CS 6210: Assignment 1

Due: Friday, September 3, 2010 (In Class or in Upson 5153 by 4pm)

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 at `http://www.cs.cornell.edu/courses/cs6210/2010fa/`. 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. (Spotting Matrix Operations)

Assume that $E, F, G, H \in \mathbb{R}^{n\times n}$ and that $A \in \mathbb{R}^{n\times n}$ is defined by

$$a_{ij} = \sum_{k_1=1}^{n}\sum_{k_2=1}^{n}\sum_{k_3=1}^{n} E(k_1,i)F(k_1,i)G(k_2,k_1)H(k_2,k_3)F(k_2,k_3)G(k_3,j)$$

Write a function `A = MatrixProdFast(E,F,G,H)` that returns the matrix A defined by this expression. Make it flop-efficient and exploit MATLAB's vectorizing capabilities. A function `MatrixProdSlow` that does the same thing is on the course website together with a test script `P1`. Note that `MatrixProdSlow` requires $O(n^5)$ flops. If you are successful, your implementation of `MatrixProdFast` will involve no loops and require $O(n^3)$ flops. Submit a listing of `MatrixProdFast` together with the output produced when `P1` is run. *Hint.* Suppose $W \in \mathbb{R}^{n\times n}$ is defined by

$$w_{ij} = \sum_{p=1}^{n}\sum_{q=1}^{n} x_{ip}y_{pq}z_{qj}$$

where $X, Y, Z \in \mathbb{R}^{n\times n}$. If we use this formula for each w_{ij} then it would require $O(n^4)$ flops to set up W. On the other hand,

$$w_{ij} = \sum_{p=1}^{n} x_{ip}\left(\sum_{q=1}^{n} y_{pq}z_{qj}\right) = \sum_{p=1}^{n} x_{ip}u_{pj}$$

where $U = YZ$. Thus, $W = XU = XYZ$ and only $O(n^3)$ flops are required. In the given problem, you will be reasoning like that. Be on the look out for transposes and pointwise products, e.g., $X.*Y$.

P2. (A Rank-k Correction)

If $A \in \mathbb{R}^{n\times n}$, $u \in \mathbb{R}^n$, and $v \in \mathbb{R}^n$ are given and k is a positive integer, then there exist $X \in \mathbb{R}^{n\times k}$ and $Y \in \mathbb{R}^{n\times k}$ such that $(A + uv^T)^k = A^k + XY^T$. Complete the following function so that it performs as specified:

```
function [X,Y] = CorrectionTerm(A,u,v,k)
% A is n-by-n, u and v are n-by-1, and k is a positive integer.
% X and Y are n-by-k such that (A+u*v')^k = A^k + X*Y'.
```

Test your implementation with the test script `P2`. Submit listing and output. *Hint.* Begin your derivation of an algorithm by taking a look at $A^{k+1} + \tilde{X}\tilde{Y}^T = (A^k + XY^T)(A + uv^T)$.

P3. (Matrix Polynomial Times Vector)

Recall *Horner's* nested multiplication scheme that is normally used for polynomial evaluation. To compute $p(x) = c_1 + c_2x + \cdots + c_dx^{d-1}$ at $x = z$ we proceed as follows:

```
pval = c(d);
for k=d-1:-1:1
   pval = z*pval + c(k);
end
```

Develop an efficient, fully vectorized saxpy implementation of the following MATLAB function:

```
  function y = Polyvec(c,A,p)
% c is a column m-vector
% A is an n-by-n matrix with lower bandwidth p, assume pm<n
% y is the first column of c(1)I + c(2)A + ... + c(m)A^{m-1}
```

Use the Horner idea and exploit A's band structure. Submit listing and output when the test script `P3` is applied.

Challenge Problem 1.

Recall that if $F, G \in \mathbb{R}^{n\times k}$ and $x \in \mathbb{R}^n$, then the n-by-n matrix-vector product $y = Ax$ where $A = FG^T$ can be computed in $4nk$ flops since $y = Ax = FG^Tx = F(G^Tx)$. This is an important observation if $k << n$ and we will refer to it as the *rank-k product fact*.

Assume $P, Q \in \mathbb{R}^{n\times k}$ and that $D \in \mathbb{R}^{n\times n}$ is diagonal. Define the matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ by

$$a_{ij} = \begin{cases} (PQ^T)_{ij} & \text{if } i < j \\ D_{ii} & \text{if } i = j \\ (QP^T)_{ij} & \text{if } i > j \end{cases}$$

We say that a matrix of this form has *Property S*. The challenge is to write a function `y = PropSprod(P,Q,d,x)` that computes the matrix-vector product $y = Ax$ in $O(nk)$ flops.

Suppose we partition $y = Ax$ as follows

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 \\ A_{21}x_1 + A_{22}x_2 \end{bmatrix}$$

where both $A_{11} \in \mathbb{R}^{n_1\times n_1}$ and $A_{22} \in \mathbb{R}^{n_2\times n_2}$. Note that both A_{12} and A_{21} have the form FG^T where the two factors have k columns. Thus, the products $A_{12}x_2$ and $A_{21}x_1$ can be computed efficiently using the rank-k product fact. The blocks A_{11} and A_{22} have Property S so the products $A_{11}x_1$ and $A_{22}x_2$ can be computed efficiently by recursively calling `PropSprod`.

Implement the function `y = PropSprod(P,Q,d,x)` exploiting these ideas. Here, P and Q are n-by-k and d is a column n-vector that defines the diagonal matrix D, i.e., $D = \text{diag}(d_1, \ldots, d_n)$.

Submit listing and the output when the test script `C1` is applied. Include comments on the number of flops that are approximately required when `PropSprod` is called. Also comment on the rationale behind your definition of the base case. (You probably do not want to recur all the way down to $n = 1$.) *Your Challenge Problem submission must not be more than two pages total.*

For hints on how to organize a recursive matrix product, check out the discussion of the Strassen algorithm in §1.3 of GVL.