Numerical Simulation and Scientific Computing

Exercice 2: Task 3

Elsayed Saleh Abrishq Bassam : 12302324

Ravelet Thomas : 12302809

Saudin Abdic : 11730114

Christoforos Eseroglou : e12331470

Compilation of codes:

“make” to compile the main.cpp file
“./solver without_source 10 10000 0.0 1.0” to execute the code without source (parameters may be changed)
“./solver with_source 50 100000 0.0 0.0 0.5 0.5 0.1” to execute the code with source (parameters may be changed)
“python plot.py” to execute the python script that will generate the output figures

Let’s consider a point N_0 such that $N_0 > N + 1$ with N the size of the grid.

Judging from the ordering of the points, there are:

- Center $c = N_0$
- West $w = N_0 - 1$
- East $e = N_0 + 1$
- North $n = N_0 + N$
- South $s = N_0 - N$

It is therefore possible to combine the different orientations together to get new ones:

- South-west $sw = s + w - c$
- South-east $se = s + e - c$
- North-west $nw = n + w - c$
- North-east $ne = n + e - c$

Now, there is a 3x3 stencil that represents a patch of the grid.

From any directions d (e.g. $d = se$), it is possible to get the coordinate of the point in the grid thanks to the relation: $x = d \,\%\, N$ -- $y = d/N$.

Let's use the Poisson's equation:

$$-\frac{\partial^2 u(x,y)}{\partial x^2} - \frac{\partial^2 u(x,y)}{\partial y^2} = f(x,y)$$

Approximation of (second) derivatives using FD will give:

$$-\frac{1}{h^2}(u_N + u_S + u_E + u_W - 4u_C) = f(c \,\%\, N, c/N)$$

Starting from this, we can compute the linear equation for the southwest point (1st point in the patch)

$$-\frac{1}{h^2}(u_w + u_{sw-N} + u_{sw+1=s} + u_{sw-1} - 4u_{sw}) = f\left(sw \,\%\, N, \frac{sw}{N}\right) \quad (1)$$

That can be rewritten as:

$$\frac{1}{h^2}(4u_{sw} - u_s + 0u_{se} - u_w + 0u_c + 0u_e + 0u_{nw} + 0u_n + 0u_{ne}) = f(x_{sw}, y_{sw}) + \frac{1}{h^2}(u_{sw-N} + u_{sw-1})$$

Or as a vector product:

$$\frac{1}{h^2} 4 \quad -1 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 * \begin{pmatrix} u_{sw} \\ u_s \\ u_{se} \\ u_w \\ u_c \\ u_e \\ u_{nw} \\ u_n \\ u_{ne} \end{pmatrix} = f(x_{sw}, y_{sw}) + \frac{1}{h^2}(u_{sw-N} + u_{sw-1})$$

If rewrite all the equations the same way, we finally get:

$$\frac{1}{h^2} * \begin{pmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{pmatrix} * \begin{pmatrix} u_{sw} \\ u_s \\ u_{se} \\ u_w \\ u_c \\ u_e \\ u_{nw} \\ u_n \\ u_{ne} \end{pmatrix} = \begin{pmatrix} f(sw) + \frac{1}{h^2}(u_{sw-N} + u_{sw-1}) \\ f(s) + \frac{u_{s-N}}{h^2} \\ f(se) + \frac{1}{h^2}(u_{se-N} + u_{se+1}) \\ f(w) + \frac{u_{w-1}}{h^2} \\ f(c) \\ f(e) + \frac{u_{e+1}}{h^2} \\ f(nw) + \frac{1}{h^2}(u_{nw-1} + u_{nw+N}) \\ f(n) + \frac{u_{n+N}}{h^2} \\ f(ne) + \frac{1}{h^2}(u_{ne+1} + u_{ne+N}) \end{pmatrix}$$

It is possible to represent the system matrix A_h thanks to three 3x3 block matrices D, I and 0:

$$D = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}$$

$$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Thanks to these matrices, it is possible to rewrite A_h as:

$$A_h = \begin{pmatrix} D & -I & 0 \\ -I & D & -I \\ 0 & -I & D \end{pmatrix}$$

The matrix A_h is diagonally dominant, because:

1st line / 3rd line / 7th line / 9th line: 4 > |-1| + |-1|
2nd line / 4th line / 6th line / 8th line: 4 > |-1| + |-1| + |-1|
5th line: 4 = |-1| + |-1| + |-1| + |-1|

To implement the matrix-free Jacobi solver, let's concentrate on the following equation:

$$-\frac{1}{h^2}\left(u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j} - 4u_{i,j}\right) = f(i,j)$$

That can be rewritten as:

$$u_{i,j} = \frac{1}{4} * \left(h^2 \cdot f(i \cdot h, j \cdot h) + u_{i,j+1} + u_{i,j-1} + u_{i+1,j} + u_{i-1,j}\right)$$

In a more "computing" way, the equation becomes:

$$u_{i,j}^{k+1} = \frac{1}{4} * \left(h^2 \cdot f(i \cdot h, j \cdot h) + u_{i,j+1}^{k} + u_{i,j-1}^{k} + u_{i+1,j}^{k} + u_{i-1,j}^{k}\right)$$

With the specified ordering of the points, for a random point n_0 the equation is:

$$u_{n_0}^{k+1} = \frac{1}{4} * \left(h^2 \cdot f(n_0) + u_{n_0+N}^{k} + u_{n_0-N}^{k} + u_{n_0-1}^{k} + u_{n_0+1}^{k}\right)$$

To calculate the residual error, we could calculate the norm of Ah*Uh - Bh. However, since Ah is of dimension 9x9, we can only calculate the error on a patch of dimension 3x3. To calculate the error of the entire simulation, we would have to split the Uh matrix into a multitude of patches of dimension 3x3, calculate the residual norm on each of these patches, then calculate the norm of all the norms. However, given that the solver algorithm is applied uniformly to the entire grid, it is sufficient to calculate the residual norm on just one randomly chosen patch of size 3x3.

The calculated error of the simulation will therefore be less precise than if it had been calculated on all the patches of dimensions 3x3, but this calculation on only one patch makes it possible to make a big saving on the calculation time of the error while providing a coherent value.

After the computation, this is the curve of norms errors:

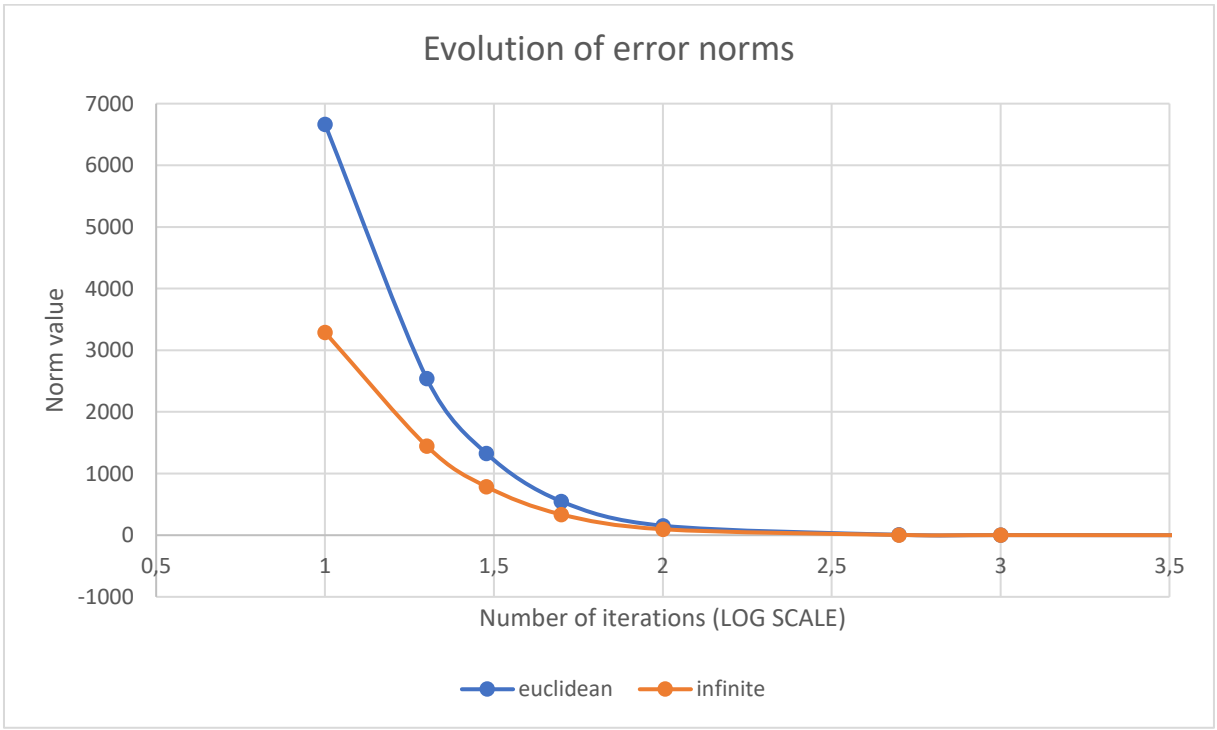

Figure 1: Evolution of error norms

This graph was generated using Excel by running the code several times and listing the error standards in an Excel table. It clearly shows that computing too much (over 10000 iterations) is a waste of resources and time, as the error decreases exponentially with the number of iterations.

The following figures show the output of two simulations, one with a source and one without, with the same boundary conditions:

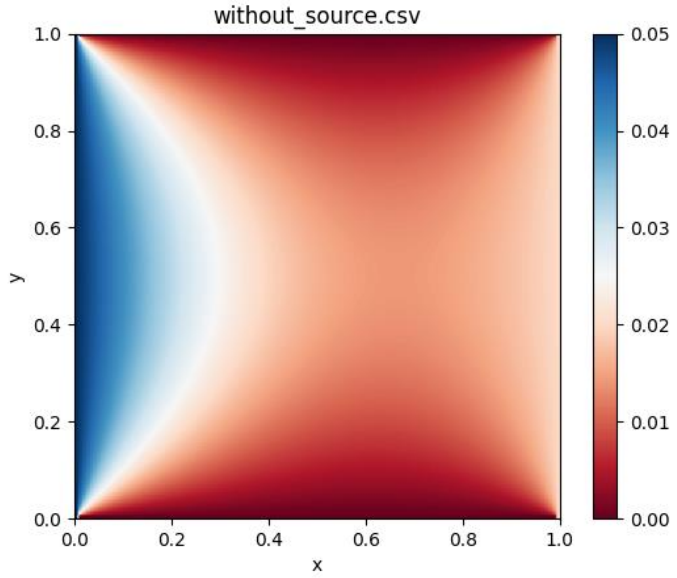

Figure 2: simulation with a source. Code executed with the command: ./solver without_source 128 10000 0.05 0.02

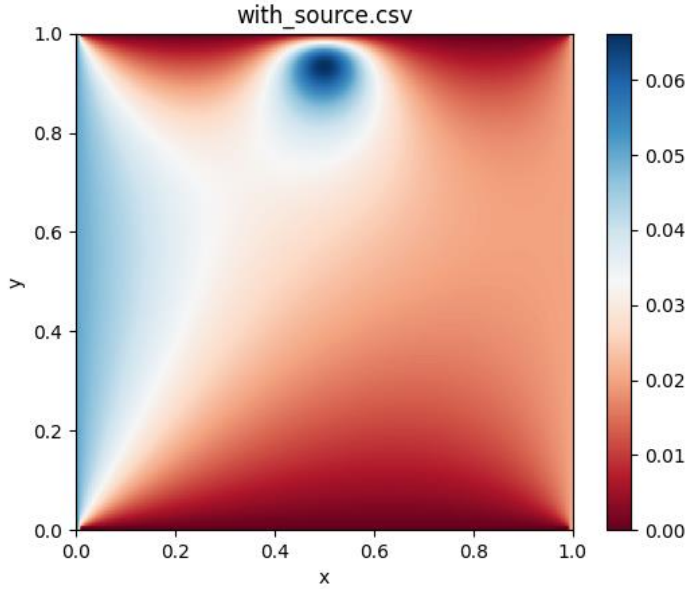

Figure 3: simulation with a source. Code executed with the command: ./solver with_source 128 10000 0.05 0.02 0.5 1.0 0.05

On these figures, we can clearly see the Gaussian source that has been added and the influence it has had on the code's output.

In particular, it can be seen that:

- The topmost and bottommost rows of the grid remain at 0 and the leftmost and rightmost columns are all at the constant value defined when the program was called. The boundary conditions are therefore met.
- The center of the task induced by the presence of a Gaussian source corresponds to the coordinates indicated when the program was called up.
- The simulation was successful and the new points were calculated successfully