

ECE 6250, Fall 2019

Homework #11

Due Tuesday December 3, 11:59pm

Turn in during class on December 2, in Coda S1117 any time, or online.

As stated in the syllabus, unauthorized use of previous semester course materials is strictly prohibited in this course.

1. Please go to <http://gatech.smartevals.com> and fill out survey for ECE 6250. I genuinely appreciate your feedback; I do read and consider carefully the comments for every course that I teach.
2. Using your class notes, prepare a 1-2 paragraph summary of what we talked about in class in the last week. I do not want just a bulleted list of topics, I want you to use complete sentences and establish context (Why is what we have learned relevant? How does it connect with other things you have learned here or in other classes?). The more insight you give, the better.

Interlude: Some notes on MATLAB function handles The next two problems require you to write code that uses function handles. These may be unfamiliar, but are very straightforward. Here is an example:

```
>> f = @(z) z + 2;
```

This defines a function `f` that takes a scalar (or vector) and adds two to it (or every entry):

```
>> f(5)
```

```
ans =
```

```
7
```

Suppose that integers `N` and `K` are already defined in the workspace, and we also define the boxcar filter

```
h = ones(K,1);
```

We can define a function handle `A` that takes a vector of length `N` and convolves it with `h`:

```
>> A = @(z) ifft(fft(z,N+K-1).*fft(h,N+K-1));
```

Of course since \mathbf{A} is a linear mapping from \mathbb{R}^N to \mathbb{R}^{N+K-1} , there is a corresponding matrix (you have generated this matrix on previous homeworks). Here $\mathbf{A}(\mathbf{x})$ implements the action of this matrix on \mathbf{x} without having to generate and store the matrix.

3. Write a MATLAB function `sdsolve.m` that implements the steepest descent algorithm. The function should be called as

```
[x, iter] = sdsolve(H, b, tol, maxiter);
```

where \mathbf{H} is a **function handle** that implements the action of an $N \times N$ symmetric positive definite matrix, \mathbf{b} is a vector of length N , and `tol` and `maxiter` specify the halting conditions. Your algorithm should iterate until $\|r_k\|_2/\|b\|_2$ is less than `tol` or the maximum number of iterations `maxiter` has been reached. For the outputs: \mathbf{x} is your solution, and `iter` is the number of iterations that were performed.

Using function handles instead of explicit matrices is easy. Instead of writing

```
r = b - H*x;
```

like you would if \mathbf{H} is a matrix, you write

```
r = b - H(x);
```

You will want to debug your code using small (sym+def) matrices that you can construct the inverses to explicitly.

When your code is ready for the big-time, download the files `imagedeconv_experiment.m`, `imagedeconv_data.mat`, `imconv.m`, and `imconv_transpose.m`. In the mat file, you will find a 305×305 image \mathbf{Y} and a 50×50 kernel \mathbf{W} . The image \mathbf{Y} was created by convolving a 256×256 image \mathbf{X} with \mathbf{W} . It is your job to figure out what \mathbf{X} must have been.

Of course, the code you wrote for `sdsolve` operates on and returns vectors, not images. But it is easy to turn a $N \times N$ image \mathbf{X} into a vector \mathbf{x} of length N^2 :

```
>> x = reshape(X, N^2, 1);
```

(the shorter `x=X(:)`; also works) and vice versa:

```
>> X = reshape(x, N, N);
```

I have graciously implemented 2D convolution and its transpose for you in the files `imconv.m` and `imconv_transpose.m`.

I have also created some function handles at the beginning of `imagedeconv_experiment.m` that will help you. All you have to do is add a few lines to `imagedeconv_experiment.m` that calls your code and does the recovery. Your solution must have relative residual error $\|r_k\|_2/\|b\|_2$ less than 10^{-4}

Turn in your code, the original image (created using `imagesc(Y); colormap(gray)`), your recovered image, and the number of iterations it took you to reduce the relative residual error to less than 10^{-4} from a starting guess of $\mathbf{0}$.

Important note: The pseudo-code at the bottom of page III.36 of the notes should be your guideline. Note that this uses only one application of H per iteration by using our trick for updating the residual (instead of explicitly calculating it). In practice, you will probably want to calculate the residual explicitly once every 50 iterations. To do this, just put in an `if-then` that substitutes `rk = b - H(xk)` for `rk = rkold - ak*q` every fifty iterations.

- Write a MATLAB function `cgsolve.m` that implements the method of conjugate gradients. The function should be called as

```
[x, iter] = cgsolve(H, b, tol, maxiter);
```

where the inputs and outputs have the same interpretation as in the previous problem.

Use your code to solve the same image deconvolution problem, and comment on the number of iterations required relative to steepest descent. Turn in your code, your recovered image, and the number of iterations it took you to reduce the relative residual error to less than 10^{-4} from a starting guess of $\mathbf{0}$.

You will want to explicitly calculate the residual every 50 iterations here as well.

- Recall the simple “pulse tracking” example we looked at in the Kalman filter notes.
 - Write down the Kalman update equations for this special case. Make them as simple as possible ... (Do you really need to compute $\hat{\mathbf{x}}_{k+1|k}$ and $\mathbf{P}_{k+1|k}$ as intermediate steps?)
 - As k gets large, the solution for $\hat{\mathbf{x}}_{k|k}$ becomes a fixed weighted sum of the previous measurements, i.e. it (approximately) obeys the convolution equation

$$\hat{\mathbf{x}}_{k|k} \approx \sum_{\ell=0}^L w[\ell]y[k - \ell].$$

This equation is extremely accurate for very moderate values of k and L . What is an appropriate value of L and what are the weights $w[\ell]$? (Hint: For a $k = 2, 3, 4, 5, 6, \dots$, create $\underline{\mathbf{A}}_k^T \underline{\mathbf{A}}_k$, invert it, and extract the appropriate row. One “easy” way to create $\underline{\mathbf{A}}_k^T \underline{\mathbf{A}}_k$ is to use the `toeplitz` command and then modify the far upper left and far lower right entries.)

- We are using a radar to track a truck moving in a 2D plane with coordinates (p_x, p_y) . At a series of times t_k indexed by k , we are interesting in estimating its position $(p_x(t_k), p_y(t_k))$ in the plane, and its velocity $(v_x(t_k), v_y(t_k))$ along each coordinate. We stack these into a single vector

$$\mathbf{x}_k = \begin{bmatrix} p_x(t_k) \\ p_y(t_k) \\ v_x(t_k) \\ v_y(t_k) \end{bmatrix}.$$

Although the velocity of the truck will drift, we expect it to remain close to a constant — that is, our best guess for $(v_x(t_{k+1}), v_y(t_{k+1}))$ is simply $(v_x(t_k), v_y(t_k))$. Our best guess for the position at time t_{k+1} is determined by the previous position $(p_x(t_k), p_y(t_k))$, previous velocity

$(v_x(t_k), v_y(t_k))$, and the time $t_{k+1} - t_k$ that has elapsed between the samples. The evolution of the parameters can be modeled by

$$\mathbf{x}_{k+1} = \mathbf{F}_k \mathbf{x}_k + \boldsymbol{\epsilon}_k.$$

At time $t = 0$, we make a direct observation that

$$(p_x(0), p_y(0)) = (0, 0), \quad \text{and} \quad (v_x(0), v_y(0)) = (1, \pi/2).$$

You might write this as $\mathbf{y}_0 = \mathbf{A}_0 \mathbf{x}_0$, where $\mathbf{A}_0 = \mathbf{I}$. At subsequent times $t_k > 0$, we make a single measurement of the position and velocity of the truck. This measurement is of the form

$$y[k] = \cos(1150\pi t_k) p_x(t_k) + \sin(1150\pi t_k) p_y(t_k) + \cos(1250\pi t_k) v_x(t_k) + \sin(1250\pi t_k) v_y(t_k).$$

- (a) Write down the “state evolution equations”. That is, write down the 4×4 matrix \mathbf{F}_k explicitly (it should depend on $t_{k+1} - t_k$).
- (b) The file `kalman_data.mat` contains 499 measurements (in the vector \mathbf{y}) for different times (in the vector \mathbf{t}). Implement a Kalman filter to track the truck, and plot each of your 500 estimates of the position in the (p_x, p_y) plane from $\hat{\mathbf{x}}_{k|k}$ on a set of axes. To plot discrete points rather than a connected line, use something like

```
plot(pxhat, pyhat, 'o')
```

- (c) Construct and solve the (large) system to estimate all 500 positions with knowledge of all of the measurements. That is, find $\hat{\mathbf{x}}_{0|499}, \hat{\mathbf{x}}_{1|499}, \dots, \hat{\mathbf{x}}_{499|499}$. Plot the corresponding position estimates on another set of axes (maybe using `'x'` for the plot command), and overlay your answer from the previous part (maybe using `'o'` for the plot command again). Comment on what you see, in particular on how it related to your answer in the previous part.

7. (Optional) The file `LMS_data.mat` contains the input (sequence \mathbf{u}) and output (sequence \mathbf{y}) of an unknown linear time-invariant system whose impulse response is supported on $n = 0, \dots, 18$.

- (a) Implement the LMS algorithm and use it to estimate the impulse response. Turn in a plot of your estimated filter coefficients. (You can tell how well you did by seeing how well their convolution with \mathbf{u} agrees with \mathbf{y}).
- (b) Try different values of the stepsize μ and comment on the effect it has on the speed of convergence (and whether or not it converges). You might gauge this by how many iterations it takes to get a relative error $\|\mathbf{h}_n - \mathbf{h}_*\|_2 / \|\mathbf{h}_*\|_2 \leq 0.01$.
- (c) Now add a little bit of noise to the observations, using

$$\mathbf{y}_n = \mathbf{y} + \text{sigma} * \text{randn}(\text{length}(\mathbf{y}), 1).$$

Comment on the convergence speed when `sigma = 0.05`. You might have to adjust your step size to ensure convergence.