Problem Set 2 Solutions

1. Let $A = \begin{bmatrix} -2 & 2 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$

   (a) Find $||A||_1, ||A||_2$, and $||A||_\infty$. (You may check these with Matlab, but you should show the work to do them by hand.)

   $||A||_1 = \max\{2+0+1, 2+0+1, 0+1+0\} = 3$

   $||A||_\infty = \max\{2+2+0, 0+0+1, 1+1+0\} = 4$

   $||A||_12 = \sqrt{\max\{\lambda_i\}}$ for $\lambda_i, e =$ eigenvalues of $AA^\intercal$

   $$AA^\intercal = \begin{bmatrix} -2 & 2 & 0 \\ 0 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} -2 & 0 & -1 \\ 2 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

   $$= \begin{bmatrix} 8 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

   Therefore, the eigenvalues are $\lambda_1 = 8, \lambda_2 = 1, \lambda_3 = 2$.

   Therefore, $||A||_2 = \sqrt{8} = 2\sqrt{2}$.

   (b) Find $A\mathbf{x}$ for $\mathbf{x} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$ and show that $||A\mathbf{x}|| \leq ||A||||\mathbf{x}||$ for all three norms.

   First of all, $||\mathbf{x}||_1 = 1+1+1 = 3$,

   $||\mathbf{x}||_2 = \sqrt{1+1+1} = \sqrt{3}$,

   $||\mathbf{x}||_\infty = 1$

   Then, $A\mathbf{x} = \begin{bmatrix} -4 \\ 1 \\ 0 \end{bmatrix}$.

   Then, $||A\mathbf{x}||_1 = 4+1 = 5$

   $||A\mathbf{x}||_2 = \sqrt{1+16} = \sqrt{17}$

   $||A||_\infty = \max\{1, 4\} = 4$

   Then, for the 1-norm, we have $5 \leq (3)(3) = 9$, which is true.

   For the 2-norm, we get $\sqrt{17} \approx 4.12 \leq (2\sqrt{2})(\sqrt{3}) \approx 4.9$, which is true.

   Lastly, for the max norm, we get $4 \leq (4)(1)$, which is true.

(c) Find $\kappa_1(A)$, and $\kappa_\infty(A)$, the relative condition numbers for the matrix A, with respect to the 1 and max norms, respectively. You may use Matlab to find A^{-1}.

Using Matlab, we find that $A^{-1} = \begin{bmatrix} -0.25 & 0 & -0.5 \\ 0.25 & 0 & -0.5 \\ 0 & 1 & 0 \end{bmatrix}$.

Then, $|||A^{-1}||_1 = \max\{0.25 + 0.25 = 0, 0 + 0 + 1, 0.5 + 0.5 + 0\} = 1$

$||A^{-1}||_\infty = \max\{0.25 + 0 + 0.5, 0.25 + 0 + 0.5, 0 + 1 + 0\} = 1$

Then $\kappa_1 = ||A||_1||A^{-1}||_1 = 3(1) = 3$

$\kappa_\infty = ||A||_\infty||A^{-1}||_\infty = 4(1) = 4$

(d) Based on your answers in part (c), how many digits of accuracy do you expect to have when performing matrix-vector multiplication with this matrix A (in double precision)? Explain.

I would expect to have about 15-16 digits of accuracy, losing at most one digit from the 16 digits I would have for storing the matrix and vector. This is because the matrix is well-conditioned, with a condition number fairly close to 1, which means that any errors (such as those that come about by storing numbers as floating point numbers) are not amplified very much in the resulting output errors.

2. Consider the matrix $V \in \mathbb{R}^{n\times n}$ whose components are expressed as

$$V_{ij} = \begin{cases} \sum_{k=1}^{i+j} \frac{1}{k} & \text{if } i \leq j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

(a) Write down the matrix V for $n = 3$.

$$V = \begin{bmatrix} 1+\frac{1}{2} & 1+\frac{1}{2}+\frac{1}{3} & 1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4} \\ 0 & 1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4} & 1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5} \\ 0 & 0 & 1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6} \end{bmatrix} = \begin{bmatrix} \frac{3}{2} & \frac{11}{6} & \frac{25}{12} \\ 0 & \frac{25}{12} & \frac{137}{60} \\ 0 & 0 & \frac{147}{60} \end{bmatrix}$$

2b Pseudocode

- Input: n, size of matrix
- Output: V, nxn matrix

```
• V = zeros(n,n)          (initialize V)
• for i=1 to n            (loop over rows)
    • for j= i to n       (loop over columns & don't fill in when j < i)
        • sum = 0         (initialize sum)
        • for k = 1 to i+j    (do sum)
            • sum = sum + 1/k     (add on pieces of sum)
        end
        • V_ij = sum      (store component of V)
    end
end
```

2c FLOP count

$$\underbrace{\sum_{i=1}^{n}}_{\text{outer loop}} \underbrace{\sum_{j=i}^{n}}_{\text{The 2nd loop only runs for } j \geq i} \underbrace{2(i+j)}_{+ \,\&\, \text{division}} = O(n^3)$$

→ The number of steps of inner loop is O(n) (worst case, 2n). The number of steps of middle loop is O(n) (worst case, n, when i=1) The number of steps of outer loop is n.

You don't need to, but to find the FLOP count exactly:

$$2 \sum_{i=1}^{n} \sum_{j=i}^{n} (i+j)$$

$$= 2\left[\sum_{i=1}^{n} \sum_{j=i}^{n} i + \sum_{i=1}^{n} \sum_{j=i}^{n} j \right]$$

$$= \sum_{i=1}^{n} \left((n-i+1)\, i \right)$$

$$= (n+1) \sum_{i=1}^{n} i - \sum_{i=1}^{n} i^2$$

$$= (n+1)\frac{(n)(n+1)}{2} - \frac{n(n+1)(2n+1)}{6}$$

$$= \frac{(n^2+2n+1)n}{2} - \frac{(n^2+n)(2n+1)}{6}$$

$$= \frac{n^3+2n^2+n}{2} - \frac{2n^3+3n^2+n}{6}$$

$$= \frac{3n^3+6n^2+3n-2n^3-3n^2-n}{6}$$

$$= \frac{n^3+3n^2+2n}{6}$$

$$= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} j - \sum_{j=1}^{i-1} j \right)$$

$$= \sum_{i=1}^{n} \left(\frac{n(n+1)}{2} - \frac{(i-1)(i)}{2} \right)$$

$$= \frac{n^2(n+1)}{2} - \sum_{i=1}^{n} \frac{i^2-i}{2}$$

$$= \frac{n^2(n+1)}{2} - \frac{1}{2}\sum_{i=1}^{n} i^2 + \frac{1}{2}\sum_{i=1}^{n} i$$

$$= \frac{n^2(n+1)}{2} - \frac{n(n+1)(2n+1)}{12} + \frac{n(n+1)}{4}$$

$$= \frac{n^3+n^2}{2} - \frac{(n^2+n)(2n+1)}{12} + \frac{n^2+n}{4}$$

$$= \frac{6n^3+6n^2-2n^3-3n^2-n+3n^2+3n}{12}$$

$$= \frac{4n^3+6n^2+2n}{12}$$

$$= \frac{2n^3+3n^2+n}{6}$$

$$\text{Total} = 2\left(\frac{3n^3+6n^2+3n}{6}\right)$$

$$= n^3+2n^2+n$$

$$= O(n^3)$$

3. An upper bidiagonal matrix is a matrix with a main diagonal and one upper diagonal:

$$A = \begin{bmatrix} a_{11} & a_{12} & & & 0 \\ & a_{22} & a_{23} & & \\ & & \ddots & \ddots & \\ & & & a_{n-1,n-1} & a_{n-1,n} \\ 0 & & & & a_{nn} \end{bmatrix} \tag{2}$$

(a) For a linear system $A\mathbf{x} = \mathbf{b}$, derive expressions for x_n and x_k $(k = n-1, \cdots, 1)$.

This is very similar to a regular upper triangular matrix, in that we should use back substitution to solve it since we can get x_n immediately. The solutions are thus:

The last equation is $a_{nn}x_n = b_n \implies x_n = \dfrac{b_n}{a_{nn}}$.

Next, we get $a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1} \implies x_{n-1} = \dfrac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}$. Similarly, we get $a_{n-2,n-2}x_{n-2} + a_{n-2,n-1}x_{n-1} = b_{n-2} \implies x_{n-2} = \dfrac{b_{n-2} - a_{n-2,n-1}x_{n-1}}{a_{n-2,n-2}}$ and so on.

Similarly, looking at $k = 1$, we would have $a_{11}x_1 + a_{12}x_2 = b_1 \implies x_1 = \dfrac{b_1 - a_{12}x_2}{a_{11}}$

To summarize, our solutions are $x_n = \dfrac{b_n}{a_{nn}}$ and $x_k = \dfrac{b_k - a_{k,k+1}x_{k+1}}{a_{kk}}$ fpr $k = n-1, ..., 1$

(b) Write a pseudocode to solve the linear system that uses only the nonzero element of A.

Again, the code will be very similar to Back Substitution, only with fewer terms since we can ignore the terms that are 0. See the next page for example Pseudocode.

4.

3b

```
• input : $A$, $\overset{\text{upper}}{\text{bidiagonal}}$ $n \times n$ matrix ; $\underline{b}$, $n \times 1$ vector
• output : $\underline{x}$, $n \times 1$ s.t. $A\underline{x} = \underline{b}$
• $x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \Big\} n$
~~• for i = n-1 : -1 : 1~~
• $x_n = \dfrac{b_n}{a_{nn}}$
• for $i = n-1 : -1 : 1$        $(n-1, n-2, \ldots, 3, 2, 1)$
      $x_i = \dfrac{b_i - a_{i,i+1} x_{i+1}}{a_{ii}}$
end.
```

3c FLOP count $= \overset{\text{finding } x_n}{1} + \overset{\text{One subtract, one mult., one div.}}{3}\underset{\text{\# of steps in loop.}}{(n-1)}$

$$= \boxed{3n-2}$$

$O(n)$

④

$$\begin{bmatrix}1&0&0&1\\-1&2&0&1\\-1&-1&1&1\\-1&3&-1&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}-3\\3\\0\\2\end{bmatrix}$$

- First, we do Gaussian Elimination to find for L & U:

Form U:

$$\begin{bmatrix}1&0&0&1\\-1&2&0&1\\-1&-1&1&1\\-1&3&-1&1\end{bmatrix}\begin{array}{l}\\+R1\\+R1\\+R1\end{array}\rightarrow\begin{bmatrix}1&0&0&1\\0&2&0&2\\0&-1&1&2\\0&3&-1&2\end{bmatrix}\begin{array}{l}\\ \\+\frac{1}{2}R2\\-\frac{3}{2}R2\end{array}\rightarrow\begin{bmatrix}1&0&0&1\\0&2&0&2\\0&0&1&3\\0&0&-1&-1\end{bmatrix}\begin{array}{l}\\ \\ \\+R3\end{array}$$

$$\rightarrow\underbrace{\begin{bmatrix}1&0&0&1\\0&2&0&2\\0&0&1&3\\0&0&0&2\end{bmatrix}}_{U}\qquad L=\begin{bmatrix}1&0&0&0\\-1&1&0&0\\-1&-\frac{1}{2}&1&0\\-1&3/2&-1&1\end{bmatrix}$$

So $A=LU$ for U & L defined above.

- Now, to solve for $\underline{x}$, first solve $L\underline{y}=\underline{b}$ for $\underline{b}$ w/ forward sub:

ie solve

$$\begin{bmatrix}1&0&0&0\\-1&1&0&0\\-1&-1/2&1&0\\-1&3/2&-1&1\end{bmatrix}\begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix}=\begin{bmatrix}-3\\3\\0\\2\end{bmatrix}$$

$y_1 = -3$

$y_2 = 3 + y_1 = 3 - 3 = 0$

$y_3 = 0 + y_1 + \frac{1}{2} y_2 = -3$

$y_4 = 2 + y_1 - \frac{3}{2} y_2 + y_3 = 2 - 3 + 0 - 3 = -4$

$$\underline{y} = \begin{bmatrix} -3 \\ 0 \\ -3 \\ -4 \end{bmatrix}$$

- Next, we solve $U\underline{x} = \underline{y}$ w/ backward sub:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ -3 \\ -4 \end{bmatrix}$$

$x_4 = \frac{-4}{2} = -2$

$x_3 = -3 - 3x_4 = -3 - 3(-2) = -3 + 6 = 3$

$x_2 = \frac{0 - 2x_4}{2} = \frac{-2(-2)}{2} = 2$

$x_1 = -3 - x_4 = -3 + 2 = -1$

$$\boxed{\underline{x} = \begin{bmatrix} -1 \\ 2 \\ 3 \\ -2 \end{bmatrix}}$$

5. (a) Write a Matlab function to perform forward substitution to solve a linear system $L\mathbf{x} = \mathbf{b}$. Your function should take as input L, an $n \times n$ lower triangular matrix with no zeros on the diagonal, and $\mathbf{b}$, a vector of length n. It should return the solution $\mathbf{x}$ as output.

See forwardsub.m for the function.

(b) Validate your code by forming a 10×10 lower triangular matrix for L made up of random numbers between 0 and 20, as well as a similarly formed random vector, $\mathbf{b}$. Solve for $\mathbf{x}$ using both your function and Matlab function `backslash`. Then find the relative difference of your solution from the Matlab solution, with respect to the max norm. (i.e. find the relative "error" of your vector, using the Matlab vector as the "exact" solution).

See the file PS2N5.m for the code. An example of randomly formed L and $\mathbf{b}$ gave a difference with a relative max norm of $3.789(10)^{-16}$.

(c) Comment on the size of your relative difference found in part (b). Why is *error* written in quotations? (Why is this not a relative error?) Lastly, how many digits would you say that you can "trust" for each component of your solution $\mathbf{x}$ (specifically for the matrix generated for part (b))? (And why?)

The answer in part (b) is pretty much down at ϵ_{mach}, so this confirms that the code is working. However, this is not the true size of the *error*. To find the size of the error, we would need to compare to the exact solution, $\mathbf{x}$, not the Matlab calculated solution. I checked the condition number, with respect to the max norm, of the matrix in part (b), and I got $7.18(1)^3$. This tells me that I can lose about 4 digits of accuracy in solving this system. Therefore, I can only "trust" about 12 digits of any of the components of my solution $\mathbf{x}$. The same is true for the Matlab algorithm.

The rest of this is for your information (not expected to be written): Many times, one would expect the relative difference between two different algorithms to be about the same size as the true error. To illustrate, imagine that my last 4 digits (of 16 digits) are incorrect, and that Matlab's last 4 digits are also incorrect, but they are also different from each other. Then the difference would be about the same size as the true error. However, I am not seeing that here. They are essentially agreeing on all 16 digits. The reason is that our two algorithms are really the same. Matlab's backslash function actually checks to see if a matrix is triangular. If it is lower triangular, it will employ forward substituion, just like we did. The algorithm for forward substitution is quite straightforward, so their version would look just like ours. Hence, we see no relative difference between our solution and Matlab's solution (down to machine epsilon).

6. (a) Write a Matlab function to perform backward substitution to solve a linear system $U\mathbf{x} = \mathbf{b}$. Your function should take as input U, an $n \times n$ upper triangular matrix with

no zeros on the diagonal, and **b**, a vector of length n. It should return the solution **x** as output. (Note: As we did not write pseudocode in class, it is advised that you begin by writing pseudocode before writing Matlab code.)

See backwardsub.m for the function.

(b) Validate your code by forming a 10×10 upper triangular matrix for U made up of random numbers between 0 and 20, as well as a similarly formed random vector, **b**. Solve for **x** using both your function and Matlab function `backslash`. Then find the relative difference of your solution from the Matlab solution, with respect to the max norm.

See the file PS2N6.m for the code. An example of randomly formed U and **b** gave a relative difference with a max norm of $8.9(10)^{-16}$, confirming that the code works. (This part is not asked for: The condition number, with respect to the max norm, is about $3.34(10)^2$, so I would expect to lose about about 2-3 digits of accuracy, so if I was able to check the exact relative error, I would expect it to be around 10^{-13})

7. (a) Write a Matlab function to use Gaussian Elimination (without pivoting) to perform LU decomposition for a matrix A. Your function should take as input A, an $n \times n$ matrix. Its output should be the lower triangular and upper triangular matrices, L and U.

See myLUdecomp.m for the function.

(b) Use your code to find the LU decomposition to the matrix in problem (4).

See PS2N7.m for the code.

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & -0.5 & 1 & 0 \\ -1 & 1.5 & -1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

These match what I got in (4).

(c) Also validate your code by forming a 1000×1000 matrix for A_1 made up of random numbers between 0 and 20. Use your function to find L and U, find the difference between A and LU, and then report the component of the difference matrix that has the largest magnitude. (Hint: Make sure to use semi-colons to suppress the output - you do not want to print out A,L,or U.)

See PS2N7.m for the code. The maximum component of the error is $1.5(10)^{-10}$. (You were not asked to comment on this, but you may wonder if this is too large. The size of the relative difference is about one order of magnitude smaller, and since the condition numbers of $A, L,$ and U are all quite large, and to find the difference, we are multiplying L and U, this is a reasonably sized difference to get.

(d) In class, we discussed that the FLOP count for this operation is $O(n^3)$. Knowing that, what do you expect to happen to the run time if you double the "size" of the problem, n? Test this by `tic toc` and and running your code on A_1, a 1000×1000 matrix and then A_2, a 2000×2000 matrix. Give the run times, and comment on your results (Is it what you expected? If not, what do you expect is the reason? Also, comment on the scalability of this algorithm.)

I would expect the time to be multiplied by 8 since the FLOP count (and hence cost) would be multiplied b 8. Whether or not you see this will likely depend on your computer and/or on which algorithm you employed. Your comments ill depend on that. Below are comments based on my code. You should also comment on the scalability.

If I use LUdecomp.m, which was the algorithm which used two for loops, the run-time for $n = 1000$ is about 4 seconds, and the run-time for $n = 2000$ is about 48 seconds. This is a little over what I was expecting (32 seconds). Initially, knowing that the "order" is in the limit that $n \longrightarrow \infty$, I thought it was possible that $n = 1000$ was not large enough to see this ratio. However, then I simplified and got the exact FLOP count, $\frac{2}{3}n^3 - \frac{1}{2}n^2 - \frac{1}{6}n$. As I discussed in class, if we do the ratio of this for $n = 2000$ and $n = 1000$, we see that these numbers are in fact large enough to have a common ration of about 8. Other people, running the same code on their computers were able to get a ratio of 8, leading me to believe it is my computer - it could be the amount of programs running in the background, or who knows. This is a good illustration as to the fact that while FLOP count is an accurate representation of work of an algorithm, and run-time is a more practical/useful representation of work, they do not always match up exactly.

Additionally, if I instead use LUdecomp2.m, which was the algorithm only using one for loop, the run time for $n = 1000$ is about 1.4 seconds, and the run time for $n = 2000$ is about 11.9 seconds. This gives a ratio very close to 8, which is what I expected. While both of these functions have the same FLOP count, the code is usually faster/more efficient if you use fewer for loops.

The run-times for the first algorithm illustrate that this algorithm for LU deposition can become expensive quite quickly. Simply doubling n makes the run time go from a few seconds to almost a minute. Using $n = 8000$ would make my run-time somewhere between 51 and 115 minutes. Therefore, for very large matrices, this algorithm is not practical, as it could take days or longer. (Try timing `lu(A2)`, and you will see it takes less than one tenth of a second, so there are of course better algorithms than the simple one we have implemented thus far.)