Danmarks Tekniske Universitet

02687 Scientific Computing for Ordinary and Partial Differential Equations

Assignement 1

The Finite Difference Method, Boundary Value Problems and Iterative solution of Linear systems

Letizia Bottani, s243157
Simon Hagerup Jensen, s153736
Magnús Jarl Skjoldager, s204244

March 30, 2025

Contents

1 The basics of Finite Difference Methods

The *Finite Difference Method* (FDM) is a widely used numerical technique for approximating derivatives and solving differential equations on discrete grids. It plays a fundamental role in numerical analysis, particularly in the approximation of solutions to *Initial Value Problems* (IVPs) and *Boundary Value Problems* (BVPs). The core idea behind FDM is to replace continuous derivatives with discrete approximations using *finite difference stencils*, which express the derivative of a function in terms of function values at nearby grid points.
In this exercise, we explore the construction of *finite difference stencils* for approximating second-order derivatives with different configurations. Using *Taylor series expansions*, we derive the stencil coefficients to ensure maximum accuracy. The accuracy of these approximations is characterized by the *order of truncation error*, which determines how well the discrete approximation converges to the true derivative as the grid spacing decreases.
The first part of this exercise focuses on deriving *two finite difference stencils* for approximating the second derivative $u''(x)$, considering:

- **An off-centered stencil** with asymmetry $(\alpha, \beta) = (4, 0)$, where all points are located to one side.
- **A centered stencil** with symmetry $(\alpha = \beta = 2)$, ensuring a balanced distribution of grid points around the target location.

The objective is to compute the *stencil weights* for each case, analyze their *accuracy order*, and compare their effectiveness in approximating second derivatives.

1.1 Stencil Construction for Second Derivative Approximation

We want to approximate the second-order derivative with the following stencils $(\alpha, \beta) = (4, 0)$ and $(\alpha, \beta) = (2, 2)$.

For $(\alpha, \beta) = (4, 0)$, the approximation takes the following form:

$$\frac{d^2u(\bar{x})}{dx^2} \approx au(\bar{x} - 4h) + bu(\bar{x} - 3h) + cu(\bar{x} - 2h) + du(\bar{x} - h) + eu(\bar{x}) \tag{1}$$

Taylor expanding each term around $\bar{x}$:

$$u(\bar{x} - 4h) \approx u(\bar{x}) - 4hu'(\bar{x}) + 8h^2u''(\bar{x}) - \frac{32}{3}h^3u'''(\bar{x}) + \frac{32}{3}h^4u''''(\bar{x}) + \mathcal{O}(h^5)$$

$$u(\bar{x} - 3h) \approx u(\bar{x}) - 3hu'(\bar{x}) + \frac{9}{2}h^2u''(\bar{x}) - \frac{9}{2}h^3u'''(\bar{x}) + \frac{27}{8}h^4u''''(\bar{x}) + \mathcal{O}(h^5)$$

$$u(\bar{x} - 2h) \approx u(\bar{x}) - 2hu'(\bar{x}) + 2h^2u''(\bar{x}) - \frac{4}{3}h^3u'''(\bar{x}) + \frac{2}{3}h^4u''''(\bar{x}) + \mathcal{O}\left(h^5\right)$$

$$u(\bar{x} - h) \approx u(\bar{x}) - hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) - \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u''''(\bar{x}) + \mathcal{O}\left(h^5\right)$$

Substituting these Taylor expansions into (1) and collecting terms leads to the following expression:

$$\begin{aligned}\frac{d^2u(\bar{x})}{dx^2} \approx\ & u(\bar{x})(a+b+c+d+e)\\ & + u'(\bar{x})h(-4a-3b-2c-d)\\ & + u''(\bar{x})h^2\left(8a+\frac{9}{2}b+2c+\frac{1}{2}d\right)\\ & + u'''(\bar{x})h^3\left(-\frac{32}{3}a-\frac{9}{2}b-\frac{4}{3}c-\frac{1}{6}d\right)\\ & + u''''(\bar{x})h^4\left(\frac{32}{3}a+\frac{27}{8}b+\frac{2}{3}c+\frac{1}{24}d\right)\\ & + \mathcal{O}(h^5)\end{aligned}$$

Since we want to approximate the second derivative, we can write up the following system of equations:

$$\begin{cases} a+b+c+d+e=0\\ 4a+3b+2c+d=0\\ 8a+\frac{9}{2}b+2c+\frac{1}{2}d=1/h^2\\ -\frac{32}{3}a-\frac{9}{2}b-\frac{4}{3}c-\frac{1}{6}d=0\\ \frac{32}{3}a+\frac{27}{8}b+\frac{2}{3}c+\frac{1}{24}d=0 \end{cases}$$

Solving this system with Matlab, we get the vector of coefficients for our stencil:

$$\begin{bmatrix} a\\ b\\ c\\ d\\ e \end{bmatrix} = \begin{bmatrix} \frac{11}{12h^2}\\ -\frac{14}{3h^2}\\ \frac{19}{2h^2}\\ -\frac{26}{3h^2}\\ \frac{35}{12h^2} \end{bmatrix}$$

Thus, the approximation formula for $u''(\bar{x})$ using the off-centered stencil is:

$$\frac{d^2u(\bar{x})}{dx^2} \approx \frac{1}{h^2}\left(\frac{11}{12}u(\bar{x}-4h)-\frac{14}{3}u(\bar{x}-3h)+\frac{19}{2}u(\bar{x}-2h)-\frac{26}{3}u(\bar{x}-h)+\frac{35}{12}u(\bar{x})\right). \quad (2)$$

A similar approach is used to derive the centered stencil $(\alpha,\beta)=(2,2)$. The approximation takes the following form:

$$\frac{d^2u(\bar{x})}{dx^2} \approx au(\bar{x}-2h)+bu(\bar{x}-h)+cu(\bar{x})+du(\bar{x}+h)+eu(\bar{x}+2h) \quad (3)$$

Expanding in Taylor series, the function value u in the points of the stencil:

$$
\begin{aligned}
u(\bar{x}-2h) &\approx u(\bar{x}) - 2hu'(\bar{x}) + 2h^2u''(\bar{x}) - \frac{4}{3}h^3u'''(\bar{x}) + \frac{2}{3}h^4u''''(\bar{x}) + \mathcal{O}(h^5),\\
u(\bar{x}-h) &\approx u(\bar{x}) - hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) - \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u''''(\bar{x}) + \mathcal{O}(h^5),\\
u(\bar{x}+h) &\approx u(\bar{x}) + hu'(\bar{x}) + \frac{h^2}{2}u''(\bar{x}) - \frac{h^3}{6}u'''(\bar{x}) + \frac{h^4}{24}u''''(\bar{x}) + \mathcal{O}(h^5),\\
u(\bar{x}+2h) &\approx u(\bar{x}) + 2hu'(\bar{x}) + 2h^2u''(\bar{x}) + \frac{4}{3}h^3u'''(\bar{x}) + \frac{2}{3}h^4u''''(\bar{x}) + \mathcal{O}(h^5).
\end{aligned}
$$

Inserting the Taylor-expansions in (3) we get the following expression:

$$
\begin{aligned}
\frac{d^2u(\bar{x})}{dx^2} \approx\ & u(\bar{x})(a+b+c+d+e)\\
&+ u'(\bar{x})h(-2a-b+d+2e)\\
&+ u''(\bar{x})h^2\left(2a+\frac{b}{2}+\frac{d}{2}+2e\right)\\
&+ u'''(\bar{x})h^3\left(-\frac{4}{3}a-\frac{1}{6}b+\frac{1}{6}d+\frac{4}{3}e\right)\\
&+ u''''(\bar{x})h^4\left(\frac{2}{3}a+\frac{1}{24}b+\frac{1}{24}d+\frac{2}{3}e\right)\\
&+ \mathcal{O}(h^5)
\end{aligned}
$$

that leads to the following linear system:

$$
\begin{cases}
a+b+c+d+e=0\\
-2a-b+d+2e=0\\
2a+\frac{b}{2}+\frac{d}{2}+2e=\frac{1}{h^2}\\
-\frac{4}{3}a-\frac{1}{6}b+\frac{1}{6}d+\frac{4}{3}e=0\\
\frac{2}{3}a+\frac{1}{24}b+\frac{1}{24}c+\frac{2}{3}e=0
\end{cases}
$$

Solving it with MATLAB, we get the following vector of coefficients:

$$
\begin{bmatrix} a\\ b\\ c\\ d\\ e \end{bmatrix} = \begin{bmatrix} \frac{-1}{12h^2}\\ \frac{4}{3h^2}\\ \frac{-5}{2h^2}\\ \frac{4}{3h^2}\\ \frac{-1}{12h^2} \end{bmatrix}
$$

Thus, the approximation formula for $u''(\bar{x})$ using the centered stencil is:

$$
\frac{d^2u(\bar{x})}{dx^2} \approx \frac{1}{h^2}\left(-\frac{1}{12}u(\bar{x}-2h) + \frac{4}{3}u(\bar{x}-h) - \frac{5}{2}u(\bar{x}) + \frac{4}{3}u(\bar{x}+h) - \frac{1}{12}u(\bar{x}+2h)\right) \qquad (4)
$$

1.2 Determination of the Order of Accuracy of the Stencils

To determine the order of accuracy, we examine the leading order term of the truncation error:

$$\tau(x) = \frac{d^2u(x)}{dx^2} - D_\bullet u(x) = \mathcal{O}\left(h^p\right) \tag{5}$$

To determine the order of accuracy, we need to consider Taylor expansions up to order 7.

For $(\alpha, \beta) = (4, 0)$, the second-order derivative is given by (2). We can approximate neighboring points in the stencil with the following Taylor expansions:

$$u(\bar{x}-4h) \approx u(\bar{x})-4hu'(\bar{x})+8h^2u''(\bar{x})-\frac{32}{3}h^3u'''(\bar{x})+\frac{32}{3}h^4u^{(4)}(\bar{x})-\frac{4^5}{5!}h^5u^{(5)}(\bar{x})+\frac{4^6}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(h^7\right)$$

$$u(\bar{x}-3h) \approx u(\bar{x})-3hu'(\bar{x})+\frac{9}{2}h^2u''(\bar{x})-\frac{9}{2}h^3u'''(\bar{x})+\frac{27}{8}h^4u^{(4)}(\bar{x})-\frac{3^5}{5!}h^5u^{(5)}(\bar{x})+\frac{3^6}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(h^7\right)$$

$$u(\bar{x}-2h) \approx u(\bar{x})-2hu'(\bar{x})+2h^2u''(\bar{x})-\frac{4}{3}h^3u'''(\bar{x})+\frac{2}{3}h^4u^{(4)}(\bar{x})-\frac{2^5}{5!}h^5u^{(5)}(\bar{x})+\frac{2^6}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(h^7\right)$$

$$u(\bar{x}-h) \approx u(\bar{x})-hu'(\bar{x})+\frac{h^2}{2}u''(\bar{x})-\frac{h^3}{6}u'''(\bar{x})+\frac{h^4}{24}u^{(4)}(\bar{x})-\frac{1}{5!}h^5u^{(5)}(\bar{x})+\frac{1}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(h^7\right)$$

Substituting these Taylor expansions into (2) and computing the truncation error with (5), we obtain:

$$\begin{aligned}\tau(\bar{x}) =& h^3\left(-\frac{11}{12}\frac{4^5}{5!}+\frac{14}{3}\frac{3^5}{5!}-\frac{19}{2}\frac{2^5}{5!}+\frac{26}{3}\frac{1}{5!}\right)u^{(5)}\\ &+h^4\left(\frac{11}{12}\frac{4^6}{6!}-\frac{14}{3}\frac{3^6}{6!}+\frac{19}{2}\frac{2^6}{6!}-\frac{26}{3}\frac{1}{6!}\right)u^{(6)}+\mathcal{O}(h^5).\end{aligned}$$

Simplifying to the relevant terms:

$$\tau(\bar{x}) = -\frac{5}{6}h^3u^{(5)}(\bar{x}) + \mathcal{O}(h^4)$$

Thus, we conclude that the order of accuracy of the approximation is 3.

For $(\alpha, \beta) = (2, 2)$, the approximation is given by (4). Expanding with Taylor series:

$$\begin{aligned}u(\bar{x}-2h) &\approx u(\bar{x})-2hu'(\bar{x})+2h^2u''(\bar{x})-\frac{4}{3}h^3u'''(\bar{x})+\frac{2}{3}h^4u^{(4)}(\bar{x})-\frac{2^5}{5!}h^5u^{(5)}(\bar{x})+\frac{2^6}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(h^7\right),\\ u(\bar{x}-h) &\approx u(\bar{x})-hu'(\bar{x})+\frac{h^2}{2}u''(\bar{x})-\frac{h^3}{6}u'''(\bar{x})+\frac{h^4}{24}u^{(4)}(\bar{x})-\frac{1}{5!}h^5u^{(5)}(\bar{x})+\frac{1}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(h^7\right),\\ u(\bar{x}+h) &\approx u(\bar{x})+hu'(\bar{x})+\frac{h^2}{2}u''(\bar{x})-\frac{h^3}{6}u'''(\bar{x})+\frac{h^4}{24}u^{(4)}(\bar{x})\frac{1}{5!}h^5u^{(5)}(\bar{x})+\frac{1}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(x^7\right),\\ u(\bar{x}+2h) &\approx u(\bar{x})+2hu'(\bar{x})+2h^2u''(\bar{x})+\frac{4}{3}h^3u'''(\bar{x})+\frac{2}{3}h^4u^{(4)}(\bar{x})+\frac{2^5}{5!}h^5u^{(5)}(\bar{x})+\frac{2^6}{6!}h^6u^{(6)}(\bar{x})+\mathcal{O}\left(x^7\right).\end{aligned}$$

Substituting these into (4) and into (5):

$$\tau(\bar{x}) = h^3 \left(\frac{1}{12}\frac{2^5}{5!} - \frac{1}{5!}\frac{4}{3} + \frac{4}{3}\frac{1}{5!} - \frac{1}{12}\frac{2^5}{5!} \right) u^{(5)} + \frac{h^4}{6!} \left(\frac{-2^6}{12} + \frac{4}{3} + \frac{4}{3} - \frac{2^6}{12} \right) u^{(6)}$$

Simplifying to the relevant terms:

$$\tau(\bar{x}) = -\frac{8}{6!} h^4 u^{(6)} + \mathcal{O}\left(h^5\right). \tag{6}$$

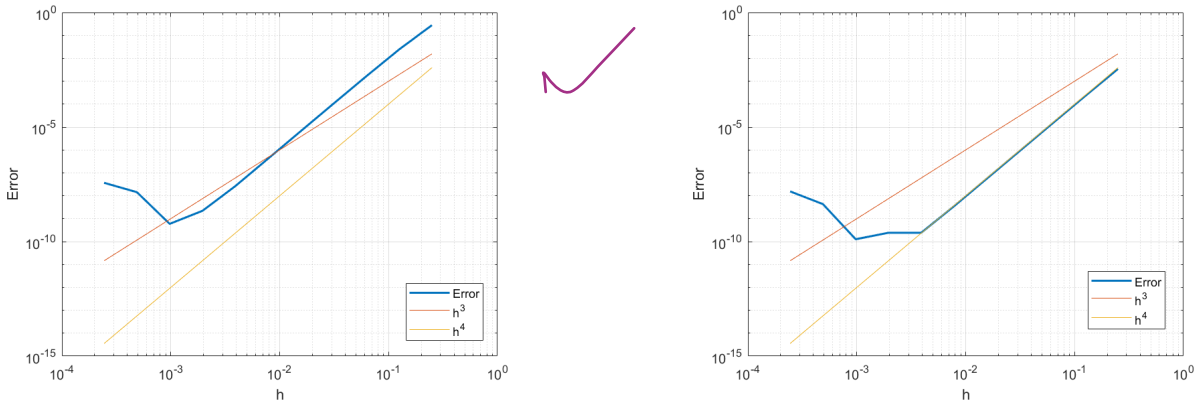
Thus, we conclude that the order of accuracy of the approximation is 4. This demonstrates that the stencil $(\alpha, \beta) = (2, 2)$ is more accurate than $(\alpha, \beta) = (4, 0)$.

1.3 Computation of the Second Spatial Derivative with MATLAB

In the MATLAB script `Ex1.m` - section `1_c`- we compute an approximation of the second spatial derivative of the function $u(x) = \exp(\cos(x))$ at $x = 0$, considering the finite difference schemes studied in the previous paragraphs on an equidistant grid.
Testing the code for $h = 0.2$ gives an error of 0.0014 for the $(\alpha, \beta) = (2, 2)$ stencil, and 0.1339 for the $(\alpha, \beta) = (4, 0)$ stencil. This result aligns with our previous theoretical analysis, as the centered scheme has a higher order of accuracy (fourth order) compared to the off-centered scheme (third order). The higher order accuracy leads to a smaller truncation error, resulting in a more accurate approximation of the second derivative.

1.4 Error Analysis and Convergence Test

In the MATLAB script `Ex1.m` - section `1_d`, we measure the error as a function of the grid increment h, where $h = \frac{1}{2^s}$ for $s = 2, 3, 4, \ldots$, for both schemes. The results are shown in Figure 1.



(a) Scheme with $(\alpha, \beta) = (4, 0)$ (b) Scheme with $(\alpha, \beta) = (2, 2)$

Figure 1: Trend of the error for the two schemes

From 1.2, we expect the error to scale as h^3 for the off-centered scheme and as h^4 for the centered one. As seen in Figure 1, for the second scheme (Figure 1b), the error slope aligns well with the theoretical prediction. On the other hand, for the first scheme (Figure 1a), the

error scales as h^4, which is better than the theoretical expectation. This discrepancy arises from the point at which the derivative is approximated. We know that

$$\tau(\bar{x}) = -\frac{5}{6}h^3u^{(5)}(\bar{x}) + \frac{119}{90}h^4u^{(6)}(\bar{x}) + \mathcal{O}(h^5),$$

but since we are approximating $u(x)$ at $\bar{x} = 0$, we have that $u^{(5)}(0) = 0$, so the dominant term of the truncation error is instead $\mathcal{O}(h^4)$ which implies that the error scales as h^4, since $u^{(6)}(0) \neq 0$.
From Figure 1, it can be observed that for $h < 10^{-2}$, the error no longer follows the expected scaling behavior. This deviation is due to the fact that the computed error becomes smaller than the machine precision, meaning that it is no longer dominated by the truncation error of the numerical scheme. Instead, it is influenced by the accumulation of numerical round-off errors, which arise from the finite precision arithmetic used in computations.

1.5 Interpolation Stencil for Order 3 Accuracy

We aim to determine a stencil for interpolation of $e^{\cos x}$, at $x = 0$ that achieves an order of accuracy of 3. We are using a uniform grid with grid distances h, where the two nodes next to the interpolation point are $h/2$ away.

If we use the symmetric stencil with two grid points:

$$u''(x) \approx a_{-\frac{1}{2}}u\left(x - \frac{h}{2}\right) + a_{\frac{1}{2}}u\left(x + \frac{h}{2}\right) \tag{7}$$

Using the Taylor series expansion around $x = 0$, we determine the coefficients:

$$a_{-\frac{1}{2}} = \frac{1}{2}, \quad a_{\frac{1}{2}} = \frac{1}{2} \tag{8}$$

which results in the truncation error:

$$\tau(0) = \frac{h^2}{8}u''(0) + \mathcal{O}(h^3) \tag{9}$$

Here we see that the stencil gives a order of accuracy of 2.
Next, we consider using three grid points, adding one on the positive side. Since $e^{\cos x}$ is symmetric, the choice of side does not affect the result.

$$u''(x) \approx a_{-\frac{1}{2}}u\left(x - \frac{h}{2}\right) + a_{\frac{1}{2}}u\left(x + \frac{h}{2}\right) + a_{\frac{3}{2}}u\left(x + \frac{3h}{2}\right) \tag{10}$$

Again, using the Taylor series approximation, we find the coefficients:

$$a_{-\frac{1}{2}} = \frac{3}{8}, \quad a_{\frac{1}{2}} = \frac{3}{4}, \quad a_{\frac{3}{2}} = -\frac{1}{8} \tag{11}$$

which gives the truncation error:

$$\tau(0) = -\frac{h^3}{16}u'''(0) - \frac{3h^4}{128}u^{(4)}(0) + \mathcal{O}(h^5) \tag{12}$$

This achieves an order of accuracy of 3. However, comparing this to the measured truncation error from the simulation (as shown in Figure 2), we observe that we actually achieve an order of accuracy of 4. This occurs because $u'''(0) = 0$, leading to the actual truncation error:

$$\tau(0) = -\frac{h^3}{16}0 - \frac{3h^4}{128}u^{(4)}(0) + \mathcal{O}(h^5) \tag{13}$$

Thus, while we theoretically require three grid points to achieve an interpolation accuracy of order 3, at $x = 0$ we obtain a higher order of accuracy, namely 4.

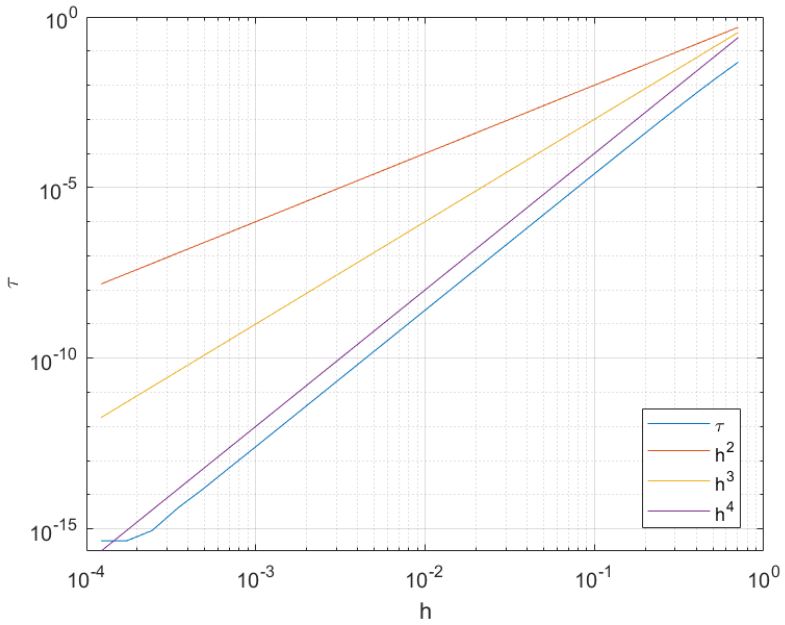


Figure 2: Observed convergence rate for the interpolation scheme.

2 Two-Point BVPs and Discretization of the Laplacian

2.1 Two-point BVPs

In this exercise, we want to solve the nonlinear BVP with Dirichlet boundary conditions:

$$\epsilon u'' + u(u' - 1) = 0, \quad 0 \leq t \leq 1, \tag{14}$$

$$u(0) = \alpha, \quad u(1) = \beta. \tag{15}$$

We will take $\alpha = -1$, $\beta = 1.5$, and set $\epsilon = 0.1$. To solve it numerically, we discretize our solution on a uniform grid. Let t_j be the grid points:

$$t_j = jh, \quad j = 0, 1, 2, \ldots, N, \quad \text{where} \quad h = \frac{1}{N} \tag{16}$$

Since the BVP has Dirichelet boundary conditions, the solution vector is given by: $\mathbf{u} = [u_1, u_2, \ldots, u_{N-1}]^T \in \mathbb{R}^{N-1}$, where $u_j = u(t_j)$ and the boundary values are fixed: $u_0 = \alpha$ and $u_{N+1} = \beta$.

2.1.1 Finite Difference Approximation

We use the central difference formula to approximate the second derivative at the inner points $j = 1, ..., N$:

$$u''(t_j) \approx \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \mathcal{O}(h^2). \tag{17}$$

We already know from theory that this is a second-order accurate approximation of $u''(t)$, meaning the truncation error is $\mathcal{O}(h^2)$.

For the first derivative $u'(t)$, we use a central difference scheme:

$$u'(t_j) \approx \frac{u_{j+1} - u_{j-1}}{2h} + \mathcal{O}(h^2). \tag{18}$$

This approximation is also second-order accurate.
Substituting (17) and (18) into the BVP equation (14), we obtain the following finite difference scheme:

$$\epsilon \frac{(u_{j+1} - 2u_j + u_{j-1})}{h^2} + u_j \left(\frac{u_{j+1} - u_{j-1}}{2h} - 1 \right) = 0 \tag{19}$$

Rearranging the terms:

$$\epsilon \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + \frac{u_j u_{j+1} - u_j u_{j-1}}{2h} - u_j = 0. \tag{20}$$

This is a nonlinear finite difference equation that must be solved with iterative methods like Newton's method.

2.1.2 Study of the Truncation Error

To analyze the local truncation error τ_j, we substitute the Taylor series expansion of u_{j+1} and u_{j-1} around t_j, into the finite difference approximation formula.
Expanding u_{j+1} and u_{j-1} up to fourth order:

$$u_{j+1} = u_j + hu'_j + \frac{h^2}{2}u''_j + \frac{h^3}{6}u'''_j + \mathcal{O}(h^4), \tag{21}$$

$$u_{j-1} = u_j - hu'_j + \frac{h^2}{2}u''_j - \frac{h^3}{6}u'''_j + \mathcal{O}(h^4). \tag{22}$$

Substituting into the finite difference approximation (17) and (18):

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = u'' + \mathcal{O}(h^2) \tag{23}$$

$$\frac{u_{j+1} - u_{j-1}}{2h} = u'_j + \frac{h^2}{6}u'''_j + \mathcal{O}(h^4) \tag{24}$$

Now, substituting (23) and (24) into (19) we obtain:

$$\epsilon \left(u''_j + \mathcal{O}\left(h^2\right) \right) + u_j \left(u'_j + \frac{h^2}{6}u'''_j + \mathcal{O}\left(h^4\right) - 1 \right) = 0$$

Simplifying:

$$\epsilon u_j'' + u_j \left(u_j' - 1\right) + \mathcal{O}\left(h^2\right) = 0 \Rightarrow$$

Thus, the local truncation error is:

$$\tau \approx \mathcal{O}(h^2)$$

Hence, we confirm that the numerical scheme is second order accurate.

2.1.3 Newton's method solution

Newton's method is used to solve the system $F(u) = 0$, where $F(u)$ represents a nonlinear function.
Starting from an initial guess, provided in eq. (2.105) by LeVeque [1], the method iteratively updates the solution using:

$$J\Delta u = -F(u), \tag{25}$$

where J is the Jacobian matrix of F, Δu is the Newton step correction and u is updated as $u^{(k+1)} = u^{(k)} + \Delta u$.
For our system, the Jacobian matrix is a sparse tridiagonal matrix given by:

$$J_{j,j-1} = \frac{\epsilon}{h^2} - \frac{u_j}{2h}, \quad J_{j,j} = -\frac{2\epsilon}{h^2} - \frac{u_{j+1} - u_{j-1}}{2h} - 1, \quad J_{j,j+1} = \frac{\epsilon}{h^2} + \frac{u_j}{2h}. \tag{26}$$

We implemented this solution in the MATLAB script called `Ex2_a.m`. The method iterates until $||\Delta u||_\infty < 10^{-6}$ or the maximum iteration count (100) is reached. The method converged in 34 iterations, demonstrating the efficiency of Newton's method in handling this nonlinear problem. The obtained numerical solution is shown in Figure 3.

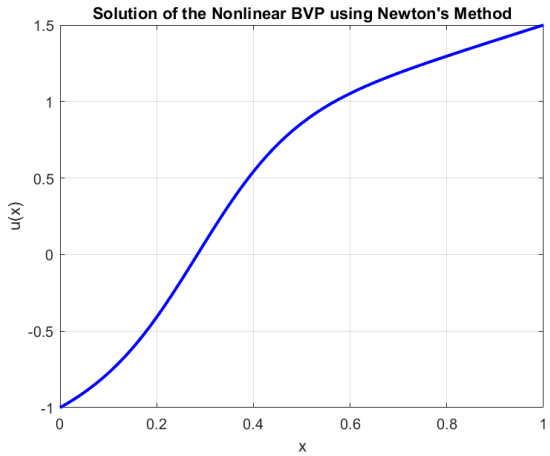


Figure 3: Numerical solution of the nonlinear BVP using Newton's method.

The solution accurately captures the expected behavior, including the boundary layers.

We want now to estimate the global error. Since the analytical solution for this BVP is too complicated, we assume that the numerical solution that we obtain with a very fine mesh ($N = 2^{10}$) is the exact solution. Then, we compute the error for different grid resolutions ($N = 2^k, k = 2, .., 8$) to see how it scales with respect to h and compare it with the error of the dine mesh. The log-log plot of the global error versus grid spacing is shown in Figure 4.

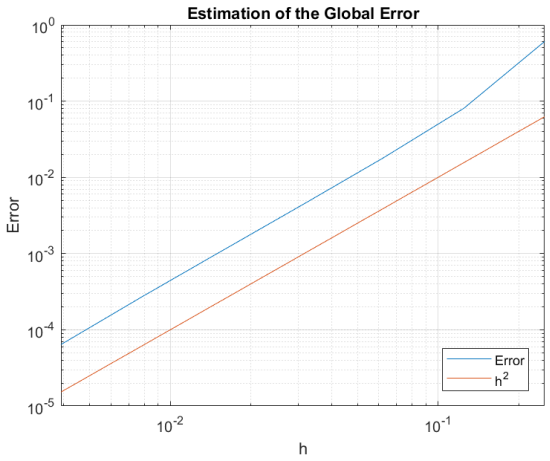


Figure 4: Global error estimation: The observed error follows the expected second-order convergence.

As expected, for a grid fine enough, the error decreases quadratically with decreasing grid spacing, confirming the second-order accuracy of the numerical scheme.

2.2 5-and 9-point Laplacian

In this exercise, we consider the problem of solving the Poisson equation on the unit square $[0,1] \times [0,1]$, subject to Dirichlet boundary conditions. Specifically, we aim to approximate the exact solution $u_{\text{exact}}(x,y) = \sin(4\pi(x+y)) + \cos(4\pi xy)$ by numerically solving the equation:

$$\Delta u = f(x,y), \quad (x,y) \in (0,1) \times (0,1),$$

where the Laplacian operator is defined as: $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.
The function $f(x,y)$ is obtained by substituting $u_{\text{exact}}(x,y)$ into the equation:

$$f(x,y) = -16\pi^2 \left(2\sin(4\pi(x+y)) + \cos(4\pi xy)(x^2+y^2)\right).$$

The domain is discretized using a uniform $(m+2) \times (m+2)$ grid, where m is the number of interior points in each direction. The grid points are given by:

$$x_i = ih, \quad y_j = jh, \quad h = \frac{1}{m+1}, \quad 0 \le i,j \le m+1.$$

Dirichlet boundary conditions are enforced by prescribing the values of u at the boundary points $x = 0, x = 1, y = 0, y = 1$, while the function values at the interior grid points are obtained by solving the discretized system.

2.2.1 Numerical Approximation Using the 5-Point Laplacian

To numerically approximate $u(x,y)$, we use the standard 5-point finite difference stencil, which provides a second-order accurate approximation to the Laplacian:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}.$$

Applying this discretization at each interior grid point results in a linear system of equations of the form:

$$Au = F,$$

where A is the sparse matrix representing the discrete Laplacian, u is the vector of unknown function values at interior points, F is the right-hand side vector, computed by evaluating $f(x, y)$ and incorporating contributions from the Dirichlet boundary conditions. The function `form_rhs(m,f,u)` constructs the right-hand side F by evaluating $f(x, y)$ at interior points while adjusting for boundary values.

Next, we construct the 5-point Laplacian matrix using the function `poisson5(m)`, solve the system, and reshape the result to obtain the numerical solution.

Convergence Analysis: To verify the expected second-order convergence of the 5-point Laplacian scheme, we compute the global error for a sequence of increasingly refined uniform grids, through the MATLAB script `Ex2_b.m`. Specifically, we consider grid spacings $h = 1/N$, where $N = [10, 20, 50, 80, 100, 150]$ is the number of interior points in each direction, and evaluate the maximum norm of the difference between the numerical and exact solutions.

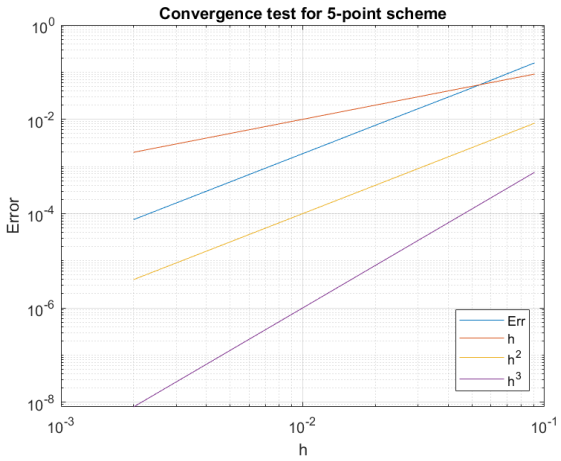


Figure 5: Global error vs. grid resolution for the 5-point Laplacian.

Figure 5 shows the global error plotted on a log-log scale as a function of the grid resolution. As the grid is refined, the error decreases approximately proportionally to h^2, confirming the theoretical second-order accuracy of the scheme. This behavior is consistent with the truncation error analysis of the 5-point stencil, which predicts an error of $\mathcal{O}(h^2)$ in the discrete approximation of the Laplacian operator.

2.2.2 Numerical Approximation Using the 9-Point Laplacian

While the standard 5-point Laplacian scheme provides second-order accuracy, we can improve accuracy by introducing higher-order approximations. One such method is the 9-point Laplacian scheme, which incorporates diagonal neighbors in addition to the standard cross pattern.
The 9-point Laplacian stencil is given by:

$$\Delta_9 u_{i,j} \approx \frac{-20u_{i,j} + 4(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}) + u_{i+1,j+1} + u_{i+1,j-1} + u_{i-1,j+1} + u_{i-1,j-1}}{6h^2}.$$

Unlike the 5-point Laplacian, the 9-point scheme is not in tensor-product form, so its truncation error must be analyzed via a two-dimensional Taylor expansion. Expanding $u(x_i, y_j)$ in terms of h, we obtain:

$$\nabla_9^2 u(x_i, y_j) = \nabla^2 u(x_i, y_j) + \frac{1}{12} h^2 \nabla^4 u(x_i, y_j) + \mathcal{O}(h^4).$$

The leading-order error term contains the biharmonic operator:

$$\nabla^4 u = \nabla^2(\nabla^2 u) = u_{xxxx} + 2u_{xxyy} + u_{yyyy}.$$

Since the Poisson equation states that $\nabla^2 u = f$, we can apply the Laplacian again:

$$\nabla^2(\nabla^2 u) = \nabla^2 f.$$

Thus, an improved fourth-order accurate solution can be obtained using deferred corrections:

$$\nabla_9^2 \bar{U}_{ij} = f_{ij} + \frac{1}{12} h^2 \nabla_5^2 f_{ij},$$

where $\bar{U}_{ij}$ represents the corrected approximation, and $\nabla_5^2 f_{ij}$ is the standard 5-point Laplacian applied to $f(x, y)$.

To implement the numerical solution to this problem, since the function poisson9(m) was already provided, we only need to construct the function that assembles the right-hand side while incorporating the deferred correction. Since the function $f(x, y)$ is known analytically, we computed its Laplacian directly rather than using the ∇_5 operator. The MATLAB function `form_rhs_9.m` implements this correction term and the right-hand side assembly.

Next, we construct the 9-point Laplacian matrix using the function poisson9(m) to solve the system. To test our method and check the order of accuracy, we consider the same problem as in the 5-point scheme and perform a convergence test. The MATLAB script related to this problem is called `Ex2_b_d.m`.

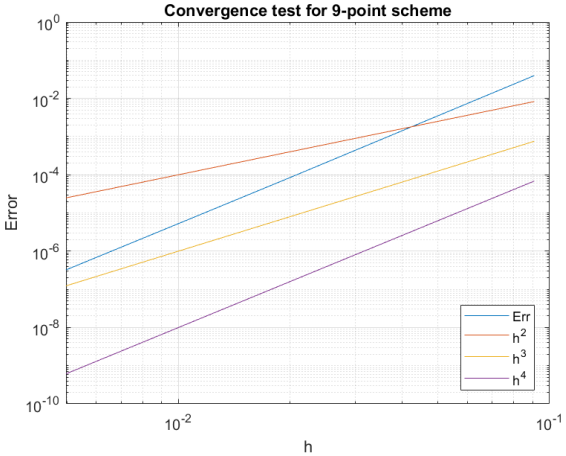


Figure 6: Global error vs. grid resolution for the 9-point Laplacian.

Figure 6 shows that our scheme achieves fourth-order accuracy, which is consistent with the theoretical expectations, demonstrating that the method provides significantly improved accuracy compared to the classical 5-point scheme.

3 Exercise 3- Iterative solvers in 2D

In this exercise we will implement a multigrid-based solver for the Poisson equation in 2D, using the same assumptions as in 2.2. In particular, we consider as spatial domain the unit square discretized with a regular grid with $m \times m$ interior points. We assume Dirichelt boundary conditions on the boundary. Thus we end up with a m^2 linear system:

$$AU = F \tag{27}$$

where the boundary conditions are included into F.
To solve this system with a multigrid-based solver, we proceed step-by-step, as follows. The related MATLAB code related to this exercise is in the folder named "Ex3".

3.1 Matrix-free 5-point Laplacian

First of all, we implemented the MATLAB function `Amult(U,m)` that computes the product $-A^hU$ without storing the matrix A^h, corresponding to the 5-point Laplacian discretization in 2D, as it performs the matrix-vector product using a matrix-free approach. This is more memory-efficient and computationally advantageous for iterative solvers like PCG, especially for large-scale problems where storing the full system matrix would be impractical. Note that the function computes $-A^hU$ since $-A^h$ is positive definite, requirement for solve iterative methods.

3.2 Solution of the system with Conjugate Gradient algorihm

We now aim to solve the discretized Poisson equation (see 2.2) using iterative methods. As a first approach, we employed MATLAB's built-in function `pcg`, which implements the Conjugate Gradient (CG) algorithm. This is an iterative method for solving linear systems that requires the matrix A to be symmetric and positive definite — which is why we apply the method to $-A$. The CG algorithm is particularly efficient for large, sparse systems, as it relies only on matrix-vector products and inner products at each iteration. This makes it especially well-suited for problems arising from the discretization of partial differential equations, such as the Poisson equation.
We then tested this solver for different mesh refinements to study its accuracy, estimate the convergence rate, and analyze the convergence history, using the MATLAB script `ex3.m`.
Knowing the true solution, we can plot the error against the mesh refinement, obtaining the following result:

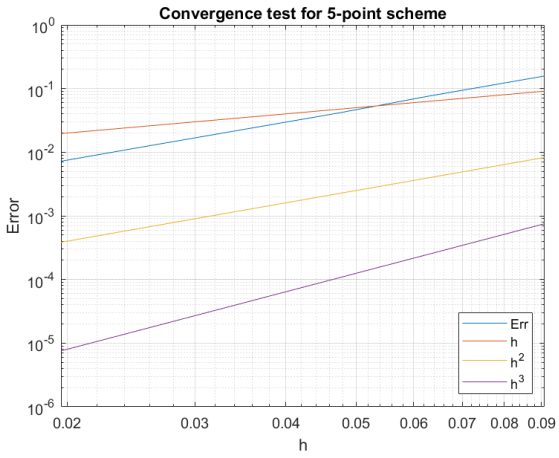


Figure 7: Order of accuracy for the 5-point scheme using PCG

We observe that the numerical scheme achieves second-order accuracy, consistent with the theoretical accuracy of the 5-point finite difference Laplacian.
To assess the performance of the solver, we also plot the convergence history of the PCG method for different grid sizes:

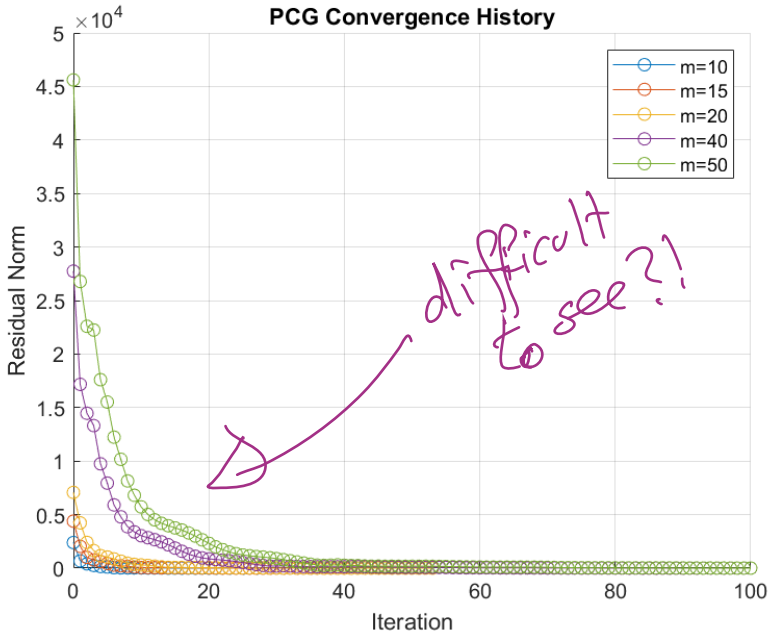


Figure 8: Convergence history of the PCG method applied to the 5-point Laplacian

The convergence rate of the PCG method can be estimated using the following theoretical expression:

$$\rho_{\text{PCG}} = 2\left(\frac{\sqrt{\kappa_2(\mathcal{A})}-1}{\sqrt{\kappa_2(\mathcal{A})}+1}\right) \tag{28}$$

From equation (28), we observe that as the condition number $\kappa_2(\mathcal{A})$ increases (i.e., the matrix becomes more ill-conditioned), the convergence factor ρ_{PCG} approaches 1, resulting in slower convergence.

By computing the convergence factor numerically in MATLAB, we find $\rho_{\text{PCG}} \approx 0.900$, which indicates that the 5-point Laplacian matrix is indeed ill-conditioned, especially for fine grids. This explains the increasingly slow convergence observed as the number of grid points increases. To address this issue, a preconditioning strategy — such as incomplete Cholesky factorization — could be implemented to improve the convergence rate of the PCG solver.

3.3 Under/over-relaxed Jacobi smoothing

We now analyse the under-relaxed Jacobi method for the 5-point Laplacian discretization in 2D and study the eigenvalues of the iteration matrix. This method will later be used as a smoother in the multigrid solver. By introducing a relaxation parameter ω, the under-relaxed Jacobi method can improve convergence compared to the standard Jacobi method by reducing the spectral radius of the iteration matrix.
To minimize the spectral radius—and thus enhance convergence—we derive a closed-form expression for

$$\max_{m/2 \le p,q \le m} \left|\lambda_{p,q}^{\omega}\right|$$

where $\lambda_{p,q}^{\omega}$ are the eigenvalues of the under-relaxed Jacobi iteration matrix.
By focusing on the range $m/2 \le p, q \le m$, we target the high-frequency components of the error, which are typically less damped by the standard Jacobi method. Finding the optimal value of ω in this range allows us to improve the smoothing properties of the method, which is particularly important in the multigrid context.
The standard Jacobi iteration matrix for the 2D Poisson problem using a 5-point Laplacian is given by:

$$G_j = I + \frac{h^2}{4}A,$$

where A is the discrete Laplacian matrix and h is the grid spacing.
The under-relaxed Jacobi iteration matrix is defined as:

$$G_\omega = (1-\omega)I + \omega G_j = (1-\omega)I + \omega\left(I + \frac{h^2}{4}A\right) = I + \frac{\omega h^2}{4}A$$

We know that the eigenvalues for G_j are given by:

$$\lambda_{p,q}(G_J) = 1 + \frac{h^2}{4}\lambda_{p,q}(A),$$

so the eigenvalues of G_ω are:

$$\lambda_{p,q}(G_\omega) = 1 + \frac{\omega h^2}{4}\lambda_{p,q}(A).$$

In 2D, the eigenvalues of the discrete Laplacian A with Dirichlet boundary conditions are:

$$\lambda_{p,q}(A) = \frac{2}{h^2}\left[\cos(p\pi h) - 1 + \cos(q\pi h) - 1\right] = \frac{2}{h^2}\left[\cos(p\pi h) + \cos(q\pi h) - 2\right].$$

Substituting into the expression for $\lambda_{p,q}(G_\omega)$, we obtain the closed-form expression:

$$\lambda_{p,q}(G_\omega) = 1 + \frac{\omega}{2}\left[(\cos(p\pi h) - 1) + (\cos(q\pi h) - 1)\right].$$

Since the convergence of the method is determined by the spectral radius of the iteration matrix, we focus on minimizing:

$$\max_{m/2 \le p,q \le m} |\lambda_{p,q}(G_\omega)| \,.$$

This allows us to find the optimal value of ω for reducing high-frequency errors. These components are the most important to smooth in a multigrid context, where the smoother primarily targets oscillatory error modes.
We can numerically determine the optimal value of ω by computing and plotting the spectral radius of G_ω over a range of $\omega \in (0, 1]$, for several grid sizes:

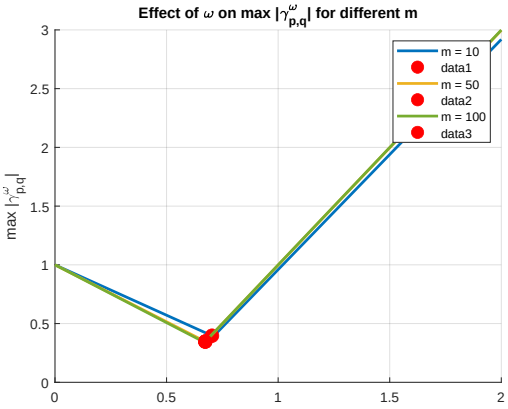


Figure 9: Maximum eigenvalues plotted against $\omega's$

m	**Optimal ω**
10	0.7102
50	0.6742
100	0.6703

Therefore we can minimize the maximum eigenvalue by choosing $\omega = 0.67$ based on the considered number of grid points. Also we see that from choosing $\omega = 0.67$ we get a maximum eigenvalue that is significantly lower than 1 and thus we expect better smoothing of high frequency errors when implementing the multigrid solver.

3.4 Smooth function

The next step to build the multigrid solver is the implementation of the `smooth` function:

```
Unew = smooth(U, omega, m, F)
```

This is a matrix-free version of the relaxed Jacobi iteration, where `U` is the current iterate, `Unew` is the new iterate, `omega` is the relaxation parameter, `m` is the number of grid points in one direction (resulting in an $m \times m$ grid), and `F` is the right-hand side vector that includes the boundary conditions.
The Jacobi iteration updates the solution using only values from the previous iteration. In the relaxed Jacobi method, the update at each point (i, j) on the grid is computed as:

$$U_{i,j}^{[k+1]} = (1-\omega)U_{i,j}^{[k]} + \frac{\omega}{4}\left(U_{i+1,j}^{[k]} + U_{i-1,j}^{[k]} + U_{i,j+1}^{[k]} + U_{i,j-1}^{[k]} - h^2 F_{i,j}\right) \tag{29}$$

where $h = \frac{1}{m+1}$ is the grid spacing, and ω is the relaxation parameter. In this case we take the optimal value of ω, found in 3.3 .
Rather than constructing the matrix A explicitly, we compute the updates by applying the stencil directly to the current iterate. This matrix-free approach avoids memory overhead and takes advantage of the regular structure of the grid. To handle boundary conditions easily, we embed the current iterate into a zero-padded array of size $(m + 2) \times (m + 2)$, effectively extending the domain by one layer of ghost cells. These ghost cells represent the fixed Dirichlet boundary values (in our case, assumed to be zero). We then compute the sum of the four neighboring values using vectorized operations, and apply the relaxation formula.

3.5 Restriction and Prolongation Operators

To enable the multigrid algorithm to correct errors efficiently across multiple scales, we must define two key grid transfer operations: the `coarsen` function, which restricts residuals from a fine grid to a coarser one, and the `interpolate` function, which prolongs corrections from the coarse grid back to the fine one.
In our implementation, we assume that the number of interior grid points satisfies $m = 2^k - 1$ for some $k > 1$. The coarse grid then has $m_c = \frac{m-1}{2}$ interior points in each spatial direction. The restriction is done via the classical *full-weighting* operator, while the prolongation uses a bilinear interpolation stencil. These are standard choices in geometric multigrid methods, ensuring both second-order accuracy and smoothing efficiency across grid levels.

Restriction: `coarsen(R, m)`

The restriction operator transfers the residual R from the fine grid to the coarse grid. In two dimensions, this is done by computing a weighted average of a 3×3 stencil around each coarse grid point, where the center point is given a weight of 4, the cardinal neighbors (north, south, east, west) a weight of 2, and the diagonal neighbors a weight of 1. The full stencil is thus normalized by a factor of 16.

Prolongation: `interpolate(Rc, m)`

The prolongation operator interpolates the coarse grid correction Rc onto the fine grid. This is achieved using a 2D bilinear interpolation, where the center point of each 2×2 block on the fine grid receives the majority of the contribution, and adjacent points receive a fraction of the value depending on their proximity. The pattern matches the adjoint of the full-weighting stencil and ensures proper correction of smooth error components.

3.6 Vcycle

At the core of the multigrid method lies the `V-cycle` algorithm, a recursive procedure designed to efficiently reduce errors across multiple grid levels. It is particularly powerful because it combines relaxation on fine grids with coarse-grid corrections, thereby targeting both high-frequency and low-frequency error components. The V-cycle implementation in this project follows the standard recursive structure and consists of the following steps:

1. Pre-smoothing: A fixed number of relaxed Jacobi iterations are applied to dampen high-frequency components of the error in the current solution U.
2. Residual computation: The residual is computed as $R = F - AU$, where A is the discretized Laplacian operator. This step quantifies the error in the current approximation.
3. Restriction to coarse grid: The residual is transferred to a coarser grid using the `coarsen` function. The coarse grid has $(m_c \times m_c)$ unknowns, with $m_c = \frac{m-1}{2}$.
4. Recursive coarse grid correction: The V-cycle is applied recursively to solve the error equation $Ae = -R$ on the coarser grid. When the coarsest level is reached ($m = 1$), the linear system is solved directly.
5. Prolongation to fine grid: The error correction computed on the coarse grid is interpolated back to the fine grid using bilinear interpolation via the `interpolate` function.
6. Solution update: The current solution is corrected by subtracting the interpolated error, i.e. $U \leftarrow U - E$.
7. Post-smoothing: A final set of Jacobi iterations is performed to remove high-frequency errors reintroduced by interpolation.

This hierarchical process ensures that all frequency components of the error are efficiently addressed at their corresponding grid scales. The recursion depth of the V-cycle is governed by the parameter $m = 2^k - 1$, ensuring that each level is exactly half the size of the previous one. The algorithm terminates when the relative residual norm $\|R\|_2/\|F\|_2$ falls below a user-defined tolerance, which in our case is set to 10^{-6}.
To verify the correctness and efficiency of our multigrid implementation, we solve the Poisson problem with the exact solution $u(x, y) = e^{\pi x}\sin(\pi y) + \frac{1}{2}(xy)^2$, for which the corresponding right-hand side is $f(x, y) = x^2 + y^2$. The computation is carries out on a grid of size $m = 63$, and the stopping criterion is set to a relative residual below 10^{-6}.The number of pre- and post-smoothing steps is set to 3.
The method converges in 8 iterations, and the computed solution exhibits the expected smooth behaviour. A 3D plot of the numerical solution is shown in Figure 10.

3.6.1 Unexpected effects of the relaxation parameter on the Multigrid solver

While implementing the V-cycle multigrid solver, and fixing some errors in the code we tested it using the relaxation parameter $\omega = \frac{3}{4}$. Using the ω that minimizes the eigenvalues of G_ω and for $m = 2^6 - 1$ gridpoints the code converged in 8 "outer" multigrid iterations, while using $\omega = \frac{3}{4}$ converged in 7.
While this isn't a big difference, we desired to look further into it. Performing some simple tests, we found that we converged in 7 iterations for at least ω' s between 0.72 and 0.93.
Looking at the process of finding the optimal ω, we observe that, due to the discrete nature of the method, the optimal ω increases for smaller values of m. Testing for the lower values expected when moving to coarser grids, we see the following pattern:

Grid size m	1	3	7	15	31
Optimal $\omega \approx$	1	0.738	0.684	0.67	0.667

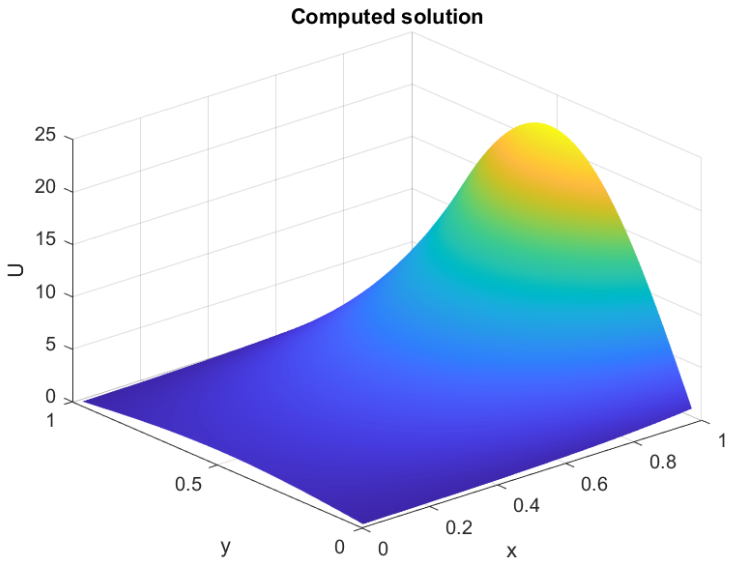


Figure 10: Computed solution $U(x, y)$ using the multigrid V-cycle for $m = 63$.

We observe that for low values of m, the optimal ω approaches 1 according to the implemented method, while as m increases, the optimal ω tends to decrease towards $\omega = 2/3$. ok
Although we are not fully confident about the extent to which this behavior reflects the true dynamics of the under-relaxed Jacobi method, the results suggest that a larger ω may be more effective at the lower levels of the V-cycle. Moreover, the findings indicate that, at least within the parameter ranges we have tested, assigning greater weight to the efficiency of the lower levels might enhance overall convergence. ok
We have not yet succeeded in deriving a mathematical explanation for the variation in optimal ω, nor have we found a general way to compute it analytically. However, these questions may be worth exploring in future work. good

3.7 Dependence of the convergence on the number of grid points

To assess the scalability and convergence properties of the multigrid V-cycle method, we perform a numerical experiment by fixing the tolerance and varying the number of grid points. Specifically, we consider grid sizes of the form $m = 2^k - 1$, with $k = 3, 4, \ldots, 9$, and we monitor the number of outer V-cycle iterations required to reduce the relative residual below 10^{-6}.
In each run, we compute the optimal relaxation parameter ω that minimizes the high-frequency amplification factor, as explained in the theoretical section.
The table below summarizes the results:

Grid size m	**Grid points** $n = m^2$	**V-cycle iterations**
$2^3 - 1 = 7$	49	8
$2^4 - 1 = 15$	225	8
$2^5 - 1 = 31$	961	8
$2^6 - 1 = 63$	3969	8
$2^7 - 1 = 127$	16129	8
$2^8 - 1 = 255$	65025	8
$2^9 - 1 = 511$	261121	8

good!
grid-independent convergence!

A key observation from this study is that the number of V-cycle iterations required to reach the desired tolerance remains constant as the grid is refined. This confirms the theoretical property of grid-independent convergence, which distinguishes multigrid methods from classical iterative solvers whose performance typically degrades on finer meshes. The result highlights the efficiency and scalability of the multigrid V-cycle, making it well suited for solving large-scale problems with optimal computational cost.

References

[1] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007.

Excellent work!
Allan