Course number: 02635, **August Semrau Andersen, s183918**
Date: October 21, 2020 (Coded with Karl Ulbæk, s183931)

Mathematical Software Programming: Assignment 2

Part I: Solving a system of equations with LAPACK

Function/program structure: how did you organize your code?
The function **call_dgesv** consists of three parts:
First it checks that inputs are not null-pointers, that the matrix A is a square matrix and that the dimensions of A and vector b fit. If not, it returns -9, -10 and -11 respectively.
Next it defines the inputs going into the LAPACK function **dgesv_**, which include explicitly defining the number of columns in b **nrhs** (which is one), creating an empty integer **info** which will hold the return/exit status of **dgesv_** and allocating an empty array of same dimension as b which will hold pivoting information for **dgesv_** (it returns -12 in case of failed allocation).
As matrix A is row-wise, but LAPACK needs it to be column-wise, the **call_dgesv** creates a transposed version **A_t**.
Finally, it calls the dgesv_ function. In case of an error inside **dgesv_**, this is outputted using **info**, which is also the return value of **call_dgesv**.

Did you consider any numerical aspects in your implementation?
No. **call_dgesv** deals solely with running **dgesv_**, not solving the system, and therefor only checks whether the inputs are correct, but not whether the solving will be successful, which **dgesv_** will tell us by the output value **info**.

How did you test your code? What tests did you perform to ensure correctness?
We wrote a script, **call_dgesv_test.c**, which allocates a matrix and vector using the functions **malloc_matrix** and **malloc_vector** given in the **matrix_io.c** script and inputs these into **call_dgesv**. We've checked that the resulting b-vectors is correct according to our own calculations, and that it returns -9, -10 and 12 according to the errors mentioned.

Part II: Command-line tool

Function/program structure: how did you organize your code?
First of all, the function **call_dgesv** from part 1 is copied to the script **solve.c** as the function doesn't exist in Codejudge and would therefor fail to compile.
The main function does the following:
Using the functions **read_matrix** and **read_vector**, the given matrix and vector from their respective .txt files, specified in the command line, are converted into structures usable by **dgesv_**. If this fails, the function prints an appropriate error message to **stderr**.
Ideally, we would now run the **call_dgesv** function from part 1 and save the return value of that function (**info**) for returning appropriate error messages, but as the error return values -9, -10, -11 and -12 not comming from **dgesv_** potentially overlap with **info** from **dgesv_**, we first need to check the inputs. If errors are found for the inputs, as described in part 1, appropriate error messages are printed to **stderr**.
The function then runs the **call_dgesv** function from part 1 ans saves the return **info** value. If **info** is non-zero, an error message is printed (same way as before), else the solution now held in vector b is written to the .txt filename given as the final input in the command line using the **write_vector** function from **matrix_io.c**.

Did you consider any numerical aspects in your implementation?
No, same story as in part 1.

How did you test your code? What tests did you perform to ensure correctness?
We made some .txt files using the given Matlab command containing matrices and vectors, and similarly to part 1 tested for our own solved systems, errors related to the input as well as errors related to typing wrong filenames etc.