Math 567 **Homework 4** **Due Oct. 30, 2020**
Grady Wright

1. *SOR method* The FD2-Poisson handout posted on the course webpage contains MATLAB code for numerically solving the Poisson equation

$$\begin{aligned} \nabla^2 u &= f(x,y), \qquad (x,y) \in \Omega = (a,b) \times (a,b) \\ u(x,y) &= g(x,y) \qquad (x,y) \in \partial\Omega\ , \end{aligned}$$

with a second order accurate FD method. The function `fd2poisson` from this handout sets up and solves the resulting linear system using full dense Gaussian elimination (you will fix this in the next problem).

Your goal is to implement a similar function called `fd2poissonsor` in MATLAB or whatever language you choose that instead numerically solves the Poisson equation using successive over relaxation (SOR) with the optimal relaxation parameter of $\omega = 2/(1+\sin(\pi h))$. Your code should produce a solution with a relative residual of $\leq 10^{-8}$. This means that the residual, which is given as

$$r_{j,k} = -4u_{j,k} + u_{j-1,k} + u_{j+1,k} + u_{j,k-1} + u_{j,k+1} - h^2 f_{j,k},$$

must satisfy

$$\sqrt{\sum_{j=1}^{m}\sum_{k=1}^{m} r_{j,k}^2} \leq 10^{-8}\sqrt{\sum_{j=1}^{m}\sum_{k=1}^{m} f_{j,k}^2}.$$

Additionally, your code should not form any matrices involving the 5-point FD stencil. You may want to refer to Sections 4.1 & 4.2 of the book for examples of how the Gauss-Seidel and SOR methods can be implemented in MATLAB.

Illustrate that your code works by solving the Poisson equation from the FD2-Poisson handout with $m = 2^7 - 1$. Produce a plot of the numerically computed solution and of the error in the solution (i.e. difference between computed solution and true solution as done in the code given in the handout for the direct solver). Include the the plots and your `fd2poissonsor` code in your homework write-up.

2. *Comparing Poisson solvers* In this problem you will compare various methods for solving the associated linear system for the second-order accurate finite-difference (FD2) approximation to the Poisson equation:

$$\begin{aligned} \nabla^2 u(x,y) &= 10\pi^2 e^{\sin(2\pi(x+2y))}(-2\sin(2\pi(x+2y)) + \cos(4\pi(x+2y)) + 1) \\ u(x,y) &= e^{\sin(2\pi(x+2y))} \ \text{ for } (x,y) \in \partial\Omega. \end{aligned} \tag{1}$$

This is the problem that is setup in the MATLAB file `testFdPoisson` on the course webpage.

(a) Download the MATLAB function `fd2poisson.m` from the course webpage and modify the code so that it uses MATLAB's *sparse matrix* capabilities (alternatively, you may implement this function in whatever language you choose, but it must use sparse matrix libraries). This means you should create sparse versions of the `D2x` and `D2y` matrices in this function instead of dense versions as the code currently does. Note that to get full credit you should *not* call the functions `sparse` or `gallery('poisson',m,m)` in MATLAB, instead you should figure out how to construct the `D2x` and `D2y` matrices yourself using `spdiags` and `kron` (and possibly `toeplitz`), if using MATLAB. Save your modified function as `fd2poissonsp`. Turn in a listing of your code.

(b) Download the MATLAB functions `fd2poissondst` and `fd2poissonmg` from the course webpage (alternatively, you may implement these function in whatever language you choose). These functions solve the FD2 linear system for the Poisson equation using the (fast) discrete sine transform (DST) and classical multigrid (using a damped Jacobi smoother and a "v-cycle"), respectively. Note that you will also need to download the updated `dst` and `idst` files from the course webpage for this code to work.

(c) Modify the `testFdPoisson` function so that it solves the Poisson equation (1) using:

- the standard dense Gaussian elimination solver (`fd2poisson`),
- the SOR solver from problem 1 (`fd2poissonsor`),
- the sparse Gaussian elimination solver (`fd2poissonsp`),
- the DST based solver (`fd2poissondst`), and
- the multigrid solver (`fd2poissonmg`).

Time how long each method takes to solve the Poisson equation for $m = 2^k - 1$, $k = 4, 5, \ldots, 10$. Produce a table that shows the timing results for each method and for each value of m. Note that for the dense solver you will not be able to go much above $k = 7$ without possibly causing serious issues for your machine. Try to estimate the timings in this case based on the previous values of m. Also, you should run all the timing comparisons a few times to capture the mean behavior of the timing results. Report the specification of the machine you used to do the computations (i.e. make, processor type and speed, and memory).

(d) Which method appears to be the best in terms of actual wall-clock time and in terms of the rate of increase with m?

Note that the wall-clock time for the MG method are higher than you would see if the method was implemented in a compiled language and if a better smoother was used (e.g. Red/Black Gauss-Seidel) [1]. Also, the DST method is not as efficient as it could be since it uses an FFT of twice the needed length to compute the solutions. Furthermore, this code uses the DST in both the x and y directions to solve the problem. More efficient methods exist by only transforming in one direction and then solving several de-coupled tridiagonal systems [2].

3. *Fast Poisson solver with Neumann boundary conditions* Consider Poisson's equation with zero Neumann boundary conditions:

$$\begin{aligned} \nabla^2 u &= f(x,y), \qquad (x,y) \in \Omega = (a,b) \times (a,b) \\ \mathbf{n} \cdot \nabla u(x,y) &= 0, \qquad (x,y) \in \partial\Omega\ , \end{aligned}$$

where $\mathbf{n}$ denotes the (unit) outward normal vector to Ω, and f satisfies the compatibility condition:

$$\int_a^b \int_a^b f(x,y)dx\,dy = 0.$$

(a) Develop a second-order accurate FD method for solving this equation using the fictitious point method for deriving the approximations $\nabla^2 u$ on the boundary (this will be exactly the same as we did in the 1-D problem). Implement a function similar to the `fd2poissondst` that solves the resulting system using the discrete cosine transform (DCT) and name the function `fd2poissondct`. Note that for this function it is unnecessary to pass in a value for g on the boundary. Also, the solution to this Poisson equation is unique up to an additive constant. As in the problem 5 of the last homework assignment, you will find that this property means the linear system from the FD2 approximation is singular with a vector of all ones (or non-zero scaled multiple) being the only eigenvector corresponding to zero eigenvalue. You need to deal with this singularity using the same approach as problem 5 of the previous homework assignment. You should set the arbitrary constant to zero.

Please download the updated `dct` and `idct` functions for solving this problem. Turn in a listing of your code.

(b) Use you code from part (a) to solve the Poisson equation with $f(x,y) = -8\pi^2\cos(2\pi x)\cos(2\pi y)$. An exact solution to the Poisson equation with this f is $u(x,y) = \cos(2\pi x)\cos(2\pi y)$. Plot the difference between your computed solution and the exact solution for $m = 2^6 - 1$. Also, generate a table showing the convergence of your solution to the true solution for $m = 2^k - 1$, $k = 4, 5, \ldots, 10$. The table should show the relative 2-norm of the error. Verify that your method is second-order accurate.

4. *Implicit FD methods*

(a) Using the technique from problem 4 of homework 2, derive the following implicit (compact) fourth-order accurate approximation to the 2-D Poisson equation $u_{xx} + u_{yy} = f$:

$$\frac{1}{6h^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix} u = \frac{1}{12}\begin{bmatrix} & 1 & \\ 1 & 8 & 1 \\ & 1 & \end{bmatrix} f + O(h^4).$$

(b) Write a code for solving the Poisson equation from problem 2, but now using the nine-node, compact, fourth-order accurate FD scheme. You can solve the problem in one of the following two ways. (1) use a sparse matrix library in whatever language you are using to implement the function to construct and solve the linear system as you did in part (a) of problem 2. (2) use SOR to solve the problem similar to what you did in problem 1 (again your code should be matrix-free). Use your code to solve the Poisson equation from problem 2 for various values of m and produce plots and tables that clearly show the fourth order accuracy of the method. Turn in a printed copy of your code and e-mail it to me.

(c) Extra credit (10 points): Do the same thing as 4b, but use the DST to solve the resulting linear system. Produce plots showing your code gives the same results as the direct sparse solver. Illustrate the improved efficiency of the method by comparing the computational time it takes to compute a solution verses the direct sparse solver for various m.

References

[1] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid.* Academic Press, London, 2000.

[2] P. N. Swarztrauber. Fast Poisson solvers. In G. H. Golub, editor, *Studies in Numerical Analysis*, pages 319–370. Mathematical Association of America, Washington, D.C, 1984.