CVXR: AN R PACKAGE FOR DISCIPLINED CONVEX OPTIMIZATION

By

Anqi Fu
Balasubramanian Narasimhan
Stephen Boyd

Technical Report No. 2017-14
November 2017

Department of Statistics
STANFORD UNIVERSITY
Stanford, California 94305-4065

CVXR: AN R PACKAGE FOR DISCIPLINED CONVEX OPTIMIZATION

By

Anqi Fu
Balasubramanian Narasimhan
Stephen Boyd
Stanford University

Technical Report No. 2017-14
November 2017

This research was supported in part by National Institutes of Health grant 1UL1 RR025744.

Department of Statistics
STANFORD UNIVERSITY
Stanford, California 94305-4065

http://statistics.stanford.edu

CVXR: An R Package for Disciplined Convex Optimization

Anqi Fu, Balasubramanian Narasimhan, and Stephen Boyd

Abstract

CVXR is an R package that provides an object-oriented modeling language for convex optimization, similar to **CVX**, **CVXPY**, **YALMIP**, and **Convex.jl**. It allows the user to formulate convex optimization problems in a natural mathematical syntax rather than the restrictive standard form required by most solvers. The user specifies an objective and set of constraints by combining constants, variables, and parameters using a library of functions with known mathematical properties. **CVXR** then applies signed disciplined convex programming (DCP) to verify the problem's convexity. Once verified, the problem is converted into standard conic form using graph implementations and passed to a cone solver such as ECOS or SCS. We demonstrate **CVXR**'s modeling framework with several applications.

Keywords: Convex Optimization, Disciplined Convex Optimization, Optimization, Regression, Penalized Regression, Isotonic Regression, R package **CVXR**.

1. Introduction

Optimization plays an important role in fitting many statistical models. Some examples include least squares, ridge and lasso regression, isotonic regression, Huber regression, support vector machines, and sparse inverse covariance estimation. Many R packages exist for solving families of such problems. Our package, **CVXR**, solves the much broader class of convex optimization problems, which encompasses these families and a wide range of other models and methods in statistics. Similar systems already exist for MATLAB (Grant and Boyd 2014; Lofberg 2004), Python (Diamond and Boyd 2016), and Julia (Udell, Mohan, Zeng, Hong, Diamond, and Boyd 2014). **CVXR** brings these capabilities to R, providing a domain-specific language (DSL) that allows users to easily formulate and solve new problems for which custom code does not exist.

As an illustration, suppose we are given $X \in \mathbf{R}^{m \times n}$ and $y \in \mathbf{R}^m$, and we want to solve the ordinary least squares (OLS) problem

$$\underset{\beta}{\text{minimize}} \quad \|y - X\beta\|_2^2$$

with optimization variable $\beta \in \mathbf{R}^n$. This problem has a well-known analytical solution, which can be determined via a variety of methods such as **glmnet** (Friedman, Hastie, and Tibshirani 2010). In **CVXR**, we can solve for β using the code

```
beta <- Variable(n)
obj <- sum((y - X %*% beta)^2)
```

```
prob <- Problem(Minimize(obj))
result <- solve(prob)
```

The first line declares our variable, the second line forms our objective function, the third line defines the optimization problem, and the last line solves this problem by converting it into a second-order cone program and sending it to one of **CVXR**'s solvers. The results are retrieved with

```
result$value              # Optimal objective
result$getValue(beta)     # Optimal variables
result$metrics            # Runtime metrics
```

This code runs slower and requires additional set-up at the beginning. So far, it does not look like progress. However, suppose we add a constraint to our problem:

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & \|y - X\beta\|_2^2 \\ \text{subject to} \quad & \beta_j \leq \beta_{j+1}, \quad j = 1, \ldots, n-1. \end{aligned}$$

This is a special case of isotonic regression. Now, we can no longer use **glmnet** for the optimization. We would need to find another R package tailored to this type of problem or write our own custom solver. With **CVXR** though, we need only add the constraint as a second argument to the problem:

```
prob <- Problem(Minimize(obj), list(diff(beta) >= 0))
```

Our new problem definition includes the coefficient constraint, and a call to `solve` with the associated constraint will produce the solution. In addition to the usual results, we can get the dual variables with

```
result$getDualValue(constraints(prob)[[1]])
```

This example demonstrates **CVXR**'s chief advantage: flexibility. Users can quickly modify and re-solve a problem, making our package ideal for prototyping new statistical methods. Its syntax is simple and mathematically intuitive. Furthermore, **CVXR** combines seamlessly with native R code as well as several popular packages, allowing it to be incorporated easily into a larger analytical framework. The user can, for instance, apply resampling techniques like the bootstrap to estimate variability, as we show in §3.2.3.

DSLs for convex optimization are already widespread on other application platforms. In R, users have access to the packages listed in the CRAN Task View for *Optimization and Mathematical Programming*. Packages like **optimx** (Nash and Varadhan 2011) and **nloptr** (Johnson 2008) implement many different algorithms, each with their own strengths and weaknesses. However, most are limited to a subset of the convex problems we handle or require analytical details (e.g., the gradient function) for a particular solver. **ROI** (Hornik, Meyer, Schwendinger, and Theussl 2017) is perhaps the package closest to ours in spirit. It offers an object-oriented framework for defining optimization problems, but still requires users to explicitly identify the type of every objective and constraint, whereas **CVXR** manages this process automatically.

In the next section, we provide a brief mathematical overview of convex optimization. Interested readers can find a full treatment in Boyd and Vandenberghe (2004). Then we give a series of examples ranging from basic regression models to semidefinite programming, which demonstrate the simplicity of problem construction in **CVXR**. Finally, we describe the implementation details before concluding. Our package and the example code for this paper are available on CRAN and the official **CVXR** site.

2. Disciplined Convex Optimization

The general convex optimization problem is of the form

$$\begin{array}{ll} \underset{v}{\text{minimize}} & f_0(v) \\ \text{subject to} & f_i(v) \leq 0, \quad i = 1, \ldots, M \\ & Av = b, \end{array}$$

where $v \in \mathbf{R}^n$ is our variable of interest, and $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^n$ are constants describing our linear equality constraints. The objective and inequality constraint functions $f_0, \ldots, f_M$ are convex, *i.e.*, they are functions $f_i : \mathbf{R}^n \to \mathbf{R}$ that satisfy

$$f_i(\theta u + (1-\theta)v) \leq \theta f_i(u) + (1-\theta) f_i(v)$$

for all $u, v \in \mathbf{R}^n$ and $\theta \in [0, 1]$. This class of problems arises in a variety of fields, including machine learning and statistics.

A number of efficient algorithms exist for solving convex problems (Wright 1997; Boyd, Parikh, Chu, Peleato, and Eckstein 2011; Andersen, Dahl, Liu, and Vandenberghe 2011; Skajaa and Ye 2015). However, it is unnecessary for the **CVXR** user to know the operational details of these algorithms. **CVXR** provides a DSL that allows the user to specify the problem in a natural mathematical syntax. This specification is automatically converted into the standard form ingested by a generic convex solver. See §4 for more on this process.

In general, it can be difficult to determine whether an optimization problem is convex. We follow an approach called disciplined convex programming (DCP) (Grant, Boyd, and Ye 2006) to define problems using a library of basic functions (atoms), whose properties like curvature, monotonicity, and sign are known. Adhering to the DCP rule,

$f(g_1, \ldots, g_k)$ is convex if f is convex and for each $i = 1, \ldots, k$, either

- g_i is affine,
- g_i is convex and f is increasing in argument i, or
- g_i is concave and f is decreasing in argument i,

we combine these atoms such that the resulting problem is convex by construction. Users will need to become familiar with this rule if they wish to define complex problems.

The library of available atoms is provided in the documentation. It covers an extensive array of functions, enabling any user to model and solve a wide variety of sophisticated optimization problems. In the next section, we provide sample code for just a few of these problems, many of which are cumbersome or impossible to solve with other R packages.

3. Examples

In the following examples, we are given a dataset (x_i, y_i) for $i = 1, \ldots, m$, where $x_i \in \mathbf{R}^n$ and $y_i \in \mathbf{R}$. We represent these observations in matrix form as $X \in \mathbf{R}^{m \times n}$ with stacked rows x_i^T and $y \in \mathbf{R}^m$. Generally, we assume that $m > n$.

3.1. Regression

Robust (Huber) Regression

In §1, we saw an example of OLS in **CVXR**. While least squares is a popular regression model, one of its flaws is its high sensitivity to outliers. A single outlier that falls outside the tails of the normal distribution can drastically alter the resulting coefficients, skewing the fit on the other data points. For a more robust model, we can fit a Huber regression (Huber 1964) instead by solving

$$\underset{\beta}{\text{minimize}} \quad \sum_{i=1}^m \phi(y_i - x_i^T \beta)$$

for variable $\beta \in \mathbf{R}^n$, where the loss is the Huber function with threshold $M > 0$,

$$\phi(u) = \begin{cases} u^2 & \text{if } |u| \leq M \\ 2Mu - M^2 & \text{if } |u| > M. \end{cases}$$

This function is identical to the least squares penalty for small residuals, but on large residuals, its penalty is lower and increases linearly rather than quadratically. It is thus more forgiving of outliers.

In **CVXR**, the code for this problem is

```
beta <- Variable(n)
obj <- sum(huber(y - X %*% beta, M))
prob <- Problem(Minimize(obj))
result <- solve(prob)
```

Note the similarity to the OLS code. As before, the first line instantiates the n-dimensional optimization variable, and the second line defines the objective function by combining this variable with our data using **CVXR**'s library of atoms. The only difference this time is we call the `huber` atom on the residuals with threshold `M`, which we assume has been set to a positive scalar constant. Our package provides many such atoms to simplify problem definition for the user.

Quantile Regression

Another variation on least squares is quantile regression (Koenker 2005). The loss is the tilted l_1 function,

$$\phi(u) = \tau \max(u, 0) - (1 - \tau) \max(-u, 0) = \frac{1}{2}|u| + \left(\tau - \frac{1}{2}\right) u,$$

where $\tau \in (0, 1)$ specifies the quantile. The problem as before is to minimize the total residual loss. This model is commonly used in ecology, healthcare, and other fields where the mean

alone is not enough to capture complex relationships between variables. **CVXR** allows us to create a function to represent the loss and integrate it seamlessly into the problem definition, as illustrated below.

```
quant_loss <- function(u, tau) { 0.5*abs(u) + (tau - 0.5)*u }
obj <- sum(quant_loss(y - X %*% beta, t))
prob <- Problem(Minimize(obj))
result <- solve(prob)
```

Here `t` is the user-defined quantile parameter. We do not need to create a new `Variable` object, since we can reuse `beta` from the previous example.

Censored Regression

Data collected from an experimental study is sometimes censored, so that only partial information is known about a subset of observations. For instance, when measuring the lifespan of mice, we may find a number of subjects live beyond the duration of the project. Thus, all we know is the lower bound on their lifespan. This right censoring can be incorporated into a regression model via convex optimization.

Suppose that only K of our observations (x_i, y_i) are fully observed, and the remaining are censored such that we observe x_i, but only know $y_i \geq D$ for $i = K+1, \ldots, m$ and some constant $D \in \mathbf{R}$. We can build an OLS model using the uncensored data, restricting the fitted values $\hat{y}_i = x_i^T \beta$ to lie above D for the censored observations:

$$\begin{array}{ll} \underset{\beta}{\text{minimize}} & \sum_{i=1}^{K} (y_i - x_i^T \beta)^2 \\ \text{subject to} & x_i^T \beta \geq D, \quad i = K+1, \ldots, m. \end{array}$$

This avoids the bias introduced by standard OLS, while still utilizing all of the data points in the regression. The constraint requires only one more line in **CVXR**.

```
beta <- Variable(n)
obj <- sum((y[1:K] - X[1:K,] %*% beta)^2)
constr <- list(X[(K+1):m,] %*% beta >= D)
prob <- Problem(Minimize(obj), constr)
result <- solve(prob)
```

We can just as easily accommodate left censoring, interval censoring, and other loss functions with minor changes to the code above.

Elastic Net Regularization

Often in applications, we encounter problems that require regularization to prevent overfitting, introduce sparsity, facilitate variable selection, or impose prior distributions on parameters. Two of the most common regularization functions are the l_1-norm and squared l_2-norm, combined in the elastic net regression model (H. Zou 2005; Friedman *et al.* 2010),

$$\underset{\beta}{\text{minimize}} \quad \frac{1}{2m}\|y - X\beta\|_2^2 + \lambda\left(\frac{1-\alpha}{2}\|\beta\|_2^2 + \alpha\|\beta\|_1\right).$$

Here $\lambda \geq 0$ is the overall regularization weight and $\alpha \in [0, 1]$ controls the relative l_1 versus squared l_2 penalty. Thus, this model encompasses both ridge ($\alpha = 0$) and lasso ($\alpha = 1$) regression.

To solve this problem in **CVXR**, we first define a function that calculates the regularization term given the variable and penalty weights.

```
elastic_reg <- function(beta, lambda = 0, alpha = 0) {
    ridge <- (1 - alpha) * sum(beta^2)
    lasso <- alpha * p_norm(beta, 1)
    lambda * (lasso + ridge)
}
```

Then, we add it to the scaled least squares loss.

```
loss <- sum((y - X %*% beta)^2)/(2*m)
obj <- loss + elastic_reg(beta, lambda, alpha)
prob <- Problem(Minimize(obj))
result <- solve(prob)
```

The advantage of this modular approach is that we can easily incorporate elastic net regularization into other regression models. For instance, if we wanted to run regularized Huber regression, **CVXR** allows us to reuse the above code with just a single changed line,

```
loss <- huber(y - X %*% beta, M)
```

Logistic Regression

Suppose now that $y_i \in \{0, 1\}$ is a binary class indicator. One of the most popular methods for binary classification is logistic regression (Cox 1958; Freedman 2009). We model the conditional response as $y|x \sim \text{Bernoulli}(g_\beta(x))$, where $g_\beta(x) = \frac{1}{1+e^{-x^T\beta}}$ is the logistic function, and maximize the log-likelihood function, yielding the optimization problem

$$\underset{\beta}{\text{maximize}} \quad \sum_{i=1}^{m} \{y_i \log(g_\beta(x_i)) + (1 - y_i) \log(1 - g_\beta(x_i))\}.$$

CVXR provides the `logistic` atom as a shortcut for $f(z) = \log(1 + e^z)$, so our problem is succinctly expressed as

```
obj <- -sum(logistic(-X[y == 0,] %*% beta)) - sum(logistic(X[y == 1,] %*% beta))
prob <- Problem(Maximize(obj))
result <- solve(prob)
```

The user may be tempted to type `log(1 + exp(X %*% beta))` as in conventional R syntax. However, this representation of $f(z)$ violates the DCP composition rule, so the **CVXR** parser will reject the problem even though the objective is convex. Users who wish to employ a function that is convex, but not DCP compliant should check the documentation for a custom atom or consider a different formulation.

We can retrieve the optimal objective and variables just like in OLS. More interestingly, we can evaluate various functions of these variables as well by passing them directly into `result$getValue`. For instance, the log-odds are

```
log_odds <- result$getValue(X %*% beta)
```

This coincides with the ratio we get from computing the probabilities directly:

```
beta_res <- result$getValue(beta)
y_probs <- 1/(1 + exp(-X %*% beta_res))
log(y_probs/(1 - y_probs))
```

Many other classification methods fit into the convex framework. For example, the support vector classifier is the solution of a l_2-norm minimization problem with linear constraints, which we have already shown how to model. Support vector machines are a straightforward extension. The multinomial distribution can be used to predict multiple classes, and estimation via maximum likelihood produces a convex problem. To each of these methods, we can easily add new penalties, variables, and constraints in **CVXR**, allowing us to adapt to a specific dataset or environment.

Sparse Inverse Covariance Estimation

Assume we are given i.i.d. observations $x_i \sim N(0, \Sigma)$ for $i = 1, \ldots, m$, and the covariance matrix $\Sigma \in \mathbf{S}_+^n$, the set of symmetric positive semidefinite matrices, has a sparse inverse $S = \Sigma^{-1}$. Let $Q = \frac{1}{m-1}\sum_{i=1}^m (x_i - \bar{x})(x_i - \bar{x})^T$ be our sample covariance. One way to estimate Σ is to maximize the log-likelihood with the prior knowledge that S is sparse (Friedman, Hastie, and Tibshirani 2008), which amounts to the optimization problem

$$\begin{array}{ll} \underset{S}{\text{maximize}} & \log\det(S) - \text{tr}(SQ) \\ \text{subject to} & S \in \mathbf{S}_+^n, \quad \sum_{i=1}^n \sum_{j=1}^n |S_{ij}| \leq \alpha. \end{array}$$

The parameter $\alpha \geq 0$ controls the degree of sparsity. Our problem is convex, so we can solve it with

```
S <- Semidef(n)
obj <- log_det(S) - matrix_trace(S %*% Q)
constr <- list(sum(abs(S)) <= alpha)
prob <- Problem(Maximize(obj), constr)
result <- solve(prob)
```

The `Semidef` constructor restricts `S` to the positive semidefinite cone. In our objective, we use **CVXR** functions for the log-determinant and trace. The expression `matrix_trace(S %*% Q)` is equivalent to `sum(diag(S %*% Q))`, but the former is preferred because it is more efficient than making nested function calls. However, a standalone atom does not exist for the determinant, so we cannot replace `log_det(S)` with `log(det(S))` since `det` is undefined for a `Semidef` object.

Figure 1 depicts the solutions for a particular dataset with $m = 1000, n = 10$, and S containing 26% non-zero entries represented by the black squares in the top left image. The sparsity of our inverse covariance estimate decreases for higher α, so that when $\alpha = 1$, most of the off-diagonal entries are zero, while if $\alpha = 10$, over half the matrix is dense. At $\alpha = 4$, we achieve the true percentage of non-zeros.

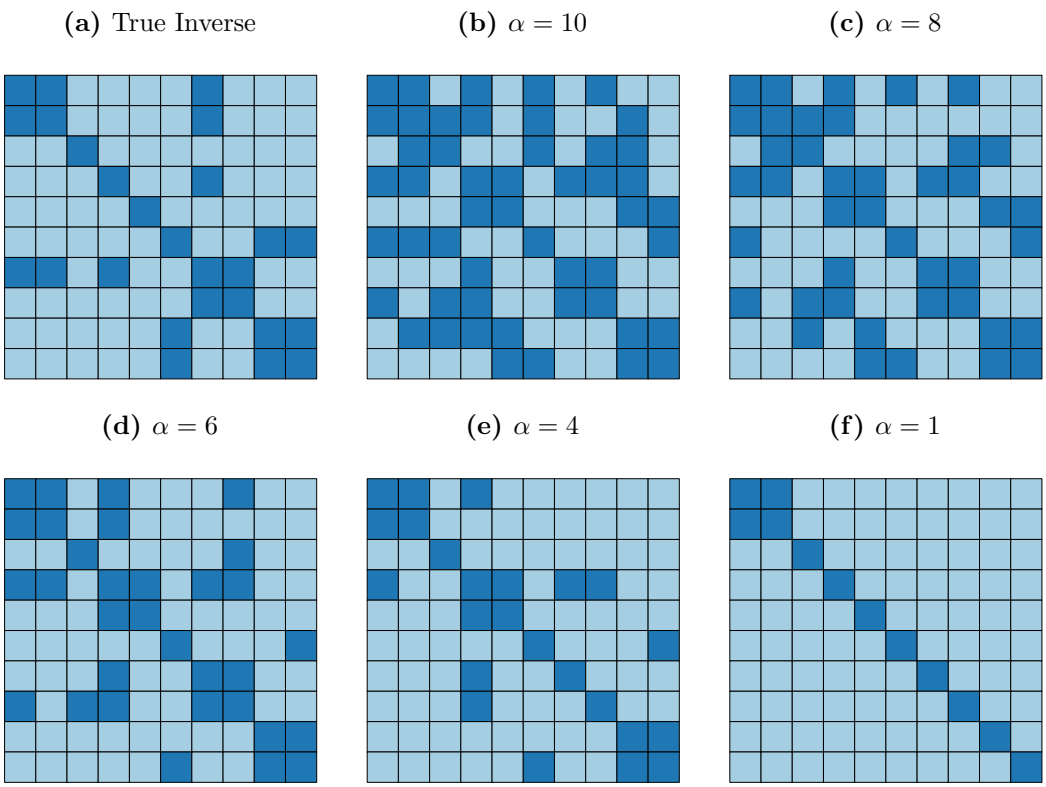

Figure 1: Sparsity patterns for (a) inverse of true covariance matrix, and estimated inverse covariance matrices with (b) $\alpha = 10$, (c) $\alpha = 8$, (d) $\alpha = 6$, (e) $\alpha = 4$, and (f) $\alpha = 1$. The dark regions indicate where S_{ij} is nonzero.

Saturating Hinges

The following example comes from work on saturating splines in Boyd, Hastie, Boyd, Recht, and Jordan (2016). Adaptive regression splines are commonly used in statistical modeling, but the instability they exhibit beyond their boundary knots makes extrapolation dangerous. One way to correct this issue for linear splines is to require they *saturate*: remain constant outside their boundary. This problem can be solved using a heuristic that is an extension of lasso regression, producing a weighted sum of hinge functions, which we call a *saturating hinge*.

For simplicity, consider the univariate case with $n = 1$. Assume we are given knots $t_1 < t_2 < \cdots < t_k$ where each $t_j \in \mathbf{R}$. Let h_j be a hinge function at knot t_j, *i.e.*, $h_j(x) = \max(x - t_j, 0)$, and define $f(x) = w_0 + \sum_{j=1}^{k} w_j h_j(x)$. We want to solve

$$\begin{array}{ll} \underset{w_0, w}{\text{minimize}} & \sum_{i=1}^{m} \ell(y_i, f(x_i)) + \lambda \|w\|_1 \\ \text{subject to} & \sum_{j=1}^{k} w_j = 0 \end{array}$$

for variables $(w_0, w) \in \mathbf{R} \times \mathbf{R}^k$. The function $\ell : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ is the loss associated with every observation, and $\lambda \geq 0$ is the penalty weight. In choosing our knots, we set $t_1 = \min(x_i)$ and $t_k = \max(x_i)$ so that by construction, the estimate $\hat{f}$ will be constant outside $[t_1, t_k]$.

We demonstrate this technique on the bone density data for female patients from Hastie, Tibshirani, and Friedman (2001, §5.4). There are a total of $m = 259$ observations. Our response y_i is the change in spinal bone density between two visits, and our predictor x_i is the patient's age. We select $k = 10$ knots about evenly spaced across the range of X and fit a saturating hinge with squared error loss $\ell(y_i, f(x_i)) = (y_i - f(x_i))^2$.

In R, we first define the estimation and loss functions:

```
f_est <- function(x, knots, w0, w) {
    hinges <- sapply(knots, function(t) { pmax(x - t, 0) })
    w0 + hinges %*% w
}
loss_obs <- function(y, f) { (y - f)^2 }
```

This allows us to easily test different losses and knot locations later. The rest of the set-up is similar to previous examples. We assume that `knots` is a R vector representing $(t_1, \ldots, t_k)$.

```
w0 <- Variable()
w <- Variable(k)

loss <- sum(loss_obs(y, f_est(X, knots, w0, w)))
reg <- lambda * p_norm(w, 1)
obj <- loss + reg
constr <- list(sum(w) == 0)
prob <- Problem(Minimize(obj), constr)
result <- solve(prob)
```

The optimal weights are retrieved using separate calls, as shown below.

```
w0s <- result$getValue(w0)
ws <- result$getValue(w)
```

We plot the fitted saturating hinges in Figure 2a. As expected, when λ increases, the spline exhibits less variation and grows flatter outside its boundaries. The squared error loss works well in this case, but as we saw in §3.1.1, the Huber loss is preferred when the dataset contains large outliers. We can change the loss function by simply redefining

```
loss_obs <- function(y, f, M) { huber(y - f, M) }
```

and passing an extra threshold parameter in when initializing `loss`. In Figure 2b, we have added 50 randomly generated outliers to the bone density data and plotted the re-fitted saturating hinges. For a Huber loss with $M = 0.01$, the resulting spline is fairly smooth and follows the shape of the original data, as opposed to the spline using squared error loss, which is biased upwards by a significant amount.

3.2. Nonparametric Estimation

Log-Concave Distribution Estimation

(a) Squared Error Loss **(b)** Saturating Hinges with Outliers

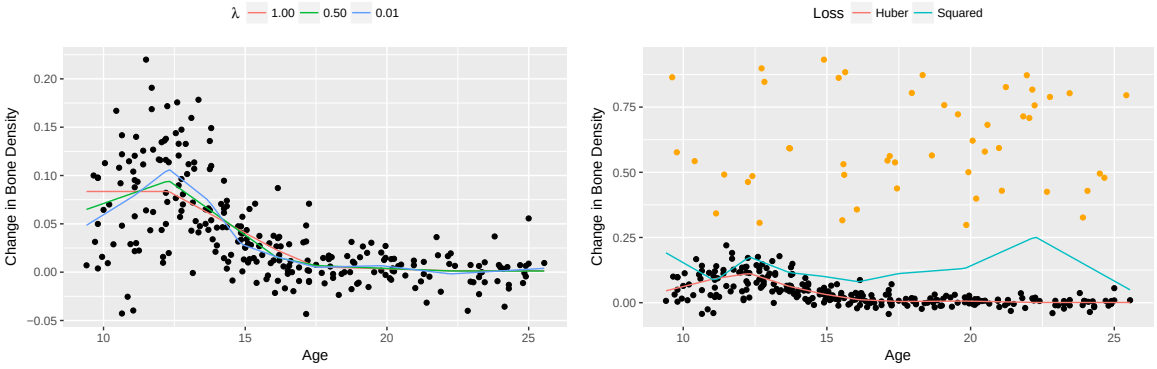

Figure 2: (a) Saturating hinges fit to the change in bone density for female patients with $\lambda = 0.01$ (blue), $\lambda = 0.5$ (green), and $\lambda = 1$ (red). (b) Hinges refit to the previous data with additional outliers (orange) using squared error (blue) and Huber loss (red).

Let $n = 1$ and suppose x_i are i.i.d. samples from a log-concave discrete distribution on $\{0, \ldots, K\}$ for some $K \in \mathbf{Z}_+$. Define $p_k := \mathsf{P}(X = k)$ to be the probability mass function. One method for estimating $(p_0, \ldots, p_K)$ is to maximize the log-likelihood function subject to a log-concavity constraint (Dümbgen and Rufibach 2009), *i.e.*,

$$\begin{array}{ll} \underset{p}{\text{maximize}} & \sum_{k=0}^{K} M_k \log p_k \\ \text{subject to} & p \geq 0, \quad \sum_{k=0}^{K} p_k = 1, \\ & p_k \geq \sqrt{p_{k-1} p_{k+1}}, \quad k = 1, \ldots, K-1, \end{array}$$

where $p \in \mathbf{R}^{K+1}$ is our variable of interest and M_k represents the number of observations equal to k, so that $\sum_{k=0}^{K} M_k = m$. The problem as posed above is not convex. However, we can transform it into a convex optimization problem by defining new variables $u_k = \log p_k$ and relaxing the equality constraint to $\sum_{k=0}^{K} p_k \leq 1$, since the latter always holds tightly at an optimal solution. The result is

$$\begin{array}{ll} \underset{u}{\text{maximize}} & \sum_{k=0}^{K} M_k u_k \\ \text{subject to} & \sum_{k=0}^{K} e^{u_k} \leq 1, \\ & u_k - u_{k-1} \geq u_{k+1} - u_k, \quad k = 1, \ldots, K-1. \end{array}$$

If `counts` is the R vector of $(M_0, \ldots, M_K)$, the code for our convex problem is

```
u <- Variable(K+1)
obj <- t(counts) %*% u
constr <- list(sum(exp(u)) <= 1, diff(u[1:K]) >= diff(u[2:(K+1)]))
prob <- solve(Maximize(obj), constr)
result <- solve(prob)
```

Once the solver is finished, we can retrieve the probabilities directly with

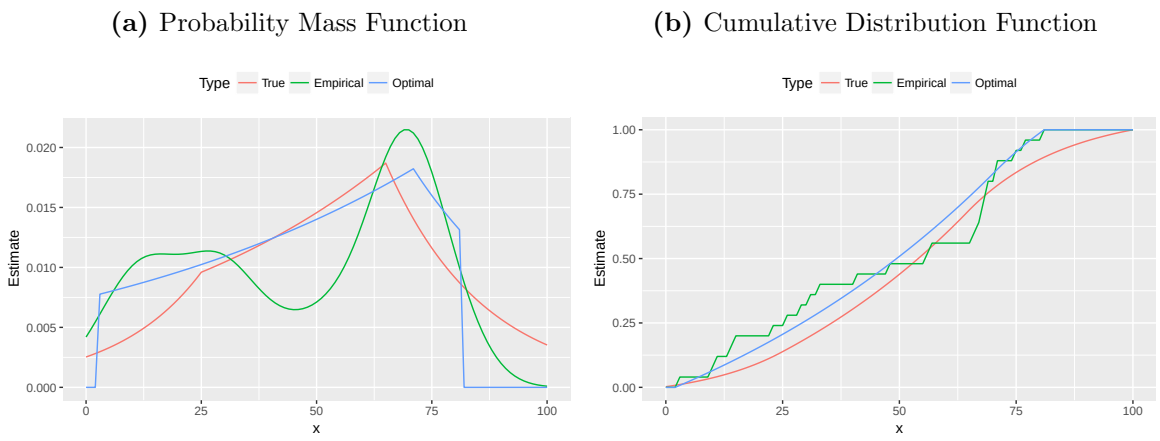

Figure 3: (a) True (red), empirical (green), and estimated (blue) probability mass functions. The empirical distribution was smoothed with a Gaussian kernel. (b) Cumulative distribution functions corresponding to the probabilities in (a).

```
pmf <- result$getValue(exp(u))
```

The above line transforms the variables u_k to e^{u_k} before calculating their resulting values. This is possible because `exp` is a member of **CVXR**'s library of atoms, so it can operate directly on a `Variable` object such as `u`.

As an example, we draw $m = 25$ observations from a log-concave distribution on $\{0, \ldots, 100\}$. We then estimate the probability mass function using the above method and compare it with the empirical distribution. From Figure 3, we see that the estimated curve is much closer to the true distribution, exhibiting a similar shape and number of peaks. In contrast, the empirical probability mass function oscillates, failing to be log-concave on parts of its domain. These differences are reflected in the cumulative distribution functions as well.

Direct Standardization

Consider a set of observations (x_i, y_i) drawn non-uniformly from an unknown distribution. We know the expected value of the columns of X, denoted by $b \in \mathbf{R}^n$, and want to estimate the true distribution of y. This situation may arise, for instance, if we wish to analyze the health of a population based on a sample skewed toward young males, knowing the average population-level sex, age, etc. The empirical distribution that places equal probability $\frac{1}{m}$ on each y_i is not a good estimate. Instead, we must determine the weights $w \in \mathbf{R}^m$ of a weighted empirical distribution, $y = y_i$ with probability w_i, which rectifies the skewness of the sample (Fleiss, Levin, and Paik 2003, §19.5). We can pose this problem as

$$\begin{array}{ll} \underset{w}{\text{maximize}} & \sum_{i=1}^m -w_i \log w_i \\ \text{subject to} & w \geq 0, \quad \sum_{i=1}^m w_i = 1, \quad X^T w = b. \end{array}$$

Our objective is the total entropy, which is concave on $\mathbf{R}_+^m$, and our constraints ensure w is a probability distribution that implies our known expectations on X. The corresponding **CVXR** code is

```
w <- Variable(m)
obj <- sum(entr(w))
constr <- list(w >= 0, sum(w) == 1, t(X) %*% w == b)
prob <- Problem(Maximize(obj), constr)
result <- solve(prob)
```

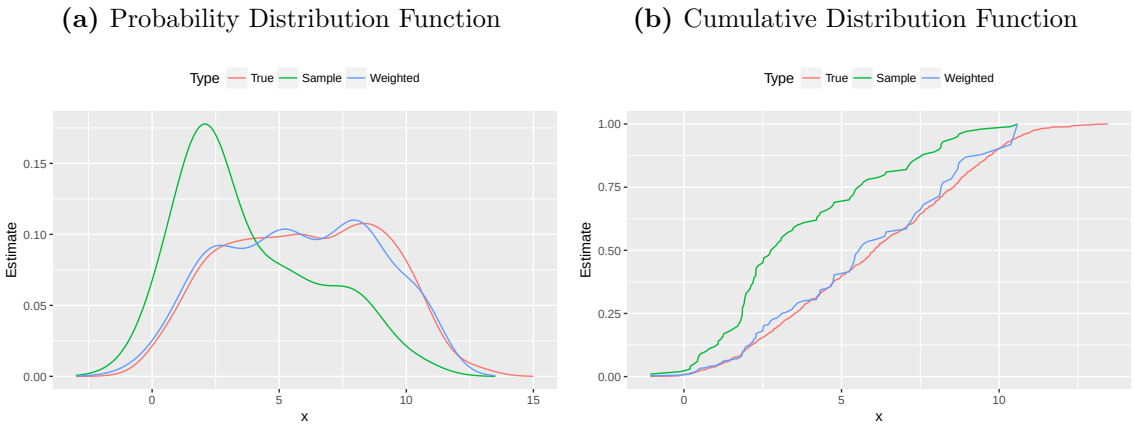

Figure 4: (a) Probability distribution functions (smoothed with a Gaussian kernel) for the population (red), skewed sample (green), and reweighted sample using direct standardization (blue). (b) Cumulative distribution functions for the datasets in (a).

To illustrate this method, we generate $m = 1000$ data points $x_{i,1} \sim \text{Bernoulli}(0.5)$, $x_{i,2} \sim \text{Uniform}(10, 60)$, and $y_i \sim N(5x_{i,1} + 0.1x_{i,2}, 1)$. Then we construct a skewed sample of $m = 100$ points that overrepresent small values of y_i, thus biasing its distribution downwards. This can be seen in Figure 4, where the sample probability distribution peaks around $y = 2.0$, and its cumulative distribution is shifted left from the population's curve. Using direct standardization, we estimate w_i and reweight our sample; the new empirical distribution cleaves much closer to the true distribution shown in red.

Nearly-Isotonic and Nearly-Convex Fits

Given a set of data points $y \in \mathbf{R}^m$, Tibshirani, Hoefling, and Tibshirani (2011) fit a nearly-isotonic approximation $\beta \in \mathbf{R}^m$ by solving

$$\underset{\beta}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{m}(y_i - \beta_i)^2 + \lambda \sum_{i=1}^{m-1}(\beta_i - \beta_{i+1})_+,$$

where $\lambda \geq 0$ is a penalty parameter and $x_+ = \max(x, 0)$. Our **CVXR** formulation follows directly as shown below. The `pos` atom evaluates x_+ elementwise on the input expression.

```
near_fit <- function(y, lambda) {
    m <- length(y)
    beta <- Variable(m)
    penalty <- sum(pos(diff(beta)))
    obj <- 0.5 * sum((y - beta)^2) + lambda * penalty
    prob <- Problem(Minimize(obj))
```

```
    result <- solve(prob)
    result$getValue(beta)
}
```

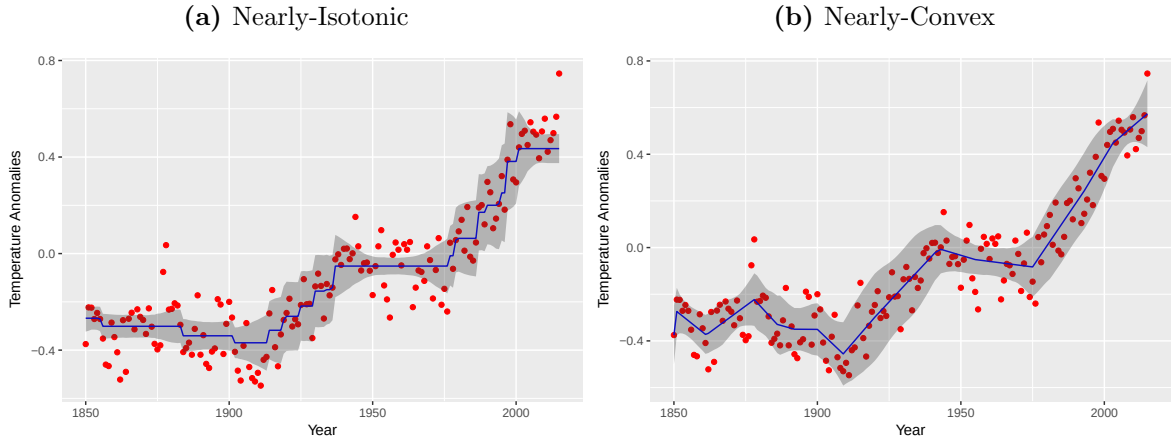

Figure 5: (a) A nearly-isotonic fit and (b) nearly-convex fit to global warming data on temperature anomalies for $\lambda = 0.44$. The 95% normal confidence intervals are shown in gray using $R = 999$ and $R = 100$ bootstrap samples, respectively.

We demonstrate this technique on the global warming data provided by the Carbon Dioxide Information Analysis Center (CDIAC). Our data points are the annual temperature anomalies relative to the 1961–1990 mean. Combining `near_fit` with the `boot` library, we can obtain the standard errors and confidence intervals for our estimate in just a few lines of code.

```
near_iso <- function(data, index, lambda) {
    sample <- data[index,]                  # Bootstrap sample of rows
    sample <- sample[order(sample$year),]   # Order ascending by year
    near_fit(sample$annual, lambda)
}
boot.out <- boot(CDIAC, near_iso, R = 999, lambda = 0.44)
```

Figure 5a shows a nearly-isotonic fit with $\lambda = 0.44$ and 95% normal confidence bands, which were generated using $R = 999$ bootstrap samples. The curve follows the data well, but exhibits choppiness in regions with a steep trend.

For a smoother curve, we can solve for the nearly-convex fit described in the same paper:

$$\underset{\beta}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{m}(y_i - \beta_i)^2 + \lambda \sum_{i=1}^{m-2}(\beta_i - 2\beta_{i+1} + \beta_{i+2})_+$$

This replaces the first difference term with an approximation to the second derivative at β_{i+1}. In **CVXR**, the only change necessary is the penalty line in `near_fit`,

```
penalty <- sum(pos(diff(x, differences = 2)))
```

The resulting curve is depicted in Figure 5b with 95% confidence bands generated from $R = 100$ samples. Note the jagged staircase pattern has been smoothed out. We can easily extend this example to higher-order differences or lags by modifying the arguments to `diff`.

3.3. Miscellaneous Applications

Worst Case Covariance

Suppose we have i.i.d. samples $x_i \sim N(0, \Sigma)$ for $i = 1, \ldots, m$ and want to determine the maximum covariance of $y = w^T x = \sum_{i=1}^m w_i x_i$, where $w \in \mathbf{R}^m$ is a given vector of weights. We are provided limited information on the elements of Σ. For example, we may know the specific value or sign of certain Σ_{jk}, which are represented by upper and lower bound matrices L and $U \in \mathbf{R}^{n \times n}$, respectively (Boyd and Vandenberghe 2004, pp. 171–172). This situation can arise when calculating the worst-case risk of an investment portfolio (Lobo and Boyd 2000). Formally, our optimization problem is

$$\begin{array}{ll} \underset{\Sigma}{\text{maximize}} & w^T \Sigma w \\ \text{subject to} & \Sigma \in \mathbf{S}_+^n, \quad L_{jk} \leq \Sigma_{jk} \leq U_{jk}, \quad j, k = 1, \ldots, n. \end{array}$$

Consider the specific case

$$w = \begin{bmatrix} 0.1 \\ 0.2 \\ -0.05 \\ 0.1 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} 0.2 & + & + & \pm \\ + & 0.1 & - & - \\ + & - & 0.3 & + \\ \pm & - & + & 0.1 \end{bmatrix},$$

where a $+$ means the element is nonnegative, a $-$ means the element is nonpositive, and a $\pm$ means the element can be any real number. In **CVXR**, this semidefinite program is

```
Sigma <- Semidef(n)
obj <- quad_form(w, Sigma)
constr <- list(Sigma[1,1] == 0.2, Sigma[1,2] >= 0, Sigma[1,3] >= 0,
               Sigma[2,2] == 0.1, Sigma[2,3] <= 0, Sigma[2,4] <= 0,
               Sigma[3,3] == 0.3, Sigma[3,4] >= 0, Sigma[4,4] == 0.1)
prob <- Problem(Maximize(obj), constr)
result <- solve(prob)
```

It is necessary to specify a quadratic form with `quad_form` rather than the usual `t(w) %*% Sigma %*% w` because the latter will be interpreted by the **CVXR** parser as a product of two affine terms and rejected for not being DCP. Our result for this numerical case is

$$\Sigma = \begin{bmatrix} 0.2000 & 0.0973 & 0.0006 & 0.0743 \\ 0.0973 & 0.1000 & -0.1012 & 0.0000 \\ 0.0006 & -0.1012 & 0.3000 & 0.0005 \\ 0.0743 & 0.0000 & 0.0005 & 0.1000 \end{bmatrix}$$

This example can be generalized to include arbitrary convex constraints on Σ. Furthermore, if we have a target estimate for the covariance, we can bound deviations from the target by incorporating penalized slack variables into our optimization problem.

Catenary Problem

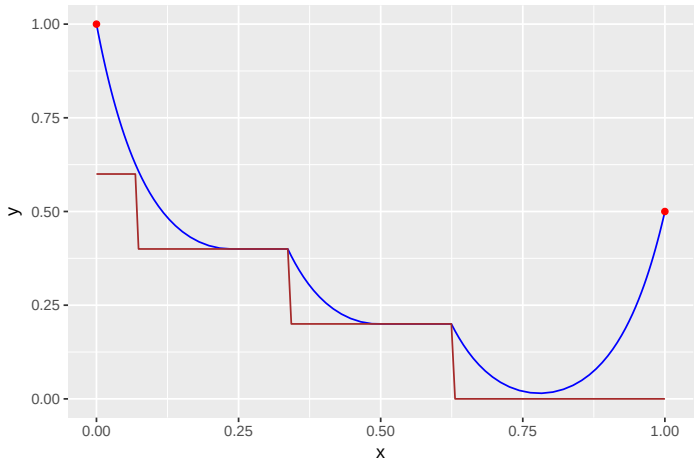

Figure 6: Solution of the catenary problem (blue) with a ground constraint (brown).

We consider a discretized version of the catenary problem in Griva and Vanderbei (2005). A chain with uniformly distributed mass hangs from the endpoints $(0, 1)$ and $(1, 1)$ on a 2-D plane. Gravitational force acts in the negative y direction. Our goal is to find the shape of the chain in equilibrium, which is equivalent to determining the (x, y) coordinates of every point along its curve when its potential energy is minimized.

To formulate this as an optimization problem, we parameterize the chain by its arclength and divide it into m discrete links. The length of each link must be no more than $h > 0$. Since mass is uniform, the total potential energy is simply the sum of the y-coordinates. Therefore, our problem is

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & \sum_{i=1}^{m} y_i \\ \text{subject to} & x_1 = 0, \quad y_1 = 1, \quad x_m = 1, \quad y_m = 1 \\ & (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 \leq h^2, \quad i = 1, \ldots, m-1 \end{array}$$

with variables $x \in \mathbf{R}^m$ and $y \in \mathbf{R}^m$. This basic catenary problem has a well-known analytical solution (Gelfand and Fomin 1963), which we can easily verify with **CVXR**.

```
x <- Variable(m)
y <- Variable(m)
obj <- sum(y)
constr <- c(x[1] == 0, y[1] == 0, x[m] == 1, y[m] == 1,
            diff(x)^2 + diff(y)^2 <= h^2)
prob <- Problem(Minimize(obj), constr)
result <- solve(prob)
```

A more interesting situation arises when the ground is not flat. Let $g \in \mathbf{R}^m$ be the elevation vector (relative to the x-axis), and suppose the right endpoint of our chain has been lowered by $\Delta y_m = 0.5$. The analytical solution in this case would be difficult to calculate. However, we need only add two lines to our constraint definition,

```
constr[[4]] <- (y[m] == 0.5)
constr <- c(constr, y >= g)
```

to obtain the new result. Figure 6 depicts the solution of this modified catenary problem for $m = 101$ and $h = 0.04$. The chain is shown hanging in blue, bounded below by the red staircase structure, which represents the ground.

Portfolio Optimization

In this example, we solve the Markowitz portfolio problem under various constraints (Markowitz 1952; Roy 1952; Lobo, Fazel, and Boyd 2007). We have n assets or stocks in our portfolio and must determine the amount of money to invest in each. Let w_i denote the fraction of our budget invested in asset $i = 1, \ldots, m$, and let r_i be the returns (*i.e.*, fractional change in price) over the period of interest. We model returns as a random vector $r \in \mathbf{R}^n$ with known mean $\mathbf{E}[r] = \mu$ and covariance $\mathbf{Var}(r) = \Sigma$. Thus, given a portfolio $w \in \mathbf{R}^n$, the overall return is $R = r^T w$.

Portfolio optimization involves a trade-off between the expected return $\mathbf{E}[R] = \mu^T w$ and associated risk, which we take as the return variance $\mathbf{Var}(R) = w^T \Sigma w$. Initially, we consider only long portfolios, so our problem is

$$\begin{array}{ll} \underset{w}{\text{maximize}} & \mu^T w - \gamma w^T \Sigma w \\ \text{subject to} & w \geq 0, \quad \sum_{i=1}^n w = 1 \end{array}$$

where the objective is the risk-adjusted return and $\gamma > 0$ is a risk aversion parameter.

```
w <- Variable(n)
ret <- t(mu) %*% w
risk <- quad_form(w, Sigma)
obj <- ret - gamma * risk
constr <- list(w >= 0, sum(w) == 1)
prob <- Problem(Maximize(obj), constr)
result <- solve(prob)
```

We can obtain the risk and return by directly evaluating the value of the separate expressions:

```
result$getValue(risk)
result$getValue(ret)
```

Figure 7a depicts the risk-return trade-off curve for $n = 10$ assets and μ and $\Sigma^{1/2}$ drawn from a standard normal distribution. The x-axis represents the standard deviation of the return. Red points indicate the result from investing the entire budget in a single asset. As γ increases, our portfolio becomes more diverse (Figure 7b), reducing risk but also yielding a lower return.

Many variations on the classical portfolio problem exist. For instance, we could allow long and short positions, but impose a leverage limit $\|w\|_1 \leq L^{max}$ by changing

```
constr <- list(p_norm(w,1) <= Lmax, sum(w) == 1)
```

An alternative is to set a lower bound on the return and minimize just the risk. To account for transaction costs, we could add a term to the objective that penalizes deviations of w from the previous portfolio. These extensions and more are described in Boyd, Busseti, Diamond,

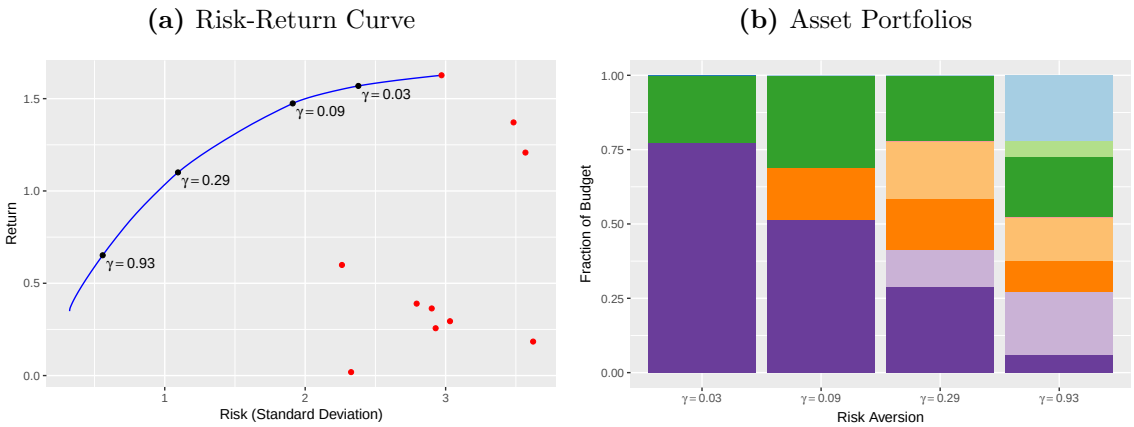

Figure 7: (a) Risk-return trade-off curve for various γ. Portfolios that invest completely in one asset are plotted in red. (b) Fraction of budget invested in each asset.

Kahn, Koh, Nystrup, and Speth (2017). The key takeaway is that all of these convex problems can be easily solved in **CVXR** with just a few alterations to the code above.

Kelly Gambling

In Kelly gambling (Kelly 1956), we are given the opportunity to bet on n possible outcomes, which yield a random non-negative return of $r \in \mathbf{R}^n_+$. The return r takes on exactly K values $r_1, \ldots, r_K$ with known probabilities $\pi_1, \ldots, \pi_K$. This gamble is repeated over T periods. In a given period t, let $b_i \geq 0$ denote the fraction of our wealth bet on outcome i. Assuming the nth outcome is equivalent to not wagering (it returns one with certainty), the fractions must satisfy $\sum_{i=1}^n b_i = 1$. Thus, at the end of the period, our cumulative wealth is $w_t = (r^T b) w_{t-1}$. Our goal is to maximize the average growth rate with respect to $b \in \mathbf{R}^n$:

$$\begin{array}{ll} \underset{b}{\text{maximize}} & \sum_{j=1}^K \pi_j \log(r_j^T b) \\ \text{subject to} & b \geq 0, \quad \sum_{i=1}^n b_i = 1. \end{array}$$

In the following code, `rets` is the $K \times n$ matrix of possible returns with row r_i, while `ps` is the vector of return probabilities $(\pi_1, \ldots, \pi_K)$.

```
b <- Variable(n)
obj <- t(ps) %*% log(rets %*% b)
constr <- list(b >= 0, sum(b) == 1)
prob <- Problem(Maximize(obj), constr)
result <- solve(prob)
```

We solve the Kelly gambling problem for $K = 100$ and $n = 20$. The probabilities $\pi_j \sim \text{Uniform}(0, 1)$, and the potential returns $r_j \sim \text{Uniform}(0.5, 1.5)$ except for $r_n = \mathbf{1}$, which represents the payoff from not wagering. With an initial wealth of $w_0 = 1$, we simulate the growth trajectory of our Kelly optimal bets over $P = 100$ periods, assuming returns are i.i.d. over time.

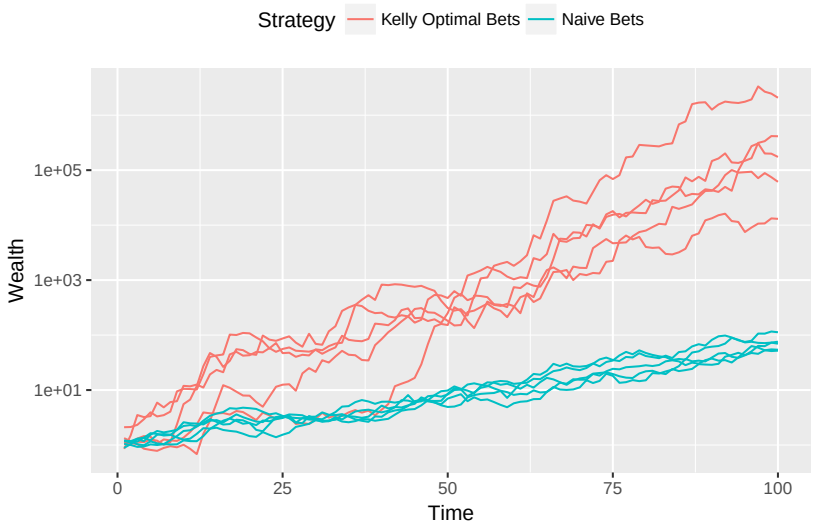

Figure 8: Wealth trajectories for the Kelly optimal bets (red) and naïve bets (cyan). The naïve betting scheme holds onto 15% of the wealth and splits the rest in direct proportion to the expected returns.

```
bets <- result$getValue(b)
idx <- sample.int(K, size = P, probs = ps, replace = TRUE)
winnings <- rets[idx,] %*% bets
wealth <- w0 * cumprod(winnings)
```

For comparison, we also calculate the trajectory for a naïve betting scheme, which holds onto 15% of the wealth at the beginning of each period and divides the other 85% over the bets in direct proportion to their expect returns.

Growth curves for five independent trials are plotted in Figure 8. Red lines represent the wealth each period from the Kelly bets, while cyan lines are the result of the naïve bets. Clearly, Kelly optimal bets perform better, producing greater net wealth by the final period. However, as observed in some trajectories, wealth tends to drop by a significant amount before increasing eventually. One way to reduce this drawdown risk is to add a convex constraint as proposed in Busseti, Ryu, and Boyd (2016, §5.3),

$$\log\left(\sum_{j=1}^{K}\exp(\log\pi_j - \lambda\log(r_j^T b))\right) \leq 0,$$

where $\lambda \geq 0$ is the risk-aversion parameter. With **CVXR**, this can be accomplished in a single line using the `log_sum_exp` atom. Other extensions like wealth goals, betting restrictions, and VaR/CVaR bounds are also readily incorporated.

Channel Capacity

The following problem comes from an exercise in Boyd and Vandenberghe (2004, pp. 207–208). Consider a discrete memoryless communication channel with input $X(t) \in \{1, \ldots, n\}$

and output $Y(t) \in \{1, \ldots, m\}$ for $t = 1, 2, \ldots$. The relation between the input and output is given by a transition matrix $P \in \mathbf{R}_+^{m \times n}$ with

$$P_{ij} = \mathsf{P}(Y(t) = i | X(t) = j), \quad i = 1, \ldots, m, \quad j = 1, \ldots, n$$

Assume that X has a probability distribution denoted by $x \in \mathbf{R}^n$, *i.e.*, $x_j = \mathsf{P}(X(t) = j)$ for $j = 1, \ldots, n$. A famous result by Shannon and Weaver (1949) states that the channel capacity is found by maximizing the mutual information between X and Y,

$$I(X, Y) = \sum_{j=1}^{n} x_j \sum_{i=1}^{m} P_{ij} \log_2 P_{ij} - \sum_{i=1}^{m} y_i \log_2 y_i.$$

where $y = Px$ is the probability distribution of Y. Since I is concave, this is equivalent to solving the convex optimization problem

$$\begin{array}{ll} \underset{x,y}{\text{maximize}} & \sum_{j=1}^{n} x_j \sum_{i=1}^{m} P_{ij} \log P_{ij} - \sum_{i=1}^{m} y_i \log y_i \\ \text{subject to} & x \geq 0, \quad \sum_{i=1}^{m} x_i = 1, \quad y = Px \end{array}$$

for $x \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$. The associated code in **CVXR** is

```
x <- Variable(n)
y <- P %*% x
c <- apply(P * log2(P), 2, sum)
obj <- c %*% x + sum(entr(y))
constr <- list(sum(x) == 1, x >= 0)
prob <- Problem(Maximize(obj), constr)
result <- solve(prob)
```

The channel capacity is simply the optimal objective, `result$value`.

Fastest Mixing Markov Chain

This example is derived from the results in Boyd, Diaconis, and Xiao (2004, §2). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected graph with vertices $\mathcal{V} = \{1, \ldots, n\}$ and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Assume that $(i, i) \in \mathcal{E}$ for all $i = 1, \ldots, n$, and $(i, j) \in \mathcal{E}$ implies $(j, i) \in \mathcal{E}$. Under these conditions, a discrete-time Markov chain on $\mathcal{V}$ will have the uniform distribution as one of its equilibrium distributions. We are interested in finding the Markov chain, *i.e.*constructing the transition probability matrix $P \in \mathbf{R}_+^{n \times n}$, that minimizes its asymptotic convergence rate to the uniform distribution. This is an important problem in Markov chain Monte Carlo (MCMC) simulations, as it directly affects the sampling efficiency of an algorithm.

The asymptotic rate of convergence is determined by the second largest eigenvalue of P, which in our case is $\mu(P) := \lambda_{\max}(P - \frac{1}{n}\mathbf{1}\mathbf{1}^T)$ where $\lambda_{\max}(A)$ denotes the maximum eigenvalue of A. As $\mu(P)$ decreases, the mixing rate increases and the Markov chain converges faster to equilibrium. Thus, our optimization problem is

$$\begin{array}{ll} \underset{P}{\text{minimize}} & \lambda_{\max}(P - \frac{1}{n}\mathbf{1}\mathbf{1}^T) \\ \text{subject to} & P \geq 0, \quad P\mathbf{1} = \mathbf{1}, \quad P = P^T \\ & P_{ij} = 0, \quad (i, j) \notin \mathcal{E}. \end{array}$$

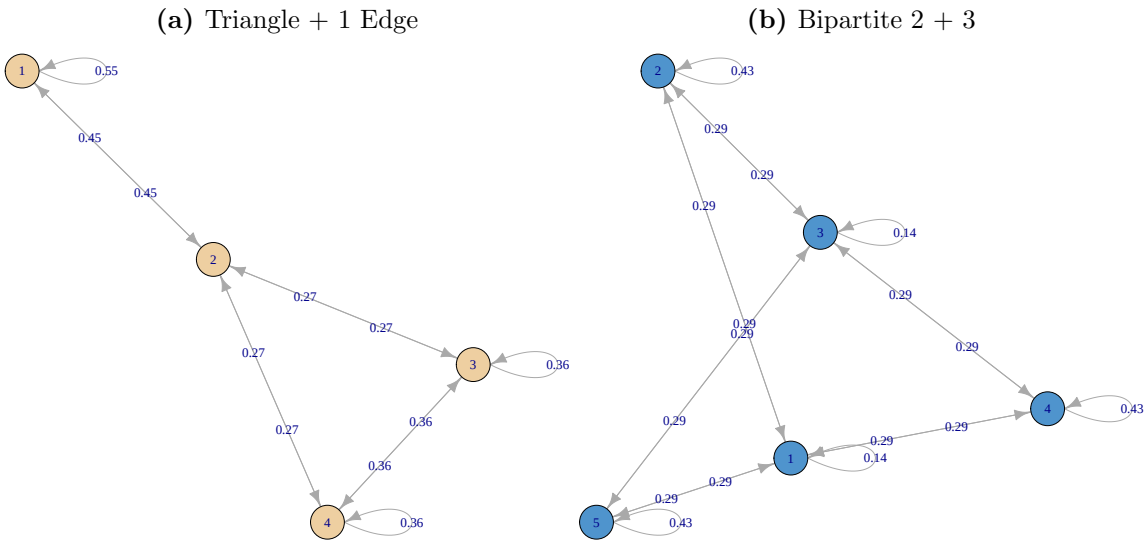

Figure 9: Markov chains with transition probabilities that achieve the fastest mixing rate.

The element P_{ij} of our transition matrix is the probability of moving from state i to state j. Our assumptions imply that P is nonnegative, symmetric, and doubly stochastic. The last constraint ensures transitions do not occur between unconnected vertices.

The function $\lambda_{\max}$ is convex, so this problem is solvable in **CVXR**. For instance, the code for the Markov chain in Figure 9a is

```
P <- Variable(n,n)
ones <- matrix(1, nrow = n, ncol = 1)

obj <- Minimize(lambda_max(P - 1/n))
constr1 <- list(P >= 0, P %*% ones == ones, P == t(P))
constr2 <- list(P[1,3] == 0, P[1,4] == 0)
prob <- Problem(obj, c(constr1, constr2))
result <- solve(prob)
```

where we have set $n = 4$. We could also have specified $P\mathbf{1} = \mathbf{1}$ with `sum_entries(P,1) == 1`, which uses the `sum_entries` atom to represent the row sums.

It is easy to extend this example to other Markov chains. To change the number of vertices, we would simply modify `n`, and to add or remove edges, we need only alter the constraints in `constr2`. For instance, the bipartite chain in Figure 9b is produced by setting $n = 5$ and

```
constr2 <- list(P[1,3] == 0, P[2,4] == 0, P[2,5] == 0, P[4,5] == 0)
```

4. Implementation

CVXR represents the atoms, variables, constraints, and other parts of an optimization problem using S4 class objects. S4 enables us to overload standard mathematical operations so

CVXR combines seamlessly with native R script and other packages. When an operation is invoked on a variable, a new object is created that represents the corresponding expression tree with the operator as the root node and the arguments as leaves. This tree grows automatically as more elements are added, allowing us to encapsulate the structure of an objective function or constraint.

Once the user calls `solve`, DCP verification occurs. **CVXR** traverses the expression tree recursively, determining the sign and curvature of each sub-expression based on the properties of its component atoms. If the problem is deemed compliant, it is transformed into an equivalent cone program using graph implementations of convex functions (Grant *et al.* 2006). Then, **CVXR** passes the problem's description to the CVXcanon C++ library, which generates data for the cone program, and sends this data to the solver-specific R interface. The solver's results are returned to the user in a list. This object-oriented design and infrastructure were largely borrowed from **CVXPY**. Currently, the canonicalization and construction of data in R for the solver dominates computation time.

CVXR interfaces with the open-source cone solvers **ECOS** (Domahidi, Chu, and Boyd 2013) and **SCS** (O'Donoghue, Chu, Parikh, and Boyd 2016) through their respective R packages. **ECOS** is an interior-point solver, which achieves high accuracy for small and medium-sized problems, while **SCS** is a first-order solver that is capable of handling larger problems and semidefinite constraints. Both solvers run single-threaded at present. Support for multi-threaded **SCS** will be added in the future, along with solvers like **MOSEK** (Andersen and Andersen 2000), **GUROBI** (Gurobi Optimization, Inc 2016), etc. It is not difficult to connect additional solvers so long as the solver has an API that can communicate with R. Users who wish to employ a custom solver may obtain the canonicalized data directly with `get_problem_data`.

We have provided a rich library of atoms, which should be sufficient to model most convex optimization problems. However, it is possible for a sophisticated user to incorporate new atoms into this library. The process entails creating a S4 class for the atom, overloading methods that characterize its DCP properties, and representing its graph implementation as a list of linear operators that specify the corresponding feasibility problem. A full mathematical exposition may be found in Grant *et al.* (2006, §10). Many problems can be reformulated to comply with DCP by defining new variables and constraints as illustrated in Boyd and Vandenberghe (2004). In general, we suggest users try this approach first before attempting to add a novel atom.

5. Conclusion

Convex optimization plays an essential role in many fields, particularly machine learning and statistics. **CVXR** provides an object-oriented language with which users can easily formulate, modify, and solve a broad range of convex optimization problems. While other R packages may perform faster on a subset of these problems, **CVXR**'s advantage is its flexibility and simple intuitive syntax, making it an ideal tool for prototyping new models for which custom R code does not exist. For more information, see the official CRAN site and documentation.

Acknowledgements

The authors would like to thank Trevor Hastie, Robert Tibshirani, John Chambers, David Donoho, and Hans Werner Borchers for their thoughtful advice and comments on this project. We are grateful to Steven Diamond, John Miller, and Paul Kunsberg Rosenfield for their contributions to the software's development. In particular, we are indebted to Steven for his work on **CVXPY**. Most of **CVXR**'s code, documentation, and examples were ported from his Python library.

Anqi Fu's research was supported by the Stanford Graduate Fellowship and DARPA X-DATA program. Balasubramanian Narasimhan's work was supported by the Clinical and Translational Science Award 1UL1 RR025744 for the Stanford Center for Clinical and Translational Education and Research (Spectrum) from the National Center for Research Resources, National Institutes of Health.

References

Andersen ED, Andersen KD (2000). "The MOSEK Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm." In *High Performance Optimization*, pp. 197–232. Springer.

Andersen M, Dahl J, Liu Z, Vandenberghe L (2011). "Interior-Point Methods for Large-Scale Cone Programming." *Optimization for machine learning*, **5583**.

Boyd N, Hastie T, Boyd S, Recht B, Jordan M (2016). "Saturating Splines and Feature Selection." *arXiv preprint arXiv:1609.06764*.

Boyd S, Busseti E, Diamond S, Kahn RN, Koh K, Nystrup P, Speth J (2017). "Multi-Period Trading via Convex Optimization." *Foundations and Trends in Optimization*.

Boyd S, Diaconis P, Xiao L (2004). "Fastest Mixing Markov Chain on a Graph." *SIAM Review*, **46**(4), 667–689.

Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers." *Foundations and Trends in Machine Learning*, **3**(1), 1–122.

Boyd S, Vandenberghe L (2004). *Convex Optimization*. Cambridge University Press.

Busseti E, Ryu EK, Boyd S (2016). "Risk–Constrained Kelly Gambling." *Journal of Investing*, **25**(3), 118–134.

Cox DR (1958). "The Regression Analysis of Binary Sequences." *Journal of the Royal Statistical Society. Series B (Methodological)*, **20**(2), 215–242.

Diamond S, Boyd S (2016). "CVXPY: A Python-Embedded Modeling Language for Convex Optimization." *Journal of Machine Learning Research*, **17**(83), 1–5.

Domahidi A, Chu E, Boyd S (2013). "ECOS: An SOCP Solver for Embedded Systems." In *Proceedings of the European Control Conference*, pp. 3071–3076.

Dümbgen L, Rufibach K (2009). "Maximum Likelihood Estimation of a Log-Concave Density and its Distribution Function: Basic Properties and Uniform Consistency." *Bernoulli*, **15**(1), 40–68.

Fleiss JL, Levin B, Paik MC (2003). *Statistical Methods for Rates and Proportions*. Wiley-Interscience.

Freedman DA (2009). *Statistical Models: Theory and Practice*. Cambridge University Press.

Friedman J, Hastie T, Tibshirani R (2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso." *Biostatistics*, **9**(3), 432–441.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22.

Gelfand IM, Fomin SV (1963). *Calculus of Variations*. Prentice-Hall.

Grant M, Boyd S (2014). "CVX: MATLAB Software for Disciplined Convex Programming, Version 2.1." http://cvxr.com/cvx.

Grant M, Boyd S, Ye Y (2006). "Disciplined Convex Programming." In L Liberti, N Maculan (eds.), *Global Optimization: From Theory to Implementation*, Nonconvex Optimization and its Applications, pp. 155–210. Springer.

Griva IA, Vanderbei RJ (2005). "Case Studies in Optimization: Catenary problem." *Optimization and Engineering*, **6**(4), 463–482.

Gurobi Optimization, Inc (2016). *Gurobi Optimizer Reference Manual*. http://www.gurobi.com.

H Zou TH (2005). "Regularization and Variable Selection via the Elastic–Net." *Journal of the Royal Statistical Society. Series B (Methodological)*, **67**(2), 301–320.

Hastie T, Tibshirani R, Friedman J (2001). *The Elements of Statistical Learning*. Springer.

Hornik K, Meyer D, Schwendinger F, Theussl S (2017). *ROI: R Optimization Infrastructure*. https://CRAN.R-project.org/package=ROI.

Huber PJ (1964). "Robust Estimation of a Location Parameter." *Annals of Mathematical Statistics*, **35**(1), 73–101.

Johnson SG (2008). *The NLopt Nonlinear-Optimization Package*. http://ab-initio.mit.edu/nlopt/.

Kelly JL (1956). "A New Interpretation of Information Rate." *Bell System Technical Journal*, **35**(4), 917–926.

Koenker R (2005). *Quantile Regression*. Cambridge University Press.

Lobo M, Boyd S (2000). "The Worst-Case Risk of a Portfolio." http://stanford.edu/~boyd/papers/pdf/risk_bnd.pdf.

Lobo MS, Fazel M, Boyd S (2007). "Portfolio Optimization with Linear and Fixed Transaction Costs." *Annals of Operations Research*, **152**(1), 341–365.

Lofberg J (2004). "YALMIP: A Toolbox for Modeling and Optimization in MATLAB." In *Proceedings of the IEEE International Symposium on Computed Aided Control Systems Design*, pp. 294–289. http://users.isy.liu.se/johanl/yalmip.

Markowitz HM (1952). "Portfolio Selection." *Journal of Finance*, **7**(1), 77–91.

Nash JC, Varadhan R (2011). "Unifying Optimization Algorithms to Aid Software System Users: optimx for R." *Journal of Statistical Software*, **43**(9), 1–14.

O'Donoghue B, Chu E, Parikh N, Boyd S (2016). "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding." *Journal of Optimization Theory and Applications*, pp. 1–27.

Roy AD (1952). "Safety First and the Holding of Assets." *Econometrica*, **20**(3), 431–449.

Shannon CE, Weaver W (1949). *The Mathematical Theory of Communication*. University of Illinois Press.

Skajaa A, Ye Y (2015). "A Homogeneous Interior-Point Algorithm for Nonsymmetric Convex Conic Optimization." *Mathematical Programming*, **150**(2), 391–422.

Tibshirani RJ, Hoefling H, Tibshirani R (2011). "Nearly-Isotonic Regression." *Technometrics*, **53**(1), 54–61.

Udell M, Mohan K, Zeng D, Hong J, Diamond S, Boyd S (2014). "Convex Optimization in Julia." In *Proceedings of the Workshop for High Performance Technical Computing in Dynamic Languages*, pp. 18–28.

Wright SJ (1997). *Primal-Dual Interior-Point Methods*. SIAM.

Affiliation:

Anqi Fu
Department of Electrical Engineering
David Packard Building
350 Serra Mall
Stanford, CA 94305
E-mail: anqif@stanford.edu
URL: https://web.stanford.edu/~anqif/

Balasubramanian Narasimhan
Department of Biomedical Data Sciences, and
Department of Statistics
Stanford University
390 Serra Mall
Stanford, CA 94305
E-mail: naras@stanford.edu
URL: https://statistics.stanford.edu/~naras/

Stephen Boyd
Department of Electrical Engineering
David Packard Building
350 Serra Mall
Stanford, CA 94305
E-mail: boyd@stanford.edu
URL: https://web.stanford.edu/~boyd/