MA 512 **Homework 10** **April 13, 2017**

1. (10 points) The purpose of this problem is to demonstrate the comparative performance of the Jacobi, Gauss–Seidel, and SOR methods on our model Poisson problem. Recall that the problem is

$$\begin{aligned} \Delta u &= f \qquad \text{in } D = [0,1] \times [0,1] \subset \mathbb{R}^2, \\ u &= 0 \qquad \text{on } \partial D. \end{aligned}$$

where $\Delta u \equiv \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ is the Laplacian of u. As we've seen, discretizing this problem using centered differences on a uniform $m \times m$ grid results in the linear system

$$\frac{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}}{h^2} = f_{i,j}, \quad 1 \le i, j \le m, \tag{1.1}$$

where $h = 1/(m+1)$, $u_{ij} \approx u(x_i, y_j)$ and $f_{ij} = f(x_i, y_j)$ for $(x_i, y_j) = (ih, jh)$. We can rewrite this system as $Au = b$, where

$$u = \begin{pmatrix} u_{11} \\ \vdots \\ u_{m1} \\ u_{12} \\ \vdots \\ u_{mm} \end{pmatrix}, \qquad b = -h^2 \begin{pmatrix} f_{11} \\ \vdots \\ f_{m1} \\ f_{12} \\ \vdots \\ f_{mm} \end{pmatrix}, \qquad A = \begin{pmatrix} T_m & -I_m & & \\ -I_m & \ddots & \ddots & \\ & \ddots & \ddots & -I_m \\ & & -I_m & T_m \end{pmatrix}, \tag{1.2}$$

in which I_m is the $m \times m$ identity matrix and

$$T_m = \begin{pmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix} \in \mathbb{R}^{\mathrm{m \times m}}. \tag{1.3}$$

For this problem, download the files `it_method_demo.m`, `laplacian.m`, `jacobi.m`, `gaussseidel.m`, `sor.m`, `loadrhs.m`, and `rhsfun.m`. Read the comments in these files to see what they do and how they work. Note in particular how the sparse Laplacian is created in `laplacian.m`. Also, read the comments on `rhsfun.m` to see the right-hand side function f used in this exercise.

Now, at the command prompt in MATLAB, type in separate lines

```
m=[32 64 128];it_method_demo(m,'Jacobi',@rhsfun);

m=[32 64 128];it_method_demo(m,'Gauss-Seidel',@rhsfun);

m=[32 64 128];it_method_demo(m,'SOR',@rhsfun);
```

In each case, print out and hand in the plots that appear in the figure window. Also in each case, print out and hand in the iteration numbers and run times[1] appearing in the command window. Comment on how the iteration numbers for the different methods compare and how well they reflect the theoretical estimates we derived in class.

2. (10 points) The purpose of this problem is to demonstrate the performance of the GMRES method with various preconditioners on a model *convection-diffusion* problem,[2] given by

$$\begin{aligned} \Delta u + c\,u + d\,\frac{\partial u}{\partial x} + e\,\frac{\partial u}{\partial y} &= f \qquad \text{in } D = [0,1]\times[0,1] \subset \mathbb{R}^2, \\ u &= 0 \qquad \text{on } \partial D. \end{aligned} \tag{2.1}$$

To discretize this problem on an $m \times m$ grid, we can discretize the Laplacian as in (1.1) and approximate the first-order terms using the centered-difference approximations[3]

$$\frac{\partial u}{\partial x}(x_i, y_j) \approx \frac{u_{i+1,j} - u_{i-1,j}}{2h}, \qquad \frac{\partial u}{\partial y}(x_i, y_j) \approx \frac{u_{i,j+1} - u_{i,j-1}}{2h}.$$

With these approximations, (1.1) becomes

$$\begin{aligned} \frac{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}}{h^2} + c\,u_{ij} + d\,\frac{u_{i+1,j} - u_{i-1,j}}{2h} \\ + e\,\frac{u_{i,j+1} - u_{i,j-1}}{2h} = f_{i,j}, \quad 1 \le i,j \le m, \end{aligned} \tag{2.2}$$

with corresponding changes in A in (1.2) and T_m in (1.3). Note that A and T_m are not symmetric if $d \neq 0$ or $e \neq 0$. Thus it is appropriate to use GMRES, which is perhaps the most popular Krylov-subspace method for general nonsymmetric problems.

For this problem, download the files `gmres_demo.m`, `condif.m`, and `adi_pre.m`. You will also need the files `loadrhs.m`, and `rhsfun.m` that you downloaded for the first problem. Read the comments in these files to see what they do and how they work.

[1] These run times are computed using the MATLAB `tic` and `toc` commands. They are not precise measures of actual run times and will vary somewhat over repeated runs.

[2] This is also called an advection-diffusion problem, drift-diffusion problem, or (scalar) transport equation, depending on the context.

[3] These approximations have $\mathcal{O}(h^2)$ error, as does our approximation of the Laplacian. They are suitable if the convection coefficients d and e are not too big relative to the mesh size. However, as d and e grow, instabilities in the form of spurious oscillations will appear in the solution unless the grid is refined. For such "convection-dominated" cases, *upwind-differencing* techniques are usually recommended.

At the command prompt in MATLAB, type in separate lines

```
m=[32 64 128];gmres_demo(m,[10 10 10],@rhsfun,'none');
m=[32 64 128];gmres_demo(m,[10 10 10],@rhsfun,'block Jacobi');
m=[32 64 128];gmres_demo(m,[10 10 10],@rhsfun,'ILU');
m=[32 64 128];gmres_demo(m,[10 10 10],@rhsfun,'ADI');
```

These apply MATLAB's GMRES implementation with a restart value of 40 to solve (2.1) with $c = d = e = 10$ using, respectively, no preconditioning, block-Jacobi preconditioning, incomplete-LU (ILU) preconditioning, and alternating-direction-implicit (ADI) preconditioning.[4] Block-Jacobi preconditioning uses the $m \times m$ blocks T_m on the diagonal of A defined by (2.2). (Ordinary Jacobi preconditioning has no effect, since the diagonal of A is just a scalar multiple of the identity.) ILU preconditioning is applied using the MATLAB `ilu` routine's defaults; there are many options available in `ilu` that might be explored to make it more effective. The ADI preconditioner, roughly speaking, first solves a system involving a "horizontal" operator H that incorporates the derivatives in the x-direction and then solves a system involving a "vertical" operator V that incorporates the derivatives in the y-direction. Since H and V are tridiagonal, these solves are very fast. Moreover, ADI preconditioning is known to provide "scalability" with GMRES.[5] You will see clear evidence of scalability with ADI preconditioning in this exercise.

In each case, print out and hand in the plots that appear in the figure window. Also in each case, print out and hand in the iteration numbers and run times appearing in the command window. Comment on how the iteration numbers for the different preconditioning choices compare among themselves and with those of the iterative methods in the first exercise.

[4]All preconditioning is on the left. Left-preconditioning is the only option explicitly provided in MATLAB's GMRES; however, users can easily implement right preconditioning if desired by providing GMRES with a matrix-vector product routine that includes it.

[5]In large-scale computing, scalability (sometimes called algorithmic scalability) is a *very* desirable property for an iterative method to have. Generally speaking, scalability means that the number of iterations required to produce a given level of accuracy does not significantly increase as the problem size increases. (Of course, the cost per iteration may — and typically does — increase with the problem size.) In the PDE context, in which the problem size increases as the grid is refined, scalability is often called "mesh-independence." Preconditioning is the key to achieving scalability with Krylov-subspace methods on PDE problems. Preconditioners other than ADI that result in scalability are multigrid and overlapping-Schwarz domain decomposition (if properly implemented, of course).