Resolution of partial differential equations with finite difference method

Implementation in C++

Authors: Yvonne Alama Bronsard, Paolo Conti.

13 January 2020

1 Description of the project

The goal of our project is to solve numerically the Dirichlet's problem for the Poisson equation:

Find a regular solution u in $\Omega =]0,1[^2$ such that

$$-\Delta u = f, \quad \text{in } \Omega \tag{1}$$

and such that u is zero on the boundary of the square Ω.
The operator Δ is defined by $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.

To do this, we will utilise a finite difference method. We will calculate an approximation of the solution u at the points $x_{n,m}$, denoted by $u_{n,m}$, where the points $x_{n,m}$ are $(h_1\, n, h_2\, m)$ with $h_1 = 1/N$ and $h_2 = 1/M$, for $n = 0, ..., N$ and $m = 0, ..., M$.

The idea behind the finite difference method is to approach Δ by the finite difference operator Δ_d at the internal points $x_{n,m}$ (i.e. for $n \neq 0, N$ and $m \neq 0, M$). More precisely we use the centered difference of order two to approximate Δ, and obtain the following explicit scheme:

$$\forall (n,m) \in \{1, ..., N-1\} \times \{1, ..., M-1\}$$
$$\Delta_d u_{n,m} = \frac{u_{n-1,m} + u_{n+1,m} - 2u_{n,m}}{h_1^2} + \frac{u_{n,m-1} + u_{n,m+1} - 2u_{n,m}}{h_2^2} = f(x_{n,m})$$

Furthermore since we are solving an homogeneous Dirichlet boundary condition, we define $u_{n,m} = 0$ at the points of the boundary, namely for $n \in \{0, N\}$ or $m \in \{0, M\}$.

1.1 Construction of the sparse linear system

Solving our discretized problem is equivalent to the one of solving a sparse linear system. In order to write it as a system of equations we must first choose an ordering of the $(N-1)(M-1)$ discretized points $x_{n,m}$, $n = 1, ..., N-1$ and $m = 1, ..., M-1$. For

implementation reasons we choose to start the enumeration at zero, and we define our ordering as follows:

$$l\colon \{1,...,N-1\}\times\{1,...,M-1\}\to\{0,...,(N-1)(M-1)-1\}$$
$$(n,m)\mapsto(M-1-m)(N-1)+(n-1).$$

This corresponds to an ordering of the nodes from left to right, and from top to bottom. Let $U_{h1,h2}\in\ \mathbb{R}^{(N-1)(M-1)}$ be the vector of unknowns whose coefficients are given by:

$$u_k=(U_{h1,h2})_k=u_{n,m}\text{ where }k=l(n,m) \tag{2}$$

Similarly we will denote $b_{h1,h2}\in\mathbb{R}^{(N-1)(M-1)}$ the right hand side vector in our linear system where,

$$b_k=(b_{h1,h2})_k=f(x_{n,m})\text{ where }k=l(n,m) \tag{3}$$

Our ordering l, is a bijective function whose inverse is given by:

$$l^{-1}\colon \{0,...,(N-1)(M-1)-1\}\to\{1,...,N-1\}\times\{1,...,M-1\}$$
$$k\mapsto(k+1-\lfloor\frac{k}{N-1}\rfloor(N-1),\ M-1-\lfloor\frac{k}{N-1}\rfloor).$$

Using this chosen ordering we can write our problem in the matrix form $A_{h1,h2}U_{h1,h2}=b_{h1,h2}$ where $A_{h1,h2}\in\mathbb{R}^{(N-1)(M-1)\times(N-1)(M-1)}$. One can show that $A_{h1,h2}$ is a sparse symmetric invertible matrix and hence that there exists and unique solution to our discretized problem.[1] The aim of our project was then to use the UMFPACK library to solve our sparse linear system.

2 UMFPACK

UMFPACK is a library of C functions for solving systems of linear equations $Ax=b$ where A is an $n\times n$ matrix and b is an n-vector. Even if it will work with any non singular A, it is particularly effective in the situations where A is large and sparse, which is going to be our case.

UMFPACK expects the $n\times n$ coefficient matrix A of the system $Ax=b$ to be given in the CCS (*Compressed Column Storage*) format, which is a way to describe a matrix by three support vectors, A_x, A_i and A_p, defined as following.

Letting nz denote the number of non zero entries of the matrix A,

- A_x is the nz-vector collecting the values of the non zero entries.
- A_i is the nz-vector collecting the index of the rows associated to the non zero entries.
- A_p is the vector of length $n+1$ with the property that $A_p[i]$ gives the index in the vector A_x of where the matrix's column i begins.

We note that the rows and columns are indexed starting by convention at zero, and that to encode the matrix in CCS format, the matrix is scanned by columns from top to bottom and then from left to right. For a detailed example see [2].
We present how we encoded our sparse matrix in CCS in the following section.

2.1 Storing the sparse matrix in the CCS format

In this section we provide an overview of how we constructed our three vectors $Ax \in \mathbb{R}^{nz}$, $Ai \in \mathbb{R}^{nz}$, and $Ap \in \mathbb{R}^{(N-1)(M-1)+1}$, required by the UMFPACK library to invert our matrix. The first step is to construct a table, T, which given a node x_k, $k \in \{0, ..., (N-1)(M-1)-1\}$ returns the stencil associated with this node. For our numerical scheme, given any $(x, y) \in \Omega \setminus \partial\Omega$ its stencil is a five point stencil of the form:

$$\{(x, y-h_2), (x-h_1, y), (x, y), (x+h_1, y), (x, y+h_2)\}.$$

We can use our enumeration l to find the stencil of a point x_k for $k \in \{0, ..., (N-1)(M-1)-1\}$. Indeed, if $l^{-1}(k) = (n, m)$ where $n \neq 1$ or $N-1$, and $m \neq 1$ or $M-1$ then the stencil of the point x_k is:

$$\{x_{k-N+1}, x_{k-1}, x_k, x_{k+1}, x_{k+N-1}\}$$

We can now read from the array of stencils which elements in the kth row of our matrix are non-zero. In addition, we know the column indices corresponding to the non-zero elements of the kth row. By symmetry of A_{h_1,h_2}, this analogously tells us the row indices corresponding to the non-zero elements of the kth column. Indeed, for the case mentioned above we have the information that the kth column (or row) of A_{h_1,h_2} has five non zero elements at the $k-N+1$, $k-1$, k, $k+1$, and $k+N-1$ row (resp. column). Similarly, if $(n, m) \in \{(1,1), (1, M-1), (N-1, 1), (N-1, M-1)\}$, then the point x_k has a three point stencil which can be explicitly found by considering each of the four border cases separately. Once again this array of stencil states that the kth column has three non zero entries whose row indices are given by the indices of the elements in the array of stencil. Lastly, in the remaining case x_k has a four point stencil, implying that the kth column of A_{h_1,h_2} has four non-zero entries.

By looping through from $k = 0$ to $k = (N-1)(M-1)-1$ one can readily populate the vector Ai and Ax, where by simple calculation one finds that $nz = 5(M-1)(N-1)-2(N+M)+4$. Moreover, by finding the lengths of the stencil arrays at each iteration one can also easily populate the vector Ap.

With these three vectors in hand we could call the UMFPACK library and solve our discretized problem for different test functions f. Before presenting our results we will discuss the consistency error of our numerical scheme which will be of use for the analysis of our results.

2.2 Consistency Error

Throughout this section we assume $u \in C^4$. By Taylor-Lagrange formula up to order four, we have:

$$u(x_{i+1}, y_j) = u(x_i, y_j) + h_1 \frac{\partial u}{\partial x}(x_i, y_j) + \frac{h_1^2}{2} u_{xx}(x_i, y_j) + \frac{h_1^3}{6} u_{xxx}(x_i, y_j) + \frac{h_1^4}{12} u_{xxxx}(\tilde{x}_i, y_j),$$

where $\tilde{x}_i \in (x_i, x_{i+1})$. Similarly, we have:

$$u(x_{i-1}, y_j) = u(x_i, y_j) - h_1 \frac{\partial u}{\partial x}(x_i, y_j) + \frac{h_1^2}{2} u_{xx}(x_i, y_j) - \frac{h_1^3}{6} u_{xxx}(x_i, y_j) + \frac{h_1^4}{12} u_{xxxx}(\hat{x}_i, y_j),$$

with $\hat{x}_i \in (x_{i-1}, x_i)$ and

$$u(x_i, y_{j+1}) = u(x_i, y_j) + h_2 \frac{\partial u}{\partial y}(x_i, y_j) + \frac{h_2^2}{2} u_{yy}(x_i, y_j) + \frac{h_1^3}{6} u_{yyy}(x_i, y_j) + \frac{h_2^4}{12} u_{yyyy}(x_i, \tilde{y}_j),$$

and

$$u(x_i, y_{j-1}) = u(x_i, y_j) - h_2 \frac{\partial u}{\partial x}(x_i, y_j) + \frac{h_2^2}{2} u_{yy}(x_i, y_j) - \frac{h_1^3}{6} u_{yyy}(x_i, y_j) + \frac{h_2^4}{12} u_{yyyy}(x_i, \hat{y}_j).$$

This means that using the intermediate value theorem, we have for some (x_i', y_j') with $x_i' \in (\hat{x}_i, \tilde{x}_i), y_j' \in (\hat{y}_j, \tilde{y}_j)$

$$u(x_{i+1}, y_j) + u(x_{i-1}, y_j) = 2u(x_i, y_j) + h_1^2 u_{xx}(x_i, y_j) + \frac{h_1^4}{6} u_{xxxx}(x_i', y_j),$$

and

$$u(x_i, y_{j+1}) + u(x_i, y_{j-1}) = 2u(x_i, y_j) + h_2^2 u_{yy}(x_i, y_j) + \frac{h_2^4}{6} u_{yyyy}(x_i, y_j').$$

All together, this yields:

$$\Delta_d u = u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j) \tag{4}$$

$$= \frac{u(x_{i+1}, y_j) + u(x_{i-1}, y_j) - 2u(x_i, y_j)}{h_1^2} + \frac{u(x_i, y_{j+1}) + u(x_i, y_{j-1}) - 2u(x_i, y_j)}{h_2^2} \tag{5}$$

$$+ \frac{h_1^2}{6} u_{xxxx}(x_i', y_j) + \frac{h_2^2}{6} u_{yyyy}(x_i, y_j') \tag{6}$$

Hence, if our analytical solution has enough regularity, namely belongs to C^4, the consistency error of numerical scheme is of order 2. In addition, if u is of degree three or less the consistency error is zero which implies that our numerical solution coincides with our analytical one.

We now have the tools necessary to discuss our results, which is done in the next section.

3 Results

In order to test our program, we constructed some solutions to the problem (1), by choosing an analytic function ϕ such that ϕ is zero on the boundary of Ω and we chose the right hand side of the equation, f, by computing $f = -\Delta\phi$. We start our analysis by considering analytical solutions that have enough regularity, and verify that our numerical observations coincides with our theoretical ones. Then, we observe and comment the different results we obtain when our analytical solution is less regular.

3.1 First solution: $\phi(x, y) = sin(\pi x)sin(\pi y) \in C^\infty(\Omega)$

Let us first consider the function:

$$\phi(x, y) = sin(\pi x)sin(\pi y), \quad (x, y) \in [0, 1]^2 \tag{7}$$

which satisfies the Dirichlet boundary conditions. With this choice of ϕ, we can calculate f as follows:

$$f(x, y) = \Delta\phi(x, y) = -2\pi^2 sin(\pi x)sin(\pi y)$$

and conclude that ϕ is a solution to Poisson's equation (1) with this choice of f. By letting $b_k = f(x_k) = f(x_{n,m})$ where $l^{-1}(k) = (n, m)$, we can construct the right hand side vector b_{h_1,h_2} of our system of equations. Together with the CCS format of our matrix one can feed it to the UMFPACK library to obtain the numerical solution.

In order to validate our program, we checked visually if the analytical solution corresponded to the numerical one that we have found, by plotting with *gnuplot* the two functions on the same graph:

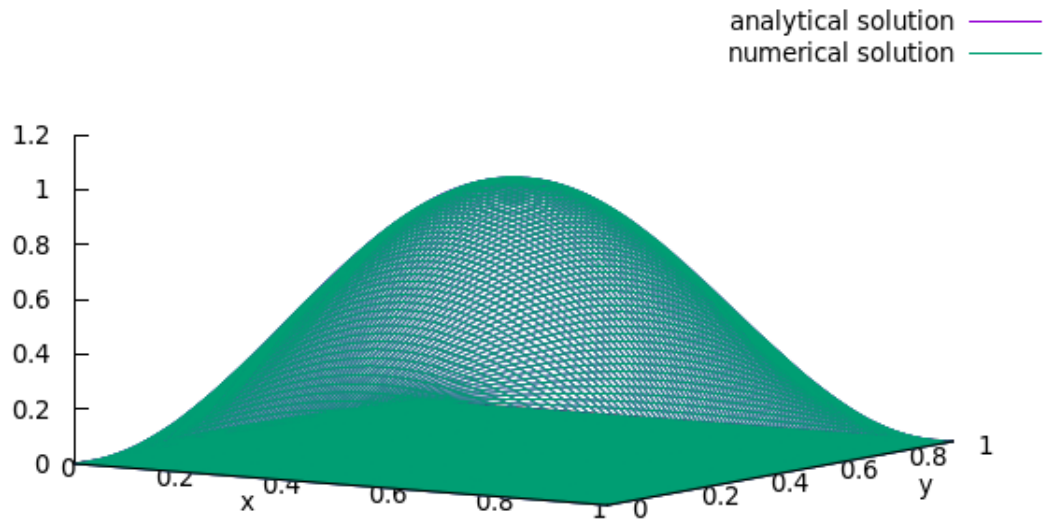


Figure 1: Plot of $\phi(x, y) = sin(\pi x)sin(\pi y)$ and of the numerical solution, for $N = M = 100$

From this graphical point of view, the numerical solution seems to be accurate, which is what we expected since $\phi \in C^\infty(\Omega)$. Indeed, since we have that in particular $\phi \in C^4(\Omega)$, it follows that the consistency error of our scheme is of order two (see section 2.2) and hence that our method converges in order two [1].

To confirm these results we plotted with *gnuplot* a graph of the discret L2 error with respect to $h = 1/N = 1/M$ in a logarithmic scale. We included the logarithmic graph of h and h^2, so we could visualize the order of convergence.

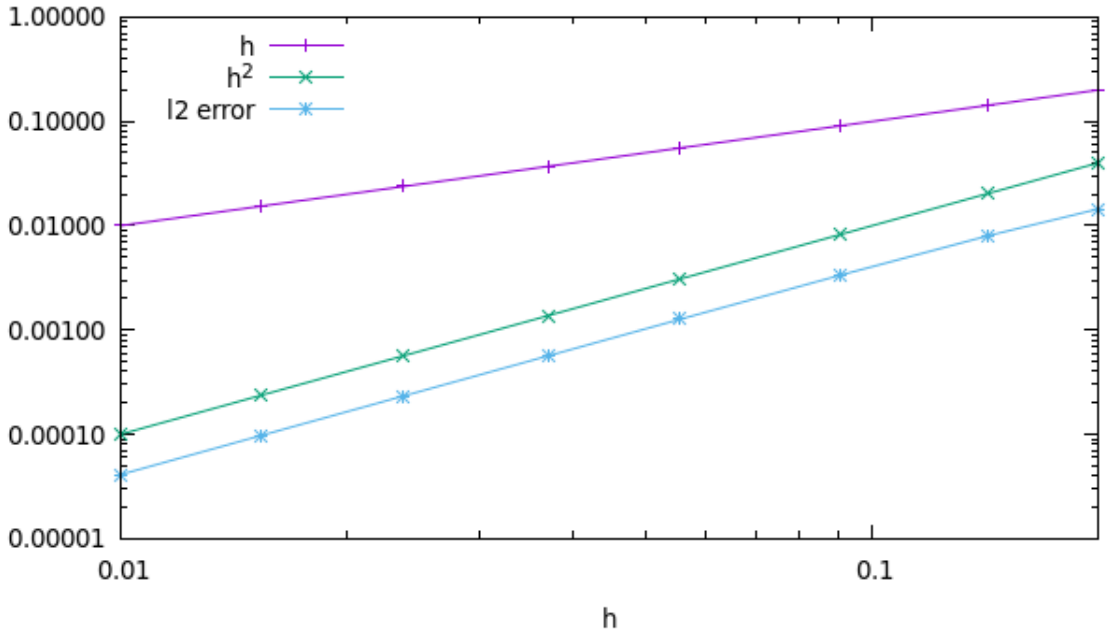


Figure 2: Plot of the discrete L_2 error for $N = M = \{5, 7, 11, 18, 27, 42, 65, 100\}$.

The graph of the error confirms what we expected from the theory: an asymptotic behaviour of the error of order h^2.

Finally, we computed the duration of our program and plotted it with respect to N, considering $M = N$, in logarithmic scales. In this way we could get the complexity of our program, which is of order $O(N^2)$.

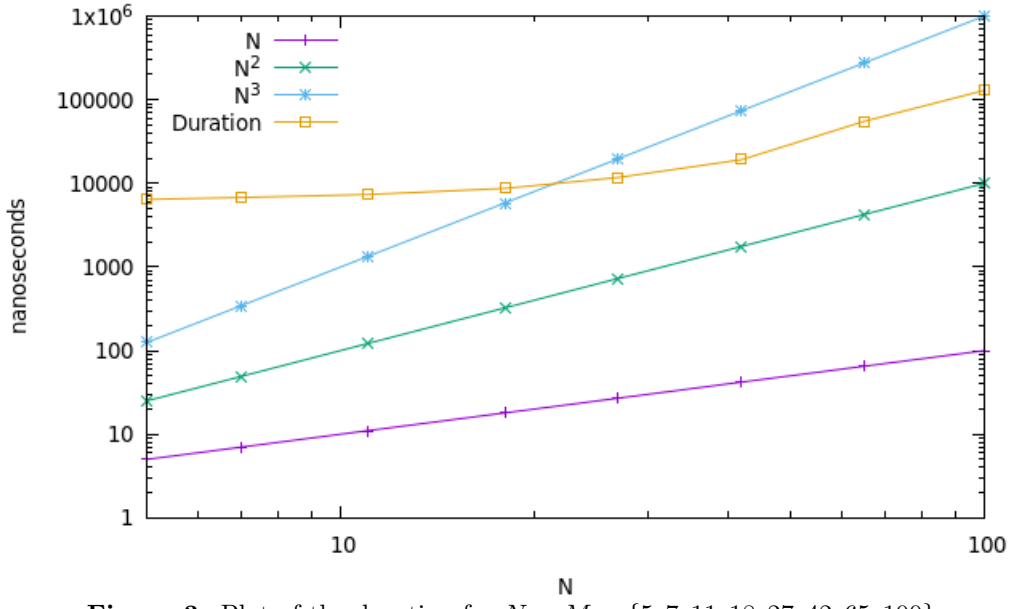


Figure 3: Plot of the duration for $N = M = \{5, 7, 11, 18, 27, 42, 65, 100\}$.

3.2 Second solution: $\phi(x, y) = xy(x-1)(y-1) \in \mathbb{P}_2(\Omega)$

Afterwards we considered the following function:

$$\phi(x, y) = xy(x-1)(y-1), \qquad (x, y) \in [0, 1]^2 \tag{8}$$

And we repeated the same procedure as before: we plotted the graphs of the analytical solution and numerical solution, of the discrete $l2-$error and of the duration of our program.

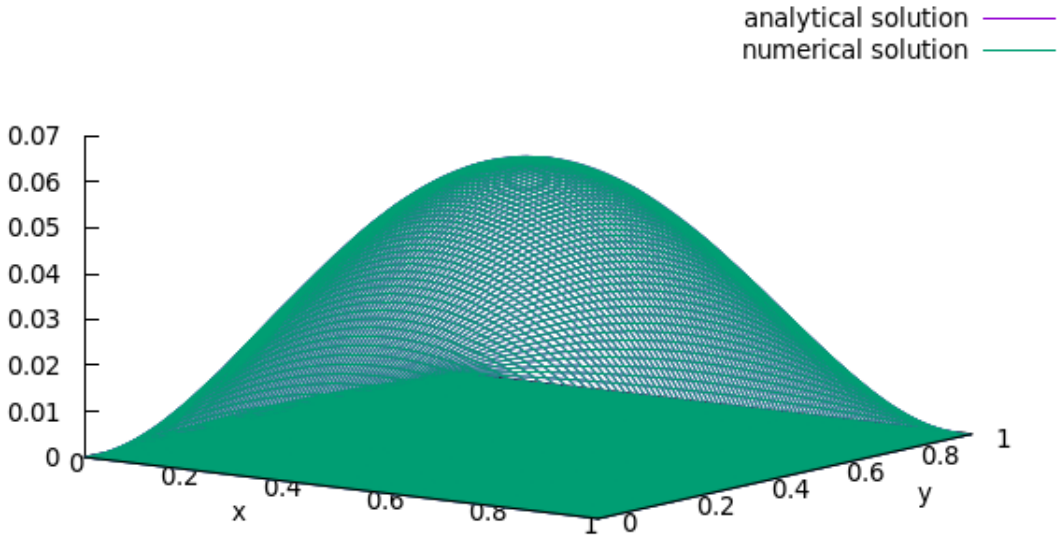


Figure 4: Plot of $\phi(x, y) = xy(x-1)(y-1)$ and of the numerical solution, for $N = M = 100$

Again, since our analytical solution has enough regularity our numerical scheme converges to our analytical solution, which is what we observe graphically in the above image.

However, unlike in the previous case, the function ϕ that we are considering is a polynomial of degree two in x and y. Hence, as mentioned in the previous section *"Consistency error"*, the consistency error is zero, so we are expecting our program to give as output a numerical solution which fits perfectly the analytical one, with no error.
This is indeed what we obtained: our errors are of the order 10^{-17} which corresponds to the zero of the computer.

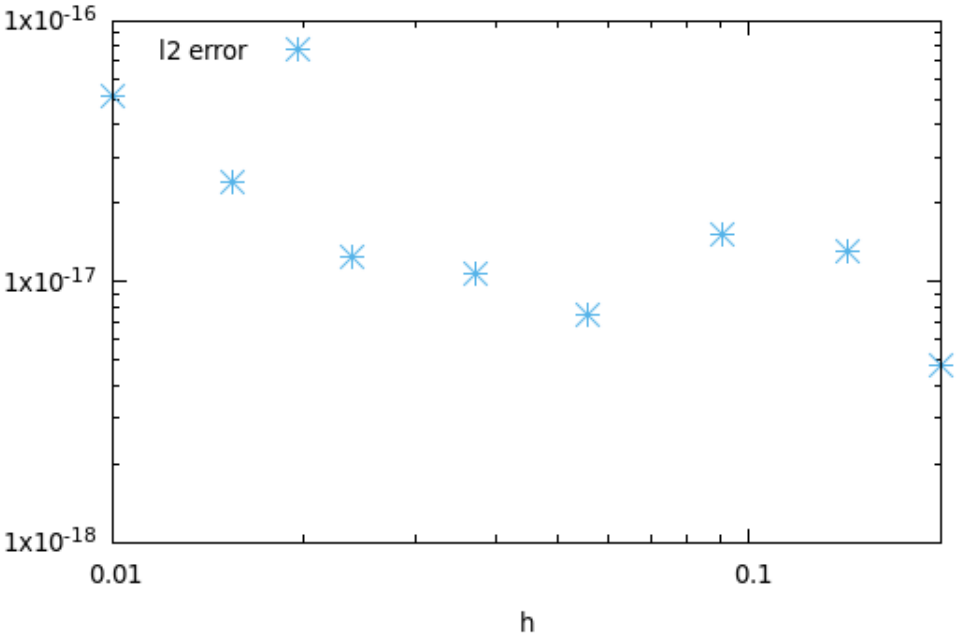
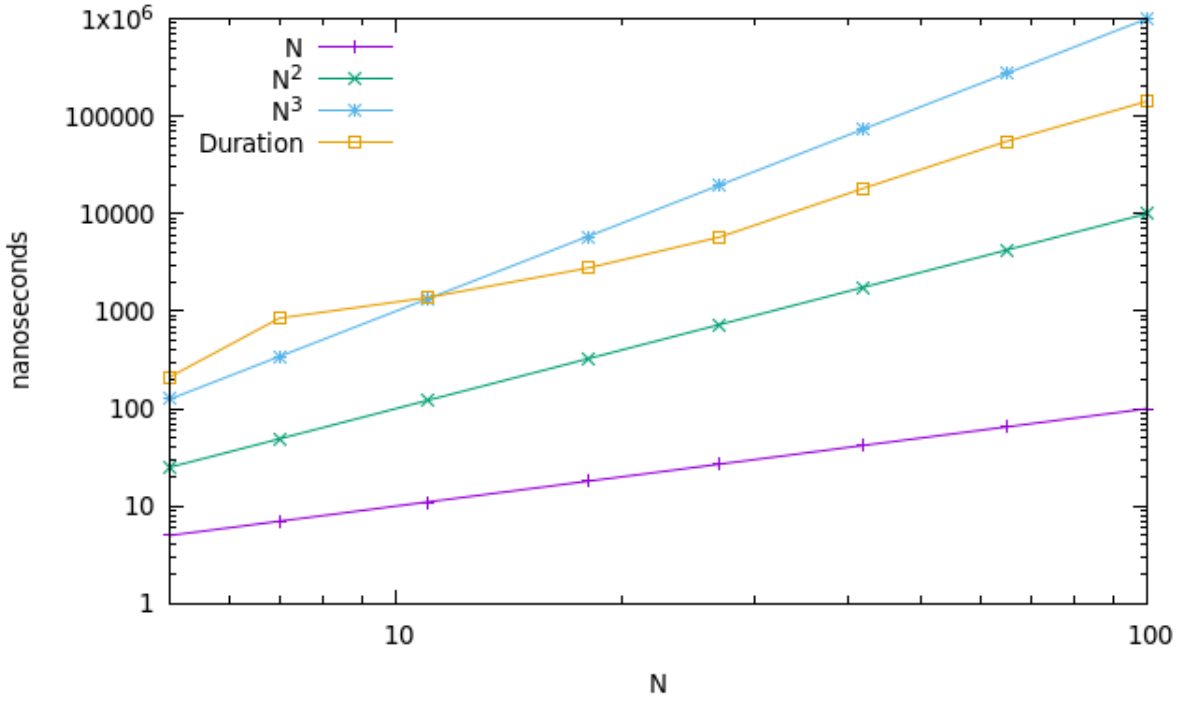


Figure 5: Plot of the discrete L_2 error for $N = M = \{5, 7, 11, 18, 27, 42, 65, 100\}$. The L_2 error is zero since there is no consistency error.

Finally, from the following plot of the duration, we confirm the same result as before: an asymptotic behaviour of order N^2.



3.3 Third solution: $\phi(x,y) = exp\left[-1000(x-0.5)^2 - 1000(y-0.5)^2\right]$

Then we wished to test our program with analytical solutions that are not as well-behaved as the previous ones. We considered the following manufactured solution:

$$\phi(x,y) = exp\left[-1000(x-0.5)^2 - 1000(y-0.5)^2\right] \ \in C^{\infty}(\Omega) \tag{9}$$

First, let us note that this solution does not fully satisfying the Dirichlet boundary condition. Nonetheless, the rate of convergence near the boundary to zero is so fast that it is numerically approximated to zero without really compromising the regularity of the function.
We consider this type of function 'ill-behaved' due to its *peak* in the center, i.e. there is a dramatical change in the function at a very localised area which leads to a very large Laplacian in that area. We were intrigued to see whether a solution with such a peak would cause problems when wanting to numerically approximate using our code. We first, graph the analytical and numerical solutions when $N = M = 100$:

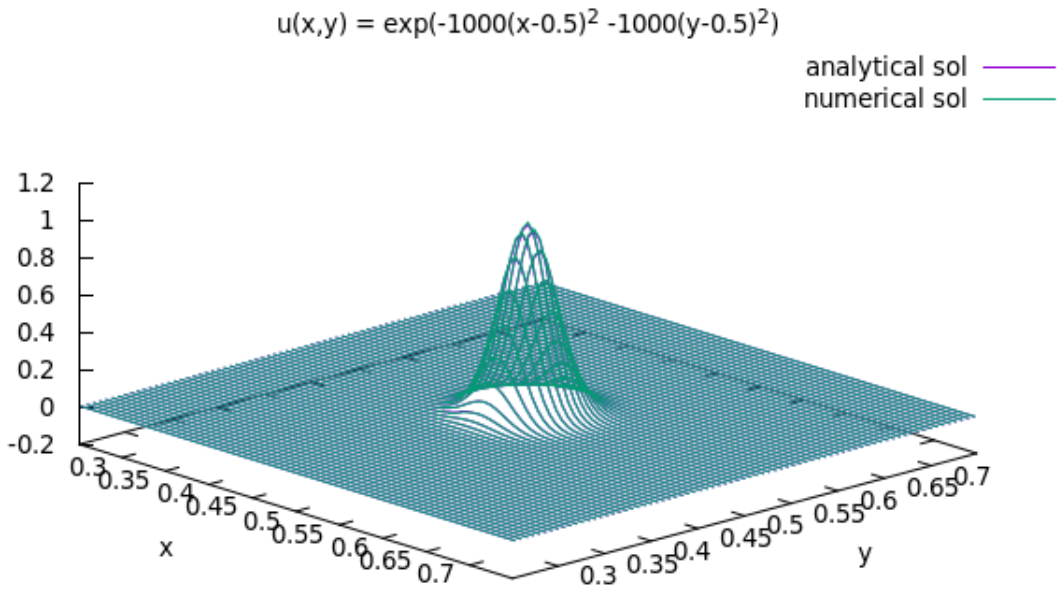
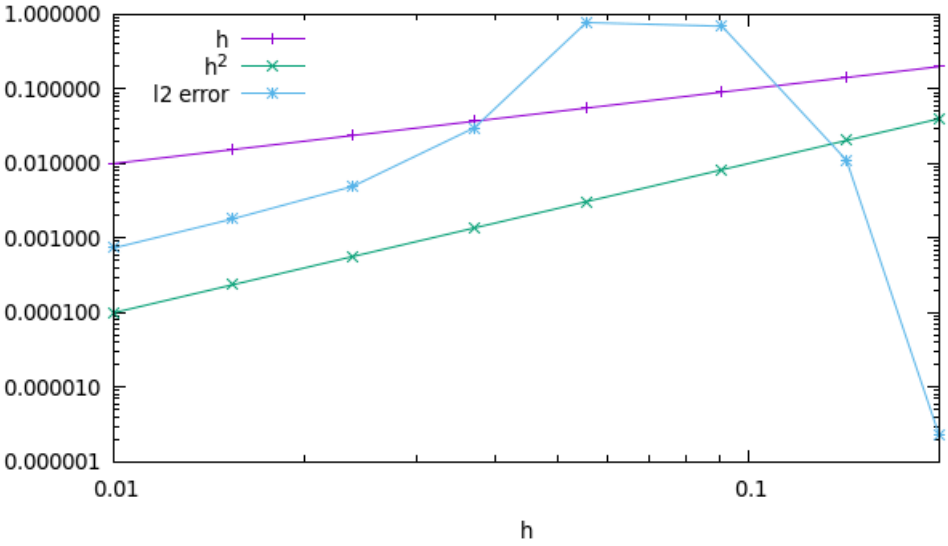


Figure 6: Plot of $\phi(x,y) = exp\left[-1000(x-0.5)^2 - 1000(y-0.5)^2\right]$ and of the numerical solution, for $N = M = 100$

We see graphically that for large N and M our numerical solution is accurate and the presence of the peak does not affect the convergence of our numerical solution to the analytical one, since our numerical solution cannot be distinguished from the analytical one.
However by looking at the graph of the error, we realize that there is a perturbation with respect to the regular behaviour of the error we had in the case of $\phi(x,y) = sin(\pi x)sin(\pi y)$.



Since our solution is smooth, it satisfies the regularity requirements of our problem, which implies that the consistency error is of order two, and hence in the limit as $N, M \to \infty$ our numerical solution converges to the analytical one (in order two). However, when N, M are small, the number of discretised points located in the area where the peak is held is small, which can lead to large errors and an overall behaviour which is far from the asymptotic one. Indeed, when $N = 5$ we observe that the error is close to zero since there are very few points of discretisation in the area where the peak is held. We can see in the case of $N = M = 18$, that the numerical solution is not corresponding to the analytical one:

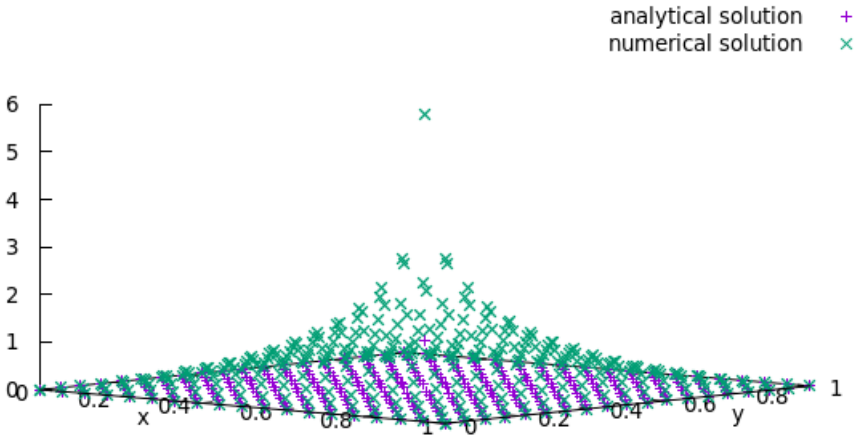
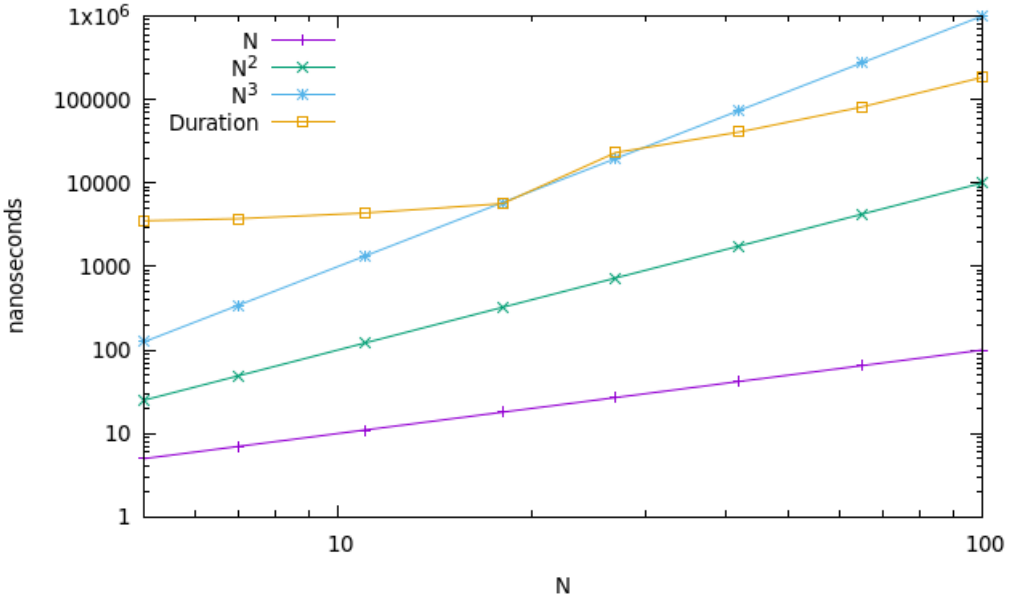


Figure 7: Plot of $\phi(x, y) = exp\left[-1000(x - 0.5)^2 - 1000(y - 0.5)^2\right]$ and of the numerical solution, for $N = M = 18$

The numerical solution presents a peak which is much higher and wider than the one of the analytical solution, leading to a very big error.

Hence, the presence of the peak affects enormously the convergence of our method in small dimensions, but, since our solution has enough regularity, by increasing N, M, we will recover the convergence and obtain an error of order h^2.

We present once again the logarithmic plot of the duration with respect to N, considering $M = N$. As previously observed we have a behaviour of $O(N^2)$.



3.4 Less regular functions:

Lastly, we wanted to analyze the behaviour of our program with less regular functions, namely:

$$\phi(x, y) = x\, y^{0.6}(x - 1)(y - 1) \tag{10}$$

$$\phi(x, y) = x\, y^{0.2}(x - 1)(y - 1) \tag{11}$$

$$\phi(x, y) = 10\, x\, y^{0.01}(x - 1)(y - 1) \tag{12}$$

These functions are such that the square of their Laplacian is not anymore integrable on Ω, because the second derivative in y diverges when $x \to 0$, therefore they are only functions in $H_0^1(\Omega)$. They are actually *weak solutions* to our problem (1), which means that they are not regular enough to guarantee the convergence of our numerical solution to the analytical one. Anyway, we wanted to test our program with this kind of functions to see what output it gave us.

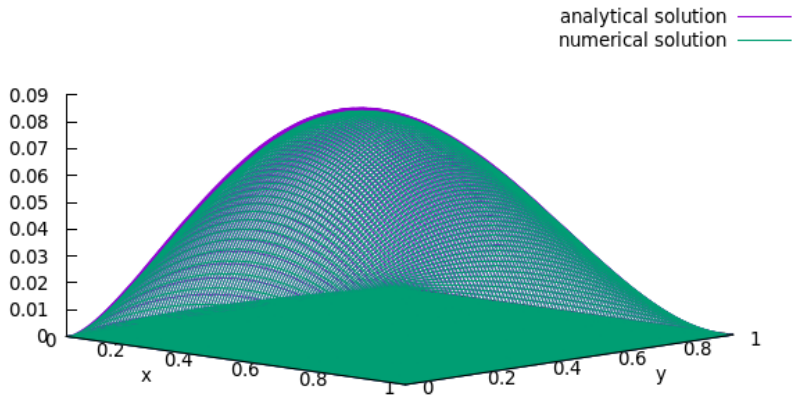


Figure 8: Plot of $\phi(x, y) = x\, y^{0.6}(x-1)(y-1)$ and of the numerical solution, for $N = M = 100$

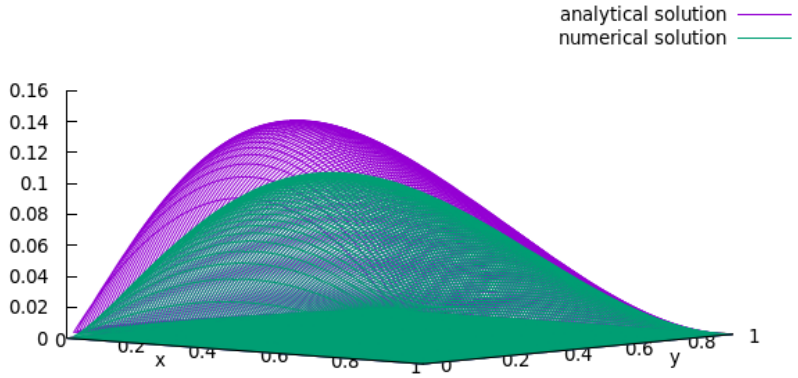


Figure 9: Plot of $\phi(x, y) = x\, y^{0.2}(x-1)(y-1)$ and of the numerical solution, for $N = M = 100$

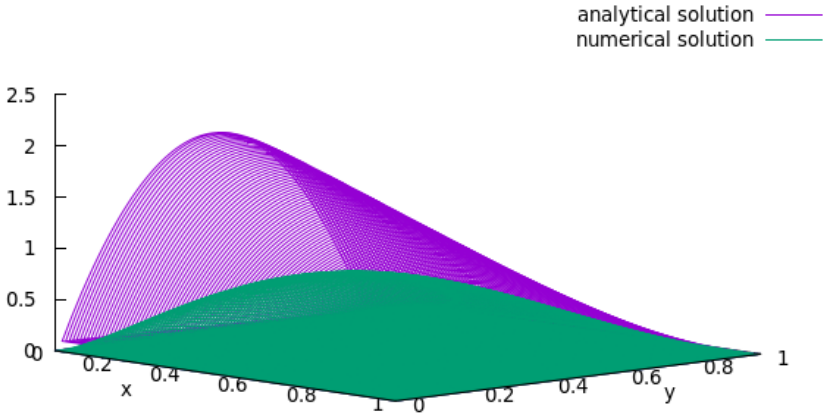
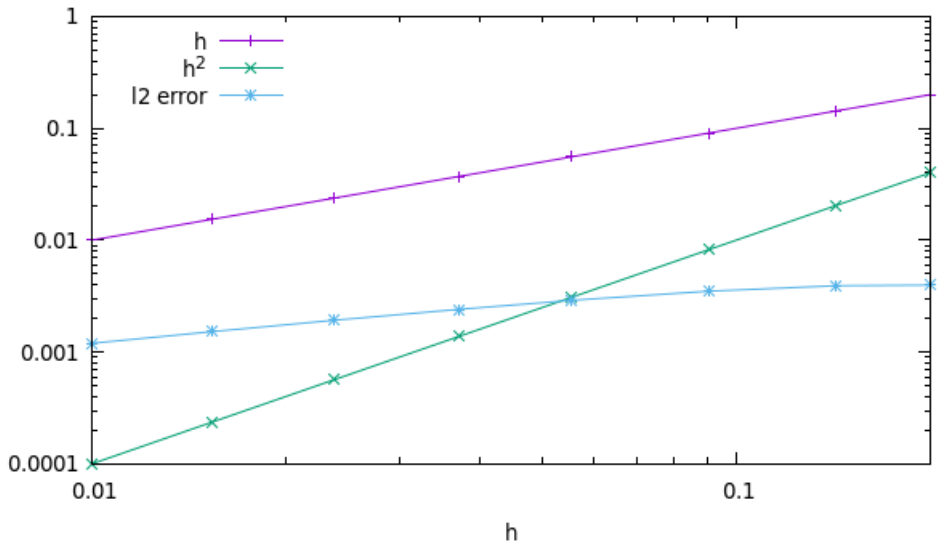


Figure 10: Plot of $\phi(x, y) = 10\, x\, y^{0.01}(x-1)(y-1)$ and of the numerical solution, for $N = M = 100$

We notice that we still get a good approximation of the analytical solution with function (10) and we suppose it is because the integral on Ω of the square of the Laplacian is diverging with a small order. The more we increase the order with which the integral diverges, the more our output moves away from the analytical solution considered. Since, taking into account the function (10), it seems like we had a quite accurate numerical solution, we computed, only for this case, the logarithmic graph of the error:



Since the function is not regular enough for our numerical model, we couldn't expect the $O(h^2)$ behaviour of error that we had in the previous cases.

With regard to the duration of our program, we didn't expect anything new, because duration is not affected by the regularity of the input function. The following logarithmic graph of the duration with respect to N, considering $M = N$ confirms our assumptions:

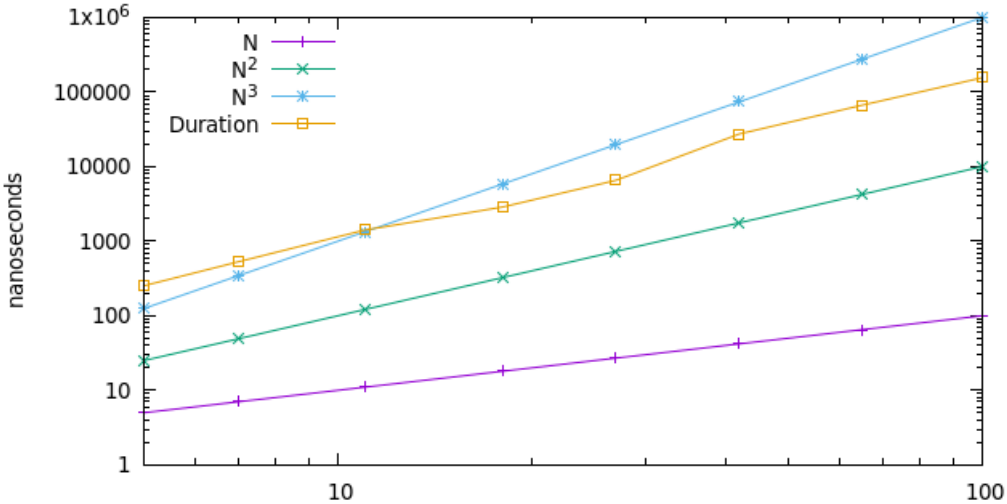


Figure 11: Plot of the duration for $N = M = \{5, 7, 11, 18, 27, 42, 65, 100\}$.

4 Appendix

4.1 Discrete L2 norm

For our analysis we have chosen to use discrete L^2-norm to study the convergence of our method. We note that using the Euclidean norm would not be a good choice since the Euclidean norm tends to ∞ as $h \to 0$ because the number of terms we are summing tend to infinity. This is why we must introduce the discrete L^2-norm. The discrete L^2-norm in 1-D is defined as follows for a vector $V = (v_0, \ldots, v_{N+1})$:

$$\|V\|_{2\Delta}^2 = h\left(\frac{1}{2}v_0^2 + \sum_{n=1}^{N} v_n^2 + \frac{1}{2}v_{N+1}^2\right).$$

In our problem, the boundary conditions are homogeneous Dirichlet, so $v_0 = 0$ and $v_{N+1} = 0$. Furthermore, we are working in 2-D, so using our enumeration our discrete L^2-norm will be:

$$\|U\|_{2\Delta}^2 = h_1 h_2 \left(\sum_{n=1}^{N}\sum_{m=1}^{M} u_{n,m}^2\right) = h_1 h_2 \left(\sum_{k=0}^{(N-1)(M-1)-1} u_k^2\right), \tag{13}$$

where $U = (u_0, \ldots, u_{(N-1)(M-1)-1}) \in \mathbb{R}^{(N-1)(M-1)}$.
For a more detailed explanation of the discrete L^2-norm see ([1], pg. 40). Lastly, we note that in finite dimensions all norms are equivalent up to a constant, hence the choice of another norm in $\mathbb{R}^n$ would also have worked. If the user wishes to graph the error in infinity norm, we have coded a function which enables this possibility. However, we have chosen to plot the errors with the discrete L^2 norm rather than the L^∞ since the L^2 norm "averages" the errors unlike the L^∞ norm which is quite "sensitive" and concentrates on just the few points where the approximation may not be good. (Convergence in the L^2-norm means that in some "average sense" the approximate solution is close to the real solution and so convergence in L^2 is in that sense easier to establish than that in L^∞.)

References

[1] Polycopié of Hervé LeDret, for the class 'Bases des méthodes numériques'. (pg. 49) `https://moodle-sciences.upmc.fr/moodle-2019/pluginfile.php/554319/mod_resource/content/1/chapitre2.pdf`

[2] Rouben Rostamian, *Programming Projects in C for Students of Engineering, Science, and Mathematics*. SIAM, 2014. (Chapter 11 and 12)

[3] Alfio Quarteroni *Numerical Models for Differential Problems*, Springer, second edition. (Chapter 3)

[4] Xu, Shi, Yin, *Deep Learning for Partial Differential Equations*, Deep Learning, Winter 2018, Stanford University, CA.