FYS3150 - PROJECT 1

Federico Nardi,[1] Davide Saccardo,[2] and Lorenzo Speri[3]

[1]*Department of Procrastination, University of Akiva*
[2]*Department of Odd Travels, University of Anoat*
[3]*Department of Even Travels, University of Ambria*

In this project we study the loss of numerical precision due to round-off errors made by the calculator. To do that we implement a method to solve a classical Poisson problem with Dirichlet boundary conditions and we analyze the step-size dependance of the relative deviation between the numerical results and the analytical expression. Furthermore we analize the efficiency of two different methods that can be implemented in the algorithm to solve linear systems: LU decomposition and Gaussian elimination. Moreover, we develope a particular algorithm to solve our specific system for the Poisson problem.

INTRODUCTION

The aim of this project is to study the loss of numerical precision due to the fact that a computer can represent a finite amount of numbers. We analize that in the context of computing a numerical second derivative to solve a Poisson problem with Dirichlet boundary conditions:

$$\begin{cases} -\frac{d^2u(x)}{dx^2} = f(x) \\ u(0) = 1 \\ u(1) = 0 \end{cases} \tag{1}$$

Where the source term is set to $f(x) = 100e^{-10x}$. To do that we define a relative error

$$\varepsilon = \frac{|v-u|}{|u|} \tag{2}$$

and we modify the step-size h to see up to when it follows the mathematics and goes like $O(h^2)$.

Numerically, we reduce the problem to solving a linear system of equations; we can implement the algorithm in two ways: by using Gaussian Elimination or LU decomposition. We analize the efficiency of both methods by computing the time the calculator needs to run the code and the number of flops. Moreover we develope and see the efficiency of a third method to solve our specific problem, that does not imply defining a matrix.

In the further sections we will go through the way we develope those algorithms, how we implement them on a C++ code and the results we obtain by running it.

METHODS AND ALGORTIHMS

According to Taylor expansion we can rewrite the second derivative as:

$$\frac{d^2u(x)}{dx^2} = \frac{u(x+h)+u(x-h)-2u(x)}{h^2} + O(h^2)$$

we discretize the interval $(0,1)$ in $n+2$ points $x_i = i\,h$ with $h = 1/(n+1)$ from $x_0 = 0$ to $x_{n+1} = 1$. If we neglect the term $O(h^2)$ the equation (1) becomes :

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

where $f_i = f(x_i)$ and $v_i = u(x_i)$. We can explicitate this equation as a linear set of equations of the form

$$\begin{aligned} 2v_1 - 1v_2 + 0v_3 + 0v_4 + \cdots &= h^2 f_1 \\ -1v_1 + 2v_2 + -1v_3 + 0v_4 + \cdots &= h^2 f_2 \\ 0v_1 - 1v_2 + 2v_3 + -1v_4 + \cdots &= h^2 f_2 \\ \vdots \end{aligned}$$

If we define $r_i = h^2 f_i$ and $\mathbf{A}$ as an $n \times n$ tridiagonal matrix, we can rewrite the set of equations as

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

$$\text{where} \qquad \mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{bmatrix}, \tag{3}$$

Now, we can use the gaussian elimination applied to this specific matrix $\mathbf{A}$. Following this procedure, we define two new variables:

$$\begin{aligned} \tilde{d}_i &= d_i - \frac{e_{i-1}^2}{\tilde{d}_{i-1}} \\ \tilde{r}_i &= r_i - \tilde{r}_{i-1}\frac{e_{i-1}}{d_{i-1}} \end{aligned} \tag{4}$$

where d_i represents the diagonal of the matrix $\mathbf{A}$, and e_i represents the elements in the upper and lower diagonals. This substitution is called *forward substitution.*
In our particular case ($d_i = 2$, $e_i = -1$), these variables become

$$\begin{aligned} \tilde{d}_i &= 2 - \frac{1}{d_{i-1}} = \frac{i+1}{i} \\ \tilde{r}_i &= r_i - \frac{\tilde{r}_{i-1}}{d_{i-1}} \end{aligned} \tag{5}$$

FIG. 1. Our results could not be better

where $i = 2, 3, \dots, n$ and $d_1 = 2$ and $\tilde{r}_1 = r_1$.

After that we obtain an upper triangular matrix:

$$\tilde{\mathbf{A}} = \begin{bmatrix} d_1 & e_1 & 0 & \dots & \dots & 0 \\ 0 & \tilde{d}_2 & e_2 & 0 & \dots & \dots \\ 0 & 0 & \tilde{d}_3 & e_3 & 0 & \dots \\ & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & & 0 & \tilde{d}_{n_1} & e_n \\ 0 & \dots & & 0 & 0 & \tilde{d}_n \end{bmatrix}, \tag{6}$$

and our system becomes

$$\tilde{\mathbf{A}}\mathbf{v} = \tilde{\mathbf{r}}$$

At this point we can find the solutions, from the last to the first, with the so called *backward substitution* i.e.

$$u_n = \frac{\tilde{r}_n}{\tilde{d}_n} \qquad u_i = \frac{(\tilde{r}_i - e_i u_{i+1})}{\tilde{d}_i} \tag{7}$$

In addition, we can represents a general system using coefficients (a_i, b_i, c_i) as

$$\tilde{\mathbf{A}} = \begin{bmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 \\ a_1 & b_2 & c_2 & 0 & \dots & \dots \\ 0 & a_2 & b_3 & c_3 & 0 & \dots \\ & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & & a_{n-3} & b_{n_1} & c_n \\ 0 & \dots & & 0 & a_{n1} & b_n \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ \vdots \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} r_1 \\ \vdots \\ \vdots \\ \vdots \\ r_n \end{bmatrix} \tag{8}$$

By following this reasoning, we develop a first general algorithm[**?**]. After that, finally, with forward and backward subst., using expressions (**??**) the solutions are found.

If we specialize the first code to our case $b_i = 2$, $a_i = -1$, $c_i = -1$, we don't have to fill any matrix, and the computer can proceed directly to solve the particular equations (**??**).

These three codes can be tested to find which one is the most efficient. At the beginning we construct the algorithms with dynamic allocation memory with pointers, but then we shift to the allocation provided by the library Armadillo.

OUR RESULTS

We present our results in Fig. .

CONCLUSIONS AND PERSPECTIVES

What a wonderful world!