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? Then, create this in Matlab for $N = 2^2$.

Note: For the Matlab part, even though the right-hand-side is a vector, 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);`
Once you have the above grid, you can define a function `f` and then do `f(xg,yg)`. This will give you the values of f on the entire grid, and this is your right-hand-side, in matrix form.
Then use the `reshape` command to change it to a vector.

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).