

Final Exam

This is a 24 hour take-home final. Please turn it in at Bytes Cafe in the Packard building, 24 hours after you pick it up.

You may use any books, notes, or computer programs, but you may not discuss the exam with anyone until 5PM June 9, after everyone has taken the exam. The only exception is that you can ask us for clarification, via the course staff email address. We've tried pretty hard to make the exam unambiguous and clear, so we're unlikely to say much.

Please make a copy of your exam, or scan it, before handing it in.

Please attach the cover page to the front of your exam. Assemble your solutions in order (problem 1, problem 2, problem 3, ...), starting a new page for each problem. Put everything associated with each problem (*e.g.*, text, code, plots) together; do not attach code or plots at the end of the final.

We will deduct points from long, needlessly complex solutions, even if they are correct. Our solutions are not long, so if you find that your solution to a problem goes on and on for many pages, you should try to figure out a simpler one. We expect neat, legible exams from everyone, including those enrolled Cr/N.

When a problem involves computation you must give all of the following: a clear discussion and justification of exactly what you did, the source code that produces the result, and the final numerical results or plots.

Files containing problem data can be found in the usual place,

http://www.stanford.edu/~boyd/cvxbook/cvxbook_additional_exercises/

Please respect the honor code. Although we allow you to work on homework assignments in small groups, you cannot discuss the final with anyone, at least until everyone has taken it.

All problems have equal weight. Some are (quite) straightforward. Others, not so much.

Be sure you are using the most recent version of CVX, CVXPY, or Convex.jl. Check your email often during the exam, just in case we need to send out an important announcement.

Some problems involve applications. But you do not need to know *anything* about the problem area to solve the problem; the problem statement contains everything you need.

Some of the data files generate random data (with a fixed seed), which are not necessarily the same for Matlab, Python, and Julia.

1. *Transforming to a normal distribution.* We are given n samples $x_i \in \mathbf{R}$ from an unknown distribution. We seek an increasing piecewise-affine function $\varphi : \mathbf{R} \rightarrow \mathbf{R}$ for which $y_i = \varphi(x_i)$ has a distribution close to $\mathcal{N}(0, 1)$. In other words, the nonlinear transformation $x \mapsto y = \varphi(x)$ (approximately) transforms the given distribution to a standard normal distribution.

You can assume that the samples are distinct and sorted, *i.e.*, $x_1 < x_2 < \dots < x_n$, and therefore we also have $y_1 < y_2 < \dots < y_n$. The empirical CDF (cumulative distribution function) of y_i is the piecewise-constant function $F : \mathbf{R} \rightarrow \mathbf{R}$ given by

$$F(z) = \begin{cases} 0 & z < y_1, \\ k/n & y_k \leq z < y_{k+1}, \quad k = 1, \dots, n-1, \\ 1 & z \geq y_n. \end{cases}$$

The *Kolmogorov-Smirnov* distance between the empirical distribution of y_i and the standard normal distribution is given by

$$D = \sup_z |F(z) - \Phi(z)|,$$

where Φ is the CDF of an $\mathcal{N}(0, 1)$ random variable. We will use D as our measure of how close the transformed distribution is to normal. Note that D can be as small as $1/(2n)$ (but no smaller), by choosing $y_i = \Phi^{-1}((i - 1/2)/n)$.

Note that D only depends on the n numbers y_1, \dots, y_n . From these numbers we extend φ to a function on \mathbf{R} using linear interpolation between these values, and extending outside the interval $[x_1, x_n]$ using the same slopes as the first and last segments, respectively. So y_1, \dots, y_n determine φ .

Our regularization (measure of complexity) of φ is

$$R = \sum_{i=2}^{n-1} \left| \frac{y_{i+1} - y_i}{x_{i+1} - x_i} - \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \right|.$$

This is the sum of the absolute values of the change in slope of φ . Note that $R = 0$ if and only if φ has no kinks, *i.e.*, is affine.

We will choose y_i (which defines φ) by minimizing R , subject to $D \leq D^{\max}$, where $D^{\max} \geq 1/(2n)$ is a parameter. It can be shown that the condition $y_i < y_{i+1}$ will hold automatically; but if you are nervous about this, you are welcome to add the constraint $y_i + \epsilon \leq y_{i+1}$, where ϵ is a small positive number.

- (a) Explain how to solve this problem using convex or quasiconvex optimization. If your formulation involves a change of variables or other transformation, justify it.
- (b) The file `transform_to_normal_data.*` contains the vector x (in sorted order) and its length n . Use the method of part (a) to find the optimal φ (*i.e.*, y) for $D^{\max} = 0.05$. Plot the empirical CDF of the original data x and the normal CDF

Φ on one plot, the empirical CDF of the transformed data y and the normal CDF Φ on another plot, and the optimal transformation φ on a third plot. Report the optimal value of R .

Hints. In Python and Julia, you should use the (default) ECOS solver to avoid warnings about inaccurate solutions. You can evaluate the normal CDF Φ using `normcdf.m/norminv.m` (Matlab), `scipy.stats.norm.cdf/ppf` (Python), or `normcdf/norminvcdf` in `StatsFuns.jl` (Julia). To plot the empirical CDFs of x and y , you are welcome to use the basic plot functions, which connect adjacent points with lines. But if you'd like to create step function style plots, you can use `ecdf.m` (Matlab), `matplotlib.pyplot.step` (Python), or `step` in `PyPlot.jl` (Julia).

2. *Inverse of product.* The function $f(x, y) = 1/(xy)$ with $x, y \in \mathbf{R}$, $\mathbf{dom} f = \mathbf{R}_{++}^2$, is convex. How do we represent it using disciplined convex programming (DCP), and the functions $1/u$, \sqrt{uv} , \sqrt{u} , u^2 , u^2/v , addition, subtraction, and scalar multiplication? (These functions have the obvious domains, and you can assume a sign-sensitive version of DCP, *e.g.*, u^2/v increasing in u for $u \geq 0$.) *Hint.* There are several ways to represent f using the atoms given above.

3. *Path planning with contingencies.* A vehicle path down a (straight, for simplicity) road is specified by a vector $p \in \mathbf{R}^N$, where p_i gives the position perpendicular to the centerline at the point ih meters down the road, where $h > 0$ is a given discretization size. (Throughout this problem, indexes on N -vectors will correspond to positions on the road.) We normalize p so $-1 \leq p_i \leq 1$ gives the road boundaries. (We are modeling the vehicle as a point, by adjusting for its width.) You are given the initial two positions $p_1 = a$ and $p_2 = b$ (which give the initial road position and angle), as well as the final two positions $p_{N-1} = c$ and $p_N = d$.

You know there may be an obstruction at position $i = O$. This will require the path to either go around the obstruction on the left, which requires $p_O \geq 0.5$, or on the right, which requires $p_O \leq -0.5$, or possibly the obstruction will clear, and the obstruction does not place any additional constraint on the path. These are the three *contingencies* in the problem title, which we label as $k = 1, 2, 3$.

You will plan three paths for these contingencies, $p^{(i)} \in \mathbf{R}^N$ for $i = 1, 2, 3$. They must each satisfy the given initial and final two road positions and the constraint of staying within the road boundaries. Paths $p^{(1)}$ and $p^{(2)}$ must satisfy the (different) obstacle avoidance constraints given above. Path $p^{(3)}$ does not need to satisfy an avoidance constraint.

Now we add a twist: You will not learn which of the three contingencies will occur until the vehicle arrives at position $i = S$, when the sensors will determine which contingency holds. We model this with the *information constraints* (also called *causality constraints* or *non-anticipatory constraints*),

$$p_i^{(1)} = p_i^{(2)} = p_i^{(3)}, \quad i = 1, \dots, S,$$

which state that before you know which contingency holds, the three paths must be the same.

The objective to be minimized is

$$\sum_{k=1}^3 \sum_{i=2}^{N-1} (p_{i-1}^{(k)} - 2p_i^{(k)} + p_{i+1}^{(k)})^2,$$

the sum of the squares of the second differences, which gives smooth paths.

- (a) Explain how to solve this problem using convex optimization.
- (b) Solve the problem with data given in `path_plan_contingencies_data.*`. The data files include code to plot the results, which you should use to plot (on one plot) the optimal paths. Report the optimal objective value. Give a *very brief* informal explanation for what you see happening for $i = 1, \dots, S$.

Hint. In Python, use the (default) solver ECOS to avoid warnings about inaccurate solutions.

4. *Total variation de-mosaicing.* A color image is represented by 3 $m \times n$ matrices R , G , and B that give the red, green, and blue pixel intensities. A camera sensor, however, measures only *one* of the color intensities at each pixel. The pattern of pixel sensor colors varies, but most of the patterns have twice as many green sensor pixels as red or blue. A common arrangement repeats the 2×2 block

$$\begin{array}{cc} R & G \\ G & B \end{array}$$

(assuming m and n are even).

De-mosaicing is the process of guessing, or interpolating, the missing color values at each pixel. The sensors give us mn entries in the matrices R , G , and B ; in de-mosaicing, we guess the remaining $2mn$ entries in the matrices.

First we describe a very basic method of de-mosaicing. For each 2×2 block of pixels we have the 4 intensity values

$$\begin{array}{cc} R_{i,j} & G_{i,j+1} \\ G_{i+1,j} & B_{i+1,j+1} \end{array}.$$

We use the value $R_{i,j}$ as the red value for the other three pixels, and we do the same for the blue value $B_{i+1,j+1}$. For guessing the green values at i, j and $i + 1, j + 1$, we simply use the average of the two measured green values, $(G_{i,j+1} + G_{i+1,j})/2$.

A more sophisticated method relies on convex optimization. You choose the unknown pixel values in R , G , and B to minimize the total variation of the color image, defined as

$$\sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \left\| \begin{bmatrix} R_{i,j} - R_{i,j+1} \\ G_{i,j} - G_{i,j+1} \\ B_{i,j} - B_{i,j+1} \\ R_{i+1,j} - R_{i,j} \\ G_{i+1,j} - G_{i,j} \\ B_{i+1,j} - B_{i,j} \end{bmatrix} \right\|_2.$$

Note that the norms in the sum here are *not* squared. The argument of the norms is a vector in \mathbf{R}^6 , an estimate of the spatial gradient of the RGB values.

We have provided you with several files in the data directory. Three images are given (in png format): `demosaic_raw.png`, which contains the raw or mosaic image to de-mosaic, `demosaic_original.png`, which contains the original image from which the raw image was constructed, and `demosaic_simple.png`, which is the image de-mosaiced by the simple method described above. Remember that the raw image, and any reconstructed de-mosaiced image, have only one third the information of the original, so we cannot expect them to look as good as the original. You don't need the original or basic de-mosaiced image files to solve the problem; they are given only so you can look at them to see what they are. You should zoom in while viewing the raw image and the

basic de-mosaic version, so you can see the pattern of 2×2 blocks in the first, and the simple de-mosaic method in the second.

The `tv` function, invoked as `tv(R,G,B)`, gives the total variation. CVXPY has the `tv` function built-in, but CVX and CVX.jl do not, so we have provided the files `tv.m` and `tv.jl` which contain implementations for you to use.

The file `demosaic_data.*` constructs arrays `R_mask`, `G_mask`, and `B_mask`, which contain the indices of pixels whose values we know in the original image, the number of rows and columns in the image, m, n respectively, and arrays `R_raw`, `B_raw`, `G_raw`, which contain the known values of each color at each pixel, filled in with zeroes for the unknown values. So if `R` is an $m \times n$ matrix variable, the constraint `R[R_mask]==R_raw[R_mask]` in Julia and Python will impose the constraint that it agrees with the given red pixel values; in Matlab, the constraint can be expressed as `R(R_mask)==R_raw(R_mask)`. This file also contains a `save_image` method, which takes three arguments, `R`, `G`, `B` arrays (that you've reconstructed) and saves the file under the name `output_image.png`. To see the image in Matlab, use the `imshow` function.

Report the optimal value of total variation, and attach the de-mosaiced image. (If you don't have access to a color printer, you can submit a monochrome version. Print it large enough that we can see it, say, at least half the page width wide.)

Hint. Your solution code should take less than 10 seconds or so to run in Python and Matlab, but up to a minute or so in Julia. You might get a warning about an inaccurate solution, but you can ignore it.

5. *Maximum Sharpe ratio portfolio.* We consider a portfolio optimization problem with portfolio vector $x \in \mathbf{R}^n$, mean return $\mu \in \mathbf{R}^n$, and return covariance $\Sigma \in \mathbf{S}_{++}^n$. The ratio of portfolio mean return $\mu^T x$ to portfolio standard deviation $\|\Sigma^{1/2}x\|_2$ is called the *Sharpe ratio* of the portfolio. (It is often modified by subtracting a risk-free return from the mean return.) The Sharpe ratio measures how much return you get per risk taken on, and is a widely used single metric that combines return and risk. It is undefined for $\mu^T x \leq 0$.

Consider the problem of choosing the portfolio to maximize the Sharpe ratio, subject to the constraint $\mathbf{1}^T x = 1$, and the leverage constraint $\|x\|_1 \leq L^{\max}$, where $L^{\max} \geq 1$ is a given leverage limit. You can assume there is a feasible x with $\mu^T x > 0$.

- (a) Show that the maximum Sharpe ratio problem is quasiconvex in the variable x .
- (b) Show how to solve the maximum Sharpe ratio problem by solving *one* convex optimization problem. You must fully justify any change of variables or problem transformation.

6. *Optimizing a set of disks.* A disk $D \subset \mathbf{R}^2$ is parametrized by its center $c \in \mathbf{R}^2$ and its radius $r \geq 0$, with the form $D = \{x \mid \|x - c\|_2 \leq r\}$. (We allow $r = 0$, in which case the disk reduces to a single point $\{c\}$.) The goal is to choose a set of n disks D_1, \dots, D_n (*i.e.*, specify their centers and radii), to minimize an objective subject to some constraints.

One constraint is that the first k disks are fixed, *i.e.*,

$$c_i = c_i^{\text{fix}}, \quad r_i = r_i^{\text{fix}}, \quad i = 1, \dots, k,$$

where c_i^{fix} and r_i^{fix} are given.

The second constraint is an overlap or intersection constraint, which requires some pairs of disks to intersect:

$$D_i \cap D_j \neq \emptyset, \quad (i, j) \in \mathcal{I},$$

where $\mathcal{I} \subset \{1, \dots, n\}^2$ is given. You can assume that for each $(i, j) \in \mathcal{I}$, $i < j$.

We consider two objectives: The sum of the disk areas, and the sum of the disk perimeters. These two objectives result in two different problems.

- (a) Explain how to solve these two problems using convex optimization.
- (b) Solve both problems for the problem data given in `disks_data.*`. Give the optimal total area, and the optimal total perimeter. Plot the two optimal disk arrangements, using the code included in the data file. Give a *very brief* comment on the results, especially the distribution of disk radii each problem obtains.

7. *Decomposing a PV array output time series.* We are given a time series $p \in \mathbf{R}_+^T$ that gives the output power of a photo-voltaic (PV) array in 5-minute intervals, over $T = 2016$ periods (one week), given in `pv_output_data.*`. In this problem you will use convex optimization to decompose the time series into three components:

- The *clear sky output* $c \in \mathbf{R}_+^T$, a smooth daily-periodic component, which gives what the PV output would have been without clouds. This signal is 24-hour-periodic, *i.e.*, $c_{t+288} = c_t$ for $t = 1, \dots, T - 288$. (The clear sky output is zero at night, but we will not use this prior information in our decomposition method.)
- A *weather shading loss* component $s \in \mathbf{R}_+^T$, which gives the loss of power due to clouds. This component satisfies $0 \preceq s \preceq c$, can change rapidly, and is not periodic.
- A *residual* $r \in \mathbf{R}^T$, which accounts for measurement error, anomalies, and other errors.

These components satisfy $p = c - s + r$.

We will assume that the average absolute value of the residual is no more than 4 (which is less than 1% of the average of p).

Smoothness of c is measured by its Laplacian,

$$\mathcal{L}(c) = (c_1 - c_2)^2 + \dots + (c_{287} - c_{288})^2 + (c_{288} - c_1)^2.$$

(Note that the term involves c_1 and c_{288} .)

We will choose c , s , and r by minimizing $\mathcal{L}(c) + \lambda \mathbf{1}^T s$ subject to the constraints described above, where λ is a positive parameter, that we take to be one.

Solve this problem, and plot the resulting c , s , r , and p (which is given), on separate plots. Give the average values of c , s , and p , and the average absolute value of r (which should be 4).

8. *Rank one nonnegative matrix approximation.* We are given *some* entries of an $m \times n$ matrix A with positive entries, and wish to approximate it as the outer product of vectors x and y with positive entries, *i.e.*, xy^T . We will use the average relative deviation between the entries of A and xy^T as our approximation criterion,

$$\frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n R(A_{ij}, x_i y_j),$$

where R is the relative deviation of two positive numbers, defined as

$$R(u, v) = \max\{u/v, v/u\} - 1.$$

If we scale x by the positive number α , and y by $1/\alpha$, the outer product $(\alpha x)(y/\alpha)^T$ is the same as xy^T , so we will normalize x as $\mathbf{1}^T x = 1$.

The data in the problem consists of *some* of the values of A . Specifically, we are given A_{ij} for $(i, j) \in \Omega \subseteq \{1, \dots, m\} \times \{1, \dots, n\}$. Thus, your goal is to find $x \in \mathbf{R}_{++}^m$ (which satisfies $\mathbf{1}^T x = 1$), $y \in \mathbf{R}_{++}^n$, and $A_{ij} > 0$ for $(i, j) \notin \Omega$, to minimize the average relative deviation between the entries of A and xy^T .

- (a) Explain how to solve this problem using convex or quasiconvex optimization.
- (b) Solve the problem for the data given in `rank_one_nmf_data.*`. This includes a matrix `A`, and a set of indexes `Omega` for the given entries. (The other entries of `A` are filled in with zeros.) Report the optimal average relative deviation between A and xy^T . Give your values for x_1 , y_1 , and $A_{11} = x_1 y_1$.

9. *Post-modern portfolio optimization metrics.* Let $r \in \mathbf{R}^T$ denote a time series (say, daily) of investment returns, *i.e.*, the increase in value divided by initial value. The value of the investment (typically, a portfolio) is the time series vector $v \in \mathbf{R}^T$ defined by the recursion

$$v_{t+1} = v_t(1 + r_t), \quad t = 0, \dots, T-1,$$

with v_0 a given positive initial value. Here we are compounding the investment returns. We will assume that all returns satisfy $r_t > -1$, which implies that $v \succ 0$. We define the *high-water value* or *last high value* as

$$h_t = \max_{\tau \leq t} v_\tau, \quad t = 1, \dots, T.$$

The value and high-water value are functions of r .

Portfolio theory as originally developed by Markowitz in the 1950s takes into account the mean return $\mu = \mathbf{1}^T r / T$ and variance (risk) $\sigma^2 = \|r - \mu \mathbf{1}\|_2^2 / T$. The idea of using a mathematical approach to choose a portfolio to maximize return and minimize risk came to be called *modern portfolio theory*. Of course, it's not so modern nowadays.

Researchers later suggested various alternative metrics that are (supposedly) closer to what we really care about than the mean return and risk. The use of these metrics was dubbed (or marketed as) *post-modern portfolio theory*. Some of these so-called post-modern portfolio metrics are described below, along with a parenthetical note about whether we'd like to minimize or maximize the metric.

For each metric we wish to minimize, determine if it is a convex or quasiconvex function of r , or neither. For each metric we wish to maximize, determine if it is a concave or quasiconcave function of r , or neither. For example, the mean return (which we wish to maximize) is a concave function of r , and the risk (variance, which we wish to minimize) is a convex function of r . When the metric is convex or quasiconvex (or concave or quasiconcave), justify your answer. When it is neither, you can simply state this; you do not need to produce a counterexample. We will deduct some points if your answer is not strong enough, *e.g.*, if you assert that a metric is quasiconvex, but it is in fact convex.

- (a) *Logarithmic or Kelly growth rate.* (Maximize.) $(1/T) \sum_t \log(1 + r_t)$. This is the average growth rate of v_t .
- (b) *Downside variance.* (Minimize.) The downside variance is $(1/T) \sum_t (r_t - \mu)_-^2$, where $(u)_- = \max\{-u, 0\}$, and μ is the mean return. This assesses a penalty for a return below the average (the 'downside'), but not for a return above the average.
- (c) *Maximum drawdown.* (Minimize.) The *drawdown* at period t is defined as $d_t = (h_t - v_t)/h_t$. The maximum drawdown is defined as $\max_t d_t$.
- (d) *Maximum consecutive days under water.* (Minimize.) A time period t is called *under water* if $v_t < h_t$, *i.e.*, the current value is less than the last high. Maximum

consecutive days under water means just that, *i.e.*, the maximum number of consecutive days under water.

Remark. Many other post-modern metrics can be derived from, or are related to, the ones described above. Examples include the Sortino, Calmar, and Information ratios. You can thank the EE364a staff for refraining from asking about these.

10. *Blending overlapping covariance matrices.* We consider the problem of constructing a covariance matrix $R \in \mathbf{S}_+^n$ from two (not necessarily consistent) estimates of submatrices S and T . We order the indices in the underlying random variable so that the first n_1 entries correspond to those in the first submatrix but not the second, the next n_2 entries correspond to the entries in both submatrices, and the last n_3 entries are those in the second submatrix but not the first. We have $n_1 + n_2 + n_3 = n$, and we assume all three are positive. We partition the matrix R as

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{12}^T & R_{22} & R_{23} \\ R_{13}^T & R_{23}^T & R_{33} \end{bmatrix}.$$

We wish to choose $R \in \mathbf{S}_+^n$ so that

$$R^{(1)} = \begin{bmatrix} R_{11} & R_{12} \\ R_{12}^T & R_{22} \end{bmatrix} \approx S = \begin{bmatrix} S_{11} & S_{12} \\ S_{12}^T & S_{22} \end{bmatrix}$$

and

$$R^{(2)} = \begin{bmatrix} R_{22} & R_{23} \\ R_{23}^T & R_{33} \end{bmatrix} \approx T = \begin{bmatrix} T_{22} & T_{23} \\ T_{23}^T & T_{33} \end{bmatrix}.$$

(Note the non-standard labeling of the block indices in T .) You can assume that $S \in \mathbf{S}_+^{n_1+n_2}$ and $T \in \mathbf{S}_+^{n_2+n_3}$ are given.

Roughly speaking, your job is to guess the six submatrices R_{ij} for $i \leq j$. For four of these, R_{11}, R_{12}, R_{23} , and R_{33} , you have only one piece of data to work with, *i.e.*, S_{11}, S_{12}, T_{23} , and T_{33} , respectively. For one of them, R_{22} , you have two pieces of data to work with, *i.e.*, S_{22} and T_{22} . For one submatrix, R_{13} , you have no pieces of data to work with.

- (a) *A simple method.* Based on the given data S and T , our guess of R is

$$\begin{aligned} R_{11} &= S_{11}, & R_{12} &= S_{12}, & R_{13} &= 0, \\ R_{22} &= (1/2)(S_{22} + T_{22}), & R_{23} &= T_{23}, & R_{33} &= T_{33}. \end{aligned}$$

For the four submatrices for which you have only one piece of data, we simply use that data as our guess. For the one submatrix for which we have two pieces of data, we average the two values. For the one submatrix for which we have no data, we guess the zero matrix.

Show by a specific numerical example that this simple method can yield an *unacceptable* value of R . (No, we will not be more specific about what we mean by this; part of the problem is to figure out what we mean. Also, we will deduct points from examples that are more complicated than they need to be.)

- (b) *Convex optimization to the rescue.* Suppose we choose R by solving the convex optimization problem

$$\begin{aligned} &\text{minimize} && \|R^{(1)} - S\|_F^2 + \|R^{(2)} - T\|_F^2 + \|R_{13}\|_F^2 \\ &\text{subject to} && R \succeq 0. \end{aligned}$$

Here the variable is $R \in \mathbf{S}^n$, and $\|U\|_F = (\mathbf{Tr}(U^T U))^{1/2}$ is the Frobenius norm of a matrix.

Let R^{sim} be the estimate of R obtained using the simple method in part (a). Show that if $R^{\text{sim}} \succeq 0$, then it is the solution of this problem.

- (c) Apply the method described in part (b) to the specific numerical example you provided in part (a), and check (numerically) that the result R^* is now acceptable.