ECE 6250, Fall 2015

Homework # 9

Due Wednesday November 16, in class

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

1. Using you 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.

2. Let $\boldsymbol{A}$ be a tri-diagonal matrix:

$$\boldsymbol{A} = \begin{bmatrix} d_1 & c_1 & 0 & 0 & 0 & \cdots & 0 \\ f_1 & d_2 & c_2 & 0 & 0 & \cdots & 0 \\ 0 & f_2 & d_3 & c_3 & 0 & \cdots & 0 \\ 0 & 0 & f_3 & d_4 & c_4 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & f_{N-2} & d_{N-1} & c_{N-1} \\ 0 & 0 & 0 & \cdots & & f_{N-1} & d_N \end{bmatrix}$$

   (a) Argue that the LU factorization of $\boldsymbol{A}$ has the form

$$\boldsymbol{A} = \begin{bmatrix} * & 0 & 0 & 0 & \cdots & 0 \\ * & * & 0 & 0 & \cdots & 0 \\ 0 & * & * & 0 & \cdots & 0 \\ 0 & 0 & * & * & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & * & * \end{bmatrix} \begin{bmatrix} * & * & 0 & 0 & 0 & \cdots & 0 \\ 0 & * & * & 0 & 0 & \cdots & 0 \\ 0 & 0 & * & * & 0 & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & \\ 0 & 0 & \cdots & & & * & * \\ 0 & 0 & \cdots & & & 0 & * \end{bmatrix},$$

   where $*$ signifies a non-zero term.

   (b) Write down an algorithm that computes the LU factorization of $\boldsymbol{A}$, meaning the $\{\ell_i\}, \{u_i\}$, and $\{g_i\}$ below

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ \ell_1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & \ell_2 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \ell_3 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & \ell_{N-1} & 1 \end{bmatrix} \begin{bmatrix} u_1 & g_1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & u_2 & g_2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & u_3 & g_3 & 0 & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & \\ 0 & 0 & \cdots & & & u_{N-1} & g_{N-1} \\ 0 & 0 & \cdots & & & 0 & u_N \end{bmatrix},$$

I will get you started:

$$
\begin{aligned}
&u_1 = d_1 \\
&\text{for } k = 2, \ldots, N \\
&\qquad g_{k-1} = \\
&\qquad \ell_{k-1} = \\
&\qquad u_k = \\
&\text{end}
\end{aligned}
$$

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 `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 `A(x)` implements the action of this matrix on `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 `H` is a **function handle** that implements the action of an $N \times N$ symmetric positive definite matrix, `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: `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 `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 `Y` and a 50×50 kernel `W`. The image `Y` was created by convolving a 256×256 image `X` with `W`. It is your job to figure out what `X` must have been.

Of course, the code you wrote for `sdsolve` operates on and returns vectors, not images. But is is easy to turn a $N \times N$ image `X` into a vector `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`, and 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 on page III.33 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.

4. 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.