

Project 1, deadline September 9

Computational Physics I FYS3150/FYS4150

Department of Physics, University of Oslo, Norway

Aug 25, 2020

Introduction

The aim of this project is to get familiar with various vector and matrix operations, from dynamic memory allocation to the usage of programs in the library package of the course. For Fortran users memory handling and most matrix and vector operations are included in the ANSI standard of Fortran 90/95. Array handling in Python is also rather trivial. For C++ user however, there are several possible options. Two are listed here.

- For this exercise we recommend that you make your own functions for dynamic memory allocation of a vector and a matrix. You don't need to write a class for this operations. Use then the library package `lib.cpp` with its header file `lib.hpp` for obtaining LU-decomposed matrices, solve linear equations etc.
- A very good and often recommended library for C++ handling of arrays is the library Armadillo, to be found at arma.sourceforge.net. We will discuss the usage of this library during the lab sessions and lectures. Armadillo has also an interface to Lapack functions for solving systems of linear equations. Alternatively you can use the [Eigen](http://eigen.tuxfamily.org) library, which has much of the same functionality as Armadillo.

Your program, whether it is written in C++, Python, Fortran or other languages, should include dynamic memory handling of matrices and vectors.

The material needed for this project is covered by chapter 6 of the lecture notes, in particular section 6.4 and subsequent sections.

Many important differential equations in Science can be written as linear second-order differential equations

$$\frac{d^2 y}{dx^2} + k^2(x)y = f(x),$$

where f is normally called the inhomogeneous term and k^2 is a real function.

A classical equation from electromagnetism is Poisson's equation. The electrostatic potential Φ is generated by a localized charge distribution $\rho(\mathbf{r})$. In three dimensions it reads

$$\nabla^2 \Phi = -4\pi\rho(\mathbf{r}).$$

With a spherically symmetric Φ and $\rho(\mathbf{r})$ the equations simplifies to a one-dimensional equation in r , namely

$$\frac{1}{r^2} \frac{d}{dr} \left(r^2 \frac{d\Phi}{dr} \right) = -4\pi\rho(r),$$

which can be rewritten via a substitution $\Phi(r) = \phi(r)/r$ as

$$\frac{d^2\phi}{dr^2} = -4\pi r\rho(r).$$

The inhomogeneous term f or source term is given by the charge distribution ρ multiplied by r and the constant -4π .

We will rewrite this equation by letting $\phi \rightarrow u$ and $r \rightarrow x$. The general one-dimensional Poisson equation reads then

$$-u''(x) = f(x).$$

Project 1 a): We have

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i \quad \text{for } i = 1, \dots, n,$$

$$-v_{i+1} - v_{i-1} + 2v_i = h^2 f_i$$

or, on vector form,

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} v_{i-1} \\ v_i \\ v_{i+1} \end{bmatrix} = h^2 f_i$$

Expanding this expression to all i gives

$$\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & 0 \\ & & -1 & 2 & -1 & \\ & 0 & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_{i-1} \\ v_i \\ v_{i+1} \\ \vdots \\ v_n \end{bmatrix} = h^2 \begin{bmatrix} f_1 \\ \vdots \\ f_{i-1} \\ f_i \\ f_{i+1} \\ \vdots \\ f_n \end{bmatrix}$$

Note that the terms 0 and $n+1$ are not included because of the boundary conditions $v_0 = v_{n+1} = 0$.

Project 1 b): The first goal is to create an algorithm which can solve equations for any tridiagonal matrix.

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & & & \\ a_1 & b_2 & c_2 & & 0 \\ & a_2 & b_3 & c_3 & \\ & & \ddots & \ddots & \ddots \\ 0 & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{bmatrix}.$$

We use forward substitution to eliminate the a -diagonal, then backward substitution to eliminate the c -diagonal. The algorithm to implement is then [?]

Forward substitution:

for i from 1 to $n - 1$ **do**

$$\tilde{b}_i = b_i - \frac{c_{i-1}a_{i-1}}{\tilde{b}_{i-1}}$$

$$\tilde{g}_i = g_i - \frac{\tilde{g}_{i-1}a_{i-1}}{\tilde{b}_{i-1}}$$

end for

Backward substitution:

for i from n to 1 **do**

$$u_i = \frac{\tilde{g}_i - c_i u_{i+1}}{\tilde{b}_i}$$

end for

When programming this, all indices start at 0, so

Project 1 c): Next, we look at the special case where the matrix has identical matrix elements along the diagonal and identical (but different) values for the nondiagonal elements. In other words

$$a_i = a, \quad b_i = b, \quad c_i = c \quad \text{for all } i.$$

Project 1 d): Compute the relative error in the data set $i = 1, \dots, n$, by setting up

$$\epsilon_i = \log_{10} \left(\left| \frac{v_i - u_i}{u_i} \right| \right),$$

as function of $\log_{10}(h)$ for the function values u_i and v_i . For each step length extract the max value of the relative error. Try to increase n to $n = 10^7$. Make a table of the results and comment your results. You can use either the algorithm from b) or c).

Project 1 e): Compare your results with those from the LU decomposition codes for the matrix of sizes 10×10 , 100×100 and 1000×1000 . Here you should use the library functions provided on the webpage of the course. Alternatively, if you use armadillo as a library, you can use the similar function for LU

decomposition. The armadillo function for the LU decomposition is called *LU* while the function for solving linear sets of equations is called *solve*. Use for example the unix function *time* when you run your codes and compare the time usage between LU decomposition and your tridiagonal solver. Alternatively, you can use the functions in C++, Fortran or Python that measure the time used.

Make a table of the results and comment the differences in execution time. How many floating point operations does the LU decomposition use to solve the set of linear equations? Can you run the standard LU decomposition for a matrix of the size $10^5 \times 10^5$? Comment your results.

To compute the elapsed time in c++ you can use the following statements

```
...
#include "time.h"    // you have to include the time.h header
int main()
{
    // declarations of variables
    ...
    clock_t start, finish; // declare start and final time
    start = clock();
    // your code is here, do something and then get final time
    finish = clock();
    ( (finish - start)/CLOCKS_PER_SEC );
    ...
}
```

Similarly, in Fortran, this simple example shows how to compute the elapsed time.

```
PROGRAM time
  REAL :: etime          ! Declare the type of etime()
  REAL :: elapsed(2)     ! For receiving user and system time
  REAL :: total          ! For receiving total time
  INTEGER :: i, j

  WRITE(*,*) 'Start'

  DO i = 1, 5000000
    j = j + 1
  ENDDO

  total = ETIME(elapsed)
  WRITE(*,*) 'End: total=', total, ' user=', elapsed(1), &
    ' system=', elapsed(2)

END PROGRAM time
```

Your results may depend on the granularity of the clock.

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.
- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.
- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliability and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.
- Try to give an interpretation of your results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008 or Python. The following prescription should be followed when preparing the report:

- Use **Canvas** to hand in your projects, log in at <https://www.uio.no/english/services/it/education/canvas/> with your normal UiO username and password.

- Upload **only** the report file! For the source code file(s) you have developed please provide us with your link to your github domain. The report file should include all of your discussions and a list of the codes you have developed. Do not include library files which are available at the course homepage, unless you have made specific changes to them. Alternatively, you can just upload the address to your GitHub or GitLab repository.
- In your git repository, please include a folder which contains selected results. These can be in the form of output from your code for a selected set of runs and input parameters.
- In this and all later projects, you should include tests (for example unit tests) of your code(s).
- Comments from us on your projects, approval or not, corrections to be made etc can be found under your **Canvas** domain and are only visible to you and the teachers of the course.

Finally, we encourage you to work two and two together. Optimal working groups consist of 2-3 students. You can then hand in a common report.