Infeasible Primal-Dual Quadratic Programming with Box Constraints

Jan de Leeuw

Version 08, May 09, 2017

Abstract

This describes a C version of a infeasible primal-dual algorithm for positive definite quadratic programming with box constraints, proposed by Voglis and Lagaris. We also give a straightforward .C() interface for R.

Contents

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

1 Introduction

The problem we address in this paper is minimization of

$$f(x) = \frac{1}{2}x'Cx + d'x \tag{1}$$

over $a \leq x \leq b$ (elementwise). Here C is positive definite, so f is strictly convex. Also we assume without loss of generality that $a < b$. As a special case we can have either $a_i = -\infty$ or $b_i = +\infty$ or both.

The Lagrangian is

$$\mathcal{L}(x, \lambda, \mu) = \frac{1}{2} x'Cx + d'x - \lambda'(x - a) - \mu'(b - x), \tag{2}$$

and the Karush-Kuhn-Tucker (KKT) necessary and sufficient conditions for a minimum are

$$Cx + d - \lambda + \mu = 0, \tag{3}$$
$$\lambda \geq 0, \tag{4}$$
$$\mu \geq 0, \tag{5}$$
$$\lambda'(x - a) = 0, \tag{6}$$
$$\mu'(b - x) = 0, \tag{7}$$
$$a \leq x \leq b. \tag{8}$$

2 Algorithm

A straightforward primal-dual active set algorithm to solve the KKT conditions was suggested, implemented, and tested by Voglis and Lagaris (2004). There is no proof that the algorithm ends in a final number of steps and is not subjected to cycling. Intermediate iterates will be infeasible, and there is no guarantee that the objective function decreases in each step. Nevertheless the method seems to work well in practice. In future work we will implement the more reliable, but much more complicated, feasible active set algorithm of Hungerländer and Rendl (2015).

Suppose in iteration k we have $(x^{(k)}, \lambda^{(k)}, \mu^{(k)})$. Define the index sets

1. Step 1:

$$\begin{aligned}
L^{(k)} &= \{i \mid x_i^{(k)} < a_i \text{ or } (x_i^{(k)} = a_i \text{ and } \lambda_i^{(k)} \geq 0)\} \\
U^{(k)} &= \{i \mid x_i^{(k)} > b_i \text{ or } (x_i^{(k)} = b_i \text{ and } \mu_i^{(k)} \geq 0)\} \\
S^{(k)} &= \{i \mid a_i < x_i^{(k)} < b_i \text{ or } (x_i^{(k)} = a_i \text{ and } \lambda_i^{(k)} < 0) \text{ or } (x_i^{(k)} = b_i \text{ and } \mu_i^{(k)} < 0)\}
\end{aligned}$$

2. Step 2:

$$\begin{aligned}
x_i^{(k+1)} &= a_i \text{ and } \mu_i^{(k+1)} = 0 \text{ for all } i \in L^{(k)}, \\
x_i^{(k+1)} &= b_i \text{ and } \lambda_i^{(k+1)} = 0 \text{ for all } i \in U^{(k)}, \\
\lambda_i^{(k+1)} &= 0 \text{ and } \mu_i^{(k+1)} = 0 \text{ for all } i \in S^{(k)}.
\end{aligned}$$

3. Step 3: Solve

$$Ax^{(k+1)} + b = \lambda^{(k+1)} - \mu^{(k+1)} \tag{9}$$

for the unknowns

$$\begin{aligned} x_i^{(k+1)} &\text{ with } i \in S^{(k)}, \\ \mu_i^{(k+1)} &\text{ with } i \in U^{(k)}, \\ \lambda_i^{(k+1)} &\text{ with } i \in L^{(k)}. \end{aligned}$$

4. Step 4: If $(x^{(k+1)}, \lambda^{(k+1)}, \mu^{(k+1)})$ satisfies the KKT conditions, stop. Otherwise set k equal to $k+1$ and go to step 1.

3 Implementation

Computation is done in C, using conventions which make it easy to call the programs from R, as well as calling them from C routines that have no access to R. So all arguments are passed by reference, and functions do not return any values. Both I/O and memory allocation are the reponsibility of the calling program, which can be either R or C. To move painlessly (well, not quite) from one-based to zero-based vector indexing in C we use the inline function `VINDEX()`, to move from column-major-one-based to row-major-zero-based matrices we use the inline function `MINDEX()`. I hope the compiler gets the inline message.

Most of the computational effort goes into solving (9). As Voglis and Lagaris (2004) show, we find $x_i^{(k+1)}$ with $i \in S^{(k)}$ by solving a positive definite linear system of order equal to the number of elements in $S^{(k)}$. The computation of the dual variables $\lambda^{(k+1)}$ and $\mu^{(k+1)}$ is then a simple matrix multiplication.

To solve the positive definite linear system we use the LAPACK routine `dposv()`. There are many ways to get at `dposv()`. We can get it from the system LAPACK library, which means on OS X from the `vecLib` framework. Or we could use Intel's `mkl` or similar. Instead we have chosen to install a separate LAPACK from HOMEBREW (n.d.) in `/user/local/opt/lapack`, and use the LAPACKE C-interface (LAPACKE (n.d.)). Your mileage may vary.

4 Code

4.1 R code

```
dyn.load("boxqp.so")

boxcqpR <- function(a, b, lower, upper, itmax=10) {
```

```
  n <- nrow (a)
  x <- -solve(a, b)
  lambda <- rep(0, n)
  mu <- rep(0, n)
  amat <- matrix (0, n, n)
  bvec <- rep (0, n)
  ind <- rep(0, n)
  h <-
    .C(
      "boxcqp",
      as.double (a),
      as.double (b),
      x = as.double (x),
      lambda = as.double (lambda),
      mu = as.double (mu),
      as.double (lower),
      as.double (upper),
      as.double (amat),
      as.double (bvec),
      as.integer (n),
      as.integer (ind),
      itel = as.integer (0),
      f = as.double(0),
      as.integer (itmax)
    )
  return (list (x = h$x, f = h$f, itel = h$itel, lambda = h$lambda, mu = h$mu))
}
```

4.2 C code

```
#include <lapacke.h>
#include <stdbool.h>

inline int VINDEX(const int);
inline int MINDEX(const int, const int, const int);

inline int VINDEX(const int i) { return i - 1; }

inline int MINDEX(const int i, const int j, const int n) {
    return (i - 1) + (j - 1) * n;
}
```

```
void boxcqp(const double *, const double *, double *, double *, double *,
            const double *, const double *, double *, double *, const int *,
            int *, int *, double *, const int *);
void in(const double *, const double *, const double *, const double *,
        const double *, const int *, int *, int *);
void sb(double *x, double *lambda, double *mu, const double *lower,
        const double *upper, const int *ind, const int *n);
void ss(const double *, const double *, double *, double *, double *,
        const double *, const double *, double *, double *, const int *,
        const int *, const int *);
void co(const double *, const double *, const double *, const double *,
        const double *, const int *, const int *, bool *);
void ff(const double *, const double *, const double *, const int *, double *);
void dposv(const int *, const int *, double *, double *, int *);
void mprint(const double *, const int *, const int *);
void check(const double *, const double *, const double *, const double *,
           const double *, const double *, const double *, const int *,
           const int *);

void dposv(const int *n, const int *nrhs, double *a, double *b, int *status) {
    lapack_int info = 0, nn = (lapack_int)*n, nr = (lapack_int)*nrhs;
    info = LAPACKE_dposv(LAPACK_COL_MAJOR, 'U', nn, nr, a, nn, b, nn);
    *status = info;
    return;
}

void boxcqp(const double *a, const double *b, double *x, double *lambda,
            double *mu, const double *lower, const double *upper, double *amat,
            double *bvec, const int *n, int *ind, int *itel, double *f,
            const int *itmax) {
    int nzero = 0, ktel = 1, isit = 0;
    bool *opt = (bool *)&isit;
    while (true) {
        (void)in(x, lambda, mu, lower, upper, n, ind, &nzero);
        (void)sb(x, lambda, mu, lower, upper, n, ind);
        (void)ss(a, b, x, lambda, mu, lower, upper, amat, bvec, n, ind, &nzero);
        (void)co(x, lambda, mu, lower, upper, n, ind, opt);
        if ((ktel == *itmax) || (*opt == true)) {
            break;
        }
        ktel++;
    }
    *itel = ktel;
    (void)ff(a, b, x, n, f);
```

```
    return;
}

void in(const double *x, const double *lambda, const double *mu,
        const double *lower, const double *upper, const int *n, int *ind,
        int *nzero) {
    int nvar = *n, nz = 0;
    for (int i = 1; i <= nvar; i++) {
        int ii = VINDEX(i);
        if (x[ii] < lower[ii]) {
            ind[ii] = -1;
        }
        if (x[ii] > upper[ii]) {
            ind[ii] = 1;
        }
        if ((x[ii] > lower[ii]) && (x[ii] < upper[ii])) {
            ind[ii] = 0;
            nz++;
        }
        if (x[ii] == lower[ii]) {
            if (lambda[ii] >= 0) {
                ind[ii] = -1;
            } else {
                ind[ii] = 0;
                nz++;
            }
        }
        if (x[ii] == upper[ii]) {
            if (mu[ii] >= 0) {
                ind[ii] = 1;
            } else {
                ind[ii] = 0;
                nz++;
            }
        }
    }
    *nzero = nz;
    return;
}

void sb(double *x, double *lambda, double *mu, const double *lower,
        const double *upper, const int *n, const int *ind) {
    int nvar = *n;
    for (int i = 1; i <= nvar; i++) {
```

```
        int ii = VINDEX(i);
        if (ind[ii] == -1) {
            x[ii] = lower[ii];
            mu[ii] = 0.0;
        }
        if (ind[ii] == 1) {
            x[ii] = upper[ii];
            lambda[ii] = 0.0;
        }
        if (ind[ii] == 0) {
            lambda[ii] = 0.0;
            mu[ii] = 0.0;
        }
    }
    return;
}

void ss(const double *a, const double *b, double *x, double *lambda, double *mu,
        const double *lower, const double *upper, double *amat, double *bvec,
        const int *n, const int *ind, const int *nzero) {
    int nvar = *n, nz = *nzero, status = 0, k = 1, nr = 1;
    for (int i = 1; i <= nvar; i++) {
        int ii = VINDEX(i);
        if (ind[ii] == 0) {
            int kk = VINDEX(k);
            bvec[kk] = -b[ii];
            int l = 1;
            for (int j = 1; j <= nvar; j++) {
                int jj = VINDEX(j);
                double aa = a[MINDEX(i, j, nvar)];
                if (ind[jj] == -1) {
                    bvec[kk] -= aa * lower[jj];
                }
                if (ind[jj] == 1) {
                    bvec[kk] -= aa * upper[jj];
                }
                if (ind[jj] == 0) {
                    amat[MINDEX(k, l, nz)] = aa;
                    l++;
                }
            }
            k++;
        }
    }
```

```
    (void)dposv(nzero, &nr, amat, bvec, &status);
    k = 0;
    for (int i = 1; i <= nvar; i++) {
        int ii = VINDEX(i);
        if (ind[ii] == 0) {
            x[ii] = bvec[k];
            k++;
        }
    }
    for (int i = 1; i <= nvar; i++) {
        int ii = VINDEX(i);
        if (ind[ii] == -1) {
            lambda[ii] = b[ii];
            for (int j = 1; j <= nvar; j++) {
                int jj = VINDEX(j);
                double aa = a[MINDEX(i, j, nvar)];
                lambda[ii] += aa * x[jj];
            }
        }
        if (ind[ii] == 1) {
            mu[ii] = -b[ii];
            for (int j = 1; j <= nvar; j++) {
                int jj = VINDEX(j);
                double aa = a[MINDEX(i, j, nvar)];
                mu[ii] -= aa * x[jj];
            }
        }
    }
    return;
}

void co(const double *x, const double *lambda, const double *mu,
        const double *lower, const double *upper, const int *n, const int *ind,
        bool *opt) {
    int nvar = *n;
    bool isit = true;
    for (int i = 1; i <= nvar; i++) {
        int ii = VINDEX(i);
        if (ind[ii] == 0) {
            if (x[ii] < lower[ii]) {
                isit = false;
            }
            if (x[ii] > upper[ii]) {
                isit = false;
```

```
            }
        }
        if (ind[ii] == 1) {
            if (mu[ii] < 0) {
                isit = false;
            }
        }
        if (ind[ii] == -1) {
            if (lambda[ii] < 0) {
                isit = false;
            }
        }
    }
    *opt = isit;
    return;
}

void ff(const double *a, const double *b, const double *x, const int *n,
        double *f) {
    int nvar = *n;
    double sa = 0.0, sb = 0.0;
    for (int i = 1; i <= nvar; i++) {
        int ii = VINDEX(i);
        sb += b[ii] * x[ii];
        for (int j = 1; j <= nvar; j++) {
            int jj = VINDEX(j);
            double aa = a[MINDEX(i, j, nvar)];
            sa += aa * x[ii] * x[jj];
        }
    }
    *f = sa / 2.0 + sb;
}
```

References

HOMEBREW. n.d. "Homebrew: the Missing Package Manager for MacOS." Accessed 2017. https://brew.sh.

Hungerländer, P., and F. Rendl. 2015. "A Feasible Active Set Method for Strictly Convex Quadratic Problems with Simple Bounds." *SIAM Journal on Optimization* 25 (3): 1633–59.

LAPACKE. n.d. "The LAPACKE C Interface to LAPACK." Accessed 2016. http://www.netlib.org/lapack/lapacke.html.

Voglis, C., and I. E. Lagaris. 2004. "BOXCQP: An Algorithm for Bound Constrained Convex

Quadratic Problems." In *First International Conference "From Scientific Computing to Computational Engineering"*. Athens, Greece.