CS 621: Assignment 3

Due: Friday, October 15, 2004 (In Lecture or 4130 Upson by 4pm)

Scoring for each problem is on a 0-to-3 scale (3 = complete success, 2 = overlooked a small detail, 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/cs621/2004fa/`. 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. ($A = LDL^H$ for Hermitian Tridiagonal A)

If $A \in \mathbb{C}^{m\times n}$ and $B = A^H \in \mathbb{R}^{n\times m}$, then b_{ji} is the complex conjugate of a_{ij}. If $A \in \mathbb{C}^{n\times n}$ and $A = A^H$, then A is *hermitian*. If $A \in \mathbb{C}^{n\times n}$ satisfies $x^H Ax > 0$ for all nonzero $x \in \mathbb{C}^n$, then A is *positive definite*. If $A \in \mathbb{C}^{n\times n}$ has the property that $a_{ij} = 0$ whenever $|i-j| > 1$, then A is tridiagonal. In MATLAB, if `A` is a complex matrix and `B = A'`, then $B = A^H$. Complete the following MATLAB function so that it performs as specified.

```
  function [d,c,e] = HermFactor(a,g,h)
% a is a real column n-vector.
% g and h are real column (n-1)-vectors.
% Let A be the n-by-n Hermitian tridiagonal matrix with diagonal a and subdiagonal
% c + i*e,  where i^2 = -1. Assume that A is positive definite so that it has
% the factorization A = L*D*L' where D is diagonal and L is unit lower bidiagonal.
%
% d is a real column n-vector so D = diag(d).
% c and e are real column (n-1)-vectors so that the subdiagonal
% of L is  c + i*e.
```

Test your implementation with the script `P1` that is available on the course website.

P2. (Symmetric Indefinite Systems)

Available on the course website is the function

```
  function [L,D] = LDLT(A)
% A is an n-by-n symmetric matrix
% L is unit lower triangular, and D is diagonal
% so A = LDL'

  [n,n] = size(A);
  D = zeros(n,n);
  L = eye(n,n);
  for k=1:n
     D(k,k) = A(k,k);
     v = A(k+1:n,k);
     L(k+1:n,k) = v/D(k,k);
     A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - L(k+1:n,k)*v';
  end
```

This implementation involves $2n^3/3$ flops, so it is flop inefficient by a factor of two. If A is positive definite, then the algorithm is stable, runs to completion, and all the `D(k,k)` are positive. If A is indefinite, then the algorithm may still run to completion, but it may be unstable. Our intention is to terminate the iteration as soon as a non-positive `D(k,k)` is encountered and to then compute a unit 2-norm $v \in \mathbb{R}^n$ so that $v^T Av \leq 0$. Sylvester's Law of Inertia says this is possible. (If $D(k,k) \leq 0$ then A has a nonpositive eigenvalue and therefore is *not* positive definite.) Complete the following MATLAB function so that it performs as specified.

```
  function v = NegVec(A)
% A is an n-by-n symmetric matrix with at least one nonpositive eigenvalue.
% v is a unit 2-norm vector with the property that v'*A*v <= 0.
```

Test your implementation with the script `P2` that is available on the course website. Your implementation should be flop-efficient. You may use \ to solve triangular systems.

P3. (Diagonal Pivoting with Skew-Symmetric Matrices)

A matrix $A \in \mathbb{R}^{n\times n}$ is *skew-symmetric* if $A^T = -A$. The eigenvalues of such a matrix lie on the imaginary axis and come in complex conjugate pairs. Thus, if $A \in \mathbb{R}^{n\times n}$ is skew-symmetric and n is odd, then A is singular. This problem and the next are about solving even-order nonsingular skew-symmetric systems using the factorization $PAP^T = LDL^T$ where L is unit lower triangular, D is block diagonal with 2-by-2 blocks, and P is a permutation chosen to make the the computation stable. The idea is to modify the Bunch-Parlett method for symmetric systems (GVL, p.168). The situation is actually simpler in the skew-symmetrix case because such a matrix has zeros on its diagonal, so all the pivots will be 2-by-2.

Suppose $P_1 \in \mathbb{R}^{n\times n}$ is a permutation and that

$$P_1 A P_1^T = \begin{bmatrix} E & C^T \\ -C & B \end{bmatrix}$$

where $E \in \mathbb{R}^{2\times 2}$. It follows that E is skew-symmetric and so must have the form

$$E = \begin{bmatrix} 0 & \mu \\ -\mu & 0 \end{bmatrix}.$$

If A is nonsingular we can choose P_1 as in the Bunch-Parlett algorithm enabling us to produce the stable factorization

$$P_1 A P_1^T = \begin{bmatrix} I_2 & 0 \\ -CE^{-1} & I_{n-2} \end{bmatrix} \begin{bmatrix} E & 0 \\ 0 & B + CE^{-1}C^T \end{bmatrix} \begin{bmatrix} I_2 & E^{-1}C^T \\ 0 & I_{n-2} \end{bmatrix}.$$

Repetition of this idea leads to what we'll call the *skew Bunch-Parlett* factorization

$$PAP^T = LDL^T \qquad D = \mathrm{diag}(D_1, \ldots, D_m)$$

where P is a permutation, L is unit lower triangular, $m = n/2$, and

$$D_k = \begin{bmatrix} 0 & \mu_k \\ -\mu_k & 0 \end{bmatrix} \qquad k = 1{:}m.$$

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

```
  function [L,mu,pVec] = SkewBunchParlett(A)
% A is an n-by-n symmetric matrix with n = 2m.
% mu is a column m-vector, L is unit lower triangular, and pVec is a permutation of 1:n
% so that if P = I(:,pVec) then PAP' = LDL' is the Skew Bunch-Parlett factorization
% where D = diag(D1,...,Dm), with Dk = [0 mu(k); -mu(k) 0] for k=1:m.
```

Test your implementation with the script `P3` that is available on the course website. Your implementation must be vectorized and involve just $n^3/3$ flops.

P4. (Using the Skew Bunch-Partlett Factorization)

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

```
  function x = SolveSkew(L,mu,pVec,b)
% b is a column n-vector with n = 2m. Solves Ax = b where A is a nonsingular n-by-n
% skew-symmetric matrix and L, mu, and pVec are output from SkewBunchParlett(A).
```

Test your implementation with the script `P4` that is available on the course website. You may use \ to solve triangular systems.