Quadratic Programming with Quadratic Constraints

Jan de Leeuw

Version 14, January 03, 2017

Abstract

We give a quick and dirty, but reasonably safe, algorithm for the minimization of a convex quadratic function under convex quadratic constraints. The algorithm minimizes the Lagrangian dual by using a safeguarded Newton method with non-negativity constraints.

Contents

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

1 Introduction

In this note we give an algorithm for minimizing a quadratic f_0 over all x for which $f_s(x) \leq 0$, where the f_s with $s = 1, \cdots, p$ are also quadratics. We assume the f_s are convex quadratics for all $s = 0, \cdots, p$. Thus we have $f_s(x) = c_s + b_s'x + \frac{1}{2}x'A_sx$, with A_s positive semi-definite for all s. In addition we assume, without loss of generality, that A_0 is non-singular (and thus positive definite). Finally, we assue the Slater condition is satisfied, i.e. there is an x such that $f_s(x) < 0$ for $s = 1, \cdots, p$.

Quadratic programming with quadratic constraints (QPQC) has been studied in great detail, both for the convex and the much more complicated non-convex case. The preferred algorithm translates the problem into a second-order cone programming (SOCP) problem, which is then solved by interior point methods. This is the method used in the R packages `DWD` (Huang et al. (2011)), `CLSOCP` (Rudy (2011)), and `cccp` (Pfaff (2015)).

We go a somewhat different route, more direct, but undoubtedly less efficient. In the appendix we collect some general expressions for the first and second derivatives of a marginal function and apply them to the Lagrangian of a constrained optimization problem. In the body of the paper we apply these formulas to the Langrangian dual of the QPQC problem, using a straightforward version of the Newton method with non-negativity constraints.

2 QPQC

Suppose the f_s are convex quadratics for all $s = 0, \cdots, p$. We have $f_s(x) = c_s + b_s'x + \frac{1}{2}x'A_sx$, with A_s positive semi-definite for all s. In addition we assume, without loss of generality, that A_0 is non-singular (and thus positive definite). It follows that the Lagrangian is

$$g(x, y) = (c_0 + \sum_{s=1}^{p} y_s c_s) + (b_0 + \sum_{s=1}^{p} y_s b_s)'x + \frac{1}{2}x' \left\{ A + \sum_{s=1}^{p} y_s A_s \right\} x,$$

and thus

$$x(y) = -\left\{ A_0 + \sum_{s=1}^{p} y_s A_s \right\}^{-1} (b_0 + \sum_{s=1}^{p} y_s b_s), \tag{1}$$

and

$$h(y) = (c_0 + \sum_{s=1}^{p} y_s c_s) - \frac{1}{2}(b_0 + \sum_{s=1}^{p} y_s b_s)' \left\{ A_0 + \sum_{s=1}^{p} y_s A_s \right\}^{-1} (b_0 + \sum_{s=1}^{p} y_s b_s). \tag{2}$$

We could use the explicit formula (2) to compute derivatives of h, but instead we apply the results of the appendix.

In the QPQC case we have, from (9),

$$\mathcal{D}h(y) = F(x(y)) = \begin{bmatrix} f_1(x(y)) \\ \vdots \\ f_p(x(y)) \end{bmatrix}.$$

For the second derivatives we need

$$\mathcal{D}_{11}g(x, y) = A_0 + \sum_{s=1}^{p} y_s A_s,$$

and

$$\mathcal{D}F(x) = \begin{bmatrix} b_1' + x'A_1 \\ \vdots \\ b_p' + x'A_p \end{bmatrix}.$$

Thus, from (10),

$$\{\mathcal{D}^2 h(y)\}_{st} = -(A_s x(y) + b_s)' \left\{ A_0 + \sum_{s=1}^p y_s A_s \right\}^{-1} (A_t x(y) + b_t)$$

We now know how to compute first and second derivatives of the dual function. In step k of the iterative process we use the quadratic Taylor approximation at the current value $y^{(k)}$ of y.

$$h(y) = h(y^{(k)}) + (y - y^{(k)})'\mathcal{D}h(y^{(k)}) + \frac{1}{2}(y - y^{(k)})'\mathcal{D}^2(y^{(k)})(y - y^{(k)})$$

We then maximize this over $y \geq 0$ the find $\overline{y}^{(k)}$, using the `pnnqp` function from the `lsei` package (Wang, Lawson, and Hanson (2015)).

Finally we stabilize Newton's method by computing

$$y^{(k+1)} = \mathop{\mathbf{argmin}}_{0 \leq \lambda \leq 1} \; h(y^{(k)} + \lambda(\overline{y}^{(k)} - y^{(k)})).$$

For this step-size or relaxation computation we use `optimize` from base R. But first we check if $(\overline{y}^{(k)} - y^{(k)})'\mathcal{D}h(\overline{y}^{(k)}) \geq 0$, in which case we choose step size equal to one.

3 Example

We give a small and artificial example. To make sure the Slater condition is satisfied we choose $c_s < 0$ for $s = 1, \cdots, p$, which guarantees that $f_s(0) < 0$. The example has quadratics of three variables and five quadratic constraints.

```
set.seed(54321)
a <- array (0, c(3, 3, 6))
for (j in 1:6) {
  a[, , j] <- crossprod (matrix (rnorm (300), 100, 3)) / 100
}
b <- matrix (rnorm (18), 3, 6)
c <- -abs (rnorm (6))
```

```
print (qpqc (c(0, 0, 0, 0, 0), a, b, c, eps = 1e-10, verbose = TRUE))
```

```
## Iteration:    1 fold:         -Inf fnew:   -2.69125228 step:    1.00000000
## Iteration:    2 fold:   -2.69125228 fnew:   -2.65100837 step:    1.00000000
## Iteration:    3 fold:   -2.65100837 fnew:   -2.65032456 step:    1.00000000
## Iteration:    4 fold:   -2.65032456 fnew:   -2.65032433 step:    1.00000000
## Iteration:    5 fold:   -2.65032433 fnew:   -2.65032433 step:    1.00000000
```

```
## $h
## [1] -2.650324329
##
## $f
## [1] -2.650324329
##
## $xmin
## [1]  0.6424151506 -1.6326182487  0.2190791799
##
## $multipliers
## [1] 0.0000000000 0.0000000000 0.0000000000 0.1040112458 0.0000000000
##
## $constraints
## [1] -7.755080598e-01 -4.529990503e+00 -9.397132870e-01  1.047517628e-11
## [5] -2.269181794e+00
```

4 Appendix: Derivatives

In this appendix we collect some formulas for first and second derivatives of minima and minimizers in constrained problems. It is just convenient to have them in a single place, and we can apply them in the body of the paper.

4.1 Marginal Function

Suppose $g : \mathbb{R}^n \otimes \mathbb{R}^m \rightarrow \mathbb{R}$, and define

$$h(y) := \min_x g(x, y) = g(x(y), y), \tag{3}$$

with

$$x(y) := \mathop{\mathbf{argmin}}_x g(x, y), \tag{4}$$

which we assume to be unique for each y.

Now assume two times continuous differentiability of g. Then $x(y)$ satisfies

$$\mathcal{D}_1 g(x(y), y) = 0,$$

and thus

$$\mathcal{D}_{11} g(x(y), y)\mathcal{D}x(y) + \mathcal{D}_{12} g(x(y), y) = 0.$$

It follows that

$$\mathcal{D}x(y) = -\mathcal{D}_{11}^{-1} g(x(y), y)\mathcal{D}_{12} g(x(y), y). \tag{5}$$

Now

$$\mathcal{D}h(y) = \mathcal{D}_1 g(x(y), y)\mathcal{D}x(y) + \mathcal{D}_2 g(x(y), y) = \mathcal{D}_2 g(x(y), y), \tag{6}$$

and

$$\begin{aligned}\mathcal{D}^2 h(y) &= \mathcal{D}_{12} g(x(y), y)\mathcal{D}x(y) + \mathcal{D}_{22} g(x(y), y) = \\ &= \mathcal{D}_{22} g(x(y), y) - \mathcal{D}_{21} g(x(y), y)\mathcal{D}_{11}^{-1} g(x(y), y)\mathcal{D}_{12} g(x(y), y).\end{aligned} \tag{7}$$

4.2 Lagrangians

We specialize the results of the previous section to Langrangian duals. For the problem of minimizing $f : \mathbb{R}^n \rightarrow \mathbb{R}$ over x such that $f_s(x) \leq 0$ for $s = 1, \cdots, p$, where $f_s : \mathbb{R}^n \rightarrow \mathbb{R}$. We can also write the inequality constraints compactly as $F(x) \leq 0$, where $F : \mathbb{R}^n \otimes \mathbb{R}^m \rightarrow \mathbb{R}$. The Lagrangian is

$$g(x, y) = f_0(x) + y' F(x). \tag{8}$$

The primal problem is

$$\min_x \max_{y \geq 0} g(x, y) = \min_x \begin{cases} f_0(x) & \text{if } F(x) \leq 0, \\ +\infty & \text{otherwise} \end{cases} = \min_x \{f_0(x) \mid F(x) \leq 0\}.$$

The dual problem is

$$\max_{y \geq 0} \min_x g(x, y) = \max_{y \geq 0} h(y).$$

For any feasible x and y we have weak duality $h(y) \leq f_0(x)$. If the f_s for $s = 0, \cdots, p$ are convex and the Slater condition is satisfied we have equal optimal values for the primal and dual problems. Moreover if y solves the dual problen then $x(y)$ solves the primal problem.

Thus, from (6),

$$\mathcal{D}h(y) = \mathcal{D}_2 g(x(y), y) = F(x(y)), \tag{9}$$

and, from (7),

$$\mathcal{D}^2 h(y) = -\mathcal{D}_{21} g(x(y), y)\mathcal{D}_{11}^{-1} g(x(y), y)\mathcal{D}_{12} g(x(y), y).$$

Also

$$\mathcal{D}_{11} g(x(y), y) = \mathcal{D}^2 f_0(x(y)) + \sum_{s=1}^{p} y_s \mathcal{D}^2 f_s(x(y)),$$

and

$$\mathcal{D}_{21} g(x(y), y) = \mathcal{D}F(x(y)),$$

so that

$$\mathcal{D}^2 h(y) = -\mathcal{D}F(x(y)) \left\{ \mathcal{D}^2 f_0(x(y)) + \sum_{s=1}^{p} y_s \mathcal{D}^2 f_s(x(y)) \right\}^{-1} (\mathcal{D}F(x(y)))' \tag{10}$$

5 Appendix: Code

5.1 dual.R

```
library (MASS)
library (lsei)
library (numDeriv)

source("nnnewton.R")
source("qpqc.R")
```

5.2 nnnewton.R

```
nnnewton <-
  function (yold,
            fnewton,
            itmax = 100,
            eps = 1e-10,
            verbose = FALSE,
            ...) {
    itel <- 1
    fold <- -Inf
    hfunc <- function (step, yold, ybar, ...) {
      z <- yold + step * (ybar - yold)
      fval <- fnewton (z, ...)
      return (fval$h)
    }
    repeat {
      fval <- fnewton (yold, ...)
      fnew <- fval$h
      ybar <-
        pnnqp (eps * diag(length (yold))-fval$d2h, -drop(fval$dh - fval$d2h %*% yold))$
      sval <- fnewton (ybar, ...)
      sd <- sum ((ybar - yold) * sval$dh)
      if (sd >= 0) step <- 1
      else {
      step <-
        optimize (
          hfunc,
```

```
        c (0, 1),
        yold = yold,
        ybar = ybar,
        maximum = TRUE,
        ...
      )$maximum
    }
    ynew <- yold + step * (ybar - yold)
    if (verbose)
      cat(
        "Iteration: ",
        formatC (itel, width = 3, format = "d"),
        "fold: ",
        formatC (
          fold,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "fnew: ",
        formatC (
          fval$h,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "step: ",
        formatC (
          step,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "\n"
      )
    if ((itel == itmax) || ((fval$h - fold) < eps))
      break
    itel <- itel + 1
    yold <- ynew
    fold <- fnew
  }
  return (list (
    solution = yold,
    value = fval$h,
```

```
      gradient = fval$dh,
      hessian = fval$d2h,
      itel = itel
    ))
  }
```

5.3 qpqc.R

```
qpqc <-
  function (yold,
            a,
            b,
            c,
            itmax = 100,
            eps = 1e-10,
            analytic = TRUE,
            verbose = FALSE) {
    hfunc <- ifelse (analytic, hfgh, hfghnum)
    hval <-
      nnnewton (
        yold,
        hfunc,
        a = a,
        b = b,
        c = c,
        eps = eps,
        verbose = verbose,
        itmax = itmax
      )
    fval <- primal (hval$solution, a, b, c)
    return (
      list (
        h = hval$value,
        f = fval$value,
        xmin = fval$solution,
        multipliers = hval$solution,
        constraints = hval$gradient
      )
    )
  }

qfunc <- function (x, a, b, c) {
  return (c + sum (b * x) + sum (x * (a %*% x)) / 2)
}
```

```r
hfgh <- function (x, a, b, c) {
  m <- length (x)
  n <- nrow (a[, , 1])
  asum <- a[, , 1]
  bsum <- b [, 1]
  csum <- c[1]
  for (j in 1:m) {
    asum <- asum + x[j] * a[, , j + 1]
    bsum <- bsum + x[j] * b[, j + 1]
    csum <- csum + x[j] * c[j + 1]
  }
  vinv <- solve (asum)
  xmin <- -drop (vinv %*% bsum)
  h <- csum + sum (bsum * xmin) / 2
  dh <- rep (0, m)
  dg <- matrix (0, n, m)
  for (j in 1:m) {
    dh[j] <- qfunc (xmin, a[, , j + 1], b[, j + 1], c[j + 1])
    dg[, j] <- drop (b[, j + 1] + a[, , j + 1] %*% xmin)
  }
  d2h <- -crossprod (dg, vinv %*% dg)
  return (list (
    h = h,
    dh = dh,
    d2h = d2h
  ))
}

hfghnum <- function (x, a, b, c) {
  hfunc <- function (x) {
    return (hfgh (x, a, b, c)$h)
  }
  return (list (h = hfunc (x), dh = grad (hfunc, x), d2h = hessian (hfunc, x)))
}

primal <- function (x, a, b, c) {
  m <- length (x)
  asum <- a[, , 1]
  bsum <- b [, 1]
  csum <- c[1]
  for (j in 1:m) {
    asum <- asum + x[j] * a[, , j + 1]
    bsum <- bsum + x[j] * b[, j + 1]
    csum <- csum + x[j] * c[j + 1]
```

```
  }
  xmin <- -drop (solve (asum, bsum))
  fmin <- qfunc (xmin, a[, , 1], b[, 1], c[1])
  return (list (solution = xmin, value = fmin))
}
```

References

Huang, H., P. Haaland, X. Lu, Y. Liu, and J.S. Marron. 2011. *DWD: DWD implementation based on A IPM SOCP solver*. {https://R-Forge.R-project.org/projects/dwd/}.

Pfaff, B. 2015. *cccp: Cone Constrained Convex Problems*. {https://R-Forge.R-project.org/projects/cccp/}.

Rudy, J. 2011. *CLSOCP: A smoothing Newton method SOCP solver*. {https://CRAN.R-project.org/package=CLSOCP}.

Wang, Y., C.L. Lawson, and R.J. Hanson. 2015. *lsei: Solving Least Squares Problems under Equality/Inequality Constraints*. http://CRAN.R-project.org/package=lsei.