x^{(k+1)} \in \operatorname*{\mathbf{argmin}}_{x \in \mathcal{I}} g(x, x^{(k)})$$

It then follows that

$$f(x^{(k+1)}) \leq g(x^{(k+1)}, x^{(k)}) \leq g(x^{(k)}, x^{(k)}) = f(x^{(k)}), \tag{1}$$

Thus a majorization step decreases the value of the objective function. The chain (1) is called the *sandwich inequality*. In (1) the inequality $f(x^{(k+1)}) \leq g(x^{(k+1)}, x^{(k)})$ follows from majorization, the inequality $g(x^{(k+1)}, x^{(k)}) \leq g(x^{(k)}, x^{(k)})$ follows from minimization. This explains why majorization algorithms are also called *MM algorithms* (Lange (2016)). Using *MM* has the advantage that it can be used for the dual family of minorization-maximization algorithms.

2 Majorizing a Maximum

Theorem 1: [**Majorization of Maximum**] Suppose $p(x) = \max_{i=1}^n f_i(x)$, and g_i majorizes f_i on $\mathcal{I}$ at y. Define $q(x,y) := \max_{i=1}^n g_i(x,y)$. Then q majorizes p on $\mathcal{I}$ at y.

Proof: First

$$q(y,y) = \max_{i=1}^n g_i(y,y) = \max_{i=1}^n f_i(y) = p(y).$$

Second

$$p(x) = \max_{i=1}^n f_i(x) = f_k(x) \leq g_k(x,y) \leq \max_{i=1}^n g_i(x,y) = q(x,y).$$

QED

2.1 Quadratic Majorization of a Maximum

Suppose the majorizing function g_i are quadratics. We have

$$g_i(x,y) = f_i(y) + f_i'(y)(x-y) + \frac{1}{2}K_i(x-y)^2. \tag{2}$$

In a majorization step we have to compute

$$x^{(k+1)} = \operatorname*{\mathbf{argmin}}_{L \leq x \leq U} \max_{i=1}^n f_i(x^{(k)}) + f_i'(x^{(k)})(x - x^{(k)}) + \frac{1}{2}K_i(x - x^{(k)})^2.$$

Now it is easy, at least conceptually, to design a majorization algorithm. For each $i < j$ we solve the quadratic equation $g_i(x, x^{(k)}) = g_j(x, x^{(k)})$. If the solutions (there may be zero, one, or two) are in $[L, U]$ we add them to a vector of *knots*. If we are done we add L and U to the knots as well, and we sort the vector of knots. In the interval between two successive knots there is a unique largest g_i, and we minimize that g_i over that interval. We visit all intervals, going from left to right. By keeping track of the various minima we find the smallest one, which provides us with the minimizer we were looking for.

2.2 Lifting

There are two aspects of our general formulation which complicate the majorization. First, the K_i in (2) can be negative, and second the K_i can be unequal. Both problems can be circumvented by realizing that if (2) defines a majorizer, then any quadratic of the same form with a larger K_i defines a majorizer as well. Thus we can always choose K_i to be positive. The majorizers will now be convex quadratics. This has the additional benefit that q, as a maximum of convex functions, is convex in x as well. This simplifies the search over the knot intervals. If the function is initially decreasing of we move from left to right we can stop as soon as it starts to increase.

And finally we can replace all K_i by $\max_{i=1}^n |K_i|$, a process we call *lifting*. The equations $g_i(x, x^{(k)}) = g_j(x, x^{(k)})$ now become linear, and they have at most one solution in $[L, U]$. Thus we will generally have fewer knots and fewer intervals.

In fact, with lifting the majorization function can be minimized by solving a single (semi-definite) quadratic program. We minimize the quadratic $\eta + \frac{1}{2}(x - x^{(k)})'K(x - x^{(k)})$ over (x, η) under the linear inequality constraints that $f_i(x^{(k)}) + f_i'(x^{(k)})(x - x^{(k)}) \leq \eta$ for all i.

3 Multivariate Extensions

Suppose

$$g_i(x, y) = f_i(y) + (x - y)'\mathcal{D}f_i(y) + \frac{1}{2}(x - y)'K(x - y)$$

where the quadratic component contains the same positive semi-definite matrix K for all i, if necessary by using multivariate lifting. Again, there is a straightforward algorithm to minimize the maximum of the g_i. The set

$$\mathcal{P}_i(x^{(k)}) := \{x \mid g_i(x, x^{(k)}) \geq g_j(x, x^{(k)}), \quad \forall j \neq i\}$$

is a convex polyhedron (which may be empty). Minimizing g_i over $\mathcal{P}_i(x^{(k)}) \cap \mathcal{I}$ is a positive semi-definite quadratic programming problem. Pivot over all i and keep the best minimizer. Clearly in this case lifting is even more important than in the univariate case. Alternatively, as in the previous section, the problem can be reduced to a single quadratic program.

4 Examples

4.1 Roots of Cubics

Consider the problem of minimizing $|f(x)|$ on $[L, U]$, which obviously can be used to find roots of the equation $f(x) = 0$. Now $|f(x)| = \max(f(x), -f(x))$. We know that

$$g(x, y) = f(y) + f'(y)(x - y) + \frac{1}{2}K_0(x - y)^2$$

with $K_0 := \max_{L \leq x \leq U} |f''(x)|$ majorizes f on $[L, U]$. This is the uniform quadratic majorizer of De Leeuw (2016). In the same way

$$h(x, y) = -f(y) - f'(y)(x - y) + \frac{1}{2}K_0(x - y)^2$$

majorizes $-f$ on $[L, U]$. It follows that $\max(g(x, y), h(x, y))$ majorizes $|f|$ on $[L, U]$.

Let us look at the example of the cubic $\frac{1}{6}(x^3 - x)$, which has roots at $-1, 0, +1$. In this case $K_0 = 2$. Figure shows $|f|$ in red and three piecewise quadratic majorizers on $[-2, 2]$ with support points -1.5, 0, and 0.5 in blue. The vertical black lines are at the support points.

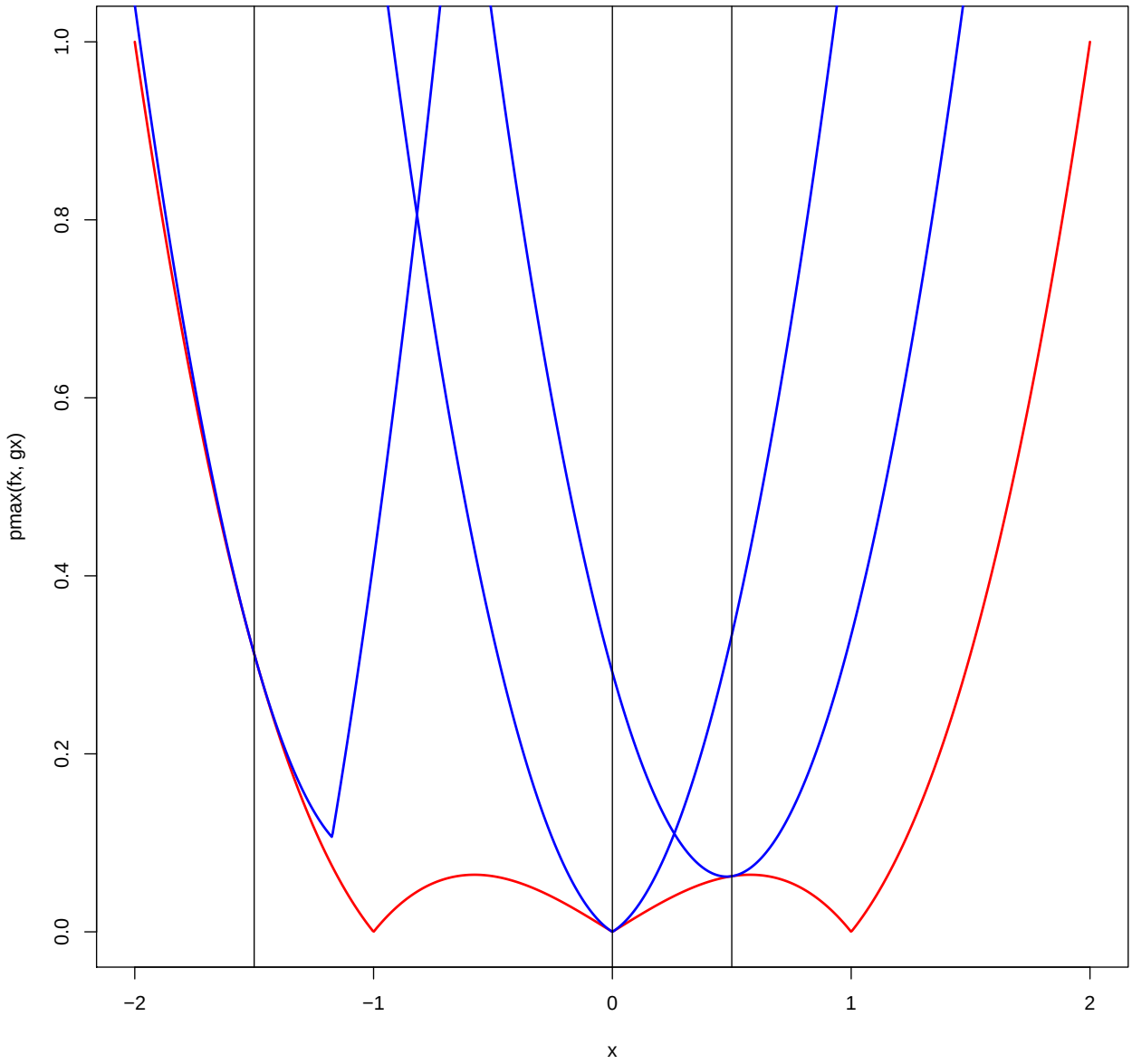

Figure 1: Uniform Piecewise Quadratic Majorizers

The iterations of the majorization algorithm, started from -1.5, 0.5, and 0.0, are given next. Convergence is fast and ultimately seems to be quadratic. Note that, in this example at least, convergence is to the point where the two quadratic majorizers are equal. At this point

$$f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)}) = -f(x^{(k)}) - f'(x^{(k)})(x^{(k+1)} - x^{(k)}),$$

or

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})},$$

which is a Newton step.

```
h <- myIterator (xinit = -1.5, f = myAbs, a = a, k = 2, lw = -2, up = 2, verbose = TRUE,
```

```
## Iteration:    1 xold:   -1.50000000 xnew:   -1.17391304 cnew:    0.32608696 rate:
```

```
## Iteration:    2 xold:   -1.17391304 xnew:   -1.03230713 cnew:    0.14160592 rate:
## Iteration:    3 xold:   -1.03230713 xnew:   -1.00145595 cnew:    0.03085117 rate:
## Iteration:    4 xold:   -1.00145595 xnew:   -1.00000317 cnew:    0.00145278 rate:
## Iteration:    5 xold:   -1.00000317 xnew:   -1.00000000 cnew:    0.00000317 rate:
## Iteration:    6 xold:   -1.00000000 xnew:   -1.00000000 cnew:    0.00000000 rate:
```

```
h <- myIterator (xinit =  0.5, f = myAbs, a = a, k = 2, lw = -2, up = 2, verbose = TRUE,
```

```
## Iteration:    1 xold:    0.50000000 xnew:    0.47916667 cnew:    0.02083333 rate:
## Iteration:    2 xold:    0.47916667 xnew:    0.45323351 cnew:    0.02593316 rate:
## Iteration:    3 xold:    0.45323351 xnew:    0.42125533 cnew:    0.03197818 rate:
## Iteration:    4 xold:    0.42125533 xnew:    0.38228601 cnew:    0.03896932 rate:
## Iteration:    5 xold:    0.38228601 xnew:    0.33548832 cnew:    0.04679769 rate:
## Iteration:    6 xold:    0.33548832 xnew:    0.28029309 cnew:    0.05519523 rate:
## Iteration:    7 xold:    0.28029309 xnew:    0.21660081 cnew:    0.06369228 rate:
## Iteration:    8 xold:    0.21660081 xnew:    0.14499646 cnew:    0.07160436 rate:
## Iteration:    9 xold:    0.14499646 xnew:    0.06691911 cnew:    0.07807734 rate:
## Iteration:   10 xold:    0.06691911 xnew:   -0.00060751 cnew:    0.06752663 rate:
## Iteration:   11 xold:   -0.00060751 xnew:    0.00000000 cnew:    0.00060751 rate:
## Iteration:   12 xold:    0.00000000 xnew:    0.00000000 cnew:    0.00000000 rate:
```

```
h <- myIterator (xinit =  0.0, f = myAbs, a = a, k = 2, lw = -2, up = 2, verbose = TRUE,
```

```
## Iteration:    1 xold:    0.00000000 xnew:    0.00000000 cnew:    0.00000000 rate:
```

We can find better quadratic majorizers of our cubics by using the results of De Leeuw (2016). Define the sharp quadratic majorizers for which

$$K(y) := \max_{L\leq x\leq U} f''(y) + \frac{1}{3}f'''(y)(x-y) = f''(y) + \frac{1}{3}\max\left(f'''(L-y), f'''(U-y)\right)$$

For $\frac{1}{6}(x^3-x)$ this gives $K_1(y) = y+\frac{1}{3}(U-y)$ and for $-\frac{1}{6}(x^3-x)$ it gives $K_2(y) = -y-\frac{1}{3}(L-y)$. If $L=-2$ and $U=+2$ then $K_1(y) = \frac{2}{3}y + \frac{2}{3}$ and $K_2(y) = -\frac{2}{3}y + \frac{2}{3}$. At $y=-\frac{3}{2}$ this means $K_1 = -\frac{1}{3}$ and $K_2 = \frac{5}{3}$. At $y = \frac{1}{2}$ we have $K_1 = 1$ and $K_2 = \frac{1}{3}$. At $y = 0$ we have $K_1 = K_2 = \frac{2}{3}$.

The piecewise quadratic majorizers are in figure 2. Note that for both $y = -\frac{3}{2}$ and for $y = \frac{1}{2}$ there are two interior knots.

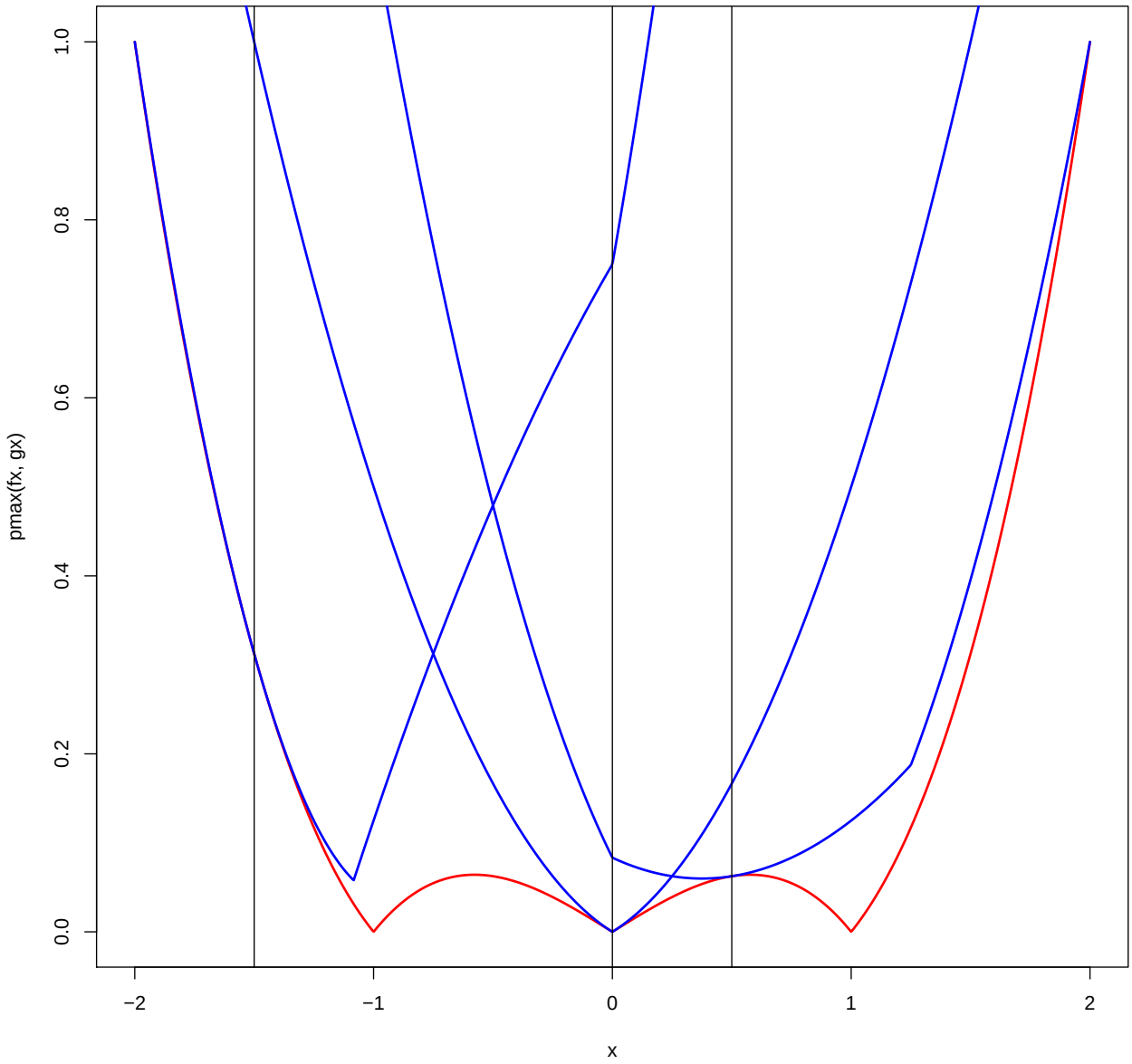

Figure 2: Sharp Piecewise Quadratic Majorizers

As expected, using no lifitng and sharp majorization requires fewer iterations to reach the roots, although of course each iteration is more work.

```
h <- myIterator (xinit = -1.5, f = myAbs, a = a, k = c(-1/3,5/3), lw = -2, up = 2, verbo
```

```
## Iteration:    1 xold:   -1.50000000 xnew:   -1.08333333 cnew:    0.41666667 rate:
## Iteration:    2 xold:   -1.08333333 xnew:   -1.00057225 cnew:    0.08276108 rate:
## Iteration:    3 xold:   -1.00057225 xnew:   -1.00000000 cnew:    0.00057225 rate:
## Iteration:    4 xold:   -1.00000000 xnew:   -1.00000000 cnew:    0.00000000 rate:
```

```
h <- myIterator (xinit =  0.5, f = myAbs, a = a, k = c(1,1/3), lw = -2, up = 2, verbose
```

```
## Iteration:    1 xold:    0.50000000 xnew:    0.37500000 cnew:    0.12500000 rate:
## Iteration:    2 xold:    0.37500000 xnew:    0.08593750 cnew:    0.28906250 rate:
```

```
## Iteration:    3 xold:    0.08593750 xnew:    0.00534433 cnew:    0.08059317 rate:
## Iteration:    4 xold:    0.00534433 xnew:    0.00002796 cnew:    0.00531637 rate:
## Iteration:    5 xold:    0.00002796 xnew:    0.00000000 cnew:    0.00002796 rate:
## Iteration:    6 xold:    0.00000000 xnew:    0.00000000 cnew:    0.00000000 rate:
```

```
h <- myIterator (xinit =  0.0, f = myAbs, a = a, k = c(2/3,2/3), lw = -2, up = 2, verbos
```

```
## Iteration:    1 xold:    0.00000000 xnew:    0.00000000 cnew:    0.00000000 rate:
```

The root-finding example can easily be extended to more interesting cases. If we have a convex function f with a quadratic majorizer, then $-F$ is concave and, if so desired, it can be lifted to have the same quadratic term.

4.2 MDS in the Chebyshev Norm

Consider the multidimensional scaling (MDS) problem of minimizing the loss function

$$\sigma(x) := \max_{i=1}^{n} w_i \left|\delta_i - \sqrt{x'A_ix}\right|.$$

Here the w_i are positive weights, the δ_i are positive dissimilarties, and the $\sqrt{x'A_ix}$ are Euclidean distances.

We give the quadratic majorizations that are needed. The first majorization, based on Cauchy-Schwartz, is actually linear. This is the basic SMACOF majorization, due to De Leeuw (1977).

$$\delta_i - \sqrt{x'A_ix} \leq \delta_i - \frac{1}{\sqrt{y'A_iy}}x'A_iy.$$

The second majorization, based on the AM/GM inequality, is quadratic. It was first used in the MDS context by (**???**).

$$\sqrt{x'A_ix} - \delta_i \leq \frac{1}{2}\frac{1}{\sqrt{y'A_iy}}\{x'A_ix + y'A_iy\} - \delta_i$$

We can add an appropriate quadratic term to the first (linear) majorization to achieve multivariate lifting.

We will not give a full implementation in this paper, but in principle it is quite straightforward. It also is of some interestto observe that the alternative loss funcion

$$\sigma(x) := \max_{i=1}^{n} w_i \left|\delta_i^2 - x'A_ix\right|$$

is already a maximum of quadratics, and consequently minimizing it does not require quadratic majorization. Without multivariate lifting, however, the regions where one quadratic function dominates all others can be quite complicated.

5 Appendix: Code

5.1 auxilary.R

```
mprint <- function (x, d = 2, w = 5) {
  print (noquote (formatC (
    x, di = d, wi = w, fo = "f"
  )))
}

myIterator <-
  function (xinit,
            f,
            eps = 1e-6,
            itmax = 100,
            verbose = FALSE,
            final = TRUE,
            ...) {
    xold <- xinit
    cold <- Inf
    itel <- 1
    repeat {
      xnew <- f (xold, ...)
      cnew <- abs (xnew - xold)
      rate <- cnew / cold
      if (verbose)
        cat(
          "Iteration: ",
          formatC (itel, width = 3, format = "d"),
          "xold: ",
          formatC (
            xold,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "xnew: ",
          formatC (
            xnew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "cnew: ",
```

```
        formatC (
          cnew,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "rate: ",
        formatC (
          rate,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "\n"
      )
    if ((cnew < eps) || (itel == itmax))
      break
    xold <- xnew
    cold <- cnew
    itel <- itel + 1
  }
  if (final)
    cat(
      "Iteration: ",
      formatC (itel, width = 3, format = "d"),
      "xinit: ",
      formatC (
        xinit,
        digits = 8,
        width = 12,
        format = "f"
      ),
      "xfinal: ",
      formatC (
        xnew,
        digits = 8,
        width = 12,
        format = "f"
      ),
      "change: ",
      formatC (
        cnew,
        digits = 8,
        width = 12,
```

```
          format = "f"
        ),
        "rate: ",
        formatC (
          rate,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "\n"
      )
    return (list (
      itel = itel,
      xinit = xinit,
      xfinal = xnew,
      change = cnew,
      rate = rate
    ))
  }


cobwebPlotter <-
  function (xold,
            func,
            lowx = 0,
            hghx = 1,
            lowy = lowx,
            hghy = hghx,
            eps = 1e-10,
            itmax = 25,
            ...) {
    x <- seq (lowx, hghx, length = 100)
    y <- sapply (x, function (x)
      func (x, ...))
    plot (
      x,
      y,
      xlim = c(lowx , hghx),
      ylim = c(lowy, hghy),
      type = "l",
      col = "RED",
      lwd = 2
    )
    abline (0, 1, col = "BLUE")
```

```
    base <- 0
    itel <- 1
    repeat {
      xnew <- func (xold, ...)
      if (itel > 1) {
        lines (matrix(c(xold, xold, base, xnew), 2, 2))
      }
      lines (matrix(c(xold, xnew, xnew, xnew), 2, 2))
      if ((abs (xnew - xold) < eps) || (itel == itmax)) {
        break ()
      }
      base <- xnew
      xold <- xnew
      itel <- itel + 1
    }
  }
```

5.2 minimax.R

```
minQuadratic <- function (y, f, g, k, lw, up) {
  func <- function (x, f, g, k) {
    return (f + g * (x - y) + 0.5 * k * (x - y) ^ 2)
  }
  fup <- func (up, f, g, k)
  flw <- func (lw, f, g, k)
  if (k <= 0) {
    if (fup <= flw)
      return (list (x = up, f = fup))
    if (fup >= flw)
      return (list (x = lw, f = flw))
  }
  xmn <- y - g / k
  fmn <- func (xmn, f, g, k)
  if (xmn >= up)
    return (list (x = up, f = fup))
  if (xmn <= lw)
    return (list (x = lw, f = flw))
  return (list (x =  xmn, f = fmn))
}

minimaxQ <- function (y, f, g, k, lw, up) {
  n <- length (f)
  res <- Inf
  for (i in 1:n) {
```

```
    xlw <- lw
    xup <- up
    fail <- FALSE
    for (j in 1:n) {
      if (j == i)
        next
      df <- f[i] - f[j]
      dg <- g[i] - g[j]
      if (dg > 0)
        xlw <- max (xlw, y - df / dg)
      if (dg < 0)
        xup <- min (xup, y - df / dg)
      if ((dg == 0) && (f[i] < f[j]))
        fail <- TRUE
    }
    if (xlw >= xup)
      fail <- TRUE
    if (!fail)
      h <- minQuadratic (y, f[i], g[i], k, xlw, xup)
    if (h$f < res) {
      res <- h$f
      sol <- h$x
    }
  }
  return (sol)
}

minimaxB <- function (y, f, g, k, lw, up) {
  n <- length (f)
  for (i in 1:(n - 1)) {
    for (j in (i + 1):n) {
      df <- f[i] - f[j]
      dg <- g[i] - g[j]
      dk <- k[i] - k[j]
      rt <- y + solve (polynomial (c (df, dg, dk / 2)))
      rt <-
        rt[which ((rt <= up) & (rt >= lw) & (!is.complex (rt)))]
    }
  }
  rt <- sort (unique (c(lw, up, rt)))
  m <- length (rt)
  fmin <- Inf
  for (j in 1:(m - 1)) {
    x <- (rt[j] + rt[j + 1]) / 2
```

```
    smax <- -Inf
    for (i in 1:n) {
      si <- f[i] + g[i] * (x - y) + 0.5 * k[i] * (x - y) ^ 2
      if (si > smax) {
        smax <- si
        l <- i
      }
    }
    h <- minQuadratic (y, f[l], g[l], k[l], rt[j], rt[j + 1])
    if (h$f < fmin) {
      fmin <- h$f
      xmin <- h$x
    }
  }
  return (xmin)
}
```

References

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://deleeuwpdx.net/janspubs/1977/chapters/deleeuw_C_77.pdf.

———. 2016. "Majorizing Cubics on Intervals." http://deleeuwpdx.net/pubfolders/cubic/cubic.pdf.

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