CS4220 Assignment 5 Due: 4/4/13 (Thur) at 11pm

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the solutions you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group. Each problem is worth 5 points. One point may be deducted for poor style.

Topics: Symmetric Eigenvalue Problem, SVD, Jacobi's Method, Sturm Sequence Property, Newton's Method, Method of Bisection

1 Parallel Jacobi SVD

The Jacobi method for computing the Schur decomposition of a symmetric matrix is easily adapted to compute the SVD of a square matrix. Download the implementation of cyclic Jacobi off the syllabus page. Instead of solving the 2x2 symmetric eigenvalue problem

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} d_p & 0 \\ 0 & d_q \end{bmatrix}$$

using `[c,s] = symSchur2(A,p,q)`, do this:

(a) Compute cosine-sine pair (c_1, s_1) so

$$\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} = S$$

is symmetric.

(b) Use `symSchur2` to compute the Schur decomposition of S:

$$\begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix}^T S \begin{bmatrix} c_2 & s_2 \\ -s_2 & c_2 \end{bmatrix} = \begin{bmatrix} d_p & 0 \\ 0 & d_q \end{bmatrix}$$

(c) Using the two rotations, build 2x2 orthogonal U_{pq} and V_{pq} so that

$$U_{pq}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} V_{pq} = \begin{bmatrix} \sigma_p & 0 \\ 0 & \sigma_q \end{bmatrix}, \qquad d_p \geq 0,\ \sigma_q \geq 0$$

Here is the overall framework for cyclic Jacobi, SVD:

$U \leftarrow I_n$, $V \leftarrow I_n$
while $\sqrt{\text{off}(A)} > tol \cdot \| A \|_F$
    for $p = 1{:}n-1$
        for $q = p+1{:}n$
            Solve the (p, q) SVD subproblem.
            Update A, U, and V.
        end
    end
end

Notice that the p and q loops takes us through all $N = n(n-1)/2$ off-diagonal index pairs. Here is a different ordering illustrated for the case $n = 8$:

Rotation Set	Rotations			
1	(1,2)	(3,4)	(5,6)	(7,8)
2	(1,4)	(2,6)	(3,8)	(5,7)
3	(1,6)	(4,8)	(2,7)	(3,5)
4	(1,8)	(6,7)	(4,5)	(2,3)
5	(1,7)	(8,5)	(6,3)	(4,2)
6	(1,5)	(7,3)	(8,2)	(6,4)
7	(1,3)	(5,2)	(7,4)	(8,6)

This is referred to as the *parallel ordering* because the subproblems and updates associated with any rotation set are decoupled and can be carried out in parallel.

To implement Jacobi with the parallel ordering, the p and q loops take the form

```
   for p=1:n-1
%     Generate the p-th rotation set.
%     Perform the updates associated with rotation set p
      for q=1:n/2
%        Solve the qth subproblem of the rotation set and update A, U, and V
```

Regarding the generation of the rotation sets, we return to the $n = 8$ example. If

$$(p_1, q_1) \quad (p_2, q_2) \quad (p_3, q_3) \quad (p_4, q_4)$$

is the "current" rotation set, then to get the "next" rotation set, permute the p's and q's as follows:

$$(p_1, q_2) \quad (q_1, q_3) \quad (p_2, q_4) \quad (p_3, p_4)$$

Verify that if you start out with $(1, 2)$, $(3, 4)$, $(5, 6)$, and $(7, 8)$ and repeat the process six times then you generate the rotation sets displayed above. Now generalize for any even n.

Write a function `[U,Sigma,V,sweeps] = ParJacSVD(A,tol)` that implements this method. You may assume that n is even. The output matrix `Sigma` should be the final A matrix, presumably within `tol` of being diagonal. The output parameter `sweeps` should be the number of times that the `while` body is executed. A test script `P1` will eventually be available from the website. Submit `ParJacSVD` to CMS.

2 Eigenvalues in an Interval

Download `ShowSturm`, `Sturm`, and `EigsToLeft` off the syllabus page. The "Sturm Sequence" theory behind `EigsToLeft` was discussed in class. `Sturm` can be used to find any specified eigenvalue of an unreduced symmetric tridiagonal matrix. (Unreduced means no zero subdiagonal entries.) By making effective use of `EigsToLeft`, develop a recursive implementation of the following function

```
   function e = EigsInInterval(T,L,R,nL,nR)
% T is an unreduced symmetric tridiagonal matrix
% L < R and neither L nor R are eigenvalues of T
% nL is the number of eigenvalues to the left of L
% nR is the number of eigenvalues to the left of R
% e is a column vector (possibly empty) consisting of all the eigenvalues in
% the interval [L,R] computed to full machine precision (as in the function Sturm).
```

Notes. If `nL = nR` then no eigenvalues are in the interval. If `nL = nR-1` then there is one eigenvalue in the interval. If `nL` $<$ `nR-1`, then proceed by recursion by computing the eigenvalues on $[L, mid]$ and and the eigenvalues on $[mid, R]$ where $mid = (L + R)/2$. You may assume that during the subdivision process, L and R are never eigenvalues. Submit `EigsInInterval` to CMS.

3 Cube Roots Via 400BC-Type Intuition

A Newton method step when applied to the function $f(x) = x^2 - A$ has the form

$$x_{new} = (x_{old} + A/x_{old})/2$$

Recall that this update can be interpreted as a way to make a rectangle with area A "more square." Assume that the "old" rectangle has length

$$L_{old} = x_{old}$$

and width

$$W_{old} = A/L_{old}.$$

The length of the new rectangle is the average

$$L_{new} = (L_{old} + W_{old})/2$$

and then we set

$$W_{new} = A/L_{new}$$

to ensure that the area is preserved. We are thus led to the following iteration for computing $z \approx \sqrt{A}$:

$L = A$
$W = 1$
while not converged
    $L = (L + W)/2$
    $W = A/L$
end
$z = L$

Take a look at the script `BetterRectangles` on the syllabus page for a display of its convergence behavior. Make sure you understand why this geometrically-derived procedure corresponds to Newton's method

$$x_{new} = x_{old} - \frac{f(x_{old})}{f'(x_{old})}$$

with $f(x) = x^2 - A$.

In this problem you are to replicate all this for the problem of computing cube roots. Start by taking a look at Newton's method for computing a zero of $f(x) = x^3 - V$ where we assume $V > 0$. Then "reverse engineer" the update formula so that it can be interpreted as a method for transforming a given L_{old} -by- W_{old} -by- H_{old} box (with volume V) into a "more cubical" L_{new} -by- W_{new} -by- H_{new} box (also with volume V. Develop the following function:

```
  function z = MyCubeRoot(V)
% V satisfies 1/8 <= V <= 1
% |z - c| <= 10^-14 where c = V^(1/3)
```

Model your implementation after `MySqrt`, a subfunction of `BetterRectangles` that is available off of the syllabus page. In particular, your implementation should maintain a triplet of variables `L`, `W`, and `H` that define the dimensions of the current, volume-V box.

To ensure that you are thinking like a calculus-free ancient Greek, you must follow these rules:

- No vectorization.
- No use of any Matlab function EXCEPT `max`, `min`, and `abs`.
- No use of the exponentiation operator.

Scoring will be based on how long it takes your function to compute the cube roots of all the values in `linspace(1/8,1,100000)` and whether or not the relative error criteria are satisfied. Submit your implementation of `MyCubeRoot` to CMS.

4 All Roots

Implement the following function:

```
  function z = AllRoots(alpha)
% alpha is a positive real number.
% z is a column vector consisting of all the zeros of the function
% f(x) = alpha*x - x that are in the interval [-1,1].
```

Make effective use of `fzero` and use the default tolerances. Submit your implementation to CMS. To get a handle on where the zeros might be, plot f for various values of α. And you may want to think about Rolle's theorem!