CS 621: Assignment 3

Due: Wednesday, October 16, 2002 (In Lecture or 4130 Upson by 4pm)

Scoring on each problem is on a 0-1-2-3 scale. 3 = complete success, 2 = overlooked a small point, 1 = germ of the right idea, 0 = missed the point of the problem. One point will be deducted for insufficiently commented code. Test drivers and related material will be posted on the course website. For each problem submit output and a listing of all the scripts/functions that you had to write/modify 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.

Problem 1. Complete the following MATLAB function so that it performs as specified:

```
  function v = MaybeChol(A)
% A is an n-by-n symmetric matrix.
% If A is positive definite then v is its Cholesky factor.
% Otherwise, v is a unit 2-norm vector such that v'*A*v <= 0
```

Your implementation should be based on Algorithm 4.2.1 in GVL. In particular, modify that algorithm so that if the square root in step j is to be of a negative number, then you jump out of the loop and get the required vector. That computation will take advantage of what you did during steps 1 through $j-1$. Test your implementation with the script `P1` that is available on the website.

Hints. At the start of the j-th step we have

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} G_{11} & 0 \\ G_{21} & I_{n-j+1} \end{bmatrix} \begin{bmatrix} I_{j-1} & 0 \\ 0 & C \end{bmatrix} \begin{bmatrix} G_{11}^T & G_{21}^T \\ 0 & I_{n-j+1} \end{bmatrix}$$

If $c_{11} \leq 0$ we've discovered that A is not positive definite.

Problem 2. Complete the following MATLAB function so that it performs as follows:

```
  function x = SpecialPD(U,b)
% b is a column n-vector and U is n-by-p with rank p.
% x solves (I + U*U')x = b
```

Make effective use of the Sherman-Morrison-Woodbury formula, the MATLAB `chol` function, and the \ operator. Test your implementation with the script `P2` that is available on the website.

Problem 3. For $z \in \mathbb{R}^n$ define the symmetric Toeplitz matrix $T(z) \in \mathbb{R}^{n \times n}$ by $t_{ij} = z_{|i-j|+1}$. Complete the following MATLAB function so that it performs as follows:

```
  function s = NearToep(r,x,b)
% r, x, and b are column n-vectors.
% s is a column n-vector that minimizes norm(T(r+s)x - b,2). If
% there is more than one minimizer then s itself has minimal 2-norm.
```

Test your implementation with the script `P3` that is available on the website.

Hint. Derive an n-by-n linear system that defines the vector s, e.g., $Ms = v$. Trouble is, M can be rank deficient. So solve this system in the LS sense using SVD. Set computed singular value $\hat{\sigma}_k$ to zero if $\hat{\sigma}_k \leq 10\text{EPS}\hat{\sigma}_1$. For matrix-vector products with Toeplitz matrices you are allowed to use `Toeplitz`, e.g., `f = toeplitz(h)*d`.

Problem 4. Complete the following MATLAB function so that it performs as follows:

```
  function [L,D,P] = DiagPiv(A)
% A is an n-by-n symmetric matrix.
% P is a permutation matrix, D is block diagonal with 1-by-1 or 2-by-2 blocks, and L is
% unit lower triangular so PAP' = LDL'
```

Your function should implement the Bunch-Kaufman pivot strategy on page 169. Your implementation should be nonrecursive. Test your implementation with the script `P4` that is available on the website.

Hint. At the start of the j-th step we have

$$P \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} P^T = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} D_1 & 0 \\ 0 & \tilde{A} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & I \end{bmatrix}$$

where P is permutation matrix (but not the final P), L_{11} is unit lower triangular, D_1 is symmetric block diagonal with $j-1$ diagonal blocks that are 1-by-1 or 2-by-2. L_{21} is available and so is the symmetric submatrix $\tilde{A}$.

Now turn your attention to $\tilde{A}$ which we factor as follows:

$$P_1 \tilde{A} P_1^T = \begin{bmatrix} E & C^T \\ C & B \end{bmatrix} P^T = \begin{bmatrix} I & 0 \\ CE^{-1} & I \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & B - CE^{-1}C^T \end{bmatrix} \begin{bmatrix} I & E^{-1}C^T \\ 0 & I \end{bmatrix}$$

The permutation P_1 is determined according to the Bunch-Kaufman strategy. E is either 1-by-1 or 2-by-2. About the updates...

(a) P is updated via

$$P \leftarrow \begin{bmatrix} 1 & 0 \\ 0 & P_1 \end{bmatrix} P$$

(b)L becomes

$$\begin{bmatrix} L_{11} & 0 \\ P_1 L_{21} & L_{22} \end{bmatrix} \qquad L_{22} = \begin{bmatrix} I & 0 \\ CE^{-1} & I \end{bmatrix}$$

(c)

$$\begin{bmatrix} D_1 & 0 \\ 0 & \tilde{A} \end{bmatrix}$$

becomes

$$\begin{bmatrix} D_1 & 0 & 0 \\ 0 & E & 0 \\ 0 & 0 & B - CE^{-1}C^T \end{bmatrix}$$

In your implementation, you do not have to be efficient–just correct. Thus, it is OK to have steps like

```
Y = E\C';
% or
Atilde = B - C*Y'
% or
L22 = P1*L22
% or
P(r:n,:) = P1*P(r:n,:)
```