# AMATH 352: Problem Set 7

Instructor: Brian de Silva

February 27, 2017

## Due: Friday March 3, 2017

## Instructions:

Complete the following problems. Submit two Matlab files to Scorelator; one which generates the dat files containing your solutions to the problems in the Matlab section and `rel_err.m`, your implementation of the relative error function. You have five chances to submit your assignment. Your score for the Matlab portion of the assignment will be the highest score from your five submissions. The remaining problems (1-4) make up the written portion of this assignment. Turn in a write up of these problems digitally (via Canvas) by 5:00pm of the due date.

---

## LU factorization:

1. In deriving the LU factorization we implicitly relied on the fact that the product of two lower triangular matrices is also lower triangular. Prove this result for the specific case of lower triangular matrices in $\mathbb{R}^{3\times 3}$. That is prove that the product of two lower triangular $3 \times 3$ matrices is also lower triangular.

2. Using the results from lecture, what is the inverse of each of the following matrices?

   (a) $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & m_1 & 1 \end{bmatrix}$

   (b) $\begin{bmatrix} 1 & 0 & 0 \\ m_1 & 1 & 0 \\ m_2 & 0 & 1 \end{bmatrix}$

   (c) $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 9 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 6 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ 8 & 0 & 0 & 1 \end{bmatrix}$

3. Compute, by hand, the LU decomposition of the following matrices. Show your work.

   (a) $\begin{bmatrix} 2 & 1 & 3 \\ 4 & 1 & 2 \\ 0 & 7 & 8 \end{bmatrix}$

   (b) $\begin{bmatrix} 2 & 1 & 2 & 3 \\ 4 & 0 & 5 & 11 \\ 12 & 10 & 15 & -13 \\ 8 & 6 & 12 & -1 \end{bmatrix}$

4. Given the following LU factorization $\boldsymbol{A} = \boldsymbol{LU}$ and the vectors $\boldsymbol{x}, \boldsymbol{y}$:

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{U} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}$$

compute

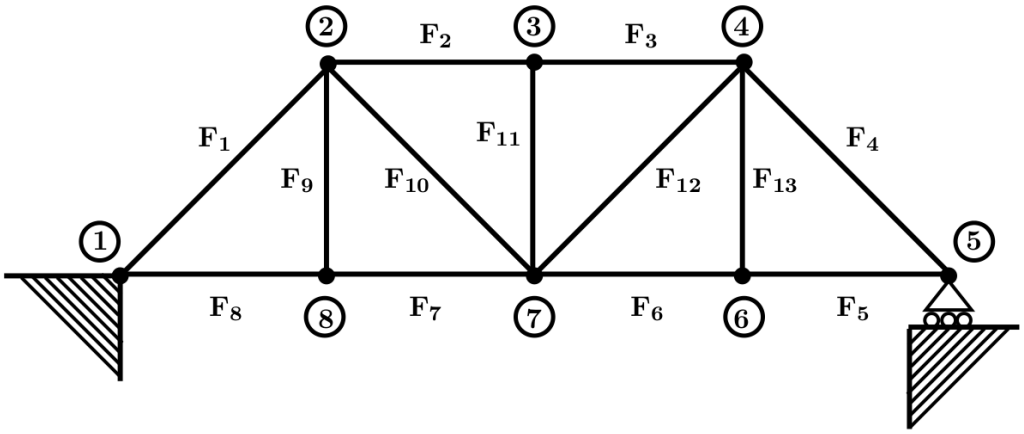
$$\boldsymbol{A}^{-1}\boldsymbol{x} + \boldsymbol{A}^{-2}\boldsymbol{y}$$

without forming $\boldsymbol{A}$, $\boldsymbol{A}^2$, $\boldsymbol{A}^{-1}$, or $\boldsymbol{A}^{-2}$ explicitly. Show your work.

# Matlab:

Create and submit to Scorelator a single Matlab script (.m file) which performs the following tasks along with the function file `rel_err.m` which contains the implementation of your relative error function. Your code should generate the dat files `A1.dat`, `A2.dat`,..., and `A7.dat`.

5. **Forces on a Bridge:** Consider the bridge truss shown below.




Given a vector of external forces $\boldsymbol{b}$ at any of the positions 1-13, we can compute the forces $\boldsymbol{x} = [\boldsymbol{F_1}, \boldsymbol{F_2}, \ldots, \boldsymbol{F_{13}}]^T$ by solving the system

$$\boldsymbol{Ax} = \boldsymbol{b},$$

where $\boldsymbol{A}$ is given by

$$\boldsymbol{A} = \begin{bmatrix} -s & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & s & 0 & 0 & 0 \\ -s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -s & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s & 0 \\ 0 & 0 & 0 & -s & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -s & -1 \\ 0 & 0 & 0 & -s & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & -s & 0 & s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & s & 1 & s & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and $s = \sqrt{2}/2$.

We will solve for the vector of forces $\boldsymbol{x}$ assuming that there are 5 ton vehicles sitting at nodes 6, 7 and 8. This means that $\boldsymbol{b} = [0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 5, 0, 5]^T$.

(a) Solve for $\boldsymbol{x}$ using the LU-decomposition. (Use the `lu` command in Matlab.) Save the intermediate answer $\boldsymbol{y}$ as `A1.dat` and the final answer $\boldsymbol{x}$ as `A2.dat`.

(b) Solve for $\boldsymbol{x}$ using the backslash command. Save your answer as `A3.dat`. To solve the equation $\boldsymbol{Ax} = \boldsymbol{b}$ for $\boldsymbol{x}$ with the backslash command in Matlab, simply type `x = A \b`. This is roughly equivalent to setting `x` equal to `inv(A) * b` (`inv(A)` computes the inverse of `A`), but it is more stable, efficient, and is the preferred way to solve the system.

(c) Now suppose that we add weight to the middle truck (which corresponds to the 11th entry of $\boldsymbol{b}$) in increments of 0.01 tons until the bridge collapses. Each bridge member is rated for no more than 30 tons of compression or tension (i.e., positive or negative forces.) That is, the bridge will collapse when the absolute value of the largest force exceeds 30. Find the weight of the middle truck at the exact moment the bridge collapses. You should re-use the LU-decomposition you computed earlier in the problem. Save your answer as `A4.dat`.

Note: you need to find the lowest weight of the middle truck such that the maximum force at any one point is greater than or equal to 30.

*Hint*: You can find the absolute value of the largest entry in a vector $\boldsymbol{x}$ using the infinity norm. In Matlab, this is `norm(x,Inf)`.

6. Watch the video tutorial on writing a Matlab function that performs Gaussian Elimination (and, with a slight modification, LU factorization), located at `https://youtu.be/QiZ-geoAaaM`. If you are comfortable enough with Matlab and the LU factorization that you think you can discern what is happening in the code below, you may choose to skip the video.

Code similar to that in the video is given below. Notice that the method of Gaussian Elimination presented in the video is slightly different from the one presented in class. Namely it computes a different ratio, `fac`, than was introduced in class, which causes the row replacement procedure to change marginally. Here we have included code that *is* associated with the method from class.

```
function [A] = GE_NMM(A)
% Gaussian Elimination code
%A = [1 6 2 0; 3 2 4 -1; 1 6 1 1; 2 1 3 0];
N = length(A);

% for each pivot along the diagonal
for ii = 1:N-1
    % for each row under the pivot
    for jj=ii+1:N
            % factor is ratio between pivot value
            % and entry below pivot in row jj
            fac = A(jj,ii) / A(ii,ii);

            % row replacement operation
            A(jj,:) = A(jj,:) - fac * A(ii,:);
    end
end

end
```

(a) As written, the code transforms `A` into an upper triangular matrix (`U` in the LU-factorization), but does not generate the lower triangular matrix in the LU-factorization, `L`. Modify the function so that it also gives `L` as output. To make it so that your function produces a second output, `L`, you need to change the function declaration from

`function [A] = GE_NMM(A)`

to

`function [L,A] = GE_NMM(A)`

At some point in the body of your function you must define the variable `L` so that the function has something to output for `L`.

Here is an example of the implementation of a function named `stat` which returns the mean and standard deviation of an input vector `x`:

```
function [m,stdev] = stat(x)
    n = length(x);
    m = sum(x) / n;
    stdev = sqrt(sum(((x - m).^2) / n));
end
```

Note that you can test your code for this part by constructing a random square matrix, feeding it into your function, then verifying that the product of the two matrices your function returns gives the original matrix, e.g. by checking that `norm(A - L * U)` is very small.

(b) Notice that in performing Gaussian elimination the function `GE_NMM` subtracts entire rows of `A` from one another. However, at the $k$th step of elimination, all the entries below the diagonal in columns 1 through $k-1$ have been zeroed out already. Therefore we making Matlab use unnecesary floating point operations to add and subtract multiples of 0 from one another. Modify the code so that it avoids these extraneous operations.

(c) Modify the function `GE_NMM` so that it gives as a *third* output the exact number of floating point operations the method needed to use. For example, if one wanted to make a function that output the 2-norm of a vector and the number of floating point operations required to do so, one implementation would be the following

```
function [op_count, nrm] = two_norm(x)
    % Length of vector x
    n = length(x);

    % Variable in which to track flops
    op_count = 0;

    % Variable where the norm will be stored
    nrm = 0;

    % Loop over entries in x
    for i = 1:n
        % Add contribution to norm
        nrm = nrm + x(i)^2;

        % One addition and one multiplication
        op_count = op_count + 2;
    end

    % Take square root to get the 2-norm
    nrm = sqrt(nrm);

    % We need to subtract one from op_count because the first addition was
    % unnecessary (since it was 0 + x(1)^2)
    op_count = op_count - 1;
end
```

You can check that your code is correct by comparing the number of floating point operations it counts against the number we derived in class.

(d) Paste the following code in your script to generate the matrix `A`:

```
rng(10,'twister');
A = rand(50,50);
```

Use your modified version of `GE_NMM` to compute the LU factorization of `A` and to count the number of floating point operations required. Store the 50th **columns** of `L` and `U` in `A5.dat` and `A6.dat` respectively. Save the number of floating point operations used in `A7.dat`.

**Be sure to submit both your .m files to Scorelator and to highlight the one that generates the dat files!**