Wavelets

Phillip K. Poon
College of Optical Sciences, University of Arizona, Tucson, Arizona
ppoon@optics.arizona.edu

Abstract: Wavelets are useful for analyzing both the local frequency and time behavior of signals. Fourier analysis of functions with large localized derivatives, such as step functions, tend to require many coefficients to represent the signal in the Fourier domain. Wavelets analysis can provide accurate representation of such signals because they not only take advantage of providing frequency and time information but the freeform to use a large set of completely different basis functions from ones that vary smoothly similar to sinc functions to ones that oscillate rapidly such as the rect function.
This project will provide a practical explanation of the continuous and discrete wavelet transforms. An in depth theory of wavelets and proofs are outside the scope (and useful length) of the paper. In doing so, we hope to show several examples of the utility of the continuous and discrete wavelet transforms (DWT). We will compare the DWT to the Discrete Fourier Transform (DFT) and show in certain we can outperform the compression capability of the DFT as measured by the image quality versus number of coefficients retained using either transform.

© 2012 Optical Society of America

References and links

1. C. S. Burrus, R. A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms: A Primer* (Prentice Hall, 1997), 1st ed.
2. G. Strang and T. Nguyen, *Wavelets and Filter Banks* (Wellesley-Cambridge Press, 1996).
3. S. G. Mallat, *A Wavelet Tour of Signal Processing, The Sparse Way* (Academic Press, 2009).
4. D. Donoho, "Unconditional bases are optimal bases for data compression and for statistical estimation," Applied and computational harmonic analysis **1**, 100–115 (1993).
5. S. Sengupta, "Digitial voice and picture communication," University Lecture (2008).
6. S. Mallat, "Multiresolution approximations and wavelet orthonormal bases of l2 (r)," Trans. Amer. Math. Soc **315** (1989).

1. Motivation for studying and using Wavelets

Wavelet analysis is powerful because it allows for a time-frequency localization of a signal [1, 2, 3] and is well suited for signals with non-periodic, transient, or time-varying phenomena. Wavelet analysis uses two types of functions, the scaling and wavelet functions. Scaling and wavelet functions are related to each other within a wavelet family. Scaling and wavelet families of functions come in many shapes and sizes. The size of the wavelet expansion coefficients drop off rapidly for a large class of signals, and this can be proven to be near optimal [4].

For example given a box function, the Fourier Transform would need an large amount of complex coefficients to capture the energy from the original signal.In fact, there is an family of scaling and wavelet functions that is essentially a series of scaled and shifted box functions, called the Haar wavelet, thus there would be small finite amount of expansion coefficients to represent the original box function.

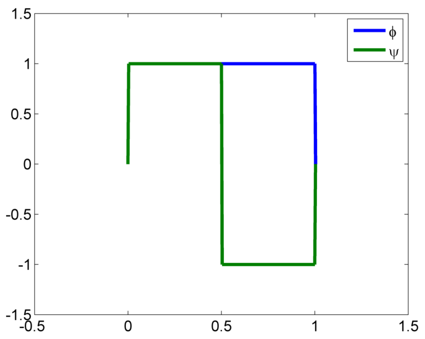

Fig. 1. Haar scaling and wavelet functions. The scaling function ϕ is in blue. The wavelet function ψ is in green. Generated using the MATLAB wavefun('Haar') command.

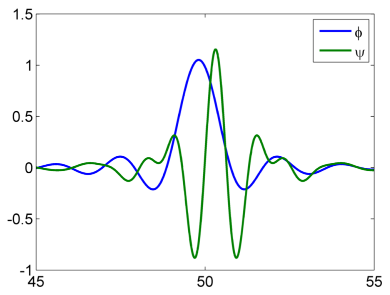

Fig. 2. Discrete Meyer scaling and wavelet functions. The scaling function ϕ is in blue. The wavelet function ψ is in green. Generated using the MATLAB wavefun('dmey') command.

Wavelets are useful for multiresolution analysis [1]. Given an initial wavelet function, one can generate a higher frequency version by scaling the original wavelet function. This is iterated until one has the desired frequency resolution to analyze a signal. Using a set of scaled and shifted wavelets, we can use them as filters to create a filter bank to examine a signal's behavior at certain frequency bands.

There are many different kinds of wavelets, the choice of which "family" of wavelets to use often depends on the application or signal one is trying to analyze [1]. For example, the Haar wavelet is box function but the Discrete Meyer wavelet and Daubechies wavelet can rapidly oscillate depending the number of taps (or coefficients), Fig. 1, 2 and 3.

2. Wavelet Transforms

We can write any well behaved signal $f(x)$ as a linear superposition of basis function which spans Hilbert space. In Fourier analysis, these basis functions are complex exponentials and the expansion is expressed through the Inverse Fourier Transform

$$f(x) = \int_{-\infty}^{\infty} F(k) \exp(ikx) dk \tag{1}$$

Using the orthogonality of complex exponentials we can simply solve for the expansion coefficients and obtain the Fourier Transform

$$F(k) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x) \exp(-ikx) dx \tag{2}$$

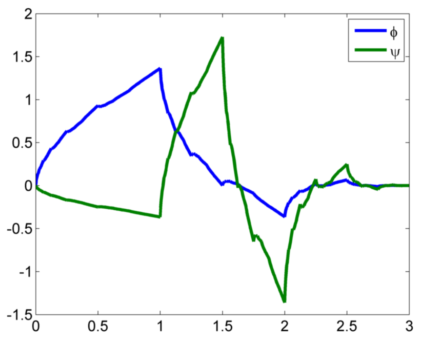

Fig. 3. Daubenchies 4 tap wavelet scaling and wavelet functions. The scaling function ϕ is in blue. The wavelet function ψ is in green. Generated using the MATLAB wavefun('Db2') command.

In similar fashion we can expand any well behaved function $f(x)$ in terms of the scaling $\phi(x)$ and wavelet functions $\psi(x)$

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \phi(x-k) + \sum_{k=-\infty}^{\infty} \sum_{j=0}^{\infty} d_{j,k} \psi(2^j x - k) \tag{3}$$

The above expansion is written in terms of a summation rather than an integral because the expansion is over discrete values of scale and shift, j and k. Many of the concepts from Fourier Analysis can be applied to Wavelet analysis. As we will see, wavelet analysis allows for more flexibility.

3. Multiresolution: The Scaling Functions

A subset of functions in the $L^2(\mathbf{R})$ space can be expanded in terms of only scaling functions [1, 3, 5]

$$f(x) = \sum_k a_k \phi_k(x). \tag{4}$$

The scaling functions are not to be confused with the wavelet functions $\psi(x)$, however they are related. A two-dimensional family of functions is generated by scaling and translating the original scaling function $\phi(x)$ by

$$\phi_{j,k} = 2^{j/2} \phi(2^j x - k). \tag{5}$$

If we fix $j = 0$ and are allowed to only vary k then we have a small subspace, V_0, that our set of functions $\{\phi_{j=0,k}\}$ spans. However if we are allowed to have $j = 1$ then the subspace V_1, the set $\{\phi_{j=1,k}\}$ spans, will have more members than the original subspace V_0. This is because $\phi(2x)$ is twice as narrow and thus has higher frequency resolution the $\phi(x)$. Similarly the subspace V_2 spanned by $\{\phi_{j=2,k}\}$ will have even more members the the previous subspaces, V_0 and V_1.

We can extend this argument and obtain a nesting of the spanned spaces [1, 5]

$$\ldots V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \ldots \subset L^2(\mathbf{R}) \tag{6}$$

It follows from the nesting of subspaces that if $\phi(x)$ is in the subspace V_0 then it is also in the subspace V_1, the space forms by $\phi(2x)$. Conceptually any $\phi(x)$ is a superposition of higher frequencies scalings of the original scaling function, this leads to what is known as the multiresolution analysis (MRA) equation or dilation equation

$$\phi(x) = \sum_n h(n)\sqrt{2}\phi(2x-n) \tag{7}$$

the MRA equation can be applied at any value of j through substitution of variables.

4. Multiresolution: The Wavelet Functions

We can define the difference between subspace V_0 and V_1 as the subspace W_1

$$V_j = V_{j-1} \oplus W_{j-1} \tag{8}$$

where $\oplus$ is the direct sum of two subspaces. It follows that

$$V_2 = V_0 \oplus W_0 \oplus W_1 \tag{9}$$

In fact the entire square integrable space $L^2(\mathbf{R})$ can be spanned if we apply this result iteratively

$$L^2 = V_0 \oplus W_0 \oplus W_1 \oplus W_2 \oplus \dots \tag{10}$$

We now are in a position to connect the wavelet functions to the scaling functions. The wavelet functions span the difference subspaces W_j, but we showed conceptually that $W_j \subset V_{j+1}$, in other words, the wavelet functions can be created by the next higher frequency (j+1) scaling functions. Thus

$$\psi(x) = \sum_n h_1(n)\sqrt{2}\phi(2x-n) \tag{11}$$

A subspace nesting argument similar to the one demonstrated in section 3 allows us to write a MRA equation or dilation equation for the wavelet functions

$$\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k) \tag{12}$$

4.1. Discrete Wavelet Transform

The argument in Eq. 10 provides the theoretical justification for an expansion of the function $f(x)$ in the space $L^2(\mathbf{R})$.

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \phi(x-k) + \sum_{k=-\infty}^{\infty}\sum_{j=0}^{\infty} d_{j,k}\psi(2^j x - k) \tag{13}$$

which is known as the inverse discrete wavelet transform (IDWT) [1]. The first term is an expansion in terms of scaling functions j is fixed and therefore the first term does not provide multiresolution information only time localization. The second term however allows for both resolution and time to vary.

It is often convention that $j = 0$ for the scaling functions in the first summation term of the IDWT, however it is not required, the choice of j for the scaling function sets the base resolution, any of the wavelet coefficients must be at least the same resolution or higher than the resolution provided by the scaling coefficients.

Assuming the scaling and wavelet functions obey orthogonality we can obtain the discrete wavelet transform

$$c(j,k) = \int g(x)\phi_{j,k}(x)dx \tag{14}$$

$$d(j,k) = \int g(x)\psi_{j,k}(x)dx \tag{15}$$

5. From Multiresolution to Filter Banks

We will now show that we can create an easy to use algorithm to compute the wavelet transform of a discrete signal [1]. The algorithm is named after French mathematician Mallet and it is recursive and is based on Eq. 7 [6]. If we replace x in Eq. 7 with $2^j x - k$ then we can rewrite the equation as

$$\phi(2^j x - k) = \sum_n h(n)\sqrt{2}\phi(2^{j+1}x - 2k - n) \tag{16}$$

Then letting $m = 2k + n$

$$\phi(2^j x - k) = \sum_n h(m - 2k)\sqrt{2}\phi(2^{j+1}x - m) \tag{17}$$

If a function f(x) can be written as a linear superposition of $\phi(2^{j+1}x - m)$, then we say it is in the subspace V_{j+1}. The superposition is written as

$$f(x) = \sum_k c(j+1,k)2^{(j+1)/2}\phi(2^{j+1}x - k) \tag{18}$$

However we can also express $f(x)$ as a superposition of the scaling functions at one lower scale as long as we allow the wavelets to cover the difference between the j and $j+1$ scale. This leads us to

$$f(x) = \sum_k c(j,k)2^{j/2}\phi(2^j x - k) + \sum_k d(j,k)2^{(j/2)}\psi(2^j x - k) \tag{19}$$

If we can assume the scaling and wavelet functions are orthonormal, $< \phi_{j,k}|\phi_{j',k'} >= \delta_{jj'}\delta_{kk'}$, $< \psi_{j,k}|\psi_{j',k'} >= \delta_{jj'}\delta_{kk'}$, and $< \phi_{j,k}|\psi_{j',k'} >= 0$, then we can express the coefficients

$$c(j,k) = \sum_m h(m-2k)c(j+1,m) \tag{20}$$

$$d(j,k) = \sum_m h_1(m-2k)c(j+1,m) \tag{21}$$

The above equations represent a convolution operation and a downsampling opertation. The convolution operation is called filtering. A filter bank is an array of bandpass filters and the combination of Eq. 20 and Eq. 21 creates a filter bank. The convolution with the scaling functions is a low-pass filtering operation, while the convolution with the wavelet functions is a high-pass filter operation, see Fig. 4. We can repeat this as many times as need to obtain the necessary level of analysis. A two level level DWT analysis is represented in Fig. 5 [1].

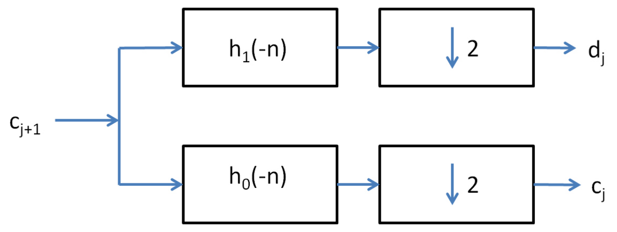

Fig. 4. Two band analysis filter bank.

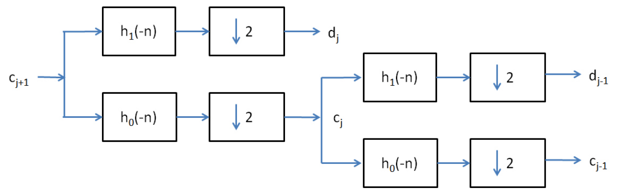

Fig. 5. Two-Stage two-band analysis filter bank.

The problem of knowing the coefficients at the highest level $c(j = J,k)$ and $d(j = J,k)$ *a priori* can be solved if we realize that for a high enough scale, the scaling functions approximate delta functions in Eq. 14. The coefficients at this level is simply the signal $f(x)$. The recursive application of Eq. 20 and Eq. 21, known as the Mallat algorithm, is a simple way to calculate the DWT of a signal [1].

In order to reconstruct (or synthesize) the signal, the operations in the decomposition (or analysis) of a function are simply reversed taking care to keep track of the signal length. Fig. 6 shows the procedure for a 2 level wavelet analysis and synthesis.

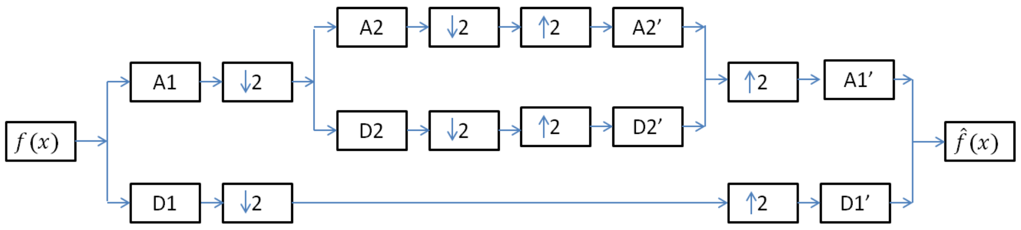

Fig. 6. Two-Stage Two-Band analysis and synthesis filter bank.

6. One dimensional signal wavelet analysis

Using the Daubechies 4 tap scaling and wavelet filter with the Mallat algorithm we just described, we can compute a one level wavelet analysis to produce the approximation and detail coefficients of a rect function, see Fig. 7. The approximation and detail coefficients are shown in Fig. 8(a) and Fig. 8(b). The approximation coefficients are the result of the convolution of the original signal with the scaling coefficients, followed by a downsampling operation. Similarly the detail coefficients are the result of the convolution of the original signal with the wavelet coefficients, followed by a downsampling operation. Reversing this procedure allows us to preform the reconstruction or synthesis of the original signal shown in Fig. 9.

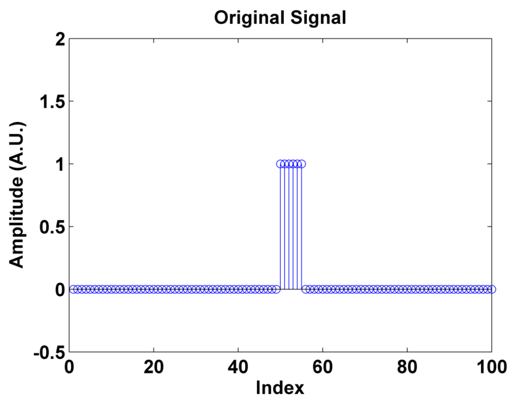

Fig. 7. Original signal is approximately a Rect function.

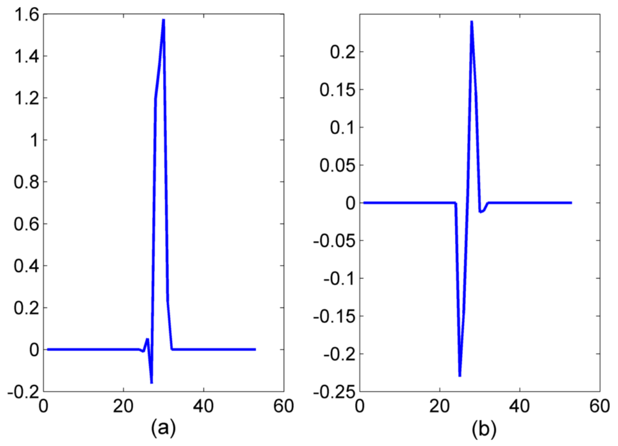

Fig. 8. One level (a) Approximation coefficients (b) Detail coefficients.

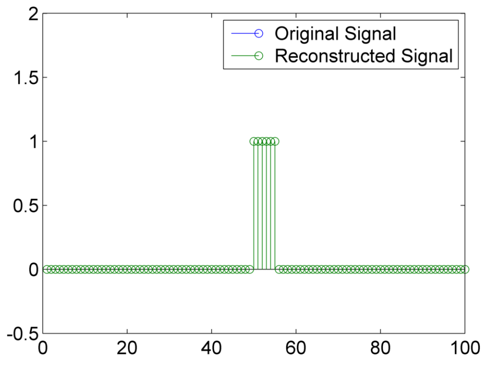

Fig. 9. One level reconstruction of the original signal.

7. Image analysis with wavelets

We will now show how wavelet analysis can be applied to digital images. We will use the iconic cameraman picture as our input signal. The process for computing the 2D wavelet transform is conceptually similar to the 1D wavelet transform. The following steps outline the 2D Mallat algorithm:

1. Take the scaling coefficients and convolve with each row in $f(x,y)$ and then deleting every other column to produce the signal at point Z in Fig. 10
2. Take the wavelet coefficients and convolve with each row in $f(x,y)$ and deleting every other column to product the signal at point C in Fig. 10
3. The columns of Z are high pass filtered with the same wavelet coefficients and then deleting every other row to produce the LH filtered image
4. The columns of Z are low pass filtered with the same scaling coefficients and then deleting every other row to produce the HL filtered image
5. Similarly C is high and low pass filtered with each column and decimated across rows to produce the HH and LL filtered images
6. Repeat using the current LL filtered image as the new input image as many times as needed to reach the appropriate level of analysis or until the size of the filtered image is one pixel in either dimension

The results of a one level 2D wavelet transform using the Haar scaling and wavelet coefficients are shown in Fig. 11(a). Starting from the upper left and going clockwise, each quadrant represents the LL, HL, HH, and LH filtered image. Each filtered image is a quarter of the area of the original digital image in pixels. The two level 2D wavelet transform is shown in Fig. 11(b). Since the two level 2D wavelet transform is a quarter of the area of the one level filtered image it replaces the the LL filtered image from the one level transform. This can be repeated as many times as needed or until the size of the filtered images is one pixel in either dimension.

Note that the LL filtered image is a lower resolution approximation to the original cameraman digital image with a quarter of the pixels. This means that the two dimensional wavelet transform of a digital image can be used to create a low resolution version of the original digital image which can be transmitted in place of the original digital image in low bandwidth communication channels. When more bandwidth is available, the three other filtered subimages (HL, LH, HH) can be sent to allow the receiver to reconstruct the original digital image.

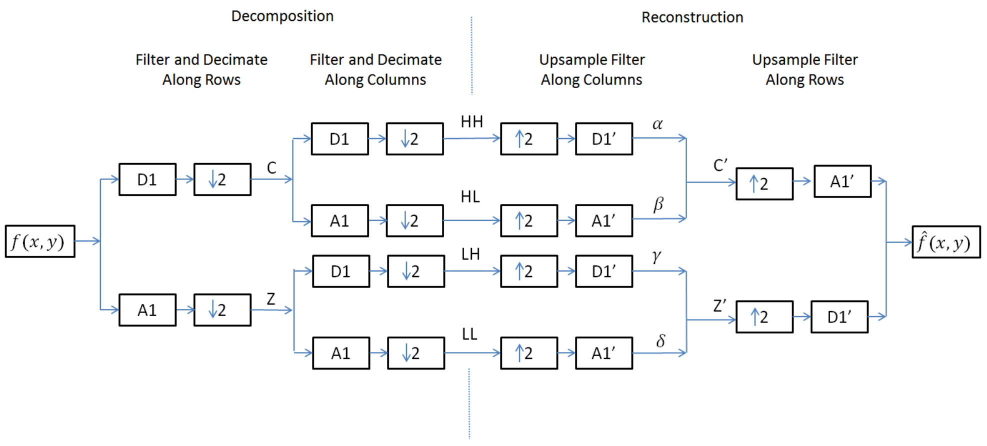
Decomposition
Reconstruction
Filter and Decimate Along Rows
Filter and Decimate Along Columns
Upsample Filter Along Columns
Upsample Filter Along Rows
$f(x,y)$
D1
A1
↓2
C
Z
HH
HL
LH
LL
↑2
D1′
A1′
α
β
γ
δ
C′
Z′
$\hat{f}(x,y)$

Fig. 10. Filter Bank Tree Graph for 2D Wavelet Analysis.

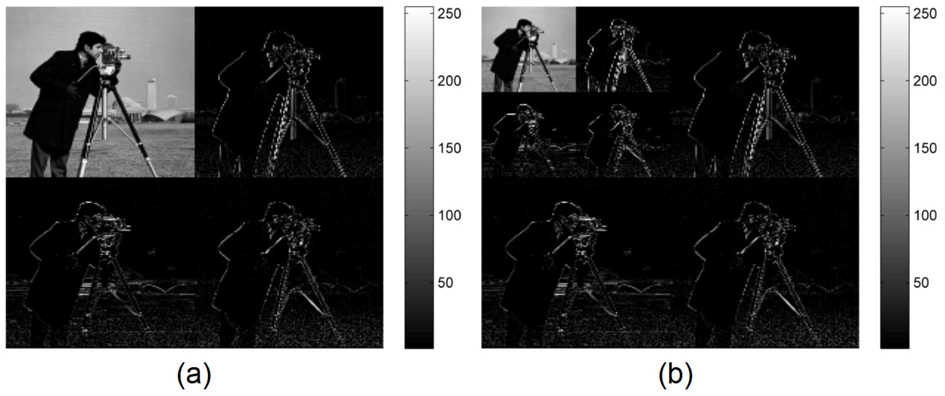

Fig. 11. 2D Wavelet Transform of the iconic cameraman digital image. (a) The one level 2D wavelet transform The upper left is the LL1 (low pass along the rows and columns), the upper right is the HL1 (high pass filtered along rows, low pass filtered along columns), the lower left is the LH1 (low pass filtered along rows, high pass filtered along columns), the lower right is the HH1 (high pass filtered along the rows and columns). (b) The two level 2D wavelet transform. The upper left portion where LL1 was in (a) is replaced by the 2 level 2D wavelet transform, each quadrant represents the same respective filter operations along the rows and the columns as in (a).

8. Conclusion

We have shown conceptually and mathematically how scaling and wavelet functions provide a multiresolution analysis of one and two dimensional signals. We also derived a rapid and easy to implement form of the wavelet transform called the Mallat algorithm. We also demonstrated the wavelet transform for a one dimensional signal and a two dimensional signal. The two dimensional wavelet transform of a digital image can be iterated to create a low resolution version of the original digital image. This low resolution version can be transmitted in place of the original digital image in low bandwidth communication channels.

9. MATLAB Code

I wrote my own simplified MATLAB code to compute the discrete forward and inverse transforms for one and two dimensional signals, rather than use the built in MATLAB code. I found that writing my own code made learning the Mallat algorithm easier to understand because the built in MATLAB wavelet transform functions are difficult to follow and is typically written to take into account special cases that are not pertinent to learning the Mallat algorithm.

9.1. Example of a 1D DWT using the custom 1D DWT MATLAB Function

```
clearvars; close all; clc

% Construct elementary one-dimensional signal s.
s = zeros(1,100);
s(50:55) = 1;

% Perform single-level dwt of s using db2.
wname = 'db4'  %Choose the wavelet family you want to use

[ca1,cd1] = custom_dwt(s,wname);

ss = custom_idwt(ca1,cd1,wname);

% Check reconstruction.
error = norm(s(:)-ss(:))

%Plotting
figure;
subplot(1,2,1)
plot(ca1)
title('(a)')
subplot(1,2,2)
plot(cd1)
title('(b)')

figure;
stem(s)
hold all
stem(ss)
ylim([-0.5 2])
legend('Original Signal','Reconstructed Signal')
```

9.2. Example of a 2D DWT using the custom 2D DWT MATLAB Function

```
close all; clc; clear all;


X = imread('cameraman.tif');
X = sum(X,3);

s = X;
clear X;


wname = 'Haar'

%Uses the custom_dwt2 to compute the 2D wavelet transform
```

```matlab

[HH1,HL1,LH1,LL1] = custom_dwt2(s,wname);

[HH2,HL2,LH2,LL2] = custom_dwt2(LL1,wname);

level1 = [wcodemat(LL1,255) wcodemat(HL1,255); wcodemat(LH1,255) ...
    wcodemat(HH1,255)];

level2_corner = [wcodemat(LL2,255) wcodemat(HL2,255); wcodemat(LH2,255) ...
    wcodemat(HH2,255)];

level2 = [level2_corner wcodemat(HL1,255); wcodemat(LH1,255) ...
    wcodemat(HH1,255)];


figure;
imagesc(s)
colormap gray
colorbar
axis off


figure;
imagesc(level1);
colormap gray
colorbar
axis off


figure;
imagesc(level2);
colormap gray
colorbar
axis off


%Calls the inverse custom DWT2 Algorithm I made
[x_hat] = custom_idwt2(HH1,HL1,LH1,LL1,wname);

figure;
imagesc(x_hat)
title('Reconstructed Image')
colormap gray
colorbar

norm(s(:) - x_hat(:))
```

9.3. *Custom 1D DWT MATLAB Function Code*

```matlab
function [a_out,c_out] = custom_dwt(s_in,wname)
%Custom 1D DWT
%s - input signal
%wname - Family of scaling and wavelet coefficients to use
%a_out - Approximation coefficients (Low Pass)
%c_out - Detail coefficients (High Pass)

[Lo_D,Hi_D] = wfilters(wname,'d');

ca1 = conv(Lo_D,s_in); %Convolves with low pass filter
cd1 = conv(Hi_D,s_in); %Convolves with high pass filter
```

```

%Downsample by 2 to perserve signal length
ca1 = ca1(2:2:length(ca1));
cd1 = cd1(2:2:length(cd1));

a_out = ca1;
c_out = cd1;

end
```

9.4. *Custom 1D DWT MATLAB Function Code*

```
function [a_out,c_out] = custom_dwt(s_in,wname)
%Custom 1D DWT
%s - input signal
%wname - Family of scaling and wavelet coefficients to use
%a_out - Approximation coefficients (Low Pass)
%c_out - Detail coefficients (High Pass)

[Lo_D,Hi_D] = wfilters(wname,'d');

ca1 = conv(Lo_D,s_in); %Convolves with low pass filter
cd1 = conv(Hi_D,s_in); %Convolves with high pass filter

%Downsample by 2 to perserve signal length
ca1 = ca1(2:2:length(ca1));
cd1 = cd1(2:2:length(cd1));

a_out = ca1;
c_out = cd1;

end
```

9.5. *Custom 1D IDWT MATLAB Function Code*

```
function [ss] = custom_idwt(ca,cd,wname)
%Custom 1D IDWT
%wname - Family of scaling and wavelet coefficients to use
%ca - Input Approximation coefficients (Low Pass)
%cd - Input Detail coefficients (High Pass)
%ss - output synthesis signal

[Lo_R,Hi_R] = wfilters(wname,'r');

%Used to compute the length of the output synthesis signal
lx = 2*length(ca);
lf = length(Lo_R);
s_len = lx-lf+2;

%Calls a custom function to upsample the input approximation and detail
%coefficients
ca = custom_upsample(ca);
cd = custom_upsample(cd);

RA = conv(ca,Lo_R); %Convolves with low pass filter
RD = conv(cd,Hi_R); %Convolves with high pass filter

```

```
%Calls a custom function to crop to preserve length
RA = custom_wkeep1(RA,s_len);
RD = custom_wkeep1(RD,s_len);

%Addition Operation
ss = RA + RD;

end
```

9.6. *Custom Upsample MATLAB Function Code*

```
function [x_temp] = custom_upsample(x)
%x is the input array
%x_upsampled is the output array, which places a single 0 in between
%the elements of the original array
%if x has an even number of elements then the length is

%Phillip K Poon 09 May 2012
%ppoon@optics.arizona.edu

%Initialize the output array
x_temp = [];

%The number of for loops depends on whether the input array has
%an even or odd number of elements
switch mod(length(x),2)
    case 0
        endlength = 2*length(x);

    case 1
        endlength = 2*length(x)+1;
end

%The maps every other element in the original array x, to the
%even indices of x_temp of
count1 = 1;
for ii = 2:2:endlength
    ii;
    x_temp(ii-1) = x(count1);
    count1 = count1 + 1;
end


end
```

9.7. *Custom Signal Length Cropping MATLAB Function Code*

```
function y = custom_wkeep1(y,len)
%WKEEP1  Keep part of a vector.
%
%   Y = WKEEP1(X,L,OPT) extracts the vector Y
%   from the vector X. The length of Y is L.
%   If OPT = 'c' ('l' , 'r', respectively), Y is the central
%   (left, right, respectively) part of X.
%   Y = WKEEP1(X,L,FIRST) returns the vector X(FIRST:FIRST+L-1).
%
%   Y = WKEEP1(X,L) is equivalent to Y = WKEEP1(X,L,'c').
```

```


if (length(y) < len )
    return;
end

d = (length(y)-len)/2;
first = 1+floor(d);
last = length(y)-ceil(d);

y = y(first:last);
```

9.8. *Custom 2D DWT MATLAB Function Code*

```
function [HH,HL,LH,LL] = custom_dwt2(s,wname)

%Customer 2D Wavelet Transform
%s is the input digital signal
%wname is the family of scaling and wavelet coefficients to use

%Uses the built-in MATLAB function to look up the scaling and wavelet
%decomposition coefficients
[Lo_D,Hi_D] = wfilters(wname,'d');

%low pass filter and decimate along the rows
for ii = 1:size(s,1);
    row_temp = conv(s(ii,:),Lo_D);
    %Decimate
    Z(ii,:) = row_temp(2:2:length(row_temp));
    clear row_temp;
end

%high pass filter and decimate along the rows
for ii = 1:size(s,1);
    row_temp = conv(s(ii,:),Hi_D);
    %Decimate
    C(ii,:) = row_temp(2:2:length(row_temp));
    clear row_temp;
end

%low pass filter and decimate along the columns after low passing the rows
for ii = 1:size(Z,2);
    col_temp = conv(Z(:,ii),Lo_D);
    %Decimate
    LL(:,ii) = col_temp(2:2:length(col_temp));
    clear col_temp;
end

%high pass filter and decimate along the columns after low passing the rows
for ii = 1:size(Z,2);
    col_temp = conv(Z(:,ii),Hi_D);
    %Decimate
    LH(:,ii) = col_temp(2:2:length(col_temp));
    clear col_temp;
end


%low pass filter and decimate along the columns after high passing the rows
for ii = 1:size(C,2);
    col_temp = conv(C(:,ii),Lo_D);
    %Decimate
    HL(:,ii) = col_temp(2:2:length(col_temp));
    clear col_temp;
end

%high pass filter and decimate along the columns after high passing the ...
    rows
for ii = 1:size(C,2);
    col_temp = conv(C(:,ii),Hi_D);
    %Decimate
    HH(:,ii) = col_temp(2:2:length(col_temp));
    clear col_temp;
end

end
```

9.9. *Custom 2D IDWT MATLAB Function Code*

```
function [x_hat] = custom_idwt2(HH,HL,LH,LL,wname)

[Lo_R,Hi_R] = wfilters(wname,'r');

%anti-high pass filter on HH
for ii = 1:size(HH,2)
    lx = 2*size(HH,2);
    lf = length(Hi_R);
    s_len = lx-lf+2;

    %Upsample
    HH_x(:,ii) = custom_upsample(HH(:,ii));
    %Convolution
    col_temp = conv(HH_x(:,ii),Hi_R);
    %Crop to preserve length
    col_temp = custom_wkeep1(col_temp,s_len);
    alpha(:,ii) = col_temp;

    clear col_temp
end
clear HH_x

%anti-low pass filter on HL
for ii = 1:size(HL,2)
    lx = 2*size(HL,2);
    lf = length(Lo_R);
    s_len = lx-lf+2;

    %Upsample
    HL_x(:,ii) = custom_upsample(HL(:,ii));
    %Convolution
    col_temp = conv(HL_x(:,ii),Lo_R);
    %Crop to preserve length
    col_temp = custom_wkeep1(col_temp,s_len);
    beta(:,ii) = col_temp;

    clear col_temp
end
clear HL_x

C_prime = alpha + beta;

```

```
%anti-high pass filter on LH
for ii = 1:size(LH,2)
    lx = 2*size(LH,2);
    lf = length(Hi_R);
    s_len = lx-lf+2;

    %Upsample
    LH_x(:,ii) = custom_upsample(LH(:,ii));
    %Convolution
    col_temp = conv(LH_x(:,ii),Hi_R);
    %Crop to preserve length
    col_temp = custom_wkeep1(col_temp,s_len);
    gamma(:,ii) = col_temp;

    clear col_temp
end
clear LH_x

%anti-low pass filter on LL
for ii = 1:size(LL,2)
    lx = 2*size(LL,2);
    lf = length(Lo_R);
    s_len = lx-lf+2;

    %Upsample
    LL_x(:,ii) = custom_upsample(LL(:,ii));
    %Convolution
    col_temp = conv(LL_x(:,ii),Lo_R);
    %Crop to preserve length
    col_temp = custom_wkeep1(col_temp,s_len);
    Δ(:,ii) = col_temp;

    clear col_temp
end
clear LL_x
Z_prime = gamma + Δ;


%anti_high pass filter on C_prime
for ii = 1:size(C_prime,1)

    lx = 2*size(C_prime,1);
    lf = length(Hi_R);
    s_len = lx-lf+2;

    %Upsample
    C_prime_x(:,ii) = custom_upsample(C_prime(ii,:));

    %Convolution
    row_temp = conv(C_prime_x(:,ii),Hi_R);
    %Crop to preserve length
    theta(ii,:) = custom_wkeep1(row_temp,s_len);
    clear row_temp;

end
clear C_prime_x;

%anti_low pass filter on C_prime
for ii = 1:size(Z_prime,1)

    lx = 2*size(Z_prime,1);
    lf = length(Lo_R);
    s_len = lx-lf+2;

    %Upsample
    Z_prime_x(:,ii) = custom_upsample(Z_prime(ii,:));

    %Convolution
    row_temp = conv(Z_prime_x(:,ii),Lo_R);
    %Crop to preserve length
    rho(ii,:) = custom_wkeep1(row_temp,s_len);
    clear row_temp;

end
clear Z_prime_x;

x_hat = theta + rho;


end
```