The sequel of **cccp**: Solving cone constrained convex programs

Bernhard Pfaff
bernhard_pfaff@fra.invesco.com

Invesco Asset Management GmbH
Frankfurt am Main

R in Finance, Chicago IL, 29 and 30 May 2015

Contents

Overview

- Optimization of convex functions.
- Problem formulations:
  1. Linear programs with cone-constraints.
  2. Linear programs with nonlinear-constraints.
  3. Quadratic programs with cone-constraints.
  4. Nonlinear, convex functions with nonlinear and/or with cone-constraints.
- In general:
  All problem formulations are solved by means of interior-point algorithms. A textbook exposition is Boyd and Vandenberghe (2009). The implemention is based on Andersen et al. (2011).
- Design of R package **cccp** for solving cone-constrained convex optimization problems.

Cone Programming
Definition

Convex optimization problems of the form:

$$
\begin{aligned}
\text{minimize} \quad & f_0(x) \\
\text{subject to} \quad & Gx \preceq h \\
& Ax = b
\end{aligned}
$$

- Objective function $f_0(x)$ is convex.
- The linear inequality is a generalized inequality (nota bene: $\preceq$ and not $\leq$) with respect to a proper convex cone.
- Possible cones: componentwise vector inequalities, second-order cone inequalities and/or linear matrix inequalities.

Cone Programming

Cone Constraints

- Constraint(s) with respect to the nonnegative orthant cone:

$$\mathcal{C}_0 = \{u \in \Re^k | u_k \geq 0, k = 1, \ldots, l\}$$

- Constraint(s) with respect to the second-order cone:

$$\mathcal{C}_{k+1} = \{(u_0, u_1) \in \Re \times \Re^{r_k - 1} | u_0 \geq \|u_1\|_2\}, \; k = 0, \ldots, M-1$$

- Constraints(s) with respect to the positive semidefinite cone:

$$\mathcal{C}_{k+M+1} = \{\text{vec}(u) | u \in S_+^{t_k}\}, k = 0, \ldots, N-1$$

- The cone $\mathcal{C}$ is defined as the Cartesian product of the constraints:

$$\mathcal{C} = \mathcal{C}_0 \times \mathcal{C}_1 \times \cdots \times \mathcal{C}_M \times \cdots \times \mathcal{C}_{M+N}$$

Cone Programming

Linear Cone Programs

Primal objective:

$$\begin{aligned} \text{minimize} \quad & q'x \\ \text{subject to} \quad & Gx + s = h \\ & Ax = b \\ & s \succeq 0 \end{aligned}$$

Dual objective:

$$\begin{aligned} \text{maximize} \quad & -h'z - b'y \\ \text{subject to} \quad & G'z + A'y + q = h \\ & z \succeq 0 \end{aligned}$$

- Primal variables x and s; dual variables y and z.
- $s \in \mathcal{C}$ and $z \in \mathcal{C}$, whereby $\mathcal{C}$ defines a Cartesion product of the cones.
- Entails: LPs, SOCPs, SDPs, l1-norm approximations.

Cone Programming

Quadratic Cone Programs

Primal objective:

$$\begin{aligned} \text{minimize} \quad & (1/2)x'Px + q'x \\ \text{subject to} \quad & Gx + s = h \\ & Ax = b \\ & s \succeq 0 \end{aligned}$$

Dual objective:

$$\begin{aligned} \text{maximize} \quad & (1/2)w'Pw - h'z - b'y \\ \text{subject to} \quad & G'z + A'y + q = Pw \\ & z \succeq 0 \end{aligned}$$

additional variable w included in dual formulation.

- Primal variables x and s; dual variables w, y and z.
- $s \in \mathcal{C}$ and $z \in \mathcal{C}$, whereby $\mathcal{C}$ defines a Cartesion product of the cones.
- Entails: QPs, QPs with quadratic constraints, l1-norm regularized least-squares problems.

Cone Programming

Nonlinear Convex Optimization

Primal objective:

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_k(x) \leq 0, \; k = 1, \ldots, m \\ & Gx \preceq h \\ & Ax = b \end{aligned}$$

Epigraph from of primal objective:

$$\begin{aligned} \text{minimize} \quad & t \\ \text{subject to} \quad & f_0(x) \leq t, \\ & f_k(x) \leq 0 \; k = 1, \ldots, m \\ & Gx \preceq h \\ & Ax = b \end{aligned}$$

- Functions f_k are convex and twice differentiable; Problem must be strictly primal and dual feasible.
- Entails: analytic centering (equality constrained or with cone constraints), robust least-squares, GPs, risk parity.

Package

Design

- Importing and linking to **Rcpp** (see Eddelbuettel and François, 2011; Eddelbuettel, 2013) and **RcppArmadillo** (see Eddelbuettel and Sanderson, 2014; Sanderson, 2010).
- Employs concept of Rcpp modules and utilizes S4-classes and methods (see Chambers, 1998).
- Suggested packages: **RUnit** (see Burger et al., 2015) and **numDeriv** (see Gilbert and Varadhan, 2012).
- Endowed with unit testing framework and demo-files (examples primarily taken from `cvxopt` user's guide).

Package I

Main functions

- Solving cone constrained convex programs (wrapper):

```
> args(cccp)

function (P = NULL, q = NULL, A = NULL, b = NULL, cList = list(),
    x0 = NULL, f0 = NULL, g0 = NULL, h0 = NULL, nlfList = list(),
    nlgList = list(), nlhList = list(), optctrl = ctrl())
NULL
```

- Defining cone constrained LPs:

```
> args(dlp)

function (q, A = NULL, b = NULL, cList = list())
NULL
```

- Defining cone constrained QPs::

```
> args(dqp)

function (P, q, A = NULL, b = NULL, cList = list())
NULL
```

Package II

Main functions

- Defining LPs with non-linear and/or cone constraints:

```
> args(dnl)

function (q, A = NULL, b = NULL, cList = list(), x0, nlfList = list(),
    nlgList = list(), nlhList = list())
NULL
```

- Defining CPs with non-linear and/or cone constraints:

```
> args(dcp)

function (x0, f0, g0, h0, cList = list(), nlfList = list(), nlgList = list(),
    nlhList = list(), A = NULL, b = NULL)
NULL
```

Package

Defining constraints

- Nonnegative orthant cone: $Gx \leq h$

```
> args(nnoc)

function (G, h)
NULL
```

- Second-order cone: $||Fx + g||_2 \leq d'x + f$

```
> args(socc)

function (F, g, d, f)
NULL
```

- Positive semidefinite matrix cone: $\sum_{i=1}^{n} x_i F_i \preceq F_0$

```
> args(psdc)

function (Flist, F0)
NULL
```

Package

C++ Design and interface to R

- C++ classes (exposed in module `CPG`: prefix `Rcpp_`):
  `DLP`, `DQP`, `DNL`, `DCP`, `PDV`, `CONEC`, `CTRL` and `CPS`.
- Important members functions of classes for CP definition objects:
  `pobj`, `dobj`, `certp`, `certd`, `rprim`, `rcent`, `rdual` and `cps`.
- Utility functions for operations on cone constraints, *e.g.*, product, inverse, norms and Nesterov-Todd scalings (see Nesterov and Todd, 1997, 1998).
- S4-methods for RC objects:
  `show`, `cps` and getter functions for the solution, control parameters and variables.

Risk Parity

Definition

- Formulation as a convex optimization problem introduced by Spinu (2013).
- Primal objective:

$$\text{minimize} \quad f_0(x) = (1/2)x'Px - \sum_{i=1}^{N} b_i \log(x_i)$$

$$\text{subject to} \quad x \geq 0$$

  The risk contributions are given by b_i (as percentages); the risk metric by P.
- Nota bene: Risk parity solutions are proportional, *i.e.* can be scaled without altering the risk contributions, hence no budget constraint required in problem formulation.

Risk Parity

Derivatives

- The Gradient is given as:

$$\nabla f_0(x) = Px - bx^{-1}$$

- The Hessian is given as:

$$\nabla^2 f_0(x) = P + \mathrm{diag}(bx^{-2})$$

Risk Parity

Example: Loading data, problem definition

```
> library(cccp)
> library(FRAPO)
> data(MultiAsset)
> PData <- timeSeries(MultiAsset, rownames((MultiAsset)))
> RData <- returns(PData, method = "discrete", percentage = TRUE)
> head(RData)

GMT
                GSPC       RUA       GDAXI      FTSE       N225        EEM
2004-12-31  3.245813  3.397233  3.58228308  2.3622215  5.4973813  4.7692308
2005-01-31 -2.529045 -2.752188 -0.44351579  0.7893152 -0.9639003 -0.5384239
2005-02-28  1.890338  2.002846  2.24778782  2.3947406  3.0999535  9.6948819
2005-03-31 -1.911765 -1.831262 -0.03953578 -1.4913958 -0.6102755 -7.8959175
2005-04-29 -2.010859 -2.284403 -3.76957163 -1.8940013 -5.6855587 -1.2664394
2005-05-31  2.995202  3.613527  6.59021611  3.3800529  2.4631764  3.1573754
               DJCBTI      GREXP     BG05.L       GLD
2004-12-31  0.9719796  0.3965631  0.6252880 -2.925532
2005-01-31  0.3760247  0.9315339 -0.2747253 -3.607306
2005-02-28 -1.3036637 -0.3228647 -1.3052604  3.079109
2005-03-31 -0.4402945  0.3468132  0.2990629 -1.608456
2005-04-29  1.9468802  1.6432996  1.4378479  1.237739
2005-05-31  0.9847673  0.7217553  0.2678163 -3.921569

> ## Defining Inputs
> n <- ncol(RData)
> S <- cov(RData)
> x0 <- runif(n)
> rcs <- rep(1 / n, n)
```

Risk Parity I

Example: ERC optimization

```
> ## Solving
> Perc <- rp(x0, S, rcs, optctrl = ctrl(trace = FALSE))
> xerc <- getx(Perc)
> xerc <- xerc / sum(xerc)
> PVarRisk <- drop(crossprod(xerc, S) %*% xerc)
> MPRC <- xerc * S %*% xerc / PVarRisk
> ans <- cbind(xerc, MPRC)
> colnames(ans) <- c("ERC weights", "MPRC")
> round(ans, 4)
```

```
       ERC weights MPRC
GSPC        0.0381  0.1
RUA         0.0367  0.1
GDAXI       0.0352  0.1
FTSE        0.0415  0.1
N225        0.0358  0.1
EEM         0.0218  0.1
DJCBTI      0.1641  0.1
GREXP       0.4198  0.1
BG05.L      0.1588  0.1
GLD         0.0482  0.1
```

Risk Parity I

Example: . . . or from scratch

```
> ## objective
> f0 <- function(x){
+     drop(0.5 * crossprod(x, 2 * S) %*% x - crossprod(rcs, log(x)))
+ }
> ## gradient
> g0 <- function(x) crossprod(2 * S, x) - rcs / x
> ## hessian
> h0 <- function(x){
+     2 * S + diag(rcs / as.vector(x)^2)
+ }
> ## non-negativity constraints
> G = -diag(n)
> h = matrix(0, nrow = n, ncol = 1)
> nno1 <- nnoc(G = G, h = h)
> ## alternatives
> P2 <- cccp(cList = list(nno1),
+            x0 = x0, f0 = f0, g0 = g0, h0 = h0,
+            optctrl = ctrl(trace = FALSE))
> P3D <- dcp(x0, f0, g0, h0, cList = list(nno1))
> P3 <- cps(P3D, ctrl(trace = FALSE))
> ## checking solutions
> all.equal(getstate(Perc), getstate(P2))

[1] TRUE

> all.equal(getstate(P2), getstate(P3))

[1] TRUE
```

Risk Parity I

Example: GRC optimization

```
> rcs <- c(0.2, 0.05, 0.05, rep(0.7 / (n - 3), n - 3))
> Pgrc <- rp(x0, S, rcs, optctrl = ctrl(trace = FALSE))
> xgrc <- getx(Pgrc)
> xgrc <- xgrc / sum(xgrc)
> PVarRisk <- drop(crossprod(xgrc, S) %*% xgrc)
> MPRC <- xgrc * S %*% xgrc / PVarRisk
> ans <- cbind(xgrc, MPRC)
> colnames(ans) <- c("GRC weights", "MPRC")
> round(ans, 4)
```

```
       GRC weights MPRC
GSPC        0.0757 0.20
RUA         0.0183 0.05
GDAXI       0.0182 0.05
FTSE        0.0419 0.10
N225        0.0364 0.10
EEM         0.0219 0.10
DJCBTI      0.1625 0.10
GREXP       0.4170 0.10
BG05.L      0.1606 0.10
GLD         0.0476 0.10
```

Outlook

- Further development:
  - Exploit sparsity of problem formulations.
  - Allow provision of custom-solver routines.
- Hosted on:
  - https://github.com/bpfaff/cccp (main development)
  - https://r-forge.r-project.org/projects/cccp/
  - http://cran.r-project.org/web/packages/cccp/index.html
    Listed in Task View 'Optimization', currently sole R package for CP.

Bibliography I

Andersen, M., J. Dahl, Z. Liu, and L. Vandenberghe (2011, September). *Optimization for Machine Learning*, Chapter Interior-point methods for large-scale cone programming, pp. 1–26. Cambridge, MA: MIT Press.

Boyd, S. and L. Vandenberghe (2009). *Convex Optimization* (seventh printing with corrections ed.). Chichester, UK: Cambridge University Press.

Burger, M., K. Jünemann, and T. König (2015). *RUnit: R Unit test framework*. R package version 0.4.28.

Chambers, J. M. (1998). *Programming with Data*. Springer.

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*. New York: Springer. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and R. François (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software 40*(8), 1–18.

Eddelbuettel, D. and C. Sanderson (2014, March). Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis 71*, 1054–1063.

Bibliography II

Gilbert, P. and R. Varadhan (2012). *numDeriv: Accurate Numerical Derivatives*. R package version 2012.9-1.

Nesterov, Y. and M. Todd (1997). Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research 22*(1), 1–42.

Nesterov, Y. and M. Todd (1998, May). Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on Optimization 8*(2), 324–364.

Sanderson, C. (2010, September). Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA. revised December 2011.

Spinu, F. (2013, July). An algorithm for computing risk parity weights. SSRN. OMERS Capital Markets.