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
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.
Convex optimization problems of the form:

minimize \( f_0(x) \)
subject to \( Gx \preceq h \)
\( Ax = b \)

- 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:
  \[ C_0 = \{ u \in \mathbb{R}^k | u_k \geq 0, \ k = 1, \ldots, l \} \]

- Constraint(s) with respect to the second-order cone:
  \[ C_{k+1} = \{ (u_0, u_1) \in \mathbb{R} \times \mathbb{R}^{rk-1} | u_0 \geq \|u_1\|_2 \}, \ k = 0, \ldots, M - 1 \]

- Constraints(s) with respect to the positive semidefinite cone:
  \[ C_{k+M+1} = \{ \text{vec}(u) | u \in S^t_k \}, \ k = 0, \ldots, N - 1 \]

- The cone \( C \) is defined as the Cartesian product of the constraints:
  \[ C = C_0 \times C_1 \times \cdots \times C_M \times \cdots \times C_{M+N} \]
Primal objective:

\[
\begin{align*}
\text{minimize} & \quad q'x \\
\text{subject to} & \quad Gx + s = h \\
& \quad Ax = b \\
& \quad s \succeq 0
\end{align*}
\]

Dual objective:

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

- Primal variables \( x \) and \( s \); dual variables \( y \) and \( z \).
- \( s \in C \) and \( z \in C \), whereby \( C \) defines a Cartesian product of the cones.
- Entails: LPs, SOCPs, SDPs, l1-norm approximations.
Cone Programming

Quadratic Cone Programs

Primal objective:

\[
\text{minimize} \quad \frac{1}{2} x' P x + q' x \\
\text{subject to} \quad G x + s = h \\
\quad A x = b \\
\quad s \succeq 0
\]

Dual objective:

\[
\text{maximize} \quad \frac{1}{2} w' P w - h' z - b' y \\
\text{subject to} \quad G' z + A' y + q = P w \\
\quad z \succeq 0 \\
\quad \text{additional variable } w \text{ included in dual formulation.}
\]

- Primal variables \( x \) and \( s \); dual variables \( w, y \) and \( z \).
- \( s \in C \) and \( z \in C \), whereby \( C \) defines a Cartesian product of the cones.
- Entails: QPs, QPs with quadratic constraints, l1-norm regularized least-squares problems.
Cone Programming
Nonlinear Convex Optimization

Primal objective:

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

Epigraph from of primal objective:

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

- 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.
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):**
  ```R
  > 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:**
  ```R
  > args(dlp)
  function (q, A = NULL, b = NULL, cList = list())
  NULL
  ```

- **Defining cone constrained QPs:**
  ```R
  > 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:
  ```r
  > 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:
  ```r
  > args(dcp)
  function (x0, f0, g0, h0, cList = list(), nlfList = list(), nlgList = list(),
           nlhList = list(), A = NULL, b = NULL)
  NULL
  ```
Nonnegative orthant cone: $Gx \leq h$

> `args(nnoc)`
```r
function (G, h)
NULL
```

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

> `args(socc)`
```r
function (F, g, d, f)
NULL
```

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

> `args(psdc)`
```r
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:

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

  subject to \( x \geq 0 \)

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.
The Gradient is given as:

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

The Hessian is given as:

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

Example: Loading data, problem definition

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

<table>
<thead>
<tr>
<th>GMT</th>
<th>GSPC</th>
<th>RUA</th>
<th>GDAXI</th>
<th>FTSE</th>
<th>N225</th>
<th>EEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004-12-31</td>
<td>3.245813</td>
<td>3.397233</td>
<td>3.582283</td>
<td>2.3622215</td>
<td>5.4973813</td>
<td>4.7692308</td>
</tr>
<tr>
<td>2005-01-31</td>
<td>-2.529045</td>
<td>-2.752188</td>
<td>-0.4435157</td>
<td>0.7893152</td>
<td>-0.9639003</td>
<td>-0.5384239</td>
</tr>
<tr>
<td>2005-02-28</td>
<td>1.890338</td>
<td>2.002846</td>
<td>2.4778782</td>
<td>2.3947406</td>
<td>3.0999535</td>
<td>9.6948819</td>
</tr>
<tr>
<td>2005-03-31</td>
<td>-1.911765</td>
<td>-1.831262</td>
<td>-0.03953578</td>
<td>-1.4913958</td>
<td>-0.6102755</td>
<td>-7.8959175</td>
</tr>
<tr>
<td>2005-04-29</td>
<td>-2.010859</td>
<td>-2.284403</td>
<td>-3.76957163</td>
<td>-1.8940013</td>
<td>-5.6855587</td>
<td>-1.2664394</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GMT</th>
<th>DJCBTI</th>
<th>GREXP</th>
<th>BG05.L</th>
<th>GLD</th>
</tr>
</thead>
<tbody>
<tr>
<td>2004-12-31</td>
<td>0.9719796</td>
<td>0.3965631</td>
<td>0.6252880</td>
<td>-2.925532</td>
</tr>
<tr>
<td>2005-01-31</td>
<td>0.3760247</td>
<td>0.9315339</td>
<td>-0.2747253</td>
<td>-3.60736</td>
</tr>
<tr>
<td>2005-02-28</td>
<td>-1.3036637</td>
<td>-0.3228647</td>
<td>-1.3052604</td>
<td>3.079109</td>
</tr>
<tr>
<td>2005-03-31</td>
<td>-0.4402945</td>
<td>0.3468132</td>
<td>0.2990629</td>
<td>-1.608456</td>
</tr>
<tr>
<td>2005-04-29</td>
<td>1.9468802</td>
<td>1.6432996</td>
<td>1.4378479</td>
<td>1.237739</td>
</tr>
<tr>
<td>2005-05-31</td>
<td>0.9847673</td>
<td>0.7217553</td>
<td>0.2678163</td>
<td>-3.921569</td>
</tr>
</tbody>
</table>

```r
> ## Defining Inputs
> n <- ncol(RData)
> S <- cov(RData)
> x0 <- runif(n)
> rcs <- rep(1 / n, n)
```
Risk Parity I
Example: ERC optimization

```r
> ## 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)

<table>
<thead>
<tr>
<th>ERC weights</th>
<th>MPRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSPC</td>
<td>0.0381</td>
</tr>
<tr>
<td>RUA</td>
<td>0.0367</td>
</tr>
<tr>
<td>GDAXI</td>
<td>0.0352</td>
</tr>
<tr>
<td>FTSE</td>
<td>0.0415</td>
</tr>
<tr>
<td>N225</td>
<td>0.0358</td>
</tr>
<tr>
<td>EEM</td>
<td>0.0218</td>
</tr>
<tr>
<td>DJCBTI</td>
<td>0.1641</td>
</tr>
<tr>
<td>GREXP</td>
<td>0.4198</td>
</tr>
<tr>
<td>BG05.L</td>
<td>0.1588</td>
</tr>
<tr>
<td>GLD</td>
<td>0.0482</td>
</tr>
</tbody>
</table>
```

Pfaff (Invesco)
R package cccp
RFinance 2015
Risk Parity I
Example: ...or from scratch

```r
> ## 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

```r
> 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)

<table>
<thead>
<tr>
<th>GRC weights</th>
<th>MPRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSPC</td>
<td>0.0757</td>
</tr>
<tr>
<td>RUA</td>
<td>0.0183</td>
</tr>
<tr>
<td>GDAXI</td>
<td>0.0182</td>
</tr>
<tr>
<td>FTSE</td>
<td>0.0419</td>
</tr>
<tr>
<td>N225</td>
<td>0.0364</td>
</tr>
<tr>
<td>EEM</td>
<td>0.0219</td>
</tr>
<tr>
<td>DJCBTI</td>
<td>0.1625</td>
</tr>
<tr>
<td>GREXP</td>
<td>0.4170</td>
</tr>
<tr>
<td>BG05.L</td>
<td>0.1606</td>
</tr>
<tr>
<td>GLD</td>
<td>0.0476</td>
</tr>
</tbody>
</table>
```

Pfaff (Invesco)
R package cccp
RFinance 2015
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.


