

Mathematics 471 Problem Set 3

Due Thursday, March 8, 2001

By submitting a paper for this assignment you declare that you are not submitting any unattributed work of any other person.

Hand in all analytical work that contributes to your solution. Submit printed copies of all programs (except those provided with the textbook), output tables, and graphs. (Use the `diary` feature to get printed output from MATLAB sessions.) Be sure to

- (a) Include explanatory comments in all programs, and make informative labels for all tables and graphs.
- (b) Edit output to make it compact and easy to read.

In this assignment you will explore iteration methods for solving large sparse systems of linear algebraic equations. The system of equations you will practice on is derived from the Laplace equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

in the square $0 < x, y < 1$, subject to the boundary conditions

$$u(x, y) = g(x, y) := x^2 - 3xy - y^2 + x - 2y,$$

when x or y is 0 or 1. The solution is $u^*(x, y) = g(x, y)$. A discrete approximation is derived by replacing the partial derivatives with difference approximations on a mesh with $h = 1/(N + 1)$ and u_{ij} the approximation for $u(x_i, y_j)$:

$$4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = 0 \quad 1 \leq i, j \leq N.$$

The discrete boundary conditions are

$$\begin{aligned} u_{i0} &= g(x_i, 0) & 0 \leq i \leq N + 1 \\ u_{i,N+1} &= g(x_i, 1) & 0 \leq i \leq N + 1 \\ u_{0j} &= g(0, y_j) & 0 \leq j \leq N + 1 \\ u_{N+1,j} &= g(1, y_j) & 0 \leq j \leq N + 1. \end{aligned}$$

Take $N = 64$ in each of the following problems.

1. Write a program to solve the discrete equations by the Jacobi method and the Gauss-Seidel method. (You may as well write your Gauss-Seidel program in C or Fortran. The `for` loops cannot be avoided, so MATLAB code would be just as long, and would run much more slowly. The Jacobi method, however, can be effectively “vectorized.”) Start iterations with the initial guess $u = 0$ and continue until $\|u - u^*\|_2 < 0.001$.

- (a) Use the known spectral radii to predict how many iterations are required to reduce the error to 0.001 by (i) the Jacobi method, (ii) the Gauss-Seidel method. (The spectral radius for Jacobi iteration is $\rho = \cos(\pi h)$; the Gauss-Seidel spectral radius is ρ^2 .) Do your computational results live up to your predictions?
- (b) Every 20 iterations or so, estimate the convergence rate by

$$\text{RateEstimate} := \frac{\|u^{(m+1)} - u^{(m)}\|}{\|u^{(m)} - u^{(m-1)}\|}.$$

Make a plot of your rate estimate against the iteration number. How does the estimated convergence rate compare to the spectral radius?

- (c) *NOTE*: Be sure to conserve array storage; at most your program will require space for the old and new u , and a copy of the true solution u^* for determining the error. It is not acceptable (and probably insufficient) to allocate $64 \times 64 \times 500$ arrays.

2. Modify your program from the previous problem to perform successive overrelaxation. Recall that the algorithm can be formulated as follows. When you visit the (i, j) mesh point, the amount by which the equation there fails to be satisfied is the *residual*

$$r_{ij}^{(m)} = u_{i-1,j}^{(m+1)} + u_{i+1,j}^{(m)} + u_{i,j-1}^{(m+1)} + u_{i,j+1}^{(m)} - 4u_{i,j}^{(m)}.$$

The Gauss-Seidel correction $u_{i,j}^{(m+1)} = u_{i,j}^{(m)} + \frac{1}{4}r_{ij}^{(m)}$ is just what is needed to “relax” the residual at the (i, j) mesh point to zero. Now insert a relaxation parameter ω , to get

$$u_{ij}^{(m+1)} = u_{ij}^{(m)} + \frac{\omega}{4}r_{ij}^{(m)}.$$

Experiment with different values of the overrelaxation parameter $1 < \omega < 2$ to find the one that gives the smallest spectral radius. You can do this by estimating the convergence rate as in the previous problem.

3. The conjugate gradient method is most effective for systems $Ax = b$ that are so large it is impractical to store the matrix A . Implement the conjugate gradient method for a general system $Ax = b$ in a way that does not require storing the matrix A , as follows.

Interface Your code accepts a right-hand-side vector b , a starting guess $x^{(0)}$, a maximum number of iterations `maxit`, and a tolerance `tol` for

stopping the iterations. Your implementation may require the dimension n or other appropriate parameters, such as workspace vectors, to be supplied.

Matrix-vector multiplication Write your conjugate gradient subprogram so that it does not store A or perform the matrix multiplication $v \mapsto Av$ itself, but passes v to a user-supplied subprogram to do the job. In the documentation to your code, specify the calling sequence for this user subprogram. You may require it to have a specific name, or have its name passed as a parameter to your code.

Function Your code solves the linear system $Ax = b$ by the conjugate gradient method, starting with $x^{(0)}$ as the initial guess. Use **double precision** arithmetic. Terminate when either the *Euclidean norm* of the residual is smaller than `tol`, or the number of iterations reaches the *smaller* of `maxit` and n . The code returns the last iterate x , the norm of the residual, and the number of iterations performed.

Efficiency Minimize the number of arithmetic operations and the number of stored vectors that your code requires. Find a tidy way to display results when the linear system is large.

Test I Use your code to solve the linear system $Ax = b$ with

$$A = \begin{bmatrix} 15 & -4 & 2 & 7 \\ -4 & 17 & -5 & 0 \\ 2 & -5 & 5 & 3 \\ 7 & 0 & 3 & 9 \end{bmatrix} \quad b = \begin{bmatrix} 2 \\ 18 \\ -11 \\ -5 \end{bmatrix}$$

For this test, your user-supplied routine for computing Av given v will be straightforward: use a copy of A .

According to the theory, the conjugate gradient method delivers the exact solution in at most 4 steps. How does your code perform?

Test II Use your code to solve the discrete Laplace equation of page 1. The user-supplied routine for computing Av given v does not store A in this case: that would require 128M bytes of memory! Instead it computes Av by simply doing

$$(Av)_{ij} = 4v_{ij} - v_{i-1,j} - v_{i+1,j} - v_{i,j-1} - v_{i,j+1} \quad 1 \leq i, j \leq N.$$