MA 512 **Homework 11** **April 20, 2017**

1. (10 points) The purpose of this problem is to demonstrate the performance of PCG with several different preconditioners on our model 2D Poisson problem. For this problem, you will need to download the M-files `pcg_demo.m`, `adi_pre.m`, `laplacian.m`, `loadrhs.m`, and `rhsfun.m`. Once you have done this, read the comments on `pcg_demo.m` to see what it does and how it works.[1] Similarly, read the comments on the other M-files. Then, in the MATLAB command window, type the following in separate lines:

```
pcg_demo([32 64 128],'none','Poisson',@rhsfun);
pcg_demo([32 64 128],'block Jacobi','Poisson',@rhsfun);
pcg_demo([32 64 128],'SSOR','Poisson',@rhsfun);
pcg_demo([32 64 128],'incomplete Cholesky','Poisson',@rhsfun);
pcg_demo([32 64 128],'ADI','Poisson',@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[2] appearing in the command window.

Note: "SSOR" stands for "Symmetric Successive Over-Relaxation". We have considered SOR with a "forward" co-ordinate sweep, in which the approximate solution components are updated in order from first to last. SSOR consists of a "forward" sweep followed by a "backward" sweep, in which the components are updated from last to first. With our usual operator splitting $A = L + D + U$, the SSOR preconditioning matrix is

$$M = \frac{\omega}{2-\omega}\left(\frac{1}{\omega}D + L\right)D^{-1}\left(\frac{1}{\omega}D + L\right)^T,$$

where ω is the relaxation parameter. It is easy to see that this is SPD for $0 < \omega < 2$. The `pcg_demo` code is currently set up to use the optimal ω for our model Poisson problem, given by $\omega = 2/[1 + \sin(\pi/(m+1))]$. With this ω, SSOR is a very effective preconditioner for PCG on this problem. In general, though, finding a good ω may be difficult.

[1] The routine is set up to solve not only the Poisson problem but also an analogous problem involving the *biharmonic operator* Δ^2. However, for that problem, incomplete-Cholesky preconditioning doesn't work for all grid sizes, and ADI preconditioning isn't enabled. Consequently, I'm not asking you to do anything with that problem on this assignment. By all means, though, feel free to run the code on that problem using block-Jacobi or SSOR preconditioning (but no need to hand anything in).

[2] As in previous assignments, the run times are computed using MATLAB's `tic` and `toc` commands and will vary somewhat over repeated runs.

2. (10 points) In class, we discussed Newton's method for solving $F(u) = 0$ and noted its typical very fast local quadratic convergence. A particular advantage for most discretized PDEs is that the convergence is usually *mesh-independent*, i.e., the number of iterations required to reach a prescribed level of accuracy does not increase significantly as the mesh is refined. However, the convergence is only local, i.e., guaranteed to hold only if the initial guess is sufficiently near a solution. In practice, Newton's method is usually augmented with a *globalization*, i.e., an auxiliary procedure intended to improve the likelihood of convergence to a solution from a poor initial guess. A popular and easily implemented globalization is *backtracking*.[3] In a backtracking globalization, the algorithm tests each step for progress toward a solution. If the step provides sufficient progress, it is accepted and added to the approximate solution; if not, it is shortened in a prescribed way and tested again.

The object of this exercise is to demonstrate the performance of Newton's method with backtracking on an interesting nonlinear PDE. The Newton-backtracking implementation is provided by the code `newtonbt.m` and is based on Algorithm ENB in [3,§6]. In this, the initial "trial" step at each Newton iteration is obtained using MATLAB's "\" operator, which invokes a direct sparse solver (very fast in this case). The measure of progress toward a solution achieved by a step s from a current approximate solution u is based on a comparison of the actual residual-norm reduction $\|F(u)\| - \|F(u+s)\|$ with the residual-norm reduction "predicted" by the *local linear model* of F (the first two terms of the Taylor series), given by $\|F(u)\| - \|F(u) + F'(u)s\|$. If these agree sufficiently well, then the step is accepted, and the next approximate solution is $u^+ = u + s$; if not, then the step is shortened in a specified way and tested again. See [3, §6] for more details. At each iteration, the code prints out the iteration number, the norm of F, the number of backtracks (step-length reductions), and the final factor by which the initial step length was reduced. On termination, a diagnostic message is produced indicating whether the method succeeded or how it failed.

The PDE of interest is the *Bratu problem*, also known as the *Gelfand problem*, given by

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

This and closely related problems have arisen in many applications, notably thermal self-ignition processes of chemically reactive mixtures, in which the solution u represents the difference in temperature between points interior to D and points on ∂D (see, e.g., [4], [5], [6]). It is also very mathematically interesting. There is a $\lambda^* > 0$ such that two solutions are known to exist for $0 < \lambda < \lambda^*$ and no solutions exist for $\lambda > \lambda^*$. The exponential nonlinearity might look scary, but a well-constructed globalized Newton's method usually doesn't have much difficulty in finding a solution.

[3] *Trust-region methods* are widely-used alternative globalizations; see [1] or [2].

For this exercise, download the codes `newton_bratu_demo.m`, `newtonbt.m`, `seesol.m` (used to view the solution surface), `f_Bratu.m`, and `fpr_Bratu.m`. You will also need the files `laplacian.m` and `loadrhs.m`, which you downloaded for the first problem. Read the comments in these files to see what they do and how they work. Run the `newton_bratu_demo.m` routine as it is. This applies `newtonbt.m` to the Bratu problem with $\lambda = 2$ using two initial guesses (the zero function and a "tent" function) and produces two distinct solutions. Print out and hand in the output of `newtonbt.m` that appears in the command window.

For your own edification (no need to hand anything in), look over the output. Note that when the initial guess is the zero function, the algorithm quickly finds the solution. No backtracking is necessary, and quadratic convergence is evident from the beginning. In contrast, when the initial guess is the "tent" function, the algorithm struggles for a number of iterations, taking very short steps that give very little reduction in $\|F\|$, before finally getting close enough to the solution to enjoy quadratic convergence in the last few iterations. (Without backtracking, the iterates diverge for this initial guess.) I encourage you to edit the `newton_bratu_demo.m` code to explore the effects of changing various features. In particular, try uncommenting the appropriate line to use the "peaked" function for the initial guess. (This function looks somewhat more like the solution than the "tent" function.) Then try making `lambda_GL`[4] smaller and smaller. You may have to replace the "45" in the "peaked" function definition by something larger in order for the iterates to converge to the second solution. Note that as `lambda_GL` becomes smaller, the second solution becomes increasingly "peaked" in the center.

References

[1] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *Trust-Region Methods*, MPS-SIAM Ser. Optim., SIAM, Philadelphia, 2000.

[2] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall Ser. Comput. Math., Prentice-Hall, Englewood Cliffs, NJ, 1983. Currently reprinted as SIAM Classics in Appl. Math. #16.

[3] S. C. Eisenstat and H. F. Walker, Globally convergent inexact Newton methods, SIAM J. Optim., 4 (1994), 393-422

[4] D. A. Frank-Kamenetskii, *Diffusion and Heat Exchange in Chemical Kinetics*, Plenum Press, New York, 1969.

[5] J. Jacobsen and K. Schmitt, J. Diff. Equations, 184 (2002), 293-298.

[6] A. Mohsen, A simple solution of the Bratu problem, Computers and Mathematics with Applications, 67 (2014), 26-33.

[4] This is a global variable used to pass the value of λ to `f_Bratu` and `fpr_Bratu`.