ECE 6397 Matlab Project

Alexander Hebert

May 2, 2014

1)

a)

The file signal.dat was downloaded and loaded into Matlab.
The sampling frequency $f_s = 1023\ Hz$, and the corresponding period is $T_s = 1/f_s = 977.517\ \mu s$. The signal h(t) is plotted vs. time in Figure 1.

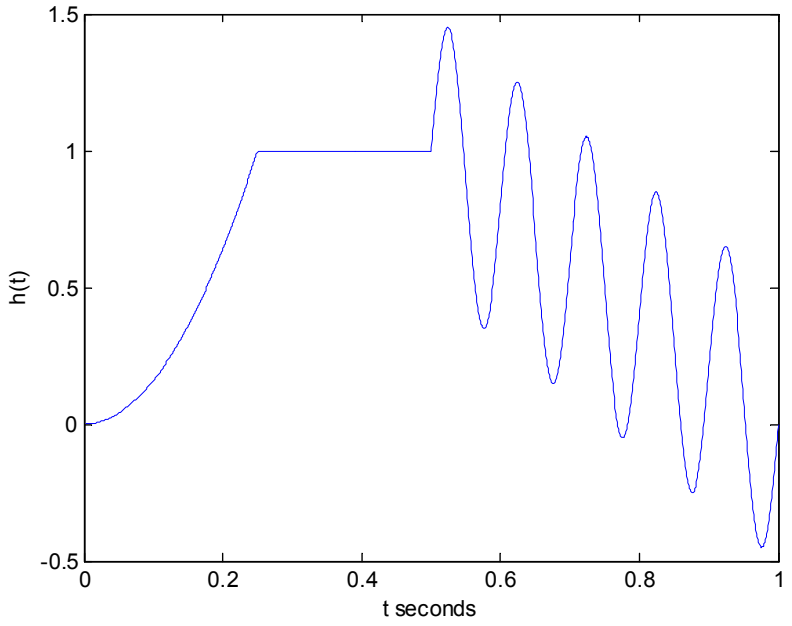

Figure 1. Signal h(t) vs. time for $0 \leq t \leq 1\ s$.

b)

The Haar wavelet system was implemented using Matlab functions: *haar.m, inner_product_f.m, phi.m,* and *psi.m*.

c)

Using the Matlab functions from part (b) and signal1.dat, the projections $\hat{}_j(t)$ of *h(t)* into W_j for j = 0,1,…,6 are shown in Figures 2 and 3.

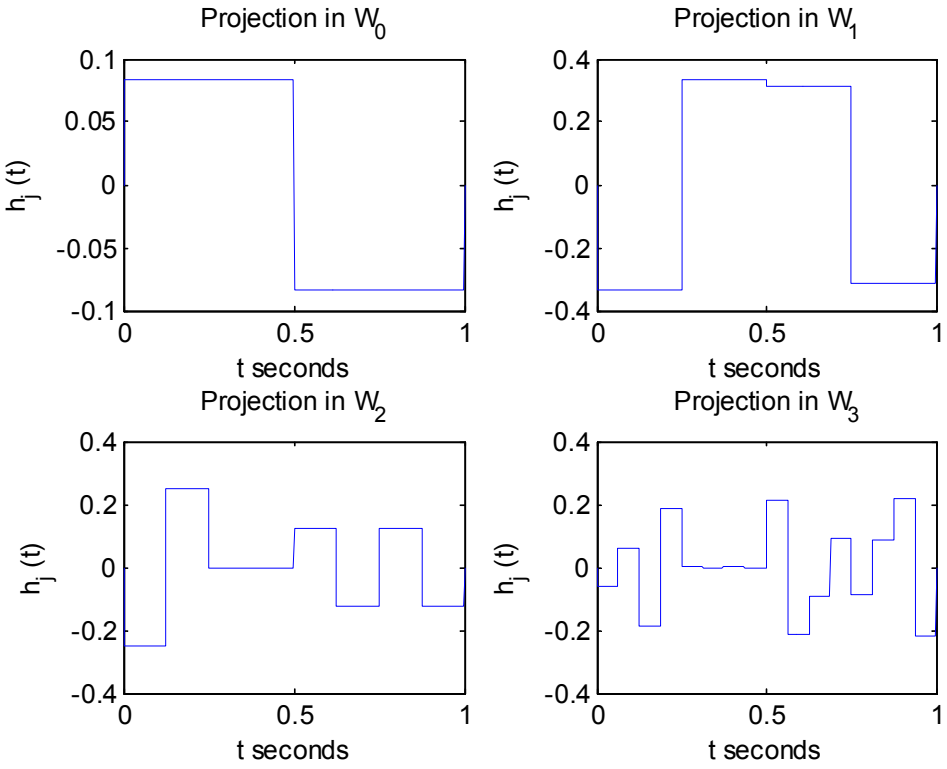

Figure 2. Projection $\hat{}_j(t)$ of *h(t)* into W_j for j = 0,1,2,3.

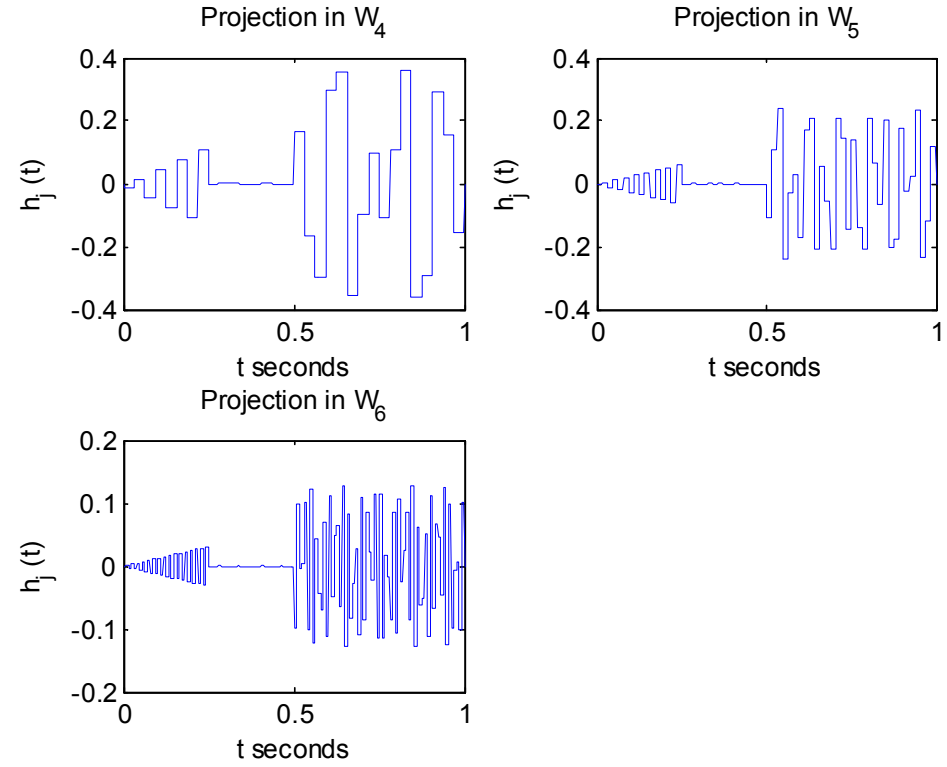

Figure 3. Projection $\hat{}_j(t)$ of *h(t)* into W_j for j = 4,5,6.

The projections $\tilde{}_j(t)$ of *h(t)* into V_j for j = 0,1,…,7 are shown in Figures 4 and 5. The projections of *h(t)* in V_j for j = 0,1,…,7 are computed from the following equations.

$$V_1 = V_0 \quad W_0$$

…

$$V_J = V_0 \quad W_0 \quad W_1 \quad \ldots \quad W_{J-1}$$

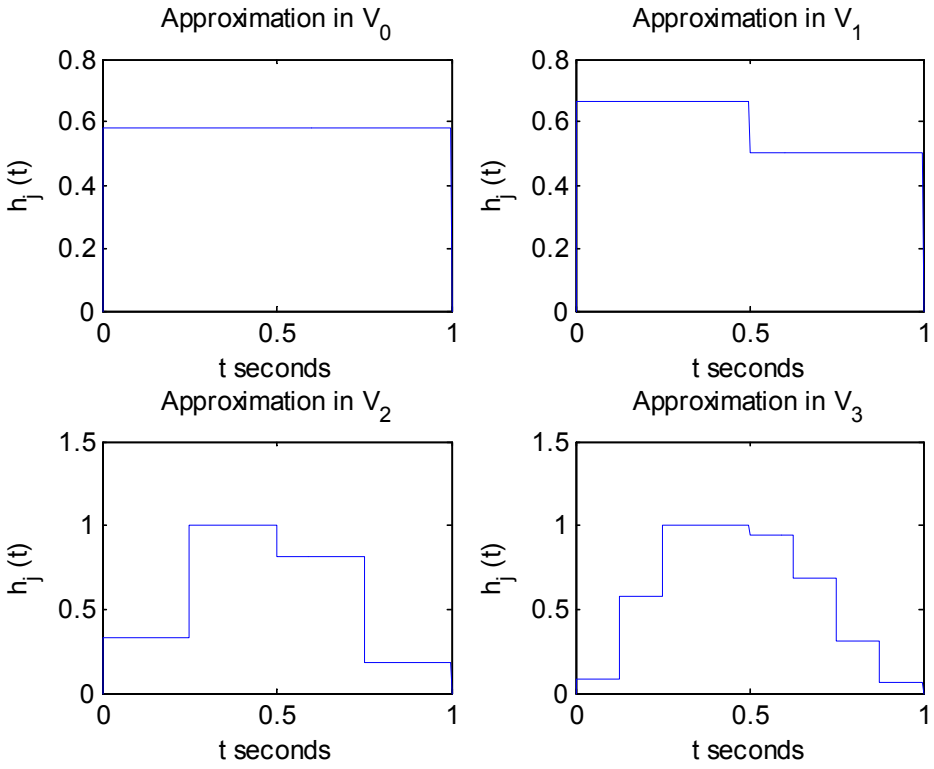

Figure 4. Projection $\tilde{}_j(t)$ of *h(t)* into V_j for j = 0,1,2,3.

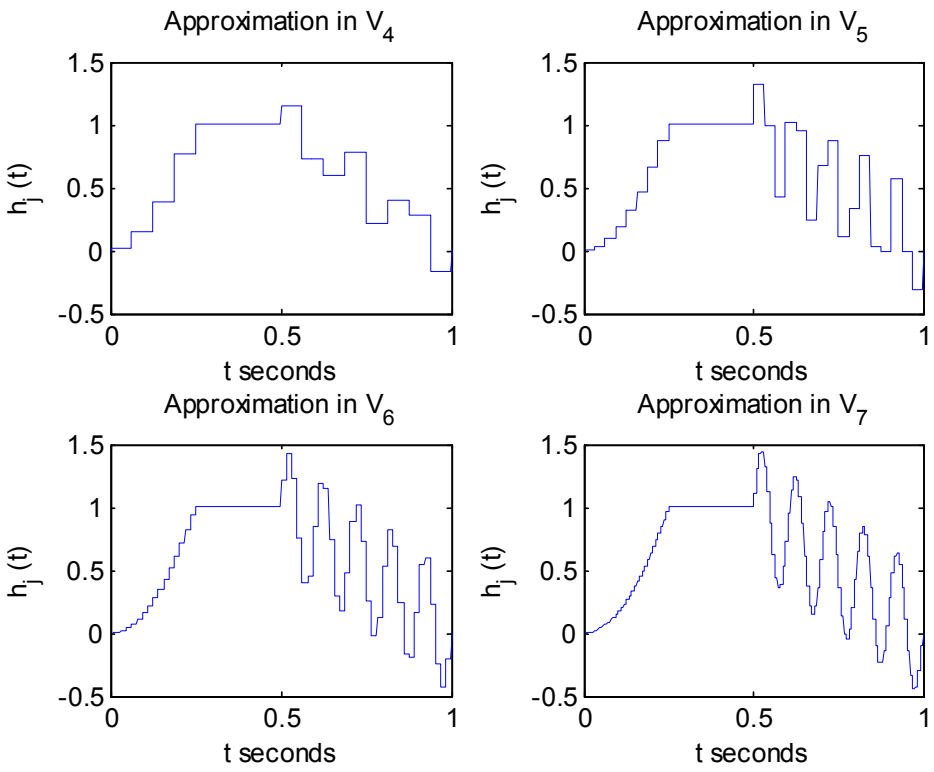

Figure 5. Projection $\tilde{}_j(t)$ of *h(t)* into V_j for j = 4,5,6,7.

In the Figure 5, the approximation in V_7 looks very similar to the original signal *h(t)*.

d)

The files signal2.dat and filter1.dat were downloaded and loaded into Matlab. Signal2 and filter1 are plotted in Figures 6 and 7 respectively.

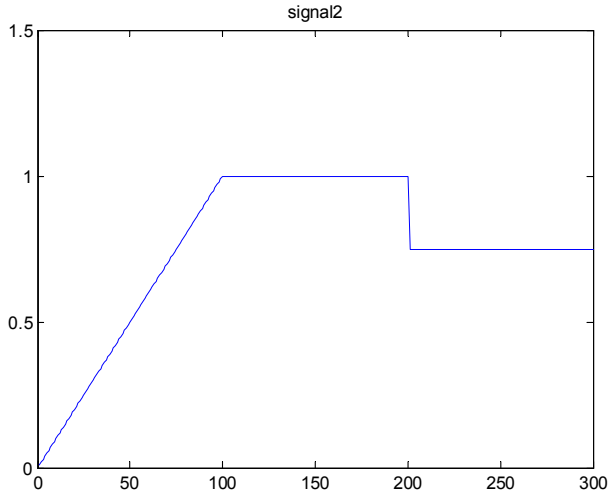

Figure 6. Plot of signal2.dat.

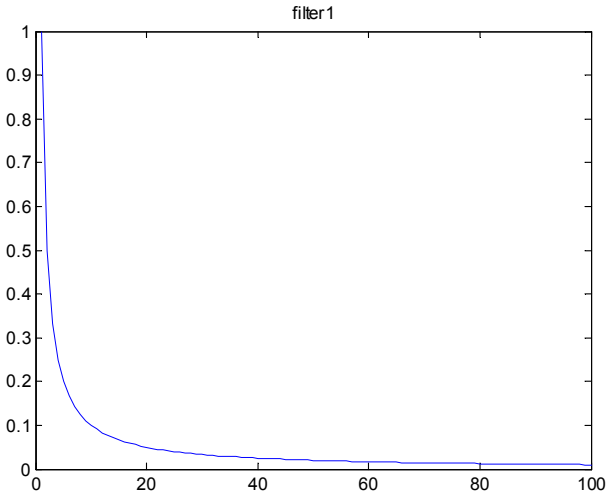

Figure 7. Plot of filter1.dat.

e)

Circular convolution was implemented in the Matlab function *circular_conv.m*. It takes two sequences as arguments. If one sequence is shorter than the other, it is padded with zeros so that they are equal in length.

f)

The linear convolution of signal2.dat and filter1.dat is plotted in Figure 8.

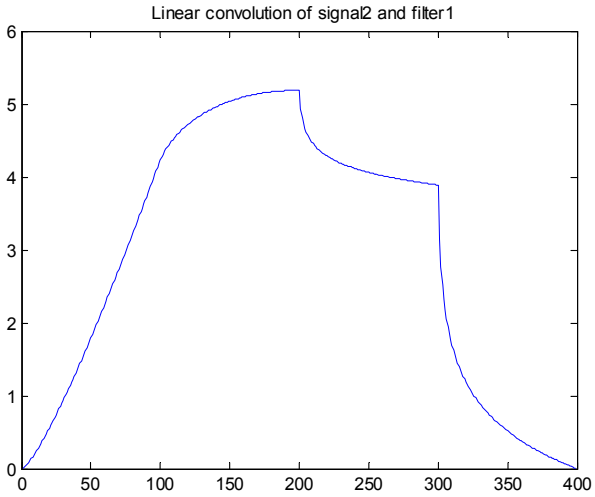

Figure 8. Plot of linear convolution of signal2 and filter1.

The circular convolution of signal2.dat and filter1.dat is plotted in Figure 9.

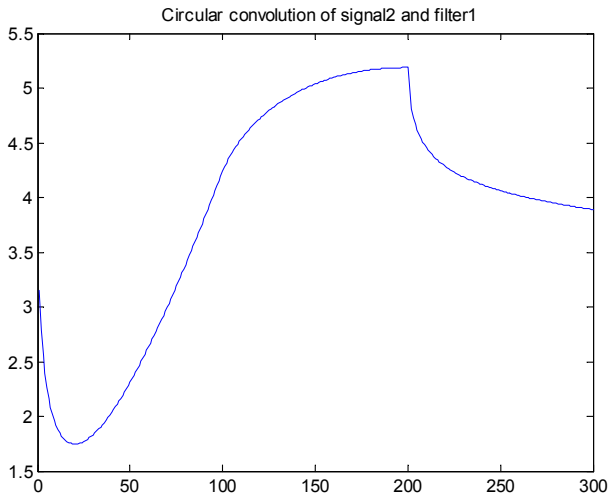

Figure 9. Plot of circular convolution of signal2 and filter1.

We observe that the linear and circular convolutions have similar range in values (vertical axis) but different shaped curves. For the linear case, the length of the output increased to about 400. For the circular case, the length of the output remained equal to signal2's length of 300.

g)

The Haar wavelet system was implemented again using circular convolution in *haar2.m*, which uses the discrete wavelet transform (DWT) function *dwtcc.m* (dwt.m is a built-in Matlab function). Given that the interval of support for *x(t)* is T = 1 and the number of samples is N, the final scale for signal1.dat is

$$J_1 = \log_2 \frac{N}{T} = \log_2 \frac{1024}{1} = 10$$

Using the Matlab functions above and signal1.dat, the projections $\hat{}_j(t)$ of *h(t)* into W_j for j = 0,1,…, J_1 – 1 are shown in Figures 10, 11, and 12.

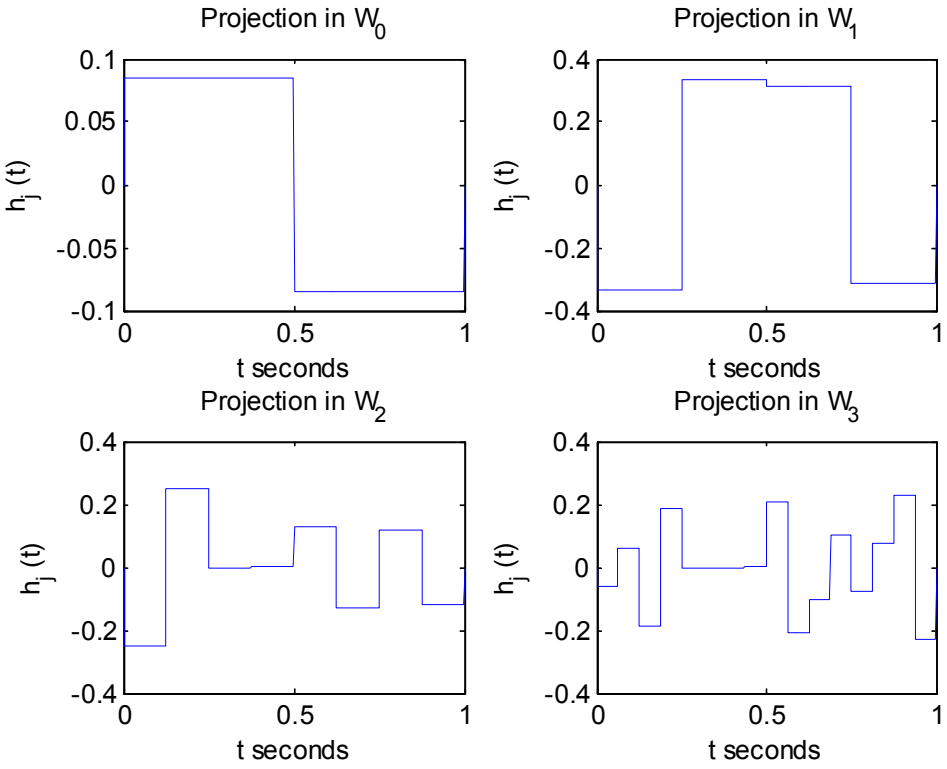

Figure 10. Projection $\hat{}_j(t)$ of *h(t)* into W_j for j = 0,1,2,3.

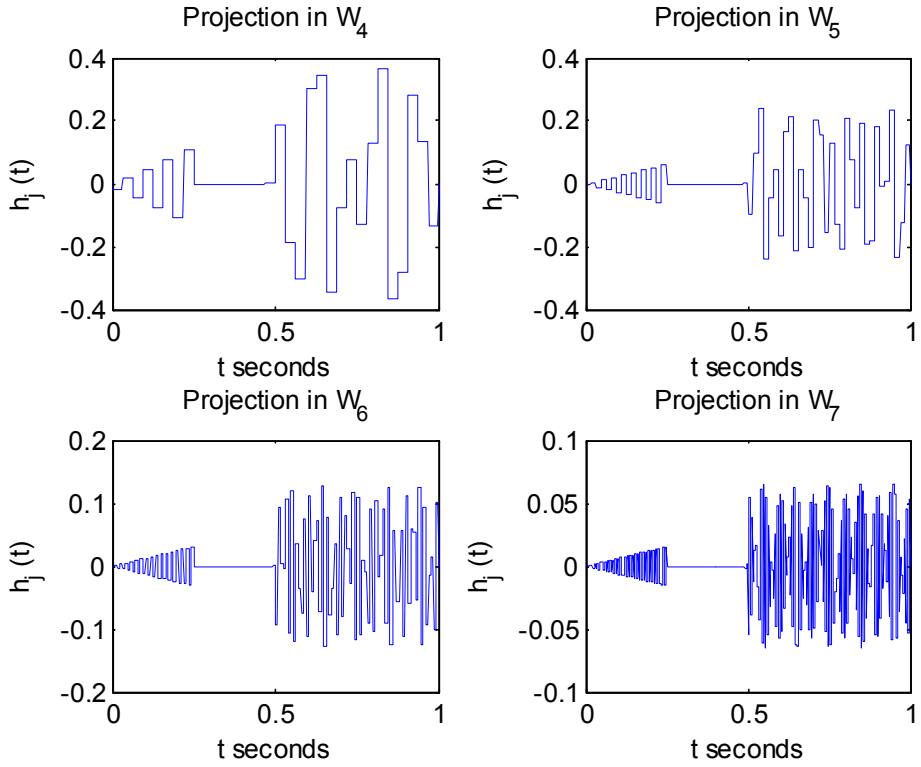

Figure 11. Projection $\hat{}_j(t)$ of *h(t)* into W_j for j = 4,5,6,7.

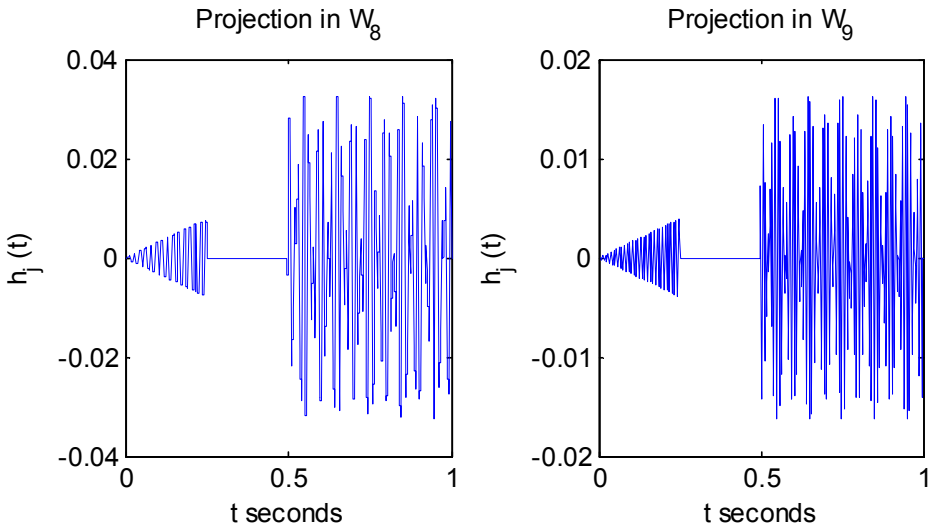

Figure 12. Projection $\hat{}_j(t)$ of *h(t)* into W_j for j = 8,9.

The projections $\tilde{}_j(t)$ of *h(t)* into V_j for j = 0,1,…, $J_1 - 1$ are shown in Figures 13, 14, and 15.

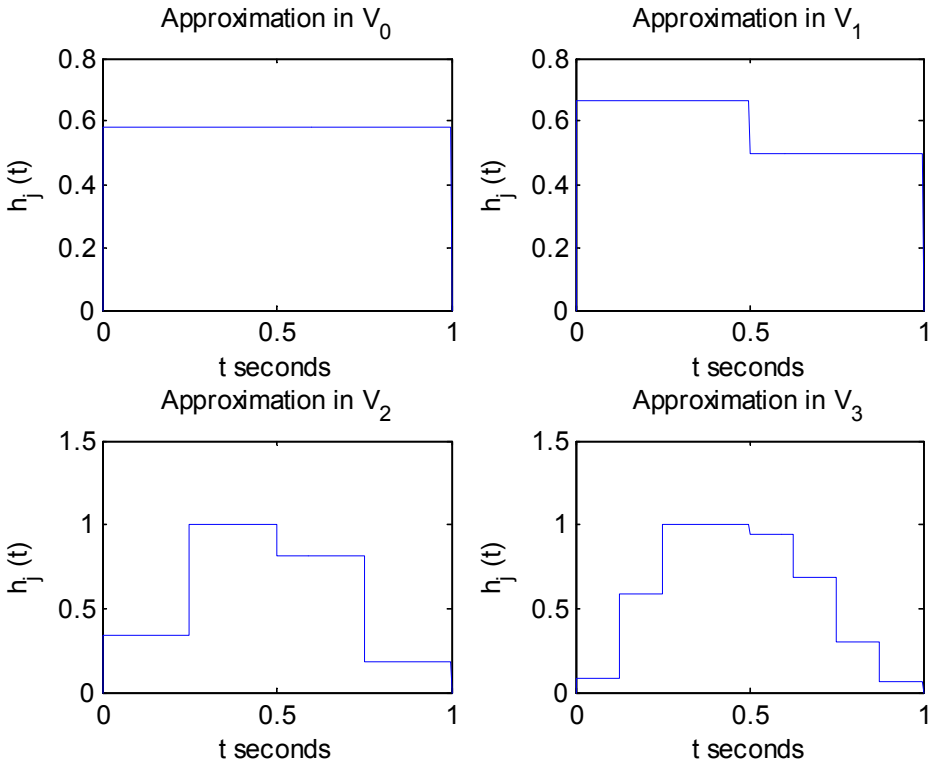

Figure 13. Projection $\tilde{}_j(t)$ of *h(t)* into V_j for j = 0,1,2,3.

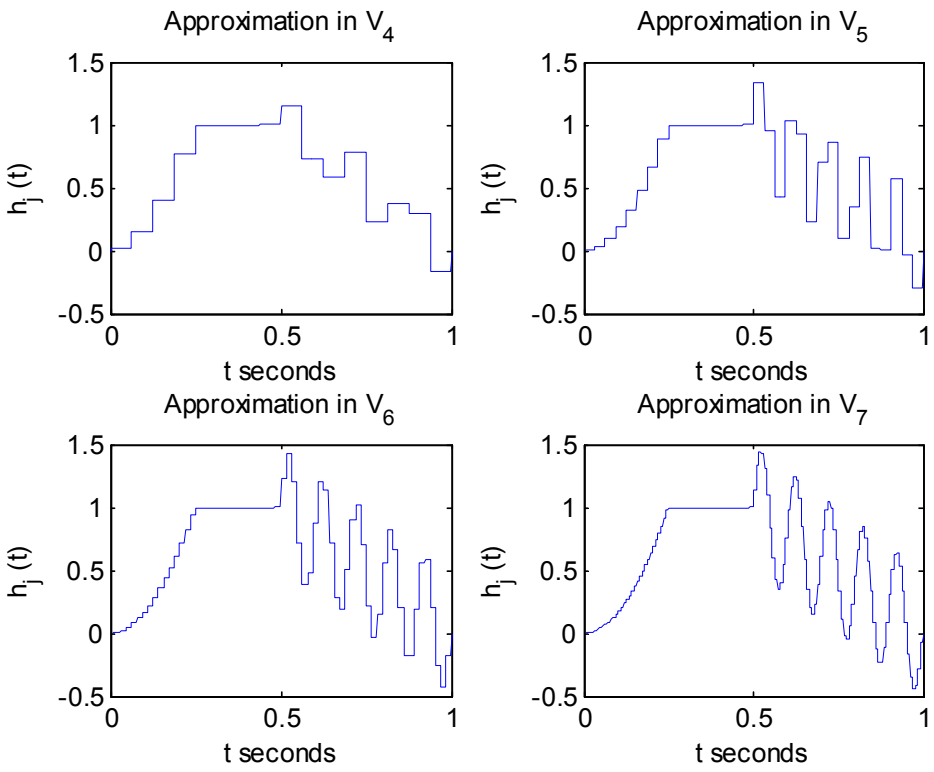

Figure 14. Projection $\tilde{}_j(t)$ of *h(t)* into V_j for j = 4,5,6,7.

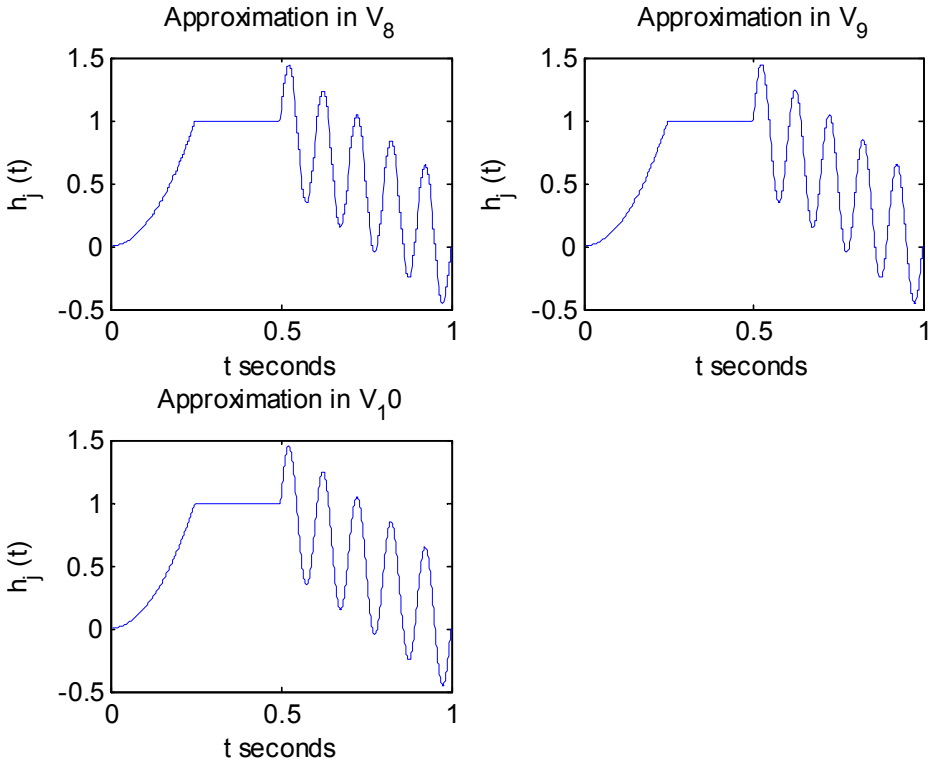

Figure 15. Projection $\tilde{}_j(t)$ of *h(t)* into V_j for j = 8,9,10.

2)

a)

A Matlab function, *encoder.m*, was coded to perform uniform scalar quantization after applying the DWT to an input signal *x(t)*. The equation for quantization is below.

$$z = Q[y] = round\left(\frac{DWT[x(t)]}{q}\right)$$

b)

A Matlab function, *decoder.m*, was coded to perform inverse quantization on the quantized DWT[x(t)] and then apply the inverse DWT to synthesize the signal *x(t)*. The equation for inverse quantization is below.

$$\hat{y} = Q^{-1}[z] = qz$$

c)

For entropy coding, the entropy of the coefficient sequence z[k] will require an average of *H* bits for each index. The equation is below.

$$H = \sum_{i=1}^{K} p_i \log_2(p_i)$$

A Matlab function *entropy.m* was coded to compute the entropy of the quantized matrix of DWT coefficients *z[k]*. The entropy was calculated for the entire matrix excluding extra zero values from different row lengths. The value of p_i was determined by the frequency of occurrence (from Matlab *hist* function) of the index value divided by the total number of elements in *z[k]*. The value of K bins was calculated as follows:

$$K = 2 \times \frac{1}{q} + 1$$

For $q = 0.001, 0.002, 0.005, 0.01, 0.02, 0.05$, the original signal *x(t)* was encoded and quantized using *encoder.m*. Then the encoded signal was decoded and is referred to as $\hat{x}(t)$. The decoded signal is plotted for q values in Figure 16. We observe that the decoded signal was more accurate compared to the original signal with smaller step sizes of q .

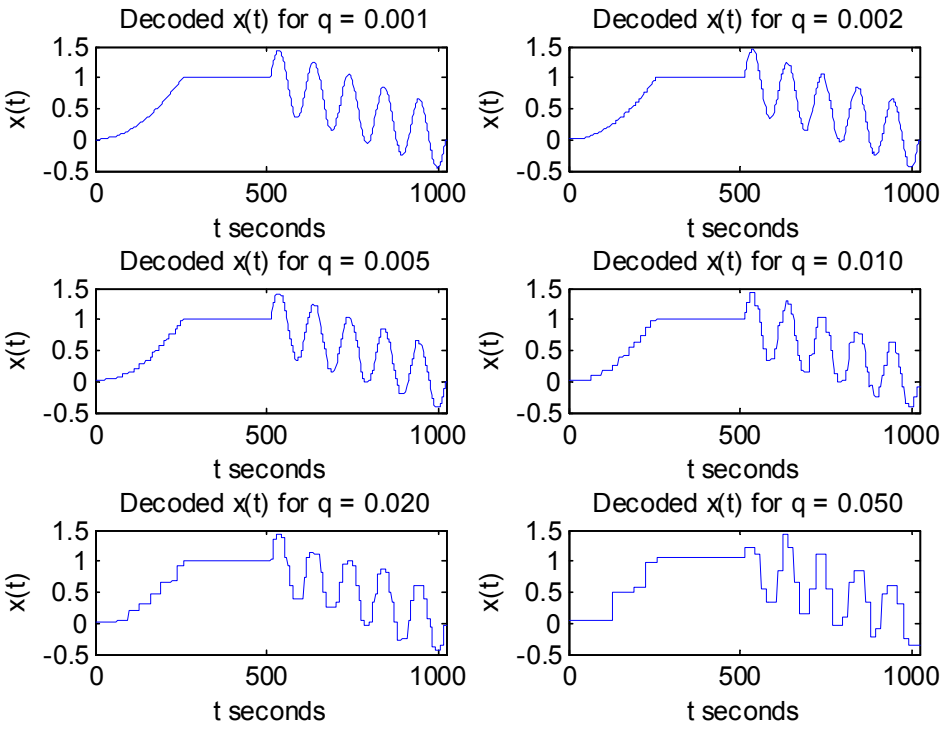

Figure 16. Decoded signal $\hat{x}(t)$ for six values of q.

The entropy of quantized matrix of DWT coefficients *z[k]* is plotted vs. q value in Figure 17. We observe that entropy increased as q decreased in value.

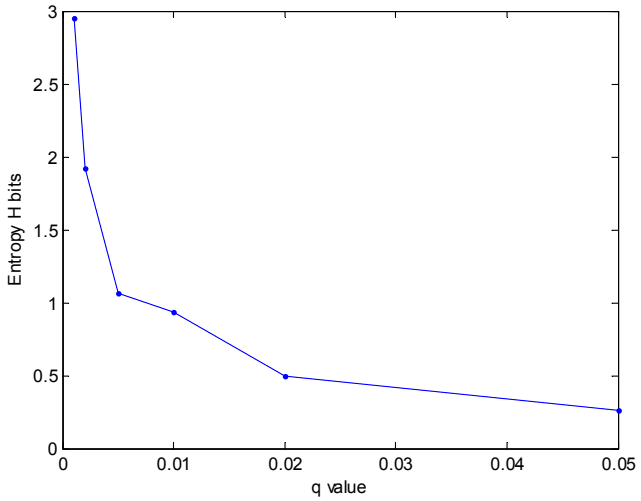

Figure 17. Entropy of quantized matrix of DWT coefficients *z[k]* is plotted vs. q values.

The mean-squared-error (MSE) between *x(t)* and $\hat{x}(t)$ was calculated with the following equation:

$$MSE = \frac{1}{N}\sum_{k=0}^{N-1}(x[k] \quad \hat{x}[k])^2$$

The MSE is plotted vs. q values in Figure 18. We observe that MSE increased as q increased in value.

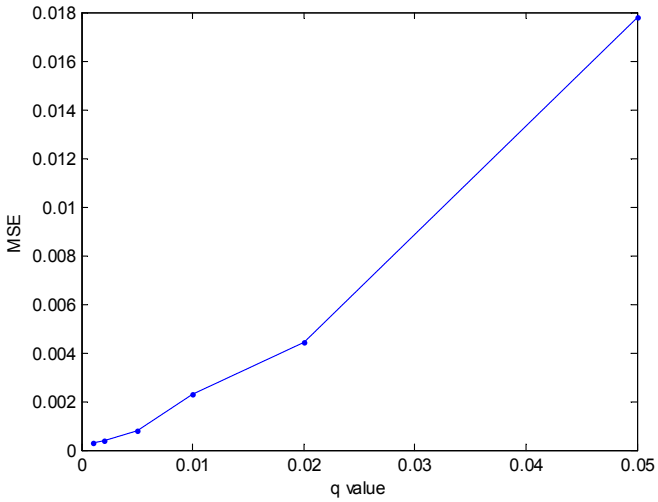

Figure 18. MSE vs. *q* values.

The entropy is plotted vs. MSE for the six values of *q* in Figure 19. Lower values of *q* resulted in lower MSE but at the cost of higher entropy in bits.

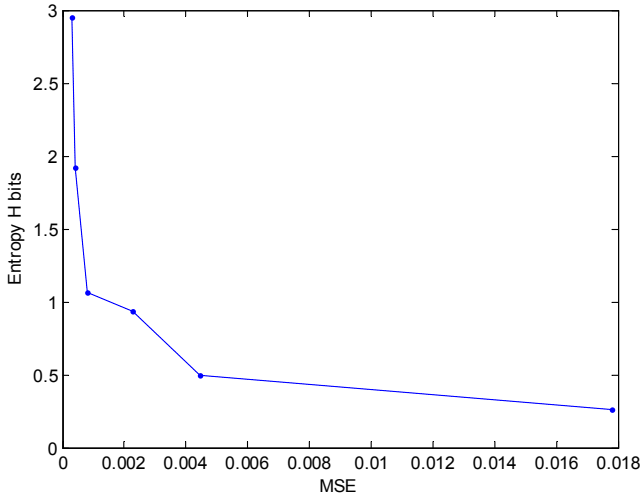

Figure 19. Entropy vs. MSE for the six values of *q*.

d)

The image given in *6397.mat* was loaded in Matlab and is displayed in Figure 20.

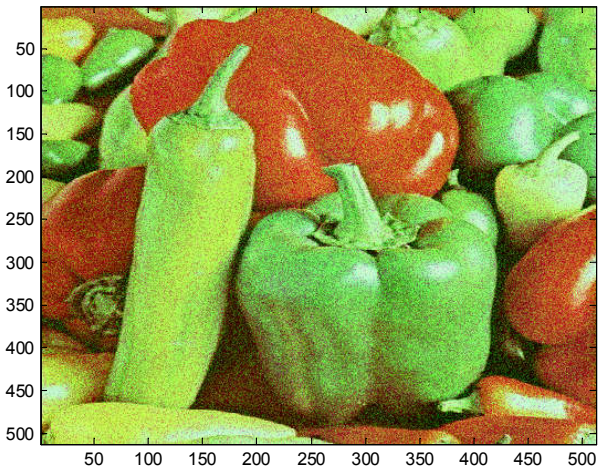

Figure 20. Image in 6397.mat file.

For the 2-D images new Matlab functions were coded for quantization (*encoder_2d.m*) and for inverse quantization (*decoder_2d.m*).

Appendix

The following Matlab code was used in this project.

phi.m

```
function y = phi(t)

y = (t>0).*(t<1);
```

psi.m

```
function y = psi(t)

y = (t>0).*(t<0.5) + -1*(t>0.5).*(t<1);
```

inner_product.m

```
function u = inner_product_f(function_type,j,k,x,t)
% Computes inner product of functions (vector x is for a signal).
% x is a vector for the signal function
% t is the time vector for x
% j,k are indices for phi and psi
% function_type specifies function phi_j,k or psi_j,k
%  value must be string 'phi' or 'psi'
% u is either cj[k] for 'phi' or dj[k] for 'psi'

if (function_type == 'phi')
    phi_jk = (2^(j/2))*phi((2^j)*t - k);
    u = trapz((phi_jk.*x),t);

elseif (function_type == 'psi')
    psi_jk = (2^(j/2))*psi((2^j)*t - k);
    u = trapz((psi_jk.*x),t);
```

```
else
    disp('error')
end
```

haar.m

```
function h_j = haar(h, J, fs)
% Haar wavelet system
% h is the input signal.
% J is the scale of the final space V_J
% fs is the sample frequency
% haar returns matrix h_j. The first row is h_tilde for the approximation
% in V_0. The second row is h_hat_0 for the projection into W_0. Subsequent
% rows are h_hat_1,..., h_hat_J-1 corresponding to projections into
% W_1,..., W_J-1. The rows can be summed to get approximations in
% V_1,...,V_J.

h_j = zeros(1+J,fs+1);
Ts = 1/fs;
t = [0:Ts:1];

% Vo
% c_0[k=0]
c_0 = inner_product_f('phi',0,0,h,t);
h_j(1,:) = c_0*phi(t);

% Wo
% d_0[k=0]
d_0 = inner_product_f('psi',0,0,h,t);
h_j(2,:) = d_0*psi(t);

% W1 ...
if (J > 1)
    for j = 1:(J-1)
        h_j_temp = zeros(1,fs+1);
        for k = 0:(2^j - 1)
            d_j = inner_product_f('psi',j,k,h,t);
            h_j_temp = h_j_temp + d_j*(2^(j/2))*psi((2^j)*t - k);
        end
        h_j(2+j,:) = h_j_temp;
    end
end
```

circular_conv.m

```
function z = circular_conv(x,y)
% x and y are vectors for signals or filters.
% x and y can have different lengths.
% The shorter is padded with zeros.
% z is the circular convolution of x and y.

Nx = length(x);
Ny = length(y);
if (Nx == Ny)
    z = zeros(1,Nx);
    for k = 0:(Nx-1)
        z_temp = 0;
        for n = 0:(Nx-1)
            z_temp = z_temp + x(n+1)*y(mod((k-n),Nx) + 1);
```

```
        end
        z(k+1) = z_temp;
    end
elseif (Nx > Ny)
    y(Nx) = 0; % pad with zeros
    z = zeros(1,Nx);
    for k = 0:(Nx-1)
        z_temp = 0;
        for n = 0:(Nx-1)
            z_temp = z_temp + x(n+1)*y(mod((k-n),Nx) + 1);
        end
        z(k+1) = z_temp;
    end
else
    x(Ny) = 0; % pad with zeros
    z = zeros(1,Ny);
    for k = 0:(Ny-1)
        z_temp = 0;
        for n = 0:(Ny-1)
            z_temp = z_temp + x(n+1)*y(mod((k-n),Ny) + 1);
        end
        z(k+1) = z_temp;
    end
end
```

haar2.m

```
function h_j = haar2(x,T)
% Haar wavelet system
% x is the input signal.
% T is the interval of support
% J1 is the scale of the final space V_J1
% haar returns matrix h_j. The first row is h_tilde for the approximation
% in V_0. The second row is h_hat_0 for the projection into W_0. Subsequent
% rows are h_hat_1,..., h_hat_J1-1 corresponding to projections into
% W_1,..., W_J1-1. The rows can be summed to get approximations in
% V_1,...,V_J1.

a = 1/sqrt(2);
h = [a,a]; % h[n]
g = [a,-a]; % g[n]

djcj = dwtcc(x,h,g,T);

[B,N] = size(djcj);
J1 = B-1;

h_j = zeros(B,2*N);
Ts = 1/(2*N-1);
t = [0:Ts:T];

% V_0
% c_0[k=0]
h_j(1,:) = djcj(B,1)*phi(t);


% W_J1-1, ..., W_0
m = 1;

for j = (J1-1):-1:0
    h_j_temp = zeros(1,2*N);
    for k = 0:(2^j - 1)
```

```
            h_j_temp = h_j_temp + djcj(m,k+1)*(2^(j/2))*psi((2^j)*t - k);
        end
        m = m+1;
        h_j(j+2,:) = h_j_temp;
    end
```

dwtcc.m

```
function djcj = dwtcc(x,h,g,T)
% dwt computes c_j[k] and d_j[k] for the discrete wavelet
% transform of signal x
% x is an even length vector
% h and g are the digital filters
% T is the interval of support for signal x

N = length(x);
J1 = log2(N/T);
djcj = zeros(J1+1,N/2);

% reverse order of h and g for h[-k] and g[-k]
h = h(end:-1:1);
g = g(end:-1:1);

c_J1 = (2^(-J1/2))*x;
m = 0;
c_jp1 = c_J1;

for j = (J1-1):-1:1
    m = m+1;
    L = N/(2^(m-1));
    dj = circular_conv(c_jp1,g);
    cj = circular_conv(c_jp1,h);
    djcj(m,1:L/2) = [dj(3:2:L),dj(1)];
    c_jp1 = cj(1:2:L);

end

m = m+1;
L = N/(2^(m-1));
dj = circular_conv(c_jp1,g);
cj = circular_conv(c_jp1,h);
djcj(m,1:L/2) = dj(1);
c_jp1 = cj(1:2:L);


m = m+1;
djcj(m,1:L/2) = c_jp1;
```

inverseDWT.m

```
function x = inverseDWT(djcj,h,g)
% inverseDWT computes the inverse of the discrete wavelet
% transform for c_j[k] and d_j[k] input in djcj matrix
% x is an even length vector
% h and g are the digital filters
% T is the interval of support for signal x

[B,N] = size(djcj);
J1 = B-1;
Nx = 2*N;
```

```
c_jp1 = djcj(B,1);

for j = J1:-1:1

    L = Nx/(2^(j-1));
    dj = upsample(djcj(j,1:L/2),2);
    c_jp1 = upsample(c_jp1,2);
    c_jp1 = circular_conv(dj,g) + circular_conv(c_jp1,h);

end

x = (2^(J1/2))*c_jp1;
```

encoder.m

```
function z = encoder(x,h,g,T,q)
% Encoder performs uniform scalar quantization on the DWT of signal x.
% x is an even length vector
% h and g are the digital filters
% T is the interval of support for signal x
% q is the step size of the quantizer.

y = dwtcc(x,h,g,T);
z = round((1/q)*y);
```

decoder.m

```
function x = decoder(z,h,g,q)
% Decoder performs inverse quantization on Q[DWT[x]] and then uses the
% inverse DWT to synthesize signal x.
% z is Q[DWT[x]]
% h and g are the digital filters
% T is the interval of support for signal x
% q is the step size of the quantizer.

y_hat = q*z;
x = inverseDWT(y_hat,h,g);
```

entropy.m

```
function H = entropy(z,q)
% entropy computes the entropy of coefficient matrix z[k]
% assumes bin is centered at zero and interval is -1 to 1

k = 2*(1/q) + 1;
[B,N] = size(z);
n = zeros(1,k);
num_elements = 0;

for m = 1:B
    [n_temp,xout] = hist(z(m,1:N),k);
    n = n + n_temp;
    num_elements = num_elements + N;
    N = N/2;
end

H = 0;

for i = 1:k
    p_i = n(i)/num_elements;
    if (p_i ~= 0)
```

```
    H = H + -1*p_i*log2(p_i);
    end
end
```

encoder_2d.m

```
function [zA,zH,zV,zD] = encoder_2d(x,q)
% Encoder performs uniform scalar quantization on the DWT of signal x.
% q is the step size of the quantizer.

[cA,cH,cV,cD] = dwt2(x,'haar');
zA = round((1/q)*cA);
zH = round((1/q)*cH);
zV = round((1/q)*cV);
zD = round((1/q)*cD);
```

decoder_2d.m

```
function x = decoder_2d(zA,zH,zV,zD,q)
% Decoder performs inverse quantization on Q[DWT[x]] and then uses the
% q is the step size of the quantizer.

cA = q*zA;
cH = q*zH;
cV = q*zV;
cD = q*zD;
x = idwt2(cA,cH,cV,cD,'haar');
```

Q1_script.m

```
clc; clear; close all;
% ECE 6397 Matlab project
% Q1

%1a
h = (load('signal1.dat', '-ascii'));
N = length(h);

fs = 1023;
Ts = 1./fs; % period for sampling freq f = 1023 Hz
t = [0:Ts:1];
figure(1)
plot(t,h)
xlabel('t seconds');
ylabel('h(t)');

%b
% inner product function and Haar function

%c
% For signal h(t), h_tilde and h_hat are plotted for j=0,1,...,6
J = 7;
h_j = haar(h, J, fs);

figure(2)

for v = 0:3
    temp_title = sprintf('Projection in W_%d',v);

    subplot(2,2,v+1);
    plot(t,-1*h_j(v+2,:));
```

```
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
end

figure(3)

for v = 4:(J-1)
    temp_title = sprintf('Projection in W_%d',v);

    subplot(2,2,v-3);
    plot(t,-1*h_j(v+2,:));
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
end

figure(4)
h_j_temp = zeros(1,N);

for v = 0:3
    h_j_temp = h_j_temp + h_j(v+1,:);
    temp_title = sprintf('Approximation in V_%d',v);

    subplot(2,2,v+1);
    plot(t,-1*h_j_temp);
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
end

figure(5)

for v = 4:J
    h_j_temp = h_j_temp + h_j(v+1,:);
    temp_title = sprintf('Approximation in V_%d',v);

    subplot(2,2,v-3);
    plot(t,-1*h_j_temp);
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
end

%d
% Plot signal2.dat and filter1.dat

signal2 = (load('signal2.dat', '-ascii'));
filter1 = (load('filter1.dat', '-ascii'));

figure(6)
plot(signal2)
axis([0 300 0 1.5])
title('signal2')

figure(7)
plot(filter1)
title('filter1')

%e
% circular convolution function

%f
```

```
% plot linear convolution of signal2 and filter1
z_linear = conv(signal2,filter1);

figure(8)
plot(z_linear)
title('Linear convolution of signal2 and filter1')

% plot circular convolution of signal2 and filter1
z_circ = circular_conv(signal2,filter1);

figure(9)
plot(z_circ)
title('Circular convolution of signal2 and filter1')

%g
%

T = 1;
h_j = haar2(h,T);

figure(10)
m = 1;
for v = 0:3
    temp_title = sprintf('Projection in W_%d',v);

    subplot(2,2,m);
    plot(t,h_j(v+2,:));
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
    m = m+1;
end

figure(11)
m = 1;
for v = 4:7
    temp_title = sprintf('Projection in W_%d',v);

    subplot(2,2,m);
    plot(t,h_j(v+2,:));
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
    m = m+1;
end

figure(12)
m = 1;
for v = 8:9
    temp_title = sprintf('Projection in W_%d',v);

    subplot(1,2,m);
    plot(t,h_j(v+2,:));
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
    m = m+1;
end

figure(13)
h_j_temp = zeros(1,N);
m = 1;
for v = 0:3
```

```
    h_j_temp = h_j_temp + h_j(v+1,:);
    temp_title = sprintf('Approximation in V_%d',v);

    subplot(2,2,m);
    plot(t,h_j_temp);
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
    m = m+1;
end

figure(14)
m = 1;
for v = 4:7
    h_j_temp = h_j_temp + h_j(v+1,:);
    temp_title = sprintf('Approximation in V_%d',v);

    subplot(2,2,m);
    plot(t,h_j_temp);
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
    m = m+1;
end

figure(15)
m = 1;
for v = 8:10
    h_j_temp = h_j_temp + h_j(v+1,:);
    temp_title = sprintf('Approximation in V_%d',v);

    subplot(2,2,m);
    plot(t,h_j_temp);
    title(temp_title);
    xlabel('t seconds')
    ylabel('h_j (t)');
    m = m+1;
end
```

Q2c_script.m

```
clc; clear; close all;
% ECE 6397 Matlab project
% Q2

%2c
x = (load('signal1.dat', '-ascii'));
N = length(x);
T = 1;

a = 1/sqrt(2);
h = [a,a]; % h[n]
g = [a,-a]; % g[n]

q = [0.001,0.002,0.005,0.01,0.02,0.05];
H = zeros(1,length(q));
MSE = zeros(1,length(q));

figure(1)

for m = 1:length(q)
    z = encoder(x,h,g,T,q(m));
```

```
    x_hat = decoder(z,h,g,q(m));

    temp_title = sprintf('Decoded x(t) for q = %5.3f',q(m));
    subplot(3,2,m);
    plot(x_hat)
    title(temp_title)
    xlabel('t seconds')
    ylabel('x(t)')
    axis([0 1024 -0.5 1.5])

    H(m) = entropy(z,q(m));
    MSE(m) = (1/N)*sum((x-x_hat).^2);
end

figure(2)
plot(MSE,H,'.--')
xlabel('MSE')
ylabel('Entropy H bits')

figure(3)
plot(q,H,'.--')
ylabel('Entropy H bits')
xlabel('q value')

figure(4)
plot(q,MSE,'.--')
ylabel('MSE')
xlabel('q value')
```

Q2d_script.m

```
clc; clear; close all;
% ECE 6397 Matlab project
% Q2

%2d
I = load('6397.mat');
figure(1)
image(I.IM6397)

q = [100,200,500,750,900];
H = zeros(1,length(q));
MSE = zeros(1,length(q));

for m = 1:length(q)
[zAr,zHr,zVr,zDr] = encoder_2d(I.IM6397(:,:,1),q(m));
[zAg,zHg,zVg,zDg] = encoder_2d(I.IM6397(:,:,2),q(m));
[zAb,zHb,zVb,zDb] = encoder_2d(I.IM6397(:,:,3),q(m));

Ir = decoder_2d(zAr,zHr,zVr,zDr,q(m));
Ig = decoder_2d(zAg,zHg,zVg,zDg,q(m));
Ib = decoder_2d(zAb,zHb,zVb,zDb,q(m));

figure(m+1)
temp_title = sprintf('Decoded image for q = %5.3f',q(m));
imshow(cat(3,Ir,Ig,Ib))

end
```