Objectives

- Understand the Wavelet transform.
- Familiarise with the decomposition process in 1D and 2D.
- Reconstruct a signal from its Wavelet coefficients.

1. 1D Wavelet Transform.

You should know from the lectures that wavelets are wave functions with compact spatial support which satisfy certain mathematical conditions, most important of which being their dilations and translations form basis functions for a signal. Obtaining wavelets is a simple process, you need to follow the following steps:

1. Take a wavelet and compare it to a section at the start of the original signal.
2. Calculate a number, C, that represents how closely correlated the wavelet is with this section of the signal. The higher C is, the more the similarity. More precisely, if the signal energy and the wavelet energy are equal to one, C may be interpreted as a correlation coefficient. Note that the results will depend on the shape of the wavelet you choose.
3. Shift the wavelet to the right and repeat steps 1 and 2 until you’ve covered the whole signal.
4. Scale (stretch) the wavelet and repeat steps 1 through 3.
5. Repeat steps 1 through 4 for all scales.

When you’re done, you’ll have the coefficients produced at different scales by different sections of the signal. The coefficients constitute the results of a regression of the original signal performed on the wavelets. This process generates a Continuous Wavelet Transforms. For a discrete wavelet, you repeat steps 1 to 3 for only two wavelets, one of which is a low-pass signal and the other should be high pass. The simplest wavelet that you can use is the Haar basis.

Let’s work a decomposition in matlab. First we need a signal to decompose, and the low pass and high pass wavelets, we will use the first 256 values of the Hallelujah:

```matlab
load handel
x=y(1:256);
lpf=[1 1]/sqrt(2);
hpf=[1 -1]/sqrt(2);
hpfi=[-1 1]/sqrt(2);
```

The filters are normalised with the factor $\sqrt{2}$. To perform the comparison a convolution can be done with the function `filter`:

```matlab
x_lp=filter(lpf,1,x);
x_hp=filter(hpf,1,x);
```
To visualise them:

```matlab
plot(x)
plot(x_hp)
plot(x_lp)
```

Notice that the filtered signals have the same dimensions as the original one, so they have to be downsampled (can you remember how to do this easily in Matlab?). Try:

```matlab
L1=x_lp(2:2:end);
H1=x_hp(2:2:end);
```

Now that the dimensions have been reduced, we can have the first level decomposition completed:

```matlab
DWT1=[L1 ;H1];
```

You can have as many levels as you want, actually as many as the points of your signal allow you. To construct the lower levels you just have to filter the low pass region in the same way as you did for the first decomposition.

```matlab
x_lp2=filter(lpf,1,L1);
x_hp2=filter(hpf,1,L1);
L2=x_lp2(2:2:end);
H2=x_hp2(2:2:end);
DWT2=[L2 ;H2; H1];
```

1. Write the decomposition process into a function, that receives a signal, the filters and returns the decomposition.

2. Use the previous function recursively to generate a multilevel decomposition in which you provide the same data, as well as the level to decompose. You need to check the size of the function to be decomposed.

To reconstruct the signal, you have to upsample the decomposed coefficients:

```matlab
H1u=zeros(2*size(H1,1),1);
H1u(1:2:end)=H1;
```
L1u=zeros(2*size(L1,1),1);
L1u(1:2:end)=L1;

Then filter them back:

H1uf=filter(hpfi,1,H1u);
L1uf=filter(lpf,1,L1u);

And finally just sum the two components together:

xr=L1uf+H1uf;

Original and reconstructed signals

Since images have one dimension more than the 1D signals, filtering is to be done in each of the
directions, namely rows and columns.

3. Write a function in which you perform the filtering, low or high pass to each row or column of a
2D signal. You should be able to use the functions previously written. Once you have these
filtering functions you can write easily a function that performs a 2D decomposition.

4. Read the Lenna image available at:

/dcs/acad/nasir/cs403/lenna.raw

using the following set of commands:

```matlab
fname='dcs/acad/nasir/cs403/lenna.raw';
fid=fopen(fname,'rb');
x=fread(fid,'uint8');
```

Make sure that the image mean is zero by executing the following command:

```matlab
x = x-mean(mean(x));
```

Since the wavelet transform preserves both mean and variance, executing the above command ensures that the magnitude of wavelet coefficients is more meaningful in that higher magnitude coefficients are more significant than the lower magnitude ones.

Now compute the 3–level wavelet transform of this image into \( x_w \) using the routine developed in Q3. Choose a threshold value equal to the half of maximum amplitude of wavelet coefficients. Use hard thresholding to set to zero all the wavelet coefficients whose magnitude is below this threshold value.

Divide the dynamic range of significant coefficients (ones with magnitude greater than the threshold) into 32 bins by using the following command:

```matlab
minData= min(min(xw_thresholded));
xw_thresholded=minData+xw_thresholded;
xw_thresholded=xw_thresholded/(max(max(xw_thresholded)));
xw_thres_quantised=round(xw_thresholded*31);
```

Compute the entropy of thresholded–and–quantized coefficients given by \( xw_{thres\_quantised} \).