Assignment #4

This assignment is due on April 16th one hour before class via email to mailto:wallraven@korea.ac.kr. If you are done with the assignment, make one zip-file of the `assignment4` directory and call this `<LASTNAME_FIRSTNAME_A4.zip>` (e.g.: `HONG_GILDONG_A4.zip`).

Please make sure to comment the code, so that I can understand what it does. Uncommented code will reduce your points!

ALSO: I can also surf on the internet for code. Downloading and copying and pasting other peoples' code is plagiarism and will NOT be tolerated. If you work as a team, the code needs to contain all team members' names!!

Part1 (55 points):

In this part you will implement a function `GaussSolve` that uses the Gaussian Elimination technique **with pivoting** as shown during class. Please take a look at the slides I uploaded. For more information about partial pivoting, please take a look at **the LU Decomposition Chapter in Clive Moler's Numerical Computing with Matlab book**!!!

a) Implement a Matlab function `GaussSolve` that solves a linear system of equations Ax=b for a square matrix A using forward elimination, **partial pivoting**, and backward substitution as shown during class. The function should be defined as

```
function [x,det] = GaussSolve(A,b)
```

and should include **error handling to**

- **check whether the matrix A is square or not**
- **whether the dimensions of b and A fit**
- **whether during the forward elimination step any of the leading coefficients become 0**

If any conditions become critical, then the function should abort, telling the user the reason for it.

Note, that the function should also return the determinant of the matrix A in the variable `det.` We saw in class that the determinant in general is a polynomial. In a lot of cases, calculating the determinant using the complex polynomial form is actually not feasible. A much more efficient way is to use the fact that the determinant of an upper triangular matrix U:

$U = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & u_{n,n} \end{pmatrix}$ is given by $\det(U) = \prod_{i=1}^{n} u_{i,i}$, that is, simply by the product of the diagonal elements of U.

Now, observe that in `GaussSolve`, we actually have created such a matrix U as the last step during forward elimination. Use this information to also return the variable `det`.

b) Which part of the code takes longest to run? Insert your observations as a comment into `GaussSolve`.

c) Use the built-in commands `tic` and `toc` to time how fast your algorithm is for **random** matrices and vectors for the following sizes: n=1,5,10,100,1000.

In order to do this, write a script `TestGaussSolve.m` that runs `GaussSolve` and compares it with the built in Matlab backslash command for **random matrices and vectors (use the matlab function `rand(n)` for this)** for the given problem sizes, records the execution time for each solution method and plots the results in a nice plot.

How much faster is the built-in Matlab function than your function for each time step? Why do you think it is faster?

Insert the answer and your observations as a comment into `TestGaussSolve.m`.

Part2 (5 points):

Use your code to solve the simple truss problem from class to determine the six unknowns.