Wavelets Assignment

Sushrut Rajesh Deshpande - r0822692, Sarat Srikala Kondamudi - r0775328

December 2020

1 Task 2.1

We compute the wavelet transform of the function for 20 levels. α is the slope of the line

$$\log_2 W_{jk} \leq \log_2 K - \frac{1}{2} - j\alpha$$

By calculating the slope of RHS we can get α. The plot of $\log |W_{jk}|$ for the given function is as follows

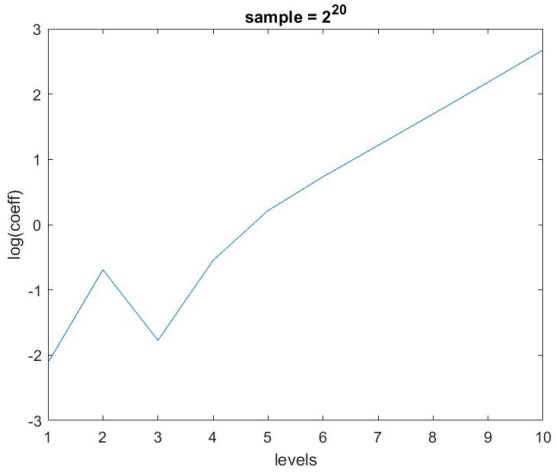

Figure 1: f_1

We calculated alpha by taking slope from 2nd level to the final level. We got alpha to be equal to 0.4210. Which is expected as it can never be truly zero even though there is a discontinuity in the function.

1.1 Parameters:

The number of samples taken is 2^{20}. The number of levels computed is 20 but plotted for the first 10 here. As when we increase the levels there are less and less coefficients to cutoff from. The same parameters are used in rest of the experiments of this section. The key part of code is as follows

```
    n = 2^20;
    t = linspace(-1,1,n)
    f1 = zeros(size(t));
    I = find(t >= 0);
    f1(I) = 1;
    N=20;  % Number of levels
    [c,l]=wavedec(f1,N,'db4');
    semilogy(abs(c)+1e-15)
    D = detcoef(c,l,level);
     for i = 1:10
    temp  = D{i};
    temp = temp(2^cut(i):end-2^cut(i)); % cut is a matrix that helps in removal of boundaries.
    largecoeff(i) = max(temp);
    end
     figure(2)
 plot(log2(largecoeff)
 alpha1 = (log2(largecoeff(10)) - log2(largecoeff(1)))/9
```

The same principal is applied to the other parts of this section.

2 Question 2.2

When the algorithm finds a particular coefficient large, it, checks its relative index in the array of coefficients and then maps it to the location on [-1,1] by uniformly spreading the coefficients on [-1,1]. The largest coefficient per level will always lie on the point of singularity(after removing the boundary coefficients). This is expected as we make a sudden jump in our function for which the wavelet needs to account for.
Center of support: for a two scale relation

$$\phi(x) = \sqrt{2}\sum_{k} h_k \phi(2x - k)$$

If h_k has N non zero coefficients then the support of the wavelet ϕ is also [0,N]. Hence for the center of support we have to take the N/2 part of the coefficients. Therefore the center of support of the wavelet is near t=0.

3 Task 2.3

The singularity of the function exist at t = 0 and the largest wavelet coefficient per level also exists at singularity(after removing the boundaries).

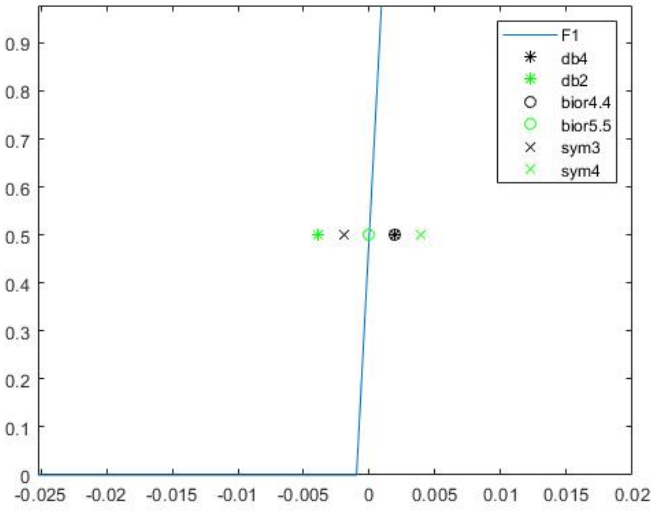

Figure 2: The location of singularity in different wavelet families

Here we can see the plot of where singularity lies for different family of wavelets.

4 Question 2.4

The difference in location of the large wavelet coefficients is due to the different shapes of the wavelets across the families. We need to see the difference in the filter coefficients of these family of wavelets. First we will use the matlab command `[LoD,HiD,LoR,HiR]=wfilters('wname')` to see all the filter coefficients of a given family of wavelets. We need to see the high pass filers, g_k. First, let us compare the difference between 'bior5.5' and 'db4': we can see that the high pass filter of the 'bior5.5' is symmetric, that is the wavelets that belong to the biorthogonal family are symmetric in nature while , if we see the wavelet 'db4' it is not symmetric.

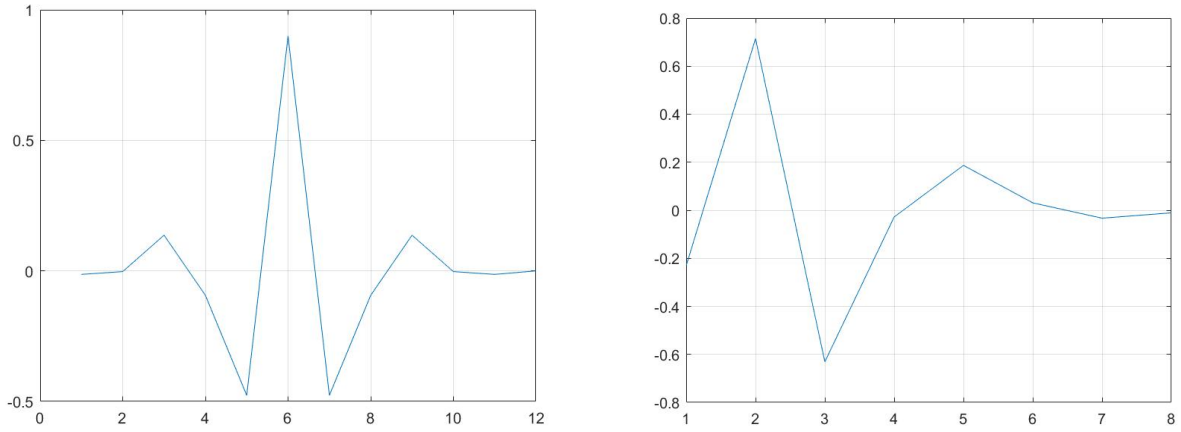

Figure 3: The bior5.5 vs db4

We can clearly see the difference in the symmetry of the two wavelets. For our given function we can see by figure [2] that the wavelet that is closest to zero is the bio orthogonal one. Hence more symmetric the wavelet closer it will be to zero. We can even confirm this claim by seeing that the wavelet of symlet is not symmetric and hence is not exactly on zero.

5 Task 2.5

	db4	sym4	bior3.5	haar	db2	
sigma	alpha	alpha	alpha	alpha	alpha	inference
0	3.8824	1.9186	3.5	1.5	2.5	This is expected as the abs(t) is no impact and the function is differentiable
1	1.489	1.5177	1.5	1.4915	1.519	This is not expected as the abs(t) has impact on the function and we can see a sharp edge at 0 which makes it non differentiable
2	4.4975	3.5609	3.5	1.4774	2.5	This is expected as the taking the square makes the function smooth, hence it is atleast differentiable once

In general if a function is differentiable then the value of $\alpha \geq 1$. But computation of α through wavelets is not straight forward as we have seen before α has to equal 0 for discontinuous function which we did not get before. The results of $\sigma = 1$ are unexpected as we expect the function to not be differentiable as the following plot shows us the function for all the three chosen sigma. As we can see that the function is not smooth at t=0 when $\sigma = 1$, the function shouldn't be uniformly differentiable. But yet we get the alpha value greater than 1.

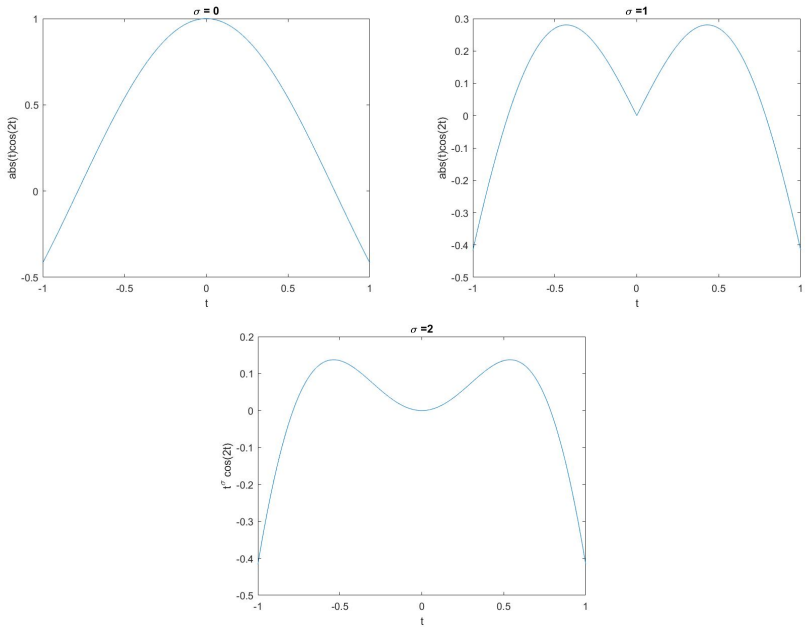

Figure 4: Function for different σ

There is another observation that all the values of alpha for a given sigma are similar across all the wavelets. **Note:** The value of α at σ equal to 1 makes no sense, as the function shouldn't be differentiable at all. This is again seen when we do the experiment for swt.

6 Task 2.6

	alpha
db4	3.0378
bior3.5	3.494
haar	1.4954

We can see a difference in the regularity across the wavelets. The function that we are plotting is smooth and differentiable, hence the value of found α should be greater that 1. As we can say the function is at least differentiable once(technically this function is infinitely differentiable). We see for the above 3 cases we have alpha to be greater than 1, but all the 3 values are not same. This also has to do with the number of vanishing moments in a wavelet for a given support. Each wavelet has vanishing moments, they are half of the number of coefficients. A vanishing moment restricts the way a wavelet can represent a polynomial. 'Db4' can encode polynomial of two coefficient while 'Db2' can only do it for one coefficient. 'bior3.5' has 3 vanishing moments and can approximate a polynomial for up to 3 coefficients. So, here we will prefer a function that can approximate the polynomial for a more coefficients. Hence with the help of the vanishing moments we can say which wavelet is more suitable than the other to approximate the given function.

7 Question 2.7

In this setting of SWT, the number of coefficients computed by the command `swt(X,L,'wname')` at any level equals the number of sample points given, hence we call this 'Redundant' wavelet transform as every coefficient corresponds to the respective location in time .

8 Task 2.8

Function 1				
	alpha	0.489		
Function 2				
	db4	sym4	bior3.5	haar
sigma				
0	1.9487	2.674	1.8918	1.5
1	1.6937	1.6942	1.6971	1.4998
2	1.9486	2.674	1.8918	1.4994
Function 3				
		alpha		
	db4	1.0771		
	bior3.5	0.8258		
	haar	1.5		
	sym4	1.1102		

The results in finding α with SWT are very similar to the ones with DWT. The main difference is the number of wavelet coefficients that we get per level is the same. Therefore we need to chose the cutoff boundary carefully. The difference in results is mainly in the values of alpha, DWT has given us the higher values of α generally, that is because the SWT gets wavelet coefficients for all values of the sample size on every level where as DWT has different size of coefficient matrix for every level.

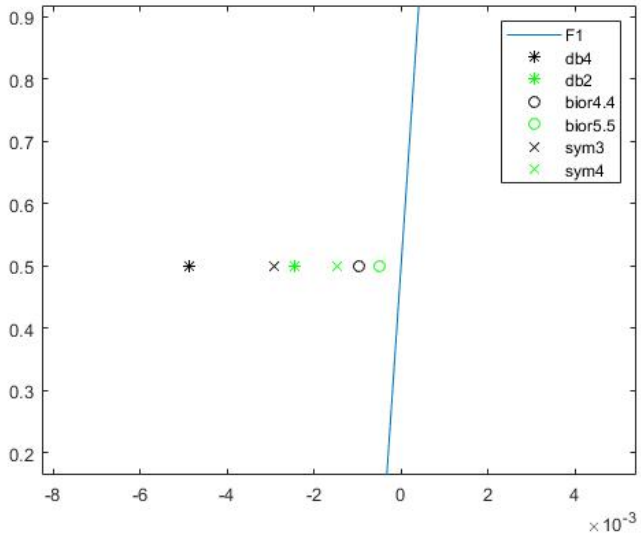

Figure 5: Position of singularity in SWT

In the above figure we see that the 'bior' family of wavelets is closest to the singularity while symlets and daubechies are a bit further. This is again due to the difference of shapes of wavelets as we had in DWT.We can again see the symmetry in biorthogonal family vs other wavelets used here.

9 Task 3.1

The images chosen here for analysis are the following

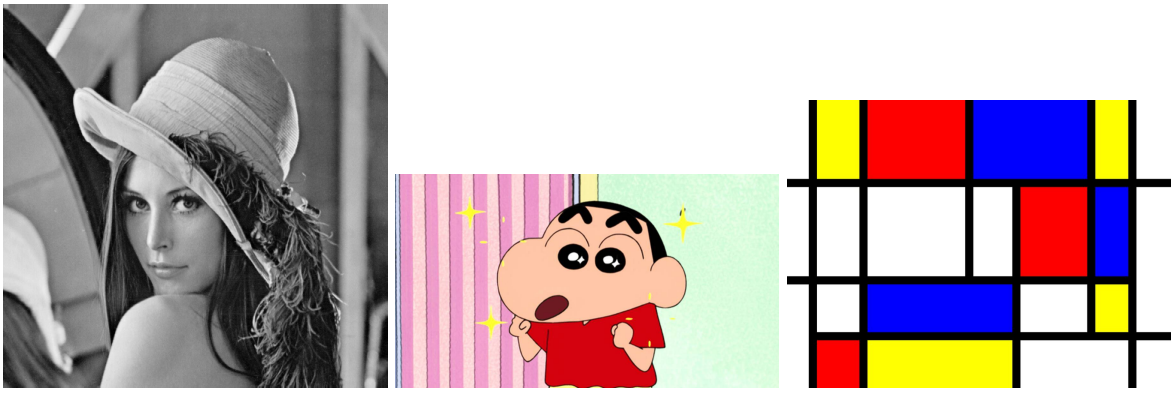

Figure 6: Images Used: Lenna, Shinchan, Mondriaan Painting

Pre-processing: The images were resized to 512 x 512. As the edge detection is based on knowing the location pixels, we could conveniently convert it into gray-scale images with little repercussions. Also, the edges of the

respective images were derived from the MATLAB inbuilt function 'edge' which uses Canny's edge detection algorithm. This image was considered to be the "standard" and our results were compared to this image.

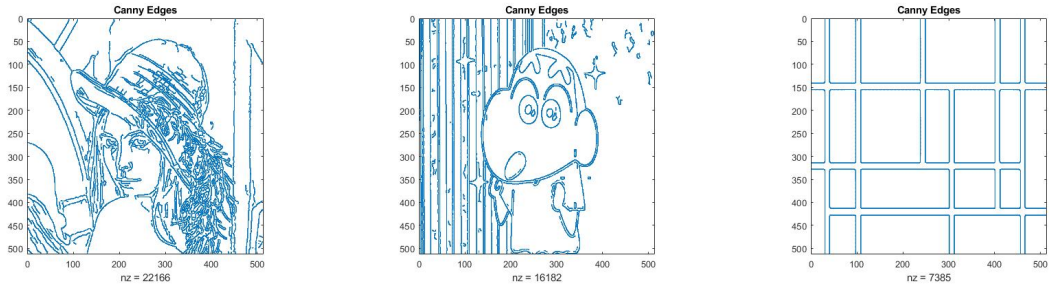

Figure 7: Edges using Canny's Algorithm

Experiments: Our experiment for each image consisted of three parts:

1. Varying the wavelet family

   We considered the wavelets: Daubechies-4, Biorthogonal-4.4, and Symlet-4 for our experiments. Since the edges are present in the levels containing "higher frequencies," we only considered level 1 coefficients. The threshold was set as 10 percent of the maximum value of the coefficients obtained. This threshold was decided by performing ablation studies and it worked well for all the three images.

2. Varying the scales of the wavelet coefficients

   The results of four consecutive scales of Daubechies-4 were compared. The threshold was set as previously described.

3. Varying the threshold for each wavelet family

   For each wavelet type, the threshold was varied from 0-100 percent of the maximum value of the level 1 coefficients in steps of 2 percent. The error was calculated as:

$$error = ||Standard - ObtainedResult||$$

Results: The results are discussed in the order of the experiments mentioned above.

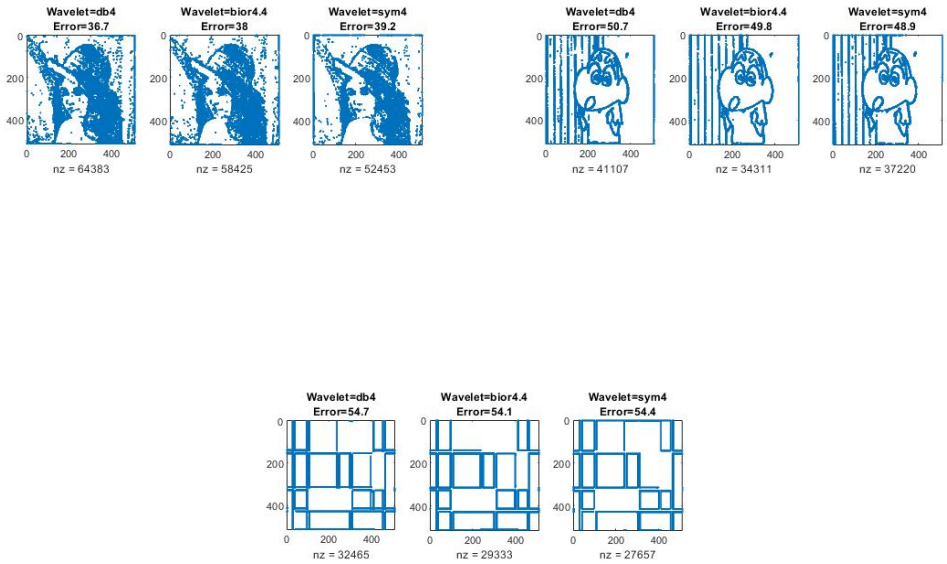

Figure 8: Variation in edge detection w.r.t wavelet type: DB4, Bior4.4, Sym4 for each image

1. Lenna's image contains more details when compared to the other two images. Therefore, its reconstruction is noisy in general. DB4 performed better than the other two here. For Shinchan, symlet-4 and for Mondriaan, it is biorthogonal-4.4. So, there is no clear consensus on the "better" wavelet here. Also, by observation, not much of a difference can be seen in the reconstruction of the last two images. The missing edges in the Mondriaan painting can be attributed to the fact that when it was converted to gray-scale, the intensities of red's and blue's were very similar to the black borders. This caused ambiguous edged there.

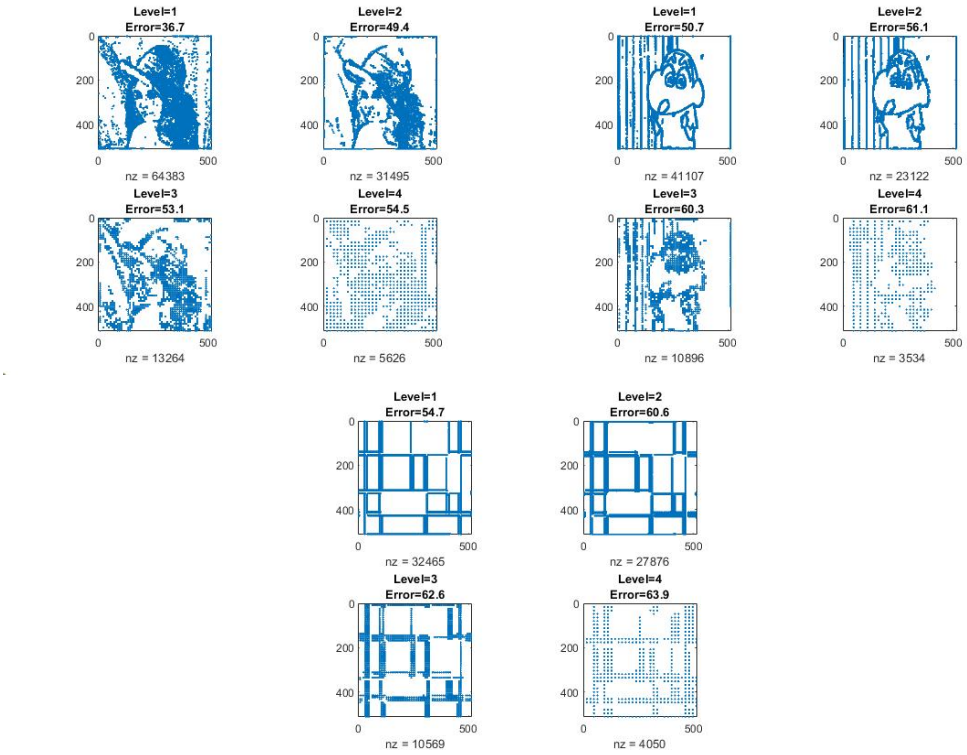

Figure 9: Variation in edge detection w.r.t. scales used

2. The quality of the edges deteriorates as we go down the levels. This is because of the fact that as we go down the levels, the frequency range decreases. The edges (discontinuities) are high frequencies and therefore, more visible in the levels with higher frequencies. Also, as we go down towards the coarser levels, downsampling occurs which makes it even more difficult to associate the coefficients to pixels in the original image. Most of the details in Lenna's and Shinchan's images has been lost in the last level. However, the Mondriaan painting has done well. This might be because the image was not very detailed and therefore can be approximated with lesser coefficients.

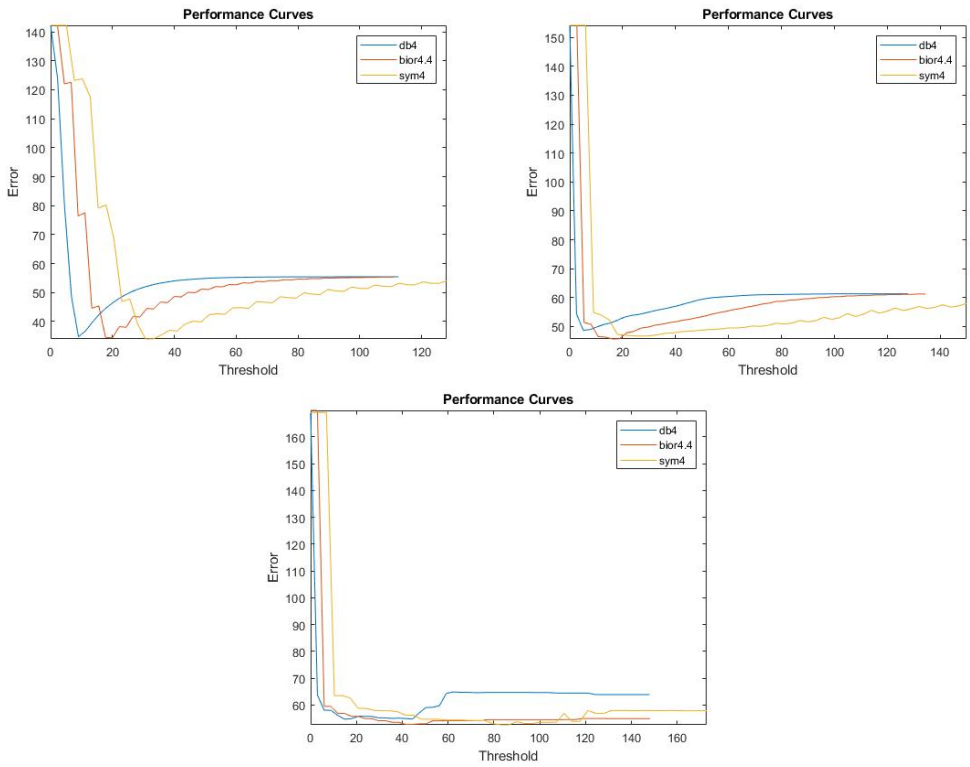

Figure 10: Error vs Threshold values plot for Lenna, Shinchan, and Mondriaan Painting, respectively

3. We expected that as the threshold increases, the edge detection deteriorates i.e. the error should increase. As the threshold increases, the error value decreases but it is not a true estimate of the quality of the image because of the data bias towards the black pixels. In all the cases, the DB-4 wavelet has a smoother curve and symlet-4 is the most susceptible to the oscillation at the end. However, when we try to reconstruct the images using the minimum threshold obtained here, they fail to provide good results because of the aforementioned data bias.

10 Question 3.2

In this case, we interpolated the images using the following logic:

$$Z(i,j) = M(i/c, j/c)$$

where

$$c = 2^k$$

where Z is the reconstructed image, M is the wavelet coefficient matrix of level k. As the matrix M is down-sampled by a factor of some integral power of 2, we associated each pixel as shown in the equation above. We did not opt to associate a coefficient to a range of pixel because it would make the final image "blocky". Then the jarring discontinuities were removed using a Gaussian filter which helps in smoothening the image.

11 Question 3.3

We need to construct a matrix M because we can have large coefficients in say the vertical direction to constitute the edge and not in the horizontal or diagonal direction. So , if we consider only one direction we will not be able to see the impact of wavelet coefficients of other directions and we will end up missing out on other edges. Hence it is important to construct the matrix M which considers all the directions.

12 Task 3.4

The same experiments with same settings have been performed for redundant wavelets too. Results:

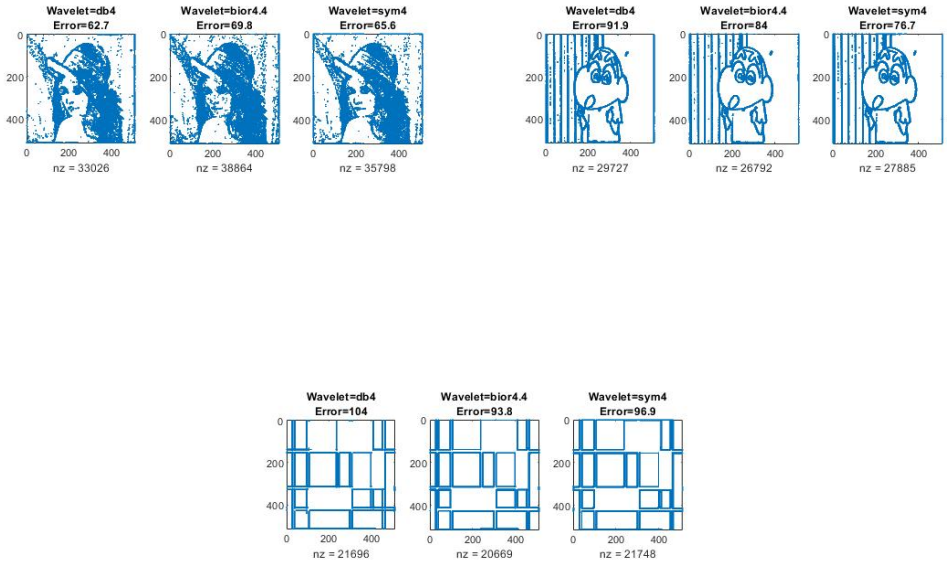

Figure 11: Variation in edge detection w.r.t wavelet type: DB4, Bior4.4, Sym4 for each image

1. The quality of the images is better than before. This may be attributed to the fact that the level 1 coefficients in this case are double than that of the normal wavelet transform.

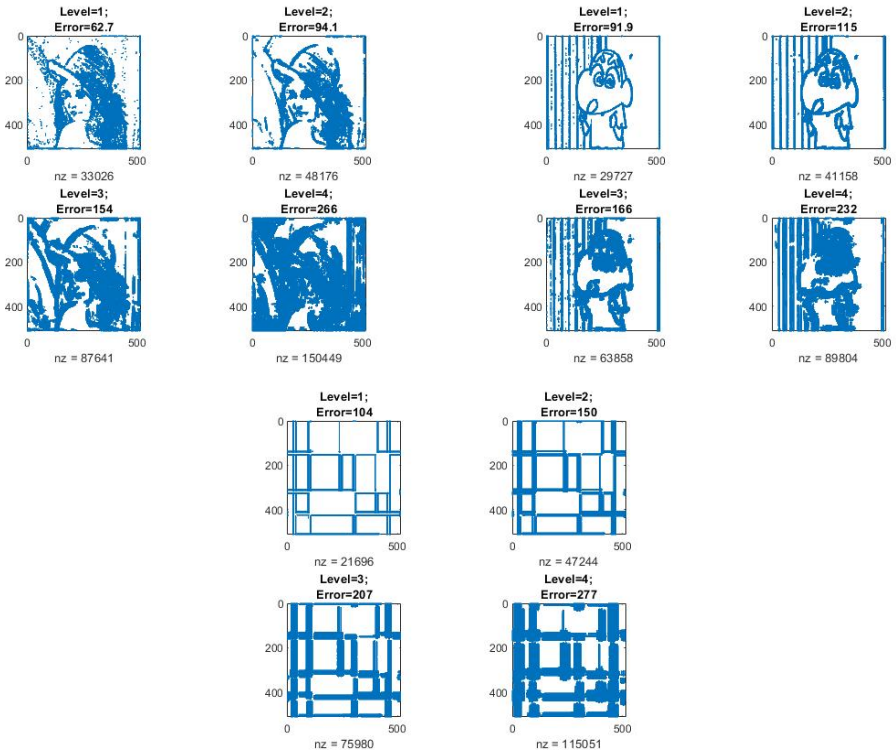

Figure 12: Variation in edge detection w.r.t. scales used

2. More details are retained in the successive coarser levels than in the previous case. As we move to coarser levels, the lines get thicker. This is because lower frequency scales retain a rough approximation. Redundant wavelets contain more information than the normal discrete wavelet transform.

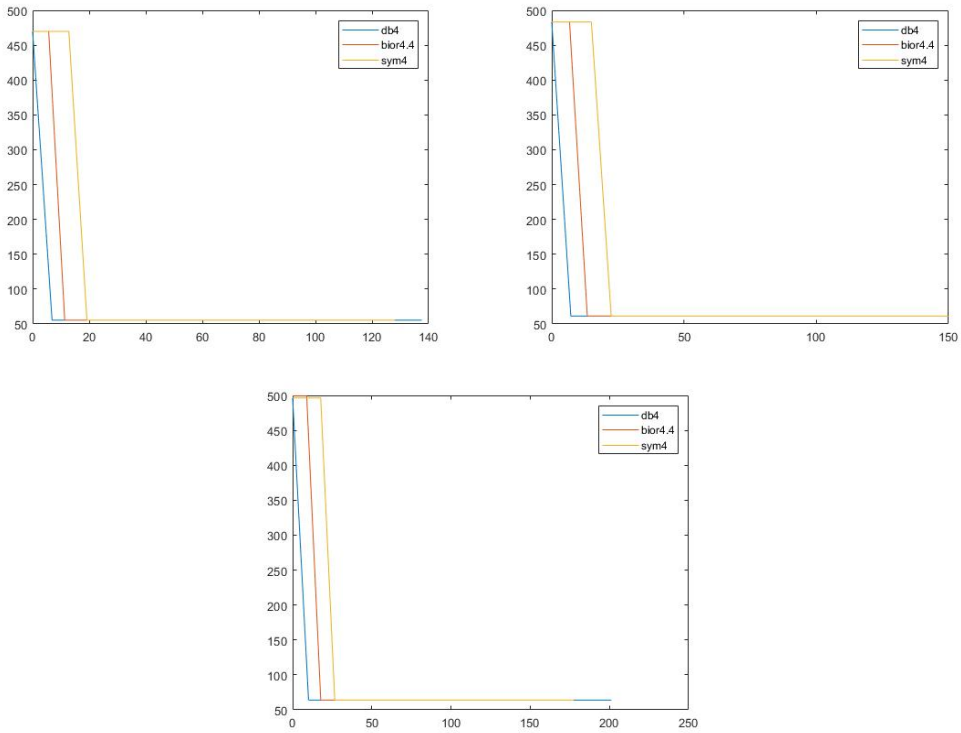

Figure 13: Error vs Threshold plots for Lenna, Shinchan, and Mondriaan Painting, respectively

3. The performance curves look like step functions here. Also, as with the previous case, it suffers from heavy data bias which makes our quantifier insensitive.

13 Question 3.4

In redundant wavelet transformation, the images are not downsampled. So, by this logic, the size of coefficient matrix of each level would be equal to the size of the image. Therefore, the coefficient matrix should have one-one correspondence with our image. So, we mapped them as:

$$Z(i, j) = M(i, j)$$

where Z is the required image (with edge locations) and M is the wavelet coefficient matrix.

14 Appendix

14.1 Code for Getting detailed coefficients and alpha values

Change the wavelet type on the following codes to get results for them. The following is for DWT function.

```
    %% Code for part 1

clc
clear all
dwtmode('per')

cut = [8,7,6,5,4,4,4,3,3,3]; % index for cutoff ranges

%% Function 1
% Define the function
n=(2^20);
t=linspace(-1,1,n);

  f1 = zeros(size(t));
  I = find(t >= 0);
  f1(I) = 1;

N=20;  % Number of levels
level=1:N;
[c,l]=wavedec(f1,N,'db4');
c=abs(c);
figure(1)
semilogy(abs(c)+1e-15) % adding small term as log(0) is infinity
% In different scales
D = detcoef(c,l,level);
 for i = 1:10
temp  = D{i};
temp = temp(2^cut(i):end-2^cut(i));
%temp = temp(floor(0.1*size) : end - floor(0.1*size));
%temp = temp(2:end-2); % removing boundaries
largecoeff(i) = max(temp);
 end

 %% Plot of the line
 figure(2)
 plot(log2(largecoeff))

 %% Calculation of alpha which is the slope of the line
 alpha1 = (log2(largecoeff(10)) - log2(largecoeff(6)))/10

 %% For the second function
 n1 = 2^20
 t=linspace(-1,1,n1);
```

```
 sigma = [0,1,2];
 f2 = (abs(t).^sigma(3)).*cos(2*t);

 N=20;  % Number of levels
level=1:N;
[c,l]=wavedec(f2,N,'db2');
%c=abs(c);
figure(1)
%semilogy(abs(c)+1e-15) % adding small term as log(0) is infinity
% In different scales
D = detcoef(c,l,level);
 for i = 1:10
temp  = D{i};
temp = temp(2^cut(i):end-2^cut(i));
%temp = temp(floor(0.1*size) : end - floor(0.1*size));
%temp = temp(2:end-2); % removing boundaries
largecoeff1(i) = max(temp);
 end

 %% Plot of the line
  figure(3)
 plot(log2(largecoeff1))

  %% Calculation of alpha which is the slope of the line
 alpha2 = (log2(largecoeff1(10)) - log2(largecoeff1(8)))/2


 %% For Function 3
  n2 = 2^20;
  t=linspace(-1,1,n2);

 f3 = test_f3(t);


  N=20;  % Number of levels
level=1:N;
[c,l]=wavedec(f3,N,'sym4');
c=abs(c);
figure(1)
semilogy(abs(c)+1e-15) % adding small term as log(0) is infinity
% In different scales
D = detcoef(c,l,level);
 for i = 1:10
```

```
temp  = D{i};

temp = temp(2^cut(i):end-2^cut(i));
largecoeff2(i) = max(temp);
 end

  %% Plot of the line
  figure(4)
 plot(log2(largecoeff2))

  %% Calculation of alpha which is the slope of the line
 alpha3 = (log2(largecoeff2(10)) - log2(largecoeff2(6)))/4
```

The following code is for the same application but for SWT

```
%% SWT for part 1

clc
clear all
dwtmode('per')

cut = [8,7,6,5,4,4,4,3,3,3];
%% Function 1
% Define the function
n=(2^20);
t=linspace(-1,1,n);

  f1 = zeros(size(t));
  I = find(t >= 0);
  f1(I) = 1;
  L = 20;
  s = swt(f1, L, 'db4');

  for i = 1:10
      temp = s(i,(2^cut(i) : end -2^cut(i)));
      temp = abs(temp);
        largecoeff(i) = max(temp);
  end

  %% plotting the largest coeff in log scale
  figure(1)
  semilogy(largecoeff)
```

```
 %% finding alpha
 alpha = (log2(largecoeff(10)) - log2(largecoeff(2)))/8

 %% Function 2

  n1 = 2^20
t=linspace(-1,1,n1);
sigma = [0,1,2];
f2 = (abs(t).^sigma(1)).*cos(2*t);
 L = 20;
s1 = swt(f2, L, 'haar');

 for i = 1:10
     temp = s1(i,(2^(cut(i)-1) : end -2^(cut(i)-1)));
     temp = abs(temp);
       largecoeff1(i) = max(temp);
 end
 log_coeff = log2(largecoeff1);
 %% plotting the largest coeff in log scale
 figure(2)
 %semilogy(largecoeff1)
 plot(log_coeff)
 %% finding alpha
 alpha1 = (log_coeff(10) - log_coeff(4))/6

 %% Function 3
   n2 = 2^20;
 t=linspace(-1,1,n2);

f3 = test_f3(t);

L = 20;
s2 = swt(f3, L, 'db4');

%%
   for i = 1:20
     temp = s2(i,(5*i : end -5*i));
     %temp = abs(temp);
     %temp = s2(i,:);
       largecoeff2(i) = max(temp);
 end
 log_coeff2 = log2(largecoeff2);
```

```
%% plotting the largest coeff in log scale
figure(3)
%semilogy(largecoeff1)
plot(log_coeff2)
%% finding alpha
alpha3 = (log_coeff2(20) - log_coeff2(5))/15
```

14.2 Code for getting the corresponding time points

Change the wavelet type on the following codes to get results for them. The following is for DWT function.

```
clear all
close all
clc

dwtmode('per')

%% Task 2.1

N=10;
n=2^N;

names={'db4','db2','bior4.4','bior5.5','sym3','sym4'};

for runs=1:2
    % Define the function

    t=linspace(-1,1,n);

    f1 = zeros(size(t));
    I = find(t >= 0);
    f1(I) = 1;

    for wav=1:length(names)

        % Wavelet Transform
        N=10;
        temp=string(names(wav));

        [c,l]=wavedec(f1,N,temp);
        level=1:N;
        D=detcoef(c,l,level);
```

```
        % Large coefficients
        if runs==1 && wav==1
            percent=0.1;
            for i=1:N
                seg=abs(D{i});

                remove=round(percent*length(seg));

                if length(seg)~=remove
                    seg=seg(remove+1:end-remove);
                end

                largecoef(i)=max(seg);
            end
        end
        [dud,compare]=alphaval(D);
        % Time at the largest coefficient
        tp(runs,wav)=support(D{1});
        n=n+1;
    end

end

% Plot
figure()
semilogy(largecoef)
xlabel('Level #')
ylabel('Coeddicient Amplitude in dB')
title('Large Coefficients in each Level of DB4')

% Table
timepoints=array2table(tp,'VariableNames',names,'RowNames',{'2^N','2^N+1'})

% Plot of timepoints
figure()
plot(t,f1)
hold on

ch={'k*','g*','kO','gO','kx','gx'};
for i=1:length(tp)
    temp=string(ch(i));
    plot(tp(i),0.5,temp)
    hold on
```

```
end
f={'F1'};
say=horzcat(f,names);
legend(say)
The following code is for the same application but for SWT:
clear all
close all
clc

dwtmode('per')

%% Task 2.1


L=[10,11];
for i=1:length(L)
    n(i)=2^L(i);
end

names={'db4','db2','bior4.4','bior5.5','sym3','sym4'};

for runs=1:length(n)
    % Define the function

    t=linspace(-1,1,n(runs));

    f1 = zeros(size(t));
    I = find(t >= 0);
    f1(I) = 1;

    for wav=1:length(names)

        % Wavelet Transform

        temp=string(names(wav));

        D=swt(f1,L(runs),temp);


        % Large coefficients
        if runs==1 && wav==1
            percent=0.1;
            for i=1:L(runs)
```

```
                seg=abs(D(i,:));

                remove=round(percent*length(seg));

                if length(seg)~=remove
                    seg=seg(remove+1:end-remove);
                end

                largecoef(i)=max(seg);
            end
        end

        % Time at the largest coefficient
        tp(runs,wav)=support(D(1,:));
    end
end

% Plot
figure()
semilogy(largecoef)
xlabel('Level #')
ylabel('Coeddicient Amplitude in dB')
title('Large Coefficients in each Level of DB4')

% Table
timepoints=array2table(tp,'VariableNames',names,'RowNames',string(n))

% Plot of timepoints
figure()
plot(t,f1)
hold on

ch={'k*','g*','kO','gO','kx','gx'};
for i=1:length(tp)
    temp=string(ch(i));
    plot(tp(i),0.5,temp)
    hold on
end
f={'F1'};
say=horzcat(f,names);
legend(say)
```

14.3 Code for Edge Detection

```
clc
clear all
close all

dwtmode('per')

% Load Data
T=imread("mondrian.jpg");
names={'db4','bior4.4','sym4'};

figure()
imshow(T)
say=sprintf('Original Image');
title(say)

if length(size(T))==3
    T=rgb2gray(T);
end

scale=2^9;
T=imresize(T,[scale,scale]);

best=edge(T,'canny');
best=double(best);

figure()
spy(best)
say=sprintf('Canny Edges');
title(say)

Threshold and Wavelets
for wav=1:length(names)


    here=string(names(wav));
    [CA1,CH1,CV1,CD1] = dwt2(T,here);
    absval=(CH1.^2+CV1.^2+CD1.^2).^0.5;
    M1{wav}=absval;
    percent=0:0.02:1;

    threshold(wav,:)=percent.*max(max(absval));
```

```
    for th=1:length(threshold)
        loc=find(absval<threshold(th));

        M=absval;
        M(loc)=0;

        [r,c]=size(T);
        Z=zeros(r,c);

        [r,c]=size(M);
        idx=2^1;            % Since level 1

        for k=1:r
            for j=1:c
                if M(k,j)~=0
                    Z(idx*k,idx*j)=1;
                else
                    Z(idx*k,idx*j)=0;
                end
            end
        end

        Z=imgaussfilt(Z);
        say=sprintf(here);

        error(wav,th)=norm(best-Z);
    end
end

% Plotting
[row,col]=size(error);

figure()


for i=1:row
    plot(threshold(i,:),error(i,:))
    hold on
end

xlabel('Threshold')
ylabel('Error')
```

```
axis tight;
legend((names))

title('Performance Curves')


Varying Wavelets
for wav=1:length(names)
    temp=M1{wav};
    % set a threshold
    percent=0.10;
    thresh=percent*max(max(temp));


    % remove points below threshold
    k=find(temp<thresh);
    temp(k)=0;

    [r,c]=size(T);
    Z=zeros(r,c);

    [r,c]=size(M);
    idx=2^1;            % Since level 1

    for k=1:r
        for j=1:c
            if temp(k,j)~=0
                Z(idx*k,idx*j)=1;
            else
                Z(idx*k,idx*j)=0;
            end
        end
    end

    Z=imgaussfilt(Z);

    dispw{wav}=Z;

    %calculate the error wrt canny edges
    [error_Mw(wav),sens(wav),spec(wav)]=performance(Z,best);

    % Plot
    subplot(1,length(names),wav)
```

```
    say=sprintf('Wavelet='+here+'\nError=%0.3g',error_Mw(wav));
    spy(temp)
    title(say)
end
figure()
montage(dispw,'size',[1,3],'BorderSize',50)

Scale
name='db4';         % Smooth curve, less oscillations, prediction is good
disp=struct([]);

[A,H,V,D] = dwt2(T,name);
coeff{1}=(H.^2+V.^2+D.^2).^0.5;

for level=1:3
    [t1,t2,t3,t4] = dwt2(A,name);
    coeff{level+1}=(t2.^2+t3.^2+t4.^2).^0.5;
    A=t1;
end
figure()
for i=1:length(coeff)
    absval=cell2mat(coeff(i));

    % Thresholding
    percent=0.1;
    t=percent*max(max(absval));
    k=find(absval<t);

    M=cell2mat(coeff(i));
    M(k)=0;

    % Reconstruction
    [r,c]=size(T);
    Z=zeros(r,c);

    [r,c]=size(M);
    idx=2^i;

    for k=1:r
        for j=1:c
            if M(k,j)~=0
                Z(idx*k,idx*j)=1;
            else
```

```
                Z(idx*k,idx*j)=0;
            end
        end
    end
    Z=imgaussfilt(Z);

    disp{i}=Z;
    error_c(i)=norm(Z-best);

    % Plot
    subplot(2,2,i)
    say=sprintf('Level=%d\nError=%0.3g',i,error_c(i));
    spy(Z)
    title(say)

end
figure()
montage(disp)
title('Varying Scale')
```

The following code is for the same application but for SWT:

```
clc
clear all
close all

dwtmode('per')

% Load Data
T=imread("mondrian.jpg");

names={'db4','bior4.4','sym4'};
M=struct([]);

Pre-processing Analysis


figure()
imshow(T)
say=sprintf('Original Image');
title(say)
```

```
if length(size(T))==3
    T=rgb2gray(T);
end

L=9;                            % Define no. of pixels as even number only
T=imresize(T,[2^L,2^L]);

best=double(edge(T,'canny'));   % Canny edges for comparison


figure()
spy(best)
say=sprintf('Canny Edges of Resized Image');
title(say)


Varying Wavelet
figure()
for wav=1:length(names)
    here=string(names(wav));
    [CA,CH,CV,CD] = swt2(T,L,here);
    temp=(CH(:,:,1).^2+CV(:,:,1).^2+CD(:,:,1).^2).^0.5;
    M{wav}=temp;
    % set a threshold
    percent=0.1;
    thresh=percent*max(max(temp));

    % remove points below threshold
    k=find(temp<thresh);
    temp(k)=0;

    % change the points to 1 which are above threshold
    w=find(temp);
    temp(w)=1;

    dispw{wav}=temp;

    %calculate the error wrt canny edges
    error_Mw(wav)=norm(best-temp);

    % Plot
    subplot(1,length(names),wav)
    say=sprintf('Wavelet='+here+'\nError=%0.3g',error_Mw(wav));
```

```
    spy(temp)
    title(say)
end
figure()
montage(dispw,'size',[1,3],'BorderSize',50)


Threshold Varying for Each Wavelet

for wav=1:length(names)
    temp=M{wav};

    percent=0:0.05:1;
    threshold(wav,:)=percent.*max(max(temp));

    for th=1:length(threshold)
        % Make the coefficients zero
        k=find(temp<threshold(th));
        temp(k)=0;

        % Find non-zero elements ans assign it 1
        w=find(temp);
        temp(w)=1;

        error_th(wav,th)=norm(best-temp);
    end
end

figure()
for i=1:length(names)
    plot(threshold(i,:),error_th(i,:))
    hold on
end
legend(names)

Varying scale

[CA,CH,CV,CD] = swt2(T,L,'db4');         % returns the values for L levels for db4

figure()
for level=1:L           % Calculate M (mod of hori, vert, diag components)
    temp=(CH(:,:,level).^2+CV(:,:,level).^2+CD(:,:,level).^2).^0.5;
```

```
    percent=0.1;
    thresh=percent*max(max(temp));

    k=find(temp<thresh);
    temp(k)=0;

    w=find(temp);
    temp(w)=1;

    if level<5
    disps{level}=temp;
    end

    error_M(level)=norm(best-temp);

    % Plot
    if level<5
    subplot(2,2,level)
    spy(temp)
    say=sprintf('Level=%d;\nError=%0.3g',level,error_M(level));
    title(say)
    end
end

figure()
montage(disps,'size',[2 NaN],'BorderSize',[50,50]);
```