FYS3150 - Project 1
Resolution of second order differential equation

Guillermo Serrera Pardueles

September 10, 2018

Abstract

I have solved numerically the one-dimensional Poisson equation with Dirichlet boundary conditions, discretizing the problem as a set of linear equations and solving that set using three different algorithms: a general Gaussian elimination method for a tridiagonal matrix, a more specific Gaussian elimination method for a particular matrix, and a standard LU decomposition method. The obtained results are satisfactory and accurate, especially for the particular Gaussian elimination algorithm.

1 Introduction

The Poisson equation is a classical electromagnetism, made to describe the electrostatic potential Φ generated by a localized charge distribution $\rho(r)$. In a three dimensional space it reads:

$$\nabla^2\Phi = -4\pi\rho(r) \tag{1}$$

Where ∇^2 is the Laplace operator. If Φ and $\rho(r)$ are spherically symmetrical then this equation can be expressed in one dimension: r, which leads to the general one-dimensional Poisson equation, which is the one we will look at in this project. The general expression for this general one-dimensional equation is:

$$u''(x) = -f(x) \tag{2}$$

Moreover, in this project the equation that has been solved is the Poisson one-dimensional equation with Dirichlet boundary conditions:

$$u''(x) = -f(x),\ x \in (0,1),\ u(0) = u(1) = 0 \tag{3}$$

Combining the Taylor approximations for $f(x+h)$ and $f(x-h)$, the second derivative of the function u can be approximated with the following expression:

$$-u''(x) = f(x) = -\frac{u(x+h) + u(x-h) - 2u}{h^2} \tag{4}$$

Thus, the function u can be now discretized as a set of points v_i with grid points $x_i = ih$ in the interval $x_0 = 0$ to $x_{n+1} = 1$. The step length between each grid point is defined by $h = 1/(n+1)$, where $n+1$ is the number of grid points. The boundary conditions can be expressed as $v_0 = v_{n+1} = 0$. Now the approximation for $f(x)$ is given by the following expression:

$$f_i = f(x_i) = -\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} \text{ for } i = 1, 2..., n \tag{5}$$

The problem is now discretized and can be expressed as a set of linear equations of the form:

$$\boldsymbol{Av} = \boldsymbol{b} \tag{6}$$

Where A is an $n \times n$ matrix which can be written as:

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 & ... & ... & 0 \\ -1 & 2 & -1 & 0 & ... & ... \\ 0 & -1 & 2 & -1 & ... & ... \\ ... & ... & ... & ... & ... & ... \\ ... & ... & ... & -1 & 2 & -1 \\ 0 & ... & ... & ... & -1 & 2 \end{bmatrix}$$

and $\boldsymbol{b} = h^2 f$.

For this specific project we will assume that $f(x) = 100e^{-10x}$. Then the problem has an analytical solution $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$, which we will compare to the numerical results that we will obtain.

2 Methods

In order to solve this set of equations, one can apply several methods. For this specific matrix $\boldsymbol{A}$, a tridiagonal matrix; three methods have been used.

The first method is a specific Gaussian elimination, which consists in substituting the leading diagonal and the $\boldsymbol{b}$ vector elements so that the below diagonal is equal to 0 (forward substitution); and then doing a backward substitution in order to get the final solution. Equation (7) contains the expressions for the forward substitution, while equation (8) contains the expression for the backward substitution.

$$\tilde{d}_i = d_i - \frac{a_i c_{i-1}}{\tilde{d}_{i-1}} \text{ , } \tilde{b}_i = b_i + \frac{a_i \tilde{b}_{i-1}}{\tilde{d}_{i-1}} \tag{7}$$

$$v_{i-1} = \frac{\tilde{b}_{i-1} - c_{i-1} v_i}{\tilde{d}_{i-1}} \tag{8}$$

where a_i are the elements below the diagonal, b_i are the diagonal elements and c_i are the elements above the diagonal. Note that $a_1 = c_n = 0$ and the endpoints of $\boldsymbol{v}$ are already known due to the boundary conditions. This method takes 8 floating point operations per iteration: 5 for the forward substitution and 3 for the backward substitution, giving a total number of $8n$ floating point operations.

However, this algorithm can be specialized for the specific matrix in this project, as this matrix has identical non-diagonal elements ($a_i = c_i$). Therefore, the substitution expressions can be rewritten as:

$$\tilde{d}_i = \frac{i+1}{i} \;,\; \tilde{b}_i = b_i + \frac{(i-1)\tilde{b}_{i-1}}{i} \tag{9}$$

$$v_{i-1} = \frac{i-1}{i}(\tilde{b}_{i-1} + v_i) \tag{10}$$

This method takes 4 floating point operations per iteration, as the elements of the diagonal can be now precalculated, giving a total number of $4n$ floating point operations.

Finally, other method for solving sets of linear equations is LU decomposition, which consists of decomposing the given matrix into a product of two matrices: a lower triangular matrix $\boldsymbol{L}$ and an upper triangular matrix $\boldsymbol{U}$, and then it transforms the problem following the pattern described in equation (11):

$$\boldsymbol{Av} = \boldsymbol{b} \rightarrow \boldsymbol{LUv} = \boldsymbol{b} \rightarrow \boldsymbol{Uv} = \boldsymbol{L}^{-1}\boldsymbol{b} = \boldsymbol{y} \rightarrow \boldsymbol{Uv} = \boldsymbol{y} \tag{11}$$

As matrix $\boldsymbol{U}$ is an upper triangular matrix, the solution is trivial. This is the most common algorithm for solving general linear sets of equations, and it gives a total number of $O(\frac{2}{3}n^3)$ floating point operations.

3 Implementation

For all programs and benchmark calculations see:
`https://github.com/gserrera/FYS3150`

4 Results and analysis

The results given by the various algorithms used in this project proved to be accurate to the actual analytical solution of the problem. In Figures 1, 2 and 3 the plots made to compare the numerical approximations with the analytical solution can be consulted.

In Table 1 the results of the CPU time and maximum relative error logarithm for each step length and algorithm are shown. One can see that the optimal step number for the general tridiagonal matrix Gaussian elimination method is 10^5, while for the specific case is 10^6. There is no significant time difference between these methods. However, the LU decomposition method proved to be inefficient for more than 10^4 grid points, as the computer's memory wasn't enough for this algorithm and the computer froze.

5 Discussion and conclusions

The results obtained with any of the shown algorithms are satisfactory and accurate. However, the specific algorithm gives a more accurate result, especially if it is compared to the LU decomposition method, which is useful for general matrices but less efficient for specific cases. This shows that a little thinking can make a big difference in performance.

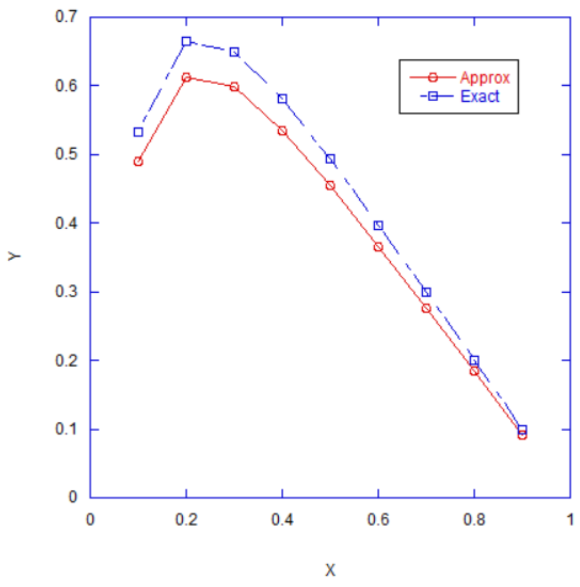

Figure 1: Plot showing the numerical approximation and the analytical solution for 10 grid points.

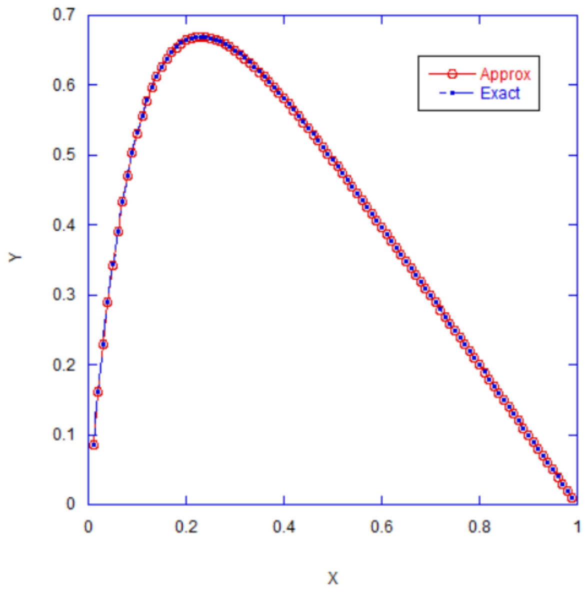

Figure 2: Plot showing the numerical approximation and the analytical solution for 100 grid points.

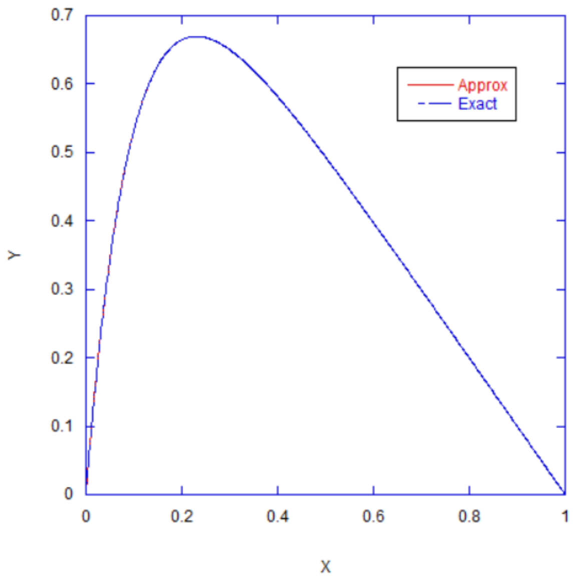

Figure 3: Plot showing the numerical approximation and the analytical solution for 1000 grid points.

	General tridiagonal matrix		Specific tridiagonal matrix		LU decomposition	
n	t/s	ε	t/s	ε	t/s	ε
10	0	-1.1005822	0	-1.1005822	0	-1.1005822
10^2	0	-3.0793984	0	-3.0793984	0	-3.0793984
10^3	0	-5.0791834	0	-5.0791834	0	-5.0791834
10^4	0	-7.0791983	0	-7.0791795	0	-7.0767951
10^5	0	-8.8432191	0	-9.0790004	-	-
10^6	0	-6.0755133	0	-10.175691	-	-
10^7	0	-5.5252302	0	-9.6534015	-	-

Table 1: Results obtained from the different methods for the values of the elapsed CPU time t and the logarithm of the relative error ε, depending on the size of the matrix n. Elapsed CPU time includes only the numerical algorithm, not the data writing.

References

[1] Hjort-Jensen, M. *Computational physics.* accesible at course github repository. 551 pages. 2015

[2] Conrad Sanderson and Ryan Curtin. *Armadillo: a template-based C++ library for linear algebra.* Journal of Open Source Software, Vol. 1, pp. 26, 2016.

[3] Conrad Sanderson and Ryan Curtin. *A User-Friendly Hybrid Sparse Matrix Class in C++.* Lecture Notes in Computer Science (LNCS), Vol. 10931, pp. 422-430, 2018.