Assignment #3

This assignment is due on April 19th one hour before class via email to mailto:christian.wallraven+AMS2016@gmail.com. If you are done with the assignment, make one zip-file of the `assignment3` directory and call this `<LASTNAME_FIRSTNAME_A3.zip>` (e.g.: `HONG_GILDONG_A3.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 (50 points):

In this part you will implement a function `GaussSolve` that uses the Gaussian Elimination technique **with pivoting** as shown during class in the **powerpoint material. Please take a look at the slides I uploaded. Please use exactly this algorithm!!!**

a) Implement a Matlab function `GaussSolve` that solves a linear system of equations Ax=b for a square matrix A using forward elimination, backward substitution, and partial pivoting **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 it does, it should break with an error**

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

In a lot of cases, calculating the determinant using the complex polynomial form is actually not feasible. A more efficient way is to use the fact that the determinant of an upper triangular matrix U:

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

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 calculate the second return variable `det`.

b) Which part of the code (forward elimination or backward substitution) takes longest to run? Insert your observations as a comment into `GaussSolve`. You can either use the built-in Matlab commands `tic` and `toc` to measure this time, or you can think about how the two parts are structured. I would prefer if you did the latter ☺

c) Now, please use the Matlab commands `tic` and `toc` to time how fast the whole GuassSolve script is for **random** matrices and vectors for the following sizes: n=1,5,10,100,1000.

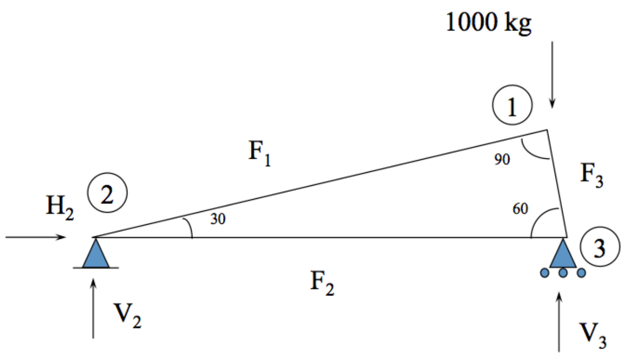
In order to do this, write a script `TestGaussSolve.m` that runs `GaussSolve` and compares it with the **built-in Matlab backslash** command (\) for **random matrices and vectors (use the matlab function `rand(n)` for this)** for the given problem sizes, records the execution time for each solution method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

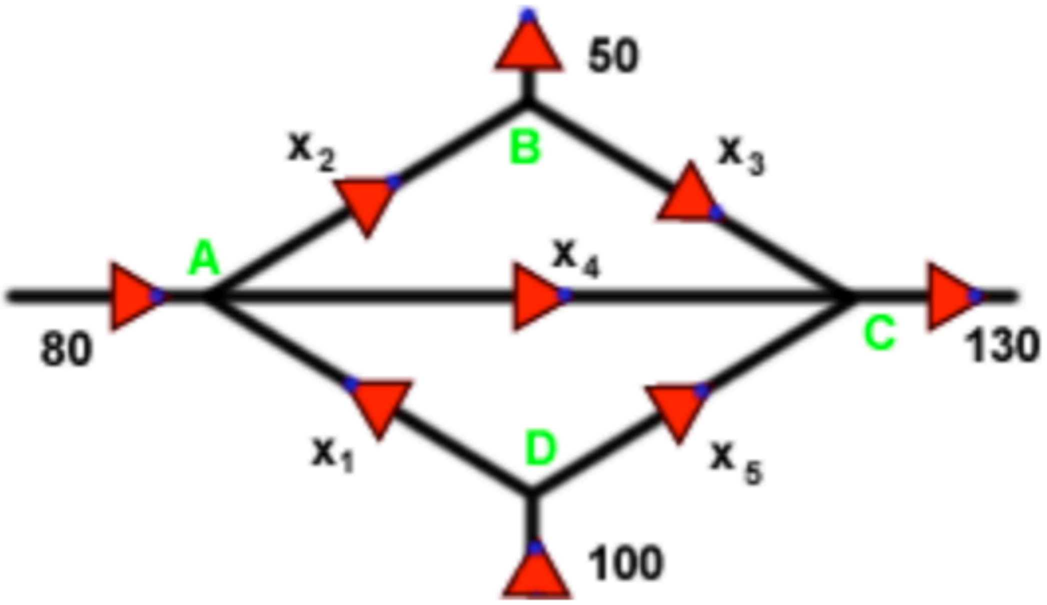
Part2 (10 points):

a) Use your code to solve the simple truss problem from class to determine the six unknown forces F1,F2,F3,H2,V2,V3. See the class material for the layout of the problem and for the matrix. You can use "kg" as the force unit (yes, you are officially allowed, but only for this assignment ☺).



b) Use your code to solve a traffic flow problem. In the picture below, you see a cutout of a city with different streets and junctions. Since you have traffic cameras positioned at the exits of the block, you can measure how many cars enter and exit the block. These numbers are given next to the arrows. Your task is to find out the missing numbers x1, x2, x3, x4, x5 around the middle block using a system of linear equations (Hint: to solve this, you obviously need five equations – you can get those by looking at the traffic numbers at intersections!

Assume that x1-x5 are positive numbers and that traffic flows in the direction of the arrows!).


50
B
x_2
x_3
A
x_4
80
C
130
x_1
D
x_5
100