CS412 Spring Semester 2011

Homework Assignment #5

Due Thursday April 21st 2011, in class

Sum of all problems : 120%, Maximum possible score : 100%.

1. [50%] In this problem you will construct a proof that the Jacobi method converges when applied to matrices that are diagonally dominant. In the following subproblems, you are free to use any of the individual questions (even if you didn't prove it) to answer the ones that come after it.

   (a) [10%] If $\mathbf{x} \in \mathbf{R}^n$ and $\mathbf{T}$ is an $n \times n$ matrix, show that for any positive integer k the following inequality holds:

   $$\|\mathbf{T}^k\mathbf{x}\| \leq \|\mathbf{T}\|^k\|\mathbf{x}\|$$

   (b) [5%] Show that the Jacobi method can be written as

   $$\mathbf{x}^{(k+1)} = \mathbf{T}\mathbf{x}^{(k)} + \mathbf{c}$$

   where $\mathbf{T} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ (using the decomposition $\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}$).

   (c) [10%] Define the *error* after the k-th iteration to be $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*$, where $\mathbf{x}^*$ is the *exact* solution to equation $\mathbf{A}\mathbf{x} = \mathbf{b}$. Show that:

   $$\mathbf{e}^{(k+1)} = \mathbf{T}\mathbf{e}^{(k)}$$

   Hint: You can use (after explaining why it is true) that $\mathbf{x}^* = \mathbf{T}\mathbf{x}^* + \mathbf{c}$.

   (d) [15%] If $\mathbf{A}$ is diagonally dominant by rows, and $\mathbf{T}$ is the matrix defined in (b) above, show that $\|\mathbf{T}\|_\infty < 1$.

   (e) [10%] Show that when $\mathbf{A}$ is diagonally dominant by rows , then:

   $$\|\mathbf{e}^{(k)}\|_\infty \leq \|\mathbf{T}\|_\infty^k\|\mathbf{e}^{(0)}\|_\infty$$

   Explain why this result implies that Jacobi is guaranteed to converge with diagonally dominant matrices.

2. [70%] In this problem you will compare the LU decomposition to the Gauss-Seidel method, as two alternative methodologies for solving a certain *sparse* linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. You will also be exposed to `MATLAB`'s notion of a *sparse* matrix, as well as the `tic/toc` commands for measuring the run time of numerical procedures.

   First let us define a benchmark linear system, on which the two different methodologies will be evaluated. For any value of the integer n, we define an $n \times n$ linear system of equations $\mathbf{A}_n\mathbf{x}_n = \mathbf{b}_n$.

   The matrix $\mathbf{A}_n$ has the form:

$$\mathbf{A}_n = \begin{pmatrix} -3 & 1/n & 1/n & 1/n & 1/n & \cdots & 1/n & 1/n & 1/n \\ 1/n & -3 & 1 & & & & & & \\ 1/n & 1 & -3 & 1 & & & & & \\ 1/n & & 1 & -3 & 1 & & & & \\ 1/n & & & 1 & -3 & 1 & & & \\ \vdots & & & & \ddots & \ddots & \ddots & & \\ \vdots & & & & & \ddots & \ddots & \ddots & \\ 1/n & & & & & & 1 & -3 & 1 \\ 1/n & & & & & & & 1 & -3 \end{pmatrix} \in \mathbf{R}^{n \times n}$$

The right hand side $\mathbf{b}_n$ is given by:

$$\mathbf{b}_n = \begin{pmatrix} -2 - 1/n \\ -2 + 1/n \\ -1 + 1/n \\ -1 + 1/n \\ \vdots \\ -1 + 1/n \\ -1 + 1/n \\ -2 + 1/n \end{pmatrix} \in \mathbf{R}^n$$

This specific $\mathbf{b}_n$ is constructed in such a way, that the solution $\mathbf{x}_n$ will be a vector with all entries equal to one.

You can find a `MATLAB` function that builds $\mathbf{A}_n, \mathbf{b}_n$ under the link:

`http://pages.cs.wisc.edu/~cs412-1/hw/test_problem_hw5.m`

This `MATLAB` M-file implements a function `test_problem_hw5`, which returns 5 results, as follows:

```
[A , b, D , L , U] = test_problem_hw5(n)
```

Here, `n` is the desired size of the problem, `A,b` are the matrix $\mathbf{A}_n$ and right hand side $\mathbf{b}_n$, respectively. The matrices `D,L,U` are the ones used in the decomposition of $\mathbf{A}_n$ as `A=D-L-U`, used in the Jacobi or Gauss-Seidel procedure.

(a) [10%] The matrix `A` returned by the provided function is created and stored in `MATLAB`'s internal format for *sparse* matrices, where only the non-zero entries of A are kept in memory. We can convert from this sparse format to a full (dense) matrix using the built-in function `full(A)`.

For `n=100,1000,10000` have `MATLAB` compute the matrix-vector product $\mathbf{A}_n\mathbf{b}_n$ using (a) the sparse matrix `A` returned by the provided function, and (b) the equivalent dense matrix `A_full=full(A)`. Report the elapsed time for each matrix-vector multiplication, measured using `MATLAB`'s `tic/toc` commands. The general mode of use for these commands is:

```
tic; <some commands> ; t=toc
```

At the end of this statement, `t` will contain the time (in seconds) that the commands which intervened between the `tic/toc` construct took to execute.

(b) [25%] For `n=100,1000,2000` use the LU decomposition to solve the system $\mathbf{A}_n\mathbf{x}_n = \mathbf{b}_n$. For every value of `n`, use the `tic/toc` commands to measure how much time was required to compute the LU factorization, and solve the two forward/backward substitution problems to determine $\mathbf{x}_n$. It is recommended that you test the computed solution against the exact solution (the vector of all ones) as a sanity check.

Note: Beware of the fact that the matrices `L` and `U` in the expression `A=D-L-U` are *not* the same as the factors produced by the LU decomposition of $\mathbf{A}$. If we denote the latter by $\hat{\mathbf{L}}$, $\hat{\mathbf{U}}$ for clarity, then these two matrices satisfy $\mathbf{A} = \hat{\mathbf{L}}\hat{\mathbf{U}}$ and need to be computed using `MATLAB`'s `lu` function as discussed in class.

(c) [35%] For `n=100,1000,2000,10000` use 10 iterations of the Gauss-Seidel method to generate an approximation for the solution $\mathbf{x}_n$. At each step, you will need to solve the lower-triangular system

$$(\mathbf{D} - \mathbf{L})\mathbf{x}^{(k+1)} = \mathbf{U}\mathbf{x}^{(k)} + b$$

to generate the next iterate in your approximation sequence (you may use the `MATLAB` backslash operator to solve this triangular system). Use the `tic/toc` commands to measure how long `MATLAB` took to complete these 10 Gauss-Seidel iterations. You can use the vector of all zeroes as your initial guess.

Also, report the infinity-norm of the *error* after completing the 10 Gauss-Seidel iterations, for each problem size. The error vector can be computed by subtracting the known exact solution (vector of all ones) from the computed approximation.