Other Final Ideas

Points: 20 points

Problem Background

Tridiagonal matrices, or similar sparse matrices, like pentadiagonal, show up in many places in applied mathematics. One such place is the solution of differential equations. Here is one instance. Assume we are solving the boundary value problem,

$$\begin{aligned} -u''(x) &= f(x) \\ u(0) = a \qquad & u(L) = b. \end{aligned}$$

This can be done by discretizing the x-axis into $N+1$ points,

$$0 = x_0 < x_1 < \ldots < x_{N-1} < x_N = L.$$

We then approximate the second derivative at the points $x_1, x_2, \ldots, x_{N-1}$ to obtain the system of equations,

$$\begin{aligned} \frac{1}{h^2}(-u_0 + 2u_1 - u_2) &= f(x_1) \\ \frac{1}{h^2}(-u_1 + 2u_2 - u_3) &= f(x_2) \\ &\vdots \\ \frac{1}{h^2}(-u_{N-2} + 2u_{N-1} - u_N) &= f(x_{N-1}). \end{aligned}$$

where $h = x_{i+1} - x_i$. The unknown variables $u_i \approx u(x_i)$ are what we are trying to solve for. We know $u_0 = a$ and $u_N = b$, so they can be moved to the right hand side. We then get $N-1$ equations for $N-1$ unknowns, a perfect linear system. Written in matrix form, this gives us a tridiagonal matrix to solve. As N increases, the approximations accuracy improves.

Often we want to test our method by considering a known solution to the differential equation and comparing it to our approximation. Sometimes it is easy to solve the differential equation for a particular $f(x), a$ and b, but sometimes it isn't so easy. A better way is to work backwards. Simply choose your solution first, say $u(x) = \sin(x)$ and the domain, $[0, \pi]$. Now we calculate what $f(x), a$, and b should be. We know $u'' = f(x)$, so this means $f(x) = -\sin(x)$. We can do the same for a and b. This works for any function that you choose for u, and can be used to come up with many different exact solutions to test against.

Program Criteria

Write a program that does the following:

- Write a `def` function that take a tridiagonal matrix `A` and a vector `b` as input. It then solves this system of equations and returns the solution vector. All this should be done with numpy arrays, not lists.
- Write a `def` function that take a pentadiagonal matrix `A` and a vector `b` as input. It then solves this system of equations and returns the solution vector.
- Pick a function $f(x)$, a domain L, and boundary conditions a and b. Pick a value for N, used to discretize your domain. Use all of this to create a right-hand side vector for the linear system described above.
- Create the tridiagonal matrix for the system described above.
- Solve the tridiagonal system with your function and plot the solution. Plot the exact solution on the same graph.

- Next, create your own pentadiagonal system that you know the solution to. Again, do this backward, start with the solution x, multiply it by the pentadiagonal matrix to find the right-hand side.
- Solve the pentadiagonal system using your function. Find the error between the approximate solution and the exact solution. Define the error by the maximum absolute difference between the components.

Deliverables

Place the following in a folder named `FinalProject` in your repository:

- A Python file `F8_Tridiagonal_Name.py` that satisfies the program criteria.
- A PDF file `F8_Tridiagonal_Name.pdf` that describes how your program works. This should be a description of how you went about solving this problem. You should go into some detail about your solution method, but I don't want to see something about every `if` statement and `for` loop. As an example of the type of description I'm looking for, see the file `Goldbach_explanation.doc` in the `Final Problem` folder of my repo.
- A plot of the approximate solution to the differential equation along with the exact solution. Do this for at least two different $f(x)$ and boundary conditions.
- Discuss the difference between the algorithm for solving a tridiagonal system and a pentadiagonal system. What had to change to create the pentadiagonal algorithm?