CS4220 Assignment 1 Due: 2/2/14 (Sunday) 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 submitted function is worth 5 points. Points may be deducted for poor style.

Topics: Elementary matrix operations in MATLAB. Matrix-vector products. Block matrices. Discrete cosine transform. Jpeg compression. Vectorization.

1 The Discrete Cosine Transform

The *discrete cosine transform* of a vector $x \in \mathbb{R}^n$ is a matrix-vector product $y = C_n x$ where the *DCT* matrix $C_n \in \mathbb{R}^{n\times n}$ is the defined by

$$C_n = (c_{ij}) \qquad c_{ij} = \cos\left(\frac{(i-1)(2j-1)}{2n}\pi\right).$$

The DCT matrix can be set up as follows in MATLAB:

```
C = cos((pi/(2*n))*(0:n-1)'*(1:2:2*n-1))
```

This matrix is highly structured making it possible to execute the matrix-vector product $C_n x$ fast. In this problem you will come to appreciate this for the case $n = 8$. The 8-point DCT is central to JPEG image compression as you will see in the second part of this assignment.

If $c_j = \cos(j\pi/16)$, for $j = 1{:}7$, then by using facts like $c_{j+16} = -c_j$ it can be shown that

$$C_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2 \\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4 \\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6 \\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \end{bmatrix}.$$

In general, an $n = 8$ matrix-vector product requires $2n^2 = 128$ flops, but if the DCT matrix is involved, then work can be reduced by about 60% from the flop point of view. The key idea is to observe that C_8 can be written as a product $C_8 = T_3T_2T_1$ where

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix} \qquad T_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_1 & c_3 & c_5 & c_7 \\ 0 & 0 & c_2 & c_6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_3 & -c_7 & -c_1 & -c_5 \\ c_4 & -c_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_5 & -c_1 & c_7 & c_3 \\ 0 & 0 & c_6 & -c_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_7 & -c_5 & c_3 & -c_1 \end{bmatrix}$$

The DCT $y = C_8 x = T_3 T_2 T_1 x = T_3(T_2(T_1 x))$ turns into a 3-step process:

- $u = T_1 x$, this involves 8 flops.
- $v = T_2 u$, this involves 4 flops.
- $y = T_3 v$, this involves 37 flops.

Implement the following function so that it performs as specified.

```
function Y = DCT8(X)
% X is an 8-by-m matrix
% Y = C*X where C is the 8-by-8 DCT matrix.
```

To receive full credit, your implementation should be vectorized. If you succeed in this regard, then your implementation will not involve any loops. Submit `DCT8` to CMS.

2 The Inverse Discrete Cosine Transform

An important property of C_n is that its inverse is a column scaling of its transpose. Indeed, it can be shown that

$$C_n C_n^T = D_n = \text{diag}(n, n/2, \ldots, n/2) \tag{1}$$

and so $C_n(C_n^T D_n^{-1}) = I_n$, i.e., $C_n^{-1} = C_n^T D_n^{-1}$. Let's check this out for $n = 3$:

$$C_3 = \begin{bmatrix} 1 & 1 & 1 \\ \cos(30^o) & \cos(90^o) & \cos(150^o) \\ \cos(60^o) & \cos(180^o) & \cos(300^o) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ \sqrt{3}/2 & 0 & -\sqrt{3}/2 \\ 1/2 & -1 & 1/2 \end{bmatrix}$$

$$C_3 C_3^T = D_3 = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3/2 & 0 \\ 0 & 0 & 3/2 \end{bmatrix}$$

$$C_3^{-1} = C_3^T D_3^{-1} = \begin{bmatrix} 1 & \sqrt{3}/2 & 1/2 \\ 1 & 0 & -1 \\ 1 & -\sqrt{3}/2 & 1/2 \end{bmatrix} \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 2/3 & 0 \\ 0 & 0 & 2/3 \end{bmatrix} = \begin{bmatrix} 1/3 & 1/\sqrt{3} & 1/3 \\ 1/3 & 0 & -2/3 \\ 1/3 & -1/\sqrt{3} & 1/3 \end{bmatrix}$$

It is easy to see that these recipes for C_3 and C_3^{-1} satisfy $C_3 C_3^{-1} = C_3^{-1} C_3 = I_3$.

Implement the following function so that it performs as specified:

```
function Y = IDCT8(X)
% X is an 8-by-m matrix
% Y satisfies C*Y = X where C is the 8-by-8 DCT matrix.
```

Your implementation should be fully vectorized (no loops) and submitted to CMS. Hint. Exploit the fact that $C_8 = T_3 T_2 T_1$ and $C_8^{-1} = C_n^T D_n^{-1}$.

3 JPEG Compression

This problem gives you a snapshot of how JPEG image compression works. Here is the essential story. A picture is a matrix of pixels. We regard the matrix as block matrix with 8x8 blocks. We perform operations on each of the blocks and discover how to "capture its essence" with many fewer than 64 numbers.

The starting point for us is an m-by-n integer matrix A whose entries are between 0 and 255. We'll call this the *picture matrix*. For a color image there are three picture matrices, one for red, one for green, and one

for blue. We will present the ideas as if the image is black and white. The calculations that we outline below would be applied separately to the red, green and blue matrices.

Entry a_{ij} is the grayness value of pixel (i, j) with 0 corresponding to black and 255 corresponding to white. Assume $m = 8m_1$ and $n = 8n_1$ and think of A as an m_1-by-n_1 block matrix with 8-by-8 blocks, e.g.,

$$A = \begin{array}{|c|c|c|c|c|} \hline A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ \hline A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ \hline A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\ \hline A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \\ \hline \end{array}$$

$m = 32$, $n = 40$

$A_{34} = A(17{:}24, 25{:}32)$

JPEG compression transforms the picture matrix A "block-by-block" to another matrix B

$$\begin{array}{|c|c|c|c|c|} \hline A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ \hline A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ \hline A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \\ \hline A_{41} & A_{42} & A_{43} & A_{44} & A_{45} \\ \hline \end{array} \quad \rightarrow \quad \begin{array}{|c|c|c|c|c|} \hline B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\ \hline B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\ \hline B_{31} & B_{32} & B_{33} & B_{34} & B_{35} \\ \hline B_{41} & B_{42} & B_{43} & B_{44} & B_{45} \\ \hline \end{array}$$

according to the rule

$$B_{ij} = \text{round}\left(\left(C_8 A_{ij} C_8^T\right) ./ Q \right)$$

where Q is an 8-by-8 *quantization matrix*. A popular choice is

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

We refer to B as the *JPEG matrix*. Let's take a look at the computation of the B_{ij} using an example. Here is a typical block from A, i.e., an 8-by-8 patch from the picture:

```
AIJ =  75  63  66  67  66  71  83  95
       72  64  71  76  78  82  88  90
       79  76  78  77  74  76  85  91
       83  79  76  67  60  64  79  93
       83  66  65  61  58  64  78  89
       77  71  72  80  91  95  89  79
       79  89  95 100 101  98  92  84
       77 105 109 107  97  89  88  90
```

We take the DCT of the columns $C_8 A_{ij}$ and then the rows $(C_8 A_{ij}) C_8^T$ and get

```
5162.000   -143.657     79.399    -56.735     21.213     26.844      6.911      5.928
-272.682    -70.761    131.075      4.546     40.651     28.310     20.479      9.593
 195.475     -8.800   -111.240    -23.561    -16.378    -15.552    -11.692     -7.375
-126.415    -58.711     17.540     77.309     21.566     17.159     13.655      6.470
 -96.167     43.771     97.150    -68.698      8.000     -4.257      1.972     -2.129
  44.224     18.136      0.204     -8.276     -5.666     -6.305     -2.313     -1.205
   6.283      0.644     -6.692      8.778     -2.957     -7.008     -3.760     -2.146
 -20.590      8.158     21.505    -11.816      9.812      7.940      3.151      1.757
```

Notice that entries range from big to small as we move from the upper left corner to the lower right corner. This is the "miracle" of the DCT; it represents A_{ij} in a particularly handy basis so that the important non-zero information is "compressed" into a small part of the matrix.

Now look at the quantization matrix Q above. The pointwise divide $(C_8 A_{ij} C_8^T)./Q$ has the effect of accentuating the big-to-small drift:

```
322.6250  -13.0597    7.9399   -3.5460    0.8839    0.6711    0.1355    0.0972
-22.7235   -5.8967    9.3625    0.2393    1.5635    0.4881    0.3413    0.1744
 13.9625   -0.6769   -6.9525   -0.9817   -0.4095   -0.2728   -0.1695   -0.1317
 -9.0296   -3.4536    0.7973    2.6658    0.4229    0.1972    0.1707    0.1044
 -5.3426    1.9896    2.6257   -1.2268    0.1176   -0.0391    0.0192   -0.0276
  1.8427    0.5182    0.0037   -0.1293   -0.0699   -0.0606   -0.0205   -0.0131
  0.1282    0.0101   -0.0858    0.1009   -0.0287   -0.0579   -0.0313   -0.0212
 -0.2860    0.0887    0.2264   -0.1206    0.0876    0.0794    0.0306    0.0177
```

Rounding this produces

```
            323   -13     8    -4     1     1     0     0
            -23    -6     9     0     2     0     0     0
             14    -1    -7    -1     0     0     0     0
BIJ  =       -9    -3     1     3     0     0     0     0
             -5     2     3    -1     0     0     0     0
              2     1     0     0     0     0     0     0
              0     0     0     0     0     0     0     0
              0     0     0     0     0     0     0     0
```

This is typical. About 75% of B_{ij}'s entries are zero. The rest of the JPEG process (not part of this problem) involves a clever "concatenation" of the nonzero information in all the B_{ij}. In the end the number of bits required to represent the picture is typically reduced by a factor of about 20.

This takes care of the compression. How do we "decompress" a JPEG matrix? The idea is simply to visit each block and "undo" the Q-divide and the DCT's:

$$\tilde{A}_{ij} = C_8^{-1}(B_{ij}. * Q)(C_8^{-1})^T$$

The resulting block matrix $\tilde{A}$ encodes the reconstructed image. Complete the following functions so that they perform as specified:

```
  function B = JPEG(A)
% A is an mxn picture matrix and B is its mxn JPEG-compressed analog.
% Assumes that both m and n are divisible by 8.

  function B = IJPEG(A)
% A is mxn JPEG-compressed matrix and B is its JPEG decompressed analog.
% Assumes that both m and n are divisible by 8.
```

To receive full credit, your implementations must be fully vectorized. Here are some linear algebra/MATLAB pointers to help you in this regard:

- Although the function `DCT8` is designed to compute products of the form $Y = C_8X$, it can also be used to compute products of the form $G = FC_8^T$ where $F \in \mathbb{R}^{m\times 8}$. It's a 1-liner: `G = DCT8(F')'`. Likewise, `IDCT8` can be used to compute $G = F(C_8^{-1})^T$.

- There are obvious double-loop implementations of `JPEG` and `IJPEG` that process one A_{ij} at a time. But by using reshape you can avoid looping. Hint: `DCT8(reshape(A,8,m*n/8))` computes the products C_8A_{ij} over all i and j.

- The operations involving Q can also be done without loops by making use of the MATLAB function `kron`. Here's a hint:

$$\text{R = kron(ones(4,5),Q)} \qquad \leftrightarrow \qquad R = \begin{bmatrix} Q & Q & Q & Q & Q \\ Q & Q & Q & Q & Q \\ Q & Q & Q & Q & Q \\ Q & Q & Q & Q & Q \end{bmatrix}$$

If vectorization is a challenge, then start by getting a working non-vectorized implementation. Then start converting the loops.

Submit both `JPEG` and `IJPEG` to CMS. Note: we supply you with a sample jpeg file, but you can try out others for fun. Use `Trim` to ensure that the pixel dimensions are divisible by 8.