EE263 Prof. S. Boyd
Oct. 26–27 or Oct. 27–28, 2007.

Midterm exam

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

Please read the following instructions carefully.

- You may use any books, notes, or computer programs (*e.g.*, Matlab), but you may not discuss the exam with anyone until Oct. 29, after everyone has taken the exam. The only exception is that you can ask the TAs or Stephen Boyd for clarification, by emailing to the staff email address. We've tried pretty hard to make the exam unambiguous and clear, so we're unlikely to say much.

- Please address email inquiries to `ee263-aut0708-staff@lists.stanford.edu`. This forwards the mail to the professor and the TAs. In particular, please do not use Stephen Boyd's or the TAs' individual email addresses.

- Since you have 24 hours, we expect your solutions to be legible, neat, and clear. Do not hand in your rough notes, and please try to simplify your solutions as much as you can. We will deduct points from solutions that are technically correct, but much more complicated than they need to be.

- Please check your email a few times during the exam, just in case we need to send out a clarification or other announcement. It's unlikely we'll need to do this, but you never know.

- Attach the official exam cover page (available when you pick up or drop off the exam, or from the midterm info page) to your exam, and assemble your solutions to the problems in order, *i.e.*, problem 1, problem 2, ..., problem 5. Start each solution on a new page.

- Please make a copy of your exam before handing it in. We have never lost one, but it might occur.

- When a problem involves some computation (say, using Matlab), we do not want just the final answers. We want a clear discussion and justification of exactly what you did, the Matlab source code that produces the result, and the final numerical result. Be sure to show us your verification that your computed solution satisfies whatever properties it is supposed to, at least up to numerical precision. For example, if you compute a vector x that is supposed to satisfy $Ax = b$ (say), show us the Matlab code

that checks this, and the result. (This might be done by the Matlab code `norm(A*x-b)`; be sure to show us the result, which should be very small.) *We will not check your numerical solutions for you, in cases where there is more than one solution.*

- In the portion of your solutions where you explain the mathematical approach, you *cannot* refer to Matlab operators, such as the backslash operator. (You can, of course, refer to inverses of matrices, or any other standard mathematical construct.)

- Some of the problems are described in a practical setting, such as signal processing, communications, or control. *You do not need to understand anything about the application area to solve these problems.* We've taken special care to make sure all the information and math needed to solve the problem is given in the problem description.

- We *do not* expect you to be able to solve all parts of all problems, so don't worry if you cannot finish them all.

- Some of the problems require you to download and run a Matlab file to generate the data needed. These files can be found at the URL

  `http://www.stanford.edu/class/ee263/midterm_mfiles/FILENAME`

  where you should substitute the particular filename (given in the problem) for `FILENAME`. *There are no links on the course web page pointing to these files, so you'll have to type in the whole URL yourself.*

- Please respect the honor code. Although we encourage you to work on homework assignments in small groups, *you cannot discuss the midterm with anyone*, with the exception of Stephen Boyd and the TAs, until everyone has taken it.

- Finally, a few hints:
  - Problems may be easier (or harder) than they might at first appear.
  - None of the problems require long calculations or any serious programming.

1. *Vector space multiple access (VSMA).* We consider a system of k transmitter-receiver pairs that share a common medium. The goal is for transmitter i to transmit a vector signal $x_i \in \mathbf{R}^{n_i}$ to the ith receiver, without interference from the other transmitters. All receivers have access to the same signal $y \in \mathbf{R}^m$, which includes the signals of all transmitters, according to

$$y = A_1 x_1 + \cdots + A_k x_k,$$

where $A_i \in \mathbf{R}^{m \times n_i}$. You can assume that the matrices A_i are skinny, *i.e.*, $m \geq n_i$ for $i = 1, \ldots, k$. (You can also assume that $n_i > 0$ and $A_i \neq 0$, for $i = 1, \ldots, k$.) Since the k transmitters all share the same m-dimensional vector space, we call this *vector space multiple access*. Each receiver knows the received signal y, and the matrices $A_1, \ldots, A_k$.

We say that the ith signal is *decodable* if the ith receiver can determine the value of x_i, no matter what values $x_1, \ldots, x_k$ have. Roughly speaking, this means that receiver i can process the received signal so as to perfectly recover the ith transmitted signal, while rejecting any interference from the other signals $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k$. Whether or not the ith signal is decodable depends, of course, on the matrices $A_1, \ldots, A_k$.

Here are four statements about decodability:

(a) Each of the signals $x_1, \ldots, x_k$ is decodable.

(b) The signal x_1 is decodable.

(c) The signals $x_2, \ldots, x_k$ are decodable, but x_1 isn't.

(d) The signals $x_2, \ldots, x_k$ are decodable when x_1 is 0.

For each of these statements, you are to give the exact (*i.e.*, necessary and sufficient) conditions under which the statement holds, in terms of $A_1, \ldots, A_k$ and $n_1, \ldots, n_k$. Each answer, however, must have a very specific form: it must consist of a conjunction of one or more of the following properties:

I. $\mathrm{rank}(A_1) < n_1$.

II. $\mathrm{rank}([A_2 \ \cdots \ A_k]) = n_2 + \cdots + n_k$.

III. $\mathrm{rank}([A_1 \ \cdots \ A_k]) = n_1 + \mathrm{rank}([A_2 \ \cdots \ A_k])$.

IV. $\mathrm{rank}([A_1 \ \cdots \ A_k]) = \mathrm{rank}(A_1) + \mathrm{rank}([A_2 \ \cdots \ A_k])$.

As examples, possible answers (for each statement) could be "I" or "I and II", or "I and II and IV". For some statements, there may be more than one correct answer; we will accept any correct one.

You can also give the response "My attorney has advised me not to respond to this question at this time." This response will receive partial credit.

For (just) this problem, we want only your answers. We do not want, and will not read, any further explanation or elaboration, or any other type of answers.

2. *Signal reconstruction for a bandlimited signal.* In this problem we refer to *signals*, which are just vectors, with index interpreted as (discrete) time. It is common to write the index for a signal as an argument, rather than as a subscript; for example, if $y \in \mathbf{R}^N$ is a signal, we use $y(t)$ to denote y_t, with $t \in \{1, 2, \ldots, N\}$. Another notation you'll sometimes see in signal processing texts is $y[t]$ for y_t.

   The *discrete cosine transformation* (DCT) of the signal $y \in \mathbf{R}^N$ is another signal, typically denoted using the corresponding upper case symbol $Y \in \mathbf{R}^N$. It is defined as

$$Y(k) = \sum_{t=1}^{N} y(t)w(k) \cos \frac{\pi(2t-1)(k-1)}{2N}, \quad k = 1, \ldots, N,$$

   where $w(k)$ are weights, with

$$w(k) = \begin{cases} \sqrt{1/N}, & k = 1, \\ \sqrt{2/N}, & k = 2, \ldots, N. \end{cases}$$

   The number $Y(k)$ is referred to as the *kth DCT coefficient of* y. The *DCT bandwidth* of the signal y is the smallest K for which $Y(K) \neq 0$, and $Y(k) = 0$ for $k = K+1, \ldots, N$. When $K < N$, the signal is called *DCT bandlimited*. (The term is typically used to refer to the case when the DCT bandwidth, K, is significantly smaller than N.)

   A signal y can be reconstructed from its DCT Y, via the *inverse DCT transform*, with

$$y(t) = \sum_{k=1}^{N} Y(k)w(k) \cos \frac{\pi(2t-1)(k-1)}{2N}, \quad t = 1, \ldots, N,$$

   where $w(k)$ are the same weights as those used above in the DCT.

   Now for the problem. You are given noise-corrupted values of a DCT bandlimited signal y, at some (integer) times $t_1, \ldots, t_M$, where $1 \leq t_1 < t_2 < \cdots < t_M \leq N$:

$$y_i^{\text{samp}} = y(t_i) + v_i, \quad i = 1, \ldots, M.$$

   Here, v_i are small noises or errors. You don't know v, but you do know that its RMS value is approximately σ, a known constant. (In signal processing, y^{samp} would be called a non-uniformly sampled, noise corrupted version of y.)

   Your job is to

   - Determine the smallest DCT bandwidth (*i.e.*, the smallest K) that y could have.
   - Find an estimate of y, $\hat{y}$, which has this bandwidth.

   Your estimate $\hat{y}$ must be consistent with the sample measurements y^{samp}. While it need not match exactly (you were told there was a small amount of noise in y^{samp}), you should ensure that the vector of differences,

$$(y_1^{\text{samp}} - \hat{y}(t_1^{\text{samp}}), \ldots, y_M^{\text{samp}} - \hat{y}(t_M^{\text{samp}})),$$

   has a small RMS value, on the order of σ (and certainly no more than 3σ).

(a) Clearly describe how to solve this problem. You can use any concepts we have used, to date, in EE263. *You cannot use (and do not need) any concepts from outside the class. This includes the Fourier transform and other signal processing methods you might know.*

(b) Carry out your method on the data in `bandlimit.m`. Running this script will define `N`, `ysamp`, `M`, `tsamp`, and `sigma`. It will also plot the sampled signal.

Give K, your estimate of the DCT bandwidth of y. Show $\hat{y}$ on the same plot as the original sampled signal. (We have added the command to do this in `bandlimit.m`, but commented it out.)

Also, give us $\hat{y}(129)$, to four significant figures.

You might find the Matlab functions `dct` and `idct` useful; `dct(eye(N))` and `idct(eye(N))` will return matrices whose columns are the DCT, and inverse DCT transforms, respectively, of the unit vectors. Note, however, that you can solve the problem without using these functions.

3. *Fitting a model for hourly temperature.* You are given a set of temperature measurements (in degrees C), $y_t \in \mathbf{R}$, $t = 1, \ldots, N$, taken hourly over one week (so $N = 168$). An expert says that over this week, an appropriate model for the hourly temperature is a trend (*i.e.*, a linear function of t) plus a diurnal component (*i.e.*, a 24-periodic component):

$$\hat{y}_t = at + p_t,$$

where $a \in \mathbf{R}$ and $p \in \mathbf{R}^N$ satisfies $p_{t+24} = p_t$, for $t = 1, \ldots, N - 24$. We can interpret a (which has units of degrees C per hour) as the warming or cooling trend (for $a > 0$ or $a < 0$, respectively) over the week.

(a) Explain how to find $a \in \mathbf{R}$ and $p \in \mathbf{R}^N$ (which is 24-periodic) that minimize the RMS value of $y - \hat{y}$.

(b) Carry out the procedure described in part (a) on the data set found in `tempdata.m`. Give the value of the trend parameter a that you find. Plot the model $\hat{y}$ and the measured temperatures y on the same plot. (The Matlab code to do this is in the file `tempdata.m`, but commented out.)

(c) *Temperature prediction.* Use the model found in part (b) to predict the temperature for the next 24-hour period (*i.e.*, from $t = 169$ to $t = 192$). The file `tempdata.m` also contains a 24 long vector `ytom` with tomorrow's temperatures. Plot tomorrow's temperature and your prediction of it, based on the model found in part (b), on the same plot. What is the RMS value of your prediction error for tomorrow's temperatures?

4. *Minimum energy roundtrip.* We consider the linear dynamical system

$$x(t+1) = Ax(t) + Bu(t), \quad x(0) = 0,$$

with $u(t) \in \mathbf{R}$ and $x(t) \in \mathbf{R}^n$. We must choose $u(0), u(1), \ldots, u(T-1)$ so that $x(T) = 0$ (*i.e.*, after T steps we are back at the zero state), and $x(t_{\text{dest}}) = x_{\text{dest}}$ (*i.e.*, at time t_{dest} the state is equal to x_{dest}). Here $x_{\text{dest}} \in \mathbf{R}^n$ is a given destination state. The time t_{dest} is *not* given; it can be any integer between 1 and $T-1$. The goal is to minimize the total input energy, defined as

$$E = \sum_{t=0}^{T-1} u(t)^2.$$

Note that you have to find t_{dest}, the time when the state hits the desired state, as well as the input trajectory $u(0), \ldots, u(T-1)$.

(a) Explain how to do this. For this problem, you may assume that $n \leq t_{\text{dest}} \leq T-n$. If you need some matrix or matrices that arise in your analysis to be full rank, you can just assume they are. But you must state this clearly.

(b) Carry out your method on the particular problem instance with data

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad T = 30, \quad x_{\text{dest}} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}.$$

Give the optimal value of t_{dest} and the associated value of E, and plot the optimal input trajectory u.

5. *Empirical algorithm complexity.* The runtime T of an algorithm depends on its input data, which is characterized by three key parameters: k, m, and n. (These are typically integers that give the dimensions of the problem data.) A simple and standard model that shows how T scales with k, m, and n has the form

$$\hat{T} = \alpha k^{\beta} m^{\gamma} n^{\delta},$$

where α, β, γ, $\delta \in \mathbf{R}$ are constants that characterize the approximate runtime model. If, for example, $\delta \approx 3$, we say that the algorithm has (approximately) cubic complexity in n. (In general, the exponents β, γ, and δ need not be integers, or close to integers.)

Now suppose you are given measured runtimes for N executions of the algorithm, with different sets of input data. For each data record, you are given T_i (the runtime), and the parameters k_i, m_i, and n_i. It's possible (and often occurs) that two data records have identical values of k, m, and n, but different values of T. This means the algorithm was run on two different data sets that had the same dimensions; the corresponding runtimes can be (and often are) a little different.

We wish to find values of α, β, γ, and δ for which our model (approximately) fits our measurements. We define the fitting cost as

$$J = (1/N) \sum_{i=1}^{N} \left(\log(\hat{T}_i/T_i) \right)^2,$$

where $\hat{T}_i = \alpha k_i^{\beta} m_i^{\gamma} n_i^{\delta}$ is the runtime predicted by our model, using the given parameter values. This fitting cost can be (loosely) interpreted in terms of relative or percentage fit. If $(\log(\hat{T}_i/T_i))^2 \leq \epsilon$, then $\hat{T}_i$ lies between $T_i/\exp\sqrt{\epsilon}$ and $T_i \exp\sqrt{\epsilon}$.

Your task is to find constants α, β, γ, δ that minimize J.

(a) Explain how to do this. If your method always finds the values that give the true global minimum value of J, say so. If your algorithm cannot guarantee finding the true global minimum, say so. If your method requires some matrix (or matrices) to be full rank, say so.

(b) Carry out your method on the data found in `empac_data.m`. Give the values of α, β, γ, and δ you find, and the corresponding value of J.