02635 Fall 2016 — Assignment II

This is the last of the two assignments in this course. Like in the first assignment, you will write a short program and a short report. The purpose of this assignment is to train and assess your ability to:

- Design, implement, and document a program that solves a mathematical problem.
- Debug and test mathematical software.
- Call external (third party) programs and libraries.

Please note that this assignment will count towards your final grade (approximately 10% of final grade).

Practical information

You may work on this assignment alone or in small groups of 2-3 students, but you must hand in an **individual report** regardless of whether you choose to work alone or as part of a small group. You may hand in your group's code if the code was written collaboratively by you and your group.

Deadline

Your **report** and **code** must be submitted via CampusNet no later than on **November 23, 2016**.

Report

Your report should be short and concise and **no longer than one A4 page**. If necessary, you may include a one-page appendix with code snippets. Your report should also include either a front page or a header/footer with the following information:

- course number and date
- name and student number
- group members (if applicable).

You must submit your report as a **PDF file**.

Code

You must submit your code as a single **zip file** that contains all **source files and a makefile**. You are encouraged to test your code and your makefile on the DTU Unix system (e.g. via ThinLinc) to make sure that there are no compilation errors.

Your code will be evaluated based on:

- program structure
- readability (use comments in the code, indentation, etc.)
- correctness, error checking, and tests.

Questions

Please post any questions that you may have about the assignment on Piazza.

Background

Systems of linear equations are ubiquitous in engineering an science, and the ability to programmatically solve such systems is regarded as one of the most fundamental problems of linear algebra. Given a square matrix $A \in \mathbb{R}^{n \times n}$ (i.e., a matrix with n rows and n columns) and a vector $b \in \mathbb{R}^n$, we seek a vector $x \in \mathbb{R}^n$ such that $Ax = b$. If such a vector exists, we say that the system of equations $Ax = b$ is consistent, and otherwise we say that the system is inconsistent. From linear algebra we know that if the matrix A is

nonsingular, then the system $Ax = b$ is consistent for all possible right-hand sides b. In this case we can solve for x using Gaussian elimination.

Implementing Gaussian elimination from scratch is tedious, so instead of implementing our own Gaussian elimination routine, we will use an external library called LAPACK (Linear Algebra PACKage) to solve the system of equations. LAPACK is originally written in FORTRAN, but a compiled LAPACK library can still be used in C. We will use a routine called **dgesv** which has the following C prototype:

```
/* C prototype for LAPACK routine DGESV */
void dgesv_(const int *n,    /* columns/rows in A          */
            const int *nrhs, /* number of right-hand sides */
            double *A,       /* array A                    */
            const int *lda,  /* leading dimension of A     */
            const int *ipiv, /* pivoting array             */
            double *B,       /* array B                    */
            const int *ldb,  /* leading dimension of B     */
            int *info        /* status code                */
            );
```

Remarks: Notice that all inputs are pointers. This is because FORTRAN uses *call-by-reference* to pass function arguments (as opposed to *call-by-value*). Notice also the underscore after the function name `dgesv`. This is added by some FORTRAN compilers, and it is (typically) required when using a routine from the LAPACK library in a C program. Also, it is important to note that the LAPACK library assumes column-major ordering of the elements in a matrix. The **dgesv** routine is documented on the LAPACK website. Note that the routine needs an integer array of length n as "workspace" for storing pivoting information (the input `ipiv`). Moreover, the argument `info` is used to return a status: a nonzero value indicates that the routine did not successfully solve the system of equations.

Assignment

Code

Write a program that

1. reads a matrix A and a vector b from two text files,
2. solves the system of equations $Ax = b$ (if possible) using the `dgesv` routine from the LAPACK library,
3. writes the solution $x^\star$ to a file.

The user should be able to specify the data files (the matrix A and the vector b) using the command-line when calling to program (as opposed to user input after starting the program), e.g.,

```
$ ./solve A.txt b.txt
```

Moreover, your program should work for problems of different sizes.

The `source` directory included in the ZIP file contains a template `solve.c` and makefile templates for Windows, Mac, and the DTU Unix system. It also includes a precompiled version of the BLAS and LAPACK libraries for Windows (`win_x86_64/libopenblas.a`); it is perfectly fine to delete this if you are not a Windows user). The source directory also includes some auxiliary routines (defined in `linalg.h` and implemented in `linalg.c`) that you may use to simplify your program (e.g., you can use `read_vector()` and `read_matrix()` to read the problem data A and b, and `write_vector()` can be used to write the solution x to a file).

You may use the following test case (included with the assignment) to test your program:

- `A.txt` (matrix A, $n = 3$)

```
-0.425105  -2.18768   0.939106
-0.642796  -0.795089  1.01099
-0.325637  -0.395732  0.207149
```

- `b.txt` (vector b, $n = 3$)

```
-1.98314
 0.799997
-0.495653
```

It is easy to verify that the test case solution to $Ax = b$ is (approximately) $x = (1, 2, 3)$.

Report

Write a short report (see requirements under "Practical information") in which you briefly address the following questions:

1. Program structure: how did you organize your code?
2. Briefly explain how your program works: what does the user need to know in order to use your program? Are there any limitations?
3. How did you test your program to ensure that it is correct? For example, what happens if the user specifies two data files with incompatible dimensions (e.g., if A is of order n and b is a vector of length $m \neq n$)? What happens if `dgesv` fails (`info` is nonzero)? What happens if the files do not exist?
4. Time complexity: measure the CPU time for problems with $n \in \{8, 16, 32, 64, 128, 256\}$. How does the CPU time scale as a function of n?