CS 621: Assignment 5

Due: Monday, November 22, 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. (Complex Householder)

Complete the following function so that it performs as specified:

```
  function v = HouseC(x)
% x is a nonzero complex column n-vector
% v is a complex unit n-vector so that if P = I - 2*v*v' then
% Px is a (complex) multiple of I(:,1).
```

Make sure that your implementation determines a "stable" v. (As in the real case v is not unique. In that situation we chose v to avoid possible catastrophic cancellation. You're expected to do the same sort of thing here.) Test your implementation onthe script `P1.m`.

P2. (Tridiagonalizing a Complex Hermitian Matrix)

If $A \in \mathbb{R}^{n\times n}$ satisfies $A^H = A$, then A is *Hermitian*. This means that $a_{ji} = \bar{a}_{ij}$. Making effective use of `HouseC`, complete the following function so that it performs as specified:

```
  function T = TriDiag(A)
% A is an n-by-n Hermitian matrix.
% T is a real tridiagonal symmetric matrix with the property that
% Q'*A*Q = T for some n-by-n unitary Q.
```

Make sure your implementation is flop efficient and vectorized. (FYI: A complex add involves two real flops and a complex multiply involves six real flops.) Test your implementation onthe script `P2.m`.

P3. (Eigenvalues of a Bordered Diagonal Matrix)

Take a look at P8.5.6 in GVL. Suppose

$$A \;=\; A(d,v,\alpha) \;\equiv\; \begin{bmatrix} \mathrm{diag}(d_1,\ldots,d_{n-1}) & v \\ v^T & \alpha \end{bmatrix} \qquad v \in \mathbb{R}^{n-1}$$

Assume that the d_i are distinct and that v has no zero components. It can be shown that if $\lambda \in \lambda(A)$ then λ is a zero of

$$f(\lambda) \;=\; \lambda \;+\; \sum_{k=1}^{n-1} \frac{v_k^2}{d_k \;-\; \lambda} \;-\alpha$$

Complete the following function so that it performs as specified

```
  function lambdaVec = BorderedDiag(d,v,alpha)
% d  is a column n-1 vector with d(1) > d(2) > ... > d(n-1)
% v is a column n-1 vector withnonzero entries
% alpha is a scalar
% lambdaVec is a column n-vector whose entries are eigenvalues of A(d,v,alpha)
```

Make effective use of the MATLAB root-finder `fzero`. Note, if `MyFun(lambda,d,v,alpha)` implements that function f above, then a call of the form

```
d =  initialized (n-1)-vector
v =  initialized (n-1)-vector
alpha = scalar
L = left endpoint of a bracketing interval
R = right endpoint of a bracketing interval

lambda_star = fzero(@(lambda) MyFun(lambda,d,v,alpha),[L,R])
```

will assign a root of f to `lambda_star` assuming that f changes sign on $[L, R]$ Test your implementation on the script `P3.m`.

P4. (Eigenvectors of a Bordered Diagonal Matrix)

Take a look at P8.5.6 in GVL again. Complete the following function so that it performs as specified

```
  function Q = BorderedDiag(d,v,alpha,lambdaVec)
% d  is a column n-1 vector with d(1) > d(2) > ... > d(n-1)
% v is a column n-1 vector withnonzero entries
% alpha is a scalar
% lambdaVec is a column n-vector whose entries are the eigenvalues of A(d,v,alpha)
% Q is orthogonal so that Q'*A(d,v,alpha)*Q = diag(lambdaVec)
```

Test your implementation on the script `P4.m`. Your Q matrix should have unit columns. But don't be surprised if it is not orthogonal to working precision! Hint. Consider the blocked eigenvector equation

$$\begin{bmatrix} \text{diag}(d_1,\ldots,d_{n-1}) & v \\ v^T & \alpha \end{bmatrix} \begin{bmatrix} z \\ \mu \end{bmatrix} = \lambda \begin{bmatrix} z \\ \mu \end{bmatrix}$$