Math 551 Fall 2018
Lab 5

Goal: In this lab, we will consider the "best" way to solve a linear system of the form $\mathbf{Ax} = \mathbf{b}$. Of course, in order to understand what "best" means, we need some means of comparison. For this exercise, we will compare methods based on their *accuracy* and *efficiency*.

Accuracy becomes important when we realize that computers cannot possibly represent all real numbers. Instead, in numerical computations, operations on real numbers require rounding to some finite number of digits. The errors caused by rounding propagate through computations and give rise to errors in the output. A more accurate algorithm is one that has smaller output errors for a given set of input data.

Efficiency often refers to the time required for an algorithm to complete, or to the amount of system memory required for the algorithm to run. We'll focus on the former. For us, a more efficient algorithm will be one that gives us the answer quicker.

In this lab, we will explore three different methods for solving a square linear system $\mathbf{Ax} = \mathbf{b}$ in Matlab: the `rref` function, the `inv` function, and the backslash operator.

Getting started: Download the file `m551lab05.m` and load it into Matlab's editor when instructed. Throughout this lab, you will make modifications to this file. You will need to turn in your modified version, so be sure to save your changes.

What you have to submit: Submit your modified version of the M-file through Canvas.

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.

Timing Computations in Matlab

1. At the Matlab command prompt run the command `tic`, then wait a few seconds and run the command `toc`. You should see something like the example output below. Repeat a few times, pausing for different lengths of time between the two commands. What is happening?

```
>> tic
>> toc
Elapsed time is 3.065801 seconds.
```

2. Now load the file `m551lab05.m` into the Matlab editor, and find and run the section labeled "Timing Multiplications in Matlab". (If you don't know how to run a single section, look for an appropriate button in Matlab's toolbar or ask your lab instructor for help.) Matlab's output should now contain a table similar to this one.

```
|     n | avg. time |
+-------+-----------+
|    10 | 1.077e-04 |
|    20 | 1.840e-05 |
|    40 | 8.951e-04 |
|    80 | 1.859e-04 |
|   160 | 3.420e-04 |
```

   Don't worry about understanding all the code. It uses `tic` and `toc` to time how long it takes Matlab to multiply two random square matrices of various dimensions. The table column labeled `n` shows the dimension of the matrices (10×10, 20×20, and so on) and the `avg. time` column reports the length of time (in seconds) required to multiply two $n \times n$ matrices for this value of n. Unfortunately, these sizes are too small for us to get a good sense of how expensive matrix multiplication is. We'll fix this in the next step.

3. Change the definition of `mul_n` so that the table includes the n values 100, 200, 400, 800 and 1600. (You'll need to run the section again.)

   Variables: `mul_n`

4. Look at the bottom two rows of this table. When n is doubled from 800 to 1600, how is the computational time affected? Is the time doubled? Tripled? Multiplied by some other factor? (This method of assessing an algorithm's behavior is not the best. You'll probably get a different answer from your classmates, but it should give you a rough idea.)

   Q1: How does the computational time change when n is doubled from 800 to 1600?

Methods for Solving Linear Systems

Now let's consider three different ways for solving $\mathbf{Ax} = \mathbf{b}$ when

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & -2 \\ 0 & 1 & -1 \\ 2 & 1 & 2 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \\ 10 \end{bmatrix}.$$

(The solution is $\mathbf{x} = [1\ 2\ 3]^T$.)

1. In the M-file, locate the Matlab comments

```
%%
% Methods for Solving Linear Systems

% define A1 and b1 here
```

Directly below these lines, define a variable `A1` holding the matrix $\mathbf{A}$ and a variable `b1` holding the vector $\mathbf{b}$.
Variables: `A1`, `b1`

2. This section of the M-file should now look like

```
%%
% Methods for Solving Linear Systems

% define A1 and b1 here
A1 = ...
b1 = ...

M1 = [ A1 b1 ]
S1 = rref(M1)

x1_rref = S1(:,4)
```

Run the section and observe that the variable `x1_rref` is assigned the values from the last column of `S1`, which contains the computed solution $\mathbf{x}$. Beneath this variable definition, define two other variables, `x1_inv` and `x1_div`. The first should hold the result of multiplying `inv(A1)` with `b1` and the second should hold the result of `A1\b1`. After running this section, check in the Matlab command window that all three solution variables agree.
Variables: `x1_inv`, `x1_div`

Comparing the Methods

Now let's compare the methods on a larger matrix. Select and run the section "Comparing the Methods" in the M-file. The output should look something like this:

```
| method |   time   |   error   | residual  |
+--------+----------+-----------+-----------+
|  rref  |  1.36167 | 9.581e-08 | 9.658e-15 |
| inv(A) |       not yet implemented        |
|  A\b   |       not yet implemented        |
```

Currently, only the solution method based on `rref` is implemented. The `time` column reports the time required to solve a moderately large (200×200) linear system. The next two columns report two measures of the error: the solution error and residual error. These two quantities are defined as follows.

Suppose the true solution to the system $\mathbf{Ax} = \mathbf{b}$ is the vector $\mathbf{x}_t$. When solving the system in Matlab, we typically won't get the true solution due to the accumulation of roundoff errors. Instead, we'll get an approximate solution $\mathbf{x}_a$. One way to measure error is through the difference $\mathbf{x}_t - \mathbf{x}_a$, called the *solution error vector*.

The biggest problem with this measure of error is that we typically won't know the true solution. In this exercise, the vector $\mathbf{b}$ was constructed so that the solution is known, but usually we won't be looking for an answer we already know. If we don't know the true solution, we can still try to understand the error through the *residual error vector*, $\mathbf{r}$, defined as $\mathbf{r} = \mathbf{Ax}_a - \mathbf{b}$. If $\mathbf{x}_a = \mathbf{x}_t$, the residual will be the zero vector. If $\mathbf{x}_a$ is a good approximation, we expect that $\mathbf{r}$ should have very small entries.

In either case, the error is a vector. In order to compare methods, we need a way to describe the sizes of these vectors. We'll discuss methods for assigning size to vectors in more detail in later lectures on *vector norms*. For now, it is enough to know that the third and fourth columns of the Matlab table describe the sizes of the two error vectors described above. Larger values indicate more error.

1. Locate the following code.

```
% solve with rref
tic;
S = rref([A b]);
x_rref=S(:,n+1);
rref_time = toc;
```

This code computes the solution to the system using `rref`, and stores the result in `x_rref` and the solution time in `rref_time`. Create a similar block of code below this one that computes `x_inv` using multiplication by `inv(A)` and store the computation time in the variable `inv_time`. Run the section again and compare the top two table rows. If you have completed this step successfully, the `inv`-based method should be much faster than `rref`, have similar solution error, but larger residual by several orders of magnitude.

Variables: `x_inv`, `inv_time`

2. Finally, make one more block of code which approximates the solution `x` using the backslash operator. Store the result in `x_div` and the computation time in the variable `div_time`.

   Variables: `x_div`, `div_time`

3. When you are finished, running the section should print a table comparing computing times and errors among all three methods.

   Q2: How do the methods compare in computing time?

   Q3: How do the methods compare in solution error?

   Q4: How do the methods compare in residual error?

   Q5: Which method do you think is "best" for this system and why?