CS 6210 Assignment 1 Due: 9/9/15 (Wed) at 11pm

Scoring for each problem is on a 0-to-5 scale (5 = complete success, 4 = overlooked a small detail, 3 = good job on half the problem, 2 = OK job on half the problem, 1 = germ of a relevant solution idea, 0 = missed the point of the problem.) Independent of this, one point will be deducted for insufficiently commented code. This will be automatic if any function specification is inadequate, i.e., if the input and output parameters are not fully described. If test scripts are supplied then they are posted on the course website `http://www.cs.cornell.edu/courses/cs6210/2015fa/`. All solution M-Files must be submitted through the CMS system. You are allowed to discuss *background* issues with other students, but the codes you submit must be your own. Never show your code to another student either directly or indirectly.

Topics: Coding in Matlab, permutations, Matrix-Matrix products, Matrix-Vector Products, the SVD, the Fast Haar Wavelet Transform.

1 New Diagonal

Assume that $A \in \mathbb{R}^{n\times n}$ and that $P = I_n(v,:)$ where v is a permutation of $[\,1,\ 2, \ldots,\ n\,]$. Write a MATLAB function `d = NewDiag(A,v,k)` that returns `diag(B)` where $B = [P^k]^T A P^k$ and $P = I_n(v,:)$. Assume $k \geq 0$. Submit `NewDiag` to CMS. To get started, check out P^TAP for a random $n = 3$ problem. Review §1.28-1.2.11.

2 Squaring a Hermitian Matrix.

$A = B + iC \in \mathbb{C}^{n\times n}$ is *Hermitian* if it equals its conjugate transpose, i.e., $a_{ji} = \bar{a}_{ij}$. It follows that the real part B is symmetric ($B = B^T$) and the imaginary part C is skew-symmetric ($B^T = -B$). Since the diagonal of a real skew-symmetric matrix is zero we can represent an n-by-n complex Hermitian matrix with a single real array. For example, if

$$A \;=\; B + i\cdot C \;+\; \left[\begin{array}{cccc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{12} & b_{22} & b_{23} & b_{24} \\ b_{13} & b_{23} & b_{33} & b_{34} \\ b_{14} & b_{24} & b_{34} & b_{44} \end{array}\right] \;+\; i\cdot \left[\begin{array}{cccc} 0 & -c_{21} & -c_{31} & -c_{41} \\ c_{21} & 0 & -c_{32} & -c_{42} \\ c_{31} & c_{32} & 0 & -c_{43} \\ c_{41} & c_{42} & c_{43} & 0 \end{array}\right]$$

then we can represent A with

$$A_{herm} \;=\; \left[\begin{array}{cccc} b_{11} & b_{12} & b_{13} & b_{14} \\ c_{21} & b_{22} & b_{23} & b_{24} \\ c_{31} & c_{32} & b_{33} & b_{34} \\ c_{41} & c_{42} & c_{43} & b_{44} \end{array}\right].$$

We will call this the *herm representation.*

It is easy to verify that if $A = B + iC$ is Hermitian, then

$$A^2 = (B + iC)(B + iC) \;=\; (B\cdot B - C\cdot C) \;+\; i\cdot(B\cdot C + C\cdot B)$$

is also Hermitian. Complete the following function so that it performs as specified:

```
   function Y_herm = HermSquare(X_herm)
 % X_herm is the herm representation of an nxn Hermitian matrix X
 % Y_herm is the herm representation of the square of X
```

Note that if the structure of B and C is ignored, then three real matrix-matrix products are required giving an algorithm that requires $6n^2$ flops. Submit your implementation of `HermSquare` to CMS. Your implementation should exploit structure and reduce the "6" to a smaller number. What is that number? Include a comment in your submission that indicates this number. Remember that MATLAB supports inner products. Points will be deducted if you compute inner products with loops. Finally, your implementation should only involve $2n^2$ storage. (A few extra work vectors are fine.) Hint. Given `X_herm`, think about how would you assemble the i-row of B and the ith row of C for a given i.

3 Computing a Certain Unit Vector

Suppose $U^TAV = \Sigma$ is the SVD of $A \in \mathbb{R}^{n\times n}$. Recall that if $v_k = V(:,k)$ and $u_k = U(:,k)$ and $\sigma = \text{diag}(\sigma_1,\ldots,\sigma_n)$, then

$$Av_k = \sigma_k u_k \qquad k = 1{:}n.$$

Given μ that satisfies $\sigma_n \le \mu \le \sigma_1$, our goal is to compute a unit 2-norm vector

$$x \in \text{span}\{v_1, v_n\}$$

so that

$$\| Ax \|_2 = \mu.$$

In particular, implement the following function so that it performs as specified:

```
  function x = SpecialVec(A,mu)
% A is nxn with SVD U*S*V'.
% mu is a real that satisfies S(n,n) <= mu <= S(1,1)
% x is a column n-vector with the property that  (a) norm(x) = 1, (b) x is a
% linear combination of V(:,1) and V(:,n), and (c) norm(A*x) = mu.
```

Hint. Write $x = cv_1 + sv_n$ where $c^2 + s^2 = 1$ and solve for c and s. The solution is not unique–don't worry about it! Submit `SpecialVec` to CMS.

4 Compression Using Haar Wavelets

This problem is about the fast implementation of Haar Wavelet Transform $y = W_n x$ that is outlined in §1.4.3. The matrix $W_n \in \mathbb{R}^{n\times n}$ is defined recursively:

$$W_n = \begin{cases} \left[\; W_m \otimes \begin{pmatrix} 1 \\ 1 \end{pmatrix} \;\middle|\; I_m \otimes \begin{pmatrix} 1 \\ -1 \end{pmatrix} \;\right] & \text{if } n = 2m > 1 \\ [\,1\,] & \text{if } n = 1. \end{cases}$$

This means that n is a power of two. Here are some examples:

$$W_2 = \left[\begin{array}{c|c} 1 & 1 \\ \hline 1 & -1 \end{array}\right]$$

$$W_4 = \left[\begin{array}{cc|cc} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ \hline 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{array}\right]$$

$$W_8 = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ \hline 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{array}\right]$$

The matrix W_n has a special block structure that is exposed through the perfect shuffle permutation $\mathcal{P}_{2,m}$ and its inverse, the odd-even sort permutation $\mathcal{P}_{2,m}^T$. To see this suppose $n = 8$ and recall that

$$\mathcal{P}_{2,4}^T \;=\; I_8([1\,3\,5\,7\,2\,4\,6\,8],:) \;=\; \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

and observe that $\mathcal{P}_{2,m}x = [x_1, x_3, x_5, x_7, x_2, x_4, x_6, x_8]^T = [x(1{:}2{:}n); x(2{:}2{:}n)]$. It is easy to verify that

$$\mathcal{P}_{2,4}^T W_8 \;=\; \left[\begin{array}{rrrr|rrrr} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{array}\right] \;=\; \left[\begin{array}{c|c} W_4 & I_4 \\ \hline W_4 & -I_4 \end{array}\right].$$

In general, if $n = 2m$ then

$$\mathcal{P}_{2,m}^T W_n \;=\; \left[\begin{array}{c|c} W_m & I_m \\ \hline W_m & -I_m \end{array}\right]. \qquad (1)$$

The *Haar transform* of $x \in \mathbb{R}^n$ is the matrix-vector product $y = W_n x$. The equation $\mathcal{P}_{2,m}^T y = \mathcal{P}_{2,m}^T W_n x$ looks like this:

$$\left[\begin{array}{c} y(1{:}2{:}n) \\ y(2{:}2{:}n) \end{array}\right] = \left[\begin{array}{c|c} W_m & I_m \\ \hline W_m & -I_m \end{array}\right]\left[\begin{array}{c} x(1{:}m) \\ x(m+1{:}n) \end{array}\right] = \left[\begin{array}{c} W_m x(1{:}m) + x(m+1{:}n) \\ W_m x(1{:}m) - x(m+1{:}n) \end{array}\right]$$

This is the math behind the Fast Haar Wavelet Transform:

```
  function y = FHT(x)
% y is a column n-vector that is the Haar transform of the column n-vector x.
% n is a power of two
if n==1
   y = x;
else
   m = n/2;
   y = zeros(n,1);
   z = FHT(x(1:m));
   y(1:2:n) = z + x(m+1:n);
   y(2:2:n) = z - x(m+1:n);
end
```

Note that if $n = 2m$ and `FHT` requires f_n flops when applied to an n-vector, then $f_n = f_m + n$. Since $f_2 = 2$ we see that $f_4 = 2 + 4 = 6$, $f_8 = 6 + 8 = 14$, $f_{16} = 14 + 16 = 30$, etc. Proof-by-small-examples tells us that the Fast Haar Transform is linear, in particular, $W_n x$ involves $2n - 2$ flops.

Now let's consider the *inverse* Haar transform. Here we are given $y \in \mathbb{R}^n$ and wish to determine $x \in \mathbb{R}^n$ so that $y = W_n x$. Let's figure out W_n^{-1} from equation (1) which we rewrite as

$$W_n \;=\; \mathcal{P}_{2,m}\left[\begin{array}{c|c} W_m & I_m \\ \hline W_m & -I_m \end{array}\right] \;=\; \mathcal{P}_{2,m}\left[\begin{array}{c|c} I_m & I_m \\ \hline I_m & -I_m \end{array}\right]\left[\begin{array}{c|c} W_m & 0 \\ \hline 0 & I_m \end{array}\right].$$

Taking the inverse of both sides and using the fact that $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$ we see that

$$W_n^{-1} = \left[\begin{array}{c|c} W_m^{-1} & 0 \\ \hline 0 & I_m \end{array}\right] \left[\begin{array}{c|c} I_m/2 & I_m/2 \\ \hline I_m/2 & -I_m/2 \end{array}\right] \mathcal{P}_{2,m}^T$$

It follows that

$$\begin{aligned} x = W_n^{-1}y &= \left[\begin{array}{c|c} W_m^{-1} & 0 \\ \hline 0 & I_m \end{array}\right] \left[\begin{array}{c|c} I_m/2 & I_m/2 \\ \hline I_m/2 & -I_m/2 \end{array}\right] \mathcal{P}_{2,m}^T y \\ &= \left[\begin{array}{c|c} W_m^{-1} & 0 \\ \hline 0 & I_m \end{array}\right] \left[\begin{array}{c|c} I_m/2 & I_m/2 \\ \hline I_m/2 & -I_m/2 \end{array}\right] \left[\begin{array}{c} y(1{:}2{:}n) \\ y(2{:}2{:}n) \end{array}\right] \\ &= \left[\begin{array}{c|c} W_m^{-1} & 0 \\ \hline 0 & I_m \end{array}\right] \left[\begin{array}{c} (y(1{:}2{:}n) + y(2{:}2{:}n))/2 \\ (y(1{:}2{:}n) - y(2{:}2{:}n))/2 \end{array}\right] \\ &= \left[\begin{array}{c} W_m^{-1}((y(1{:}2{:}n) + y(2{:}2{:}n))/2) \\ (y(1{:}2{:}n) - y(2{:}2{:}n))/2 \end{array}\right] \end{aligned}$$

This is the math behind

```
  function x = IFHT(y)
% x is a column n-vector that is the inverse Haar transform of the column n-vector y.
% n is a power of two
n = length(y);
if n==1
   x = y;
else
   yO = y(1:2:n);
   yE = y(2:2:n);
   x = [IFHT((yO+yE)/2) ; (yO-yE)/2];
end
```

If $n = 2m$ and f_n is the number of flops required when `IFHT` is applied to an n-vector, then $f_n = f_m + 2n$. Since $f_2 = 3$ it follows that $f_4 = 3 + 8 = 11$, $f_8 = 11 + 16 = 27$, $f_{16} = 27 + 32 = 59$, etc we conclude that $f_n = 4n - 5$.

We now proceed to describe the function that you are to write for this part of the assignment. It involves playing with products of the form

$$Y \;=\; W_{n_1} X W_{n_2}^T \qquad X \in \mathbb{R}^{n_1\times n_2}, Y \in \mathbb{R}^{n_1\times n_2}$$

where we assume that both n_1 and n_2 are powers of two. In particular, you are to write a function

```
function [Ytilde,CF] = HaarThreshApprox(Y,tau)
```

that returns the *Haar threshold approximation* $\tilde{Y}$ and the associated *compression factor* `CF`. To define the matrix $\tilde{Y}$, assume that $X \in \mathbb{R}^{n_1\times n_2}$ solves

$$Y = W_{n_1} X W_{n_2}^T.$$

and that $m_X = \max|x_{ij}|$. Given τ satisfying $0 \le \tau \le 1$, the Haar threshold approximation is given by

$$\tilde{Y} \;=\; W_{n_1} \tilde{X} W_{n_2}^T \tag{2}$$

where

$$\tilde{x}_{ij} \;=\; \begin{cases} 0 & \text{if} |x_{ij}| \le \tau m_X \\ x_{ij} & \text{otherwise} \end{cases} .$$

Thus, $\tilde{X}$ is obtained from X by setting its small elements to zero. An element is "small" if it is less than τ times the largest element in X. The compression factor is the ratio $n_1 n_2 / n_z$ where n_z is the number of nonzero entries in $\tilde{X}$. Depending on the matrix Y and the threshold parameter τ, the number of nonzeros in the matrix $\tilde{X}$ may be much less than $n_1 n_2$ rendering a large compression factor. Thus, we can think of $\tilde{Y}$ as a *data sparse* representation of Y. If we are in possession of $\tilde{X}$ then we can reconstruct $\tilde{Y}$ from (2).

Before you start coding up `HaarThreshApprox`, develop answers to the following questions:

- How would you use `IFHT` to solve $F = W_{n_1} G$ for G given F?
- How would you use `IFHT` to solve $F = G W_{n_2}^T$ for G given F? Hint. Take transposes.
- How would you use `FHT` to compute$F = W_{n_1} G$ given G?
- How would you use `FHT` to compute$F = G W_{n_2}$ given G? Hint. Take transposes.

A test script `P4` and the functions `FHT` and `IFHT` are provided on the course website. After you get things working, develop vectorized versions of `IFHT` and `FHT` that can handle (without loops) matrix inputs. These can be subfunctions of the finished `HaarThreshApprox` that you are to submit to CMS. No penalty for extra matrix storage, but you might want to see how frugal you can be with memory. Finally, we mention that nonrecursive implementations of `FHT` and `IFHT` are possible (but not required. See Problem 1.4.5.