Project 1.
FYS3150. Computational Physics.

Ana Planes Hernandez.
github.com/anaplanes

September 9, 2019

ABSTRACT

The purpose of this project is to solve the one dimensional Poisson equation with Dirichlet boundary conditions, dealing with matrix operations and relative errors in order to get familiar with the aspects of numerical derivation and linear algebra, making use of memory handling in the program.

INTRODUCTION

The Poisson equation is a partial differential equation widely used in several fields of physics, such as electromagnetism, to calculate the potential field of a charge. In this project, it will be solved as a one dimensional equation, using two different methods.
First, it will be rewritten it in terms of a tridiagonal matrix to set an algorithm based on Gaussian elimination with the intention to find a numerical solution for the problem, aditionally discussing errors where the efficiency of the program will be tested. The second method consists of LU decomposition, which will be compared to the first algorithm in terms of floating point operations, checking the suitability of both methods for the present case.

ALGORITHMS

In order to solve the one dimensional Poisson equation,

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

with the conditions $xe(0,1), u(0) = u(1) = 0$, it will first be rewtitten using a discretised approximation of the second derivative.
The aproximation is defined with grid points $x_i = ih$ from $x_0 = 0$ to $x_{n+1} = 1$. The stepsize is $h = 1/(n+1)$ and the boundary conditions $u_0 = u_{n+1} = 0$

$$f_i = \frac{u_{i+1} + u_{i-1} - 2u_i}{h^2} \; i = 1, ...n$$

```
n=10
h=1/(n-1)

u = np.zeros(n+1)      #create empty arrays to fill in later with the loop
x = np.zeros(n+1)
f = np.zeros(n+1)
u[0] = 0               #boundary conditions
u[n] = 0
x[0] = 0               #interval considered x∈(0,1)
x[n] = 1

for i in range (n):
    x[i] = i*h
    u[i]= 1 - (1 - np.e**(-10))*x[i] - np.e**(-10*x[i])
    f[i] = ( u[i+1] + u[i-1] - 2*u[i]) / (h**2)
    print (i,x[i],f[i])
```

Figure 1: Function discretisation

Now,

if $i = 1$, $f_1 = \frac{u_2 + u_0 - 2u_1}{h^2}$

if $i = 2$, $f_2 = \frac{u_3 + u_1 - 2u_2}{h^2}$

if $i = n - 1$, $f_1 = \frac{u_n + u_{n+2} - 2u_{n-1}}{h^2}$

in summary, we can rewrite it in terms of a matrix

$$Au = g$$

where $g = h^2 f_i$

$$\begin{bmatrix} 2 & -1 & 0 & 0 & ... \\ -1 & 2 & -1 & 0 & ... \\ 0 & -1 & 2 & -1 & ... \\ ... & 0 & -1 & ... & ... \\ ... & ... & ... & ... & -1 \\ ... & ... & ... & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_n \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ . \\ . \\ . \\ g_n \end{bmatrix}$$

In order to resolve the equations and find the desired solution, we need to develop an algorithm, the **Gaussian elimination**, which is a method for solving a system of linear equations, by modifying the matrix so that all of the entries except for those in the diagonal are removed to be able to solve the equations in an immediate way.
To make it simpler, we will consider a 4x4 matrix. Consequently, we are left with the following equation:

$$\begin{bmatrix} d_1 & c_1 & 0 & 0 \\ a_1 & d_2 & c_2 & 0 \\ 0 & a_2 & d_3 & c_3 \\ 0 & 0 & a_3 & d_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}$$

$$\begin{gathered} d_1u_1 + c_1u_2 + 0 + 0 = g_1 \\ a_1u_1 + d_2 + c_2u_3 + 0 = g_2 \\ 0 + a_2u_2 + d_3u_3 + c_3u_4 = g_3 \\ 0 + 0 + a_3u_3 + d_4u_4 = g_4 \end{gathered}$$

There are two steps for Gaussian elimination: forward substitution and backward substitution.
The purpose of the forward substitution is to modify the matrix so that we will have an upper tridiagonal matrix.
We start by multiplying the first equation so as to remove the first term, bearing in mind that all of the other terms will be modified as well once we do it. Then, we repeat the same steps for the next few rows in the matrix.

Hence, we multiply the first equation by $\frac{c_1}{d_1}$, and subtract the second and first row, which will modify the matrix so that:

$$\begin{bmatrix} d_1 & c_1 & 0 & 0 \\ 0 & d_2' & c_2 & 0 \\ 0 & a_2 & d_3' & c_3 \\ 0 & 0 & a_3 & d_4' \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2' \\ g_3' \\ g_4' \end{bmatrix}$$

obtaining:

$$\begin{aligned} d_2' &= d_2 - \frac{a_1c_1}{d_1} \\ g_2' &= g_2 - g_1\frac{a_1}{d_1} \end{aligned}$$

If we multiply the third row by $\frac{e_2}{d_2'}$ and subtract once more, we obtain a similar pattern for the modified elements of the matrix.

$$\begin{aligned} d_3' &= d_3 - \frac{a_2c_2}{d_2'} \\ g_3' &= g_3 - g_2\frac{a_2}{d_2'} \end{aligned}$$

Generalising, the algorithm needed to carry out the forward substitution is the following:

$$d_i' = d_i - \frac{a_{i-1}c_{i-1}}{d_{i-1}'}$$

$$g_i' = g_i - \frac{g_{i-1}'a_{i-1}}{d_{i-1}'}.$$

$$d_1' = d_1$$

In Phyton, the algorithm may be written in the following way:

```
#Algorithm for forward substitution

for i in range (2, n+1):                                    # 6 floating point operations.
    dt[i] = d[i] - (a[i-1]*c[i-1])/dt[i-1]
    gt[i] = g[i] - gt[i-1]*a[i-1]/dt[i-1]
```

Figure 2: Forward substitution

The agorithm for the backward substitution is quite similar. The objective is, by working with the already modified matrix, to turn it into a lower triangular matrix.
After performing the forward substitution, we have:

$$\begin{bmatrix} d_1 & c_1 & 0 & 0 \\ 0 & d_2' & c_2 & 0 \\ 0 & 0 & d_3' & c_3 \\ 0 & 0 & 0 & d_4' \end{bmatrix}$$

We seek to remove the remaining c's up the diagonal, by multiplying the remaining elements repeatedly.
The algorithm needed to do this is:

$$u_i = \frac{g_i' - c_i u_{i+1}}{d_i'}$$

In Phyton, the backward substitution can be written as:

```
#Algorithm for backward substitution

u2[n] = gt[n] / dt[n]
for i in range(n-1, 0, -1):                                 # 3 floating point operations in total.
    u2[i] = (gt[i]-c[i]*u2[i+1]) / dt[i]
```

Figure 3: Backward substitution

If we now consider a particular case in which the elements up and down the diagonal are equal, things may be simplified, but the algorithm is esentially the same.

$$\begin{bmatrix} d_1 & e_1 & 0 & 0 \\ e_1 & d_2 & e_2 & 0 \\ 0 & e_2 & d_3 & e_3 \\ 0 & 0 & e_3 & d_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix}$$

$$\begin{gathered} d_1u_1 + e_1a_2 + 0 + 0 = g_1 \\ e_1u_1 + d_2 + e_2u_3 + 0 = g_2 \\ 0 + e_2u_2 + d_3u_3 + e_3u_4 = g_3 \\ 0 + 0 + e_3u_3 + d_4u_4 = g_4 \end{gathered}$$

The algorithm for the forward substitution in this case will be:

$$d'_i = d_i - \frac{e^2_{i-1}}{d'_{i-1}}$$

$$g'_i = g_i - \frac{g'_{i-1}e_{i-1}}{d'_{i-1}}.$$

$$d'_1 = d_1$$

And for backward substitution:

$$u_i = \frac{g'_i - e_iu_{i+1}}{d'_i}$$

The same results are obtained, but this second way is more effective, as the number of floating points is reduced.

```
#Algorithm for forward substitution

for i in range (2, n+1):
    dt[i] = d[i] - (e[i-1]**2)/dt[i-1]
    gt[i] = g[i] - gt[i-1]*e[i-1]/dt[i-1]

#Algorithm for backward substitution

u22[n] = gt[n] / dt[n]
for i in range(n-1, 0, -1):
    u22[i] = (gt[i]-e[i]*u22[i+1]) / dt[i]
```

Figure 4: A different option

The **LU decomposition** method consists of rewriting our matrix A as a product of two different matrices, a lower triangular one, L, and an upper triangular one, U in order to simplify the resolution of the equation.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

if we normalise so that $l_{11} = l_{22} = l_{33} = l_{44}$, two equations need to be solved:

$$LZ = y$$
$$Ux = Z$$

First, we resolve the first one iterating forwards.

$$\begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$$z_1 = y_1$$
$$l_{21}z_1 + z_2 = y_2$$
$$l_{31}z_1 + l_{32}z_2 + z_3 = y_3$$
$$l_{41}z_4 + l_{42}z_2 + l_{43}z_3 + z_4 = y_4$$

once the matrix Z is found, we proceed to find the matrix X working backwards.

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

RESUTS

Carrying out the Gaussian elimination method, we obtain very similar results for the numerical and analytical solution of the problem. A plot is attached to show it in a more graphical way.

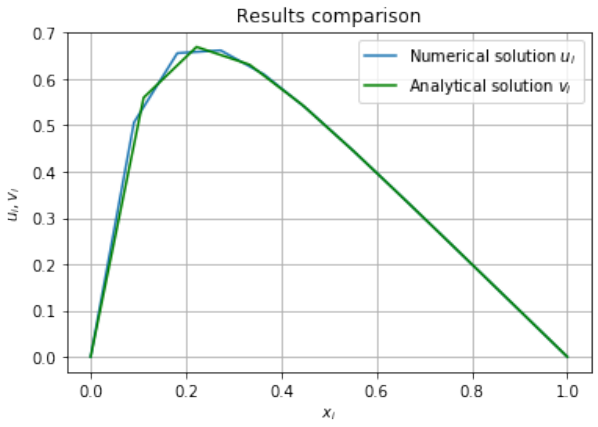

Figure 5: Plot for n=10

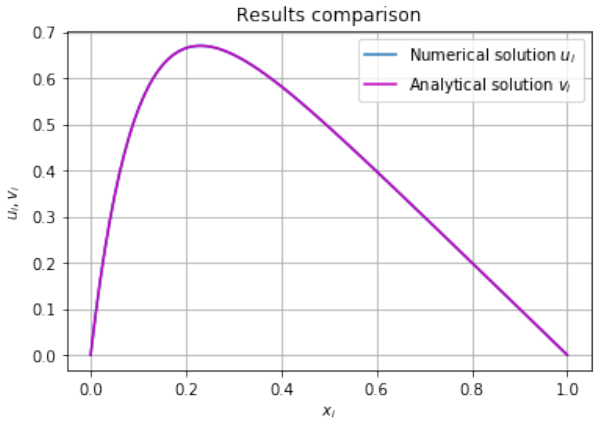

Figure 6: Plot for n=100

We can observe how much better the results are when n is bigger.
Now, we represent the maximum relative error as a function of $log_{10}(h)$. As we make h small, the error goes down. However, there's a certain point where, if we make h even smaller, the relative error will start going up again.

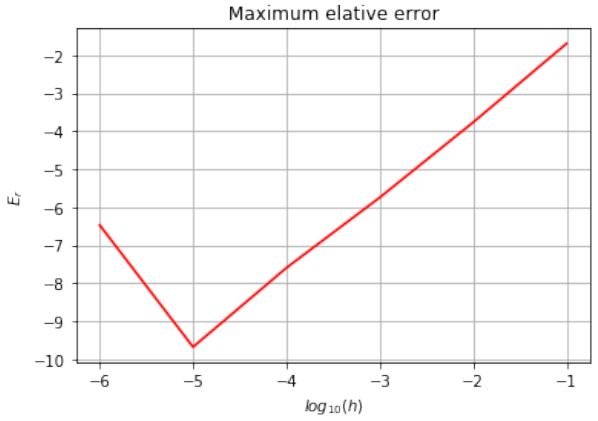

Figure 7: Maximum relative error

The number of floating points needed for Gaussian elimination is $O(8n)$, while if we use LU decomposition the flops are $O(\frac{2}{3}n^3)$. Therefore, the Gaussian elimination method is much more efficient than the LU decomposition method.

CONCLUSION

In this project, we have studied two important methods to solve a set of linear equations, being able to see the pros and cons of each one.
By the results we have obtained, we can conclude that Gaussian elimination is a much more efficient and suitable algorithm than LU decomposition when it comes to solving linear equations. The fact that LU requires a higher number of floating points decreases the performance of the algorithm, as it takes more time and memory to do the same thing as the Gaussian elimination.

REFERENCES

[1] Hjorth-Jensen, M. (2015). *Computational Physics. Lecture Notes 2015.* University of Oslo.