Scientific Computing

Project 1 - Report

Lucia Filippozzi, student ID 223447 May, 2021

This is the report of the second project of Scientific Computing and it summarizes the investigations and reports on the results obtained during the implementation of a multigrid method for the numerical solution of the elliptic boundary value problem in $\mathbb{R}^d$, $d = 1,\ 2$: Let $\Omega = (-1, 1)^d \subset \mathbb{R}^d$. Given the functions $c, f \in C^0(\Omega)$ find a function $u \in C^2(\Omega)$ such that

$$\begin{cases} \Delta u + cu = f & in\ \Omega \\ u = 0 & on\ \Omega \end{cases}$$

where $c(x) = |x|^2$.

In particular, for both 1dimensional and 2 dimensional case, a very brief description of the performed methods is provided. After that the results of the requested investigations are reported as well as some explanatory and useful plots.
Subsequently, for each case, it will be provided - as requested - a plot of the approximated solution with an estimate of the maximum over the closure of the domain.

Multigrid in 1d

Let $\Omega = (-1, 1) \subset \mathbb{R}$. Given the continuos functions $c(x) = |x|^2$ and $f(x) = \left(x - \frac{1}{2}\right)^4$, the aim is to find a function $u \in C^2(\Omega)$ such that:

$$\begin{cases} \Delta u + cu = f & in\ \Omega \\ u = 0 & on\ \Omega \end{cases} \tag{1}$$

In particular we will provide different methods to find an approximated solution of u in order to finally perform a final full multigrid method.

For this purpose let $x_0 = -1 < x_1 < \ldots < x_{N-1} < 1 = x_N$ be a uniform partitioning of the interval $[-1, 1]$ with mesh size $h = 2/N$ with $N = 2^l,\ l \geq 1$. A uniform grid can be therefore defined as $\Omega_l = \{x_0,\ x_1, \ldots,\ x_N\}$ and the standard finite difference approximation to the given boundary value problem can be represented as usual, as a linear system:

$$A_h u_h = f_h$$

where $A_h \in \mathbb{R}^{(N-1)} \times \mathbb{R}^{(N-1)}$.
We also consider the sequence of grids $\Omega_1, \ldots, \Omega_l$, where Ω_1 is the trivial grid with $x = 0$ as the only interior node (that is $\Omega_1 = \{-1,\ 0,\ 1\}$).

Investigation

The first method that has been implemented is a **weighted Jacobi** with $\omega = 2/3$ as weight. After that a **two-grid correction scheme** has been implemented, where instead of solving the residual equation on Ω_{2h}, it performs ten weighted Jacobi steps on the coarse grid.
In both cases the iteration, starting from $u_h^{(0)} = 0$, stops when the pseudo-residual satisfies $|s_h^{(k+1)}|_\infty = |u_h^{(k+1)} - u_h^{(k)}|_\infty < \text{tol} = 10^{-8}$.
In this setting an investigation on the performance of these two methods applied to the linear system (1) for the stated model problem, as h decreases, has been made. In particular we are interested on the effect on the CPU time and on the number of iterations needed for the methods; for this investigation we have considered a sequence of grid sizes which includes $N = 2^l$, with $j = 3, \ldots, 7$.

The obtained results are summarized and compared in the following table:

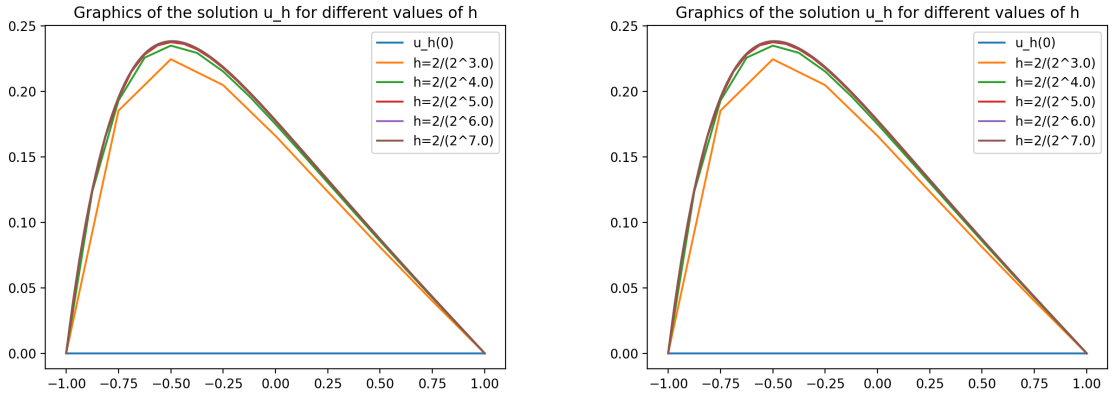
l	N	h	Jacobi iter.	Jacobi time	Two-grids Iter.	Two-grid time
3	8	0.25	255	0.0102s	9	0.0041 s
4	16	0.125	925	0.1011s	24	0.0348 s
5	32	0.0625	3298	0.5513s	80	0.3361 s
6	64	0.03125	11558	4.7057s	277	0.8937 s
7	128	0.015625	39679	26.4714s	946	6.2421 s

Table 1: Investigation on Jacobi methods as h decreases

We can easily notice that the two-grid correction scheme provides quite an improvement

compared to the classical Jacobi method. However, as we expected, the number of iterations needed to reach convergence and the CPU time increases with N, for both method. This is due to the fact that for the classical iterative methods (such as Jacobi, Gauss-Seidel, SOR, ...) as the mesh size h decreases, the number of iterations (and consequentially the time) needed to reduce the error by a given factor will increase because all these methods suffer from a very slow convergence once h is getting smaller.
To overcome this problem, that is the impossibility to devise a method that produces the desired solution with complexity $\mathcal{O}(N)$, one can perform **Multigrid methods** instead of the classical iteratives ones.

Before moving on on the next investigation about different methods based on the multigrid, let's have a look at the plot of the solution obtained with the previous methods.



(a) Jacobi

(b) Two Grid Correction

Figure 1: Graphics of the initial guess $u_h^{(0)}$ and the obtained approximated solutions u_h

As said before, we can obtain simulations with better performances if we compute the solutions using multigrid methods.

For this purpose, the first function that has been implemented iterates **V-cycles** for the linear system (1), using for each of them one weighted Jacobi step with weight $\omega = 2/3$ as pre- and post- smoothing. On the coarsest grid Ω_1 uses an exact solve which is given by the non zero element:

$$u_h[1] = A_h[1]^{-1} \cdot f_h[1] \;=\; \frac{h^2}{4 + c_h[1] \cdot h^2} \cdot f_h[1] \;=\; \frac{h^2}{4} \cdot f_h[1] \;=\; \frac{f_h[1]}{N^2}$$

Moreover, at each step it computes and returns the pseudo-residual of the post-smoothing Jacobi step, unless it is called on the coarsest grid Ω_1, when it returns zero. Here the iteration, starting from a zero initial guess as usual, stops when the pseudo-residual satisfies $|s_h^{(k+1)}|_\infty < \text{ tol } = 10^{-8}$ for the post-smoothing step on the finest grid.

To test the performances an investigation on the number of iterations needed for the implemented V-cycle and the effect on the CPU time as h decreases, has been made. After that we implemented a function that performs a **full multigrid**, utilizing the V-cycle implementation to perform a single V-cycle step in each grid, for the linear system (1). This function returns

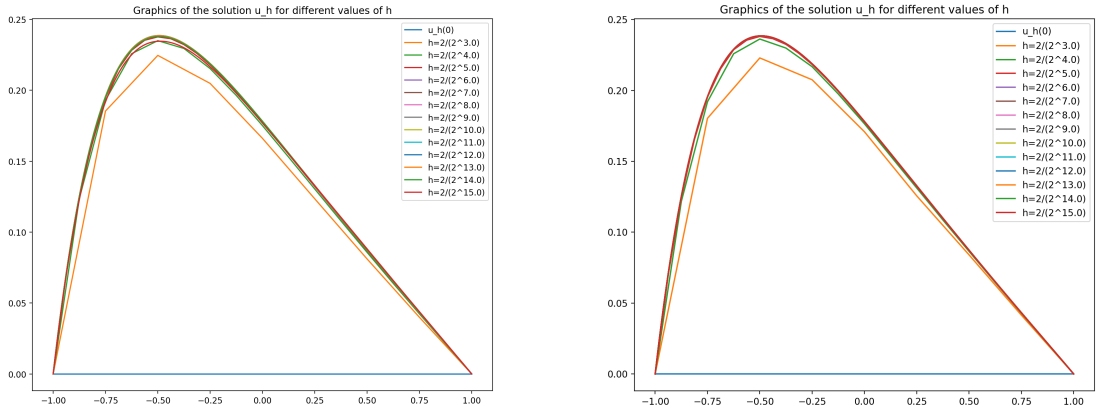
the result of the call to the V-cycle step on the finest grid. Also in this case we investigates the CPU time needed, and the sizes of the final pseudo-residual $|s_h^{(k+1)}|_\infty$ as well as the size of the residual $|r_h|_\infty = |f_h - A_h u_h|_\infty$ after one full multigrid step.
For the sequence of the grid sizes we have taken $N = 2^l,\ l = 3, 4, \dots 15$.
We can compare and summarize the obtained results in the following table. We expect that now, using multigrid, the developed methods have convergence rate which does not heavily depend on h:

l	N	V-cycle iter.	V-cycle time	FullMG s_h.	FullMG r_h	FullMG time
3	8	9	0.0059 s	7.978e-03	1.358e-01	4.752e-04 s
4	16	10	0.0058 s	2.920e-03	2.409e-01	4.704e-03 s
5	32	9	0.0091 s	1.147e-03	2.409e-01	4.704e-03 s
6	64	9	0.0172 s	3.828e-04	3.690e-01	6.577e-03 s
7	128	8	0.0286 s	1.141e-04	4.637e-01	1.057e-02 s
8	256	8	0.0520 s	3.191e-05	5.212e-01	1.816e-02 s
9	512	7	0.0881 s	8.645e-06	5.528e-01	3.123e-02 s
10	1024	7	0.1692 s	2.249e-06	5.694e-01	5.708e-02 s
11	2048	6	0.2876 s	5.736e-07	5.786e-01	1.074e-01 s
12	4096	5	0.4881 s	1.448e-07	5.861e-01	2.046e-01 s
13	8192	4	0.7597 s	3.639e-08	5.899e-01	4.152e-01 s
14	16384	3	1.1727 s	9.120e-09	5.918e-01	1.079 s
15	32768	2	1.5004 s	2.283e-09	5.928e-01	1.681 s

Table 2: Investigation on V-cycle and Full Multigrid methods as h decreases

We can notice that the convergence is faster than the one of Jacobi method, as we expected, and the committed errors (that can be visualized in the residuals) are quite small.
The convergence's improvement, compared to the one before, is clearly visible already in the first method "V-cycle": this is due to the fact that using the multigrid method the number of iterates (and consequentially the CPU time) does not deteriorate as h decreases.
The CPU performances enhance even more in the Full multigrid method which can be saw as a final refining of the multigrid in the form of nested iterations. One can indeed notice that the time spent in order to achieve convergence is very small compared to the previous times.

Also in this case some plots are provided in order to check that the implemented methods work actually fine:



(a) V-steps

(b) Full multigrid

Figure 2: Graphics of the initial guess $u_h^{(0)}$ and the obtained approximated solutions u_h

At this point we investigate the number of **additional V-cycles** needed, and the effect on the total CPU time, by using the full multigrid method to provide a (more accurate) initial guess for the V-cycle multigrid method. For doing that, after calling the function which performs the Full multigrid method, we use the produced approximation u_h as the initial guess for the V-cycle multigrid method.

The stopping criterion and the sequence of grid sizes are the same as the previous ones.

The obtained results are visualized in a compact way, together with the previous ones, in the following table (in which for simplicity the columns of l and N have been dropped):

V-cycle it.	V-cycle time	FullMG s_h.	FullMG r_h	FullMG time	Iter.	CPU time
9	0.0059 s	7.978e-03	1.358e-01	4.752e-04 s	7	0.0019 s
10	0.0058 s	2.920e-03	2.409e-01	4.704e-03 s	7	0.0064 s
9	0.0091 s	1.147e-03	2.409e-01	4.704e-03 s	6	0.0073 s
9	0.0172 s	3.828e-04	3.690e-01	6.577e-03 s	6	0.0137 s
8	0.0286 s	1.141e-04	4.637e-01	1.057e-02 s	5	0.0250 s
8	0.0520 s	3.191e-05	5.212e-01	1.816e-02 s	4	0.0425 s
7	0.0881 s	8.645e-06	5.528e-01	3.123e-02 s	4	0.0746 s
7	0.1692 s	2.249e-06	5.694e-01	5.708e-02 s	3	0.1222 s
6	0.2876 s	5.736e-07	5.786e-01	1.074e-01 s	2	0.1932 s
5	0.4881 s	1.448e-07	5.861e-01	2.046e-01 s	2	0.3878 s
4	0.7597 s	3.639e-08	5.899e-01	4.152e-01 s	0	0.4175 s
3	1.1727 s	9.120e-09	5.918e-01	1.079 s	0	0.8307 s
2	1.5004 s	2.283e-09	5.928e-01	1.681 s	0	1.6906 s

Table 3: Investigation on V-cycle and Full Multigrid methods as h decreases

The second-last column refers to the number of added V-cycle iterations and the one titled "CPU time" includes the time spent performing the full multigrid method and all the successive V cycles.

From this table one can notice that, providing the V-cycles method with a more precise initial guess more precise than $u_h^{(0)} \equiv 0$ quite improves the performances: for a same value of N (and therefore h) the iterations needed to achieve convergence with the last method are less than the

ones needed starting with $u_h^{(0)} \equiv 0$. Moreover, in general we record an improvement in the time, too.

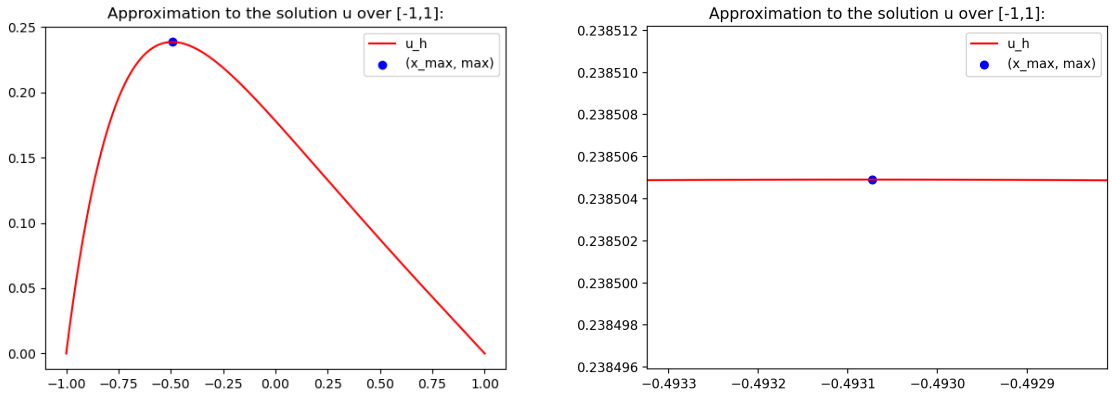
In particular, for $l \geq 13$, the full multigrid method is enough because with just one call of this method the required tolerance is reached: indeed 0 additional iterations of V-cycle are needed!

Approximation of the maximum

After all this investigation we are able to provide an accurate approximation of the solution u and therefore we are also able to compute a precise estimate of the $\max_{x\in[-1,1]} u(x)$.
Let u_h be the final approximated solution, obtained with a V-cycle multigrid method that uses as initial guess the one provided by a full multigrid method. Therefore we can estimate the maximum value of the exact solution u by taking the maximum of u_h.

The resulted estimate is:

$$(x_{\max},\ \max u_h) \approx (-0.4931,\ 0.2385)$$



(a) Plot

(b) Zoom on the point of maximum

Figure 3: Graphics of the obtained approximated solutions u_h with the point of maximum $(x_{\max}, \max u_h)$

Multigrid in 2d

Let $\Omega = (-1,1)^2 = (-1,1) \times (-1,1) \subset \mathbb{R}^2$. Given the continuos functions $c(\mathbf{x}) = ||\mathbf{x}||^2$ and $f(\mathbf{x}) = f(x_1, x_2) = x_1 \cdot e^{2x_2}$, the aim is to find a function $u \in C^2(\Omega)$ such that:

$$\begin{cases} \Delta u + cu = f & in\ \Omega \\ u = 0 & on\ \Omega \end{cases} \tag{2}$$

here $c(\mathbf{x}) = ||\mathbf{x}||^2$ denotes the usual Euclidean norm in $\mathbb{R}^2$.

In particular, as in the one dimensional case, we will provide different methods to find an approximated solution of u in order to finally perform a final full multigrid method.

For this purpose we consider a uniform partitioning of the intervals $[-1,1]$ with mesh size $h = 2/N$ with $N = 2^l,\ l \geq 1$ and therefore a uniform grid Ω_l as in the one dimensional problem. In this case, the standard finite difference approximation to the given boundary value problem can be represented with the following linear system:

$$A_h u_h = f_h$$

where, this time, $A_h \in \mathbb{R}^{(N-1)^2} \times \mathbb{R}^{(N-1)^2}$ and $f_h,\ u_h$ are two dimensional arrays.

Investigation

The first method that has been implemented is a **weighted Jacobi** with $\omega = 2/3$ as weight. After that a **two-grid correction scheme** has been implemented, where instead of solving the residual equation on Ω_{2h}, it performs ten weighted Jacobi steps on the coarse grid.
In both cases the iteration, starting from $u_h^{(0)} = 0$, stops when the pseudo-residual satisfies $|s_h^{(k+1)}|_\infty < \text{tol} = 10^{-8}$.
In this two dimensional setting an investigation on the performance of these two methods applied to the linear system (2) for the stated model problem, as h decreases, has been made. Such as in the 1dimensional case, we are interested on the effect on the CPU time and on the number of iterations needed for the methods, but we have considered a sequence of grid sizes which include $N = 2^l$, with $j = 2, \ldots, 6$.

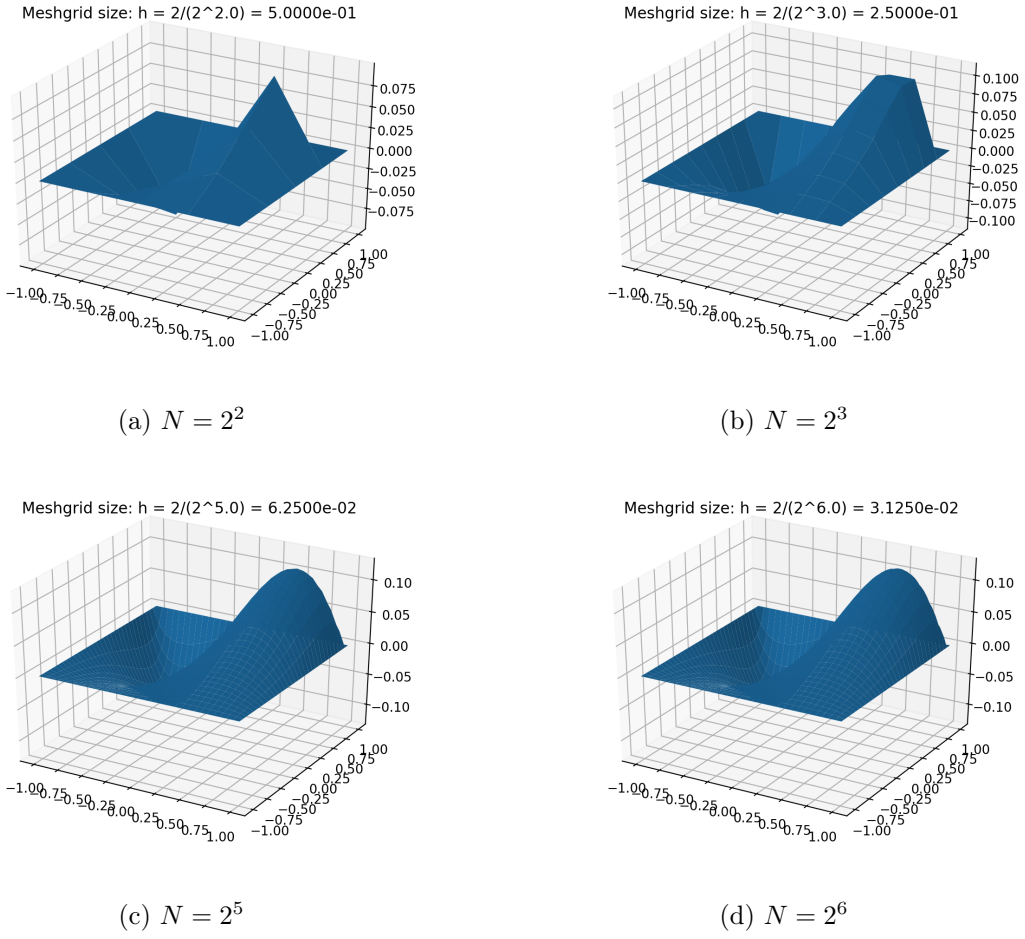
The obtained results are the following:

l	N	h	Jacobi iter.	Jacobi time	Two-grids Iter.	Two-grid time
2	4	0.5	28	0.0051 s	14	0.0380 s
3	8	0.25	104	0.0600 s	15	0.0465 s
4	16	0.125	379	0.3049 s	15	0.1006 s
5	32	0.0625	1353	3.6424 s	34	0.7352 s
6	64	0.03125	4745	48.5064 s	115	7.5695 s

Table 4: Investigation on Jacobi methods as h decreases

As in the one dimensional case the two-grid correction scheme has provided quite an improvement compared to the classical Jacobi method. However, as we expected the number of iterations needed to reach convergence and the CPU time increase with N, for both method. To overcome this problem one can perform again the **Multigrid method** instead of the classical iteratives ones.

Before moving on on the next investigation about different methods based on the multigrid, let's have a look at the plot of the solution obtained with the previous method. Since the two grid correction schemes gave us plots that are very similar to the Jacobi's ones, we will report only the latter ones.



(a) $N = 2^2$ (b) $N = 2^3$

(c) $N = 2^5$ (d) $N = 2^6$

Figure 4: Graphics of the the obtained approximated solutions u_h for different values of the mesh grid size $h = 2/N$, using iterations of weighted Jacobi.

As said before, simulations with better performances can be achieved computing the solutions using multigrid methods. For this purpose, the first function that has been implemented iterates **V-cycles** for the linear system (2), using for each of them one weighted Jacobi step with weight $\omega = 2/3$ as pre- and post- smoothing. On the coarsest grid Ω_1 uses an exact solve which is given by the non zero element:

$$u_h[1,1] = A_h[1,1]^{-1} \cdot f_h[1,1] \;=\; \frac{h^2}{4 + c_h[1,1] \cdot h^2} \cdot f_h[1,1] \;=\; \frac{h^2}{4} \cdot f_h[1,1] \;=\; \frac{f_h[1,1]}{N^2}$$

Moreover, at each step it computes and returns the pseudo-residual of the post-smoothing Jacobi step, unless it is called on the coarsest grid Ω_1, when it returns zero. Again the iteration, starting from a zero initial guess as usual, stops when the pseudo-residual satisfies $|s_h^{(k+1)}|_\infty < \text{ tol } = 10^{-8}$ for the post-smoothing step on the finest grid.

To test the performances an investigation on the number of iterations needed for the implemented V-cycle and the effect on the CPU time as h decreases, has been made.

After that we implemented a function that performs a **full multigrid**, utilizing the V-cycle implementation to perform a single V-cycle step in each grid, for the linear system (1). This function returns the result of the call to the V-cycle step on the finest grid.

Also in this case we investigated the CPU time needed and the sizes of the final pseudo-residual $|s_h^{(k+1)}|_\infty$ a well as the size of the residual $|r_h|_\infty = |f_h - A_h u_h|_\infty$ after one full multigrid step.
In this case, for the sequence of the grid sizes we have taken $N = 2^l,\ l = 2, 3, \dots 9$.
We can compare and summarize the obtained results in the following table:

l	N	V-cycle iter.	V-cycle time	FullMG s_h.	FullMG r_h	FullMG time
2	4	14	0.0097 s	2.1620e-02	2.4834e-01	0.0061 s
3	8	16	0.0388 s	1.6476e-02	8.2092e-01	0.0112 s
4	16	17	0.0834 s	6.7169e-03	1.3881	0.0279 s
5	32	16	0.2322 s	2.1211e-03	1.7813	0.0542 s
6	64	16	0.8474 s	6.0914e-04	2.1220	0.1124 s
7	128	15	3.1093 s	1.6539e-04	2.3341	0.4472 s
8	256	14	11.7592 s	4.3019e-05	2.4982	1.3393 s
9	512	12	41.4192 s	1.1033e-05	2.5979	6.0310 s

Table 5: Investigation on V-cycle and Full Multigrid methods as h decreases

We can notice that the convergence is faster than the one of Jacobi method, as we expected. The convergence's improvement, compared to the one before, is clearly visible already in the "V-cycle": again, as in the one dimensional case, this is due to the fact that using the multigrid method the number of iterates does not deteriorate as h decreases.
The CPU performances enhance even more in the Full multigrid method.

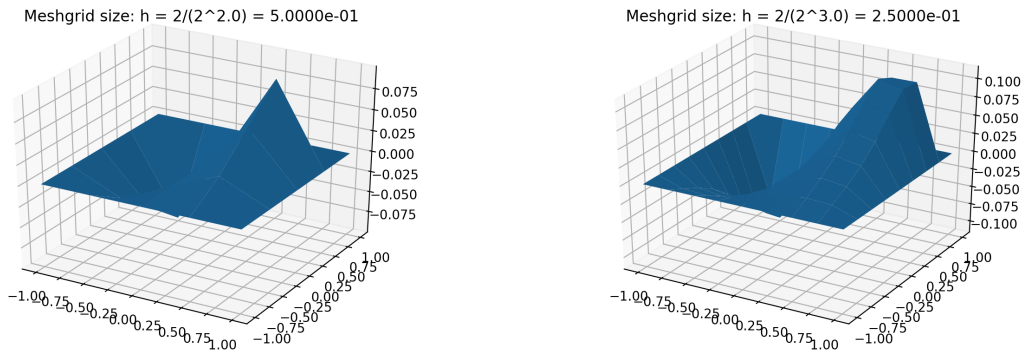
At this point we investigate the number of **additional V-cycles** needed, and the effect on the total CPU time, by using the full multigrid method to provide a (more accurate) initial guess for the V-cycle multigrid method. For doing that, after calling the function which performs the Full multigrid method, we use the produced approximation u_h as the initial guess for the V-cycle multigrid method.
The stopping criterion and the sequence of grid sizes are the same as the previous ones.
The obtained results are visualized in a compact way, together with the previous ones, in the following table (the columns l and N have been dropped):

V-cycle it.	V-cycle time	FullMG s_h.	FullMG r_h	FullMG time	Iter.	CPU time
14	0.0097 s	2.1620e-02	2.4834e-01	0.0061 s	13	0.0172 s
16	0.0388 s	1.6476e-02	8.2092e-01	0.0112 s	15	0.0462 s
17	0.0834 s	6.7169e-03	1.3881	0.0279 s	15	0.1005 s
16	0.2322 s	2.1211e-03	1.7813	0.0542 s	14	0.2657 s
16	0.8474 s	6.0914e-04	2.1220	0.1124 s	14	0.8474 s
15	3.1093 s	1.6539e-04	2.3341	0.4472 s	12	3.0761 s
14	11.7592 s	4.3019e-05	2.4982	1.3393 s	11	11.6290 s
12	41.4192 s	1.1033e-05	2.5979	6.0310 s	9	35.7496 s

Table 6: Investigation on V-cycle and Full Multigrid methods as h decreases

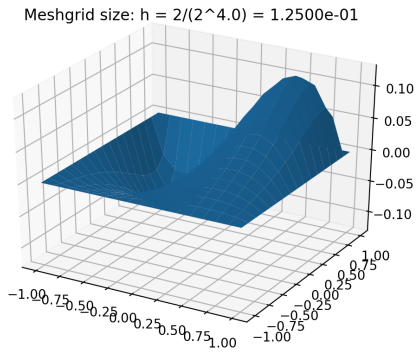
Also in this case the second-last column refers to the number of added V-cycle iterations and the one titled "CPU time" includes the time spent performing the full multigrid method and all the successive V cycles. From this table one can notice that also in the two dimensional case, providing the V-cycles method with a more precise initial guess more precise than $u_h^{(0)} \equiv 0$ quite improves the performances.

To conclude all these investigations, a table of plots representing the approximated solution found with this method (V-cyles iterations with initial guess provided with a full multigrid methods) is provided:

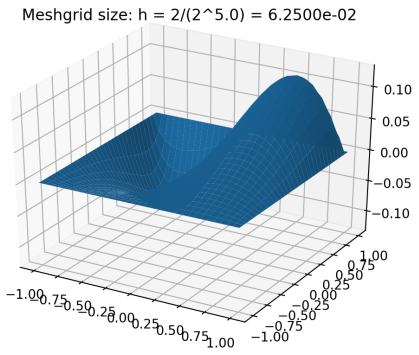


(a) $N = 2^2$ (b) $N = 2^3$

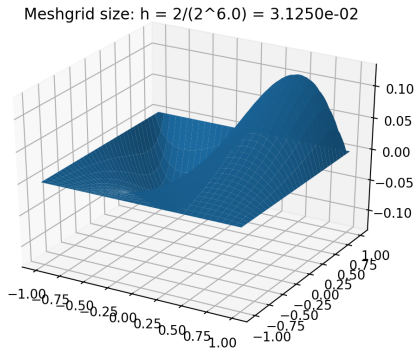
Figure 5: Graphics of the obtained approximated solutions u_h for different values of the mesh grid size $h = 2/N$, $N = 2^l$ with $l = 2, 3$.



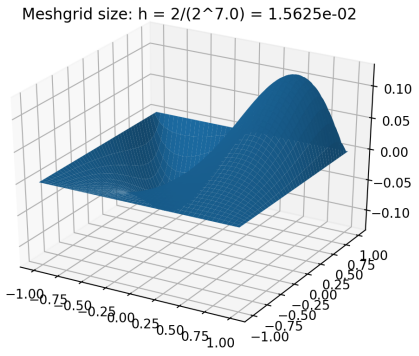
(a) $N = 2^4$



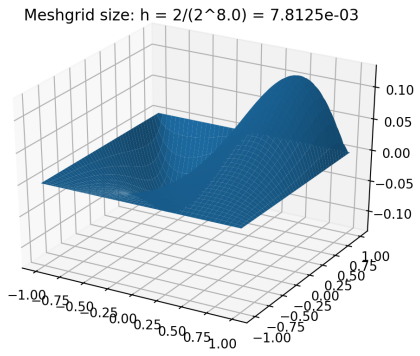
(b) $N = 2^5$



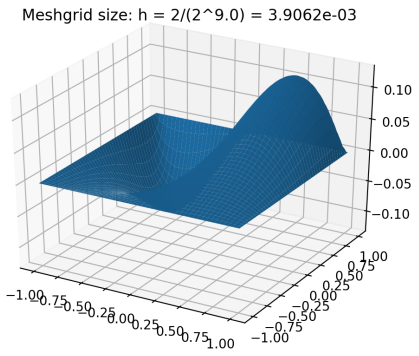
(c) $N = 2^6$



(d) $N = 2^7$



(e) $N = 2^8$



(f) $N = 2^9$

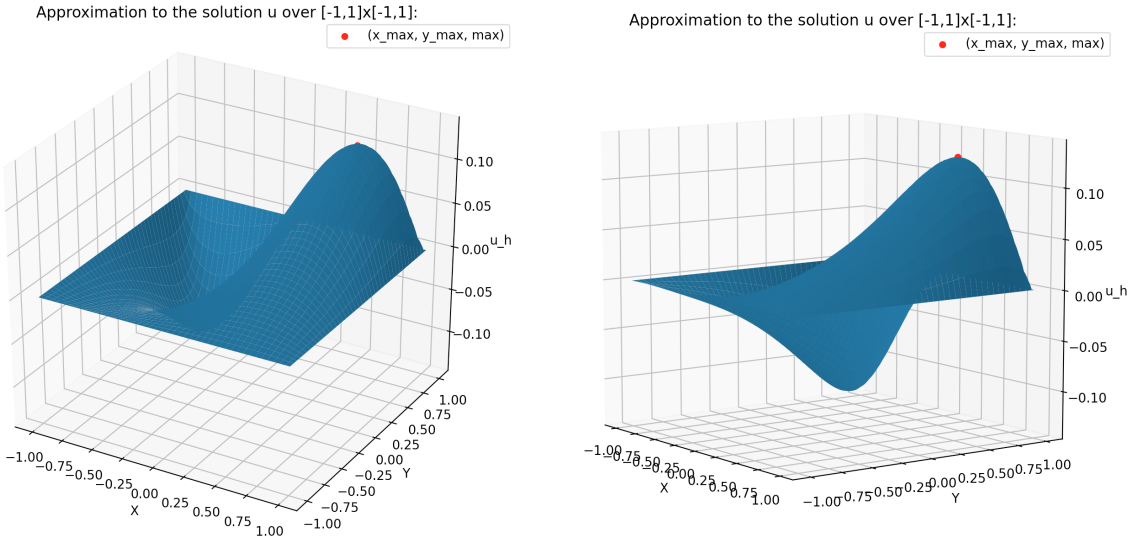
Figure 6: Graphics of the obtained approximated solutions u_h for different values of the mesh grid size $h = 2/N$, $N = 2^l$ with $l = 4, \dots 9$.

Approximation of the maximum

After all this investigation we are able to provide an accurate approximation of the solution u and therefore we are also able to compute a precise estimate of the $\max_{x\in[-1,1]} u(x)$.
Let u_h be the final approximated solution, obtained with a V-cycle multigrid method that uses as initial guess the one provided by a full multigrid method. Therefore we can estimate the maximum value of the exact solution u by taking the maximum of u_h.

The resulted estimate is:

$$(x_{\max}, y_{\max},\ \max u_h) \approx (0.625,\ 0.625,\ 0.1276)\,.$$



(a) Plot

(b) Plot rotated

Figure 7: Graphic of the obtained approximated solutions u_h with the point of maximum $(x_{\max}, y_{\max} \max u_h)$ for a mesh griz of size $h = \frac{2}{2^9}$