Math 551 Fall 2018
Lab 10

Goal: In this lab, you will learn to use Matlab's backslash operator for signal denoising.

Getting started: You will need to download the file `m551lab10.m` and put it in your working directory.

What you have to submit: The file `m551lab10.m`, which you will modify during the lab session.

Reminders About the Grading Software

Remember, the labs are graded with the help of software tools. If you do not follow the instructions, you will lose points. If the instructions ask you to assign a value to a variable, you must **use the variable name given in the instructions**. If the instructions ask you to make a variable named `x1` and instead you make a variable named `x` or `X` or `X1` or any name other than `x1`, the grading software will not find your variable and you *will* lose points. Required variables are listed in the lab instructions in a gray box like this:

Variables: `x1`, `A`, `q`

At times you will also be asked to answer questions in a comment in your M-file. You must **format your text answers correctly**. Questions that you must answer are shown in gray boxes in the lab. For example, if you see the instruction

Q7: What does the Matlab command `lu` do?

your file must contain a line similar to the following somewhere

```
% Q7: It computes the LU decomposition of a matrix.
```

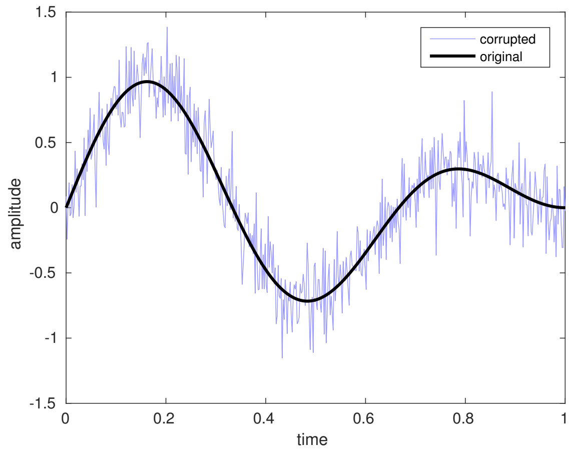
The important points about the formatting are

- The answer is on a single line.
- The first characters on the line is the comment character `%`.
- The answer is labeled in the form `Q`n`:`, where n stands for the question number from the gray box.

If you do not format your answers correctly, the grading software will not find them and you *will* lose points.

Basic Ideas

Consider the following figure.



This is a simulation of a noisy signal. The idea is that a broadcasting station (e.g., a radio transmitter) has sent out a smooth, time-varying signal $s(t)$ (shown with the black curve). However, because of noise in the environment (e.g., scattering from trees, interfering signals, etc.), what we have received is a corrupted version of the signal $s_{\text{cor}}(t)$ (the blue line). The noisy signal follows the general shape of the original signal, but also jumps up and down rapidly in time. Our job is to reconstruct as well as possible the original signal from the noisy signal by filtering out the noise.

Our first step is to convert the problem into the language of linear algebra. To do this, we will *discretize* time. That is, rather than working with the values of $s(t)$ and $s_{\text{cor}}(t)$ for all real numbers t, we'll *sample* s and s_{cor} at a finite set of time values $\{t_1, t_2, \ldots, t_n\}$. That allows us to represent s and s_{cor} as vectors in $\mathbb{R}^n$.

$$\mathbf{x} = \begin{bmatrix} s(t_1) \\ s(t_2) \\ \vdots \\ s(t_n) \end{bmatrix} \qquad \text{and} \qquad \mathbf{x}_{\text{cor}} = \begin{bmatrix} s_{\text{cor}}(t_1) \\ s_{\text{cor}}(t_2) \\ \vdots \\ s_{\text{cor}}(t_n) \end{bmatrix}.$$

Our goal, then, is to use the corrupted signal $\mathbf{x}_{\text{cor}}$ to produce a smoothed signal $\mathbf{y}$ that is hopefully close to the original signal $\mathbf{x}$ ($\mathbf{y} \approx \mathbf{x}$). One way to do this would be to use orthogonal projection onto a suitable basis. This time, however, we'll use a different technique, called *regularization.*

Let $\delta > 0$ be a positive number (called the *regularization parameter*), and, for every $\mathbf{y}$ in $\mathbb{R}^n$, define

$$f(\mathbf{y}) = \|\mathbf{y} - \mathbf{x}_{\text{cor}}\|^2 + \delta^2 \sum_{i=1}^{n-1} (y_{i+1} - y_i)^2.$$

The first term in $f(\mathbf{y})$ measures how close $\mathbf{y}$ is to the received signal $\mathbf{x}_{\text{cor}}$; it is small for $\mathbf{y} \approx \mathbf{x}_{\text{cor}}$ and large for $\mathbf{y} \not\approx \mathbf{x}_{\text{cor}}$. The second term measures how "rough" $\mathbf{y}$ is. If each entry y_{i+1} is not too different from the previous y_i, the sum of squares will be small. But if there are major changes from some y_i to the next y_{i+1}, these will cause the second term of $f(\mathbf{y})$ to be large.

Now, consider choosing $\mathbf{y}$ to make $f(\mathbf{y})$ as small as possible. By our previous analysis, we should expect the best $\mathbf{y}$ to strike a balance between being close to the received signal $\mathbf{x}_{\text{cor}}$ and being not-too-rough. The parameter δ quantifies the trade-off between these two goals. If δ is very close to 0, we won't care too much about roughness and so $\mathbf{y}$ will be very close to $\mathbf{x}_{\text{cor}}$. If δ is very very large, we will focus almost entirely on roughness, and make each value of y_{i+1} approximately equal to the previous:

$$y_n \approx y_{n-1} \approx y_{n-2} \approx \cdots \approx y_2 \approx y_1.$$

By tuning δ correctly, we might hope to balance between these two extremes and find a $\mathbf{y}$ that's not too rough but also not too far from $\mathbf{x}_{\text{cor}}$.

Now, before we jump into the code, we only need to recognize the problem of minimizing $f(\mathbf{y})$ as a least squares problem. To do this, we define the $(n-1) \times n$ matrix $\mathbf{D}$ as

$$\mathbf{D} = \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 1 \end{bmatrix} \qquad \text{so that} \qquad \mathbf{D}\mathbf{y} = \begin{bmatrix} y_2 - y_1 \\ y_3 - y_2 \\ \vdots \\ y_n - y_{n-1} \end{bmatrix}.$$

Then

$$f(\mathbf{y}) = \|\mathbf{y} - \mathbf{x}_{\text{cor}}\|^2 + \delta^2 \|\mathbf{D}\mathbf{y}\|^2.$$

Now we will apply one final standard trick from linear algebra. (You don't need to understand why this works for the lab, but thinking about it after the lab is highly recommended. In working out why the following is true, you will be practicing a number of important skills from this class.) If we define the $(2n-1) \times n$ matrix $\mathbf{A}$ and the $(2n-1)$-vector $\mathbf{b}$ as

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_{n\times n} \\ \delta\mathbf{D} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -\delta & \delta & 0 & \cdots & 0 \\ 0 & -\delta & \delta & \cdots & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\delta & \delta \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} \mathbf{x}_{\text{cor}} \\ \mathbf{0}_{n-1} \end{bmatrix} = \begin{bmatrix} s_{\text{cor}}(t_1) \\ s_{\text{cor}}(t_2) \\ s_{\text{cor}}(t_3) \\ \vdots \\ s_{\text{cor}}(t_n) \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

respectively, then

$$f(\mathbf{y}) = \|\mathbf{A}\mathbf{y} - \mathbf{b}\|^2.$$

Tasks

1. Often, when defining matrices like $\mathbf{A}$ above, it is helpful to begin with n small enough that you can actually look at the matrix and check that it is correct. The first cell of the M-file, titled "A small example" demonstrates this technique. Run the cell and observe the output in the Matlab command window. Then read the code in the cell. Notice that, since $\mathbf{A}$ is a sparse matrix, we are using the sparse matrix functions `speye` and `spdiags` to create $\mathbf{A}$. (Remember, if you don't know how to use a function, you can always use `doc`.)

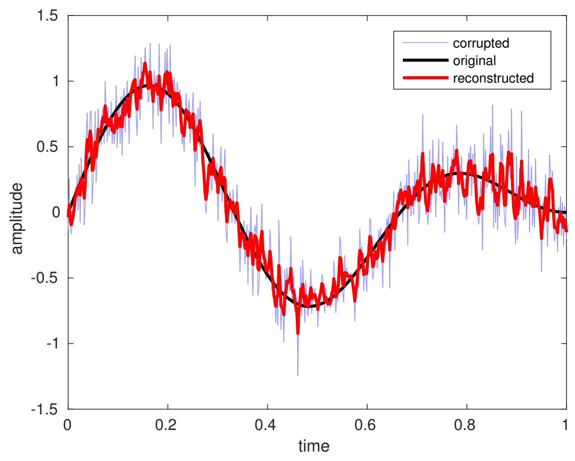
2. Now, run the cell labeled "Part 1." It should produce a figure like the one above. To complete this cell, you need to define the variables `A1` and `b1`, corresponding to $\mathbf{A}$ and $\mathbf{b}$, in the places indicated. (Hint: you can copy and paste from the first cell. Just make sure to rename `A` and `b`.)

   Variables: `A1`, `b1`

   If you look at the documentation for Matlab's backslash operator, you will see that, when applied to an overdetermined system, the result will be the least-squares solution. Thus, the code

   ```
   % find the solution
   y1 = A1\b1;
   ```

   beneath your variable definitions produces the solution $\mathbf{y}$ that we're looking for. If you have defined your variables correctly, running the cell should produce the following figure.



3. As you can see from the figure, the choice $\delta = 1$ for this problem does filter some of the noise, but is still quite rough. Try several other values for δ. Notice that $\delta = 1000$ is too large; the smoothed signal is essentially constant. A δ around 10 or 15 seems fairly good for this example.

4. Now, let's try a different choice of matrix $\mathbf{D}$. We'll use the $(n-2) \times n$ matrix

$$\mathbf{D} = \begin{bmatrix} 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 \end{bmatrix}.$$

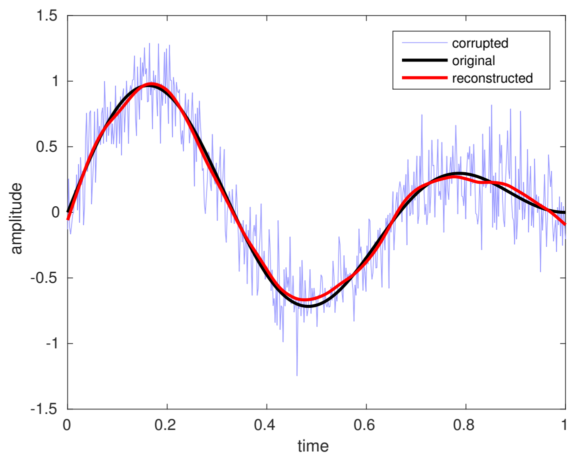
   In the cell labeled "Part 2: Another small example," define the variable `D2` where indicated. Hint:

```
>> e = ones(5,1); full(spdiags( [e,-2*e,e], [0,1,2], 3, 5 ))

ans =

     1    -2     1     0     0
     0     1    -2     1     0
     0     0     1    -2     1
```

   Run the cell and make sure the output looks correct.

   Variables: `D2`

5. In the cell labeled "Part 3," complete the code to solve the problem (similar to "Part 1") with the new choice of $\mathbf{D}$. Be sure the variable names for $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{y}$ are `A3`, `b3` and `y3` respectively. Again, try a few choices for δ. When $\delta = 100$, the figure should look like this:



   Variables: `A3`, `b3`, `y3`

   Q1: Describe the $\mathbf{y}$ that results when $\delta = 10^5$.