Introduction to Linear Dynamical Systems EE263, Summer 2019
July 20-21 2019

Midterm Exam

This is a 8 hour take-home exam with 2 problems. Please turn it in on Gradescope 8 hours after it is emailed to you.

- You may use any books, notes, or computer programs (*e.g.*, matlab), but you may not discuss the exam with others until July 21, after everyone has taken the exam. The only exception is that you can ask the course staff for clarification, by emailing to the staff email address `{gorish, lucasfv, amomenis} @stanford.edu`. We've tried pretty hard to make the exam unambiguous and clear, so we're unlikely to say much. Please do not post any exam related questions on Piazza.
- Since you have 8 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 and canvas 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.
- Assemble your solutions to the problems in order, *i.e.*, problem 1, problem 2. 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.
- If a problem asks for some specific answers, make sure they are obvious in your solutions. You might put a box around the answers, so they stand out from the surrounding discussion, justification, plots, etc.
- 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 power systems or computer vision. *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.
- The zip file for the datasets required for the exam have been emailed to you alongside the exam pdf.
- Please respect the honor code. Although we encourage you to work on homework assignments in small groups, *you cannot discuss the final exam with anyone*, with the exception of EE263 course staff, until July. 21 when everyone has taken it and the solutions are posted online.
- 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. *Optimal correction of facial features used by face recognition algorithms* [50 marks]

   Given an image, the goal of face recognition algorithms is to determine whether it is the image of a particular face, or not. Most of these algorithms work by first transforming the image into features, after which a classifier trained on *example faces* decides whether we have the image of a particular example face, or not.

   To describe an example face, we consider N *facial features* $x_1, \ldots, x_N \in \mathbf{R}^2$, which for us are 2-D locations of the facial parts such as the eyes, nose, outline of the mouth, etc, on the particular face. We take the center of the face image to be the origin. We collect the set of the facial features for the ith example face using the *features matrix*

$$F^{(i)} = \left[\begin{array}{ccc} x_1^{(i)} & \cdots & x_N^{(i)} \end{array} \right] \in \mathbf{R}^{2\times N}.$$

   Here is the problem. We are given the facial features for K possible example faces, represented using K feature matrices $F^{(1)}, \ldots, F^{(K)}$. We are asked to identify the face in a newly captured image, which is known to be one of the example faces.

   However, while capturing any image, the image is subjected to an arbitrary rotation and positive scaling around the origin.

   For example, let $x \in \mathbf{R}^2$ be a point on the perfect image of the face, then the rotation by an angle θ and positive scaling by $\alpha > 0$ would put the point at

$$\tilde{x} = \alpha \left[\begin{array}{cc} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{array} \right] x = Rx,$$

   The facial features of the captured image are given by $y_1, ..., y_N \in \mathbf{R}^2$.

   a) (10 marks) You are asked to find which of the example faces does the captured image correspond to. Propose a method to solve this problem. Do explain why the method works.

   b) (7 marks) Run your method on the data given in face_features_data.m, which contains K features matrices $F_1, \ldots, F_K$, and a matrix of measured facial features

$$Y_{rot} = \left[\begin{array}{ccc} y_1 & \cdots & y_N \end{array} \right] \in \mathbf{R}^{2\times N}.$$

      Which example face does the measured Y_{rot} correspond to?

   Now, due to a fault in the image capturing process, the rotated and scaled image was also subjected to a arbitrary translation in the 2D space (plus some small random measurement noise). For example, let $x \in \mathbf{R}^2$ be a point on the perfect image of the face, then the rotation by an angle θ and positive scaling by $\alpha > 0$ would put the point at

$$\tilde{x} = \alpha \left[\begin{array}{cc} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{array} \right] x = Rx,$$

while the translation by $t \in \mathbf{R}^2$ and additive small noise $v \in \mathbf{R}^2$ would produce

$$\hat{x} = \tilde{x} + t + v.$$

(c) (15 marks) You are now asked to correct the rotation, positive scaling, and translation for the given image in the root-mean-square (RMS) sense, *i.e.*, find coefficients of the corresponding rotation plus positive scaling matrix R and a translation vector t which minimize

$$\rho = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|y_i - Rx_i^{(k)} - t\|^2},$$

where $k \in \{1, \ldots, K\}$ is the index of the appropriate example face. Be sure to clearly describe your proposed method and to explain when and why it works.

(d) (15 marks) Run your method on the data given in face_features_data.m, which contains K features matrices $F_1, \ldots, F_K$, and a matrix of measured facial features

$$Y_{noisy} = \begin{bmatrix} y_1 & \cdots & y_N \end{bmatrix} \in \mathbf{R}^{2\times N}.$$

Write down your RMS error ρ, and the estimated rotation plus scaling matrix R and the translation vector t for each example face. Which example face would the measured Y_{noisy} most likely correspond to? Plot this example face and the recovered facial features from the captured Y_{noisy} using the coefficients computed above.

(e) (3 marks) Comment on the decimal point precision of the captured facial features required for method used in part(b) vs part(d).

Hint: Consider one example face at a time.
Face plotting can be done using the below code:

```
function plot_face(face_features)
% plots face features on 2 x 2 square
plot(face_features(1,:),face_features(2,:),'o')
axis([-2 2 -2 2]), axis square
end
```

2. *Fitting the power consumption of a system* [50 marks + 10 bonus marks]

In an ordinary least squares problem, we are given $A \in \mathbf{R}^{m \times n}$ (skinny and full rank) and $y \in \mathbf{R}^m$, and we choose $x \in \mathbf{R}^n$ in order to minimize

$$\|Ax - y\|_2^2 = \sum_{i=1}^{m} (\tilde{a}_i^T x - y_i)^2.$$

Note that the penalty that we assign to a measurement error does not depend on the sensor from which the measurement was taken. However, this is not always the right thing to do: if we believe that one sensor is more accurate than another, we might want to assign a larger penalty to an error in the measurement from the more accurate sensor. We can account for differences in the accuracies of our sensors by assigning sensor i a weight $w_i > 0$, and then minimizing

$$\sum_{i=1}^{m} w_i (\tilde{a}_i^T x - y_i)^2.$$

By giving larger weights to more accurate sensors, we can account for differences in the precision of our sensors.

(a) (10 marks) *Weighted least squares.* Explain how to choose x in order to minimize

$$\sum_{i=1}^{m} w_i (\tilde{a}_i^T x - y_i)^2,$$

where the weights $w_1, \ldots, w_n > 0$ are given.

(b) (15 marks) *Iteratively reweighted least squares for ℓ_1-norm approximation.* Consider a cost function of the form

$$\sum_{i=1}^{m} w_i(x) (\tilde{a}_i^T x - y_i)^2. \tag{1}$$

One heuristic for minimizing a cost function of the form given in (1) is *iteratively reweighted least squares*, which works as follows. First, we choose an initial point $x^{(0)} \in \mathbf{R}^n$. Then, we generate a sequence of points $x^{(1)}, x^{(2)}, \ldots \in \mathbf{R}^n$ by choosing $x^{(k+1)}$ in order to minimize

$$\sum_{i=1}^{m} w_i(x^{(k)}) (\tilde{a}_i^T x^{(k+1)} - y_i)^2.$$

Each step of this algorithm involves updating our weights, and solving a weighted least squares problem. Suppose we want to use this method to solve minimize the ℓ_1-norm approximation error, which is defined to be

$$\|Ax - y\|_1 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i|,$$

where the matrix $A \in \mathbf{R}^{m \times n}$ and the vector $y \in \mathbf{R}^m$ are given. How should we choose the weights $w_i(x)$ to make the cost function in (1) equal to the ℓ_1-norm approximation error?

We now consider an important special case of the multi-objective least squares least squares problems. In piece-wise constant fitting problems, we start with a signal represented by a vector $x \in \mathbf{R}^n$ that is almost piece-wise constant. The entries x_i correspond to the value of some function of time, evaluated (or sampled, in the language of signal processing) at equal time intervals. Such piece-wise constant signals are quite common in biological and medical systems, economics and electrical systems. In this problem we consider signals in one dimension, but the same ideas can be applied to signals in two or more dimensions, (e.g., images or video signals.)

Our measurement of the signal x is corrupted by an additive noise v:

$$x_{noisy} = x + v$$

The noise can be modeled in many different ways, but here we simply assume that it is unknown, small, and, unlike the signal, rapidly varying. The goal is to form a piece-wise constant **estimate** $\hat{x}$ of the original signal x, given the noisy measurement x_{noisy}. This process is called piece-wise constant fitting.

One simple formulation of this fitting problem is the bi-objective problem

$$\text{minimize} \quad (\|\hat{x} - x_{\text{noisy}}\|_2^2, \phi(\hat{x})), \tag{2}$$

where $\hat{x}$ is the optimization variable and x_{noisy} is the problem data. The function $\phi : \mathbf{R}^n \to \mathbf{R}$ is a special *regularization function*. It is meant to measure how close the estimate $\hat{x}$ is to a piece-wise constant function. Hence, the optimization problem (2) seeks signals that are close (in ℓ_2-norm) to the corrupted signal, and that are close to piece-wise constant, *i.e.*, for which $\phi(\hat{x})$ is small. We can find the optimal trade-off curve for the optimization problem (2) by minimizing the weighted-sum objective for different values of $\mu \geq 0$:

$$\text{minimize} \quad \|\hat{x} - x_{\text{noisy}}\|_2^2 + \mu\phi(\hat{x}) \tag{3}$$

Now finding a good measure for piece-wise constantness of a signal is tricky business!

Let's define the first difference of the vector $\hat{x}$ as follows:

$$D\hat{x} = \begin{bmatrix} \hat{x}_2 - \hat{x}_1 \\ \hat{x}_3 - \hat{x}_2 \\ \vdots \\ \hat{x}_n - \hat{x}_{n-1} \end{bmatrix}$$

A piece-wise constant $\hat{x}$ will only have a few nonzero entries in its first difference $D\hat{x}$. Therefore, the ideal measure for piece-wise constantness of a signal would be the number of nonzero entries in its first difference, and we can have:

$$\phi(\hat{x}) = card(D\hat{x}), \tag{4}$$

where the card function returns the number of nonzero entries in a vector.

(c) (10 marks) Explicitly specify the entries of matrix D used in (4) and its dimension.

The problem with using the card function for our regularization function $\phi(\hat{x})$ is that it makes the optimization problem (3) almost impossible to solve. Thus, we need to use a heuristic for our regularization function which can make our optimization problem more tractable. One of those heuristics is using the quadratic approximation.

$$\phi_{\text{quad}}(\hat{x}) = \|D\hat{x}\|_2^2$$

Using this heuristic function, our problem of fitting a piece-wise constant fit $\hat{x}$ to a noisy signal measurement x_{noisy} can be cast into the following bi-objective optimization problem:

$$\text{minimize} \quad \|\hat{x} - x_{\text{noisy}}\|_2^2 + \mu\|D\hat{x}\|_2^2, \tag{5}$$

where $\mu \geq 0$ parameterizes the optimal trade-off curve.

(d) (15 marks) In this part we will use the above method of quadratic approximation for piece-wise constant data fitting to model the power consumption of different subsystems in a health monitoring device. Imagine a health monitoring implant device that consists of two subsystems: A glucose sensing unit (GSU), and a low energy Bluetooth unit (BLE). The device is programmed to wake up every 100ms, measure the blood glucose of a patient, transmit the data to a base station over Bluetooth and go back to sleep. Hence, the power consumption for each of the two subsystems will have a periodic pattern with a period of 100ms. In addition, we know that the true power consumption for each subsystem is best modeled by a piece-wise constant signal over the course of a 100ms. In data file sys_power_model_data.m, you are given $x_{gsu,noisy}$, and $x_{ble,noisy} \in \mathbf{R}^{1000}$, which are noisy measurements of the power consumption (in milliwatts [mW]) for the two subsystems over 100ms at a sampling interval of 0.1ms intervals. For each noisy measurement, sweep the parameter μ in (5) and plot the optimal trade-off curve between $\|\hat{x} - x_{\text{noisy}}\|_2$ and $\|D\hat{x}\|_2^2$. Based on the optimal trade-off curve for each case, choose a reasonable value of μ that gives you a good piece-wise constant fit to signals x_{gsu}, and x_{ble}. (Note that best choice of μ does not have to be the same for all two cases). Report your choices of μ for the two cases.

Provide plots of (i) the noisy signal and the piece-wise constant fit to it on the same figure for each case (2 separate figures) and (ii) the optimal trade-off curve for each case (2 separate figures).

As you should see from the plots for the noisy signal and the piece-wise constant fit, the quadratic approximation performs quite poorly in estimating a piecewise constant fit. This is because it tries to smoothen out any large rapid variations in the original signal and meanwhile, also tries to overfits the smaller variations in the noise in the signal. It turns out that in most practical applications, the ℓ_1-norm approximation function is an extremely good substitute for the card or the ℓ_2-norm approximation, and makes the optimization problem completely tractable. Therefore, we will try to work out the part (d) again using the new regularization function:

$$\phi_{l_1}(\hat{x}) = \|D\hat{x}\|_1$$

Now, our problem of finding a piece-wise constant fit $\hat{x}$ to a noisy signal measurement x_{noisy} can be cast into the following bi-objective optimization problem:

$$\text{minimize} \quad \|\hat{x} - x_{\text{noisy}}\|_2^2 + \mu\|D\hat{x}\|_1 \tag{6}$$

(e) (10 bonus marks) Formulate the new optimization problem (6) as an Iterative reweighted least squares problem as seen in part(b). Repeat the problem in part(d) by now solving the new optimization problem (6) for different values of μ. Plot the optimal trade-off curve by sweeping over μ, choose a value of μ that results in a good piece-wise constant fit to the signals, and plots the curves of (i) the noisy signal and the piece-wise constant fit to it on the same figure for each case (2 separate figures) and (ii) the optimal trade-off curve for each case (2 separate figures).

Compare your results of piece-wise constant fit curves with the previous part.

Hint: First you need to use the results of parts (a) and (b) to reformulate the bi-objective problem (6) as an iteratively reweighted least squares problem. The solution for this problem could be found by an iterative approach as described in part(b). You can choose your $\hat{x}^{(0)}$ for each case to be equal to x_{noisy} and then solve the problem for each value of μ in a few iterations. Generally, 5 to 10 iterations should be enough to get you decent convergence. Note that since our approach to this problem is a heuristic, in some cases it will not lead to a prefect piece-wise constant fit. Nevertheless, your results should be very close to piece-wise constant fits. Also note that when you are solving for $w_i(x^{(k)})$, you might encounter cases where it goes to infinity ($i.e. w_i(x^{(k)}) = 1/0$). In such cases, do not update the weight and reuse the old iterate ($i.e. w_i(x^{(k)}) = w_i(x^{(k-1)})$).

If your implementation is efficient, it should take only a few minutes to run your code for each case.