Imperial College London

EE4-45 Wavelets and Applications

MATLAB Mini-Project

Coursework Report

By:
Ming Jin Siew
CID: mjs311
CID Number: 00684218

March, 2015

Exercise 1

The Daubechies scaling function. Which Daubechies scaling function should be used in order to reproduce polynomials of maximum degree 3? Compute the coefficients to reproduce polynomials of degree 0 to 3 for $n = 0, \dots, 32 - L$ where L is the support of the kernel. Plot the results with the shifted kernels and the reproduced polynomials as in Figure 5.2 of the lecture notes.

To reproduce polynomials of maximum degree N, we need to choose a scaling function which generate wavelets with $N + 1$ vanishing moments. Hence, to reproduce polynomials of maximum degree 3, a Daubechies scaling function with 4 vanishing moments will be chosen. In MATLAB, the scaling function can be obtained by using the following function and inputs: **wavefun('db4')**, where 'db4' represents a Daubechies scaling function with 4 vanishing moments.

In order to find the coefficients to reproduce polynomials of degree 0 to 3, we start by introducing the equality below:

$$\sum_{n\in\mathbb{Z}} C_{m,n}\varphi(t-n) = t^m \qquad m = 0, 1, \dots, N \tag{1}$$

where $\varphi(t - n)$ is the sampling kernel, $C_{m,n}$ is the set of coefficients, t is the time componenet, m is the order, and n is the length of the sample. To find $C_{m,n}$, we apply the dual-basis $\tilde{\varphi}(t - n)$ on both sides of Eq (1) to arrive at $C_{m,n} = \langle t^m, \tilde{\varphi}(t - n)\rangle$.

Hence, the next task is to find $\tilde{\varphi}(t - n)$, and this can be done by using the Daubechies formula for spectral factorisation, given below in general form:

$$P(\omega) = 2\left(\frac{1+\cos\omega}{2}\right)^p \sum_{k=0}^{p-1}\binom{p+k-1}{k}\left(\frac{1-\cos\omega}{2}\right)^k \tag{2}$$

where ω is frequency, p is the number of zeros at π, or vanishing moments. Hence, we choose $p = 4$ to arrive at

$$P(\omega) = 2\left(\frac{1+\cos\omega}{2}\right)^4 \left[\sum_{k=0}^{3}\binom{3+k}{k}\left(\frac{1-\cos\omega}{2}\right)^k\right]$$
$$P(\omega) = 2\left(\frac{1+\cos\omega}{2}\right)^4 \left[1 + 4\left(\frac{1-\cos\omega}{2}\right) + 10\left(\frac{1-\cos\omega}{2}\right)^2 + 20\left(\frac{1-\cos\omega}{2}\right)^3\right]$$

Replacing $\cos\omega = \frac{e^{j\omega}+e^{-j\omega}}{2} = \frac{z+z^{-1}}{2}$ to express in terms of z

$$P(z) = 2\left(\frac{2+z+z^{-1}}{4}\right)^4 \left[1 + 4\left(\frac{2-z-z^{-1}}{4}\right) + 10\left(\frac{2-z-z^{-1}}{4}\right)^2 + 20\left(\frac{2-z-z^{-1}}{4}\right)^3\right]$$
$$P(z) = \frac{2}{4^4 \times 16}\left(2+z+z^{-1}\right)^4 \left[16 + 16(2-z-z^{-1}) + 10\left(2-z-z^{-1}\right)^2 + 5\left(2-z-z^{-1}\right)^3\right]$$

Before proceeding, $2+z+z^{-1}$ and $2-z-z^{-1}$ can be factorised and their results substituted into equation above:

$$2+z+z^{-1} = (z+2+z^{-1}) \times \frac{z}{z} = (z^2+2z+1) \times \frac{1}{z} = (z+1)(z+1) \times \frac{1}{z} = (z+1)(z^{-1}+1)$$

and

$$2-z-z^{-1} = (-z+2-z^{-1}) \times \frac{z}{z} = (-z^2+2z-1) \times \frac{1}{z} = -(z^2-2z+1) \times \frac{1}{z} = -(z-1)(z-1) \times \frac{1}{z}$$
$$= -(z-1)(1-z^{-1}) = (z-1)(z^{-1}-1)$$

$$P(z) = \frac{1}{2048}(z+1)^4(z^{-1}+1)^4(16 + 16(z-1)(z^{-1}-1) + 10[(z-1)(z^{-1}-1)]^2 + 5[(z-1)(z^{-1}-1)]^3) \tag{3}$$

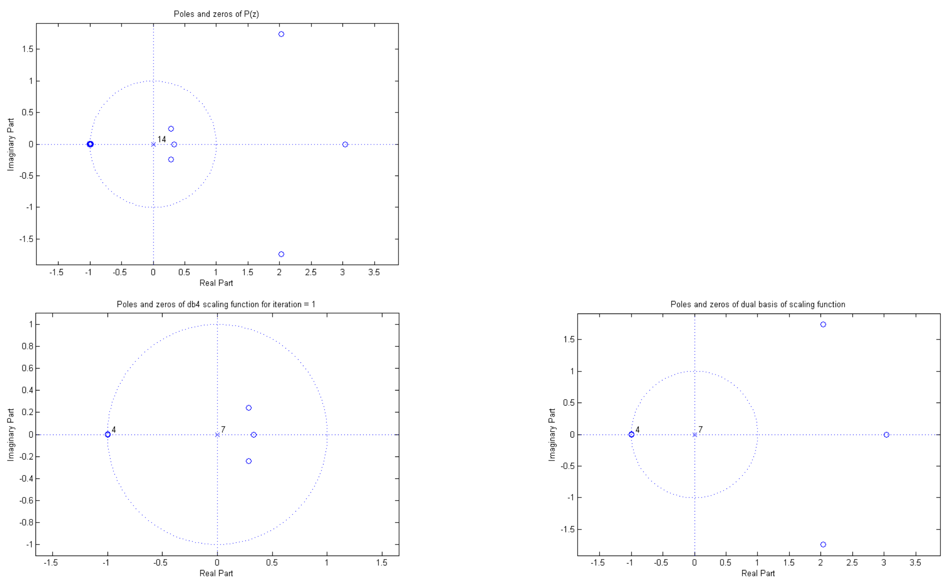

Figure 1: Poles and zeros of $P(z), \varphi(t)\text{and}\tilde{\varphi(t)}$

If we denote $G_0(z)$ as the scaling function as well as the low-pass filter, we require the following to be true to satisfy condition for orthogonality and halfband property:

$$\langle g_0[n], g_0[n-2k]\rangle = \delta_k \leftrightarrow G_0(z)G_0(z^{-1}) + G_0(-z)G_0(-z^{-1}) = 2 \tag{4}$$

Alternatively, using $P(z) = G_0(z)G_0(z^{-1})$, we require $P(z) + P(-z) = 2$ to be true. Had we assgin $z = z^{-1}$, then conditions set out above will still be satifisied, implying that if z_k is a zero of $P(z)$, then $1/z_k$ is also a zero of $P(z)$. This is shown to be true in Figure 1: a db4 scaling function has three zeros inside the unit circle, which corresponds to the three zeros outside the unit circle (Note that the reciprocal of a complex number within the unit circle will produce a complex number outside of the unit circle). Furthermore, the db4 Daubechies scaling function has 4 zeros at $\omega = \pi$ and 3 zeros inside the unit circle and 7 poles at the origin.

Using Equation (3), we can assign the dual-basis if we assign expression with z to $G_0(z)$ and z^{-1} to its dual-basis. However, given that it is possible to generate a Daubechies scaling function with 4 vanishing moments, i.e. $G_0(z)$, we can instead find $H_0(z) = P(z)/G_0(z)$ if we assign the scaling function generated in one iteration as $G_0(z)$. To start with, Eq (3) is expressed in the following form, to ease usage in MATLAB:

$$P(z) = \frac{1}{2048}\left(-5z^7 + 49z^5 - 245z^3 + 1225z + 1 + 1225z^{-1} - 245z^{-3} + 49z^{-5} - 5z^{-7}\right)$$

Hence, the dual-basis, $\varphi(t)$, is $H_0(z)$, in time domain. The dual-basis obtained has lower resolution as compared to what is required in this project, T = 64, an input signal has to be iterated 6 times. Hence, this calls for the use of a tree-structured filter bank. We can then find the equivalent filter using the following formula

$$G_0^J(z) = G_0^{J-1}(z)G_0(z^{2^{J-1}}) = \prod_{k=0}^{J-1} G_0(z^{2^k}) \tag{5}$$

Figure 2 shows the dual-basis on each iteration from $J = 1, \ldots, 6$, while Figure 3 compares the scaling function generated using **wavefun('db4', 6)** with the dual-basis generated as described above

Hence, by finding the inner product of t^m and $\tilde{\varphi}(t-n)$, $C_{m,n}$ is generated. Using the coefficients, shifted kernels for the degree of 0 to 3 is plotted for $n = 0, \ldots, 26$ using black dashed lines. The reproduced polynomials is super-imposed on the same plots using red solid lines. These are found in Figure 4

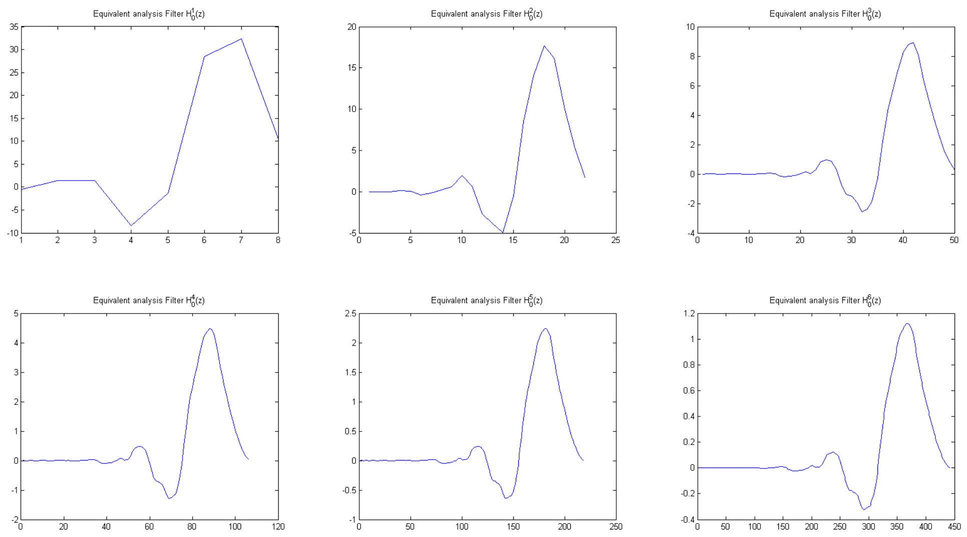

Figure 2: Dual-basis at each iteration from J = 1 to 3 (top three) and 4 to 6 (bottom three)

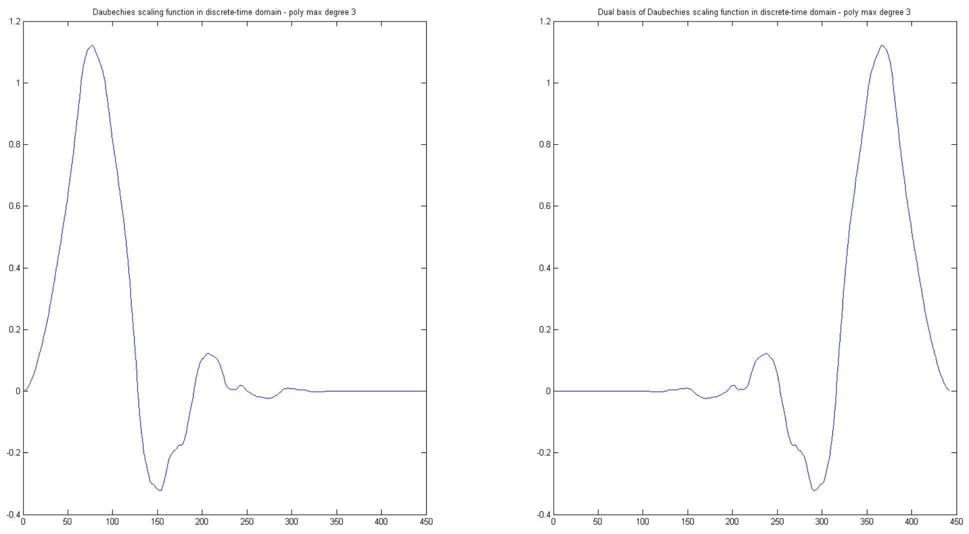

Figure 3: Daubechies scaling function with 4 vanishing moments (left) and its dual-basis (right)

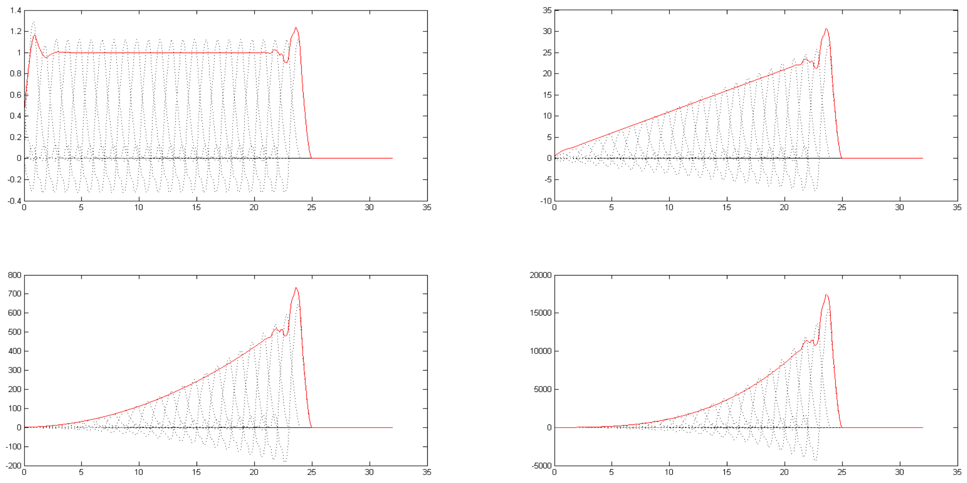

Figure 4: Reproduction of polynomial of maximum degree three using db4 scaling function. Dashed black lines are the shifted kernels. Solid red lines are the reproduced polynomials

Exercise 2

(optional) ***The B-Spline scaling function. Which B-Spline scaling function should be used in order to reproduce polynomials of maximum degree 3? Find the dual-basis of the scaling function and compute the coefficients to reproduce polynomials of degree 0 to 3 for $n = 0, . . . , 32 - L$ where L is the support of the kernel. Plot the results with the shifted kernels and the reproduced polynomials as in the previous exercise.***

To reproduce polynomials of maximum degree 3, a B-Spline scaling function with 4 vanishing moments in the synthesis wavelet and, similarly, 4 vanishing moments in the analysis wavelet as dual-basis. Although it is possible, in this exercise to find the dual-basis of the B-Spline using the same method employed in Exercise 1, I used the following line in MATLAB to obtain the dual-basis, that is stored in the variable ‘phi_T_tilde’:

[phi_T, ~, phi_T_tilde, ~] = wavefun(’bior4.4’, 6)

Figure 5 shows the appropriate B-Spline scaling function with its corresponding dual-basis. Figure 6 shows the results of shifted kernels and the reproduced polynomials for degree 0 to 3

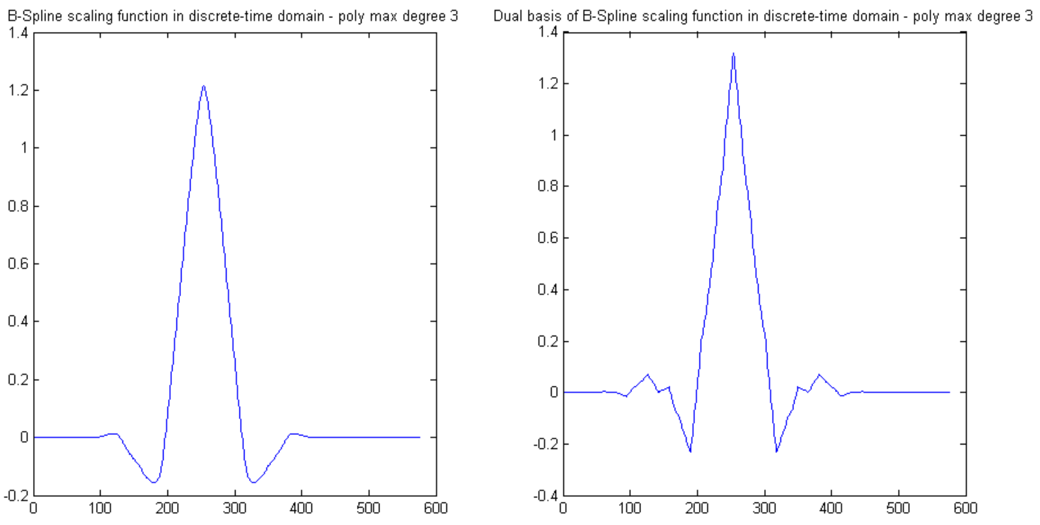

Figure 5: B-Spline scaling function with 4 vanishing moments (left) and its dual-basis (right)

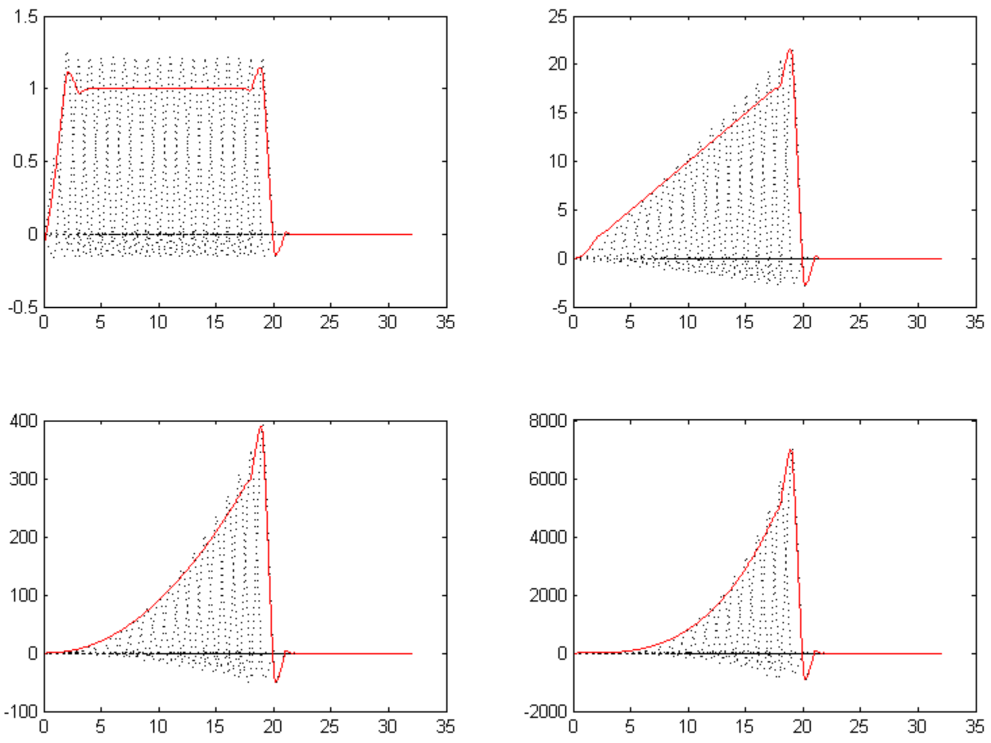

Figure 6: Reproduction of polynomial of maximum degree three using ‘bior4.4’ scaling function. Dashed black lines are the shifted kernels. Solid red lines are the reproduced polynomials

Exercise 3

Write a program that finds the annihilating filter $h[m]$ given a signal $\tau[m]$. Apply your program to the signal $\tau[m]$ available on the course website (m-file: tau.mat). Retrieve the locations t_k and the amplitudes a_k. Remark: we have set $K = 2$.

Assumption: Length of signal = 2048, T= 64

Results:

Dirac	**Location**	**Amplitude**
1	14.25	1.32
2	15.375	0.78

Table 1: Locations t_k and their corresponding amplitudes a_k

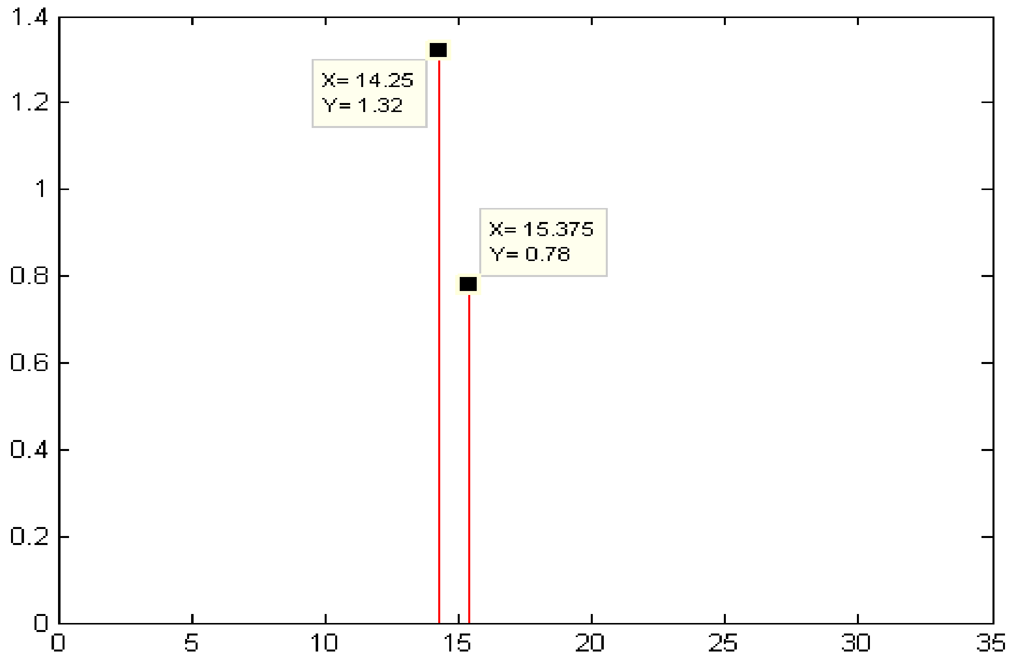

Figure 7: Retrieved locations and their corresponding amplitudes

Figure 7 is a plot of the amplitudes and locations recovered. To prove that the location and amplitude retrieved is accurate, a few steps are done to calculate $\tau[m]$ from the retrieved locations and amplitudes using the formula $\tau[m] = \sum_{k=0}^{K-1} a_k t_k^m$. The code snippet to do exactly this is as below:

```
for m = 0:3
    tau_est(m+1, 1) = amp.'*loc_t.^m;
end

same = tau - tau_est < 1e-9;
if prod(same)
    disp('Location retrieved is accurate')
else
    disp('Location retrieved is NOT accurate');
end
```

where 'amp' and 'loc_t' are the recovered amplitudes and locations, respectively.

Exercise 4

Create a stream of Diracs where $K = 2$. Sample the signal with an appropriate Daubechies scaling function using a sampling period of $T = 64$. Reconstruct the continuous time Diracs from the samples.

The position of diracs chosen, in time domain, is $t_0 = 8.1563$ and $t_1 = 22.0469$. They correspond to amplitudes $a_0 = 1$ and $a_1 = 3$.

The appropriate Daubechies scaling function must be able to reproduce polynomials of maximum degree $N \geq 2K - 1 = 3$. Therefore, assuming N = 3, there needs to be at least 4 vanishing moments, and that is achievable using a 'db4' Daubechies scaling function. This is used to sample the signal $x(t)$. Since this was the same scaling function used in Exercise 1, the coeffients generated from that exercise is reused here.

Results:
The exact locations and corresponding amplitudes are successfully retrieved. Figure 8 shows the results of this exercise.

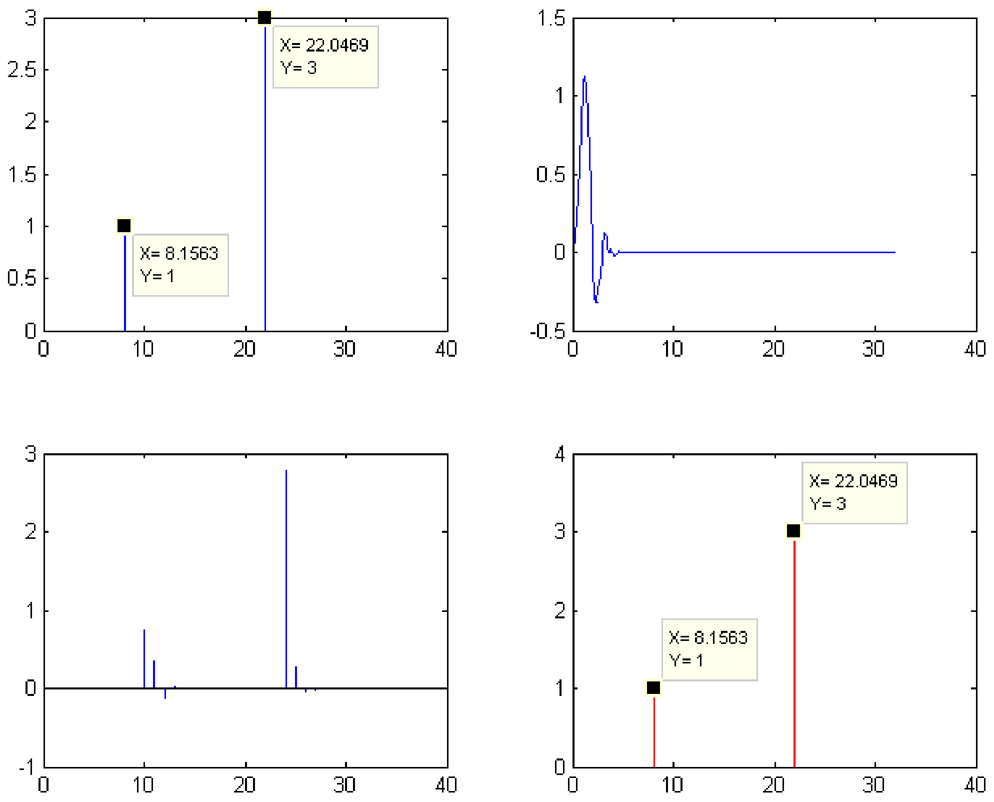

Figure 8: ***Top left:*** The known locations and amplitudes of the stream of dirac is plotted. ***Top right:*** 'db4' Daubechies scaling function that is used to sample $x(t)$. ***Bottom left:*** Samples as a result of sampling the stream of diracs using the scaling function. ***Bottom right:*** Recovered locations and corresponding amplitudes illustrated using a red line

Exercise 5

We have sampled a stream of Diracs ($K = 2$, $T = 64$) with a db4 Daubechies scaling function. The observed samples are available on the course website (m-file: samples.mat). Use the reconstruction algorithm to exactly recover the locations and the amplitudes of all the Diracs.

The locations obtained do not correspond to a location that would have been obtained using a time resolution of 1/64. In other words, the location obtained t_k when multipled by T does not produce an exact integer. However, the results accurate to 4 decimal places are available in the table below. The illustration of the samples and the diracs retrieved are available in Figure 9
Results:

Dirac	Location	Amplitude
1	12.3986	2.6879
2	15.5520	1.4221

Table 2: Locations t_k and their corresponding amplitudes a_k

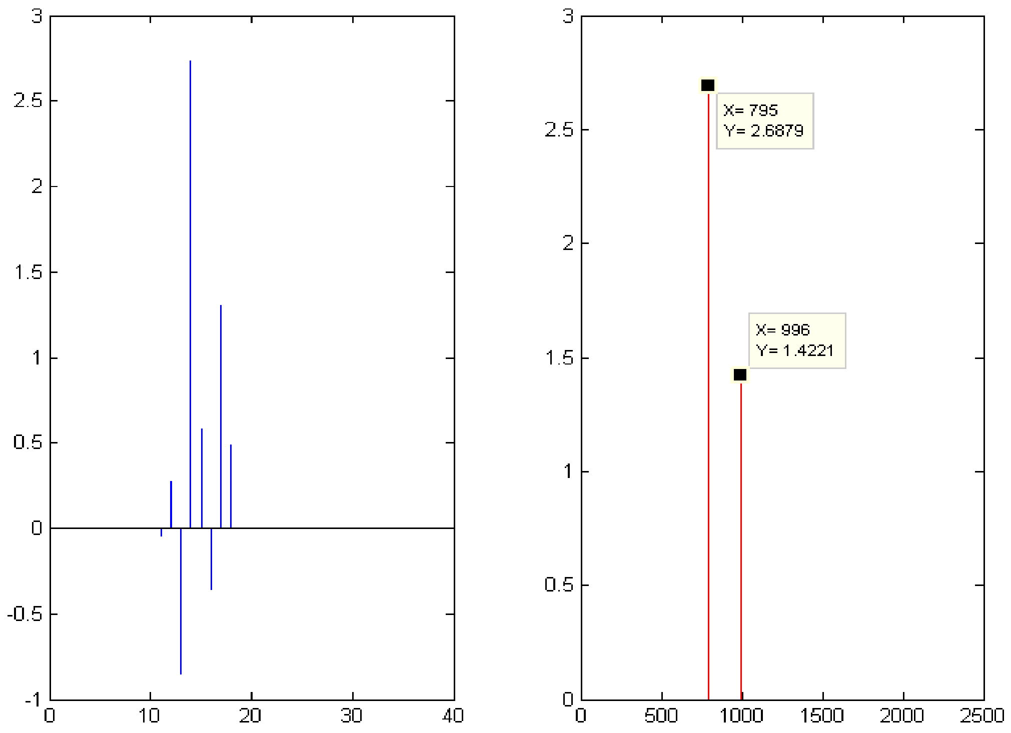

Figure 9: ***Left:*** Observed samples, samples.mat ***Right:*** 2 Diracs retrieved with X component being the index within a 2048-long signal

Exercise 6

In order to check the stability of the algorithm, create a stream of Diracs where $K = 1, \ldots, 4$. For each case, sample the signal with an appropriate Daubechies scaling function and check if you can always reconstruct the Diracs. Comment on the results.

Results:

The input, which are the stream of diracs, and the output results can be found in the table below. The illustration of results are attached in Appendix B.

K	1	*Input*	Amplitude **Location**	1 **11.703**			
		Output	*Accurate. Same as input*				
	2	*Input*	Amplitude **Location**	1 **11.703**	2 **16.3872**		
		Output	*Accurate. Same as input*				
	3	*Input*	Amplitude **Location**	1 **11.703**	2 **16.3872**	3 **17.1687**	
		Output	*Accurate. Same as input*				
	4	*Input*	Amplitude **Location**	1 **11.7031**	2 **16.3872**	3 **17.1687**	4 **19.5155**
		Output	Amplitude **Location**	1 **11.703**	1.9806 **16.3872**	3.0188 **17.1687**	4.0006 **19.5155**

Table 3: Amplitudes and locations of diracs for cases of $K = 1$ to 4

Comments:
The locations used in the Table 3 are expressed in the time domain. It is helpful to explain these cases in discrete indices, which are 750, 1050, 1100, 1250 (11.7031, 16.3872, 17.1687, 19.5155). Note that index 1 in MATLAB correspond to $t = 0$ in time domain, i.e, time begins with 0 in time domain followed by 1/64. On each cases, the Daubechies scaling functions used are db3, db4, db6 and db8. These are easily calculated using the condition in which the order to reproduce maximum degree of polynomials is $N \geq 2K - 1$. Following this, the number of vanishing moments needed is N+1.

There are two types of areas which can be addressed in this exercise: the conditions on the position of a dirac which ensures precise retrieval of amplitude and location, and why the case in which K = 4 is always resulting in error.

In observation, cases of K = 1, 2 and 3 are accurate to numerical precision for both amplitudes and locations. For the case of K = 4, the locations are correct but the amplitudes are wrong. Many different variation of position of 4 diracs are tried and tested and they usually result in errors, which is explained in the next paragraph. The first three cases work accurately because there are sufficient spaces between the beginning or the end of the stream of diracs to accomodate the support length of a sampling kernel. Here are the four different length of support for db3, db4, db6 and db8 Daubechies scaling function, respectively: 321, 449, 705, 961. As it is evident, only the case of K = 4 that this condition is not adhered. However, it was also shown that even if all the postions of the diracs allow enough space for a sampling kernel at the beginning and end of a stream in case of K = 4, i.e. positions of diracs are 1000, 1020, 1040, 1060, the amplitudes retrieved are still erroneous, although locations are retrieved correctly.

It seemed like even if I had chosen a Daubechies scaling function that is suitable for the case of K = 4 and if I had also allowed enough space on each ends of the stream of diracs to accomodate a db8 Daubechies scaling function, the result is still errorneous.

Dirac	**Location**	**Amplitude**
1	23.4219	1
2	31.2344	2
3	39.0469	3
4	46.8594	4

Table 4: Successful retrieval of locations t_k and their corresponding amplitudes a_k for the case K= 4 when length of sample is 4096. T = 64

A possible explanation for the consistently erroneous results is the size of the stream of diracs is smaller than KL. Since the length of db8 is 961, at least a length of 3844 is needed for $K = 4$. To test this explanation, I increased the size of the length of the signal to 4096 which results in successful retrievals of the four diracs. The results are shown in Table 4.

Exercise 7

Set A = 2.345, t0 = 8.8125 and t1 = 21.1875. Sample x(t) using $\varphi(t) = \beta_0(t)$ as a sampling kernel. Apply the finite differences to the samples and use your program to recover the locations and amplitudes of the equivalent signal. Recover the original x(t). Plot all the results.

Below are the B-spline scaling functions used in this exercise, order 0 and 1. $\beta_0(t)$ is used as the sampling kernel to sample $x(t)$ to produce $y[n]$. On the other hand, $\beta_1(t)$ is used to sample dx/dt to obtain a sampled $z[n]$. In my MATLAB code, $y[n]$ is only used for the purpose of illustration. The calculation of $z[n]$ is done through finding the inner product of dx/dt and $\varphi_{equ}(t) = \beta_1(t)$. The formula used for this exercise is as below:

$$z[n] = \langle \frac{dx(t)}{dt}, \varphi(t-n) * \beta_0(t-n) \rangle \tag{6}$$

where $x(t)$ is the piecewise constant function, $\varphi(t-n)$ is the sampling kernel and $\beta_0(t-n)$ is 1st order B-Spline scaling function. In this exercise, we take $\varphi(t-n)$ as 0th order B-Spline function, $\beta_0(t-n)$. Below are the sampling kernels used in this exercise:

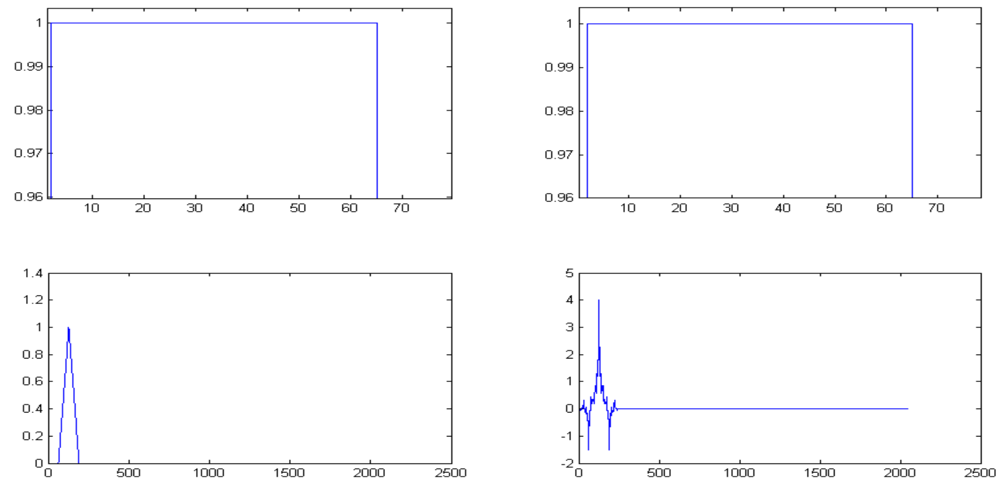

Figure 10: ***Top left:*** 0th order B-spline scaling function ***Top right:*** Dual-basis of the 0th order B-spline function ***Bottom left:*** 1st order B-spline scaling function ***Bottom right:*** Dual-basis 1st order B-spline function

The explanation for which it is possble to retrieve the two diracs individually is because the number of 'true' zero samples in between the two diracs is larger than the length of the sampling kernel, L = 320. This is easily observed from $(21.1875 - 8.8125) \times 64 = 12.375 \times 64 = 792 > 320$. Hence, none of amplitude in the sample is contributed by both diracs, which makes separate retrival possible.
Results:

Below are the results of the separate retrival of the two diracs which make up the piecewise constant function. They are both accurate in terms of locations and amplitudes, i.e. $t_0 = 8.8125, t_1 = 21.1875, A = 2.345$

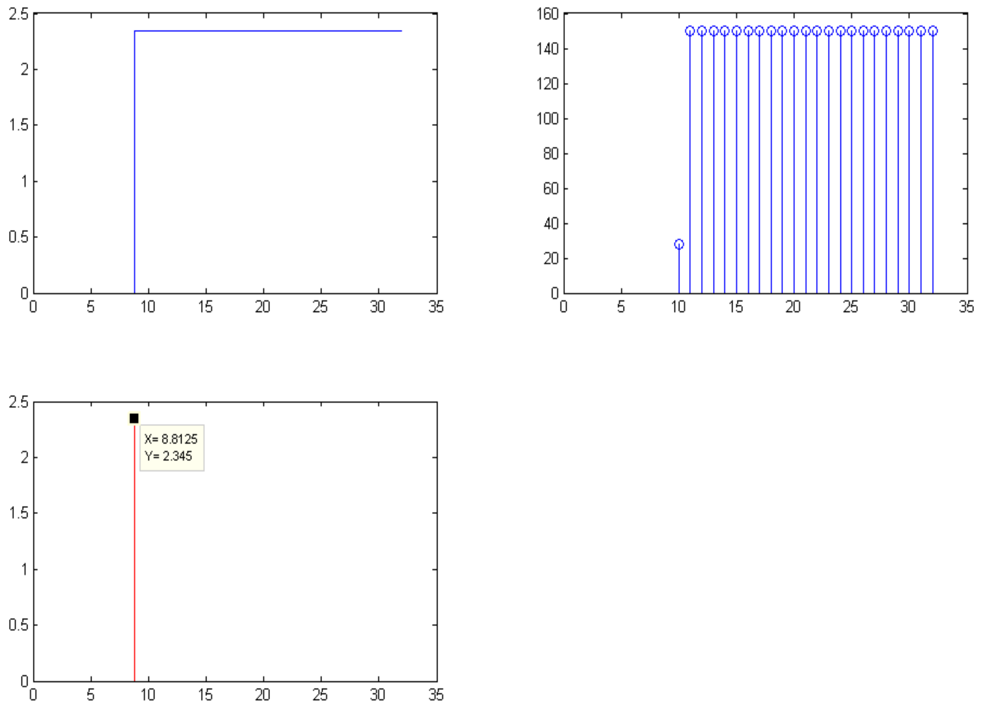

Figure 11: ***Top left:*** u(t-8.8125) ***Top right:*** y[n] ***Bottom left:*** Recovered dirac at $t_0 = 8.8125$, $A = 2.345$

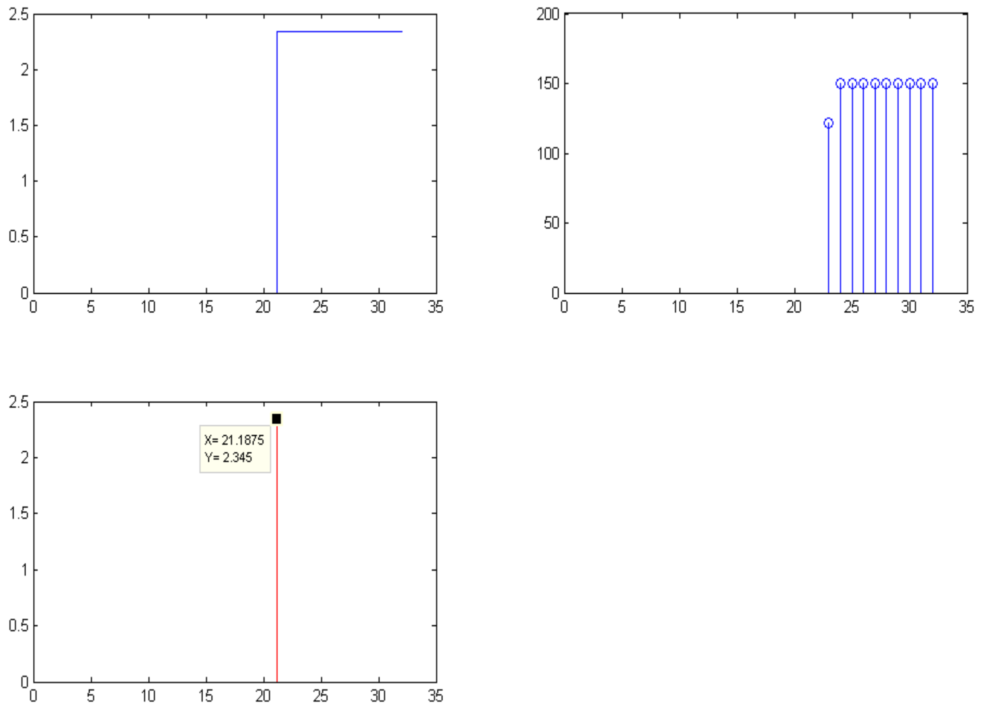

Figure 12: ***Top left:*** u(t-21.1875) ***Top right:*** y[n] ***Bottom left:*** Recovered dirac at $t_1 = 21.1875$, $A = 2.345$

Exercise 8

In this exercise, you have access to N = 40 low-resolution color images of 64x64 pixels. The goal is to generate a super-resolved color image of 512x512 pixels. The images observe the same scene and have been taken from different viewpoints. We assume that the geometric transformations between the views are 2-D translations. Write a program that registers each image with respect to the first image and which fits into the skeleton of the super-resolution algorithm provided.

Results:

The **PSNR is 23.55dB** for my code. This is the highest PSNR value I could have achieved by adjusting the noise threshold differently for each R, G and B layer. Below shows the figure displaying the resolved image adjacent to the reference image:

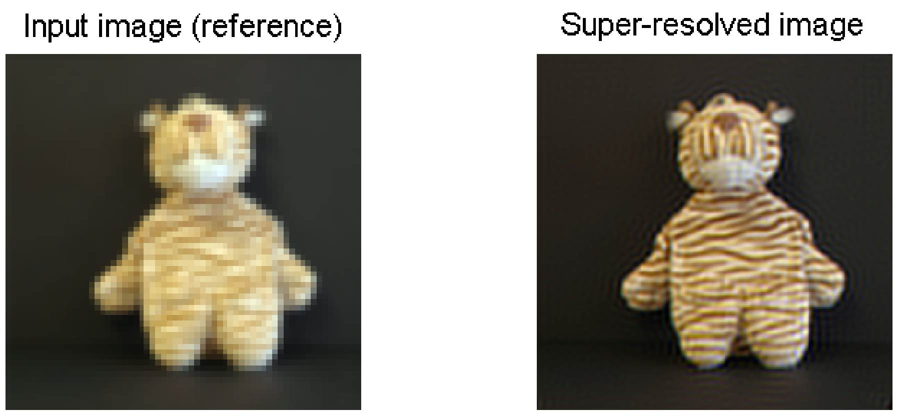

Figure 13: ***Left:*** First image, used as reference ***Right:*** Super-resolved image

Comments:

The noise threshold used for each layer is different because improvement in PSNR value can be found when different noise threshold is assigned for each layer. The values are below:

Layer Colour	Threshold value
Red	160
Green	140
Blue	130

Table 5: Noise threshold for each layer of colours

The 2-D cubic B-spline used is from the 2DBspline.mat and is as below:

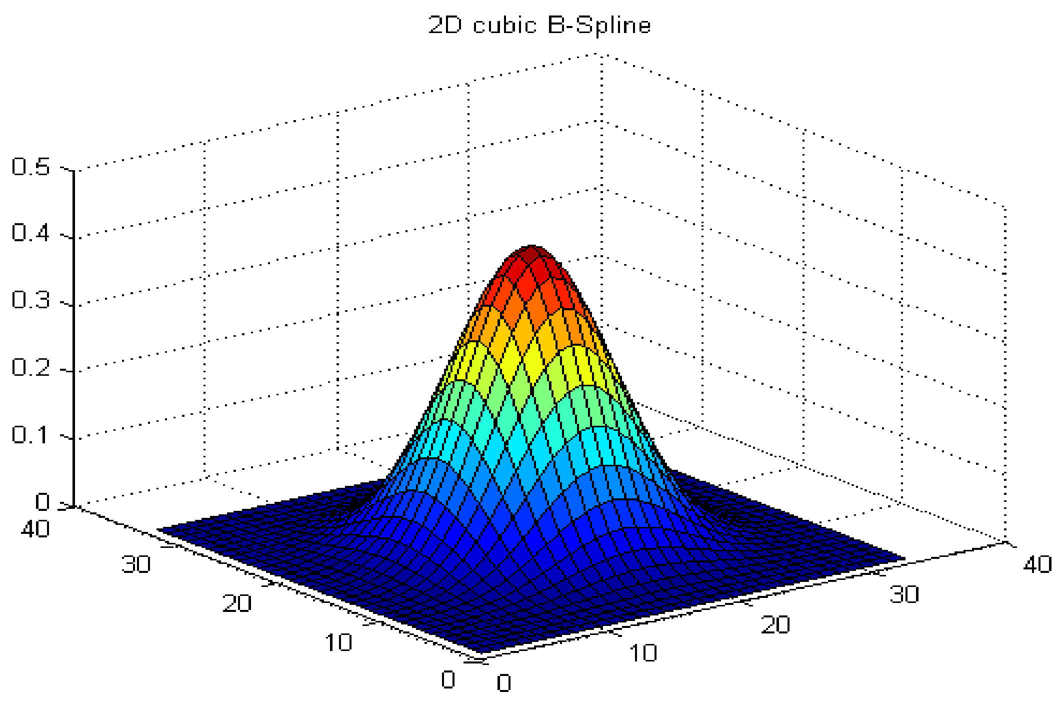

Figure 14: 2-D cubic B-spline

Appendix A MATLAB codes

A.1 Exercise 1

```
% Exercise 1
clear;
close all;

% parameters
T = 64;
N_sample = 2048;
N = N_sample/T;
d = 3;
wav = 'db4';
iter = log2(T);
P_z = (1/2048)*[-5, 0, 49, 0, -245, 0, 1225, 2048, 1225, 0, -245, 0, 49, 0, -5];
xval = [0 1/T:1/T:N-(1/T)];

% subplot config
splot_width = 0.45;
splot_height = 0.4;

% Plot poles and zeros of P(z), db4 scaling function and its dual basis
figure(1);
subplot('Position',[0.07 0.55 splot_width splot_height]);
zplane(P_z);
title('Poles and zeros of P(z)');

[phi_T_1, ~] = wavefun(wav, 1);
subplot('Position',[0.07 0.07 splot_width splot_height]);
zplane(phi_T_1(phi_T_1 ~= 0));
title('Poles and zeros of db4 scaling function for iteration = 1');

phi_T_tilde_1 = deconv(P_z, phi_T_1(phi_T_1 ~= 0));
subplot('Position',[0.52 0.07 splot_width splot_height]);
zplane(phi_T_tilde_1);
title('Poles and zeros of dual basis of scaling function');
```

```
% Iterate filter and plot scaling functions
figure(2);
for i = 1:iter
    phi_T_tilde = T*equi_filter(phi_T_tilde_1, i);
    subplot(2, 3, i);
    plot(phi_T_tilde);
    titlestr = ['Equivalent analysis Filter H^' int2str(i) '_0(z)'];
    title(titlestr);
end
phi_tilde = zeros(1,N_sample);
phi_tilde(N_sample - length(phi_T_tilde)+1:end) = phi_T_tilde;

% Plot db4 at at iteration = 6 and compare results using plot
phi = zeros(1,N_sample);
[phi_T, psi_T] = wavefun(wav, iter);
L_T = length(phi_T);
L_n = (L_T - 1)/T;
phi(1:L_T) = phi_T;

figure(3);
subplot(1, 2, 1);
plot(phi_T);
title('Daubechies scaling function in discrete-time domain - poly max degree 3');

subplot(1, 2, 2);
plot(phi_T_tilde);
title('Dual basis of Daubechies scaling function in discrete-time domain - poly max degree 3');

% Plot shifted kernels and reproduced polynomials
for m = 0:d
    for n = 0:N-1
        phi_tilde_shifted = [phi_tilde, zeros(1,(N-n)*T)];
        phi_tilde_shifted = phi_tilde_shifted((N-n)*T + 1:end);

        phi_shifted = [zeros(1,(N-n)*T), phi];
        phi_shifted = phi_shifted(1:N_sample);

        C(m+1, n+1) = (1/T)*phi_tilde_shifted*(xval.^m).';
        shifted_kernel(n+1,:,m+1) = C(m+1, n+1)*phi_shifted;
        shifted_kernel(n+1,:,m+1) = fliplr(shifted_kernel(n+1,:,m+1));
    end
end

reprod_poly = zeros(1,N_sample);
figure(4);
for m = 0:d
    subplot(2,2,m+1);
    for n = 0:N-L_n
        curr_kernel = shifted_kernel(n+1,:,m+1);
        reprod_poly = reprod_poly + curr_kernel;
        plot(xval, curr_kernel, ':k');
        hold on;
    end
    plot(xval, reprod_poly, 'r');
end
hold off;
```

A.2 Equivalent Filter - Support Exercise 1

```
function fun_out = equi_filter(fun,iter)

% If iter = 1
fun_out = fun;

if iter ~= 1
    for J = 2:iter
        fun_up_sampled = zeros(1,J*length(fun));
        num_zero_paddding = 2^(J-1);
        % Zero Padding
        for j = 1:length(fun)
            fun_up_sampled(num_zero_paddding*j) = fun(j);
        end

        fun_out = conv(fun_out,fun_up_sampled);
    end
    % Only take non-zeros
    fun_out = fun_out(fun_out ~= 0);
end

end
```

A.3 Exercise 2

```
% Exercise 2
clear;
close all;

% parameters
T = 64;
N_sample = 2048;
N = N_sample/T;
d = 3;
wav = 'bior4.4';
iter = log2(T);
xval = [0 1/T:1/T:N-(1/T)];

% Generate B-spline scaling function with its dual-basis
phi = zeros(1,N_sample);
[phi_T_tilde, ~, phi_T, ~] = wavefun(wav, iter);
L_T = length(phi_T);
L_n = (L_T - 1)/T;
phi(1:L_T) = phi_T;

phi_tilde = zeros(1,N_sample);
phi_tilde(N_sample - length(phi_T_tilde)+1:end) = phi_T_tilde;

figure(1);
subplot(1, 2, 1);
plot(phi_T);
title('B-Spline scaling function in discrete-time domain - poly max degree 3');

subplot(1, 2, 2);
plot(phi_T_tilde);
title('Dual basis of B-Spline scaling function in discrete-time domain - poly max degree 3');

% Plot shifted kernels and reproduced polynomials
for m = 0:d
    for n = 0:N-1
        phi_tilde_shifted = [phi_tilde, zeros(1,(N-n)*T)];
        phi_tilde_shifted = phi_tilde_shifted((N-n)*T + 1:end);

        phi_shifted = [zeros(1,(N-n)*T), phi];
        phi_shifted = phi_shifted(1:N_sample);

        C(m+1, n+1) = (1/T)*phi_tilde_shifted*(xval.^m).';
        shifted_kernel(n+1,:,m+1) = C(m+1, n+1)*phi_shifted;
        shifted_kernel(n+1,:,m+1) = fliplr(shifted_kernel(n+1,:,m+1));
    end
end

reprod_poly = zeros(1,N_sample);
figure(2);
for m = 0:d
    subplot(2,2,m+1);
    for n = 0:N-L_n
        curr_kernel = shifted_kernel(n+1,:,m+1);
        reprod_poly = reprod_poly + curr_kernel;
        plot(xval, curr_kernel, ':k');
        hold on;
    end
    plot(xval, reprod_poly, 'r');
end
hold off;
```

A.4 Exercise 3

```
clear;
close all;
% Load sample
load project_files_&_data/project_files_&_data/tau.mat
tau = tau(:);

% Parameters
K = 2;
N_sample = 2048;
T = 64;
N = N_sample/T;
xval = [0 1/T:1/T:N-(1/T)];

% Recover diracs
[amp, loc_t] = recover_x(tau, K);
loc_n = uint16(loc_t*T) + 1;

% Plot diracs
figure(1);
x_est = zeros(1, N_sample);
x_est(loc_n) = amp;
x_est(x_est == 0) = NaN;
stem(xval, x_est, 'Marker', 'x', 'Color', 'r');

% Calculate tau from estimated diracs and compare with given tau
for m = 0:3
    tau_est(m+1, 1) = amp.'*loc_t.^m;
end

same = tau - tau_est < 1e-9;
if prod(same)
    disp('Location retrieved is accurate');
else
    disp('Location retrieved is NOT accurate');
end
```

A.5 Recover Diracs function - Support multiple functions

```
function [a, loc_t] = recover_x(tau, K)

% Given tau, find coefficents h
h = anni_filter(tau, K);

% Recover locations and amplitudes
loc_t = roots(h);
V = vander(loc_t);
V = fliplr(V);
V = V.';

a = V\tau(1:K);

end
```

A.6 Anihilating Filter - Supports recovery of dirac function

```
function h = anni_filter(tau, K)
    tau = tau(:);

    toe_K = toeplitz(tau(K:end-1), tau(K:-1:1));
    T = -1*tau(K+1:end);
    h = toe_K\T; % Ax = B; x = A\B
    h = [1; h];
end
```

A.7 Exercise 4

```
% Exercise 4
clear;
run('ex1.m');
close all;

% Paramters
loc_known = [523, 1412];
amp_known = [1, 3];
K = length(loc_known);
T = 64;
wav = 'db4';

% Daubechies scaling function with sampling period T = 64
phi = zeros(1,N_sample);
[phi_T, psi_T] = wavefun(wav, 6);
phi(1:L_T) = phi_T;

% Create a stream of Diracs
x = zeros(1, N_sample);
x(loc_known) = amp_known;

% Sample signals with Daubechies scaling function
y = conv(phi, x);
y = y(1:T:N_sample);
tau = C*y.';
tau = tau(:);

% Reconstruct continuous time Diracs from sample
[amp, loc_t] = recover_x(tau, K);
loc_n = uint16(loc_t*T) + 1;

% Draw figures
figure(1);
subplot(2, 2, 1);
x_stem = x;
x_stem(x==0) = NaN;
stem(xval, x_stem);

subplot(2, 2, 2);
plot(xval, phi);

subplot(2, 2, 3);
stem(y, 'Marker', 'none');

% Calculate MSE
x_est = zeros(1, N_sample);
x_est(loc_n) = amp;
mse  = (1/2*N_sample) * real((x-x_est) * (x-x_est).');
disp(['MSE = ' num2str(mse)]);

subplot(2, 2, 4);
x_est(x_est==0) = NaN;
stem(xval, x_stem);
hold on;
stem(xval, x_est, 'Marker', 'x', 'Color', 'r');
hold off;
```

A.8 Exercise 5

```
% Exercise 5
clear;
run('ex1.m');
close all;

% load data and parameters
load project_files_&_data/project_files_&_data/samples.mat
K = 2;
T = 64;

% Find sample tau
tau = C*y_sampled.';

% Reconstruct continuous time Diracs from sample
[amp, loc_t] = recover_x(tau, K);
loc_n = uint16(loc_t*T + 1);

% Draw figures
figure(1);
subplot(1, 2, 1);
stem(y_sampled, 'Marker', 'none');

subplot(1, 2, 2);
x_est = zeros(1, N_sample);
x_est(loc_n) = amp;
x_est(x_est==0) = NaN;
stem(x_est, 'Marker', 'x', 'Color', 'r');
```

A.9 Exercise 6

```
% Exercise 6
clear;
close all;

% Paramters
K = 1:4;
loc_known = [750, 1050, 1100, 1250];
amp_known = [1, 2, 3, 4];
wav = ['db3'; 'db4'; 'db6'; 'db8'];
wav = cellstr(wav);
d = [2, 3, 5, 7];
T = 64;
N_sample = 2048;
N = N_sample/T;
xval = [0 1/T:1/T:N-(1/T)];

fig_iter = 0;
for num_of_diracs = K
    % Daubechies scaling function with sampling period T = 64
    phi = zeros(1,N_sample);
    db_type = wav{num_of_diracs};
    [phi_T, psi_T] = wavefun(db_type, 6);
    phi(1:length(phi_T)) = phi_T;
    compact_L = nnz(phi_T < 1e-3);
    L_T = length(phi_T);
    L_n = (L_T - 1)/T;
    phi_tilde = fliplr(phi); % Dual basis

    % Find coefficent C
    for m = 0:d(num_of_diracs)
        for n = 0:N-1
            phi_tilde_shifted = [phi_tilde, zeros(1,(N-n)*T)];
            phi_tilde_shifted = phi_tilde_shifted((N-n)*T + 1:end);
            C(m+1, n+1) = (1/T)*phi_tilde_shifted*(xval.^m).';
        end
    end
    % Create a stream of Diracs
    x = zeros(1, N_sample);
    x(loc_known(1:num_of_diracs)) = amp_known(1:num_of_diracs);

    % Sample signals with Daubechies scaling function
    y = conv(phi, x);
    y = [0 y];
    y = y(1:T:N_sample);

    tau = C*y.';
    tau = tau(:);

    % Reconstruct continuous time Diracs from sample
    [amp, loc_t] = recover_x(tau, num_of_diracs);
    loc_n = uint16(loc_t*T) + 1;

    % Draw figure
    fig_iter = fig_iter + 1;
    figure(fig_iter);


    subplot(1, 2, 1);
    stem(y, 'Marker', 'none');

    % Calculate MSE
    x_est = zeros(1, N_sample);
    x_est(loc_n) = amp;
    mse  = (1/2*N_sample) * real((x-x_est) * (x-x_est).');
    disp(['MSE = ' num2str(mse) ' for K = ' num2str(num_of_diracs)]);

    x_stem = x;
    x_stem(x==0) = NaN;
    x_est(x_est==0) = NaN;

    subplot(1, 2, 2);
    stem(xval, x_stem);
    hold on;
    stem(xval, x_est, 'Marker', 'x', 'Color', 'r');
    hold off;
end
```

A.10 Exercise 7

```
% Exercise 7
clear;
close all;

% Parameters
K = 1;
T = 64;
N = 32;
N_sample = N*T;
A = 2.345;
t_known = [8.8125, 21.1875];
wav1 = 'bior1.1';
wav2 = 'bior2.2';
iter = log2(T);
xval = [0 1/T:1/T:N-(1/T)];

% Create x
u_temp = ones(1, N_sample);
u_temp = [zeros(1, t_known(1)*T), u_temp];
u(1,:) = u_temp(1:N_sample);

u_temp = ones(1, N_sample);
u_temp = [zeros(1, t_known(2)*T), u_temp];
u(2,:) = u_temp(1:N_sample);

% Sampling kernel
```

```
beta_0 = zeros(1,N_sample);
beta_tilde_0 = zeros(1, N_sample);
[phi_tilde_0_T, ~, phi_0_T, ~] = wavefun(wav1, iter);
beta_0(1:length(phi_0_T)) = phi_0_T;
beta_tilde_0(1:length(phi_tilde_0_T)) = phi_tilde_0_T;

% Equivalent kernel;
beta_1 = zeros(1,N_sample);
beta_tilde_1 = zeros(1, N_sample);
[phi_tilde_1_T, ~, phi_1_T, ~] = wavefun(wav2, iter);
phi_tilde_1_T = phi_tilde_1_T(phi_tilde_1_T~=0);

beta_1(1:length(phi_1_T)) = phi_1_T;
beta_tilde_1(1:length(phi_tilde_1_T) - 1) = phi_tilde_1_T(2:end);

% Find coefficent C
for m = 0:K
    for n = 0:N-1
        beta_tilde_1_shifted = [fliplr(beta_tilde_1), zeros(1,(N-n)*T)];
        beta_tilde_1_shifted = beta_tilde_1_shifted((N-n)*T + 1:end);
        C(m+1, n+1) = (1/T)*beta_tilde_1_shifted*(xval.^m).';
    end
end

% For loop for each location
for i = 1:2
    % Create x
    x = A*u(i,:);

    % Sample signals with B-spline scaling function
    y = conv(beta_0, x);
    y = y(1:T:N_sample);

    % Find z[n]
    dx = diff(x);
    z = conv(dx, beta_1);
    z = z(1:T:N_sample);

    % Find tau
    tau = C*z.';
    tau = tau(:);

    % Recover dirac
    [amp, loc_t] = recover_x(tau, K);
    loc_n = uint16(loc_t*T) + 1;

    disp(['Location ', num2str(i), ' : ', num2str(loc_t)]);

    % Plot results
    figure(i);
    subplot(2, 2, 1);
    plot(xval, x);
    subplot(2, 2, 2);
    y_stem = y;
    y_stem(y_stem==0) = NaN;
    stem(y_stem);


    subplot(2, 2, 3);
    x_stem = zeros(1, N_sample);
    x_stem(t_known(i)*T + 1) = A;
    x_stem(x_stem==0) = NaN;
    x_est = zeros(1, N_sample);
    x_est(loc_n) = amp;
    x_est(x_est==0) = NaN;
    stem(xval, x_stem);
    hold on;
    stem(xval, x_est, 'Marker', 'x', 'Color', 'r');
    hold off;
end

% Plot beta_0 and beta_1 scaling function with their dual basis (right)
figure(3);
```

```
subplot(2, 2, 1); plot(beta_0);
subplot(2, 2, 2); plot(beta_tilde_0);
subplot(2, 2, 3); plot(beta_1);
subplot(2, 2, 4); plot(beta_tilde_1);
```

A.11 Exercise 8 - Image registration function

```
function [Tx_RGB Ty_RGB]= ImageRegistration
% ***************************************************************************
% Wavelets and Applications Course - Dr. P.L. Dragotti
% MATLAB mini-project 'Sampling Signals with Finite Rate of Innovation'
% Exercice 6
% ***************************************************************************
%
% FOR STUDENTS
%
% This function registers the set of 40 low-resolution images
% 'LR_Tiger_xx.tif' and returns the shifts for each image and each layer
% Red, Green and Blue. The shifts are calculated relatively to the first
% image 'LR_Tiger_01.tif'. Each low-resolution image is 64 x64 pixels.
%
%
% OUTPUT:   Tx_RGB: horizontal shifts, a 40x3 matrix
%           Ty_RGB: vertical shifts, a 40x3 matrix
%
% NOTE: _Tx_RGB(1,:) = Ty_RGB(1,:) = (0 0 0) by definition.
%       _Tx_RGB(20,2) is the horizontal shift of the Green layer of the
%       20th image relatively to the Green layer of the firs image.
%
%
% OUTLINE OF THE ALGORITHM:
%
% 1.The first step is to compute the continuous moments m_00, m_01 and m_10
% of each low-resolution image using the .mat file called:
% PolynomialReproduction_coef.mat. This file contains three matrices
% 'Coef_0_0', 'Coef_1_0' and 'Coef_0_1' used to calculate the continuous
% moments.
%
% 2.The second step consists in calculating the barycenters of the Red,
% Green and Blue layers of the low-resolution images.
%
% 3.By computing the difference between the barycenters of corresponding
% layers between two images, the horizontal and vertical shifts can be
% retrieved for each layer.
%
%
% Author:   Loic Baboulaz
% Date:     August 2006
%
% Imperial College London
% ****************************************************************************c
close all;

% Load the coefficients for polynomial reproduction
load('PolynomialReproduction_coef.mat','Coef_0_0','Coef_1_0','Coef_0_1');

% -------- include your code here -----------
load 2DBspline;

figure;
surf(Spline2D);
title('2D cubic B-Spline');

% Parameters
N = 40;
basename = 'LR_Tiger_';
C_row_size = size(Coef_0_0, 1);
noise_T = [160, 140 130]/255;

% Initialise
```

```
Tx_RGB = [];
Ty_RGB = [];

for i = 1:N
    % Load LR image
    idx_str = num2str(i);
    if i < 10 % If index is less than ten
        idx_str = strcat('0', num2str(i));
    end
    filename = strcat(basename, idx_str, '.tif');
    ori_img = imread(filename);
    ori_img = double(ori_img)/255;

    dx_colour = [];
    dy_colour = [];
    for layer = 1:3
        % Work only on one layer then suppress signal below threshold as
        % noise
        f_x_y = ori_img(:, :, layer);
        f_x_y(f_x_y <= noise_T(layer)) = 0;

        % Calculate sample and only choose middle "high" values
        S = conv2(f_x_y, Spline2D);
        S = S(17:80, 17:80);

        % Calculate the continuous moment
        prod_over_n1 = [];
        prod_over_n2 = [];
        prod_over_n3 = [];
        for m = 1:C_row_size
            prod_over_n1 = [prod_over_n1, Coef_0_0(m, :)*S(m,:).'];
            prod_over_n2 = [prod_over_n2, Coef_1_0(m, :)*S(m,:).'];
            prod_over_n3 = [prod_over_n3, Coef_0_1(m, :)*S(m,:).'];
        end
        m_0_0 = sum(prod_over_n1);
        m_1_0 = sum(prod_over_n2);
        m_0_1 = sum(prod_over_n3);

        % Compute the barycenter
        x_bar = m_1_0/m_0_0;
        y_bar = m_0_1/m_0_0;

        % If current image is first, make center as reference
        if i == 1
            x_bar1(layer) = x_bar;
            y_bar1(layer) = y_bar;
        end

        % Join translation for current layer with the rest
        dx_colour = [dx_colour, x_bar - x_bar1(layer)];
        dy_colour = [dy_colour, y_bar - y_bar1(layer)];
    end
    Tx_RGB = [Tx_RGB; dx_colour];
    Ty_RGB = [Ty_RGB; dy_colour];
end
```

Appendix B Figures

B.1 Exercise 6

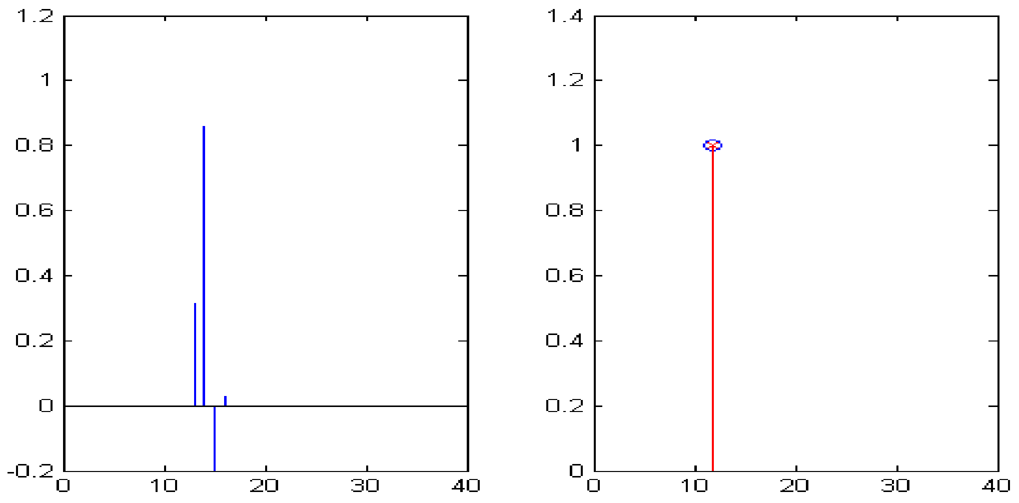

Figure 15: ***Left:*** Samples, $y[n]$ ***Right:*** Dirac retrieved

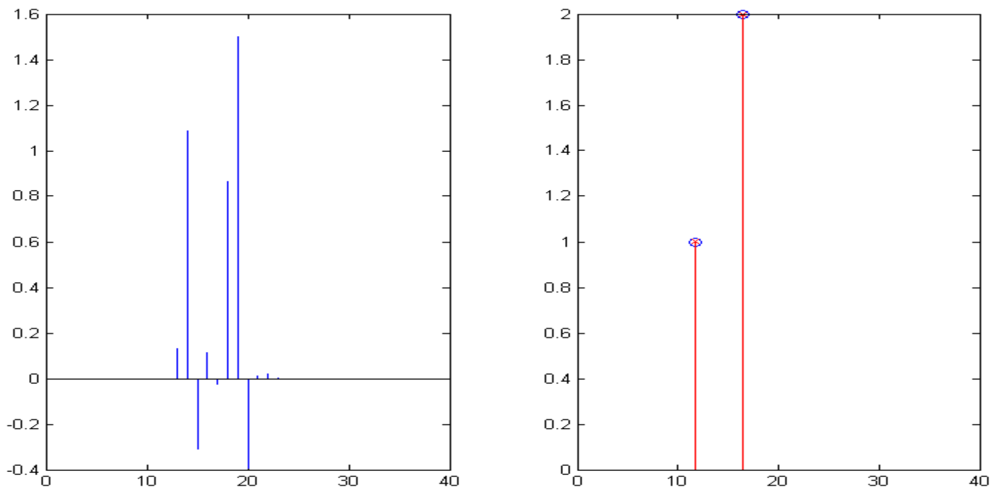

Figure 16: ***Left:*** Samples, $y[n]$ ***Right:*** Stream of diracs retrieved

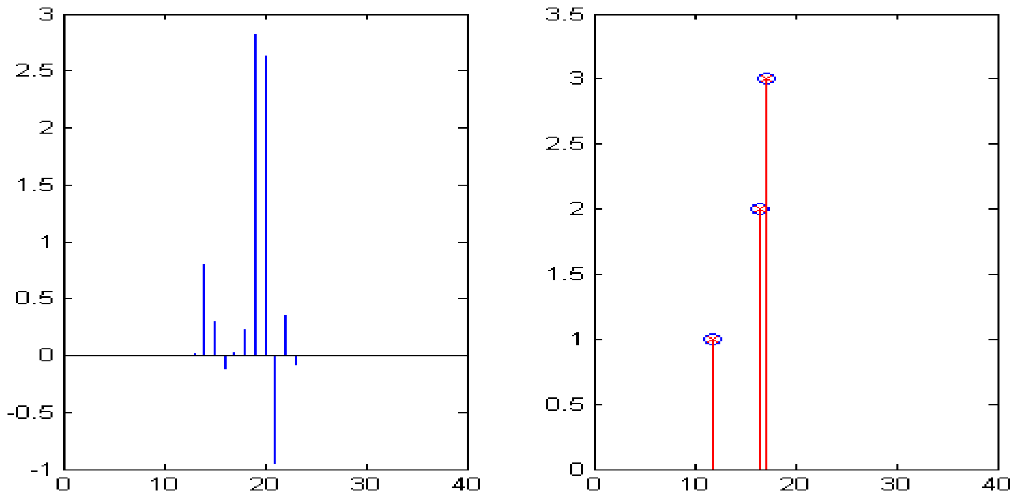

Figure 17: ***Left:*** Samples, $y[n]$ ***Right:*** Stream of diracs retrieved

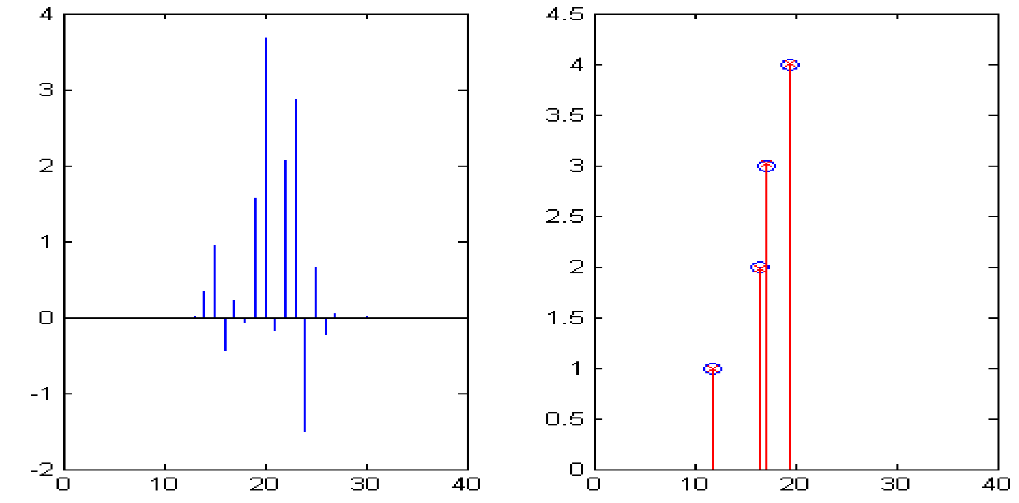

Figure 18: ***Left:*** Samples, $y[n]$ ***Right:*** Stream of diracs retrieved. Blue solid line with circle marker is the stream of diracs created. Red solid line with cross marker is the recovered diracs