Vera Xie
Ronald Nap
3L = F 3:30 – 4:20 PM

Problem Set 6
Math 146 Spring 2023
Due: Monday, April 10, 11:59 PM

Note: Numbers 1-2,4a-b are "pen and paper" problem for which you should show work. Numbers 3,4c-7 are Matlab problems that require code submissions. All answers should be submitted in a write-up.

1. (15 points) Show that the error in the centered difference approximation of the first derivative is second order. In other words, for $D_0u(x) = \dfrac{u(x+h) - u(x-h)}{2h}$, show that $E_0 = D_0u(x) - u'(x) = \mathcal{O}(h^2)$.

2. (10 points) Show that finite difference approximation of the second derivative given by

$$u''(x) \approx D_2u(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \tag{1}$$

   is also second-order accurate.
   (Note: This also means that the five-point finite difference approximation of the 2-D Laplacian (that we went over in class) is also second-order accurate.)

3. (20 points) Write a Matlab function to use 5-point 2-D Laplacian to estimate $\Delta u(x, y)$ at *one* point, (x, y). Your function should take as inputs the *function* u, as well as x,y, and h. Use it to approximate the value of $\Delta u(1, 2)$ for $u = e^{xy} + x^2$. Use `u=@(x,y) exp(x*y)+x^2` to define this function in your script. Perform a refinement study, using relative errors, for values of $h = 2^{-2}, 2^{-3}, ..., 2^{-12}$ and demonstrate that the finite difference operator is second-order accurate. Explain how you have demonstrated this.

4. Consider the PDE given by $u''(x) = \sin(\pi x)$ on $[0, 1]$ such that $u(0) = 0$ and $u(1) = 1$.

   (a) (5 points) Solve this boundary value problem analytically.

   (b) (10 points) Using the second-order approximation of the second derivative, write the associated linear algebra problem that can be solved to approximate the solution to this PDE for a given value of h. For $[a, b]$, use gridpoints located at $x_0 = a, x_1 = a + h, x_2 = a + 2h, ...x_N = b - h, x_{N+1} = b$. Therefore, your unknowns will be $u_1, ..., u_N$, and you will have $h = \dfrac{b-a}{N+1}$.

   (c) (20 points) Solve this PDE in Matlab with Matlab's backslash for values of $N = 2^3, 2^4, ..., 2^{12}$. Perform a refinement study, using the relative error of your solution, in the max norm. In your refinement study log log plot, use N for the x-axis. (This is more natural than the use of h that we must use for problem (3). This is because convergence shows error going down as you increase N, left to right.) What order of convergence do you get? Also plot your approximate solutions and the true solution.

5. Consider the PDE given by $\Delta u = -e^{(x-0.25)^2+(y-0.25)^2}$ on $[0, 1] \times [0, 1]$, with *homogeneous Dirichlet boundary conditions*, which means that $u = 0$ on the border around the square domain. (Note: This simplifies the right-hand-side of the linear algebra problem that results from using finite differences.)

(a) (10 points)If we use finite differences, as described in 2-D in class, what is the right-hand-side of the resulting linear algebra equation? Create this in Matlab for $N = 2^2$. Note: For the Matlab part, I recommend finding the values in a matrix first, using the grid given by :
`xg=h*(1:N)+xmin; yg=h*(1:N)+ymin; [xg,yg]=ndgrid(xg,yg);`
Then use the `reshape` command. Note: Again, your *interior* gridpoints, for which your solution is unknown should run from $x_1 = a + h, x_2 = a + 2h, ...x_N = b - h$ an similarly for y.

(b) (25 points) A matlab function, `lap2d.m`, is provided in Catcourses which gives the 2-D discrete Laplacian matrix, without the $1/h^2$ factor. Use this and part (a), and solve the system using `backslash` to solve the system for $N = 2^3 - 1, 2^4 - 1, ...2^{10} - 1$.

Provide a refinement study. Instead of using relative error, use the max norm of the difference of successive solutions. To do this, find the difference between your solutions at $N = 2^3$ and $N = 2^4$ and then the difference of your solutions at $N = 2^4$ and $N = 2^5$, etc. For each of those pairs, evaluate these differences on the coarser mesh. Hints: You will find it easier to restrict the finer solution to the coarser grid if you first reshape it back into a matrix. Then, the "-1" in the values of N will allow you to easily do this restriction.

What order of convergence do you see? Also use the command `mesh` to provide a 3-D plot of the solution on the finest grid.

6. (a) (15 points) Write a Matlab function to solve $A\mathbf{x} = \mathbf{b}$ using the Jacobi iterative method. It should take as an input A,$\mathbf{b}$, an initial guess $\mathbf{x}_0$, a tolerance, and a maximum number of iterations (optional). It should give as output the approximate solution $\mathbf{x}$, and the number of iterations performed. Use the *relative* stopping criteria of your choice. What did you use? Note: In class, I wrote it in component form, and therefore used a loop over i. Try to code it without such a loop in order to increase efficiency

(b) (15 points) Write a Matlab function to solve $A\mathbf{x} = \mathbf{b}$ using the Gauss-Seidel iterative method. It should take as an input A,$\mathbf{b}$, an initial guess $\mathbf{x}_0$, a tolerance, and a maximum number of iterations (optional). It should give as output the approximate solution $\mathbf{x}$, and the number of iterations performed. Use the same stopping criteria as you did for part (a). Make sure to not use any built-in Matlab functions to invert any matrices

(c) (25 points) Use your functions to solve $A\mathbf{x} = \mathbf{b}$ for A, a 1000×1000 matrix.

In class, we discussed how/when you would know that these Splitting methods would converge using the eigenvalues, but we did not discuss what matrices you would therefore get convergence for. To ensure that you are using a matrix for which these methods will converge, make a matrix that is *strictly diagonally dominant*, which means that, for every row, the magnitude of the diagonal entry is larger than the sum of the magnitudes of the other entries in the row, ie

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \tag{2}$$

Do this by creating a random matrix of numbers from 0 to 10 and then changing the diagonal entries to ensure the above relationship holds.

What tolerance did you use? Time these two methods, as well as using LU decomposition with pivoting and Matlab's backslash. Compare the relative differences, in max norm, between your solutions and Matlab's in order to validate your codes. Also report the number of iterations the methods required. Comment on your results.

7. (30 points) The function `FDjacobi_2D.m` gives the Jacobi method for this finite difference implementation of the Poisson equation, and `FDGaussSeidel_2D.m` gives one version of the Gauss-Seidel method for this finite difference implementation of the Poisson equation. What does one step of the Gauss-Seidel method look like (using the same notation as the end of page 4 of Lecture 17 notes)? What is the difference between it and Jacobi method?

   Use the Gauss-Seidel method to solve the PDE from number (5) for $N = 2^3, 2^4, ...2^6$ with a tolerance of 10^{-8}.

   The function `FDSOR_2D.m` gives a related method called SOR, or Successive Over-Relaxation. Use this to solve the PDE from number (5) for $N = 2^3, 2^4, ...2^9$ with a tolerance of 10^{-8}. Note: The last one is likely to take about

   For GS and SOR, how many iterations were required for each solve? Comment on your results. Also comment on why we would solve it this way instead of the way we did it in number (5).

1. (15 points) Show that the error in the centered difference approximation of the first derivative is second order. In other words, for $D_0u(x) = \dfrac{u(x+h) - u(x-h)}{2h}$, show that $E_0 = D_0u(x) - u'(x) = \mathcal{O}(h^2)$.

(1) CENTERED DIFFERENCE APPROX. of the 1ST derivative : $D_0u(x) = \dfrac{u(x+h) - u(x-h)}{2h}$

↳ Show that the ERROR is 2ND-ORDER : $E_0 = D_0u(x) - u'(x) = O(h^2)$

⟶ TAYLOR'S SERIES EXPANSIONS

↳ $u(x+h) = u(x) + hu'(x) + \dfrac{h^2}{2!}u''(x) + \dfrac{h^3}{3!}u'''(\xi_1)$

↳ $u(x-h) = u(x) - hu'(x) + \dfrac{h^2}{2!}u''(x) - \dfrac{h^3}{3!}u'''(\xi_2)$

⟶ $u(x+h) - u(x-h) = \left[u(x) + hu'(x) + \dfrac{h^2}{2!}u''(x) + \dfrac{h^3}{3!}u'''(\xi_1)\right] - \left[u(x) - hu'(x) + \dfrac{h^2}{2!}u''(x) - \dfrac{h^3}{3!}u'''(\xi_2)\right]$

$= \cancel{u(x)} + hu'(x) + \cancel{\dfrac{h^2}{2!}u''(x)} + \dfrac{h^3}{3!}u'''(\xi_1) - \cancel{u(x)} + hu'(x) - \cancel{\dfrac{h^2}{2!}u''(x)} + \dfrac{h^3}{3!}u'''(\xi_2)$

$= 2hu'(x) + \dfrac{h^3}{3!}\left[u'''(\xi_1) + u'''(\xi_2)\right]$

$= 2hu'(x) + \dfrac{h^3}{6}\left[u'''(\xi_1) + u'''(\xi_2)\right]$

↳ $D_0u(x) = \dfrac{u(x+h) - u(x-h)}{2h} = \left(2hu'(x) + \dfrac{h^3}{6}\left[u'''(\xi_1) + u'''(\xi_2)\right]\right) \cdot \dfrac{1}{2h}$

$= u'(x) + \dfrac{h^2}{12}\left[u'''(\xi_1) + u'''(\xi_2)\right]$

⟶ $E_0 = D_0u(x) - u'(x) = \left(\cancel{u'(x)} + \dfrac{h^2}{12}\left[u'''(\xi_1) + u'''(\xi_2)\right]\right) - \cancel{u'(x)}$

$= \dfrac{h^2}{12}\left[u'''(\xi_1) + u'''(\xi_2)\right]$

$= O(h^2)$ ✓

⟹ ∴ $D_0u(x) = \dfrac{u(x+h) - u(x-h)}{2h}$ is a CENTERED DIFFERENCE APPROX. of $u'(x)$, whose ERROR is of the 2ND-ORDER : $E_0 = D_0u(x) - u'(x) = O(h^2)$

2. (10 points) Show that finite difference approximation of the second derivative given by

$$u''(x) \approx D_2 u(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \tag{1}$$

is also second-order accurate.
(Note: This also means that the five-point finite difference approximation of the 2-D Laplacian (that we went over in class) is also second-order accurate.)

(2) FINITE DIFFERENCE APPROX. of the 2nd derivative : $u''(x) \approx D_2 u(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$ is also 2nd-ORDER ACCURATE

NOTE : 5-point FINITE DIFFERENCE APPROX. of the 2D LAPLACIAN is also 2nd-ORDER ACCURATE

→ TAYLOR'S SERIES EXPANSIONS

$$u(x+h) = u(x) + hu'(x) + \frac{h^2}{2!}u''(x) + \frac{h^3}{3!}u'''(x) + \frac{h^4}{4!}u^{(4)}(\xi_1)$$

$$u(x-h) = u(x) - hu'(x) + \frac{h^2}{2!}u''(x) - \frac{h^3}{3!}u'''(x) + \frac{h^4}{4!}u^{(4)}(\xi_2)$$

$$u(x+h) + u(x-h) = \left[u(x) + hu'(x) + \frac{h^2}{2!}u''(x) + \frac{h^3}{3!}u'''(x) + \frac{h^4}{4!}u^{(4)}(\xi_1)\right] + \left[u(x) - hu'(x) + \frac{h^2}{2!}u''(x) - \frac{h^3}{3!}u'''(x) + \frac{h^4}{4!}u^{(4)}(\xi_2)\right]$$

$$= 2u(x) + \frac{2h^2}{2!}u''(x) + \frac{h^4}{4!}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]$$

$$= 2u(x) + \frac{2h^2}{2}u''(x) + \frac{h^4}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]$$

$$= 2u(x) + h^2u''(x) + \frac{h^4}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]$$

$$u(x+h) - 2u(x) + u(x-h) = \left(2u(x) + h^2u''(x) + \frac{h^4}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]\right) - 2u(x)$$

$$= h^2u''(x) + \frac{h^4}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]$$

$$D_2u(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} = \left(h^2u''(x) + \frac{h^4}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]\right) \cdot \frac{1}{h^2}$$

$$= u''(x) + \frac{h^2}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]$$

$$E_2 = D_2u(x) - u''(x) = \left(u''(x) + \frac{h^2}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]\right) - u''(x)$$

$$= \frac{h^2}{24}\left[u^{(4)}(\xi_1) + u^{(4)}(\xi_2)\right]$$

$$= O(h^2) \checkmark$$

$\Longrightarrow \therefore$ $D_2u(x) = \dfrac{u(x+h) - 2u(x) + u(x-h)}{h^2}$ is a FINITE DIFFERENCE APPROX. of $u''(x)$,

whose ERROR is of the 2ND-ORDER : $E_2 = D_2u(x) - u''(x) = O(h^2)$

3. (20 points) Write a Matlab function to use 5-point 2-D Laplacian to estimate $\Delta u(x, y)$ at *one* point, (x, y). Your function should take as inputs the *function* u, as well as x,y, and h. Use it to approximate the value of $\Delta u(1, 2)$ for $u = e^{xy} + x^2$. Use `u=@(x,y) exp(x*y)+x^2` to define this function in your script. Perform a refinement study, using relative errors, for values of $h = 2^{-2}, 2^{-3}, ..., 2^{-12}$ and demonstrate that the finite difference operator is second-order accurate. Explain how you have demonstrated this.

(3) MATLAB

— 5-point 2D LAPLACIAN to estimate $\Delta u(x,y)$ at 1 POINT (x,y)

└→ INPUTS : u , function
x
y
h

$(x,y) = (1,2)$

To approximate $\Delta u(1,2)$

for $u = e^{xy} + x^2$

[use `u = @(x,y) exp(x*y) + x^2`]

└→ Perform a REFINEMENT STUDY

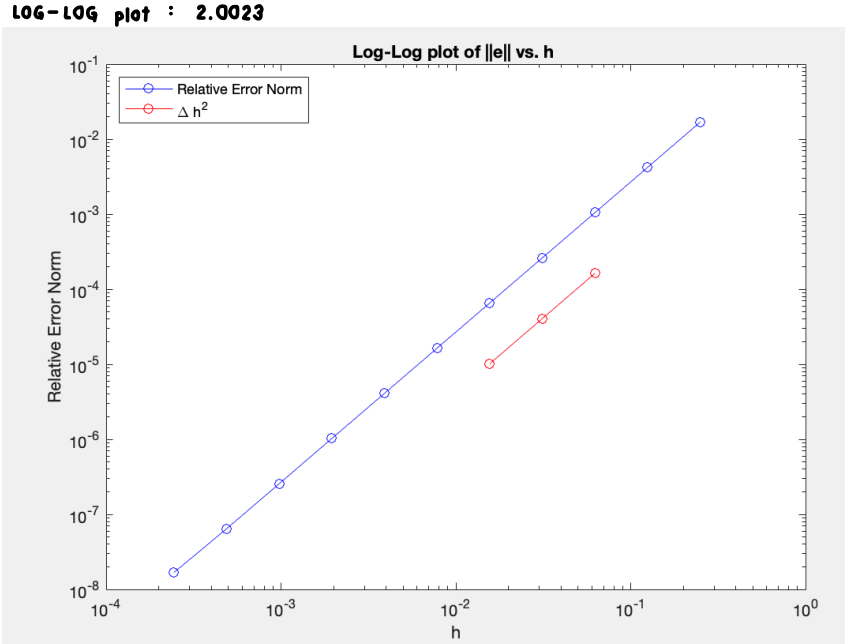
— using RELATIVE ERRORS for values of $h = 2^{-2}, 2^{-3}, \ldots, 2^{-12}$ [$h = 2^{-n}$ from $n = 2$ TO 12]

└→ & Demonstrate that the FINITE DIFFERENCE OPERATOR is 2ND-ORDER ACCURATE

└→ EXPLAIN how you have demonstrated this

└→ In your refinement study LOG-LOG plot, use h for the x-axis

$\Longrightarrow$ SLOPE of LOG-LOG plot : 2.0023



To demonstrate that the finite difference operator of the 5-point finite difference approximation of the 2D laplacian is 2nd-order accurate, a refinement study was performed in 2-D with respect to the max-norm. By comparing the approximated solutions to MATLAB's backslash solution, we determined the relative difference for values of h from 2^{-2} to 2^{-12}. As the 5-point finite-difference approximation of the 2D Laplacian is a 2nd-order approximation, the approximated solution to estimate $\Delta u(x, y)$ at a point (x,y) will be 2nd-order accurate. Therefore, we expect the order of the convergence rate to be O(h^2). Thus, we expect the log-log plot of the error norm vs. the values of h to resemble a line with a slope of 2. This can be represented as $y = C(hvals)^p$, where p = 2 to be 2nd-order.

As we expect the solution to be 2nd-order, we plotted a small segment using the equation, $y = C(hvals)^p$, where p = 2 to be 2nd-order. We plotted the small segment using the 3rd and 5th index of our values of h. We can observe from the log-log plot of the error norm vs. the values of h that the solution converges to the 2nd-order as it is parallel to the line we expected. The actual slope of the log-log plot is 2.0023, which is relatively close to 2. This confirms the order of convergence is 2nd-order and thus, the finite difference operator is 2nd-order accurate.

4. Consider the PDE given by $u''(x) = \sin(\pi x)$ on $[0, 1]$ such that $u(0) = 0$ and $u(1) = 1$.

   (a) (5 points) Solve this boundary value problem analytically.

   (b) (10 points) Using the second-order approximation of the second derivative, write the associated linear algebra problem that can be solved to approximate the solution to this PDE for a given value of h. For $[a, b]$, use gridpoints located at $x_0 = a, x_1 = a + h, x_2 = a + 2h, ...x_N = b - h, x_{N+1} = b$. Therefore, your unknowns will be $u_1, ..., u_N$, and you will have $h = \dfrac{b - a}{N + 1}$.

   (c) (20 points) Solve this PDE in Matlab with Matlab's backslash for values of $N = 2^3, 2^4, ..., 2^{12}$. Perform a refinement study, using the relative error of your solution, in the max norm. In your refinement study log log plot, use N for the x-axis. (This is more natural than the use of h that we must use for problem (3). This is because convergence shows error going down as you increase N, left to right.) What order of convergence do you get? Also plot your approximate solutions and the true solution.

(4) PDE : $u''(x) = \sin(\pi x)$ on $[0,1]$, s.t. $u(0) = 0$ & $u(1) = 1$

a.) Solve the BVP analytically

$$u''(x) = \sin(\pi x) \implies \frac{d^2u}{dx^2} = \sin(\pi x)$$

$$\int d^2u = \int \sin(\pi x)\, dx^2 \longrightarrow \int du = \int -\frac{1}{\pi}\cos(\pi x) + C_1\, dx$$

$$\therefore \int du = -\frac{1}{\pi}\int \cos(\pi x) + C_1 \; dx$$

$$\downarrow$$

$$u = -\frac{1}{\pi}\left[\frac{1}{\pi}\sin(\pi x) + C_1 x + C_2\right]$$

$$= -\frac{1}{\pi^2}\sin(\pi x) \underbrace{-\frac{1}{\pi}C_1 x}_{\text{let } C_1 = -\frac{1}{\pi}C_1} \underbrace{-\frac{1}{\pi}C_2}_{\text{let } C_1 = -\frac{1}{\pi}C_1}$$

$$\boxed{u(x) = -\frac{1}{\pi^2}\sin(\pi x) + C_1 x + C_2}$$

$\hookrightarrow$ $u(0) = -\frac{1}{\pi^2}\cancel{\sin(0)} + \cancel{C_1[0]} + C_2 = 0 \implies C_2 = 0$ ($\sin(0) = 0$)

$\hookrightarrow$ $u(1) = -\frac{1}{\pi^2}\cancel{\sin(\pi)} + C_1[1] + \cancel{C_2} = 1 \implies C_1 = 1$ ($\sin(\pi) = 0$, $C_2 = 0$)

$\longrightarrow$ $u(x) = -\frac{1}{\pi^2}\sin(\pi x) + C_1 x + C_2$

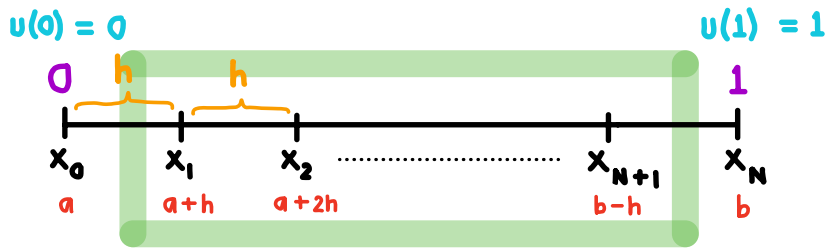
$= -\frac{1}{\pi^2}\sin(\pi x) + [1]x + \cancel{[0]}$ $\implies$ $\boxed{u(x) = x - \frac{\sin(\pi x)}{\pi^2}}$

b.) Using the 2ND-ORDER APPROX. for the 2ND DERIVATIVE, write the associated LINEAR ALGEBRA problem that can be used to approx. the solution to this PDE for a given value of h

— For $[a, b]$, use the gridpoints located @ $x_0 = a$, $x_1 = a + h$, $x_2 = a + 2h$, ..., $x_N = b - h$, $x_{N+1} = b$

$\hookrightarrow$ ∴ your unknowns will be $u_1, \ldots, u_N$ & $h = \frac{b-a}{N+1}$

$\longrightarrow$ $u''(x) = \sin(\pi x)$ & B.C.s : $u(0) = 0$, $u(1) = 1$



$\longrightarrow$ Let u_j be the approx. to u at x_j : $u_j \approx u(x_j)$

→ Then, the PDE can be approx. at x_j : $f(j) = \dfrac{U_{j+1} - 2U_j + U_{j-1}}{h^2}$

→ For $j = 1, \ldots, N$ ⟹ These are our discrete equations for UNKNOWNS $U_1, \ldots, U_N$

∴ it is now a LINEAR ALGEBRA PROBLEM ! [once the numerical solution of the PDE becomes DISCRETIZED]

→ @ $j = 1$ ⟹ $f_1 = \dfrac{U_2 - 2U_1 + \overbrace{U_0}^{=0}}{h^2} = \dfrac{U_2 - 2U_1}{h^2}$ } $f_1 - \dfrac{\overbrace{U_0}^{=0}}{h^2} = f_1$

@ ENDPOINTS (U_0 OR U_{N+1}), subtract $\frac{\alpha}{h^2}$ OR $\frac{\beta}{h^2}$ on BOTH sides

→ @ $j = 2$ ⟹ $f_2 = \dfrac{U_3 - 2U_2 + U_1}{h^2}$ } f_2

→ @ $j = 3$ ⟹ $f_3 = \dfrac{U_4 - 2U_3 + U_2}{h^2}$ } f_3

→ @ $j = N$ ⟹ $f_N = \dfrac{\overbrace{U_{N+1}}^{=1} - 2U_N + U_{N-1}}{h^2} = \dfrac{1 - 2U_N + U_{N-1}}{h^2}$ } $f_N - \dfrac{\overbrace{U_{N+1}}^{=1}}{h^2} = f_N - \dfrac{1}{h^2}$

⟹ LHS : $A\underline{U} = \dfrac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & & 0 \\ 0 & 1 & -2 & 1 & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & & 0 & 1 & -2 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_N \end{bmatrix}$

⟹ RHS : $\underline{b} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \\ f_N - \frac{1}{h^2} \end{bmatrix}$

→ $\dfrac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & & 0 \\ 0 & 1 & -2 & 1 & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & & 0 & 1 & -2 \end{bmatrix}\begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_N \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N - \frac{1}{h^2} \end{bmatrix}$

II

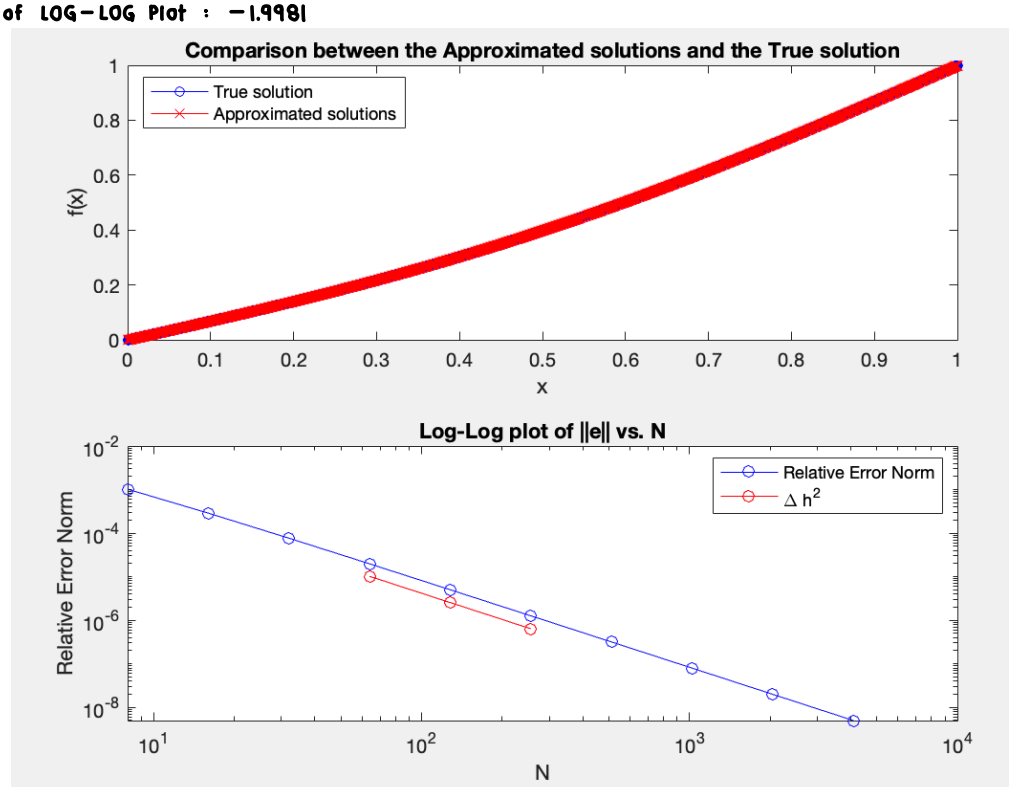
$$\Downarrow$$

$$\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & & 1 & -2 & 1 \\ 0 & & \cdots & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \end{bmatrix} = \begin{bmatrix} \sin(\pi h) \\ \sin(2\pi h) \\ \vdots \\ \sin(n\pi h) \\ \sin(n\pi h) - \frac{1}{h^2} \end{bmatrix}$$

C.) MATLAB

- use MATLAB's backslash for values of $N = 2^3, 2^4, \ldots, 2^{12}$
- Perform a refinement study, using the rel. error of your solution, in MAX-NORM
  - In your refinement study LOG-LOG plot, use **N for the x-axis**
    - Convergence shows error going DOWN as you INCREASE N
- What ORDER of convergence do you get ?
- PLOT your APPROX. solutions & the TRUE solution

$\Longrightarrow$ SLOPE of LOG-LOG Plot : -1.9981



We can observe from the first subplot that our approximated solutions converge to the true solution. A refinement study was performed in 1-D with respect to the max-norm in order to determine the rate of convergence of the global solution. As we are using a 2nd-order approximation of the 2nd derivative, the approximated solution to the PDE is 2nd-order accurate. Therefore, we expect the order of the convergence rate to be O(h^2), where $h = \frac{b-a}{N+1}$. Thus, we expect the log-log plot of the error norm vs. the values of N to resemble a line with a slope of -2. The 2nd subplot shows the log-log plot of the error norm vs. the values of N. We can observe from this plot that the solution converges to the 2nd-order. The actual slope of the log-log plot is -1.9981, which is relatively close to -2. This confirms the order of convergence is 2nd-order

5. Consider the PDE given by $\Delta u = -e^{(x-0.25)^2+(y-0.25)^2}$ on $[0,1] \times [0,1]$, with *homogeneous Dirichlet boundary conditions*, which means that $u = 0$ on the border around the square domain. (Note: This simplifies the right-hand-side of the linear algebra problem that results from using finite differences.)

(a) (10 points)If we use finite differences, as described in 2-D in class, what is the right-hand-side of the resulting linear algebra equation? Create this in Matlab for $N = 2^2$. Note: For the Matlab part, I recommend finding the values in a matrix first, using the grid given by :
`xg=h*(1:N)+xmin; yg=h*(1:N)+ymin; [xg,yg]=ndgrid(xg,yg);`
Then use the `reshape` command. Note: Again, your *interior* gridpoints, for which your solution is unknown should run from $x_1 = a + h, x_2 = a + 2h, ...x_N = b - h$ an similarly for y.

(b) (25 points) A matlab function, `lap2d.m`, is provided in Catcourses which gives the 2-D discrete Laplacian matrix, without the $1/h^2$ factor. Use this and part (a), and solve the system using `backslash` to solve the system for $N = 2^3 - 1, 2^4 - 1, ...2^{10} - 1$.

Provide a refinement study. Instead of using relative error, use the max norm of the difference of successive solutions. To do this, find the difference between your solutions at $N = 2^3$ and $N = 2^4$ and then the difference of your solutions at $N = 2^4$ and $N = 2^5$, etc. For each of those pairs, evaluate these differences on the coarser mesh. Hints: You will find it easier to restrict the finer solution to the coarser grid if you first reshape it back into a matrix. Then, the "-1" in the values of N will allow you to easily do this restriction.

What order of convergence do you see? Also use the command `mesh` to provide a 3-D plot of the solution on the finest grid.

⑤ PDE given by $\Delta U = -e^{(x-0.25)^2 + (y+0.25)^2}$ on $[0,1] \times [0,1]$

w/ HOMOGENEOUS DIRCHLET CONDITIONS : $u = 0$ on the BORDER around the square domain

└→ NOTE : simplifies the RHS of the linear algebra problem that results from using FINITE DIFFERENCES

a.) If we use FINITE DIFFERENCES in 2D, what is the RHS of the resulting LINEAR ALGEBRA EQN. ?

— Create this in MATLAB for $N = 2^2$

└→ NOTE : First, find the values in a MATRIX, using the GRID given by :

```
xg = h*(1:N) + xmin ;
yg = h*(1:N) + ymin ;
[xg,yg] = ndgrid(xg,yg) ;
```

Then, use the **reshape** command

└→ NOTE : The INTERIOR GRIDPOINTS, for which your solution should run from

$$x_1 = a+h\ ,\ x_2 = a+2h\ ,\ \ldots,\ x_N = b-h$$

& similarly for y : $y_1 = a+h\ ,\ y_2 = a+2h\ ,\ \ldots,\ y_N = b-h$

⟹ RHS of the resulting LINEAR ALGEBRA EQN. :

$$\underline{b} = \begin{bmatrix} 1.5223 \\ 0.3072 \\ -0.9080 \\ -2.1231 \\ 1.8594 \\ 0.3752 \\ -1.1090 \\ -2.5932 \\ 2.2710 \\ 0.4583 \\ -1.3545 \\ -3.1673 \\ 2.7739 \\ 0.5597 \\ -1.6544 \\ -3.8686 \end{bmatrix}$$

b.) MATLAB function, lap2d.m , is provided

└→ gives the 2D discrete Laplacian matrix w/o the $1/h^2$ factor

——→ Use this & part (a) ⟹ solve the system using BACKSLASH for $N = 2^3-1,\ 2^4-1,\ \ldots,\ 2^{10}-1$

——→ Provide a REFINEMENT STUDY

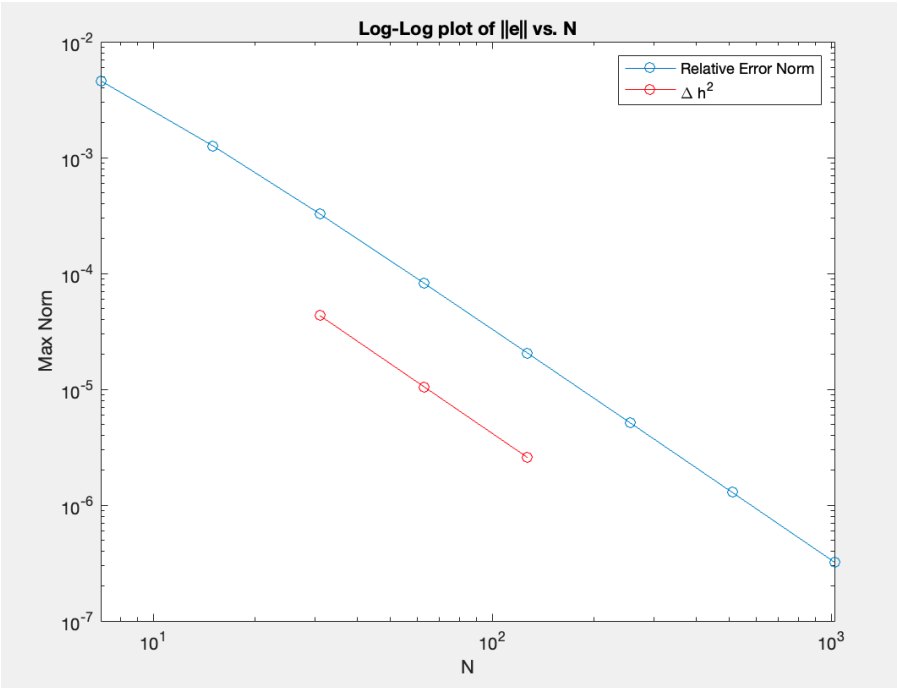
└→ Instead of using rel. error, use the MAX NORM of the DIFFERENCE OF SUCCESSIVE SOLUTIONS

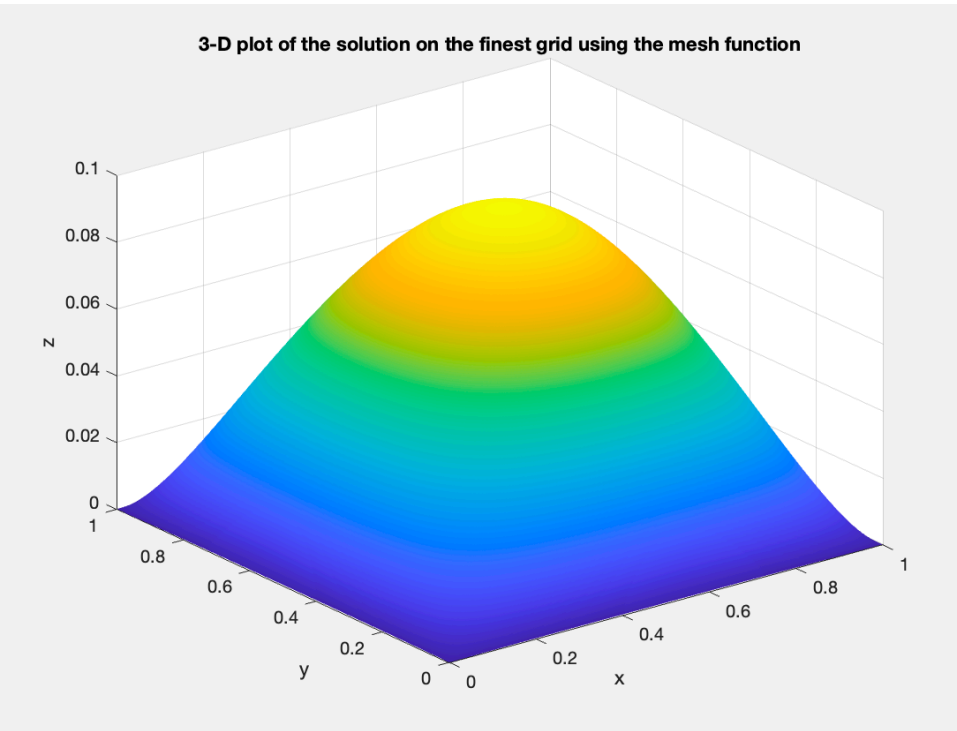
└→ Find the difference b/w your solutions @ $N = 2^3$ & $N = 2^4$

- & Then, find the difference of your solutions @ $N = 2^4$ & $N = 2^5$, etc.
  - For each of these pairs, evaluate these differences on the COARSER mesh

HINTS :
- Easier to restrict the finer solution to the coarser grid if you first reshape it back into a matrix
- Then, the "-1" in the values of N will allow you to easily do this restriction

- What ORDER of convergence do you get ?
- Also, use the command **mesh** to provide a 3D plot of the solution on the finest grid





Based on the results of the refinement study, we can observe that the size of our error with respect to the max-norm goes down as N increases. It can also be noted that the rate of convergence is quadratic. To find the order of convergence we can use the equation, $y = \frac{C}{(Nvals)^p}$, where p is the expected order of convergence, C is an artificial constant, Nvals is the number of interior gridpoints, and y is the error. From the plot, we are able to distinguish that the slope is approximately -2, hence we plug in $p = 2$. When using $p = 2$, we find the actual slope of the log-log plot to be -1.9661. This may not be exactly -2, likely because of round-off error, but since it is relatively close, we have confidence that our method is indeed 2nd-order accurate. Additionally, we plotted a segment using the 4th and 6th index of our N points and we can see that our segment is indeed parallel meaning the order of convergence is 2.

6. (a) (15 points) Write a Matlab function to solve $A\mathbf{x} = \mathbf{b}$ using the Jacobi iterative method. It should take as an input A,$\mathbf{b}$, an initial guess $\mathbf{x}_0$, a tolerance, and a maximum number of iterations (optional). It should give as output the approximate solution $\mathbf{x}$, and the number of iterations performed. Use the *relative* stopping criteria of your choice. What did you use? Note: In class, I wrote it in component form, and therefore used a loop over i. Try to code it without such a loop in order to increase efficiency

   (b) (15 points) Write a Matlab function to solve $A\mathbf{x} = \mathbf{b}$ using the Gauss-Seidel iterative method. It should take as an input A,$\mathbf{b}$, an initial guess $\mathbf{x}_0$, a tolerance, and a maximum number of iterations (optional). It should give as output the approximate solution $\mathbf{x}$, and the number of iterations performed. Use the same stopping criteria as you did for part (a). Make sure to not use any built-in Matlab functions to invert any matrices

   (c) (25 points) Use your functions to solve $A\mathbf{x} = \mathbf{b}$ for A, a 1000×1000 matrix.

   In class, we discussed how/when you would know that these Splitting methods would converge using the eigenvalues, but we did not discuss what matrices you would therefore get convergence for. To ensure that you are using a matrix for which these methods will converge, make a matrix that is *strictly diagonally dominant*, which means that, for every row, the magnitude of the diagonal entry is larger than the sum of the magnitudes of the other entries in the row, ie

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \tag{2}$$

   Do this by creating a random matrix of numbers from 0 to 10 and then changing the diagonal entries to ensure the above relationship holds.

   What tolerance did you use? Time these two methods, as well as using LU decomposition with pivoting and Matlab's backslash. Compare the relative differences, in max norm, between your solutions and Matlab's in order to validate your codes. Also report the number of iterations the methods required. Comment on your results.

6

a.) MATLAB

— function to solve $A\underline{x} = \underline{b}$ using the JACOBI ITERATIVE METHOD

└→ INPUTS : A

$\underline{b}$

x_0 ⟶ INITIAL GUESS

tol ⟶ TOLERANCE

Maximum # of iterations ⟶ OPTIONAL

└→ OUTPUT : $\underline{x}$ ⟶ APPROX. SOLUTION

iter ⟶ # of ITERATIONS performed

⟶ use the RELATIVE STOPPING CRITERIA of your choice

└→ What did you use ?

⟶ In class, it is in component form & ∴ used a loop over i

└→ Try to code it W/O such a LOOP to increase efficiency

Since relative tolerance takes into account the size of the error, we used relative tolerance as our stopping criteria of choice to ensure that the relative residual at the kth iteration is small enough or nearly 0. Thus, the iteration will only terminate when the residual is less than the relative tolerance. We chose the tolerance level as 10×10^{-12} as it is relatively close to machine epsilon and is sufficient enough as it accounts for up to 12 digits of accuracy. This will allow us to yield a relatively accurate result, such that the relative difference between the Jacobi solution (or Gauss-Seidel in part b) and MATLAB's solution is relatively close to machine epsilon. We also set the maximum number of iterations allowed to perform to 25000. Since this number is very large, this will guarantee that both the Jacobi and Gauss-Seidel method will converge to a solution.

b.) MATLAB

— function to solve $A\underline{x} = \underline{b}$ using the GAUSS – SEIDEL iterative method

└→ INPUTS : A

$\underline{b}$

x_0 ⟶ INITIAL GUESS

tol ⟶ TOLERANCE

Maximum # of iterations ⟶ OPTIONAL

└→ OUTPUT : $\underline{x}$ ⟶ APPROX. SOLUTION

iter ⟶ # of ITERATIONS performed

⟶ use the SAME stopping criteria as in part (a)

C.) MATLAB

Use your functions to solve $A\underline{x} = \underline{b}$ for A = 1000 × 1000 Matrix

— To ensure that you are using a matrix for which these methods will CONVERGE, make a STRICTLY DIAGONALLY DOMINANT Matrix

└→ STRICTLY DIAGONALLY DOMINANT Matrix : For every row, the magnitude of the diagonal entry is LARGER than the sum of the magnitudes of the other entries in the row

$$|a_{ii}| > \sum_{j \neq i}^{\infty} |a_{ij}|$$

→ Do this by creating a RANDOM Matrix of #'s from [0, 10]
& then, changing the diagonal entries to ensure the above relationship holds

→ What TOLERANCE did you use ?

→ TIME these 2 methods & use LU DECOMPOSITION w/ PIVOTING & MATLAB'S BACKSLASH

→ Compare the RELATIVE DIFFERENCES, in MAX−NORM, b/w your solutions & MATLAB's backslash in order to validate your codes

→ Report the # of iterations the methods required

└→ COMMENT ON YOUR RESULTS

We used the same relative tolerance as in part (a) and part (b), which was a tolerance level of 10×10^{-12} and the maximum number of iterations is 25000. 3 methods were used to compare their solutions to MATLAB's backslash solution, which were Jacobi and Gauss-Seidel iteration methods as well as LU Decomposition with partial pivoting. Comparing the 4 methods, MATLAB has the shortest run time of ~0.09 seconds. Gauss-Seidel had a short run time of ~0.203 seconds. Jacobi had a longer run time of ~2.19 seconds. LU Decomposition had the longest run time of ~3.47 seconds. There were 12,611 iterations performed using the Jacobi method, whereas only 16 iterations were performed using the Gauss-Seidel method. Thus, the Gauss-Seidel method has a faster convergence rate than the Jacobi method. Therefore, the Gauss-Seidel method is more efficient than the Jacobi method as it required (substantially) less number of iterations to converge to the solution. Using MATLAB's backslash solution as the "true" solution, the relative difference of all 3 methods with respect to max-norm was relatively close to machine epsilon. This confirms our algorithms for all 3 methods are working properly.

Iterative methods (such as Jacobi and Gauss-Seidel) are more useful when the matrix is extremely large (i.e., n = 1000). This is because the iterative methods will yield an approximate solution that is acceptable within the convergence stopping criteria, whereas direct methods (such as LU Decomp) will yield the most accurate result. Iterative methods can be faster than direct methods depending on the digits of accuracy we are looking for. This is because we choose the tolerance level for iterative methods as $O(10^{-12})$, so it will converge to a solution faster than direct methods, which determines a solution closest to machine epsilon or $O(10^{-16})$. Thus, Direct Methods are more accurate compared to iterative methods

To make a strictly diagonally dominant matrix, we first created a random matrix of numbers between [0, 1] and made a copy of the original matrix to replace the diagonal entries with 0. Then, take the sum of each row in the matrix and store these values in an array. Finally, Iterate through the diagonal entries of the original matrix and update those values by taking the sum of its row and adding 1. This ensures that the diagonal is greater than the sum of the nondiagonal entries, and all the entries in the matrix are between [0, 10]

7. (30 points) The function `FDjacobi_2D.m` gives the Jacobi method for this finite difference implementation of the Poisson equation, and `FDGaussSeidel_2D.m` gives one version of the Gauss-Seidel method for this finite difference implementation of the Poisson equation. What does one step of the Gauss-Seidel method look like (using the same notation as the end of page 4 of Lecture 17 notes)? What is the difference between it and Jacobi method?

   Use the Gauss-Seidel method to solve the PDE from number (5) for $N = 2^3, 2^4, ...2^6$ with a tolerance of 10^{-8}.

   The function `FDSOR_2D.m` gives a related method called SOR, or Successive Over-Relaxation. Use this to solve the PDE from number (5) for $N = 2^3, 2^4, ...2^9$ with a tolerance of 10^{-8}. Note: The last one is likely to take about

   For GS and SOR, how many iterations were required for each solve? Comment on your results. Also comment on why we would solve it this way instead of the way we did it in number (5).

⑦ MATLAB

— function FDjacobi_2D.m gives the JACOBI method for this finite difference implementation of the POISSON EQN.

— function FDGaussSeidel_2D.m gives one version of the GAUSS – SEIDEL method for this finite difference implementation of the POISSON EQN.

→ What does 1 STEP of the Gauss – Seidel method look like (using the same notation as the end of pg. 4 of Lecture 17 notes)?

→ What is the DIFFERENCE b/w it & Jacobi method?

— use the GAUSS – SEIDEL method to solve the PDE from #5 for $N = 2^3, 2^4, \ldots, 2^6$ w/ a tolerance of 10^{-8}

— function FDSOR_2D.m gives a related method called SOR or Successive Over – Relaxation
  └→ use this to solve the PDE from #5 for $N = 2^3, 2^4, \ldots, 2^9$ w/ a tolerance of 10^{-8}

→ For GS & SOR, how many ITERATIONS were required for each solve?
  └→ comment on your results

→ Comment on why we would solve it this way, instead of the way we did it in #5

Here is one step of the Gauss-Seidel Method:
u(i,j)^(k+1) = (¼) * (u(i-1,j)^(k+1) + u(i+1,j)^(k) + u(i,j-1)^(k+1) + u(i,j+1)^(k) - h^2 * b(i,j))

Here is one step of the Jacobi Method:
u(i,j)^(k+1) = (¼) * (u(i-1,j)^(k) + u(i+1,j)^(k) + u(i,j-1)^(k) + u(i,j+1)^(k) - h^2 * b(i,j))

The difference is that in Jacobi we update the solution at each interior point by averaging the values of the solution at the neighboring grid points from the previous iteration, whereas in Gauss-Seidel we are updating each grid point using the neighboring values that have already been updated in the current iteration.

Forward Difference for 2D Gauss-Seidel

Number_Gridpoints	Number_Iterations
8	133
16	440
32	1513
64	5289

Elapsed time is 2.880116 seconds (for N=2^3 to 2^6).

Forward Difference for 2D Successive Over-Relaxation (SOR)

Number_Gridpoints	Number_Iterations
8	69
16	93
32	145
64	246
128	438
256	804
512	1504

Elapsed time is 133.952836 seconds (for N=2^3 to 2^9).

When running the Gauss-Seidel method for 32 grid points, it takes 0.207849 seconds and a total of 1,513 iterations. If we increase the grid points to 64, it takes a total of 2.880116 seconds and 5,289 iterations. In comparison, the Successive Over-Relaxation (SOR) method takes only 0.028096 seconds and 145 iterations for 32 grid points. If we increase the grid points to 64, it takes a total of only 0.159340 seconds and 246 iterations. Thus, there are significantly fewer iterations performed when using the SOR method for #5. The results show that the Successive Over-Relaxation (SOR) method is significantly faster in terms of convergence.

Direct methods can be more accurate than iterative methods when solving PDEs. The drawback is they are computationally expensive and can require a significant amount of memory. In comparison, iterative methods are great when the problem is large and not possible to compute an exact solution. They are more efficient and require less memory, especially for large problems compared to finite differences.