CS 290H: Preconditioning iterative methods // Homework 2

Assigned October 24, 2004

Due by class time Monday, November 7

1. [25 points] This problem and the next concern "modified" incomplete LU factorization, or MILU. Like ordinary ILU, MILU computes lower triangular L and upper triangular U by Gaussian elimination on the input matrix A, dropping some of the entries of L and U to keep them sparse. However, when MILU drops an element of either L or U, it compensates by adding the value of the dropped element to the diagonal of U in the same row. Different dropping rules lead to different versions of MILU. In MILU0, all elements are dropped except diagonal elements and those in positions that are nonzero in A. There's also an MIC, or modified incomplete Cholesky factorization, in which the dropped elements are added to the diagonal before its square root is taken.

For this problem, you are to write two C or Fortran mexfiles, one for MILU0 and one for MIC0. The Matlab calls should be

```
[L, U, flag] = milu(A);
```

and

```
[R, flag] = mic(A);
```

The optional `flag` return is 0 if the factorization completed correctly, or nonzero if a problem was encountered. Possible problems are a zero pivot element in MILU or MIC (which would require dividing by zero in the factorization), a negative pivot element in MIC (which would require the square root of a negative number), and a computed zero on the diagonal of R or U (which would make it impossible to apply the preconditioner).

If you wish, you may allocate the storage for the output matrices by calling Matlab's routines "`tril`" and "`triu`" to create lower and upper triangular matrices with the same pattern as A; then you just need to compute the values of the nonzeros and fill them in.

There's a description of MILU in Section 10.3.5 of Saad's book. However, notice that Saad uses a row-oriented factorization; you will need to use a column-oriented factorization to match Matlab's sparse data structure. In particular, this will mean that you need to keep track of the modification to the diagonal element of U differently than Saad does.

Test your routine on some small examples that you make up by hand. In addition to a program listing, turn in a diary of the outputs of MIC and MILU on the example `grid5(3)`, which is the 9-by-9 matrix of the 2D model problem. ("`grid5`" is in the mesh partitioning toolbox, which you can download from a link on the course references page.) You might also verify that if all the elements of A are nonzero, then the product of L and U (or R^T and R) is equal to A, since nothing is dropped.

2. [25 points] Let L and U be the output of MILU0 on A. Let $B = LU$ be the preconditioner.

(a) Prove that if $a_{ij} \neq 0$ then $a_{ij} = b_{ij}$.

(b) Prove that if e is the n-vector of all ones, then $Ae = Be$.

3. [25 points] The object of this problem is to measure experimentally the convergence rate of CG on the model problem with various preconditioners. You can generate the n-by-n matrix of the 3D model problem by

```
A = grid3d(k);
```

where $n = k^3$. Generate b as a random n-vector. Now use conjugate gradient, via Matlab's `pcg` routine, to solve the system $Ax = b$ to a tolerance of 10^{-8} in each of the following five ways:

- No preconditioning.
- Precondition A with IC0 (via Matlab's `cholinc` routine).
- Precondition A with MIC0 (via the routine you wrote for problem 1 above).
- Permute A by a red-black ordering, so that A takes on a 2-by-2 block form in which both the $(1,1)$ and $(2,2)$ blocks are diagonal. (This will be easier if you choose k to be odd—why?) Then solve $Ax = b$ by CG preconditioned with IC0.
- Permute A by a red-black ordering, then solve $Ax = b$ by CG preconditioned with MIC0.

Do this for as large a range of values of k as you can (all odd, if you wish), and record the number of CG iterations to convergence in each case. Use a log-log plot (generated by Matlab) to estimate the number of iterations as a function of n. Compare your results to some of the entries in the table on slide 15 from the 28 September class.

4. [25 points] A matrix A is *symmetric positive semidefinite* (SPSD for short) if it is symmetric (which implies that its eigenvalues are real) and also its eigenvalues are all nonnegative. A nonsingular SPSD matrix is positive definite, but unlike symmetric positive definite matrices an SPSD matrix can have one or more zero eigenvalues. The purpose of this problem is to prove a few technical lemmas that we'll use in our discussion of support theory.

(a) Prove that a symmetric matrix A is SPSD if and only if $x^T Ax \geq 0$ for all vectors x.

(b) Let A and B be SPSD matrices. Define the *support of B for A* as

$$\sigma(A, B) = \min\{\tau \mid x^T(tB - A)x \geq 0 \text{ for all } x \text{ and for all } t \geq \tau\}.$$

Prove that if B is nonsingular (i.e. positive definite) then

$$\sigma(A, B) = \max\{\lambda \mid Ax = \lambda Bx \text{ for some } x \neq 0\}.$$

(c) Prove that the sum of two SPSD matrices is SPSD.

(d) Suppose $A = A_1 + A_2 + \cdots + A_k$ and $B = B_1 + B_2 + \cdots + B_k$, and suppose A_i and B_i are SPSD for all i. Prove that

$$\sigma(A, B) \leq \max_i \sigma(A_i, B_i).$$