CS 621: Assignment 2

Due: Wednesday, September 28, 2005 (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/cs621/2005fa/`. 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. (The Determinant of an Orthogonal Matrix)

If $Q \in \mathbb{R}^{n\times n}$ is orthogonal, then $\det(Q) = \pm 1$. Complete the following function so that it performs as specified

```
  function z = OrthoDet(Q)
% Q is an n-by-n orthogonal matrix and z is its determinant.
  [L,U,P] = lu(Q);
       :
       :
```

The code you add after the LU factorization should involve $O(n^2)$ work and must not use any built-in MATLAB functions except things like `max`, `min`, `find`, etc. In particular, it should not invoke `det`! Some facts about the determinant that you may wish to use are (a) $\det(AB) = \det(A)\det(B)$ and (b) if you swap two rows (or columns) of a square matrix, then the determinant is multiplied by minus one. Test your implementation with the script `P1`.

P2. (1-norm Condition Estimation)

Suppose

$$T = \begin{bmatrix} \tau & w^T \\ 0 & S \end{bmatrix}$$

is nonsingular where $S \in \mathbb{R}^{k\times k}$ is upper triangular, $w \in \mathbb{R}^k$, and $\tau \in \mathbb{R}$. Assume that $d \in \mathbb{R}^k$ has unit 1-norm and that $Sy = d$. Show how to choose scalars α and β so that the solution to

$$Tz = \begin{bmatrix} \beta \\ \alpha d \end{bmatrix}$$

has the largest possible 1-norm subject to the constraint that the right hand side has unit 1-norm. We'll call this the "$\alpha\beta$-idea". Without loss of generality you can assume that $0 \le \alpha \le 1$. By repeatedly using the $\alpha\beta$-idea we can obtain a "large norm" solution for an arbitrary upper triangular matrix $T \in \mathbb{R}^{n\times n}$. Start with

$$(t_{nn})y = d$$

where $d = (1)$. Then update y and d using the $\alpha\beta$-idea to get a large norm solution to

$$\begin{bmatrix} t_{n-1,n-1} & t_{n-1,n} \\ 0 & t_{nn} \end{bmatrix} y = d$$

Then update the new y and d using the $\alpha\beta$-idea to get a large norm solution to

$$\begin{bmatrix} t_{n-2,n-2} & t_{n-2,n-1} & t_{n-2,n} \\ 0 & t_{n-1,n-1} & t_{n-1,n} \\ 0 & 0 & t_{nn} \end{bmatrix} y = d$$

And so on. In the spirit of condition estimation (see GVL §3.5.4) we would expect $\kappa_1(T) = \| T \|_1 \| T^{-1} \|_1 \approx \| T \|_1 \| y_{final} \|_1$. Using these notions, implement

```
  function mu = Cond1Norm(T)
% T is an n-by-n nonsingular upper triangular matrix.
% mu is an estimate of its 1-norm condition number.
```

and test it with the script `P2`.

P3. (A Markov Problem with Low rank Change)

Suppose $f, g \in \mathbb{R}^n$ and that $(f_1, g_1), \ldots, (f_n, g_n)$ are distinct. Define the matrix $C \in \mathbb{R}^{n \times n}$ by

$$c_{ij} = \begin{cases} 0 & \text{if } i = j \\ \\ 1/((f_i - f_j)^2 + (g_i - g_j)^2) & \text{if } i \neq j \end{cases}$$

and the vector $d \in \mathbb{R}^n$ by

$$d_j = \sum_{i=1}^{n} c_{ij}$$

Three observations:

- If you move (f_1, g_1) then the C matrix changes by a rank-2 matrix.
- If $A = CD^{-1}$ where $D = \text{diag}(d_1, \ldots, d_n)$, then A is a stochastic matrix, i.e., it has unit column sums and nonnegative entries.
- Think of a_{ij} as the probability that a flea hops from "landing point" (f_j, g_j) to landing point (f_i, g_i), with a kind of inverse square law underlying the hopping probabilities. (A flea is much more likely to hop to a nearby landing point.) To emphasize the dependence of A on the coordinates of the underlying landing points, we use the notation $A(f, g)$.

Develop a method for solving $Ax = b$ that involves the LU factorization of C, $PC = LU$. This will enable you to explore efficiently (via the Sherman-Morrison-Woodbury formula) how the solution changes if we move (f_1, g_1). Using these ideas, implement the following function so that it performs as specified.

```
  function [x0,X] = MarkovSol(f,g,u,v,b)
% f, g, and b are column n-vectors
% u and v are column q-vectors
% x0 = A(f,g)\b
% X is an n-by-q matrix with the property that if e1 is the first column of eye(n,n)
% then X(:,j) = A(f+u(j)*e1,g+v(j)*e1)\b for j=1:q
```

Test your implementation with the script `P3`.

Go-the-Distance 2. (An SVD Nearness Problem)

Complete the following function so that it performs as specified.

```
  function A0 = nearest(A)
% A is an n-by-n matrix
% A0 minimizes f(B) = norm(A - B,'fro') subject to the constraint that det(B) = abs(det(A))
```

Justify your method as best you can. Submit your implementation as an email attachment. If you have a write-up, submit that along with the rest of the assignment.