Department of Physics
Umeå University
October 24, 2018

Matrix computations and applications
Assignment 4

Gustav Nystedt - oi14gnt

Department of computer science
Supervisors: Niclas Börlin & Andrii Dmytryshyn

Contents

1 Introduction

In this assignment we look into some theoretical properties of orthogonality/orthonormality as well as symmetric- and Hermitian matrices. Further, we implement jpeg-compression through DCT compression and quantization. In the end we look at how the visible quality depends on how much we choose to impair the quality of the image.

2 Theory

2.1

We want to show that the vectors

$$v_1 = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} \text{ and } v_2 = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix}$$

are orthonormal for any $\theta \in \mathbb{R}$:

$$\bar{v}_1 \cdot \bar{v}_2 = -cos(\theta)sin(\theta) + cos(\theta)sin(\theta) = 0 \quad \Rightarrow \quad \text{orthogonal!} \tag{1}$$

Further,

$$\begin{cases} \|\bar{v}_1\|_2 = \sqrt{\cos^2\theta + \sin^2\theta} = 1 \\ \|\bar{v}_2\|_2 = \sqrt{(-\sin\theta)^2 + \cos^2\theta} = 1 \end{cases} \quad \Rightarrow \quad \text{orthonormal!} \tag{2}$$

2.2

$$Z = \frac{1}{\sqrt{2}} \begin{pmatrix} Q & Q \\ -Q & Q \end{pmatrix} \quad \Rightarrow \quad Z^T = \frac{1}{\sqrt{2}} \begin{pmatrix} Q & -Q \\ Q & Q \end{pmatrix} \tag{3}$$

$$\Rightarrow \quad ZZ^T = \frac{1}{\sqrt{2}} \begin{pmatrix} Q & Q \\ -Q & Q \end{pmatrix} \begin{pmatrix} Q & -Q \\ Q & Q \end{pmatrix} \quad = \quad \frac{1}{2} \begin{pmatrix} 2Q^2 & 0 \\ 0 & 2Q^2 \end{pmatrix} \tag{4}$$

$$= \quad \begin{pmatrix} Q^2 & 0 \\ 0 & Q^2 \end{pmatrix} \quad = \quad \begin{pmatrix} QQ^T & 0 \\ 0 & QQ^T \end{pmatrix} \quad = \quad \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \quad = \quad I \tag{5}$$

$\underline{Q.E.D.}$

2.3

$$m = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \theta = \frac{3}{4}\pi \quad \Rightarrow \quad m = \begin{pmatrix} -\sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} \\ \sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} \end{pmatrix} \tag{6}$$

$$\Rightarrow Z = \frac{1}{\sqrt{2}} \begin{pmatrix} -\sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} \\ \sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} & \sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} \\ \sqrt{\frac{2}{4}} & \sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} \\ -\sqrt{\frac{2}{4}} & \sqrt{\frac{2}{4}} & \sqrt{\frac{2}{4}} & -\sqrt{\frac{2}{4}} \end{pmatrix} = \frac{1}{2} \begin{pmatrix} -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 \end{pmatrix} \quad (7)$$

2.4

$A \in \mathbb{R}^{n\times n}$, find symmetric matrix R (R^T=R) and a skew-symmetric matrix S (S^T=-S) such that A = R + S. Below is an example:

$$R = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad S = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad \Rightarrow \quad A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}. \quad (8)$$

2.5

In this assignment we want to find an example of a hermitian matrix, R, and a skew-hermitian matrix, S, for a complex matrix A = R + S, $A \in \mathbb{C}^{n\times n}$.

$$R = \begin{pmatrix} 1 & i & 1+i \\ -i & -5 & 2-i \\ 1-i & 2+i & 3 \end{pmatrix} = \bar{R}^T, \quad (9)$$

$$S = \begin{pmatrix} i & 1-i & 2 \\ -1-i & 3i & i \\ -2 & i & 0 \end{pmatrix} = -\bar{S}^T \quad (10)$$

$$\Rightarrow \quad A = \begin{pmatrix} 1-i & -1+2i & -1+i \\ 1 & -5-3i & 2-2i \\ 3-i & 2 & 3 \end{pmatrix} \quad (11)$$

2.6

We want to prove that every eigenvalue of a Hermitian matrix A is real.

$$Az = \lambda z \quad (12)$$

Multiply by z^H:

$$z^H Az = \lambda z^H z \quad (13)$$

This gives us,

$$\begin{cases} z^H Az = \bar{z}^T(Az) = (Az)^T\bar{z} = z^T A^T \bar{z} \\ \lambda z^H z = \lambda \bar{z}^T z = \lambda \|z\| \end{cases} \quad (14)$$

$$\Rightarrow \quad z^T A^T \bar{z} = \lambda \|z\| \quad (15)$$

Taking complex conjugate of this expression gives:

$$\bar{z}^T \bar{A}^T z = \bar{\lambda}\|z\|. \tag{16}$$

Since A is Hermitian $\Rightarrow A^T = A$, which gives:

$$\bar{\lambda}\|z\| = \bar{z}^T A z = [\text{Eq. } 12] = \bar{z}^T \lambda z = \lambda \bar{z}^T z = \lambda\|z\| \tag{17}$$

$$\Rightarrow \ \bar{\lambda}\|z\| = \lambda\|z\|, \quad \underline{\text{note}}\text{: } \|z\| \neq 0 \tag{18}$$

$$\Rightarrow \ \bar{\lambda} = \lambda \ \Rightarrow \ \lambda \text{ is real.} \tag{19}$$

<u>*Q.E.D.*</u>

2.7

We want to prove that eigenvectors y and z of a Hermitian matrix A are orthogonal if they are associated with different eigenvalues. Define:

$$\begin{cases} Az = \lambda_1 z \\ Ay = \lambda_2 y \end{cases}, \tag{20}$$

and note that $A = A^H$ which implies,

$$\underline{\langle z, Ay\rangle} = Y^H A^H z = Y^H A z = Y^H \lambda_1 z = \lambda_1 y^H z = \underline{\lambda_1 \langle z, y\rangle}. \tag{21}$$

Further:

$$\langle z, Ay\rangle = \langle z, \lambda_2 y\rangle = \lambda_2^* \langle z, y\rangle = \lambda_2 \langle z, y\rangle \tag{22}$$

$$\Rightarrow \ \lambda_1 \langle z, y\rangle = \lambda_2 \langle z, y\rangle \tag{23}$$

$$\Rightarrow \ (\lambda_1 - \lambda_2)\langle z, y\rangle = 0. \tag{24}$$

Hence, we can conclude that

$$\lambda_1 \neq \lambda_2 \ \Rightarrow \ \langle z, y\rangle = 0 \quad \underline{\&} \quad \langle z, y\rangle \neq 0 \ \Rightarrow \ \lambda_1 = \lambda_2. \tag{25}$$

Thus, eigenvectors corresponding to different eigenvalues of Hermitian matrices are orthogonal.

<u>*Q.E.D.*</u>

2.8

The expressions given in the list below are orthogonal.

- **UV -** Because the product of two orthogonal matrices are itself orthogonal, by definition.
- **UVUV -** Because UV is orthogonal UVUV is orthogonal for the same reason.
- $\mathbf{U}^2\mathbf{V}^2$ **-** Because $U^2 = UU^T = I = V^2 \ \Rightarrow \ I \times I$, which is orthogonal.
- $\begin{bmatrix} \mathbf{U} & \mathbf{0} \\ \mathbf{0} & \mathbf{V} \end{bmatrix}$ **-** Since if Q $= \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix} = Q^T \ \Rightarrow \ QQ^T = \begin{bmatrix} U^2 & 0 \\ 0 & V^2 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$,

  which is <u>orthogonal</u>.

2.9

If A is a real matrix, $A \in \mathbb{R}^{n \times n}$, $C = A + iI$ is invertible, because the columns of C are linearly independent.

3 Applications

In this part of the assignment we have developed functions for compression and decompression of images, using JPEG standards. The functions developed are `jpegcompress.m` and `jpegdecompress.m` which can both be found in the appendix.

3.1 Tests

To answer the questions in the assignments, a test script called `testJpeg.m` was developed. This can also be found in the appendix. We use the image `mandrill.png` illustrated in Figure 1 to perform our tests on.

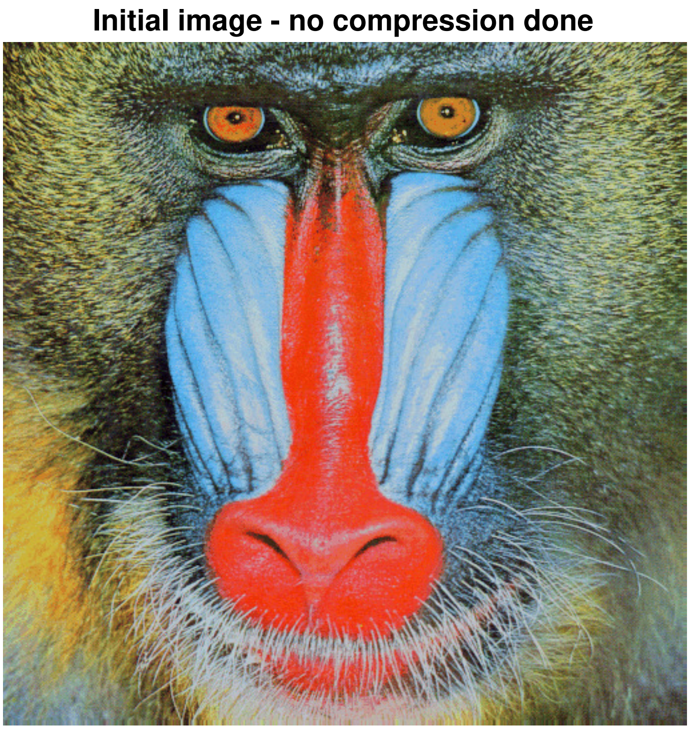

Figure 1: Original photo before compression.

3.1.1

By compressing the image using `jpegcompress.m` we obtain the matrix of quantized DCT coefficients for a specified quality number q. In this test we use q = 1, and after the DCT coefficients had been retrieved we can use them to call `jpegdecompress.m` with the same quality number. We then obtain a compressed version of our image, which is illustrated in Figure 2 below. So far, no visible difference can be detected in the compressed image.

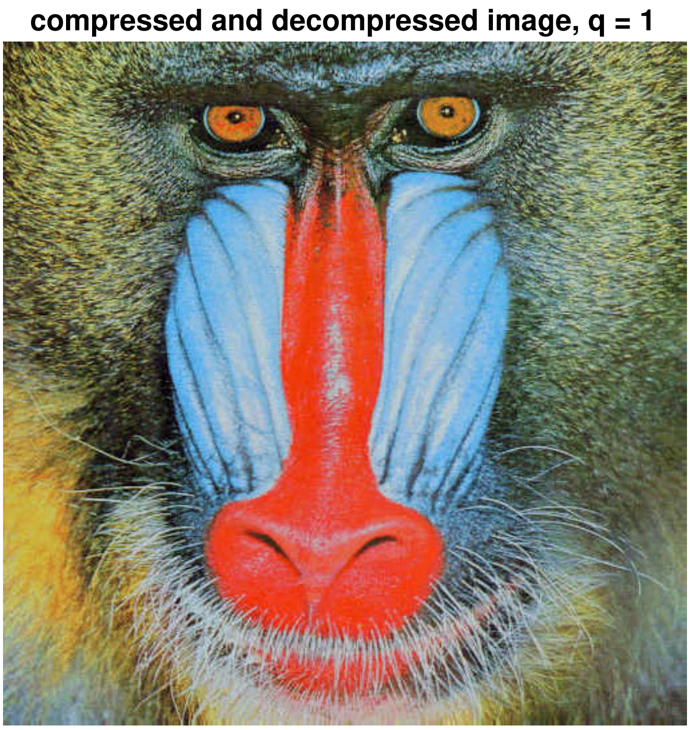

Figure 2: Image compressed using q = 1.

The quantized DCT coefficients (using q=1) of all three bands of the first tile is:

$$C_Y = \begin{bmatrix} -26 & 6 & -6 & 2 & -1 & -1 & 1 & 0 \\ -1 & 3 & 5 & 4 & 2 & 0 & -1 & 0 \\ 0 & -2 & -3 & 1 & 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C_{C_r} = \begin{bmatrix} -8 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad C_{C_b} = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

They have 41, 60 and 63 zero elements and further 11, 4 and 1 distinct values respectively. The large values are concentrated in the left/top-most corner, since these represent low frequency DCT element. These are the easiest for the human eye to detect and hence distinguish, while differences in the more high frequency elements are not as noticeable.

3.1.2

In this part we compress the same image, but with two different quality numbers. In Figure 3 we use q = 0.7, which does not cause any noticeable difference to the observer.

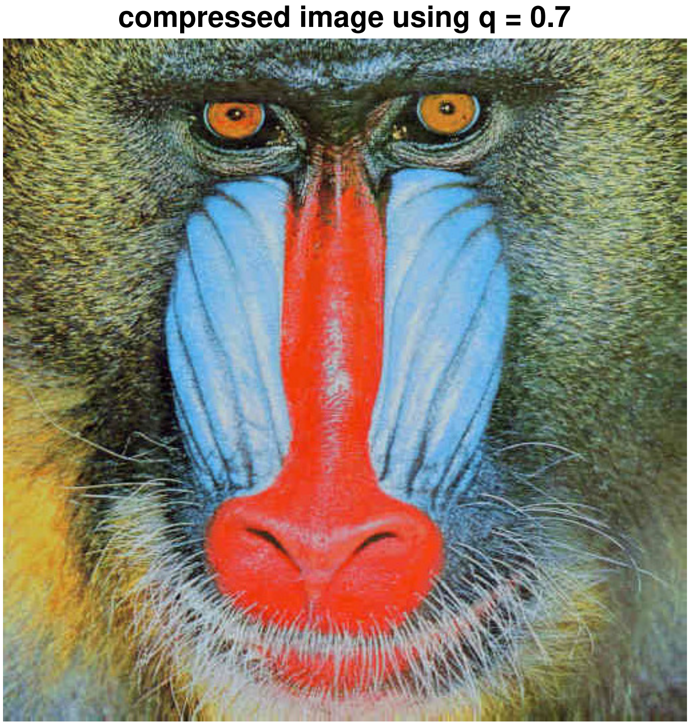

Figure 3: Image compressed using q = 0.7.

Further, in Figure 4 we use q = 0.1. In this case it is clear that the quality of our visible result is not as good as for q = 0.7.

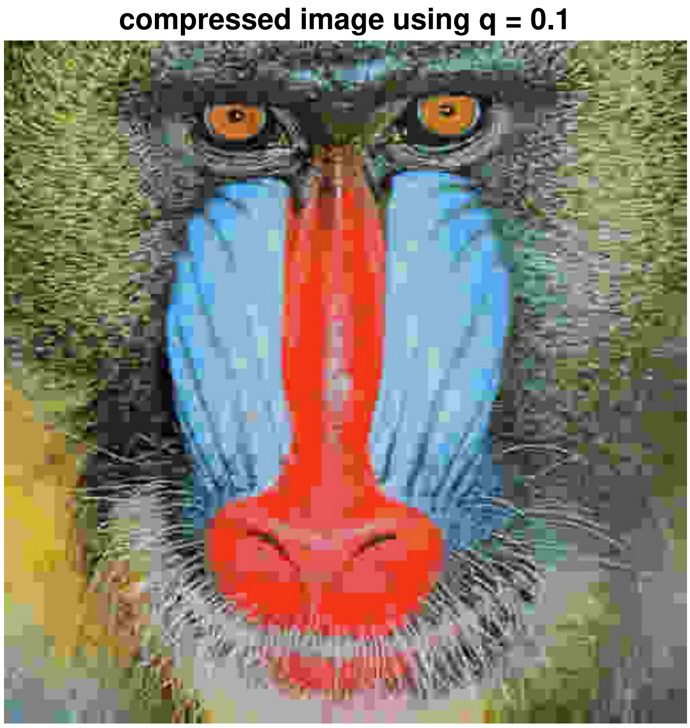

Figure 4: Image compressed using q = 0.1.

Hence, it is clear that a compression with q = 0.7 of our image seems good enough for the observer, while for q = 0.1, it does not deliver a desireable result.

A Matlab Code

A.1 jpegcompress.m

jpegcompress.m

```
function coeff = jpegcompress(im,quality)
%JPEGCOMPRESS - Takes an RGB image and a quality number (0<quality<1) and
%               compresses it, returning the quantized DCT coefficient for
%               further decompression usage.
%
%   MINIMAL WORKING EXAMPLE: Load and compress an image in RGB format and
%                            compress it with quality number equal to 0.8:
%
%   >> im = imread(image_filename); %read image
%   >> q = 0.8; %set quality
%   >> coeff = jpegcompress(im, q); %compress image and get the coeffs

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-24: Initial version .
%
% Function code starts here...

% Get the size of the input image.
[r,c,b]=size(im);

% Verify the input image is of the correct type.
if b~=3 || ~isa(im,'uint8')
    error('Image must be uint8 RGB');
end

% Convert from RGB to YCbCr color space.
YCbCr=rgb2ycbcr(im);

% Preallocate array for coefficients
coeff=zeros(size(YCbCr));

% Get the DCT matrix and the quantization matrices.
% <your code here>
%DCT matrix
D = dctmtx(8);
Q = {};
% Luminance- and Chrominance quantization tables.
[Q{1},Q{2}] = jpegquantmat;
Q{3} = Q{2};

% For each band...
for b=1:3
    % Get the quantization matrix for this band, properly scaled.
    % <your code here>
    Qq = Q{b}/quality;

    % Compute coefficients
    for i=1:8:r
        for j=1:8:c
            % Extract an 8-by-8 tile, convert to floating point and
            % shift to make signed.
```

```
            tile=double(YCbCr(i:i+7,j:j+7,b))-128;

            % Do the DCT transformation and quantization
            % <your code here>
            tile = round((D*tile*D')./Qq);

            % Store the coefficients in the correct place.
            coeff(i:i+7,j:j+7,b)=tile;
        end
    end
end
```

A.2 jpegdecompress.m

jpegdecompress.m

```
function im=jpegdecompress(coeff,quality)
%JPEGDECOMPRESS - Takes a matrix of DCT coefficients for a compressed image
%                 and a quality number (0<quality<1). Returns a
%                 decompressed image from the DCT coefficients.
%
%   MINIMAL WORKING EXAMPLE: Decompress an image from DCT coefficients
%                            using a quality number equal to 0.8:
%
%   >> q = 0.8; %set quality
%   >> imDecomp = jpegdecompress(coeff, q); %decomp image from DCT coeffs

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-24: Initial version .
%
% Function code starts here...

% Get the size of the input image.
[r,c,b]=size(coeff);

% Preallocate output image.
YCbCr=zeros(r,c,b,'uint8');

% Get the DCT matrix and the quantization matrices.
% <your code here>
%DCT matrix
D = dctmtx(8);
Q = {};
% Luminance- and Chrominance quantization tables.
[Q{1},Q{2}] = jpegquantmat;
Q{3} = Q{2};

% For each band...
for b=1:3
    % Get the quantization matrix for this band, properly scaled.
    % <your code here>
    Qq = Q{b}/quality;

    % Restore pixels
    for i=1:8:r
        for j=1:8:c
            % Extract tile of the quantized coefficients.
            tile=coeff(i:i+7,j:j+7,b);

            % De-quantize and do the inverse DCT.
            % <your code here>
            tile = (D'*(tile.*Qq)*D);

            % Shift back to signed, convert to uint8 and store in
            % the correct place.
            tile=tile+128;
            YCbCr(i:i+7,j:j+7,b)=uint8(tile);
        end
    end
end
```

```
% Convert image back to rgb
im=ycbcr2rgb(YCbCr);
```

A.3 testJpeg.m

testJpeg.m

```
%TESTJPEG - Script for running relevant tests on our jpegcompress and
%           jpegdecompress functions.
%
%   MINIMAL WORKING EXAMPLE:
%
%   >> testJpeg.m

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-24: Initial version .
%
% Function code starts here...

clear all; clc;
%read the image
im = imread('mandrill.png');
%plot image for reference to future compressed/decompressed image
figure
imshow(im)
title(['Initial image - no compression done'])

%set quality (q = 1 initially)
q = 1;

%compress image and get the quantized DCT-coefficients
coeff = jpegcompress(im,q);
%use the DCT-coefficients and decompress the compressed image
imMod = jpegdecompress(coeff,q);

%plot the compressed/decompressed image in new window
figure
imshow(imMod)
title(['compressed and decompressed image, q = 1'])

%pre-allocate reference variables
tileExample = cell(3,1);
zeroEl = zeros(3,1);
uniEl = zeros(3,1);

%loop over all bands for only one tile
for b = 1:3
    %Look at the first tile
    tileExample{b} = coeff(1:8,1:8,b);
    %Number of zero elements
    zeroEl(b) = length(find(tileExample{b} == 0));
    %Number of distinct non-zero elements
    uniEl(b) = length(unique(tileExample{b}))-1;
end

%% Compression/Decompression using different qualities
%set vector of qualities
qSweep = [0.7, 0.1];
%pre-allocate cell array for storing images
imSweep = cell(length(qSweep),1);
%pre-allocate cell array for storing coefficients
coeffSweep = cell(length(qSweep),1);

%loop over all q's
```

```
for i = 1:length(qSweep)
    %compress image
    coeffSweep{i} = jpegcompress(im,qSweep(i));
    %decompress image
    imSweep{i} = jpegdecompress(coeffSweep{i},qSweep(i));
    %plot image in new window
    figure
    imshow(imSweep{i})
    title(['compressed image using q = ', num2str(qSweep(i))]);
end
```