Project 1

Michael Saybolt
Bjon Charlery

March 9, 2017

Abstract

This project explored solving differential equations, specifically linear second order equations, by using a couple of different methods which include LU Decomposition, Gauss Forward and Backward Substitution : General algorithm

1 Introduction

The objective of this project was to solve a differential equation in the form of:

$$-u^{"}(x) = f(x), x\epsilon(0,1), u(0) = u(1) = 0.$$

that can be written into the linear equation $(Ax = b)$. Since A is a tridiagonal matrix there is an actual or analytical solution that can be made for this particular matrix. The following sections will describe the origins for the algorithms used and their respective results.

2 Description

This section will detail the methods used to solve the linear equation system. It is solved using two methods:

- LU decomposition
- Tridiagonal Solver - Gauss Elimination

2.1 LU Decomposition Solver

This method consists of creating a Lower-Triangular matrix L and a Upper-Triangular matrix U for factorizing the linear system $(A = LU)$.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 & 0 \\ l_{41} & l_{42} & l_{43} & 1 & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & 1 \end{bmatrix} \quad and \quad \begin{bmatrix} 1 & l_{12} & l_{13} & l_{14} & l_{15} \\ 0 & 1 & l_{23} & l_{24} & l_{25} \\ 0 & 0 & 1 & l_{34} & l_{35} \\ 0 & 0 & 0 & 1 & l_{45} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Once (L) and (U) are found they can be used to solve the following equation: ($Av = LUv = h^2 f$) by allowing ($Ly = h^2 f$) and ($Uv = y$) with backwards substitution, in this case that would be required two times.

2.2 Tridiagonal Solver: General Gauss Elimination

This method consists of using Gauss elimination due to the matrix (A) only having values along the three diagonals of the matrix.

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & \dots & \dots & 0 \\ a_1 & b_2 & c_2 & 0 & \dots & \dots & 0 \\ 0 & a_2 & b_3 & c_3 & 0 & \dots & 0 \\ \vdots & \dots & \ddots & \ddots & \ddots & \dots & \vdots \\ \vdots & \dots & \dots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & a_{\text{n-1}} & b_{\text{n-1}} & c_{\text{n-1}} \\ 0 & \dots & \dots & \dots & 0 & a_{\text{n}} & b_{\text{n}} \end{bmatrix}$$

The diagonal of the matrix will be changed using the following formula while the "Upper Diagonal" will not be changed:

$$d_{\text{i}} = b_{\text{i}} - a_{\text{i}} c_{\text{i}} / d_{\text{i-1}}$$

The "Lower Diagonal" will be changed so that all elements below the diagonal are zeros. The vector (f) changes with the following formula:

$$w_{\text{i}} = f_{\text{i}} - a_{\text{i}} w_{\text{i-1}} / d_{\text{i-1}}$$

With backwards substitution then can be performed after following:

$$v_{\text{i}} = w_{\text{i-1}} - c_{\text{i-1}} v_{\text{i}} / d_{\text{i-1}}$$

with

$$d_1 = b_1, w_1 = f_1, and v_{\text{n}} = h^2 w_{\text{n}} / d_{\text{n}}$$

3 Results

This section will show the overall correctness of each calculation, and the average speed of each method.

3.1 Accuracy

The analytical solution to f(x) = 100*exp(-10*x) is known to be u(x) = 1-(1-exp(-10))*x-exp(-10*x). This is used to compare the accuracy of the numerical solutions obtained with both the general Gaussian elimination algorithm as well as the LU decomposition version for various n. Plots are generated to give a summarized view of the generated output files.

3.2 Gauss General:

Figure x shows the analytical solution and the numerical solution for various n using the general Gaussian elimination solver. As n increases, the result is more tightly fit to the analytical solution.

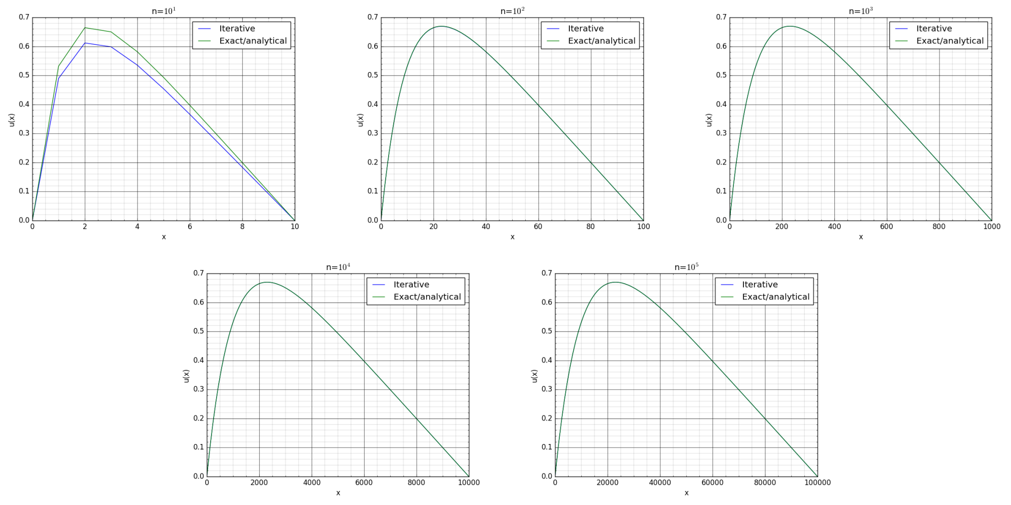

Figure 1: The relative error is computed using: e_i=—(v_i-u_i)/(u_i) where v_i is the analytical solution value, and u_i is the numerical solution value

The relative error is plotted for various n in figure y.

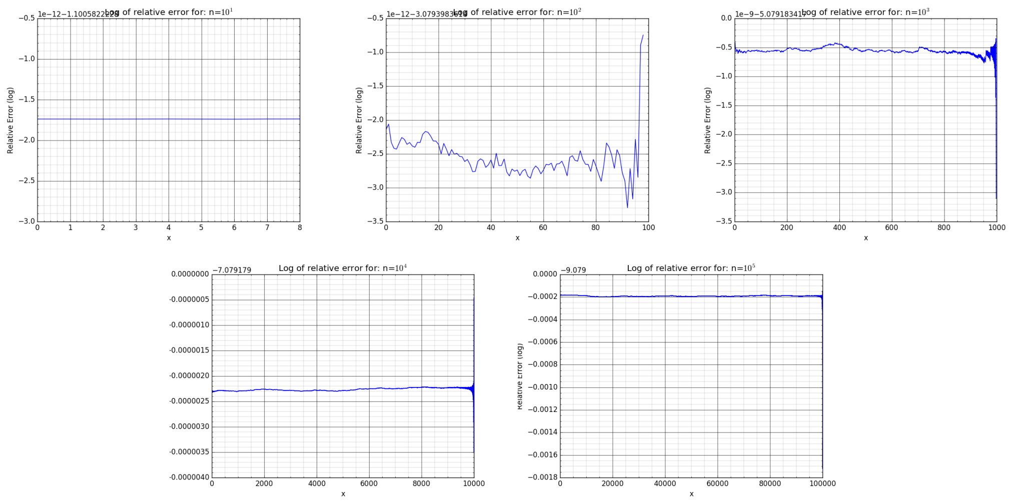

Figure 2: Plots of the relative error between analytical and exact solution for numerical Gaussian elimination

Plots of the relative error between analytical and exact solution for numerical Gaussian elimination. Additionally, to verify that the algorithm was working correctly, the overall error as the solver progresses was captured and plotted since the variations in relative error can seem unintuitive. For each n, the average of the relative error is acquired. Additionally, to verify that the algorithm was working correctly, the overall error as the solver progresses was captured and plotted since the variations in relative error can seem unintuitive. For each n, the average of the relative error is acquired.

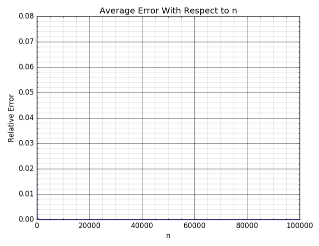

Figure 3: Average error with respect to n for Gaussian elimination on a linear scale

As n is increased, the overall relative error decreases, which is to be expected as the approximation is closer to the exact result. It is easier to see the trend when plotted on a log scale.

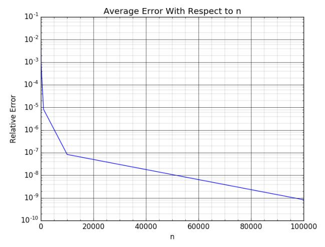

Figure 4: Average error with respect to n for Gaussian elimination on a log scale

3.3 LU Decomposition:

Figure a shows the analytical solution and the numerical solution for various n using the LU decomposition function-based solver. As n increases, the result is more tightly fit to the analytical solution which is expected.

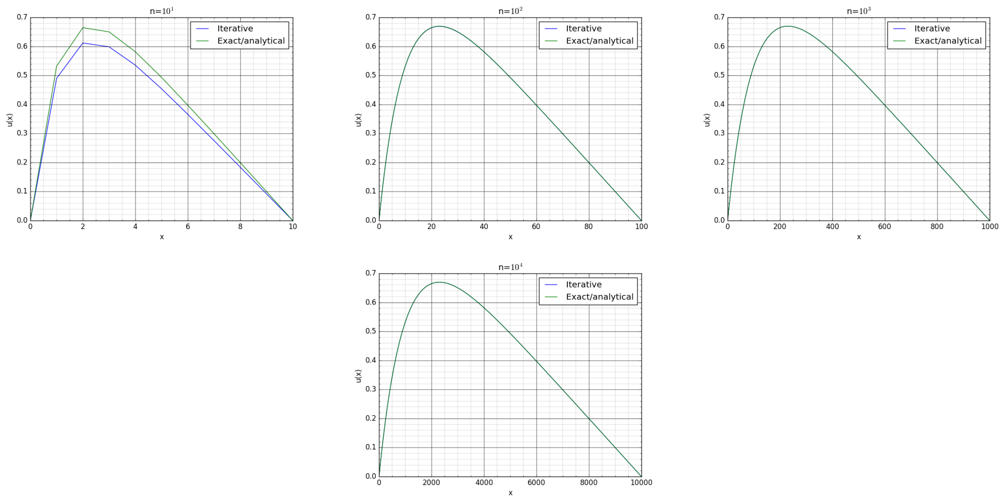

Figure 5: Plots of the analytical solution and numerical solution for solving with LU decomposition. The relative error is plotted for the LU decomposition as well in figure b.

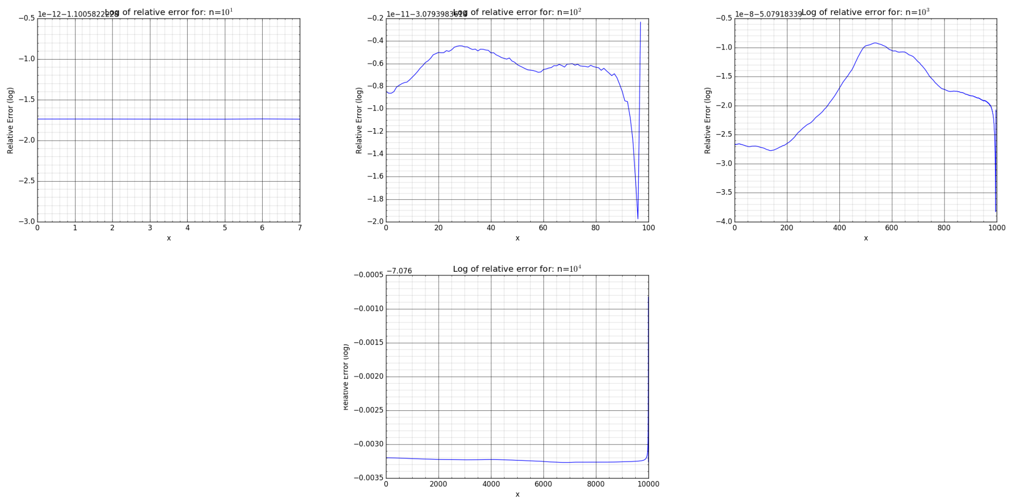

Figure 6: Plots of the relative error between analytical and exact solution when solving with LU decomposition. The overall error is also plotted, both on linear and log scales

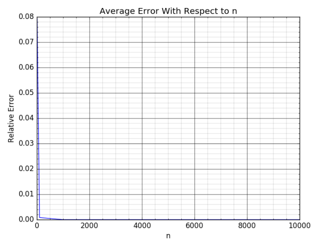

Figure 7: Average error for LU decomposition on a linear scale

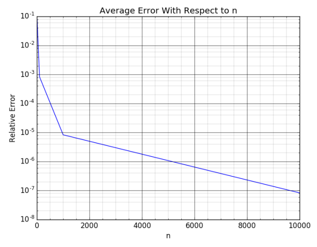

Figure 8: Average error for LU decomposition on a log scale

Interestingly enough, the LU decomposition could only go to $n = 10^4$ before running out of memory instead of $n = 10^5$ for the Gaussian elimination solver. Perhaps the implementation in the *Numpy* library does not allocate memory well.

3.4 Speed of Execution

Each method that has been utilized to solve the original equation has been timed using the *date.now*() functionality in Python. The fastest method was the General Gauss Elimination while LU Decomposition was lagging behind it. While I was definitely limited by my computing power I believe that these times could be reproduced by any other computer with reasonable computer power. The LU Decomposition was an average of 10 times slower than the General Solver.

	n = 2	n = 3	n = 4	n = 5
LU Decomposition	0.001441	0.014482	0.12671	54.6002
Gauss Elimination	0.008974	0.063471	0.575171	5.686529

Table 1: This chart shows how the average time for each method

The chart shows the average time for each method run ten (10) times. At $n < 5$ the LU Decomposition no longer is of the same magnitude, but it 10 times larger than the average times of the Gauss Elimination.

4 Conclusions

From examining all of the methods used to solve the original linear equation we have concluded that the Gauss Elimination method is the most efficient at finding our solution, followed by the LU Decomposition. Lastly the least efficient method used to solve the linear equation was the LU Decomposition Solver.

5 References

- Hjort-Jensen, M., 2015. Computational physics.
  https : //github.com/CompPhysics/ComputationalPhysics/