Project 1, deadline September 13

FYS3150/FYS4150 - Hannah Berg, Stian Bilek & Frida Furmyr

September 12, 2017

Introduction

In science, we often have to solve linear set of equations numerically. This can be done easily when the set of equations are small. However, there is a limit to how many sets can be solved numerically using basic linear algebra, as computers have a limited memory. This forces us to solve things smarter, so that we will not run out of memory.

In this project, we were to solve the one-dimensional Poisson equation with Dirichlet boundary conditions by rewriting it as a set of linear equations. The general form of the one-dimensional Poisson is given as

$$-u''(x) = f(x). \tag{1}$$

The accuracy of the solution depends on how many sets of equations we rewrite equation **??** to. First, we rewrite equation **??** to

$$-u''(x) = f(x), \qquad x \in (0,1), \qquad u(0) = u(1) = 0.$$

and we define the discretized approximation to u as v_i with grid points $x_i = ih$ in the interval from $x_0 = 0$ to $x_{n+1} = 1$. The step length or spacing is defined as $h = 1/(n+1)$. We have then the boundary conditions $v_0 = v_{n+1} = 0$. We approximate the second derivative of u with

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

where $f_i = f(x_i)$.

Show that you can rewrite this equation as a linear set of equations of the form

$$\mathbf{Av} = \mathbf{d},$$

where $\mathbf{A}$ is an $n \times n$ tridiagonal matrix which we rewrite as

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix},$$

and $d_i = h^2 f_i$.

In our case we will assume that the source term is $f(x) = 100e^{-10x}$, and keep the same interval and boundary conditions. Then the above differential equation has a closed-form solution given by $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$. We will compare our numerical solution with this result in the next exercise.

We can rewrite our matrix $\mathbf{A}$ in terms of one-dimensional vectors a, b, c of length $1 : n$. Our linear equation reads

$$\mathbf{A} = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots \\ a_1 & b_2 & c_2 & \dots & \dots & \dots \\ & a_2 & b_3 & c_3 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_{n-1} & b_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \dots \\ \dots \\ \dots \\ d_n \end{bmatrix}.$$

Note well that we do not include the endpoints since the boundary conditions are used resulting in a fixed value for v_i. A tridiagonal matrix is a special form of banded matrix where all the elements are zero except for those on and immediately above and below the leading diagonal. Develop a general algorithm first which does not assume that we have a matrix with the same elements along the diagonal and the non-diagonal elements. The algorithm for solving this set of equations is rather simple and requires two steps only, a decomposition and forward substitution and finally a backward substitution.

The general algorithm for solving this linear equations, when we assume that $a_{n-1} \neq c_{n-1}$ is possible to find by first row reducing the matrix $\mathbf{A}$. The echelon form of the matrix $[\mathbf{A}d]$ is

$$\begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & \dots & d_1 \\ a_1 & b_2 & c_2 & \dots & \dots & \dots & d_2 \\ & a_2 & b_3 & c_3 & \dots & \dots & d_3 \\ & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} & d_{n-1} \\ & & & & a_{n-1} & b_n & d_n \end{bmatrix} \sim \begin{bmatrix} \tilde{b}_1 & 0 & 0 & \dots & \dots & \dots & \tilde{d}_1 \\ 0 & \tilde{b}_2 & 0 & \dots & \dots & \dots & \tilde{d}_2 \\ & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & 0 & \tilde{b}_{n-1} & 0 & \tilde{d}_{n-1} \\ & & & & 0 & \tilde{b}_n & \tilde{d}_n \end{bmatrix}$$

Here we have redefined the pivot columns so that

$$\tilde{b_1} = b_1 \tilde{b_{i+1}} = b_{i+1} \tag{2}$$

$$\tag{3}$$

Your first task is to set up the general algorithm (assuming different values for the matrix elements) for solving this set of linear equations. Find also the precise number of floating point operations needed to solve the above equations. For the general algorithm you need to specify the values of the array elements a, b and c by inserting their explicit values.

Then you should code the above algorithm and solve the problem for matrices of the size 10×10, 100×100 and 1000×1000. That means that you select $n = 10$, $n = 100$ and $n = 1000$ grid points.

Compare your results (make plots) with the closed-form solution for the different number of grid points in the interval $x \in (0, 1)$. The different number of grid points corresponds to different step lengths h.

Project 1 c): Use thereafter the fact that the matrix has identical matrix elements along the diagonal and identical (but different) values for the non-diagonal elements. Specialize your algorithm to the special case and find the number of floating point operations for this specific tri-diagonal matrix. Compare the CPU time with the general algorithm from the previous point for matrices up to $n = 10^6$ grid points.
We compare the different CPU times for different number of grid points in table **??**.

Table 1: CPU times for n grid points

Grid points n	CPU time $[s]$
10	
10^2	
10^3	
10^4	
10^5	
10^6	

Project 1 d): Compute the relative error in the data set $i = 1, \ldots, n$,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 d): 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.

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 reliabilty 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 you 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 Devilry to hand in your projects, log in at http://devilry.ifi.uio.no with your normal UiO username and password and choose either 'fys3150' or 'fys4150'. There you can load up the files within the deadline.
- 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.
- 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 Devilry 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.