Project 1

Antonio Gómez Garrido and David Martínez Hernández

(Dated: September 13, 2022)

https://github.com/GatoMartinez/Pachuco-Gavotte.git

PROBLEM 1

First, we need to check if the given function satisfies the boundary conditions, which are: $u(0) = u(1) = 0$
In our case,

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{1}$$

So, when computing the boundary conditions, we get:

For the case of $u(0)$:
$u(0) = 1 - (1 - e^{-10})0 - e^{-10 \cdot 0}$
$u(0) = 1 - e^{0}$
$u(0) = 1 - 1$
$u(0) = 0$

For the case of $u(1)$:
$u(1) = 1 - (1 - e^{-10})1 - e^{-10 \cdot 1}$
$u(1) = 1 - 1 + e^{-10} - e^{-10}$
$u(1) = 0$

Now that we have verified that the function given in this section satisfies the boundary conditions, we must prove that it also satisfies the Poisson equation.

$$-\frac{d^2}{dx^2}\big(1 - (1 - e^{-10})x - e^{-10x}\big) = -\frac{d}{dx}\big(-1 + e^{-10} + 10e^{-10x}\big)$$
$$-\frac{d^2u}{dx^2} = -(-100e^{10x})$$
$$-\frac{d^2u}{dx^2} = 100e^{-10x}$$

PROBLEM 2

2.a)

The values of x and $u(x)$ are all stored inside the file 'valores.txt'

2.b)

The plot obtain in python from the previously mentioned data is the following:

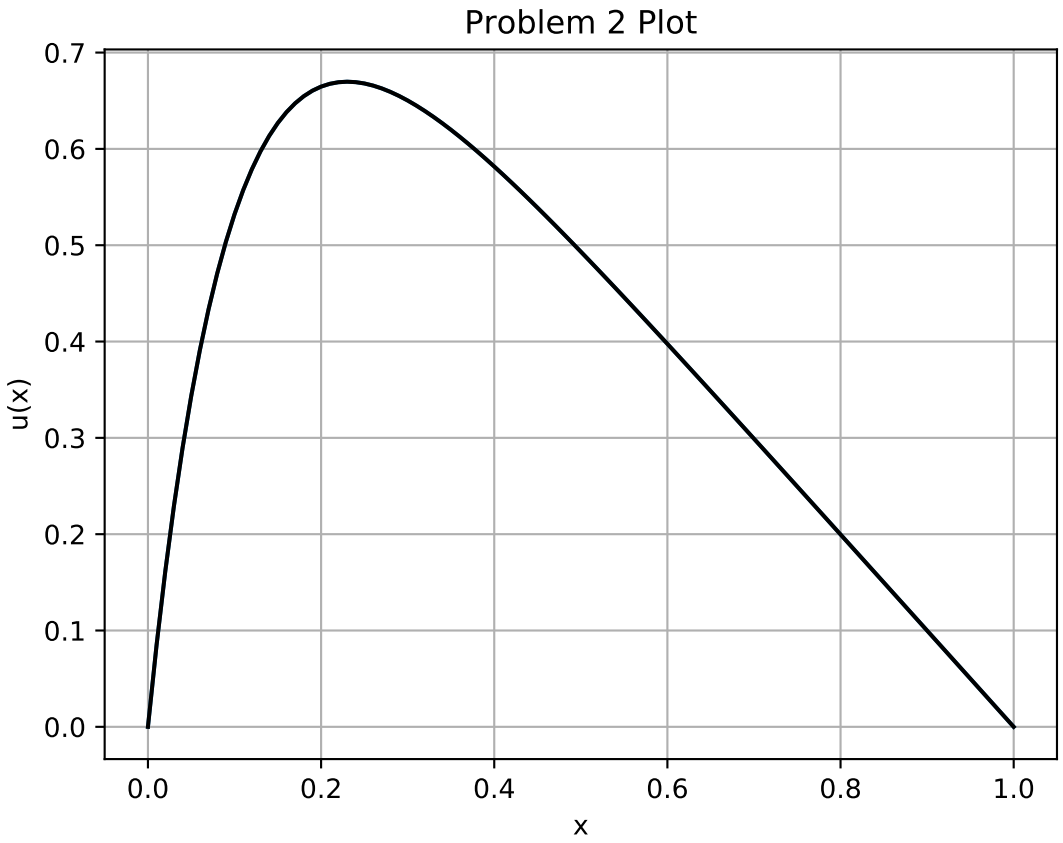

Figure 1. $u(x)$ against x

PROBLEM 3

If we Taylor expand our function $u(x)$ and evaluate it at points $x_o = x+h$ and $x_1 = x-h$, we obtain the expresions:

$$u(x+h) = u(x) + u'(x)\cdot h + \frac{1}{2!}u''(x)\cdot h^2 + \frac{1}{3!}u'''(x)\cdot h^3 + \mathcal{O}(h^4)$$
$$u(x-h) = u(x) - u'(x)\cdot h + \frac{1}{2!}u''(x)\cdot h^2 - \frac{1}{3!}u'''(x)\cdot h^3 + \mathcal{O}(h^4)$$

By summing these two expresions and rearranging the terms, we get the well known formula for the second derivative of a function:

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

Once we get to this, the next step is to simplify the equation by setting h to be arbitrarilly small and thus disregarding the last term of the sum.

$$\frac{d^2v}{dx^2} = -\frac{v(x+h) - 2v(x) + v(x-h)}{h^2} \tag{3}$$

Here, we have redefined our function to take into account the fact that this is an approximation.

Now we insert this expresion in the Poisson formula and discretize the domain of the function. This is, instead of taking a continuous set of values of x, we choose a finite set of points of this domain and evaluate the function in such points.

Finally, the problem we're trying to solve adquires the form:

$$\frac{v_{i+1} - 2v_i + v_{i-1}}{h^2} = f_i \tag{4}$$

Here, the subindex i indicates the point of the previosly mentioned set of points that we're taking into account.

PROBLEM 4

First of all, we need to rearrange the terms in equation 4 to make them of the form: $-v_{i+1} + 2v_i - v_{i-1} = h^2 f_i$. Suppose that, in our discretization, we have chosen a set of N points from the function's domain (without counting the extremes). We, as a result, have N equations:

$$\begin{aligned}
-v_0 + 2v_1 - v_2 &\quad ... &&= h^2 f_1 \\
-v_1 + 2v_2 - v_3 &\quad ... &&= h^2 f_2 \\
-v_2 + 2v_3 - v_4 &\quad ... &&= h^2 f_3 \\
... &\quad ... &&\quad ... \\
... &\quad -v_{N-1} + 2v_N - v_{N+1} &&= h^2 f_N
\end{aligned}$$

But, as both v_0 and v_{N+1} are already known (they are the boundary conditions in our problem), we can rearrange the terms in the first and last equations in order to have all of the unknowns on one side and all of the given data on the other one. If we now redefine the right sides of the equations (which we will choose to be the one with the problem's data) as follows: $h^2 f_1 + v_0 \equiv g_1$, $h^2 f_i \equiv g_i \ \forall i = 2, 3, ..., N-1$, $h^2 f_N + v_{N+1} \equiv g_N$ we get the system of equations:

$$\begin{aligned}
2v_1 - v_2 &\quad ... &&= h^2 g_1 &(5)\\
-v_1 + 2v_2 - v_3 &\quad ... &&= h^2 g_2 &(6)\\
- v_2 + 2v_3 - v_4 &\quad ... &&= h^2 g_3 &(7)\\
... &\quad ... &&\quad ... &(8)\\
... &\quad -v_{N-1} + 2v_N &&= h^2 g_N &(9)
\end{aligned}$$

Which, if we take into account the rules for the multiplication of matrices, can be written as:

$$A\vec{v} = \vec{g} \tag{10}$$

Where the vector $\vec{v} = (v_1, v_2, ..., v_N)$ contains the values of our function $v(x)$ (which is the unknown of the problem), $\vec{g} = (g_1, g_2, ..., g_N)$ contains all of the g_i defined above and A represents a tridiagonal matrix with all of its diagonal elements equal to two and all of the elements on its subdiagonal and superdiagonal equal to -1.

PROBLEM 5

From the reasoning developed in the previous section, we can easily see that, when setting the equations 5 to 9, two of the elements of $\vec{v^*}$ (v_0^* and v_{N+1}^* to be more precise) are missing. Therefore, the relation between m and n is that $m = n + 2$. Apart from this, as we've stated previously, the part of the solution that we get from solving equation 10 is the values of the function in every point of its domain except for the extremes.

PROBLEM 6

6.a)

This algorithm is based on the Gaussian elimination method which, in a nutshell, consists on performing a series of row-operations on the matrix of the system (A) in order to turn it into a triangular matrix; this is, a matrix for which every element below the diagonal is equal to 0.

Take, for example, the first two rows of our A matrix (let R_1 be the first, and R_2 the second one). In order to remove the element a_{21}, which is right below the first element of the diagonal,we must do the operation $\tilde{R_2} = R_2 - \frac{a_2}{b_1}R_1$. After that, we do the same for all of the consecutive rows to eliminate every element below the diagonal.

After doing all of these operations, the new elements of the matrix's diagonal and $\vec{g}$ look like this:

$$\tilde{b}_1 \equiv b_1 \qquad \tilde{g}_1 \equiv g_1 \tag{11}$$

$$\tilde{b}_i \equiv b_i - \frac{a_i}{\tilde{b}_{i-1}}c_{i-1} \qquad \tilde{g}_i \equiv g_i - \frac{a_i}{\tilde{b}_{i-1}}\tilde{g}_{i-1} \tag{12}$$

Consequently,the last one of the system's equations adquires the form:

$$\tilde{b}_N \cdot v_N = \tilde{g}_N$$

From which we can obtain the value for v_N. Once we have this value, we can insert it in the previous equation in order to know v_{N-1} and so on.

$$v_N = \frac{\tilde{g}_N}{\tilde{b}_N} \tag{13}$$

$$v_i = \frac{\tilde{g}_i - c_i \cdot v_{i+1}}{\tilde{b}_i} \tag{14}$$

Once we take all of this into account, the algorithm consists on the following steps:

1. Store the elements of the matrix's subdiagonal, main diagonal and superdiagonal in three different vectors of the type std::vector<double>.

2. Create three empty vectors of the same size and type of the matrix's diagonal (one for storing the values of $\vec{\tilde{b}}$, another one for the values of $\vec{\tilde{g}}$, and one last vector for storing the solution to the problem $\vec{v}$)

3. Run a loop for computing the values of $\tilde{b_i}$ and $\tilde{g_i}$ given by the expressions in eq: 12 and storing them in the previously mentioned vectors.

4. Run another loop for doing this same operation for the case of the v_i values which are now given bi eq: 14. This can't be done in the same loop of the other two vectors because the values of $\tilde{b_i}$ and $\tilde{g_i}$ are needed in order to cumpute the values of the different v_i.

In a more technical way, our algorithm would look something like this:

Algorithm 1 General Algorithm

```
std :: vector < double > a = a1, a2, ..., aN − 1; ;          ▷ This is the definition
std :: vector < double > b = b1, b2, ..., bN;                ▷ of the subdiagonal, main diagonal and
std :: vector < double > c = c1, c2, .., cN − 1              ▷ superdiagonal vectors respectively
std :: vector < double > g = g1, g2..., gN                   ▷ We also define the g vector
std :: vector < double > b_tilde(N);                         ▷ Now define empty vectors
std :: vector < double > g_tilde(N);                         ▷ where we will store the values of
std :: vector < double > v(N);                               ▷ b̃_i, g̃_i and v_i
if a.size() ≠ c.size() or b.size() − a.size() ≠ 1 or b.size() ≠ g.size() then
    std :: cout "the equality is not possible";              ▷ we run this if to make sure that
    return 0                                                 ▷ the probblem's data have been introduced correcticly
else
    b_tilde[0] = b[0]; g_tilde[0] = g[0]; v[0] = 0;
    for i=2,3,...,N do                                       ▷ in our c++ code the indexes will be
        b_tilde[i] = b[i] − (a[i−1] / b_tilde[i−1]) · c[i − 1]   ▷ different because of the nature of std::vectors
        g_tilde[i] = g[i] − (a[i−1] / b_tilde[i−1]) · g_tilde[i − 1]
    v[N] = g_tilde[N] / b_tilde[N]
    for i=1,2,...,N-1 do
        int j = N − i                                        ▷ This is the way we've come up with to run the loop "backwards"
        v[j] = (g_tilde[j] − c[j] · v[j + 1])/(b_tilde[j])
```

6.b)

To count the number of FLOPs in this algorithm we need to take a deeper look into equations 11, 12, 13 and 14

Following the procedure used in class, every FLOP will be preceded by $fl()$ to make it easier to count them. Now, if we look at eq. 12 (we will only be examining the expressions for the values of $\tilde{b}$ given that they are identical to the ones of $\tilde{g}$):

$$fl\left\{fl(b_i) - fl\left[fl(c_{i-1}) \cdot fl\left(\frac{fl(a_{i-1})}{fl(\tilde{b}_{i-1})}\right)\right]\right\} \tag{15}$$

We can count up to seven FLOPs for every one of these operations. If we now take into account the fact that this operation is repeated twice (one time for $\tilde{b}$ and another one for $\tilde{g}$) and that, on top of that, these operations are repeated $N-1$ times in the for loop, we get $14(N-1)$ FLOPs. We must now add the 2 FLOPs present in the definitions of $\tilde{b}_1$ and $\tilde{g}_1$.

Now for the case of eq.14:

$$fl\left\{\frac{fl\left[fl(g_i) - fl\left(fl(c_i) \cdot fl(v_{i+1})\right)\right]}{fl(b_i)}\right\} \tag{16}$$

Where we count 7 more FLOPs for each one of the $N-1$ iterations in the loop.

Finally, when calculating v_N we have 3 more FLOPs:

$$fl\left(\frac{fl(\tilde{g}_N)}{fl(\tilde{b}_N)}\right) \tag{17}$$

In conclusion, the total ammount of FLOPs performed in or algorithm is: $FLOPs = 21(N-1) + 5$.

PROBLEM 7

7.b)

From the program in secton 7.a we have obtained the following solutions:

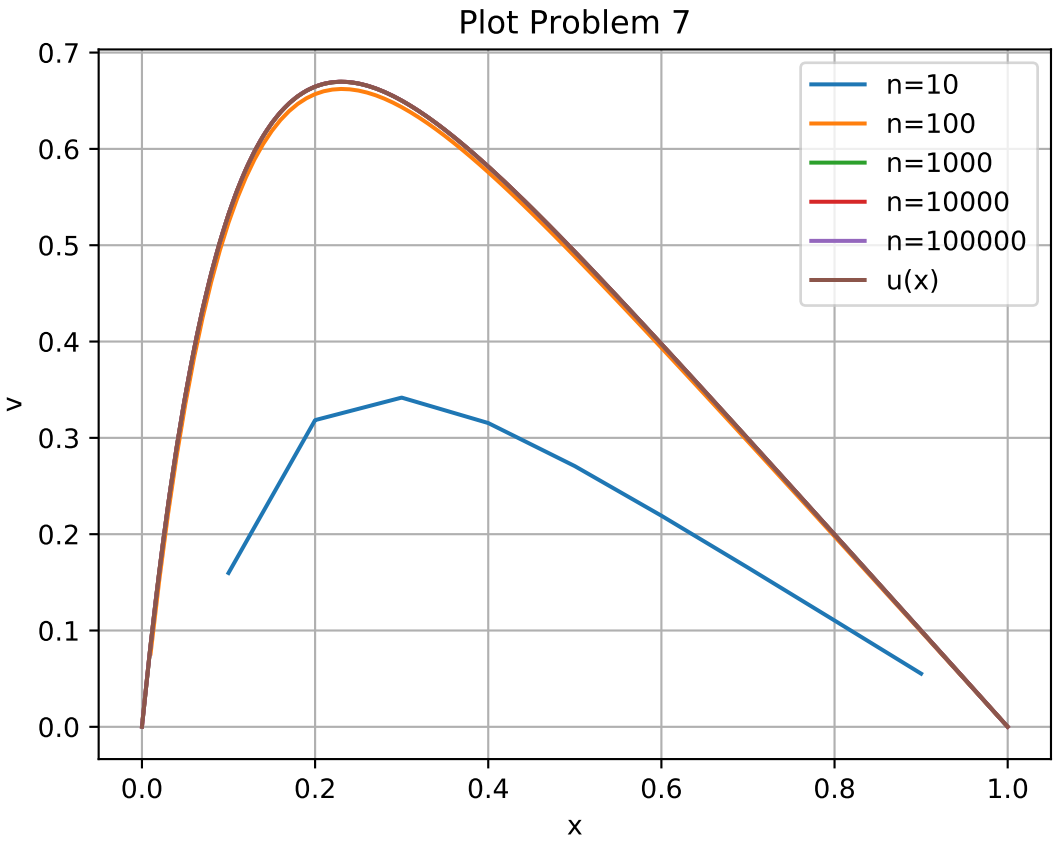

Figure 2. The different colors represent the solutions obtained from different approximations (different values of n_{steps})

As we can see, when we increase the number of n_{steps}, the approximations become closer to the analytic solution.

PROBLEM 8

8.a)

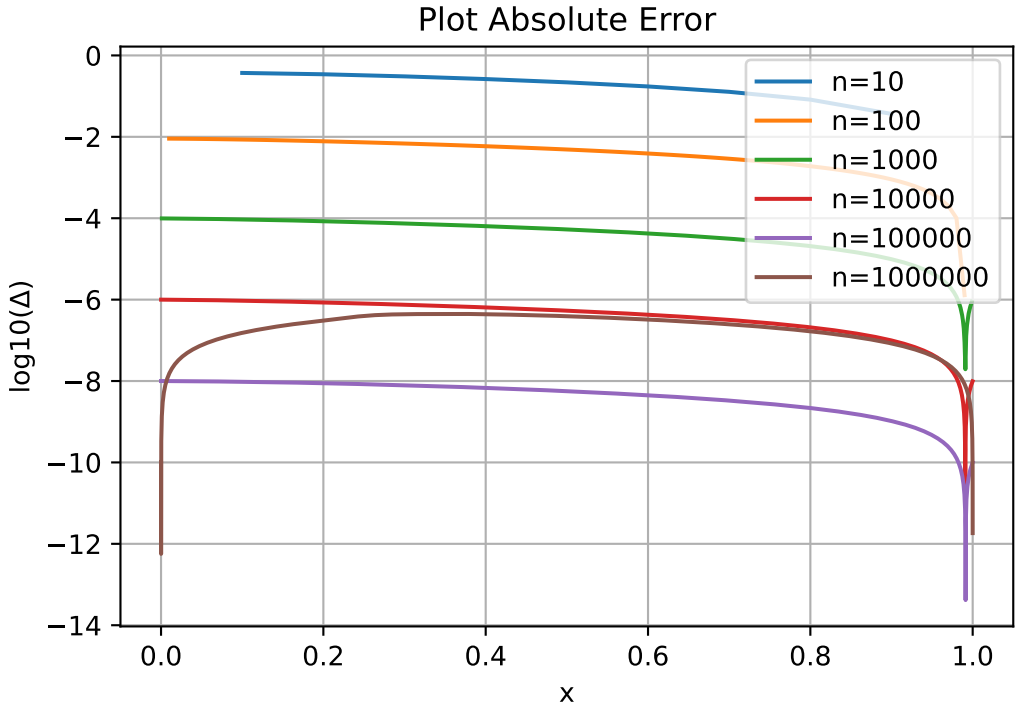

Figure 3. Representation of the absolute errors (their logarithms) in our approximations for different values of n_{steps}

8.b)

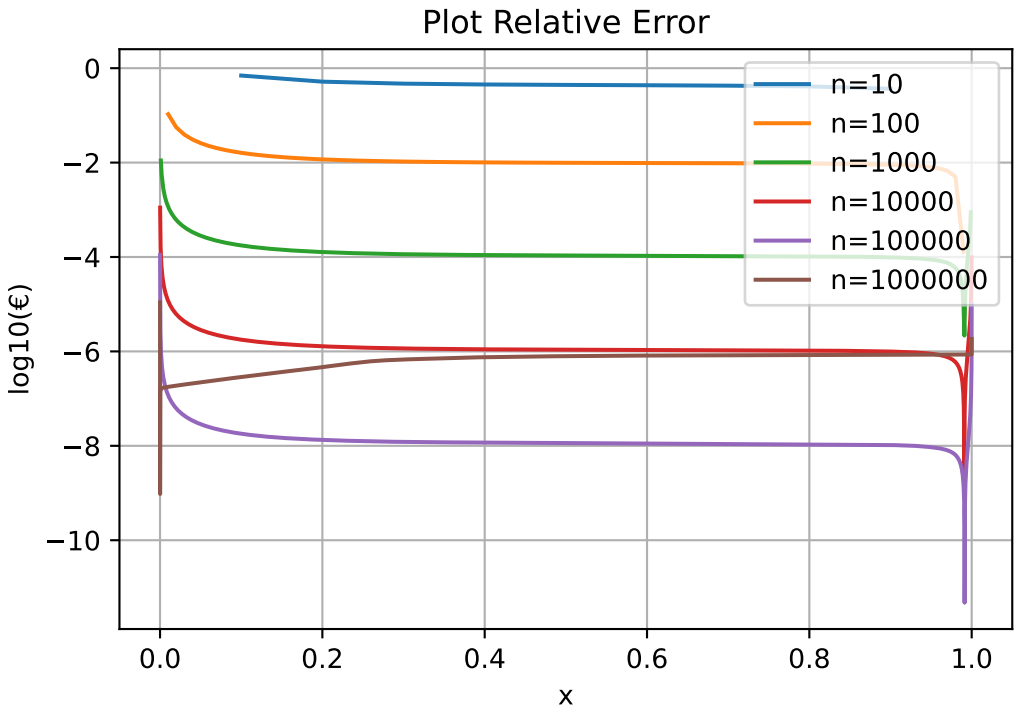

Figure 4. Representation of the relative errors ϵ_i against x for different values of n_{steps}

8.c)

n_{steps}	$log_{10}(\epsilon_i)$	ϵ_i
10	$-1.560 \cdot 10^{-1}$	$6.982 \cdot 10^{-1}$
100	$-9.747 \cdot 10^{-1}$	$1.060 \cdot 10^{-1}$
1000	$-1.956 \cdot 10^{0}$	$1.106 \cdot 10^{-2}$
10000	$-2.954 \cdot 10^{0}$	$1.111 \cdot 10^{-3}$
100000	$-3.954 \cdot 10^{0}$	$1.111 \cdot 10^{-4}$
1000000	$-4.961 \cdot 10^{0}$	$1.094 \cdot 10^{-5}$
10000000	$-5.525 \cdot 10^{0}$	$2.983 \cdot 10^{-6}$

PROBLEM 9

9.a)

The theoretical background for this problem is the same as the one for the case discussed un problem 6. This is because the case we're about to treat is just a concrete example of the more general problem discussed in section 6.

Contrary to what was set in such section, now the values for the elements of $\vec{a}$,$\vec{b}$ and $\vec{c}$, are not only constant, but they are also equal ($\vec{a} = \vec{c} = (-1, -1, ..., -1)$; $\vec{b} = (2, 2, ..., 2)$

As a result of this, equations 12 and 14 adquire an easier form, as now the values of a_i,b_i and c_i can be replaced by constants. Such form is the following one:

$$\tilde{b}_1 \equiv 2 \qquad \tilde{g}_1 \equiv g_1 \tag{18}$$

$$\tilde{b}_i \equiv 2 - \frac{-1}{\tilde{b}_{i-1}}(-1) \qquad \tilde{g}_i \equiv g_i - \frac{-1}{\tilde{b}_{i-1}}\tilde{g}_{i-1} \tag{19}$$

$$\tilde{b}_i \equiv 2 - \frac{1}{\tilde{b}_{i-1}} \qquad \tilde{g}_i \equiv g_i + \frac{1}{\tilde{b}_{i-1}}\tilde{g}_{i-1} \tag{20}$$

$$v_N = \frac{\tilde{g}_N}{\tilde{b}_N} \tag{21}$$

$$v_i = \frac{\tilde{g}_i - (-1)\cdot v_{i+1}}{\tilde{b}_i} \tag{22}$$

$$v_i = \frac{\tilde{g}_i + v_{i+1}}{\tilde{b}_i} \tag{23}$$

Apart from this, the fact that the three main diagonals of the matrix are constant imply that, for this new algorithm, there is no need to define the three vectors (their values are already reflected in the expressions 20 and 23). Therefore, this new algporithm looks like this:

Algorithm 2 Special Algorithm

```
std :: vector¡double¿g = g1, g2..., gN
std :: vector < double > b_tilde(N);
std :: vector < double > g_tilde(N);
std :: vector < double > v(N);
if a.size() ≠ c.size()  or  b.size() − a.size() ≠ 1  or  b.size() ≠ g.size() then
    std :: cout "the equality is not possible";
    return 0
else
    b_tilde[0] = 2; g_tilde[0] = g[0]; v[0] = 0;
    for i=2,3,...,N do
        b_tilde[i] = 2 − 1/b_tilde[i−1]                ▷ We replace the a,b and c variables with constants
        g_tilde[i] = g[i] + 1/b_tilde[i−1] · g_tilde[i − 1]
    v[N] = g_tilde[N]/b_tilde[N]
    for i=1,2,...,N-1 do
        int  j = N − i
        v[j] = (g_tilde[j]+v[j+1])/(b_tilde[j])
```

9.b)

In order to count the number of FLOPs in this algorithm, we will follow the same method as in section 6.b) but applied to equations 20 and 23.

$$fl\left\{fl(2.0) - fl\left[\frac{fl(1.0)}{fl(\tilde{b}_{i-1})}\right]\right\}$$

$$fl\left\{fl(g_i) + fl\left[fl(\tilde{g}_{i-1})\cdot fl\left(\frac{fl(1.0)}{fl(\tilde{b}_{i-1})}\right)\right]\right\}$$

$$fl\left\{\frac{fl\left[fl(\tilde{g}_i) + fl(v_{i+1})\right]}{fl(\tilde{b}_i)}\right\}$$

Between the three of these equations there are 17 FLOPs which are repeated $N-1$ times inside the for loops. Therefore, the total number of FLOPs in this algorithm is: $FLOPs = 17(N-1)+5$

PROBLEM 10

We are now going to compare the time taken by each, the general and the special algorithm, by making a table that represents the timing results and paying attention to the evolution of the difference as number of steps increases:

[h!]

n_{steps}	General Algorithm	Special Algorithm	Time Difference
10	$2.670 \cdot 10^{-5}$	$2.167 \cdot 10^{-5}$	$5.033 \cdot 10^{-6}$
100	$3.277 \cdot 10^{-5}$	$2.727 \cdot 10^{-5}$	$5.511 \cdot 10^{-6}$
1000	$1.358 \cdot 10^{-4}$	$8.836 \cdot 10^{-5}$	$4.744 \cdot 10^{-5}$
10000	$1.085 \cdot 10^{-3}$	$8.025 \cdot 10^{-4}$	$2.824 \cdot 10^{-4}$
100000	$1.033 \cdot 10^{-2}$	$7.770 \cdot 10^{-3}$	$2.562 \cdot 10^{-3}]$
1000000	$9.956 \cdot 10^{-2}$	$9.489 \cdot 10^{-2}$	$4.659 \cdot 10^{-3}$

We can check that, not only is the special algorithm faster but the time difference becomes significantly larger as n_{steps} grows.