CS 4220: Assignment 5

Due: Sunday, 3pm, April 11, 2010 (By Single PDF or Equivalent to cv@cs.cornell.edu)

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. (Cube Roots)

Implement a function `y = MyCubeRoot(x)` that returns in `y` the cube root of a real scalar `x`. Your implementation should have the following attributes:

- The relative error in `y` should be in the vicinity of the unit roundoff `eps`.
- It should capitalize on "domain reduction" as we now describe. Without loss of generality, assume that x is positive and note that we can write

  $$x = m \cdot 8^e$$

  where $1 \leq m < 8$. Thus

  $$\sqrt[3]{x} \;=\; 2^e \cdot \sqrt[3]{m}$$

  showing that the general cube root problem reduces to the problem of computing cube roots on the restricted domain $[1, 8)$. Note: You are allowed to use MATLAB's log functions to get m and e.
- It should apply Newton's method for cube roots on the restricted domain:

  $$z_{new} \;=\; z_{old} \;-\; \frac{z_{old}^3 - m}{3 z_{old}^2}$$

- The initial value for the Newton iteration should be intelligently chosen. Hint: Approximate the function $\sqrt[3]{x}$ on the restricted domain with a linear function $L(x)$ and use the value $L(m)$ as a starting value.

Through experimentation, determine how many Newton iterations are required to obtain the required accuracy and "hardwire" that value into your implementation. That way I can assess the efficiency of your technique. Submit a listing of `MyCubeRoot` and the output when it is run with the test script `P1`.

P2. (Smallest Eigenvalue in Absolute Value)

Assume that the n-by-n symmetric tridiagonal matrix

$$T \;=\; T(a,c) \;=\; \begin{bmatrix} a_1 & c_1 & 0 & 0 & 0 \\ c_1 & a_2 & c_2 & 0 & 0 \\ 0 & c_2 & a_3 & c_3 & 0 \\ 0 & 0 & c_3 & a_4 & c_4 \\ 0 & 0 & 0 & c_4 & a_5 \end{bmatrix} \qquad (n = 5)$$

has no zero subdiagonal elements. This means that it has n distinct eigenvalues and they are the zeros of the characteristic polynomial $p_n(x) = \det(T - xI)$. This polynomial can be evaluated using the recursion

$$p_k(x) \;=\; (a_k - x)p_{k-1}(x) \;-\; c_{k-1}^2\, p_{k-2}(x) \qquad k \geq 2$$

where $p_0(x) = 1$ and $p_1(x) = (a_1 - x)$.

Another interesting property has to do with the number of sign changes in the sequence

$$\{p_0(\mu),\ p_1(\mu), \ldots,\ p_n(\mu)\ \}$$

If $s(\mu)$ is that number, then T has precisely $s(\mu)$ eigenvalues that are $\leq \mu$. Thus, if

$$\{\ p_0(10),\ p_1(10),\ p_2(10),\ p_3(10),\ p_4(10),\ ,p_5(10)\ \} \ = \ \{\ 1,\ 4,\ -2,\ 3,\ 6,\ 1\ \} \ = \ \{\ +,\ +,\ -,\ +,\ +,\ +\ \},$$

then $s(10) = 2$ and T has two eigenvalues ≤ 10. Note, if one of the $p_k(\mu)$ is zero, assign to it the opposite sign of $p_{k-1}(\mu)$. Thus, if

$$\{\ p_0(10),\ p_1(10),\ p_2(10),\ p_3(10),\ p_4(10),\ ,p_5(10)\ \} \ = \ \{\ 1,\ 4,\ -2,\ 3,\ 0,\ 1\ \} \ = \ \{\ +,\ +,\ -,\ +,\ -,\ +\ \},$$

then $s(10) = 4$ and T has four eigenvalues ≤ 10.

We can use the sign-change property to compute, via bisection, the k-th smallest eigenvalue of T. Here is the basic idea:

$L = -\| T \|_1$ (Can show $s(L) = 0$.)
$R = \| T \|_1$ (Can show $s(R) = n$.)
while $(R - L > tol)$
    $m = (L + R)/2$
    if $s(m) > k$
        $R = m$
    else
        $L = m$
    end
    % $[L, R]$ contains the k-th smallest eigenvalue of T.
end

Think about how you would use this idea to compute T's smallest eigenvalue in absolute value assuming that T has at least one strictly positive eigenvalue and one strictly negative eigenvalue. Hint: $s(0)$ is important. Implement the following function so that it performs as specified:

```
function x = LambdaSmall(a,c)
% a is a column n-vector and c is a column (n-1)-vector with nonzero entries.
% The tridiagonal matrix T(a,c) has at least one positive and one negative eigenvalue.
% x is the eigenvalue of T(a,c) that is closest to zero.
```

Bisection can obviously be used to solve this problem. However, your implementation must use bisection only to obtain a bracketing interval that houses x. It should then use `fzero` (applied to the polynomial p_n) to compute x with absolute error 10^{-12}. Submit listing and output when the script `P2` is applied.

P3. (Point-To-Ellipse Distance)

Check out my ellipse slides at `http://www.cs.cornell.edu/cv/OtherPdf/Ellipse.pdf`. Among other things you will find lots of facts about how an ellipse can be represented. Complete the following function so that it performs as specified:

```
function [xStar,yStar,dist] = E2Pdist(A,B,C,D,E,F,x0,y0)
% Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0 defines an ellipse in the plane.
% (xStar,yStar) is the closest point on the ellipse to (x0,y0) and dist is the
% euclidean distance between them.
```

Make effective use of `fminbnd` and think hard about the initial bracketing interval. Hint: Through rotation and translation, transform the given problem to a "canonical problem" where the ellipse is specified by $((x - h)/a)^2 + ((y - k)/b)^2 = 1$ and the given point is $(\tilde{x}_0, \tilde{y}_0)$. Solve the canonical problem and then "unrotate" and "untranslate" to get the solution to the original problem. Submit listing and output when the script `P3` is applied.