CS 4220: Assignment 3

Due: Monday, March 1, 2010 (In Class or in Upson 5153 by 4pm)

Scoring for each problem is on a 0-to-5 scale (5 = complete success, 4 = overlooked a small detail, 3 = good start, 2 = right idea, 1 = germ of the right idea, 0 = missed the point of the problem.) One point will be deducted for insufficiently commented code. Unless otherwise stated, you are expected to utilize fully MATLAB's vectorizing capability subject to the constraint of being flop-efficient. Test drivers and related material are posted on the course website `http://www.cs.cornell.edu/courses/cs4220/2010sp/`. For each problem submit output and a listing of all scripts/functions that *you* had to write in order to produce the output. You are allowed to discuss *background* issues with other students, but the codes you submit must be your own.

P1. (Symmetric Positive Definite Block Tridiagonal Systems)

A block matrix $M = (M_{ij})$ is block tridiagonal if $M_{ij} = 0$ whenever $|i - j| > 1$, e.g.,

$$M = \begin{bmatrix} M_{11} & M_{12} & 0 & 0 \\ M_{21} & M_{22} & M_{23} & 0 \\ 0 & M_{32} & M_{33} & M_{34} \\ 0 & 0 & M_{43} & M_{44} \end{bmatrix}.$$

Matrices with this structure arise in many applications. This problem is about solving block triadiagonal linear systems when the matrix is also symmetric and positive definite. A symmetric block tridiagonal matrix looks like this:

$$\tilde{A} = \begin{bmatrix} A_1 & C_1^T & 0 \\ C_1 & A_2 & C_2^T \\ 0 & C_2 & A_3 \end{bmatrix} \qquad A_i, C_i \in \mathbb{R}^{m \times m}$$

Note that the diagonal blocks are symmetric. Positive definiteness implies that there is a Cholesky factorization $\tilde{A} = \tilde{G}\tilde{G}^T$. It turns out that the Cholesky factor of such a matrix is block bidiagonal. To see this and to derive computable recipes, we equate blocks in the equation

$$\begin{bmatrix} A_1 & C_1^T & 0 \\ C_1 & A_2 & C_2^T \\ 0 & C_2 & A_3 \end{bmatrix} = \begin{bmatrix} G_1 & 0 & 0 \\ H_1 & G_2 & 0 \\ 0 & H_2 & G_3 \end{bmatrix} \begin{bmatrix} G_1^T & H_1^T & 0 \\ 0 & G_2^T & H_2^T \\ 0 & 0 & G_3^T \end{bmatrix} = \tilde{G}\tilde{G}^T.$$

and reach the following conclusions:

$(1,1)$:	$A_1 = G_1 G_1^T$	Get G_1 via Cholesky of A_1.
$(2,1)$:	$C_1 = H_1 G_1^T$	Solve for H_1 since G_1 is known.
$(2,2)$:	$A_2 = H_1 H_1^T + G_2 G_2^T$	Get G_2 via Cholesky of $A_2 - H_1 H_1^T$.
$(3,2)$:	$C_2 = H_2 G_2^T$	Solve for H_2 since G_2 is known.
$(3,3)$:	$A_3 = H_2 H_2^T + G_3 G_3^T$	Get G_3 via Cholesky of $A_3 - H_2 H_2^T$.

It can be shown that the matrices $A_{i+1} - H_i H_i^T$ are positive definite thereby guaranteeing the existence of the "little" Cholesky factorizations. The MATLAB function `chol` can be used for this. The H_i can obtained by triangular system solving. (You'll have to work out the details; it involves the backslash operator \ and the matrix transpose identity $(PQ)^T = Q^T P^T$.)

Once the block Cholesky factorization of A is obtained it is an easy matter to solve $\tilde{A}\tilde{x} = \tilde{b}$ via the sequence $\tilde{G}\tilde{y} = \tilde{b}$, $\tilde{G}^T\tilde{x} = \tilde{y}$. For example, to solve

$$\begin{bmatrix} G_1 & 0 & 0 \\ H_1 & G_2 & 0 \\ 0 & H_2 & G_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

we equate block components:

$$\begin{aligned} (1): &\quad G_1 y_1 = b_1 && \text{Solve for } y_1 \text{ via } \backslash. \\ (2): &\quad H_1 y_1 + G_2 y_2 = b_2 && \text{Solve for } y_2 \text{ via } \backslash. \\ (3): &\quad H_2 y_2 + G_3 y_3 = b_3 && \text{Solve for } y_3 \text{ via } \backslash. \end{aligned}$$

This is just the block version of forward elimination.

Cell arrays are a handy way to represent block vectors and matrices in MATLAB. For example,

```
m = 10; A = cell(3,1); C = cell(2,1); b = cell(3,1);
for i=1:3
   M = randn(m,m); A{i} = M + M'; b{i} = randn(m,1);
   if i<3
      C{i} = randn(m,m);
   end
end
```

encodes a 3-by-3 symmetric block tridiagonal system

$$\begin{bmatrix} A_1 & C_1^T & 0 \\ C_1 & A_2 & C_2^T \\ 0 & C_2 & A_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

with 10-by-10 blocks. Implement the functions

```
  function [G,H] = BlockTriDiag(A,C)
% A is a length N cell array with symmetric m-by-m blocks
% C is a length N-1 cell array with m-by-m blocks
% Together, A and C represent an N-by-N symmetric block tridiagonal matrix
% Atilde that is positive definite.
% G and H are cell arrays that encode Atilde's Cholesky factor.

  function x = BlockSol(G,H,b)
% G and H are cell arrays produced by BlockTriDiag(A,C)
% b is a N-by-1 cell array that encodes a block vector with m-by-1 entries.
% x is a N-by-1 cell array that encodes a block vector with m-by-1 entries
%    that solves the system Atilde*x = b.
```

You may use `chol` to compute the small Cholesky factorizations. Note that if `R = chol(A)` then `R` is upper triangular and $A = R^T R$. Thus, `G = chol(A)'` assigns to `G` a lower triangular matrix such that $A = GG^T$. When computing symmetric matrices of the form $S = SymMatrix - YY^T$, you do *not* have to be flop efficient. Submit listings of these functions together with the output generated by the script `P1`.

P2. (A Constrained Underdetermined System)

The SVD also exists for matrices that have more columns than rows, e.g.,

$$\begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}^T \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \end{bmatrix}$$

Note that if

$$U^T A V \;=\; \Sigma \qquad A \in \mathbb{R}^{m\times n},\; m < n$$

is the SVD and A has full row rank, then the columns of $V(:, m+1{:}n)$ span the null space of A. To appreciate this fact, compare columns $m+1$ through n in the matrix equation $AV = U\Sigma$.

The SVD $U^T A V = \Sigma$ can be used to solve an underdetermined system $Ax = b$. Assume that $A \in \mathbb{R}^{m\times n}$ and that $\text{rank}(A) = m$. Since

$$Ax = b \qquad \Leftrightarrow \qquad \underbrace{(U^T A V)}_{\Sigma}\underbrace{(V^T x)}_{y} \;=\; \underbrace{U^T b}_{\tilde{b}}$$

we conclude that $x = Vy$ solves the system provided $y_i = \tilde{b}/\sigma_i$ for $i = 1{:}m$. Note that $y(m+1{:}n)$ can take on *any* value. To see why, take a look at this example of a $\Sigma y = \tilde{b}$ system:

$$\begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

A nice way to sum up the nonuniqueness is to observe that

$$x = V(:, 1{:}m)y(1{:}m) + V(:, m+1{:}n)z \tag{1}$$

solves the underdetermined system where $z \in \mathbb{R}^{n-m}$ is arbitrary. An obvious choice is to set $z = 0$. But there are other strategies. For example, we could choose z so that the solution (1) is as close as possible to a given vector x_*. Implement a method for solving this problem:

```
function x = UnderD(A,b,xStar)
% A is an m-by-n matrix with rank(A) = m < n.
% b is m-by-1 and xStar is n-by-1.
% x minimizes norm(x - xStar,2) subject to the constraint that Ax = b
```

Make effective use of the MATLAB SVD function. You are not allowed to use the backslash operator. Submit a listing of `UnderD` and the output produced when the test script `P2` is run.

P3. (Least Squares Curve Fitting)

Suppose we are given points $(z_1, y_1), \ldots, (z_m, y_m)$ with distinct z_i. We say that

$$p^{(d)}(x) = c_1 + c_2 x + \cdots + c_{d+1} x^d$$

is the best degree-d polynomial fit of the data if it minimizes

$$\text{Error}(p) \;=\; \sqrt{\frac{1}{m}\sum_{i=1}^{m} |p(z_i) - y_i|^2}$$

over all degree-d polynomials. Define the vectors z and y by

```
m = 200;
z = linspace(0,10,m);
y = exp(-z/2).*sin(3*z);
```

Write a function `P3(d)` that plots in a single window the function $f(x) = e^{-x/2}\sin(3x)$ and the best fit polynomial $p^{(d)}$ across the interval [0,10]. The function `P3` should also display in a separate figure window, a graph that shows how $\text{Error}(p^{(j)})$ decreases for $j = 1{:}d$. Submit a listing of `P3` and the output when it is run with $d = 15$. Your implementation of `P3` should solve all least squares problems using the MATLAB function `qr`. For full credit, your implementation should compute just one QR factorization.