Journal of Statistical Software

November 2014, Volume 61, Issue 12. http://www.jstatsoft.org/

# coneproj: An R Package for the Primal or Dual Cone Projections with Routines for Constrained Regression

**Xiyue Liao**
Colorado State University

**Mary C. Meyer**
Colorado State University

**Abstract**

The **coneproj** package contains routines for cone projection and quadratic programming, plus applications in estimation and inference for constrained parametric regression and shape-restricted regression problems. A short routine `check_irred` is included to check the irreducibility of a matrix, whose rows are supposed to be a set of cone edges used by `coneA` or `coneB`. For the `coneA` and `coneB` functions, the vector to project is provided by the user, along with the cone specification and a weight vector. For `coneA`, a constraint matrix is specified to define the cone, and for `coneB`, the cone edges are provided. The `coneA` and `coneB` algorithms have been coded and compiled in C++, and are called by R. The `qprog` function transforms a quadratic programming problem into a cone projection problem and calls `coneA`. The `constreg` function does estimation and inference for parametric least-squares regression with constraints on the parameters (using `coneA`). A $p$ value for the "one-sided" test is provided. The `shapereg` function uses `coneB` to provide a least-squares estimator for a regression function with several choices of constraints including isotonic and convex regression functions, as well as estimates of parametrically modeled covariate effects. Results from hypothesis tests for significance of the effects are also provided. This package is now available from the Comprehensive R Archive Network at http://CRAN.R-project.org/package=coneproj.

*Keywords*: cone projection, isotonic regression, quadratic programming, partial linear, R.

## 1. Introduction and overview

The projection of $\boldsymbol{y} \in \mathbb{R}^n$ onto a set $\mathcal{C} \subseteq \mathbb{R}^n$ is defined as the point $\hat{\boldsymbol{\theta}} \in \mathcal{C}$ that minimizes the Euclidean distance

$$\| \boldsymbol{y} - \boldsymbol{\theta} \|^2 = \sum_{i=1}^{n} (y_i - \theta_i)^2.$$

A unique minimum exists if $\mathcal{C}$ is closed and convex. We are concerned with projecting onto convex polyhedral cones expressed as

$$\mathcal{C} = \{\boldsymbol{\theta} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{\theta} \geq \mathbf{0}\}, \tag{1}$$

for an $m \times n$ constraint matrix $\boldsymbol{A}$. The set $\mathcal{C}$ is a cone because given $\boldsymbol{\theta} \in \mathcal{C}$, we have $\alpha\boldsymbol{\theta} \in \mathcal{C}$, for all non-negative real numbers $\alpha$, and it is straightforward to verify that $\mathcal{C}$ is convex. We require that $\boldsymbol{A}$ be "irreducible" as defined by Meyer (1999); the intuitive meaning is "non-redundant." The term "polyhedral" means finitely generated, so that points in the cone can be characterized as linear combinations of a finite set of points (generators) where the coefficients of the linear combination are non-negative.

The function `coneA` will return the projection given the vector $\boldsymbol{y} \in \mathbb{R}^n$ and the $m \times n$ matrix $\boldsymbol{A}$. The function `coneB` will return the projection given $\boldsymbol{y}$ and the generators of $\mathcal{C}$. In addition, if a positive weight vector $\boldsymbol{w}$ is provided, the functions return the minimizer of $\sum_{i=1}^n w_i(y_i - \theta_i)^2$ over $\mathcal{C}$.

Cone projection is a special case of quadratic programming, which is concerned with minimizing

$$\boldsymbol{\theta}^\top \boldsymbol{Q}\boldsymbol{\theta} - 2\boldsymbol{c}^\top\boldsymbol{\theta}$$

over $\mathcal{C}$. If $\boldsymbol{Q}$ is positive-definite there is a unique minimum. The function `qprog` returns this minimum, given $\boldsymbol{Q}$, $\boldsymbol{c}$, and a constraint matrix $\boldsymbol{A}$.

Constrained parametric regression is an application of quadratic programming. Given an $n \times k$ design matrix $\boldsymbol{X}$, a data vector $\boldsymbol{y} \in \mathbb{R}^n$ and the regression model

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{2}$$

where the errors $\boldsymbol{\varepsilon}$ are assumed to be mean zero, and identically distributed, the least-squares estimator $\hat{\boldsymbol{\beta}} \in \mathbb{R}^k$, is found by projecting $\boldsymbol{y}$ onto the linear space spanned by the columns of $\boldsymbol{X}$. However if it is known *a priori* that $\boldsymbol{A}\boldsymbol{\beta} \geq 0$ for a given $m \times k$ matrix $\boldsymbol{A}$, the constrained least-squares estimator can be found using a quadratic programming routine with $\boldsymbol{Q} = \boldsymbol{X}^\top\boldsymbol{X}$. The function `constreg` provides the constrained estimate of $\boldsymbol{\beta}$, and in addition tests $H_0 : \boldsymbol{\beta} \in V$ versus $H_1 : \boldsymbol{\beta} \in \mathcal{C}$ , where $V$ is the null space of $\boldsymbol{A}$ and $\mathcal{C}$ is the cone defined by $\boldsymbol{A}$. The one-sided test was shown to have more power than the two-sided test in Meyer and Wang (2012). For derivation of the test, see Raubertas, Lee, and Nordheim (1986).

The semi-parametric regression model

$$y_i = f(t_i) + \boldsymbol{z_i}^\top\boldsymbol{\alpha} + \varepsilon_i \tag{3}$$

is fit using the function `shapereg`. The function $f$ can be assumed to be either increasing or decreasing, convex or concave, or any combination of monotonicity and convexity. The standard errors for $\hat{\boldsymbol{\alpha}}$ are provided with approximate $p$ values for the (two-sided) tests for significance. A hypothesis test for $H_0 : f$ is constant is performed if the shape assumption includes monotonicity, and $H_0 : f$ is linear is tested if the shape assumption is either convex or concave.

The methods are discussed in detail in the next section. In Section 3 a user's guide for the package **coneproj** (Meyer and Liao 2014) in R (R Core Team 2014) is provided, and the data sets in the package are analyzed and discussed. The discussion in Section 4 compares our

routines with some existing R packages, and presents some simulations showing how these packages can be useful in practice.

## 2. Cone projection routines in this package

The cone projection algorithm of Meyer (2013b) forms the main subroutine of the package. Previous methods include interior point methods (see Fang and Puthenpura 1993 for further details), which can be used more generally for projecting onto a convex set. The methods of Goldfarb and Idnani (1983) and Fraser and Massam (1989) use primal-dual constraint ideas. Silvapulle and Sen (2005, Chapter 3) provide a thorough treatment of optimization problems involving convex cones. The algorithm for `coneA` and `coneB`, which are the fundamental part of all routines in this package, can be summarized in three simple steps.

### 2.1. The basic cone projection: `coneA`, `coneB` and `check_irred`

For the `coneA` routine, the user specifies $\boldsymbol{y} \in \mathbb{R}^n$, an $m \times n$ matrix $\boldsymbol{A}$, and an optional weight vector $\boldsymbol{w}$ with positive elements. The routine returns $\hat{\boldsymbol{\theta}}$ to minimize

$$\sum_{i=1}^{n} w_i (y_i - \theta_i)^2$$

over $\boldsymbol{\theta} \in \mathcal{C}$, where $\mathcal{C}$ is defined in (1). The matrix $\boldsymbol{A}$ is required to be irreducible, that is, the rows of $\boldsymbol{A}$ form an irreducible set. The rows also form the "primal basis" for a cone projection problem as defined in Fraser and Massam (1989). A set of vectors is irreducible if none can be written as a positive linear combination of two or more of the others, and the origin is not a positive linear combination of two or more vectors in the set. (The phrase "positive linear combination" means a linear combination with positive coefficients.)

Let $V$ be the null space of $\boldsymbol{A}$; that is, the linear space orthogonal to the space spanned by the rows of $\boldsymbol{A}$. The space $V$ is contained in $\mathcal{C}$. An element in $\mathcal{C}$ can be written as the sum of a vector in $V$ and a linear combination of the *edges* or *generators* of $\mathcal{C}$ with non-negative coefficients. If $\boldsymbol{A}$ has full row rank, it is shown in Meyer (2013b) that the edges $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_m$ of the cone are the columns of $\boldsymbol{\Delta} = \boldsymbol{A}^\top (\boldsymbol{A}\boldsymbol{A}^\top)^{-1}$. If $\boldsymbol{A}$ does not have full row rank, Proposition 1 of Meyer (1999) can be used to obtain the edges $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$, where $M \geq m$. The cone (1) can alternatively be written as

$$\mathcal{C} = \left\{ \boldsymbol{\theta} \in \mathbb{R}^n : \boldsymbol{\theta} = \boldsymbol{v} + \sum_{j=1}^{M} b_j \boldsymbol{\delta}_j, \quad \boldsymbol{v} \in V \quad \text{and} \quad b_1, \ldots, b_M \geq 0 \right\}, \tag{4}$$

where $M$ is the number of generators and $M = m$ if $\boldsymbol{A}$ has full row rank. The generators $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$ and the basis for $V$ form the "dual basis" for a cone projection problem as defined in Fraser and Massam (1989). The generators $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$ are orthogonal to $V$, so the projection of $\boldsymbol{Y}$ onto $\mathcal{C}$ is the sum of the projections onto $V$ and onto the cone

$$\Omega = \left\{ \boldsymbol{\theta} \in \mathbb{R}^n : \boldsymbol{\theta} = \sum_{j=1}^{M} b_j \boldsymbol{\delta}_j, \quad b_1, \ldots, b_M \geq 0 \right\}.$$

The algorithm of Meyer (2013b) provides the projection onto $\Omega$ by determining the *face* of the cone on which the projection lands. The faces are used for the inference methods and are indexed by subsets of $\{1, \ldots, M\}$. For such a subset $J$, the corresponding face is

$$\mathcal{F}_J = \left\{ \boldsymbol{\theta} \in \mathbb{R}^n : \boldsymbol{\theta} = \sum_{j \in J} b_j \boldsymbol{\delta}_j, \;\; b_j > 0 \;\text{ for }\; j \in J \right\}.$$

The faces cover the cone; once the face containing the projection is determined, the projection onto $\Omega$ is simply the projection onto the linear space spanned by the edges making up the face. For more details and proofs, see Meyer (1999). The algorithm finds the projection by determining the set $J$.

The initial guess $J_0$ can be any subset of $\{1, \ldots, M\}$ for which the corresponding $\boldsymbol{\delta}_j$, $j \in J$, form a linearly independent set. At the $k$th iteration,

1. Project $\boldsymbol{y}$ onto the linear space spanned by $\{\boldsymbol{\delta}_j, j \in J_k\}$, to get $\boldsymbol{\theta}^{(k)} = \sum_{j \in J_k} b_j^{(k)} \boldsymbol{\delta}_j$.
2. Check to see if all $b_j^{(k)}$ are non-negative:
   - If yes, go to Step 3.
   - If no, choose $j$ for which $b_j^{(k)}$ is minimized, and remove it from $J$; go to Step 1.
3. Compute $\langle \boldsymbol{y} - \boldsymbol{\theta}^k, \boldsymbol{\delta}_j \rangle$ for each $j \notin J_k$. If these are all non-positive, then stop. If not, choose $j$ for which this inner product is largest, add it to the set, and go to Step 1.

See Meyer (2013b) for the proof of convergence.

The *polar cone* is defined as

$$\Omega^o = \{\boldsymbol{\rho} : \langle \boldsymbol{\theta}, \boldsymbol{\rho} \rangle \leq 0, \; \forall \, \boldsymbol{\theta} \in \mathcal{C}\},$$

and it can be shown that the projection $\hat{\boldsymbol{\rho}}$ of $\boldsymbol{y}$ onto $\Omega^o$ is $\boldsymbol{y} - \hat{\boldsymbol{\theta}}$, i.e., the residual of the projection onto $\mathcal{C}$.

The constraint cone edges $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$ are not needed if $\boldsymbol{A}$ is provided, because the rows of $-\boldsymbol{A}$ are the edges of the polar cone. See Meyer (1999) for a proof.

The function `coneA` requires the specification of the constraint matrix (and hence the polar cone edges), while the function `coneB` requires the user to specify the cone edges $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$ and a basis for the linear space $V$ that is contained in the cone. When there is no linear space in the cone, the user only needs to provide $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$. In either case, the function returns the projection, the dimension of the face of the cone on which the projection lands (which may be used as a surrogate degrees of freedom of the model), and the number of iterations. It also returns a message concerning convergence when the algorithm does not converge; although theoretically the algorithm must converge, the presence of rounding error in the real world results in a small possibility of non-convergence.

The function `check_irred` takes a matrix as its argument, whose rows are supposed to form a set of edges, and checks the irreducibility of the rows. For example, the user can choose to use this routine to check the constraint matrix $\boldsymbol{A}$ required by `coneA` and the cone edges $\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_M$ required by `coneB` in terms of irreducibility before they make a cone projection

by `coneA` or `coneB`. If a row is a positive linear combination of other rows, then it can be removed without affecting the problem, and if there is a positive linear combination of rows of the matrix that equals the zero vector, then there is an implicit equality constraint in the matrix which can be dealt with separately, see Meyer (2013b). In the former case, this routine deletes the redundant rows and returns a set of irreducible rows. In the latter case, this routine will return the original matrix, and leave the equality issue to the user to address. In either case, this routine will return a vector storing the positions of the redundant rows and the number of equality constraints in the rows.

## 2.2. Quadratic programming

Given a positive definite $n \times n$ matrix $\boldsymbol{Q}$ and a constant vector $\boldsymbol{c} \in \mathbb{R}^n$, the object of `qprog` is to find $\hat{\boldsymbol{\theta}} \in \mathbb{R}^n$ to minimize

$$\boldsymbol{\theta}^\top \boldsymbol{Q} \boldsymbol{\theta} - 2\boldsymbol{c}^\top \boldsymbol{\theta} \tag{5}$$

subject to $\boldsymbol{A}\boldsymbol{\theta} \geq \boldsymbol{b}$. We require that the $m \times n$ constraint matrix $\boldsymbol{A}$ be irreducible, and there exists a $\boldsymbol{\theta}_0 \in \mathbb{R}^n$ such that $\boldsymbol{A}\boldsymbol{\theta}_0 = \boldsymbol{b}$.

The unconstrained solution is $\boldsymbol{Q}^{-1}\boldsymbol{c}$. For the constrained solution, we transform it into a cone projection problem as follows. Let $\boldsymbol{U}^\top \boldsymbol{U}$ be the Cholesky decomposition of $\boldsymbol{Q}$, and define $\boldsymbol{\phi} = \boldsymbol{U}(\boldsymbol{\theta} - \boldsymbol{\theta_0})$ and $\boldsymbol{z} = (\boldsymbol{U^{-1}})^\top(\boldsymbol{c} - \boldsymbol{Q}\boldsymbol{\theta_0})$. Then we minimize

$$\| \boldsymbol{z} - \boldsymbol{\phi} \|^2$$

subject to $\tilde{\boldsymbol{A}}\boldsymbol{\phi} \geq \boldsymbol{0}$, where $\tilde{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{U}^{-1}$. The quadratic programming routine returns the projection $\hat{\boldsymbol{\phi}}$, and $\hat{\boldsymbol{\theta}} = \boldsymbol{U}^{-1}\hat{\boldsymbol{\phi}} + \boldsymbol{\theta_0}$. The routine also returns the dimension of the face of the cone on which the projection $\hat{\boldsymbol{\phi}}$ lands, the number of iterations before the algorithm converges, and a message concerning convergence of the algorithm when the algorithm does not converge.

## 2.3. Constrained parametric regression

The least-squares regression model (2) is considered, where the object is to find $\hat{\boldsymbol{\beta}}$ to minimize $\| \boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta} \|^2$, subject to constraints $\boldsymbol{A}\boldsymbol{\beta} \geq \boldsymbol{0}$. We assume that $\boldsymbol{X}$ is an $n \times k$ design matrix which has full column rank, and $\boldsymbol{A}$ is an $m \times k$ irreducible constraint matrix. The constrained estimator $\hat{\boldsymbol{\beta}}$ is found using the quadratic programming routine.

Interest is in testing $H_0 : \boldsymbol{\beta} \in V$ versus $H_1 : \boldsymbol{\beta} \in \mathcal{C}$, where $V$ is the null space of $\boldsymbol{A}$. For example, suppose that in a clinical trials scenario there are three treatments and a placebo, and several covariates. The response is improvement of a condition, and the researchers want to know if any of the treatments is significantly better than the placebo. If $\beta_0$ is the placebo effect, and treatment effects are $\beta_1, \beta_2$, and $\beta_3$, then the first four columns, say, of the design matrix might be dummy variables for the placebo and treatment groups, and the other $k - 4 \geq 0$ columns represent the covariate values. If the researchers assume that any of the treatments is at least as good as the placebo, the constraint matrix (for $k = 7$, say)

$$\boldsymbol{A} = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

may be used. The null hypothesis describes the "no treatment effect" scenario ($\boldsymbol{A\beta} = \boldsymbol{0}$) and the alternative is that at least one treatment has greater effect than the placebo. This "one-sided" test will have greater power than the standard two-sided $F$ test for sub-models.

Defining $SSE_0$ to be the sum of squared residuals for the null hypothesis fit ($\boldsymbol{\beta}$ is in the null space of $\boldsymbol{A}$) and $SSE_1$ to be the sum of squared residuals for the constrained least-squares fit, the test statistic

$$E_{01} = \frac{SSE_0 - SSE_1}{SSE_0}$$

has a mixture-of-betas distribution when $H_0$ is true and $\boldsymbol{\varepsilon} \sim N_n(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ for $\sigma^2 > 0$. Specifically, the null distribution is: for $c > 0$, $d_0 = \dim(V)$, and $m$ is the dimension of the smallest linear space containing $\mathcal{C}$,

$$\mathsf{P}(E_{01} \leq c) = \sum_{d=0}^{m} p_d \mathsf{P}\left[B\left(\frac{d}{2}, \frac{n-d-d_0}{2}\right) \leq c\right],$$

where $B(a, b)$ is a beta random variable and $\mathsf{P}[B(0, b) \leq c] = 1$ if $c > 0$. The mixing parameters $p_d$ correspond to the probabilities under $H_0$ that the projection of $\boldsymbol{y}$ onto $\mathcal{C}$ lands on a face of dimension $d$; these can readily be found through simulations. For more details see Meyer and Wang (2012).

The function `constreg` has inputs $\boldsymbol{y}$, $\boldsymbol{X}$, and $\boldsymbol{A}$. The optional parameter `test` is a logical scalar to determine if the user wants to perform the test or not. If `test = TRUE` is specified, the number of simulations used to obtain the mixing distribution parameters for the test has the default value 10,000, and the test is performed; otherwise the test is skipped. The optional parameter $\boldsymbol{w}$ is a vector of weights. If $\boldsymbol{w}$ is specified, the minimization of $(\boldsymbol{y} - \boldsymbol{X\beta})^\top \boldsymbol{W}(\boldsymbol{y} - \boldsymbol{X\beta})$ over $\mathcal{C}$ is returned, where the diagonal matrix $\boldsymbol{W}$ has elements $\boldsymbol{w}$. The default value of $\boldsymbol{w}$ is a vector of ones.

Returned are the constrained fit, the unconstrained fit, the constrained parameters and the $p$ value for the test if `test = TRUE`.

## 2.4. Shape-restricted regression

The regression model (3) is considered, where the only assumptions about $f$ concern its shape. The `shapereg` function allows eight options, indicated by the value of `shape`: 1 = increasing, 2 = decreasing, 3 = convex, 4 = concave, 5 = increasing-convex, 6 = decreasing-convex, 7 = increasing-concave, and 8 = decreasing-concave. The vector expression for the model is

$$\boldsymbol{y} = \boldsymbol{\theta} + \boldsymbol{Z\alpha} + \boldsymbol{\varepsilon},$$

where the $n \times k$ matrix $\boldsymbol{Z}$ is assumed to be of full column rank and $\theta_i = f(t_i)$. The column space of $\boldsymbol{Z}$ must contain the constant vector, so that if there are no covariates, $\boldsymbol{Z}$ should be a column of ones. The shape restrictions can be written as $\boldsymbol{\theta} \in \mathcal{C}$ where $\mathcal{C}$ is a convex polyhedral cone; see Meyer (2013a) for details and conditions for identifiability of $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$.

The isotonic regression estimator ($f$ is increasing, no covariates) was proposed in Brunk (1958); Robertson, Wright, and Dykstra (1988, Chapter 1) provides details for the least-squares solution. An algorithm for convex regression was proposed by Fraser and Massam (1989). These can both be solved with the cone projection algorithm, and combinations of constraints such as monotone and convex are simple extensions.

For the monotone case, Cheng (2009) proposed a back-fitting algorithm to find $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$; the algorithm for `shapereg` is taken from Meyer (2013a), which employs a single cone projection to find $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$ simultaneously for more general shapes.

The test of constant versus increasing regression function uses the same family of statistics as Raubertas *et al.* (1986), and is discussed in Robertson *et al.* (1988, Chapter 2). The test for linear versus convex regression was derived in Meyer (2003). More generally, the test of $H_0 : \mathsf{E}(\boldsymbol{y}) \in V$ versus $H_1 : \mathsf{E}(\boldsymbol{y}) \in \mathcal{C}$ uses an $E_{01}$ statistic and is exact for *iid* mean-zero normal errors. The optional parameter `test` is a logical scalar to determine if the user wants to perform the test or not. If `test = TRUE` is specified, the number of simulations used to obtain the mixing distribution parameters for the test has the default value 10,000, and the test is performed; otherwise the test is skipped. The performance of the test might be computationally intensive for large sample sizes.

Cheng (2009) argued that $\hat{\boldsymbol{\alpha}}$ is consistent and asymptotically normal. The `shapereg` routine returns the standard errors for $\hat{\boldsymbol{\alpha}}$ as the square roots of the first $k$ diagonal elements of $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\hat{\boldsymbol{\sigma}}^2$, where the columns of $\boldsymbol{X}$ contain the columns of $\boldsymbol{Z}$, plus the edges of the cone indexed by the set $J$ determined by the cone projection. The variance is estimated by $\hat{\boldsymbol{\sigma}}^2 = SSE/(n - 1.5df)$, where $SSE$ is the sum of squared residuals, and the effective degrees of freedom $df$ are returned by `coneB`. For more details about this variance estimator and the motivation for the 1.5 multiplier see Meyer and Woodroofe (2000). To get approximate $p$ values for the significance of $\hat{\boldsymbol{\alpha}}$, a $t$ distribution with the degrees of freedom specified as $1.5df$ is used.

Returned are the constrained parameters, the constrained fit, the linear fit, the approximate standard errors, the approximate $p$ values for the significance of the parameters, the sum of squared residuals for the linear part, the sum of squared residuals for the full model, and the $p$ value for the test $H_0 : \mathsf{E}(\boldsymbol{y}) \in V$ versus $H_1 : \mathsf{E}(\boldsymbol{y}) \in \mathcal{C}$ if `test = TRUE`.

# 3. User guide

We demonstrate the functions using simulated data sets and real data sets. For a more complete demonstration including code for making the plots, see the official reference manual of this package at http://CRAN.R-project.org/package=coneproj.

This package depends on another two packages: **Rcpp** (Eddelbuettel and François 2011; Eddelbuettel 2013) and **RcppArmadillo** (Eddelbuettel and Sanderson 2014), based on which the C++ code is written, and they will be installed along with package **coneproj**:

```
R> install.packages("coneproj")
R> library("coneproj")
```

## 3.1. Constrained regression using `coneA` and `coneB`

We demonstrate the cone projection algorithm by fitting a regression function that is increasing over a range but then decreasing.

First, we create a data set and the corresponding constraint matrix for constrained regression:

```
R> set.seed(123)
R> n <- 50
```

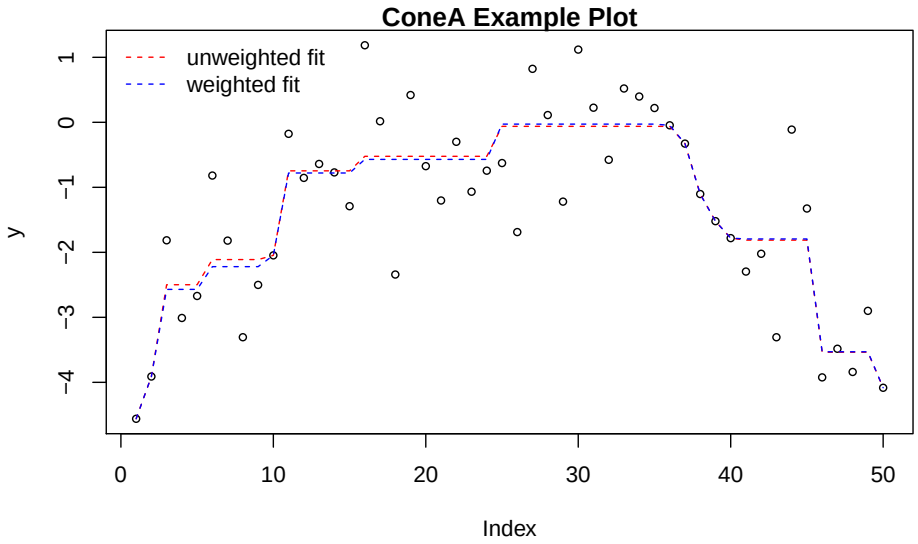


Figure 1: Demonstration of constrained regression using the `coneA` function, with simulated data.

```
R> x <- seq(-2, 2, length = 50)
R> y <- -x^2 + rnorm(n)
R> amat <- matrix(0, n - 1, n)
R> for(i in 1:(n/2 - 1)) {
+    amat[i, i] <- -1
+    amat[i, i + 1] <- 1
+  }
R> for(i in (n/2):(n - 1)) {
+    amat[i, i] <- 1
+    amat[i, i + 1] <- -1
+  }
```

Before we call `coneA`, we can use `check_irred` to check if the rows of the constraint matrix are irreducible or not:

```
R> check <- check_irred(amat)

[1] "edges are irreducible!"
```

Then we call `coneA` to find the non-decreasing vector closest to the first half of $\boldsymbol{y}$ and the non-increasing vector for the rest of $\boldsymbol{y}$, without and with a weight vector $\boldsymbol{w}$:

```
R> ans1 <- coneA(y, amat)
R> ans2 <- coneA(y, amat, w = (1:n)/n)
```

The unweighted and weighted fits are compared in Figure 1. `coneA` returns `df`, `thetahat`, and `steps` as shown:

```
R> names(ans1)
```

Then we call `coneB` to make a constrained regression to $\boldsymbol{y}$, without and with a weight vector $\boldsymbol{w}$:

```
R> ans3 <- coneB(y, delta, vmat)
R> ans4 <- coneB(y, delta, vmat, w = (1:n)/n)
```

The fit to the data and the effective degrees of freedom returned by `coneB` are identical to that of `coneA`, but the number of steps for `coneB` is considerably smaller. This is because the number of edges of the constraint cone used in the fit is smaller than the number of polar cone edges used in the residual vector.

```
R> names(ans3)

[1] "df"          "yhat"        "steps"       "coefs"

R> ans3$df

[1] 15

R> ans3$steps

[1] 14
```

The `coefs` returned by `coneB` are the coefficients of the basis of the linear space contained in the convex cone (4) and the constraint cone edges.

```
R> head(ans3$coefs)

           [,1]
[1,] -1.3533516
[2,]  0.6501649
[3,]  1.4116531
[4,]  0.0000000
[5,]  0.0000000
[6,]  0.3872442
```

To show how the routine `check_irred` works when the rows of a matrix are reducible, we consider a simple example in which $\boldsymbol{x}$ is a sequence of 4 elements and we want to make a constraint matrix such that the 4 elements have an increasing order. We know that we do not need to compare the 1st and the 3rd elements if we already know that the 1st element is smaller than the 2nd element, and the 2nd element is smaller than the 3rd element.

First, we can make a reducible set of edges which makes a total of 6 comparisons:

```
R> amat
     [,1] [,2] [,3] [,4]
[1,]   -1    1    0    0
[2,]   -1    0    1    0
```

```
[3,]   -1    0    0    1
[4,]    0   -1    1    0
[5,]    0   -1    0    1
[6,]    0    0   -1    1
```

Then, we can use `check_irred` to remove the redundant rows. We can tell that the rows of the new matrix returned by `check_irred` are irreducible since this matrix is of full rank, which implies irreducibility.

```
R> check <- check_irred(amat)
R> check$edge
```

```
     [,1] [,2] [,3] [,4]
[1,]   -1    1    0    0
[2,]    0   -1    1    0
[3,]    0    0   -1    1
```

We can also get the positions of the redundant rows in the reducible matrix:

```
R> check$reducible
```

```
[1] 2 3 5
```

Meanwhile, we can check that there is no equality constraint in the rows:

```
R> check$equal
```

```
[1] 0
```

### 3.2. Application of `qprog` to "cubic" data set

To show how `qprog` works, we use a simulated data set "cubic". In this data set, $\boldsymbol{y}$ is a continuous response variable and $\boldsymbol{x}$ is a continuous predictor variable with values in $[0, 2]$. The constraint is that the true regression is increasing, convex and non-negative for $\boldsymbol{x} \in [0, 2]$, so the corresponding constraint matrix is

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 6 \end{pmatrix},$$

which is of full row rank and hence irreducible.

First, we load the data set and extract the predictor $\boldsymbol{x}$ and the response $\boldsymbol{y}$:

```
R> data("cubic", package = "coneproj")
R> x <- cubic$x
R> y <- cubic$y
```

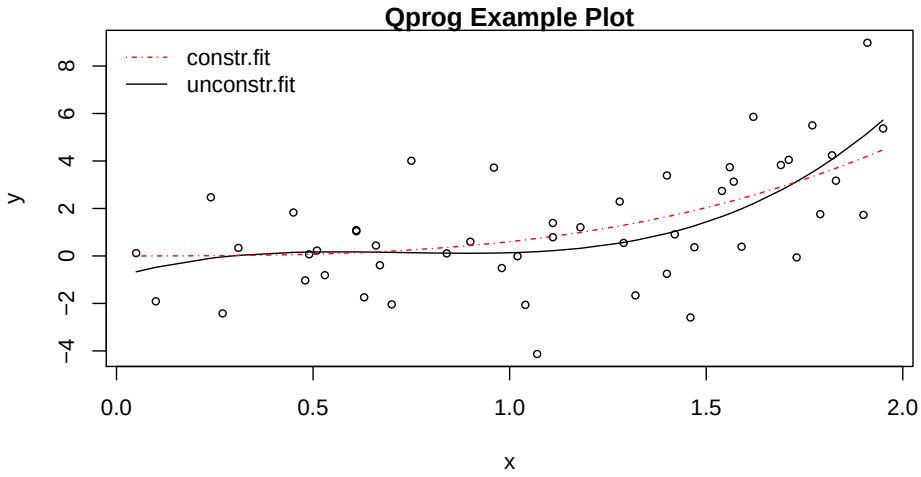


Figure 2: Comparison of an unconstrained cubic least-squares fit (solid) with the least-squares cubic fit, constrained to be positive, increasing, and convex.

Next, we make the design matrix in the "cubic" model:

```
R> xmat <- cbind(1, x, x^2, x^3)
```

Then, we make the $\boldsymbol{Q}$ matrix, the $\boldsymbol{c}$ vector in $\boldsymbol{\theta}^\top \boldsymbol{Q}\boldsymbol{\theta} - 2\boldsymbol{c}^\top\boldsymbol{\theta}$, and the constraint matrix $\boldsymbol{A}$ in the constraint condition $\boldsymbol{A}\boldsymbol{\theta} \geq \boldsymbol{b}$. In this case, $\boldsymbol{b}$ is a zero vector:

```
R> q <- crossprod(xmat)
R> c <- crossprod(xmat, y)
R> amat <- matrix(0, 4, 4)
R> amat[1, 1] <- 1
R> amat[2, 2] <- 1
R> amat[3, 3] <- 1
R> amat[4, 3] <- 1
R> amat[4, 4] <- 6
R> b <- rep(0, 4)
```

We call `qprog` to get the constrained fit and call `lm` to get the unconstrained fit:

```
R> ans <- qprog(q, c, amat, b)
R> betahat <- ans$thetahat
R> fitc <- crossprod(t(xmat), betahat)
R> fitu <- lm(y ~ x + I(x^2) + I(x^3))
```

These fits are shown in Figure 2. We can also check what `qprog` returns:

```
R> names(ans)

[1] "df"          "thetahat"    "steps"

R> ans$df
```

```
[1] 1
```

```
R> ans$thetahat
```

```
              [,1]
[1,] -4.796163e-14
[2,]  1.798561e-13
[3,] -1.614264e-13
[4,]  6.028080e-01
```

```
R> ans$steps
```

```
[1] 3
```

### 3.3. Constrained parametric fit to "FEV" data

To see how `constreg` works, we analyze the "FEV" data set, provided by Kahn (2005). This data set consists of 654 observations on children aged 3 to 19. "FEV" stands for forced expiratory volume, a measure of lung capacity, and is the response variable. "age" and "height" are continuous predictors, while "sex" and "smoke" are categorical predictors. One reasonable assumption is that "FEV" must be non-decreasing in "age", given any value of "height", and similarly "FEV" is non-decreasing in "height", when "age" is fixed. To allow for interaction between the continuous predictors, we fit the model

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{1i} x_{2i} + \beta_4 d_{1i} + \beta_5 d_{2i} + \varepsilon_i,$$

where $x_{1i}$ is the age of the $i$th child, $x_{2i}$ is the height of the $i$th child, $d_{1i}$ is an indicator for "boy," and $d_{2i}$ is an indicator for "smoking." Without loss of generality, we scale "age" and "height" to be within the unit interval (0, 1], and the corresponding constraint matrix is

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix},$$

which is not of full row rank but irreducible.

First, we load this data set, extract the variables and scale "age" and "height" to be within the unit interval:

```
R> data("FEV", package = "coneproj")
R> y <- FEV$FEV
R> age <- FEV$age
R> height <- FEV$height
R> sex <- FEV$sex
R> smoke <- FEV$smoke
R> scale_age <- (age - min(age)) / (max(age) - min(age))
R> scale_height <- (height - min(height)) / (max(height) - min(height))
```

Next, we make the design matrix $\boldsymbol{X}$ which contains two continuous variables, i.e., scaled "age" and scaled "height", which form a warped plane, and two categorical covariates, i.e., "sex" and "smoke":

```
R> xmat <- cbind(1, scale_age, scale_height, scale_age * scale_height,
+    sex, smoke)
```

Then we make the constraint matrix $\boldsymbol{A}$ in the constraint condition $\boldsymbol{A\beta} \geq 0$:

```
R> amat <- matrix(0, 4, 6)
R> amat[1, 2] <- 1
R> amat[2, 2] <- 1
R> amat[2, 4] <- 1
R> amat[3, 3] <- 1
R> amat[4, 3] <- 1
R> amat[4, 4] <- 1
```

After this, we call `constreg` to get constrained coefficient estimates and call `lm` to get unconstrained coefficient estimates:

```
R> ans1 <- constreg(y, xmat, amat)
R> bhat1 <- ans1$coefs
R> ans2 <- lm(y ~ xmat[, -1])
R> bhat2 <- coef(ans2)
```

We can check what the function `constreg` returns:

```
R> names(ans1)

[1] "constr.fit"   "unconstr.fit" "coefs"

R> ans1$coefs

                      [,1]
              9.171990e-01
scale_age    -6.522560e-16
scale_height  2.053492e+00
              2.119287e+00
sex           1.259409e-01
smoke        -1.538562e-01

R> head(ans1$constr.fit)

         [,1]
[1,] 2.036144
[2,] 3.002522
[3,] 1.701419
[4,] 1.755196
[5,] 2.162085
[6,] 2.372075
```

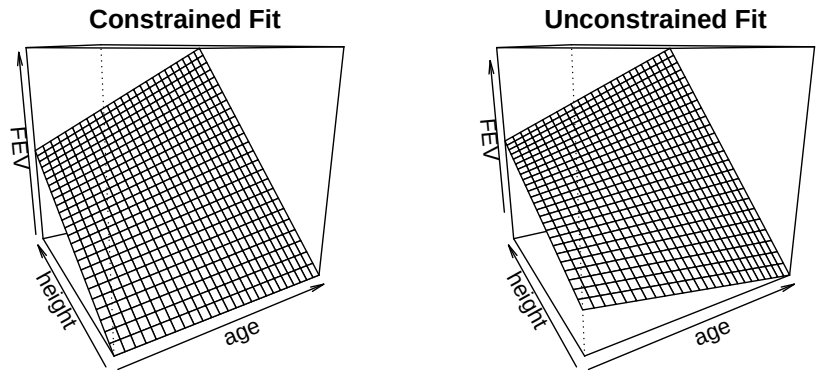


Figure 3: Comparison of constrained and unconstrained fits to the "FEV" data set, using a linear model with interaction.

```
R> head(ans1$unconstr.fit)
```

```
         [,1]
[1,] 2.003506
[2,] 3.051108
[3,] 1.720991
[4,] 1.666875
[5,] 2.103337
[6,] 2.388481
```

If we specify `test = TRUE`, we can get a $p$ value for the $E_{01}$ test in Section 2.3. The null hypothesis is that the regression function depends only on the "smoking" and "sex" variables, so we are not surprised that $H_0$ is rejected.

```
R> ans <- constreg(y, xmat, amat, test = TRUE)
R> names(ans)
```

```
[1] "constr.fit"   "unconstr.fit" "pval"         "coefs"
```

```
R> ans$pval
```

```
[1] 0
```

The constrained fit is compared with the unconstrained fit in Figure 3, where the surfaces are shown for non-smoking girls. For the unconstrained fit, "FEV" increases as "age" increases when "height" is large, but decreases as "age" increases when "height" is small. The constrained fit avoids this situation by keeping the fit of "FEV" non-decreasing with respect to "age".

### 3.4. Shape-restricted regression example

To show how `shapereg` works, we use another real data set, namely "feet". This data set was collected by the second author in a fourth-grade classroom, with the purpose of determining

if boys have wider feet than girls at this age. We use the `shapereg` function to make a shape-restricted fit to this data set. "width" is a continuous response variable, "length" is a continuous predictor variable, and "sex" is a categorical covariate. The constraint is that "width" is increasing in "length".

First, we load the data set and extract the continuous, constrained predictor "length" and the continuous response variable "width":

```
R> data("feet", package = "coneproj")
R> l <- feet$length
R> w <- feet$width
```

Next, we extract the categorical covariate "sex" and transform "sex" into a numerical vector coded as 0 and 1:

```
R> s <- feet$sex
R> n <- length(s)
R> x <- as.numeric(s != "G")
```

Next, we create the $\boldsymbol{X}$ matrix. It is supposed that users provide an $\boldsymbol{n}$ by $\mathbf{1}$ vector, or an $\boldsymbol{n}$ by $\boldsymbol{k}$ matrix, $\boldsymbol{k} \geq 1$. $\boldsymbol{X}$ represents a categorical variable. The user can choose to have a constant vector in $\boldsymbol{X}$ or not.

```
R> xmat <- x
R> shape <- 1
R> ans <- shapereg(w, l, shape, xmat)
```

If we let $\mu_B$ represent the average foot width for fourth-grade boys and $\mu_G$ represent the average foot width for fourth-grade girls, then we can get that the $p$ value is 0.0512 for the test $H_0 : \mu_B = \mu_G$ versus $H_1 : \mu_B > \mu_G$. This indicates that given the test size $\alpha = 0.05$, $\mu_B$ is barely significantly larger than $\mu_G$. Moreover, by the `coefs` returned by `shapereg`, which are the coefficients for $\boldsymbol{X}$, we know that, for a fixed foot length, fourth-grade boys' feet width is estimated to be 0.24 cm larger than fourth-grade girls' feet width, on average.

```
R> names(ans)

[1] "coefs"      "constr.fit" "linear.fit" "se.beta"    "pvals.beta"
[6] "shape"      "test"       "SSE0"       "SSE1"       "call"

R> ans$coefs

[1] 8.8678003 0.2427895

R> head(ans$constr.fit)

         [,1]
[1,] 8.884579
[2,] 9.300000
[3,] 9.033895
[4,] 9.300000
[5,] 9.033895
[6,] 9.490439
```

```
R> head(ans$linear.fit)

        [,1]
[1,] 9.11059
[2,] 9.11059
[3,] 9.11059
[4,] 9.11059
[5,] 9.11059
[6,] 9.11059

R> ans$se.beta

[1] 0.09835963 0.14329232

R> ans$pvals.beta

[1] 0.0000000 0.1023769
```

We can call the R function `summary` to check the estimate, the standard error, the $t$ value and the $p$ value for the coefficient of the linear part. Also we can check the sum of squared residuals for the linear part and the sum of squared residuals for the full model.

```
R> summary(ans)

Call:
shapereg(y = w, t = l, shape = shape, xmat = xmat)

Coefficients:
          Estimate  StdErr t.value p.value
intercept  8.86780 0.09836 90.1569  <2e-16 ***
xmat[,1]   0.24279 0.14329  1.6944  0.1024
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
============================================================
Call:
shapereg(y = w, t = l, shape = shape, xmat = xmat)

     SSE.Linear SSE.Full
[1,]     8.5221    4.251
```

If we set `test = TRUE` in this function, we can get that the $p$ value is around 0.0002 for the $E_{01}$ test for $H_0 : \mathsf{E}(y) \in V$, i.e., $\mathsf{E}(y)$ is constant for both genders versus $H_1 : \mathsf{E}(y) \in \mathcal{C}$, i.e., $\mathsf{E}(y)$ is increasing in foot length for both genders.

```
R> ans <- shapereg(w, l, shape, xmat, test = TRUE)
R> names(ans)
```

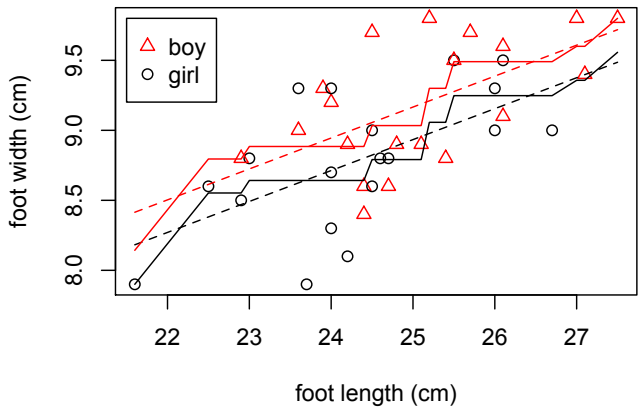


Figure 4: Least-squares increasing fit to "feet" data set (solid lines) as well as least-squares linear fit (dashed lines).

```
 [1] "pval"       "coefs"      "constr.fit" "linear.fit" "se.beta"
 [6] "pvals.beta" "shape"      "test"       "SSE0"       "SSE1"
[11] "call"
```

```
R> ans$pval

[1] 0.000177307
```

We can also call `summary` to check the $p$ value if `test = TRUE` is specified.

```
R> summary(ans)

Call:
shapereg(y = w, t = l, shape = shape, xmat = xmat, test = TRUE)

Coefficients:
          Estimate  StdErr t.value p.value
intercept  8.86780 0.09836 90.1569  <2e-16 ***
xmat[,1]   0.24279 0.14329  1.6944  0.1024
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
============================================================
Call:
shapereg(y = w, t = l, shape = shape, xmat = xmat, test = TRUE)

     SSE.Linear SSE.Full P.value.f(t)
[1,]     8.5221   4.2513    0.0001773 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The scatterplot in Figure 4 shows that while boys tend to have wider feet, they also tend to have longer feet (boys in this classroom were, on average, older and larger), so it makes sense

to control for foot length when comparing foot width by sex. By specifying the argument `shape = 1`, we can make an increasing fit to "width" on "length" with "sex" as a categorical covariate.

# 4. Discussion and simulations

## 4.1. Extension of isotonic regression

The R routine `gpava()` in the package **isotone** (de Leeuw, Hornik, and Mair 2009), and `pava()` in the package **Iso** (Turner 2013) perform isotonic regression, and `conreg()` in the package **cobs** (Ng and Maechler 2011) provides convex or concave regression. However, none of these allows for parametrically modeled covariates. The `shapereg` semi-parametric regression routine will estimate a constrained regression function and a parameter vector using a single cone projection instead of a cyclical back-fitting algorithm. For practical problems, dealing with covariates is essential, for two main reasons. First, users often want to determine the effect of a predictor when other important predictors are controlled for, in order to avoid problems with confounding, as can happen when predictors are related to each other. Second, the power for the test for significance of the predictor of interest is often larger when other predictors are used, because the sum of squared residuals is decreased.

Because multiple parametric regression is so widely available (e.g., the function `lm` in R), practitioners will often use a plausible parametric form even when there is no *a priori* evidence or theory concerning such a function $f$. Instead, there is often vague information such as concerning the shape of $f$.

To demonstrate that the $E_{01}$ test can have higher power than a standard one-sided $t$ test, we consider a simulated data set shown in Figure 5, where two predictors are shown, one continuous ($\boldsymbol{t}$) and one categorical ($\boldsymbol{x}$). The responses $y_i \sim N(f(t_i) + \alpha_0 + \alpha_1 x_i, \sigma^2)$, $i = 1, \ldots, n$, and $y_i$'s are independent; $\boldsymbol{t}$ is a continuous predictor equally spaced on (0, 3], and $\boldsymbol{x}$ is a categorical covariate with two levels 0 and 1.

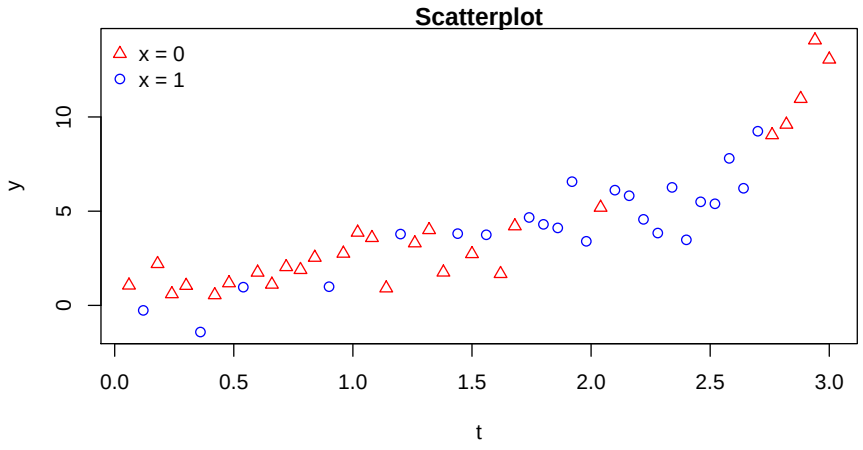


Figure 5: Scatterplot of simulated data where the two predictors ($\boldsymbol{t}$ and $\boldsymbol{x}$) are related to each other.

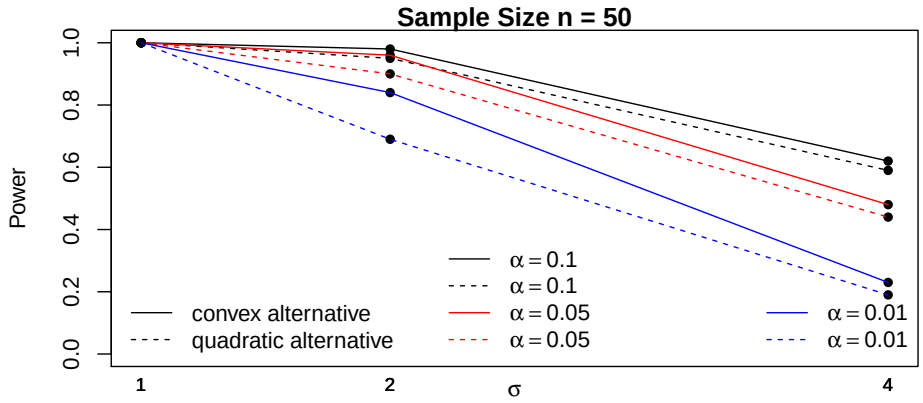


Figure 6: Power for the test of $H_0$ : $f$ is linear against $H_{1a}$ : $f$ is convex and $H_{2b}$ : $f$ is quadratic. The true $f$ is linear on $(0, 2.4]$ and exponential on $(2.4, 3]$, continuous with continuous first derivative on $(0, 3]$.

Suppose the researchers are interested in whether or not the relationship between $\boldsymbol{y}$ and $\boldsymbol{t}$ is linear, while controlling for the effect of $\boldsymbol{x}$. From the scatterplot of $\boldsymbol{t}$, $\boldsymbol{x}$, and $\boldsymbol{y}$, one might be tempted to use a quadratic alternative. However, because the true function is a spliced-together linear and exponential function rather than a quadratic, better power is obtained with the more general convex alternative. The powers are compared in Figure 6, for three test sizes, and $\sigma = 1, 2, 4$.

More importantly, a shape-restricted fit can help eliminate a spurious relationship between the response variable and a confounding variable. In the above example, suppose $\alpha_1 = 0$ so that $\mathsf{E}(\boldsymbol{y})$ does not depend on $\boldsymbol{x}$. Because $\boldsymbol{x}$ and $\boldsymbol{t}$ are related, we know that confounding can occur, that is, we can get a spurious significance for $\boldsymbol{x}$, if the effect of $\boldsymbol{t}$ is not "controlled for" by including $\boldsymbol{t}$ in the model. However, if the functional form of $\boldsymbol{t}$ is mis-specified, confounding can still occur. Using vague assumptions about $f$ will guard against confounding due to mis-specification.

To demonstrate this, we simulated 100,000 data sets with $\alpha_1 = 0$ and $\sigma^2 = 1$, to compare results for the test of $H_0 : \alpha_1 = 0$ versus $H_a : \alpha_1 \neq 0$, using a model that specifies a quadratic in $\boldsymbol{t}$ and using a semi-parametric model with the assumption that $\mathsf{E}(y)$ is convex in $\boldsymbol{t}$. For both tests, we use 0.05 for the test size. For the quadratic model, we (erroneously) reject $H_0$ for 84.2% of the simulated data sets, whereas for our semi-parametric model with convex assumption, we reject $H_0$ for only 5.7% of the data sets.

## 4.2. Speed

The routines `coneA`, `coneB` and `qprog` load a C++ sub-routine for the core part of their computations, which make them considerably faster than if coded completely in R. For example, `coneB` is used for the previous `shapereg` simulations. When the number of simulations is 100,000, for a data set of 50 observations, the time used to get the power for the $E_{01}$ test is roughly between 40 and 90 seconds, using a laptop with a 2.53GHz dual-core Intel(R) Core(TM) i3 CPU. When we call `coneB` for 10,000 times to get the $p$ value for the "feet" data set whose sample size is 39, the time is roughly between 1 and 3 seconds. Finally, when we call `coneA` for 10,000 times to get the $p$ value for the "FEV" data set whose sample size is

654, the time is roughly between 2 and 5 seconds.

We compare the main routines `coneA` and `coneB` in this package with the routine `solve.QP` in the package **quadprog** (Turlach and Weingessel 2013) for solving quadratic programming problems. The scenario we consider is that given a predictor $\boldsymbol{x}$ equally spaced on $[0, 1]$, the response $\boldsymbol{y}$ satisfies that $\boldsymbol{y} = (\boldsymbol{x} - \bar{\boldsymbol{x}})^2 + \boldsymbol{\varepsilon}$, and the elements of $\boldsymbol{\varepsilon}$ are independently normal with the standard deviation being set to 0.2. Fitting a convex regression function is a quadratic programming problem and we use each routine to get a fit for 4 different sample sizes. For small sample sizes, `coneB` is the fastest, and the speeds of `coneA` and `solve.QP` are close. For example, when $n = 10$, for the same scenario, the time `coneB` uses is around 0.05 milliseconds; the time `coneA` uses is around 0.08 milliseconds; the time `solve.QP` uses is around 0.1 milliseconds, and when $n = 100$, the time `coneB` uses is around 0.5 milliseconds; the time `coneA` uses is around 9 milliseconds; the time `solve.QP` uses is around 11 milliseconds. For large sample sizes, the routine `coneB` is faster than `coneA` and `solve.QP` by magnitude. For example, when $n = 1500$, for the same scenario, the time `coneB` uses is roughly between 0.2 and 0.4 seconds; the time `solve.QP` uses can range from 50 to 140 seconds; the time `coneA` uses can range from 170 to 350 seconds, and when $n = 3000$, the time `coneB` uses is roughly between 1 and 3 seconds; the time `solve.QP` uses can range from 500 to 1100 seconds; the time `coneA` uses can range from 2000 to 4300 seconds.

## Acknowledgments

This work was partially funded by the National Science Foundation grant DMS 0905656. The authors are grateful for the helpful comments and suggestions of two referees and an associate editor.

## References

Brunk HD (1958). "On the Estimation of Parameters Restricted by Inequalities." *The Annals of Mathematical Statistics*, **29**(2), 437–454.

Cheng G (2009). "Semiparametric Additive Isotonic Regression." *Journal of Statistical Planning and Inference*, **139**(6), 1980–1991.

de Leeuw J, Hornik K, Mair P (2009). "Isotone Optimization in R: Pool-Adjacent-Violators Algorithm (PAVA) and Active Set Methods." *Journal of Statistical Software*, **32**(5), 1–24. URL http://www.jstatsoft.org/v32/i05/.

Eddelbuettel D (2013). *Seamless R and C++ Integration with **Rcpp***. Springer-Verlag, New York.

Eddelbuettel D, François R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel D, Sanderson C (2014). "**RcppArmadillo**: Accelerating R with High-Performance C++ Linear Algebra." *Computational Statistics & Data Analysis*, **71**, 1054–1063.

Fang SC, Puthenpura S (1993). *Linear Optimization and Extensions*. Prentice Hall, Englewood Cliffs.

Fraser DAS, Massam H (1989). "A Mixed Primal-Dual Bases Algorithm for Regression under Inequality Constraints. Application to Concave Regression." *Scandinavian Journal of Statistics*, **16**(1), 65–74.

Goldfarb D, Idnani A (1983). "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs." *Mathematical Programming*, **27**(1), 1–33.

Kahn MJ (2005). "An Exhalent Problem for Teaching Statistics." *Journal of Statistics Education*, **13**(2).

Meyer MC (1999). "An Extension of the Mixed Primal-Dual Bases Algorithm to the Case of More Constraints than Dimensions." *Journal of Statistical Planning and Inference*, **81**(1), 13–31.

Meyer MC (2003). "A Test for Linear vs. Convex Regression Function using Shape-Restricted Regression." *Biometrika*, **90**(1), 223–232.

Meyer MC (2013a). "Semi-Parametric Additive Constrained Regression." *Journal of Nonparametric Statistics*, **25**(3), 715–743.

Meyer MC (2013b). "A Simple New Algorithm for Quadratic Programming with Applications in Statistics." *Communications in Statistics – Simulation and Computation*, **42**(5), 1126–1139.

Meyer MC, Liao X (2014). ***coneproj**: Primal or Dual Cone Projections with Routines for Constrained Regression*. R package version 1.5, URL http://CRAN.R-project.org/package=coneproj.

Meyer MC, Wang JC (2012). "Improved Power of One-Sided Tests." *Statistics and Probability Letters*, **82**(8), 1619–1622.

Meyer MC, Woodroofe M (2000). "On the Degrees of Freedom in Shape-Restricted Regression." *The Annals of Statistics*, **28**(4), 1083–1104.

Ng PT, Maechler M (2011). ***cobs**: COBS – Constrained B-Splines (Sparse Matrix Based)*. R package version 1.2-2, URL http://CRAN.R-project.org/package=cobs.

Raubertas RF, Lee CIC, Nordheim EV (1986). "Hypothesis Tests for Normals Means Constrained by Linear Inequalities." *Communications in Statistics – Theory and Methods*, **15**(9), 2809–2833.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Robertson T, Wright FT, Dykstra RL (1988). *Order Restricted Statistical Inference*. John Wiley & Sons, New York.

Silvapulle MJ, Sen PK (2005). *Constrained Statistical Inference*. John Wiley & Sons.

Turlach BA, Weingessel A (2013). ***quadprog***: *Functions to Solve Quadratic Programming Problems*. R package version 1.5-5, URL http://CRAN.R-project.org/package=quadprog.

Turner R (2013). ***Iso***: *Functions to Perform Isotonic Regression*. R package version 0.0-15, URL http://CRAN.R-project.org/package=Iso.

## Affiliation:

Xiyue Liao
Department of Statistics
Colorado State University
Fort Collins, Colorado 80523, United States of America
E-mail: xiyue@rams.colostate.edu

Mary C. Meyer
Department of Statistics
Colorado State University
Fort Collins, Colorado 80523, United States of America
E-mail: meyer@stat.colostate.edu

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 61, Issue 12 *Submitted:* 2013-09-18
November 2014 *Accepted:* 2014-06-11