Midterm Exam Solutions

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, amommenis\} @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 \texttt{norm(A*x-b)}; be sure to show us the result, which should be very small.) \textit{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 $x_1, \ldots, x_N \in \mathbb{R}^2$, 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 $i$th example face using the features matrix

$$F^{(i)} = \begin{bmatrix} x_1^{(i)} & \cdots & x_N^{(i)} \end{bmatrix} \in \mathbb{R}^{2 \times N}.$$ 

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

$$\tilde{x} = \alpha \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} x = Rx,$$

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

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 $K$ feature matrices $F_1, \ldots, F_K$, and a matrix of measured facial features

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

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

$$\tilde{x} = \alpha \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} x = Rx,$$
while the translation by \( t \in \mathbb{R}^2 \) and additive small noise \( v \in \mathbb{R}^2 \) 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
\[
\rho = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \| y_i - R x_i^{(k)} - t \|^2},
\]
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
\[
Y_{noisy} = \begin{bmatrix} y_1 & \cdots & y_N \end{bmatrix} \in \mathbb{R}^{2 \times N}.
\]
Write down your RMS error \( \rho \), 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:

```matlab
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
```

(a) A rotation by \( \theta \) radians in \( \mathbb{R}^2 \) is given by
\[
U_\theta = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix},
\]
while positive scaling is a multiplication by a scalar \( \alpha > 0 \). Therefore, we have
\[
R = \alpha U_\theta = \begin{bmatrix} a & -b \\ b & a \end{bmatrix},
\]

4
where \( a = \alpha \cos \theta \in \mathbb{R} \) and \( b = \alpha \sin \theta \in \mathbb{R} \) are unknown.

Any captured image will be of the form \( Y = R \ast F \), where \( F \) is the actual image. Thus, for any captured image to correspond to the example image,

\[
Y^T = F^T R^T \implies Y_i^T \in \text{Range}(F^T)
\]

for both columns \( i \) of \( Y^T \).

(b) To check if \( Y_i^T \in \text{Range}(F^T) \) for both \( i \), you can look at the rank of both \( F^T \) and \( [F^T : Y_i^T] \) for \( i = 1,2 \) or combine them together to check rank of \( F^T \) and \( [F^T : Y^T] \).

Thus, the example face for which \( \text{rank}(F^T) = \text{rank}([F^T : Y^T]) \), that will correspond to the captured image \( Y \). With the given data, the 2nd example face corresponds to the captured image.

(c) A translation in \( \mathbb{R}^2 \) is given by a vector \( t \in \mathbb{R}^2 \) (which is added to the original vector).

In this part we are trying to find the best parameters \( a, b, \) and \( t = (t_1, t_2) \in \mathbb{R}^2 \) such that we minimize

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

for an appropriate \( k \in \{1, \ldots, K\} \).

Now let’s see what can be done when we pick some \( k \), i.e., we choose an example face to which we match obtained facial features by finding appropriate rotation, scaling, and translation parameters. To minimize \( \rho^{(k)} \) for each \( k \), we use the standard trick: we square the \( \rho^{(k)} \) expression and ignore the constant \( 1/N \) term, both of which do not change the problem solution. We obtain minimization of an equivalent objective function

\[
\sum_{i=1}^{N} \|y_i - Rx_i^{(k)} - t\|^2 = \sum_{i=1}^{N} \|Rx_i^{(k)} + t - y_i\|^2
\]

\[
= \sum_{i=1}^{N} \left\| \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} x_{i,1}^{(k)} \\ x_{i,2}^{(k)} \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} - \begin{bmatrix} y_{i,1} \\ y_{i,2} \end{bmatrix} \right\|^2
\]

\[
= \sum_{i=1}^{N} \left\| \begin{bmatrix} x_{i,1}^{(k)} - x_{i,2}^{(k)} & 1 & 0 \\ x_{i,2}^{(k)} & x_{i,1}^{(k)} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ t_1 \end{bmatrix} - \begin{bmatrix} y_{i,1} \\ y_{i,2} \end{bmatrix} \right\|^2
\]

\[
= \sum_{i=1}^{N} \|G_{i}^{(k)} z - h_i\|^2 = \|G^{(k)} z - h\|^2,
\]

for an appropriate \( k \in \{1, \ldots, K\} \).
which is a least squares problem with the unknown vector $z = (a, b, t_1, t_2) \in \mathbb{R}^4$ and data

$$G^{(k)} = \begin{bmatrix} G_1^{(k)} \\ \vdots \\ G_n^{(k)} \end{bmatrix} \in \mathbb{R}^{2n \times 4}, \quad h = \begin{bmatrix} h_1 \\ \vdots \\ h_n \end{bmatrix} \in \mathbb{R}^{2n}. $$

Given that the number of facial features is $n \geq 2$ and $G^{(k)}$ is full rank for each example face $k$, we can compute the optimal parameters using

$$z^{(k)}_{\text{opt}} = \left(G^{(k)T}G^{(k)}\right)^{-1}G^{(k)T}h,$$

and we can find optimal $\rho^{(k)}$. We pick the index $k$ with the smallest possible $\rho^{(k)}$ as the most plausible example face, and give the corresponding matrix $R_{\text{opt}}$ and the translation vector $t_{\text{opt}}$.

(d) The following Matlab code solves the numerical part of the problem.

```matlab
% load data
face_feature_data;

% solve least-squares problem for each face
[rho_1,Ropt_1,topt_1] = face_feature_ls(F1,Y);
[rho_2,Ropt_2,topt_2] = face_feature_ls(F2,Y);
[rho_3,Ropt_3,topt_3] = face_feature_ls(F3,Y);
[rho_4,Ropt_4,topt_4] = face_feature_ls(F4,Y);
[rho_5,Ropt_5,topt_5] = face_feature_ls(F5,Y);

rhos = [rho_1 rho_2 rho_3 rho_4 rho_5]

plot_face(inv(Ropt_4)*Y-topt_4*ones(1,20))
```

where the function `face_feature_ls` is given by

```matlab
function [rho,R,t] = face_feature_ls(F,Y)
    % perform least-square fitting to the rotated-scaled-translated facial features

    h = Y(:);
    G = [];
    [M,N] = size(F);
    for k = 1:N
        G = [G;
            F(1,k) -F(2,k) 1 0;
            F(2,k) F(1,k) 0 1;];
    end
```
\[ z = G\h; \]
\[ \rho = 1/\sqrt{N} \times \text{norm}(Gz-h); \]
\[ R = [z(1) -z(2); z(2) z(1)]; \]
\[ t = [z(3); z(4)]; \]

We compute the following values of \( \rho^{(k)} \):

\[
\rho^{(1)} = 0.071, \quad \rho^{(2)} = 0.061, \quad \rho^{(3)} = 0.080, \quad \rho^{(4)} = 0.025, \quad \rho^{(5)} = 0.080.
\]

From the above RMS errors we conclude that the measured facial features are the best match to the features of the example face 4 (when subjected to rotation, positive scaling, and translation in \( \mathbb{R}^2 \)). The corresponding \( R \) and \( t \) are given by

\[
R_{opt} = \begin{bmatrix} 1.065 & -0.746 \\ 0.746 & 1.065 \end{bmatrix}, \quad t_{opt} = \begin{bmatrix} 0.254 \\ 0.101 \end{bmatrix}.
\]

We give the plots of all example faces (which are all very similar).

A plot of the measured facial feature vectors and the recovered facial features (face 4) is shown below.

(e) Due to the non-noisy capturing of iamges in part (b), the features of the captured image needed for comparison of the rank with true face features had to be highly precise, as precise as the actual features, in this case upto 16 decimal point precision. The alternative could have been setting a tolerance limit in the rank computation.
In part (d), a noise was added to the captured image which could have been quantization noise, and least-squares approximation was applied in order to estimate the original image. Thus, the decimal point precision could be much lower in this case.

Figure 1: Left. Rotate, scaled, and translated noisy facial features. Right. Recovered facial features when matched to the example face 4.
2. **Fitting the power consumption of a system** [50 marks + 10 bonus marks]

In an ordinary least squares problem, we are given $A \in \mathbb{R}^{m \times n}$ (skinny and full rank) and $y \in \mathbb{R}^m$, and we choose $x \in \mathbb{R}^n$ 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 $w_1, \ldots, w_m > 0$ are given.

(b) (15 marks) **Iteratively reweighted least squares for $\ell_1$-norm approximation.** Consider a cost function of the form

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

(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 $x^{(0)} \in \mathbb{R}^n$. Then, we generate a sequence of points $x^{(1)}, x^{(2)}, \ldots \in \mathbb{R}^n$ 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 $\ell_1$-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 $A \in \mathbb{R}^{m \times n}$ and the vector $y \in \mathbb{R}^m$ are given. How should we choose the weights $w_i(x)$ to make the cost function in (1) equal to the $\ell_1$-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 $x \in \mathbb{R}^n$ 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 $v$:

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

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 $x_{\text{noisy}}$. This process is called piece-wise constant fitting.

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

$$\minimize \ (\|\hat{x} - x_{\text{noisy}}\|_2^2, \phi(\hat{x})),$$  \hspace{1cm} (2)

where $\hat{x}$ is the optimization variable and $x_{\text{noisy}}$ is the problem data. The function $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 $\ell_2$-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 $\mu \geq 0$:

$$\minimize \ \|\hat{x} - x_{\text{noisy}}\|_2^2 + \mu\phi(\hat{x})$$  \hspace{1cm} (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:

\[
\phi(\hat{x}) = \text{card}(D\hat{x}),
\]

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 \( x_{\text{noisy}} \) can be cast into the following bi-objective optimization problem:

\[
\minimize \quad \|\hat{x} - x_{\text{noisy}}\|_2^2 + \mu\|D\hat{x}\|_2^2,
\]

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 \( x_{\text{gsu, noisy}} \), and \( x_{\text{ble, noisy}} \in \mathbb{R}^{1000} \), 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 \( \mu \) 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 \( \mu \) that gives you a good piece-wise constant fit to signals \( x_{\text{gsu}} \), and \( x_{\text{ble}} \). (Note that best choice of \( \mu \) does not have to be the same for all two cases). Report your choices of \( \mu \) 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 $\ell_1$-norm approximation function is an extremely good substitute for the card or the $\ell_2$-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:

$$\phi_{\ell_1}(\hat{x}) = \|D\hat{x}\|_1$$

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

$$\text{minimize} \quad \|\hat{x} - x_{\text{noisy}}\|_2^2 + \mu\|D\hat{x}\|_1$$  \hspace{1cm} (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 $\mu$. Plot the optimal trade-off curve by sweeping over $\mu$, choose a value of $\mu$ 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 $x_{\text{noisy}}$ and then solve the problem for each value of $\mu$ 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.

**Solution.**
(a) We can express the objective function as

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

\[ = \left\| \begin{bmatrix} \sqrt{w_1} \tilde{a}_1^T x - \sqrt{w_1} y_1 \\ \vdots \\ \sqrt{w_m} \tilde{a}_m^T x - \sqrt{w_m} y_m \end{bmatrix} \right\|^2 \]

\[ = \| W^{\frac{1}{2}} A x - W^{\frac{1}{2}} y \|^2, \]

where \( W^{\frac{1}{2}} = \text{diag}(\sqrt{w_1}, \ldots, \sqrt{w_m}) \). Note that \((W^{\frac{1}{2}})^2 = W = \text{diag}(w_1, \ldots, w_m)\). Thus, minimizing this objective function is a least-squares problem; the solution is

\[ x = ((W^{\frac{1}{2}} A)^T (W^{\frac{1}{2}} A))^{-1} (W^{\frac{1}{2}} A)^T (W^{\frac{1}{2}} y) = (A^T W A)^{-1} A^T W y. \]

(b) If we choose \( w_i(x) = 1/|\tilde{a}_i^T x - y_i| \), then we have that

\[ \sum_{i=1}^{m} w_i(x)(\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} \frac{1}{|\tilde{a}_i^T x - y_i|} (\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i| = \| A x - y \|_1. \]

Note that \( w_i(x) \) is undefined if \( w_i(x) = |\tilde{a}_i^T x - y_i| = 0 \). In this case, we can just take \( w_i(x) \) to be some large value \( w_{\text{max}} \). However, note that in practice it is extremely unlikely that one of the residuals will be exactly equal to zero.

(c) Matrix \( D \in \mathbb{R}^{(n-1) \times n} \) is the bidiagonal matrix.

\[
D = \begin{pmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{pmatrix}
\]

(d)

\[
J = \| \hat{x} - x_{\text{noisy}} \|_2^2 + \mu \| D \hat{x} \|_2^2
\]

\[
= \left\| \begin{bmatrix} I \\ \sqrt{\mu} D \end{bmatrix} \hat{x} - \begin{bmatrix} x_{\text{noisy}} \\ 0 \end{bmatrix} \right\|_2^2
\]

The following Matlab code plots the optimal trade-off curve between the objectives \( \| \hat{x} - x_{\text{noisy}} \|_2 \) and \( \| D \hat{x} \|_2 \) for BLE. A reasonable value for \( \mu \) is one that results in the values of objectives being close to the knee of the optimal trade-off curve. The chosen \( \mu \) values for BLE, and GSU are correspondingly, 25.37, and 25.37. Values of \( \mu \) between reasonable range are acceptable. Below are plots of constant fits with the chosen \( \mu \) values. It can be seen that \( \hat{x} \) is not a very good piece-wise constant fit.
\% part_d

clear all

close all

sys_power_model_data

x = x_ble_noisy;
A = eye(length(x));
g = zeros(length(x)-1,1);
D = -eye(length(x)) + diag(ones(length(x)-1,1),1);
D(end,:) = [];

mu = [linspace(0,0.002,3) linspace(0.002,0.01,5) ...
linspace(0.015,0.15,5) linspace(0.50,100,5) linspace(200,2000,5)];

for i = 1:length(mu)
    y_tilde = [x; sqrt(mu(i))*g];
    A_tilde = [A; sqrt(mu(i))*D];
    xhat(:,i) = A_tilde\y_tilde;
    J1(i) = sqrt(sum((xhat(:,i)-x).^2));
    J2(i) = norm(D*xhat(:,i));
end

mu_index = 15;
mu(mu_index)

figure(1)
plot(J1,J2, 'linewidth', 2 )
hold on
plot(J1(mu_index), J2(mu_index), 'o')
set(gca,'fontsize',16)
ylabel('$\|D\hat{x}_\text{ble}\|_2$','interpreter','latex')
xlabel('$\| x_\text{ble}-\hat{x}_{\text{ble}}\|_2$','interpreter','latex')
legend('Trade off Curve', strcat('Mu =', num2str(mu(mu_index))))
grid on

figure(2)
hold on
plot([1:1000]/10,x,'linewidth',1.5);
plot([1:1000]/10,xhat(:,10),'r','linewidth',0.5);
ylim([-0.02 0.2])
legend({'$x_\text{ble}$','$\hat{x}_\text{ble}$'},'Interpreter','latex')
grid on
set(gca,'MarkerFaceAlpha',.2);
set(gca,'fontsize',16)
xlabel('$t$ (ms)', 'interpreter','latex')
ylabel('$x_{ble}$ (mW)', 'interpreter','latex')

The plots of the optimal tradeoff curve for a wide range of $\mu$ are shown below.

![Trade off Curve](image1)

**Figure 2:** *Left.* Optimal trade-off curve for the BLE noisy data *Right.* Optimal trade-off curve for the GSU noisy data.

We give the plots of noisy and piecewise constant fit for the chosen $\mu$ values (which are both very noisy).

\[(e)\]

\[ J = \|\hat{x} - x_{noisy}\|_2^2 + \mu \|D\hat{x}\|_1 \]
\[ = \|I\hat{x} - x_{noisy}\|_2^2 + \mu \|WD\hat{x}\|_2^2 \]
\[ = \|I\left[\sqrt{\mu W D}\right] \hat{x} - \begin{bmatrix} x_{noisy} \\ 0 \end{bmatrix}\|_2^2 \]

where $W = \text{diag}(\sqrt{w_1}, \ldots, \sqrt{w_n})$. 

15
Figure 3: Left. BLE data and its fit using the \( \ell_2 \)-norm approximation for \( \mu = 25.37 \).
Right. GSU data and its fit using the \( \ell_2 \)-norm approximation for \( \mu = 25.37 \).

The following MATLAB code plots the optimal trade-off curve between the objectives \( \| \hat{x} - x_{\text{noisy}} \|_2 \) and \( \| D \hat{x} \|_2^2 \) for BLE. A reasonable value for \( \mu \) is one that results in the values of objectives being close to the knee of the optimal trade-off curve. The chosen \( \mu \) values for BLE, and GSU are correspondingly, .0488 and 0.0150. Values of \( \mu \) between reasonable range are acceptable. Below are plots of constant fits with the chosen \( \mu \) values. It can be seen that \( \hat{x} \) is now a much better piece-wise constant fit, though still not perfect.

\[
\begin{align*}
\text{clear all} \\
\text{close all} \\
\text{sys\_power\_model\_data} \\
x = x\_ble\_noisy; \\
A = \text{eye}(\text{length}(x)); \\
g = \text{zeros}(\text{length}(x)-1,1); \\
D = -\text{eye}(\text{length}(x)) + \text{diag}(\text{ones}(\text{length}(x)-1,1),1); \\
D(\text{end,:}) = []; \\
N\_itr = 10; \% \text{Number of iterations for IRWLS}
\end{align*}
\]
mu = [linspace(0,0.002,3) linspace(0.002,0.01,5) ... 
  linspace(0.015,0.15,5) linspace(0.50,100,5)];

for i = 1:length(mu)
    xhat0 = x;
    y_tilde = [x; sqrt(mu(i))*g];
    for j = 1:N_itr
        w = 1./abs(D*xhat0);
        if j>1
            w(isinf(w)) = w_old(isinf(w));
        end
        W = diag(sqrt(w));

        D_tilde = W*D;
        A_tilde = [A; sqrt(mu(i))*D_tilde];
        xhat_temp = A_tilde\y_tilde;
        xhat0 = xhat_temp;
        w_old = w;
    end
    xhat(:,i) = A_tilde\y_tilde;
    J1(i) = sqrt(sum((xhat(:,i)-x).^2));
    J2(i) = sum(abs(D*xhat(:,i)));
end
mu_index = 10;
mu(mu_index)

figure(1)
pplot(J1,J2, 'linewidth', 2)
hold on
plot(J1(mu_index), J2(mu_index), 'o')
set(gca,'fontsize',16)
ylabel('\|D\hat{x}_\mathrm{ble}\|_1','interpreter','latex')
xlabel('\| x_\mathrm{ble}-\hat{x}_{\mathrm{ble}}\|_2','interpreter','latex')
legend('Trade off Curve', strcat('Mu =', num2str(mu(mu_index))))
grid on

figure(2)
hold on
plot([1:1000]/10,x);
The plots of the optimal tradeoff curve for a wide range of $\mu$ are shown below.

**Figure 4:** *Left.* Optimal trade-off curve for the BLE noisy data  *Right.* Optimal trade-off curve for the GSU noisy data.

We give the plots of noisy and piecewise constant fit for the chosen $\mu$ values (which are both very noisy).
Figure 5: Left. BLE data and piecewise-constant fit using the approximation for $\mu = 0.0488$ 
Right. GSU data and piecewise-constant fit using the approximation for $\mu = 0.0150$