SIGNAL COMPRESSION

1. Lossless
   (a) invertible
   (b) compression ratio: 2 - 3
   (c) Huffman coding
   (d) Ziv-Lempel algorithm
2. Lossy
   (a) non-invertible
   (b) compression ration: 5 - 15
   (c) JPEG (Image compression)
   (d) LPC (Speech compression)
   (e) MP3 (Audio compression)

TRANSFORM CODING

1. Transform

$$y = F\,x$$

2. Quantize

$$y_q = Q[y]$$

1. Frequently, F is an orthonormal matrix.
2. F should decorrelate the elements of x.
3. The elements of y will have different amplitudes.
4. F is chosen so that many elements of y will have small amplitudes.
5. Usually an efficiently implemented F is chosen. (So that $y = F\,x$ can be performed with less than N^2 operations.)

TRANSFORM DECODING

1. Inverse Transform

$$r = F^{-1}\,y_q$$

If F is orthonormal, then

$$\begin{aligned}\sum_k y(k)^2 &= y^t\, y \\ &= (F\,x)^t\,(F\,x) \\ &= x^t\,F^t\,F\,x \\ &= x^t\,I\,x \\ &= x^t\,x \\ &= \sum_n x(n)^2\end{aligned}$$

So the energy (sum of squares) is the same in the signal domain as in the transform domain. (This is a general form of Parseval's theorem.)

Also:

$$\begin{aligned}\sum_k (y_q(n) - y(n))^2 &= (y_q - y)^t\,(y_q - y) \\ &= (F\,r - F\,x)^t\,(F\,r - F\,x) \\ &= (r - x)^t\,F^t\,F\,(r - x) \\ &= (r - x)^t\,(r - x) \\ &= \sum_n (r(n) - x(n))^2\end{aligned}$$

So the distortion can be computed in either the signal domain or in the transform domain.

The distortion is defined here as

$$E = \sum_n (r(n) - x(n))^2.$$

TRANSFORM CODING

Different transforms can be used.

1. DCT (Discrete Cosine Transform) - JPEG

2. DWT (Discrete Wavelet Transform) - JPEG 2000

3. Filter banks - MP3 Audio

DISCRETE COSINE TRANSFORM (DCT)

1. The DCT is like a 'real-valued' version of the DFT.
2. There are 4 types of DCT. (DCT-I, DCT-II, DCT-III, DCT-IV)
3. Each DCT is an orthonormal transform. $(F^t F = I.)$
4. The DCT-II is the most commonly used type.
5. The element $y(k)$ is an inner product of x with the kth row of F.
$$y(k) = \sum_{n=0}^{N-1} f(k,n)\, x(n)$$
6. The DCT gives a real-valued frequency decomposition of a real-signal $x(n)$.
7. The DCT can be derived from the DFT by symmetric extension.
8. Different types of symmetric extension give rise to the four types of DCT.
9. The DCT can be computed using the DFT (or FFT).

The DCT-II is given by

$$y(k) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} b(k)\, x(n)\, \cos\left(\frac{\pi k}{N}(n+0.5)\right)$$

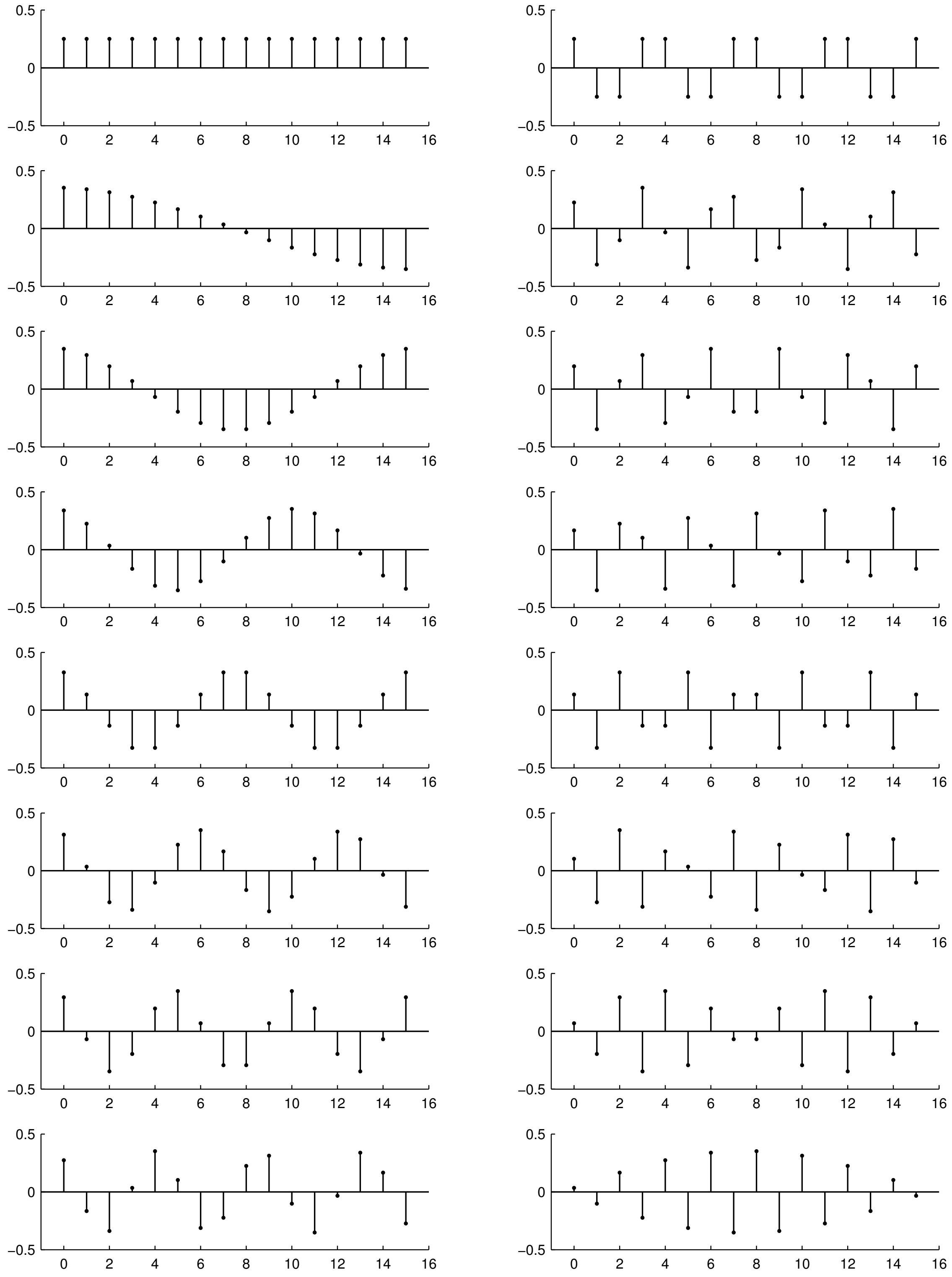
$$x(n) = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} b(k)\, y(k)\, \cos\left(\frac{\pi k}{N}(n+0.5)\right)$$

where

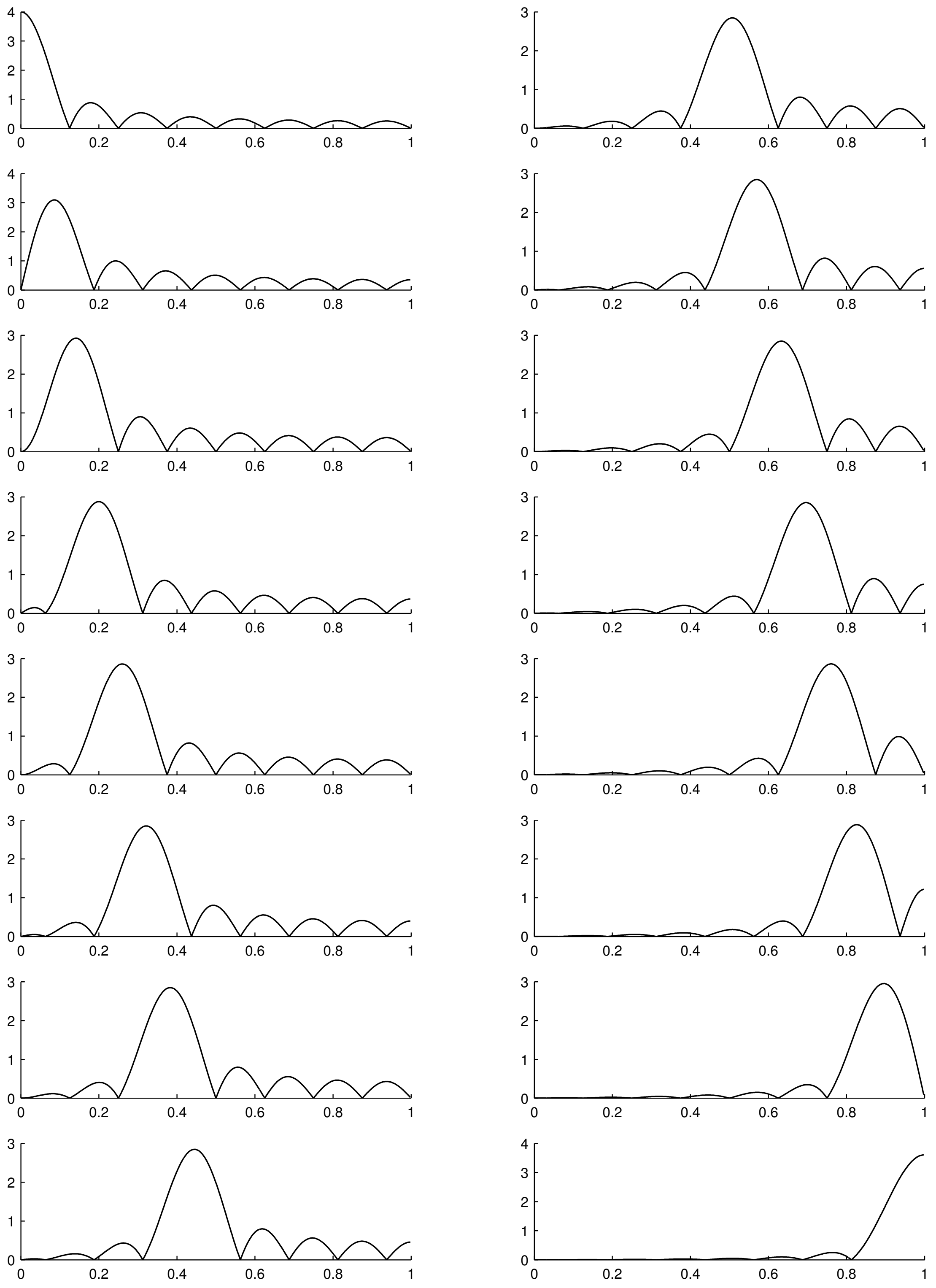
$$b(0) = 1/\sqrt{2}, \qquad b(k) = 1,\ 1 \le k \le N-1$$

The signal $x(n)$ is given by a sum of cosine functions, so the DCT is a discrete-time version of the Fourier series using only cosine.

The basis functions $f(k,n)$ of the 16-point DCT-II

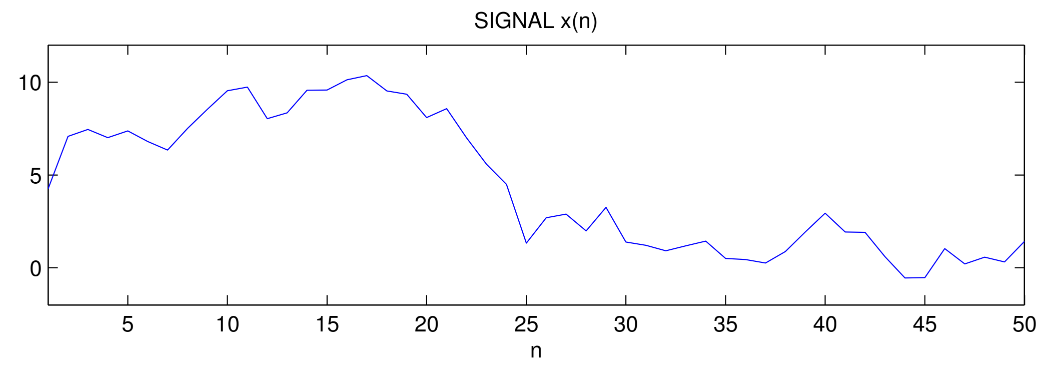


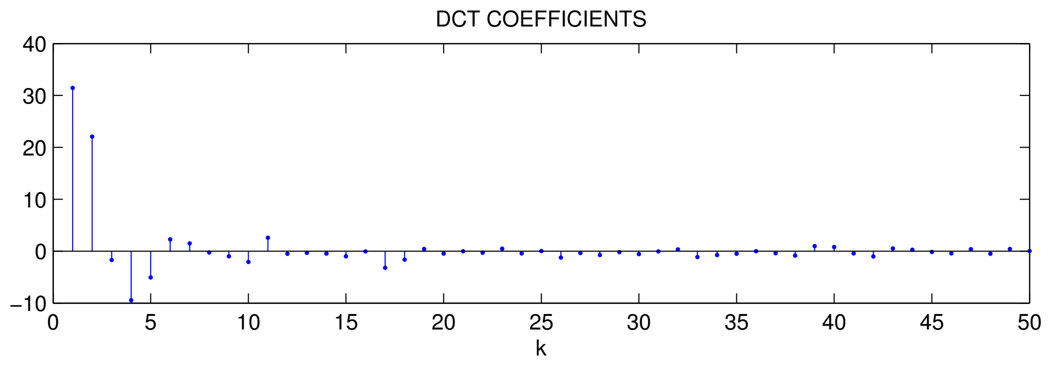
The spectrum of each of the 16-point DCT-II basis functions

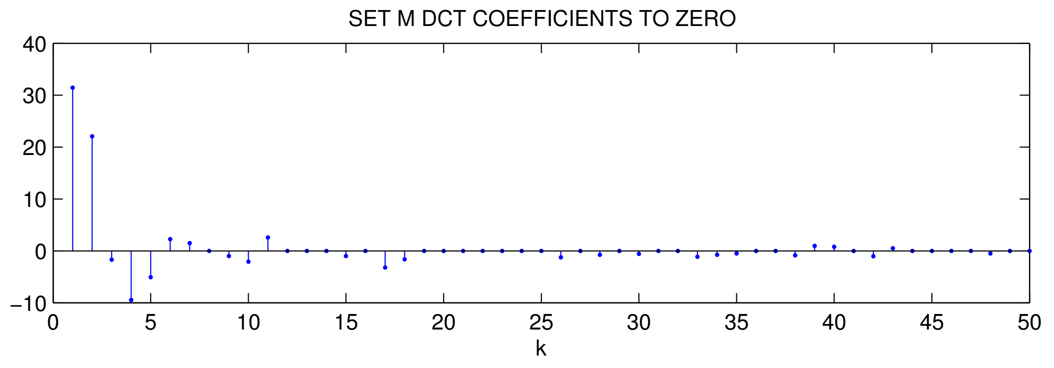


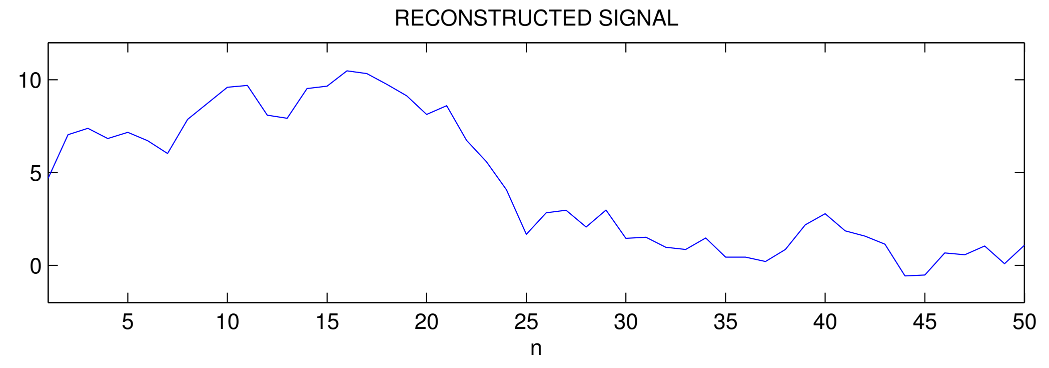
DISCRETE COSINE TRANSFORM (DCT)

As a simple illustration of Transform Coding with the DCT we can use a very simple quantization scheme - set the smallest M coefficients to zero. Here we set 25 (out of 50) coefficients to zero.







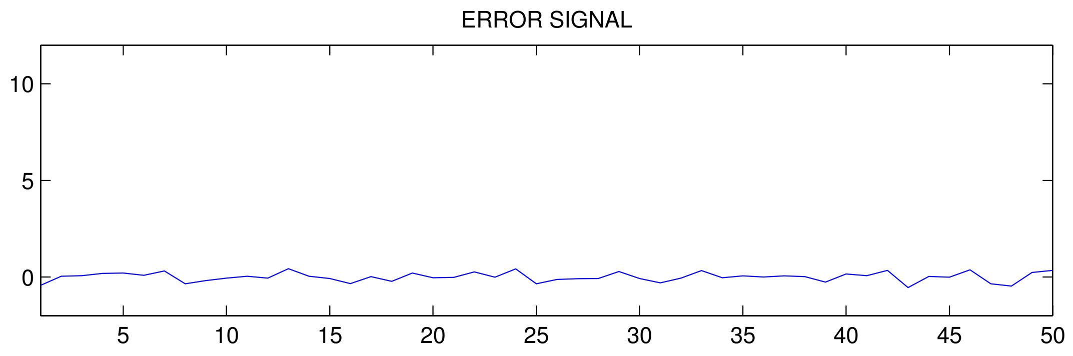


DISCRETE COSINE TRANSFORM (DCT)

To set the smallest M DCT coefficients to zero we can use the following Matlab code.

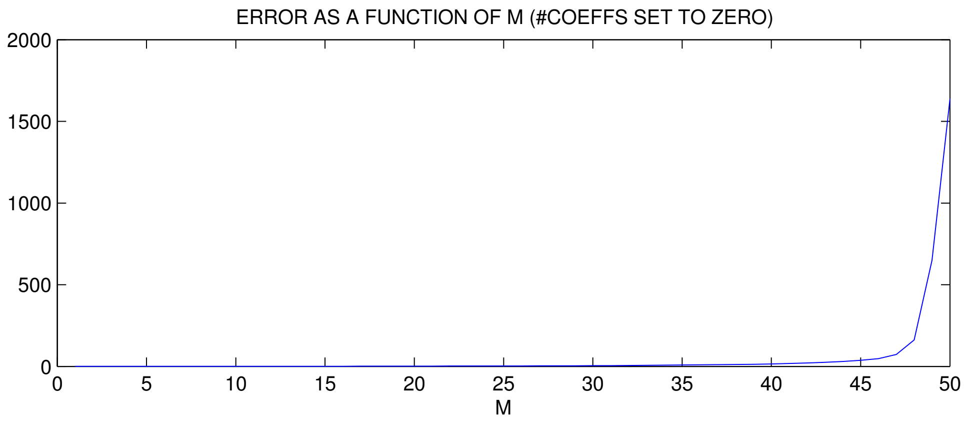
```
% Compute DCT of Signal
y = dct(x);
% Find indices of smallest coefficients
[y_sorted, k] = sort(abs(y));
% Set M smallest coefficients to zero
M = 25; yq = y; yq(k(1:M)) = 0;
% Compute Inverse DCT
r = idct(yq);
```

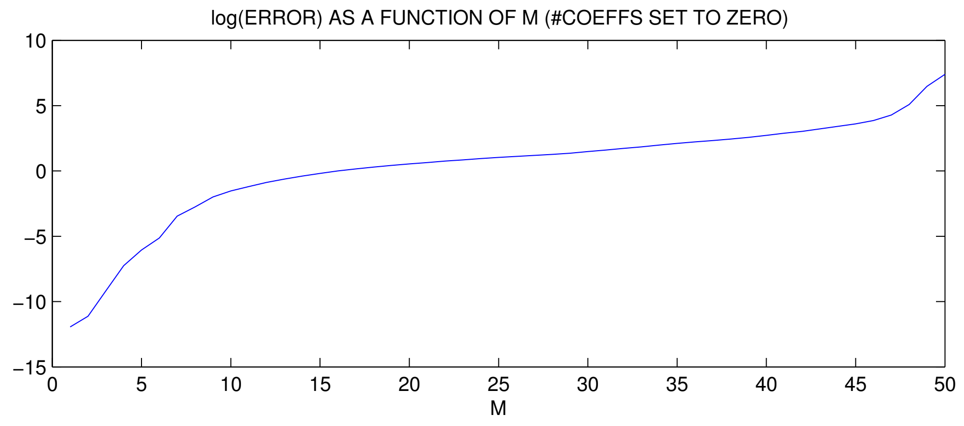
The error between the original signal and the reconstructed signal is very small:



From the plot of the DCT coefficients we could predict that the error is small — because the smallest 25 coefficients are very small in magnitude compared with the remaining 25 coefficients.

We can make a plot of the distortion (square error) as a function of the number of coefficients set to zero (M).





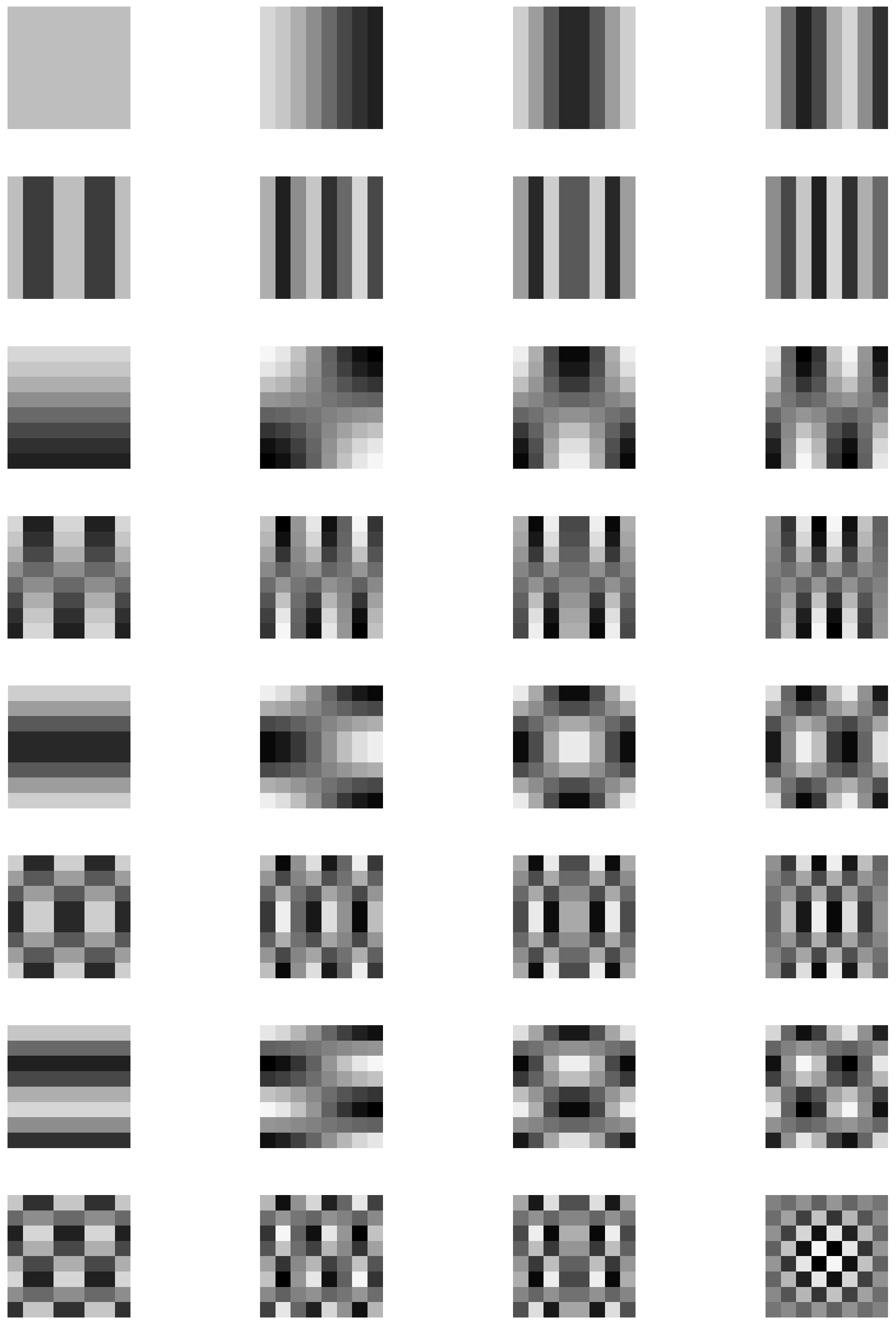
TWO-DIMENSIONAL DCT

1. For image compression (as in JPEG) we need to take the DCT of a 2D array.
2. JPEG stands for *Joint Photographic Experts Group*.
3. The 2D DCT of a 2D array is performed by computing the DCT of each row of the array, followed by computing the DCT of each column of the resulting 2D array.
4. The 2D DCT represents a 2D array as a sum of 2D cosine functions. We can view these 2D basis functions as we viewed the 1D basis functions of the 1D DCT.

The 2D DCT can be implemented in Matlab using the commands

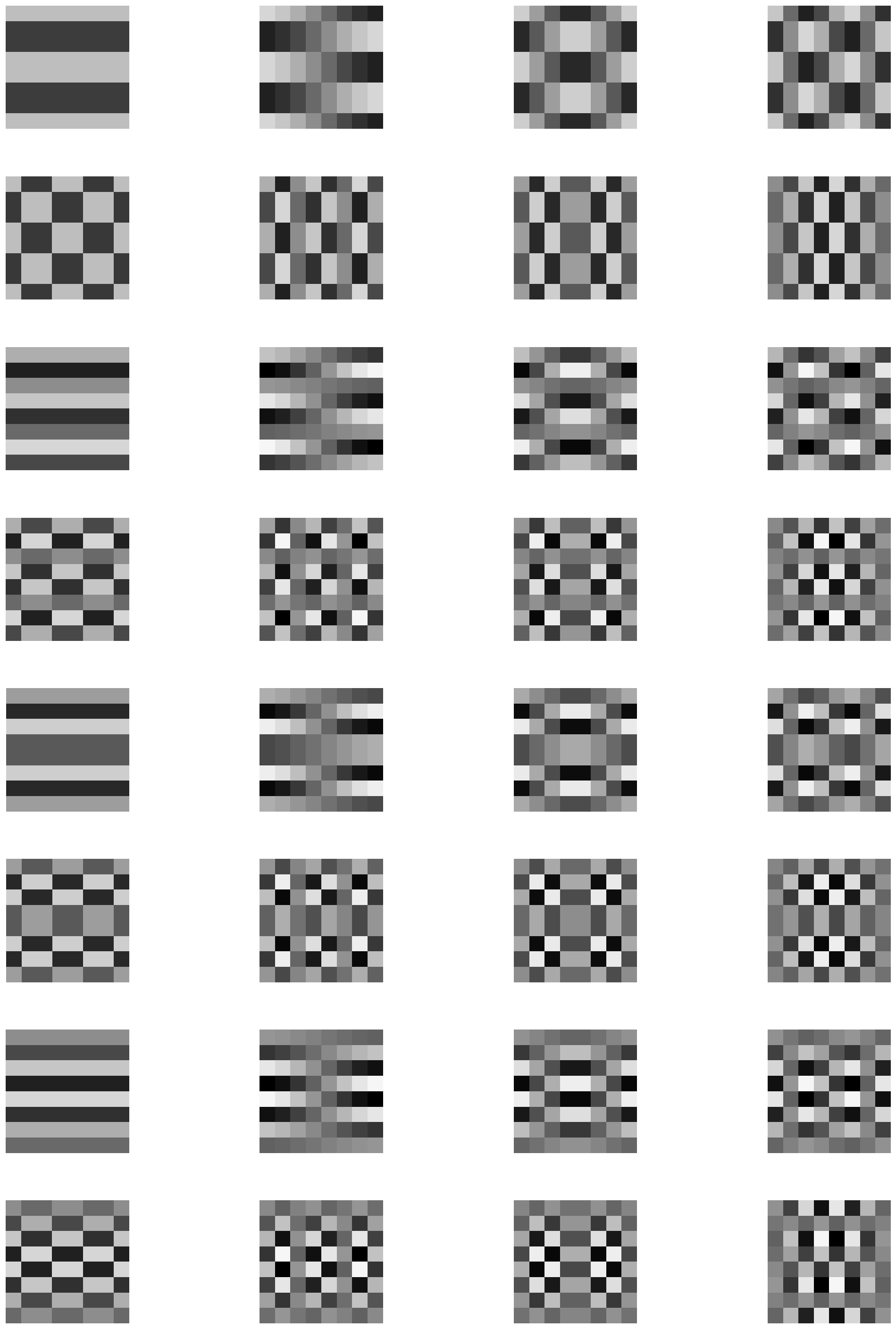
```
% x is a 2D array
>> x = rand(16);

% Compute the 2D DCT of x
>> y = dct(dct(x).').';
% dct(x) computes the DCT of each column of x
% dct(x.').' compute the DCT of each row of x

% Compute the 2D Inverse DCT of y
>> r = idct(idct(y).').';

% Verify that r == x
>> e = r - x; max(abs(e(:)))

      5.5511e-016
```

The first 32 of the 64 basis functions of the 8 by 8 point 2D DCT.



TWO-DIMENSIONAL DCT

The second 32 of the 64 basis functions of the 8 by 8 point 2D DCT.



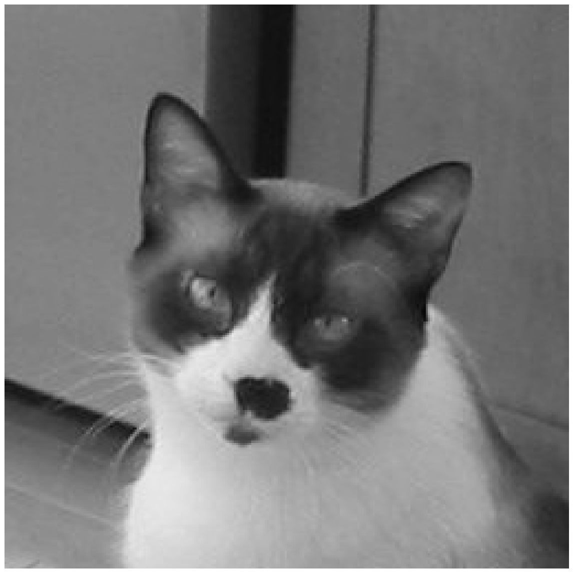
TWO-DIMENSIONAL DCT

1. In practice (eg: JPEG), an image is segmented into blocks (8 by 8 pixels, or 16 by 16 pixels) and the 2D DCT is applied to each block.

2. To illustrate the effect of DCT-based compression, lets set the smallest 48 DCT coefficients of each 8 by 8 block to zero.

ORIGINAL IMAGE

SETTING 3/4 THE DCT COEFFS TO ZERO



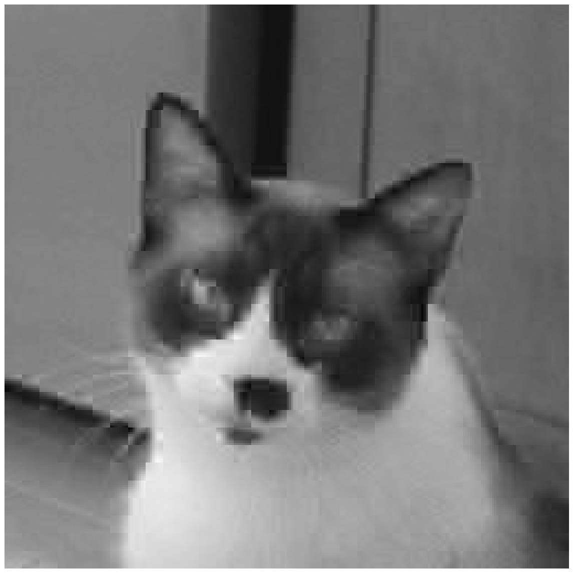
There is almost no noticeable difference!

TWO-DIMENSIONAL DCT

If we set 60 DCT coefficients to zero out of every 64-coefficient block, we get the following result.

ORIGINAL IMAGE

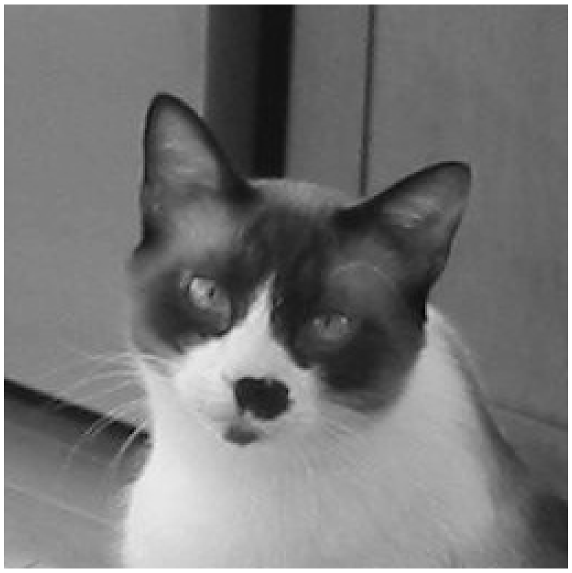
SETTING 60/64 OF THE DCT COEFFS TO ZERO



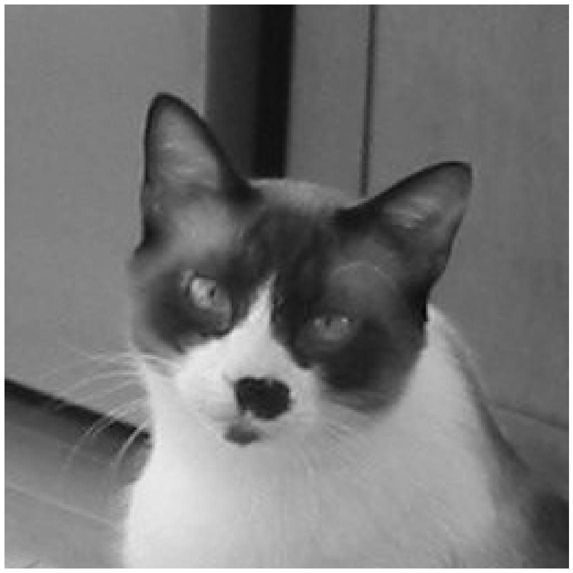
This is a compression ration of $60/64 = 93\%$.

TWO-DIMENSIONAL DCT

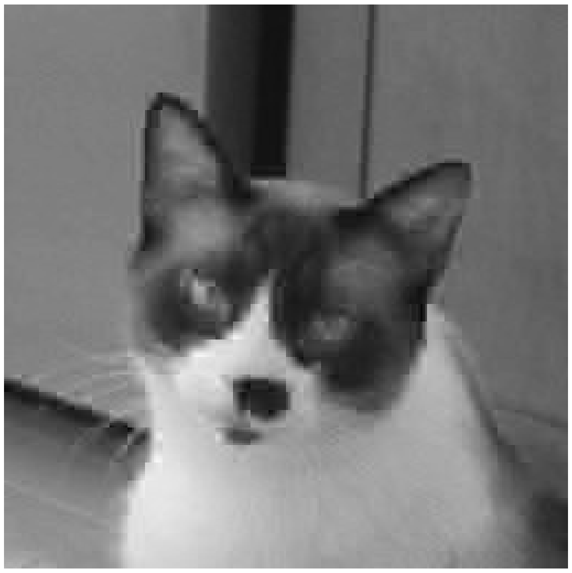
ORIGINAL IMAGE



SETTING 3/4 THE DCT COEFFS TO ZERO



SETTING 60/64 OF THE DCT COEFFS TO ZERO



SETTING 63/64 OF THE DCT COEFFS TO ZERO