Imperial College London

Department of Electrical and Electronic Engineering

Sampling Signals with Finite Rate of Innovation with an Application to Image Super-Resolution

MATLAB Mini-Project

Coursework Report for Wavelets, Representation Learning and their Applications

ELEC97092 - 2021-2022

Author Name:	Manginas Vasileios
Author Email:	vm3218@ic.ac.uk
Author CID:	01542774

1 Exercise 1

In this exercise we use a Daubechies scaling function to reproduce polynomials signals of maximum degree 3, as shown in Equation (1) below. The Daubechies filters are the shortest orthogonal FIR filters with maximum flat frequency responses at $\omega = 0$ and $\omega = \pi$. This is equivalent to saying that they provide us with the maximum number of vanishing moments for a given support. More specifically, we know that an n^{th} order Daubechies wavelet has n vanishing moments. Thus, for polynomial signals of maximum degree $n-1$ the wavelet coefficients will be equal to zero, meaning that the scaling function alone is able to fully reproduce these signals. So, in order to be able to reproduce polynomials of maximum degree 3 we use the 4^{th} order Daubechies scaling function.

$$\sum_{n \in Z} c_{m,n} \varphi(t-n) = t^m \quad m = 0, 1, \ldots, 3 \tag{1}$$

Approximations of the scaling function $\varphi(t)$ as well as of the corresponding wavelet $\psi(t)$ can be obtained through the MATLAB $wavefun$ function. To use this we have to provide the name of the orthogonal wavelet we are interested in, as well as the number of iterations for the cascade algorithm that approximates it. In this case, the former is $'db4'$, for 4^{th} Daubechies, while the latter is 6 to provide a resolution of 1/64.

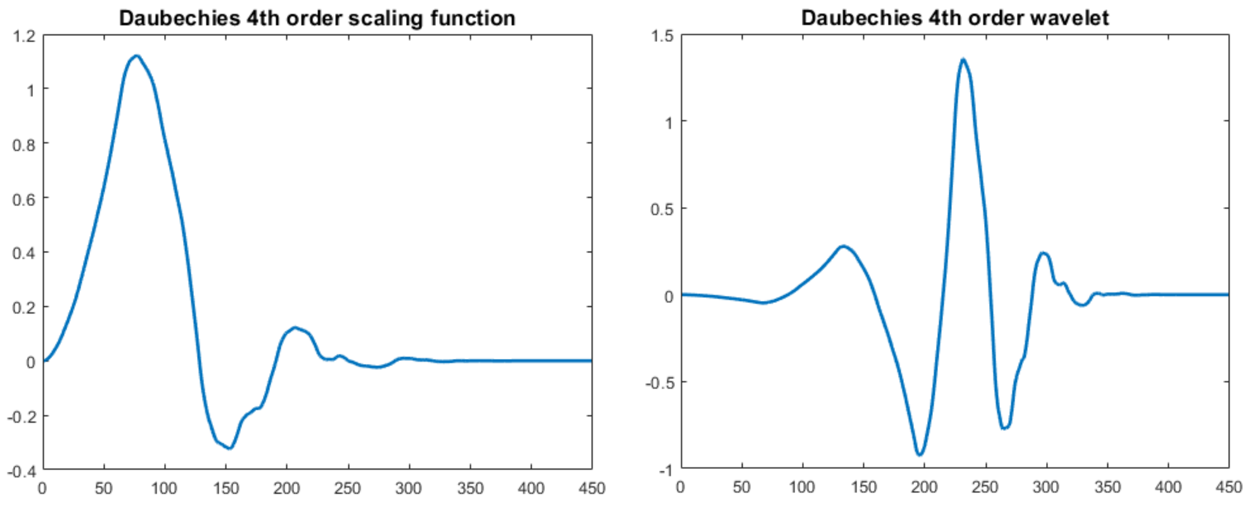


Figure 1: Approximations of the 4^{th} order Daubechies scaling function $\varphi(t)$ and wavelet $\psi(t)$

Having obtained the correct scaling function, we now need to calculate the coefficients $c_{m,n}$ which will satisfy Equation (1). These can generally be found using the dual of the scaling function as shown in Equation (2). However, since Daubechies wavelets are orthogonal $\tilde{\varphi}(t) = \varphi(t)$, the scaling function we have obtained.

$$c_{m,n} = \langle t^m, \tilde{\varphi}(t-n) \rangle \tag{2}$$

For each polynomial degree from 0 to 3, we have obtained two plots. The first depicts the shifted and scaled versions of the scaling function $\varphi(t)$, as well as the resulting reconstructed polynomial created from their sum. The second depicts the reconstruction error.

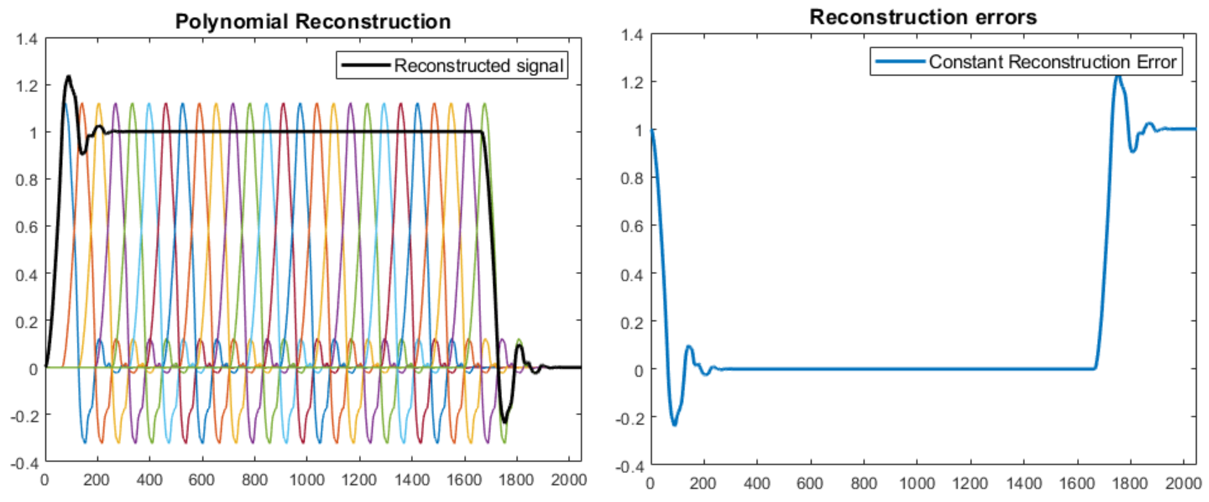


Figure 2: Reconstruction of constant polynomial and resulting error using the *db*4 scaling function

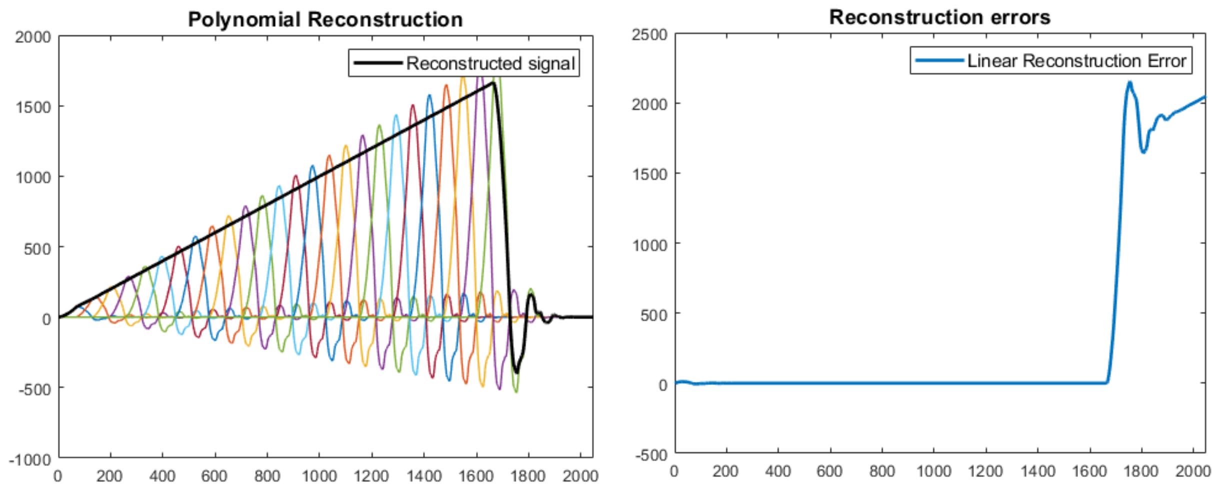


Figure 3: Reconstruction of linear polynomial and resulting error using the *db*4 scaling function

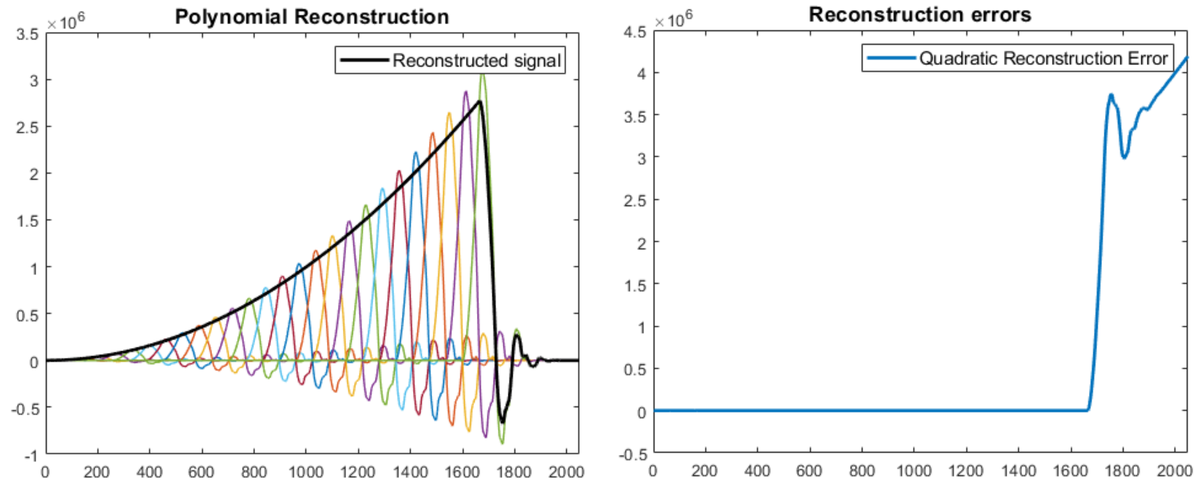
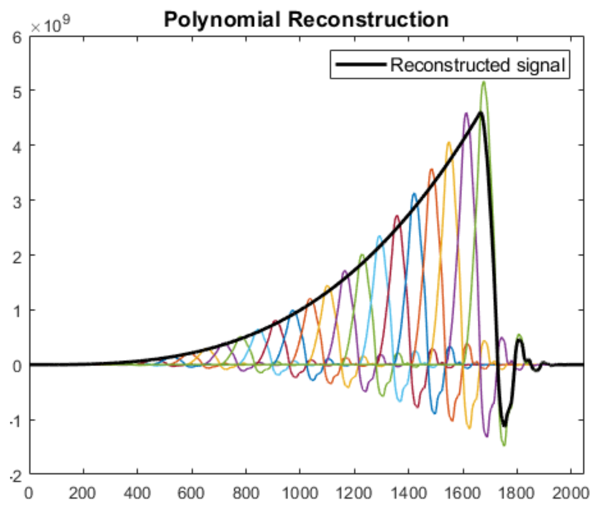


Figure 4: Reconstruction of quadratic polynomial and resulting error using the *db*4 scaling function



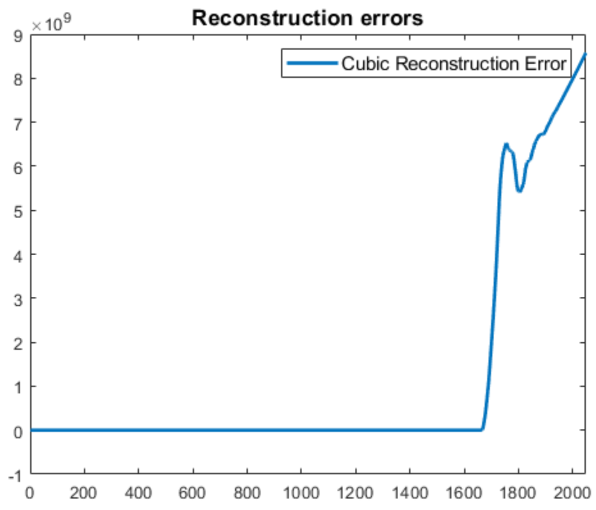


Figure 5: Reconstruction of cubic polynomial and resulting error using the $db4$ scaling function

As can clearly be seen, for all 4 polynomial degrees, the reconstruction error is zero apart from the beginning and end of our x-axis range. This non-polynomial behaviour towards the edges of the spectrum is expected, and occurs because we are always truncating the sums seen in Equation (1). While in theory n, the running parameter, should go from $-\infty$ to $+\infty$, we are only taking a finite interval, which naturally results in imperfect reconstruction towards the ends of the spectrum. In this case, we shift and multiply the scaling function we have obtained for $n = 0 - 32 - L$, where L is the local support of the sampling kernel. As indicated by the $xval$ array that is returned by the $wavefun$ function, the support in our case is $L = 7$, and so we use $n = 0 - 25$. This results in 26 distinct scaled and shifted versions of the scaling function. It is worth noting that the MATLAB code written for this exercise, as well as all other exercises, has been placed in this GitHub repository rather than in the Appendix for clarity and ease of reading.

2 Exercise 2

In Exercise 2 we replicate what we have done in Exercise 1 but using a different scaling function. In this case, the scaling function is not provided by the Daubechies family but rather from the B-spline family. We focus on two differences between the families.

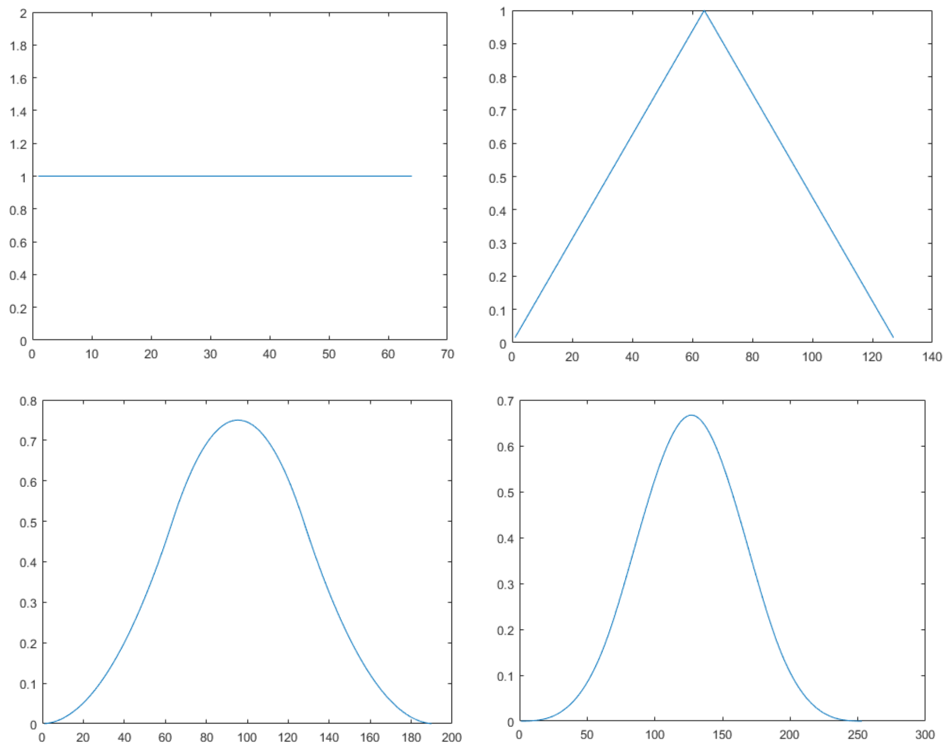


Figure 6: B-splines of order 0 to 3

The first is that an n^{th} order B-spline has $n+1$ vanishing moments, as opposed to the n vanishing moments found in n^{th} order Daubechies. Therefore, to reconstruct polynomials of maximum degree 3, we require N=3, the cubic B-spline $\beta_3(t)$. In general, the N^{th} order spline $\beta_N(t)$ can be obtained by convolving the 0^{th} order B-spline $\beta_0(t)$, the box function, with itself N times. Thus, we create $\beta_3(t)$, the cubic B-spline, by defining $\beta_0(t)$ and convolving it with itself 3 times. This process yields the first 4 B-splines as seen in the Figure above.
The second important difference that we focus on is that B-splines, contrary to Daubechies and with the exception of the 0^{th} spline $\beta_0(t)$, are not orthogonal. For us, this entails that we can no longer plug the scaling function $\varphi(t)$ in Equation (2), but rather have to obtain the dual of the scaling function $\tilde{\varphi}(t)$. We outline the entire process we will follow in the list below.

- **Step 1:** Find $g_0[n]$ through satisfying the two-scale relationship below using $\varphi(t) = \beta_3(t)$.

$$\varphi(t) = \sqrt{2} \sum_{n=-\infty}^{\infty} g_0[n]\varphi(2t-n) \tag{3}$$

- **Step 2:** Get the z-transform of $g_0[n]$: $G_0(z)$.
- **Step 3:** Obtain any $H_0(z)$ as long as $P(z) = H_0(z)G_0(z)$ satisfies the halfband property $P(z)+P(-z) = 2$. One way to obtain this is by using the formula of $P(z)$ given by Daubechies which does satisfy the halfband property. We can then find $H_0(z)$ through spectral factorizaiton of $P(z)$.
- **Step 4:** Get the inverse z-transform of $H_0(z)$: $h_0[n]$.

- **Step 5:** Get an iterated estimation of $\tilde{\varphi(t)}$ by using the expression below:

$$\tilde{\varphi}^{(i)}(t) = 2^{i/2} h_0^{(i)}[n] \tag{4}$$

Here $h_0^{(i)}[n]$ is the inverse z-transform of $H_0^{(i)}(z) = \prod_{k=0}^{i-1} H_0\left(z^{2^k}\right)$. We can calculate $h_0^{(i)}[n]$ in the time domain by iteratively convolving the sequence $h_0[n]$ with the same $h_0[n]$ upsampled by 2.

- **Step 6:** Compute the coefficients $c_{m,n}$ using the estimate of the dual of the scaling function $\tilde{\varphi(t)}$ in Equation (2).
- **Step 7:** Implement Equation (1) using the calculated coefficients.

We now show the work done for each of the steps shown above.

- **Step 1:** We find that the cubic spline $\beta_3(t)$ can be expressed using squashed, shifted, and scaled versions of itself in the following way:

$$\beta_3(t) = \frac{1}{8}\beta_3(2t) + \frac{1}{2}\beta_3(2t-1) + \frac{3}{4}\beta_3(2t-2) + \frac{1}{2}\beta_3(2t-3) + \frac{1}{8}\beta_3(2t-4) \tag{5}$$

Using this we obtain the $g_0[n]$ which satisfies the two-scale relationship:

$$g_0[n] = \frac{1}{8\sqrt{2}}[1,4,6,4,1] \tag{6}$$

- **Step 2:** Obtain the z-transform $G_0(z)$:

$$G_0(z) = \frac{1}{8\sqrt{2}}\left(1 + 4z^{-1} + 6z^{-2} + 4z^{-3} + 1z^{-4}\right) = \frac{1}{8\sqrt{2}}(1+z)^2\left(1+z^{-1}\right)^2 \tag{7}$$

- **Step 3:** We use the Daubechies formula with $p = 4$ to obtain $P(z)$ and consequently $H_0(z)$ through spectral factorization. We choose $p = 4$ since our cubic spline has 4 vanishing moments, which is also equivalent to 4 zeros at $\omega = \pi$. Using the fact that $cos(\omega) = \frac{1}{2}(z + z^{-1})$ we obtain:

$$\begin{aligned}
P(z) &= 2\left(\frac{1}{4}(2+z+z^{-1})\right)^4 \cdot \sum_{k=0}^{3}\begin{pmatrix} 3+k \\ k \end{pmatrix}\left(\frac{1}{4}(2+z+z^{-1})\right)^k \\
&= \frac{1}{128}(1+z)^4(1+z^{-1})^4 \cdot \sum_{k=0}^{3}\begin{pmatrix} 3+k \\ k \end{pmatrix}\left(\frac{1}{4}(2+z+z^{-1})\right)^k \\
&= \frac{1}{128}\cdot 8\sqrt{2}\cdot G_0(z)\cdot(1+z)^2(1+z^{-1})^2 \cdot \sum_{k=0}^{3}\begin{pmatrix} 3+k \\ k \end{pmatrix}\left(\frac{1}{4}(2+z+z^{-1})\right)^k \\
&= G_0(z)\cdot\frac{\sqrt{2}}{16}\cdot(1+z)^2(1+z^{-1})^2\cdot\frac{1}{16}\left(-5z^{-3} + 40z^{-2} - 131z^{-1} + 208 - 131z + 40z^2 - 5z^3\right)
\end{aligned} \tag{8}$$

Therefore, since $P(z) = H_0(z)G_0(z)$, the remaining terms in the expression above besides $G_0(z)$ make up our $H_0(z)$:

$$H_0(z) = \frac{\sqrt{2}}{256}\left(-5z^{-5} + 20z^{-4} - z^{-3} - 96z^{-2} + 70z^{-1} + 280 + 70z - 96z^2 - z^3 + 20z^4 - 5z^5\right) \quad (9)$$

- **Step 4:** Obtain the discrete sequence $h_0[n]$:

$$h_0[n] = \frac{\sqrt{2}}{256}[-5, 20, -1, -96, 70, 280, 70, -96, -1, 20, -5] \quad (10)$$

- **Step 5:** This step of constructing $\tilde{\varphi}^{(i)}(t)$ was completed in MATLAB. As mentioned, we took the sequence $h_0[n]$, which in this case is notated as $h_0^{(1)}[n]$ and convolved it with an upsampled-by-2 version of itself to obtain $h_0^{(2)}[n]$. This is then convolved with a version of $h_0[n]$ which has been upsampled by 4 to obtain $h_0^{(3)}[n]$ and so on, for the specified number of iterations i.
- **Step 6:** Having obtained an approximation for $\tilde{\varphi}(t)$ we follow the exact same steps as in Exercise 1 to calculate the coefficients $c_{m,n}$.
- **Step 7:** Finally, we reconstruct the polynomial with the scaled and shifted $\tilde{\varphi}(t)$s again as in Exercise 1.

The process above yielded the Figures in the following pages. We observe that in the constant polynomial case the reconstruction is perfect and the error is zero, excluding of course the boundary effect brought forth by truncating sums as mentioned in Exercise 1. However, for the linear, quadratic, and cubic polynomial reconstruction the error is not zero. In fact, for each of these polynomials the error within the region unaffected by the boundary effects appears to behave as a polynomial of degree one less than the polynomial being reconstructed. For example, when reconstructing a quadratic polynomial, the error is linear. This in turn means that our signal reconstructs a polynomial which is not a monomial, as we would have expected, but rather has additional terms.

This observed error cannot be attributed to boundary effects as mentioned and thus requires further investigation. Other methods for obtaining the dual basis were also examined. However, we considered these to be unfit as they did not follow the coursework guidelines/hints and so were ultimately not pursued or placed in the report.

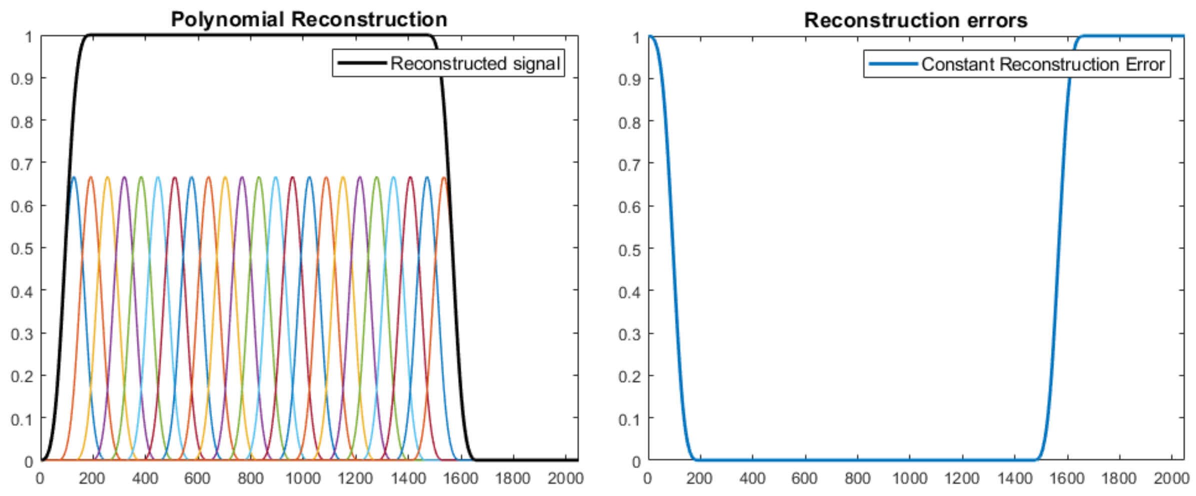


Figure 7: Reconstruction of constant polynomial and resulting error using the $\beta_3(t)$ scaling function

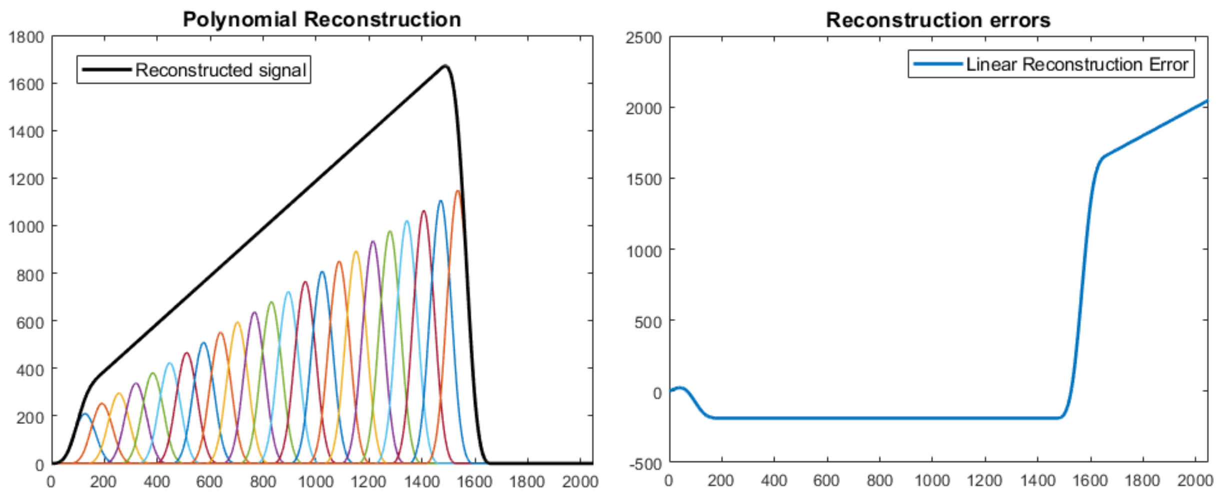


Figure 8: Reconstruction of linear polynomial and resulting error using the $\beta_3(t)$ scaling function

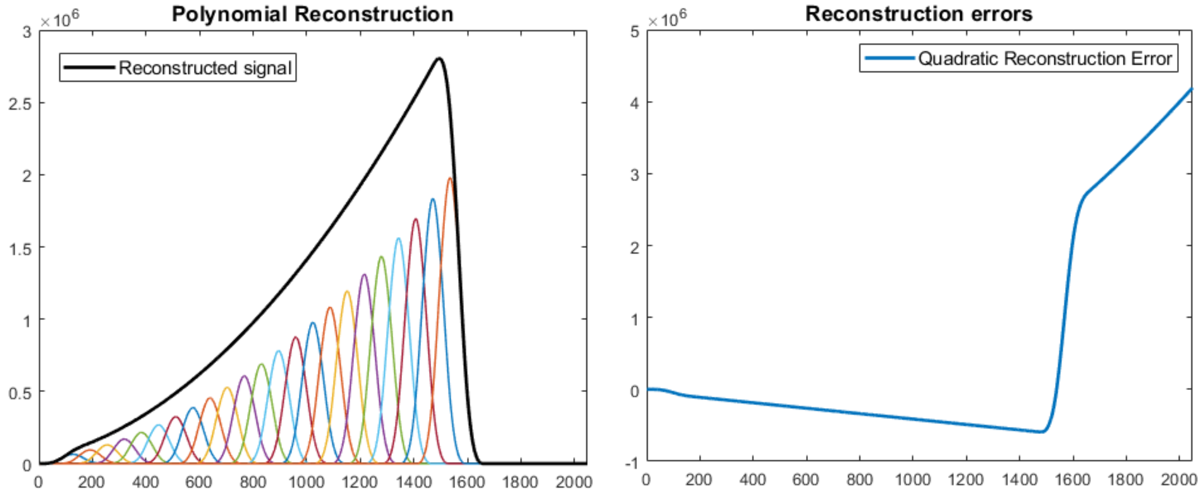


Figure 9: Reconstruction of quadratic polynomial and resulting error using the $\beta_3(t)$ scaling function

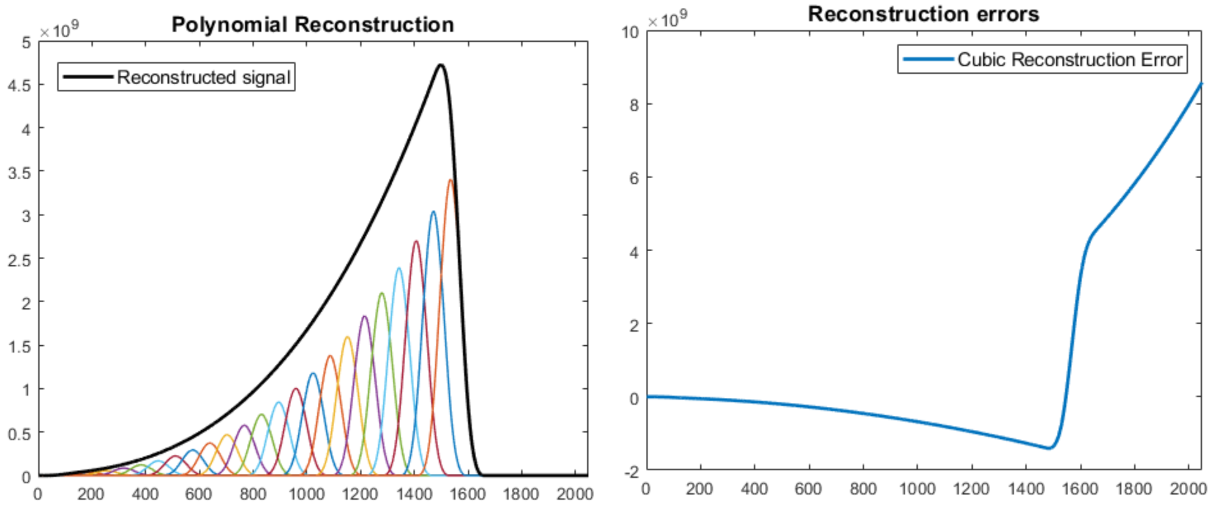


Figure 10: Reconstruction of cubic polynomial and resulting error using the $\beta_3(t)$ scaling function

3 Exercise 3

Exercises 3, 4, and 5 all revolve around the same signal: a stream of K Diracs. The form of our signal $x(t)$ is given in Equation (11) below. The aim of the algorithm implemented in these next exercises is to be able to retrieve the locations t_k as well as the amplitudes a_k of the Diracs. The only additional assumption that we impose is that our sampling kernel $\varphi(t)$ is able to reproduce polynomials of maximum degree $N \geq 2K-1$.

$$x(t) = \sum_{k=0}^{K-1} a_k \delta\left(t - t_k\right) \tag{11}$$

The algorithm for the location and amplitude retrieval can be broken down in 3 steps:

1. Construct the first $N+1$ moments τ_m of the signal $x(t)$.
2. Extract the locations t_k using the moments τ_m.
3. Extract the amplitudes a_k using the moments τ_m as well as the locations t_k.

Step 1 is the focus of Exercise 4 and will be explained in detail there. For Exercise 3, the moments have already been given, leaving steps 2 and 3 to be implemented.

To find the locations t_k we introduce the annihilating filter h_m, $m = 0, 1, ..., K$ with z-transform as given below.

$$H(z) = \prod_{k=0}^{K-1} \left(1 - t_k z^{-1}\right) \tag{12}$$

Since the locations t_k are the roots of the filter, than a convolution with the moments τ_m will result in zero, hence the filter name. In matrix/vector form we get the annihilation equation: $\boldsymbol{TH} = 0$. Using the fact that $h_0 = 1$ we can obtain the rest of the filter and consequently its roots by solving the system shown below. Note that the indexes have been adapted to MATLAB's one-based indexing.

$$\begin{bmatrix} \tau_K & \tau_{K-1} & \cdots & \tau_1 \\ \tau_{K+1} & \tau_K & \cdots & \tau_2 \\ \vdots & \vdots & \ddots & \vdots \\ \tau_N & \tau_{N-1} & \cdots & \tau_{N-K+1} \end{bmatrix} \begin{pmatrix} h_2 \\ h_3 \\ \vdots \\ h_{K+1} \end{pmatrix} = - \begin{pmatrix} \tau_{K+1} \\ \tau_{K+2} \\ \vdots \\ \tau_{N+1} \end{pmatrix} \tag{13}$$

Having found the locations t_k, we can use them in order to obtain the amplitudes a_k. This is achieved by solving the following Vandermonde system:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ t_0 & t_1 & \cdots & t_{K-1} \\ \vdots & \vdots & \ddots & \vdots \\ t_0^{K-1} & t_1^{K-1} & \cdots & t_{K-1}^{K-1} \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{K-1} \end{pmatrix} = \begin{pmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_K \end{pmatrix} \tag{14}$$

Implementing the methods outlined above for $K = 2$ allows us to extract the parameters of interest for the moments array τ_m given. These are presented below.

t_k	a_k
13.3	1.5
16.2	0.3

Table 1: Extracted locations $\mathrm{t_k}$ and corresponding weights $\mathrm{a_k}$ for the given moments

4 Exercise 4

As mentioned in the previous section, Exercise 4 aims to implement the first step of the algorithm, generating the moments of the signal $x(t)$. We can calculate the moments τ_m using the sampling process we implemented in Exercise 1. More specifically, for each polynomial degree m that our scaling function can reproduce we calculate the coefficients $c_{m,n}$ as shown below, and multiply each with the corresponding sample y_n. Finally, we sum across n, resulting in one moment per polynomial degree, on which we will then use our annihilating filter.

$$\begin{aligned}\tau_m &= \sum_n c_{m,n} y_n \\ y_n &= \langle x(t), \varphi(t-n)\rangle \\ c_{m,n} &= \langle t^m, \tilde{\varphi}(t-n)\rangle\end{aligned} \tag{15}$$

As mentioned in the analysis of Exercise 3, one of the assumptions for our entire algorithm to work is that our sampling kernel $\varphi(t)$ is able to reproduce polynomials of maximum degree $N \geq 2K-1$. Given that in this scenario we use $K = 2$, we need $N \geq 3$. Therefore, we use the *db*4 scaling function, since, as explained in Exercise 1, it has 4 vanishing moments and can thus reproduce polynomials up to $N = 3$. For an attempt at a more complete analysis, we also provide the sequence of equations below, which prove that the formula we compute in the first expression of Equation (15) is in fact the moments we are aiming to find.

$$\begin{aligned}\tau_m &= \sum_n c_{m,n} y_n \\ &= \left\langle x(t), \sum_n c_{m,n}\varphi(t-n)\right\rangle \\ &= \left\langle \sum_{k=0}^{K-1} a_k \delta\left(t-t_k\right), \sum_n c_{m,n}\varphi(t-n)\right\rangle \\ &= \int_{-\infty}^{\infty} \sum_{k=0}^{K-1} a_k \delta\left(t-t_k\right) t^m dt \\ &= \sum_{k=0}^{K-1} a_k t_k^m \quad m = 0, 1, \ldots, N\end{aligned} \tag{16}$$

The figures below show the results of our algorithm. The plots contain the signal $x(t)$, the true Diracs, as well as Diracs placed at the extracted locations and amplitudes. We can see that in most cases, such as the leftmost Figure, the true and extracted signals overlap perfectly with zero error. In this case the Diracs were placed at $t_k = [500, 1200]$. The results, however, have been observed to deviate the closer we place the Diracs of $x(t)$ to the beginning or the end of the x-axis range of length 2048. For example, in the center Figure $t_k = [100, 1200]$, while in the rightmost Figure $t_k = [500, 1800]$. In both cases we see that the error is non-zero.

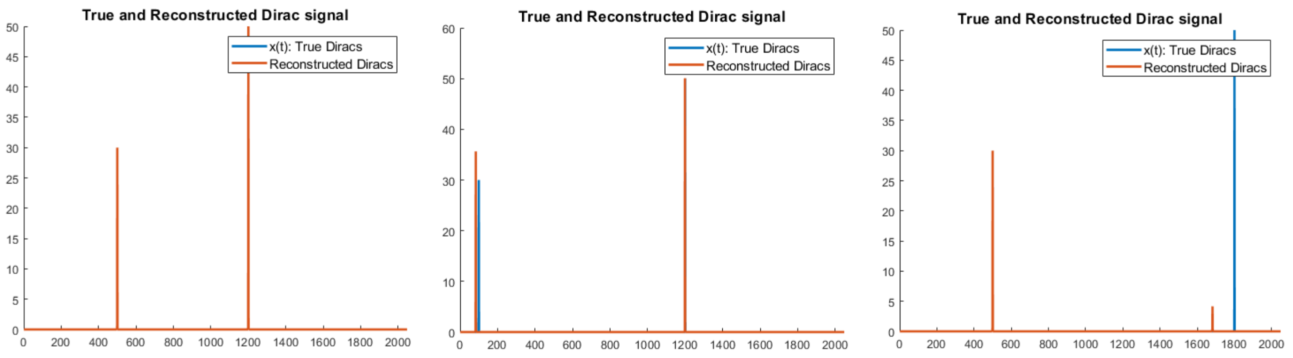


Figure 11: True and reconstructed Diracs

We ran another experiment to further investigate these results. In this experiment we kept the first Dirac at $t_1 = 1000$ while moving the second Dirac throughout the entire x-range in steps of 20. Therefore, the second Dirac would start at location $t_2 = 1$ and move in steps of 20 until it hit the end of the range (in this case, the final location was $t_2 = 2041$). Note that for all locations $a_1 = 30$ and $a_2 = 50$. For each iteration of this process we collected the extraction error in both t_k and a_k. This led to the plots in the Figure below.

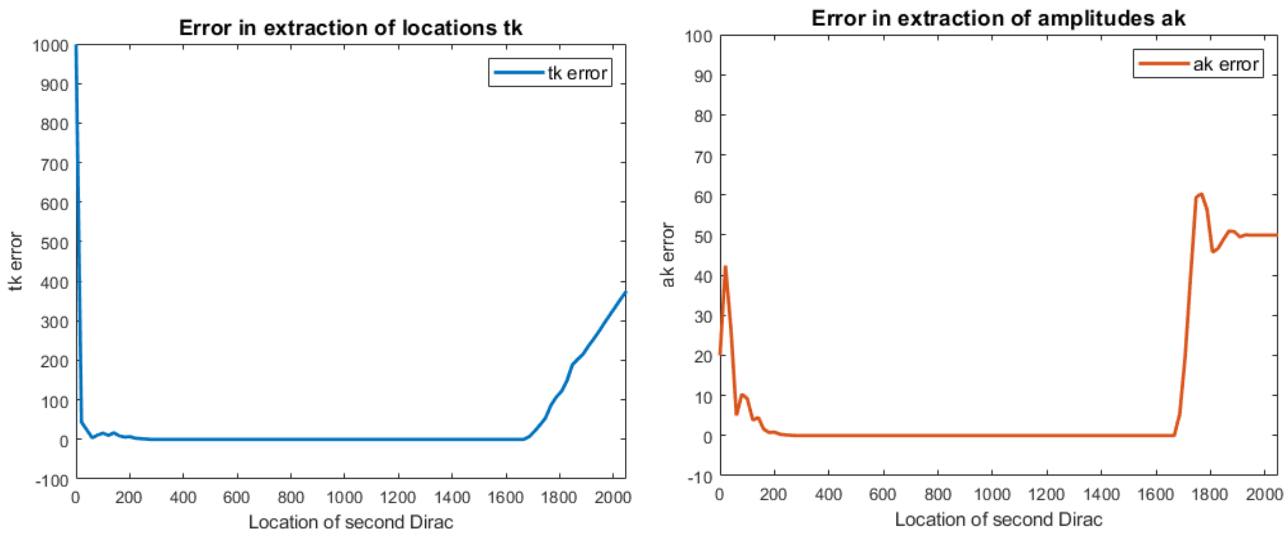


Figure 12: Extraction error of t_k and a_k based on Dirac position

These figures further enforce the claims above, namely that the error for the biggest part of the range is zero for both the locations and for the amplitudes, but increases towards the ends of the x-range.

5 Exercise 5

In Exercise 5 we are given the samples y_n as mentioned in Equation (15), and are prompted to recover the locations and amplitudes of the Diracs. Therefore, we use the exact same algorithm as in Exercise 4, but without constructing the samples y_n. We calculate the coefficients $c_{m,n}$, and subsequently multiply them with the samples provided, obtaining the moments τ_m. We then use the annihilating filter from Exercise 3 as we did previously to obtain the following locations t_k and amplitudes a_k.

t_k	a_k
1264	1.2580
1273	2.3650

Table 2: Extracted locations $\mathrm{t_k}$ and corresponding weights $\mathrm{a_k}$ for the given samples

6 Exercise 6

In this exercise we essentially replicate the algorithm developed in Exercise 4 but with two key differences. The first is that we sample the signal $x(t)$ with a different scaling function $\varphi(t)$. In this case, we require that $\varphi(t)$ can reproduce polynomials up to degree $N > 2K-1$, rather than the previous assumption of $N \geq 2K-1$. Thus, for $K = 2$, our Daubechies scaling function now has to be able to reproduce at least up to quartic, rather than cubic, polynomials. As explained before, this means that the wavelet must have at least 5 vanishing moments. The lowest order Daubechies scaling function that achieves this is the 5^{th} order Daubechies $db5$, so this is what we use.

The second difference that we implement in our algorithm from Exercise 4 is the addition of noise. This is does not happen until the end of the first step, namely on the moments array. Therefore, we construct the samples y_n and the coefficients $c_{m,n}$ with our new scaling function, and compute the moments s_m in the same way as before, exactly as shown in Equation (15). We then add zero-mean Gaussian noise ϵ_m on top of the moments, leading to the noisy moments vector $\hat{s}_m = s_m + \epsilon_m$.

The point of this change is to show that the annihilating filter method alone is not sufficient for correct extraction of the locations t_k and amplitudes a_k, and therefore correct reconstruction of the signal $x(t)$, when noise is added into the system. We show this further with the following experiment. The plots below have been generated by running the exact same process as the one that was run in Figure 8 of Exercise 4, but with different values for the variance of the noise. For conciseness and clarity, as well as to prove our point, we have used only 5 exponentially increasing values for the noise variance. We run this type of experiment because we believe it provides a lot more information than providing just one pair of t_k, a_k extracted with and without noise.

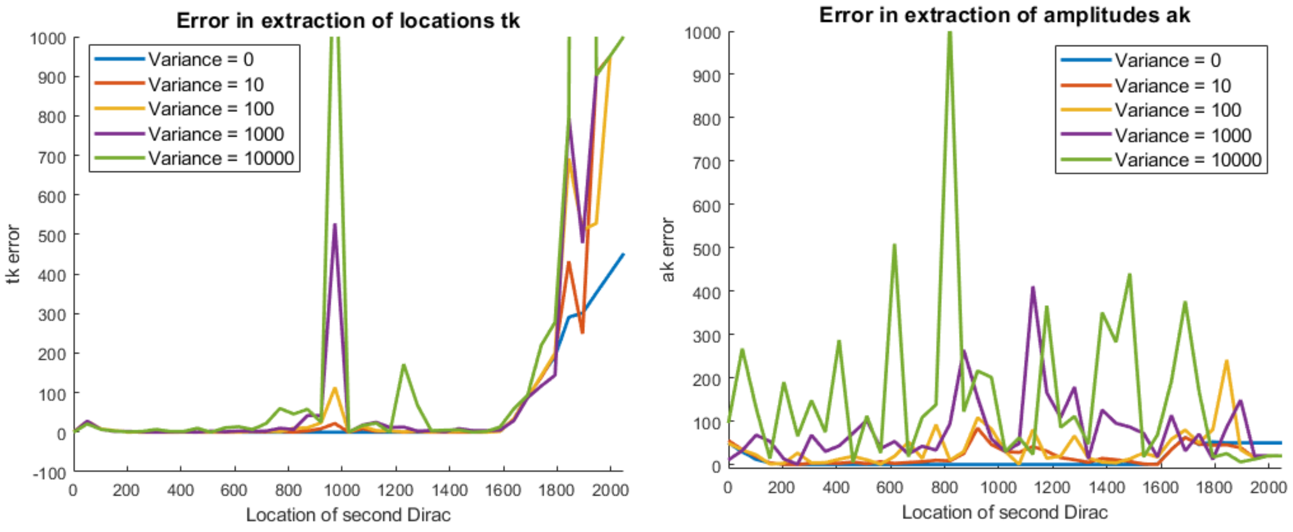


Figure 13: Extraction error of t_k and a_k based on Dirac position for different values of noise variance

While noise in our case is obviously completely stochastic and the plots above are just one instance that may occur, it is clear that there is a positive correlation between increasing noise variance and increasing extraction error. While this is more pronounced for the extraction of the amplitudes a_k, the effect is visible for the estimation of both quantities.

7 Exercise 7

Having generated noisy moments, in this exercise we aim to develop algorithms which are resilient to the existence of noise and can still reliably extract the locations and amplitudes of our Diracs.

7.1 Total Least Squares (TLS)

As we mentioned in Exercise 3, in the absence of noise the annihilating equation $\boldsymbol{T}H = 0$ is exactly satisfied. This, however, is not the case when we add noise to the system. Therefore, it is reasonable to try to develop a routine which approximates the original relationship. This is exactly what TLS attempts to achieve. In mathematical terms the TLS problem is the following:

$$\text{Minimize: } \|\mathbf{T}H\|_2 \text{ under the constraint } \|H\|_2 = 1$$

An automatic solution to this problem exists and starts by taking the Singular Value Decomposition (SVD) of the moments matrix $\boldsymbol{T} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^T$. Then H is given by the singular vector of $\boldsymbol{V}$ related to the smallest singular value of $\boldsymbol{T}$. If $\boldsymbol{\Lambda}$ is a diagonal matrix with decreasing positive elements, then H will be the last column of $\boldsymbol{V}$. Indeed, MATLAB returns the singular values in descending order and therefore we obtain H as the last column of $\boldsymbol{V}$. Note that this vector has norm 1 by construction of the SVD and also minimizes the required norm $\|\mathbf{T}H\|_2$. Having obtained H, we calculate its roots using the same method as before to obtain the t_k and a_k.

7.2 Cadzow's Algorithm

We can further improve our algorithm for cases with low SNR by using Cadzow's algorithm before applying TLS. The aim of Cadzow's algorithm is to attempt to denoise the moments to some amount using matrix operations. We begin by observing that in the absence of noise $\boldsymbol{T}$ is a Toeplitz matrix by construction. Further, we can prove that in that scenario it has rank K, where K is the number of Diracs, whereas in the noisy case $\boldsymbol{T}$ is full rank. The idea behind Cadzow's algorithm is that if we try to make our noisy matrix $\boldsymbol{T}$ simultaneously Toeplitz and rank K, as it would be in the noiseless case, then this process will have to some degree removed the noise. We do this mathematically with the following iterative process:

- **Step 1:** Obtain the SVD of $\boldsymbol{T} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{V}^T$
- **Step 2:** Keep the K largest singular values of $\boldsymbol{\Lambda}$ and set the rest to zero. Call the resulting diagonal matrix $\boldsymbol{\Lambda'}$.
- **Step 3:** Reconstruct $\boldsymbol{T'} = \boldsymbol{U}\boldsymbol{\Lambda'}\boldsymbol{V}^T$
- **Step 4:** Step 2 made the matrix rank K by construction. However, the matrix is not Toeplitz. We make it Toeplitz by averaging along the diagonal.
- **Step 5:** Iterate.

After some iterations of the Cadzow routine we run the TLS routine that we have already defined to obtain a hopefully more accurate extraction of the locations t_k and aplitudes a_k. We present our results in the following tables. For each value of the variance that we tried, we used the bare annihilating filter method, the TLS method, as well as the Cadzow + TLS method for 4 different scaling functions, $db5, db6, db7, db8$. The values that we tried for the variance were $[0.5, 2.5, 10, 30]$. These are indicated in the table caption for clarity. Also note that for all the runs the two Diracs had $t_k = [500, 1200]$ and $a_k = [30, 50]$. Finally, the iterations for the Cadzow routine were always set to 10.

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[500, 1200]	[500, 1200]	[500, 1200]	[442, 1195]
TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]
Cadzow + TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]

Table 3: Extraction of locations t_k for the three methods, variance=0.5

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[32.044, 49.149]	[31.159, 49.518]	[32.063, 49.141]	[27.280, 50.195]
TLS	[32.037, 49.156]	[31.158, 49.519]	[32.063, 49.141]	[28.656, 48.820]
Cadzow + TLS	[32.037, 49.156]	[31.158, 49.519]	[32.063, 49.141]	[28.656, 48.820]

Table 4: Extraction of locations a_k for the three methods, variance=0.5

---

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[500, 1200]	[500, 1200]	[500, 1200]	[442, 1195]
TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]
Cadzow + TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]

Table 5: Extraction of locations t_k for the three methods, variance=2.5

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[29.837, 50.064]	[28.303, 50.706]	[30.547, 49.771]	[29.647, 49.322]
TLS	[29.850, 50.050]	[28.303, 50.706]	[30.547, 49.771]	[31.141, 47.827]
Cadzow + TLS	[32.037, 49.156]	[31.158, 49.519]	[32.063, 49.141]	[28.656, 48.820]

Table 6: Extraction of locations a_k for the three methods, variance=2.5

---

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[500, 1200]	[500, 1200]	[500, 1200]	[442, 1195]
TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]
Cadzow + TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]

Table 7: Extraction of locations t_k for the three methods, variance=10

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[30.419, 49.828]	[31.684, 49.298]	[28.775, 50.509]	[27.904, 49.964]
TLS	[30.396, 49.851]	[31.683, 49.299]	[28.775, 50.509]	[29.311, 48.558]
Cadzow + TLS	[30.396, 49.851]	[31.683, 49.299]	[28.775, 50.509]	[29.311, 48.558]

Table 8: Extraction of locations a_k for the three methods, variance=10

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[500, 1200]	[500, 1200]	[500, 1200]	[442, 1195]
TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]
Cadzow + TLS	[500, 1200]	[500, 1200]	[500, 1200]	[478, 1195]

Table 9: Extraction of locations t_k for the three methods, variance=30

	*db*5	*db*6	*db*7	*db*8
annihilating filter	[40.937, 45.452]	[34.027, 48.322]	[39.267, 46.145]	[27.904, 51.805]
TLS	[40.892, 45.497]	[34.026, 48.323]	[39.267, 46.145]	[24.110, 50.645]
Cadzow + TLS	[40.892, 45.497]	[34.026, 48.323]	[39.267, 46.145]	[24.110, 50.645]

Table 10: Extraction of locations a_k for the three methods, variance=30

The results shown in the tables above are slightly surprising. We can make the following observations:
All three methods show approximately the same behaviour in extracting the locations t_k and are successful in doing so for the values of the variance that were tested. Indeed looking at Figure 13, we can see that for values of the variance below 100 the error in t_k extraction was negligible. We only see a degradation in performance when using the largest of the four scaling functions tested, *db*8, in which case, TLS and Cadzow+TLS gave consistently better results.

For variance levels of $0.5, 2.5, 10$, the accuracy in a_k extraction is relatively good for the annihilating filter. For a variance of 30, the error increases. Again, we can confirm this through Figure 13, where the extraction error is very small for variances less than 10 but increases afterwards. The part which surprised us was the observed improvement of TLS and Cadzow + TLS over the annihilating filter. This was found to be consistent throughout trials but very small, potentially even negligible. Even though the improvement appears to increase as we increase the noise variance, even in the maximum variance tested the improvement was only seen in the first decimal place. Further, performing Cadzow before TLS introduced extremely little benefit, only rarely improving the 4^{th} decimal.

Finally, we comment on the difference between scaling functions. As mentioned, this made a significant difference only in the case of *db*8 regarding the t_k. With regards to a_k, changing the scaling function appeared to produce random results with no consistent trend in improvement or degradation. Potentially this could be explained through the fact that a_k are calculated directly through the moments vector in the Vandermonde system on which there is still noise regardless of the method used. Therefore, even if accurate t_k are obtained, the a_k can still be heavily affected by the noise.

8 Exercise 8

This final exercise aims to apply some of the concepts taught in lectures and implemented during the previous parts of this coursework in a more realistic application: image super-resolution. More specifically, our task is concerned with the first of two steps for image super-resolution, image registration. The aim of image registration in this case is to estimate the geometric differences between the multiple Low Resolution (LR) images, so that they can then be overlaid together accurately into a High Resolution (HR) grid, which is finally then restored to obtain the super-resolved image. To obtain these geometric differences for each LR image with regards to a reference LR image we have to perform the steps as shown below.

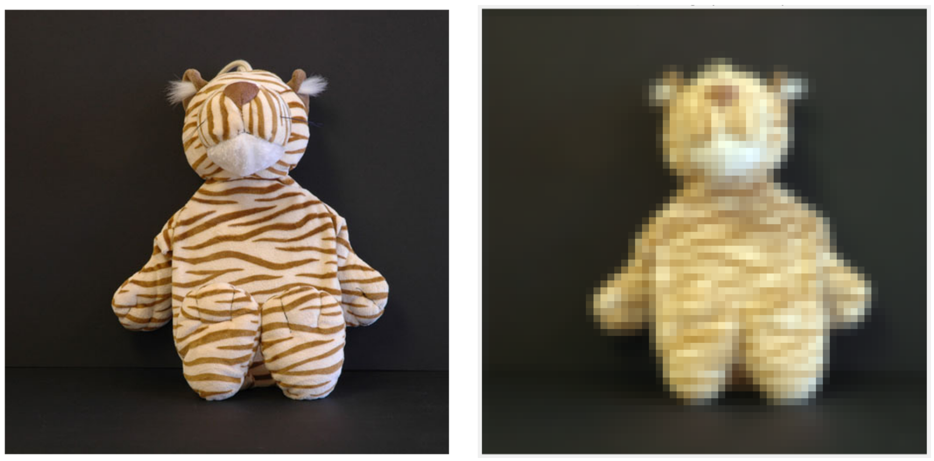


Figure 14: True HR image and example LR image

- **Step 1:** Obtain the samples. Once again, much like Exercise 5, we are given the samples $S^{(i)}_{m,n}$ as shown in Equation (17) below. The sampling kernel $\varphi(x, y)$ which filters the observed scene $f_i(x, y)$ is referred to as the point spread function of the camera and is modeled by a 2-D cubic B-spline.

$$S^{(i)}_{m,n} = \langle f_i(x, y), \varphi(x - m, y - n)\rangle \tag{17}$$

- **Step 2:** Obtain the coefficients. Similar to what we show in Equation (1) in Exercise 1 for the 1-D case, using different coefficients the splines can reconstruct polynomials, in this case, in two dimensions. These are also given for three cases: the 2-D spline reproducing a constant in both dimensions ($p = q = 0$, Coeff_0_0), and the 2-D spline reproducing a constant in one dimension and a linear in the other ($p = 1, q = 0$, Coeff_1_0 and $p = 0, q = 1$, Coeff_0_1). These can now be visualized as planes.

$$\sum_{m\in Z}\sum_{n\in Z} c^{(p,q)}_{m,n}\varphi(x - m, y - n) = x^p y^q \tag{18}$$

- **Step 3:** Construct the moments. As explained in Exercise 4 this is given by the sum of the products of the coefficients and the samples. Therefore, using the coefficient matrices which are given as mentioned above we can obtain $m_{0,0}, m_{1,0}, m_{0,1}$.

$$m_{p,q} = \sum_m \sum_n c^{(p,q)}_{m,n} S_{m,n} \tag{19}$$

- **Step 4:** Compute the barycenter of $f(x, y)$. We can do this by knowing the moments from above and computing the following:

$$(\bar{x}, \bar{y})^T = \left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}}\right)^T \tag{20}$$

- **Step 5:** Obtain the geometric differences. By calculating the difference between the barycenter of an image with regards to a reference image, we can compute the geometric difference between that image and the reference image.

$$(dx_i, dy_i)^T = (\bar{x}_i, \bar{y}_i)^T - (\bar{x}_1, \bar{y}_1)^T \tag{21}$$

Having outlined and implemented the procedure for obtaining a super-resolved image, we need to define thresholds for each of the RGB layers of the images, below which the samples will be set to zero. We do this to limit the noise created by the background of the image, which will allow us to obtain accurate moments and ultimately a better (in the PSNR sense at least) super-resolved image.

We estimated the optimal threshold for each layer by trying to maximize the PSNR by using an iterated grid search approach. Since all samples have been scaled to $[0, 1]$, for each of the RGB layers we initially checked the values $[0.1, 0.3, 0.5, 0.7, 0.9]$. This led to $5^3 = 125$ distinct PSNR values with a search resolution of 0.2. We observed that the PSNR was higher in the middle of the range and so we then repeated the process but with possible values $[0.3, 0.4, 0.5, 0.6, 0.7]$. So we again got 125 PSNR values but with a resolution of 0.1. Again, we observed the areas for each layer where the PSNR was noticeably higher and iterated. We continued on to a search resolution of 0.05 and finally 0.01. From the PSNR values of the final search we chose the largest and noted the thresholds for which this occurred. In the figure below we show one super-resolved image with all three threshold equal to zero, as well as one super-resolved image with the optimal thresholds we found with our search. We provide the values of the thresholds as well as corresponding PSNR in the table below.

	No Threshold	Optimal Threshold
red_threshold	0.00	0.61
green_threshold	0.00	0.45
blue_threshold	0.00	0.29
PSNR	17.262dB	24.704dB

Table 11: Thresholds for RGB layers and corresponding PSNR

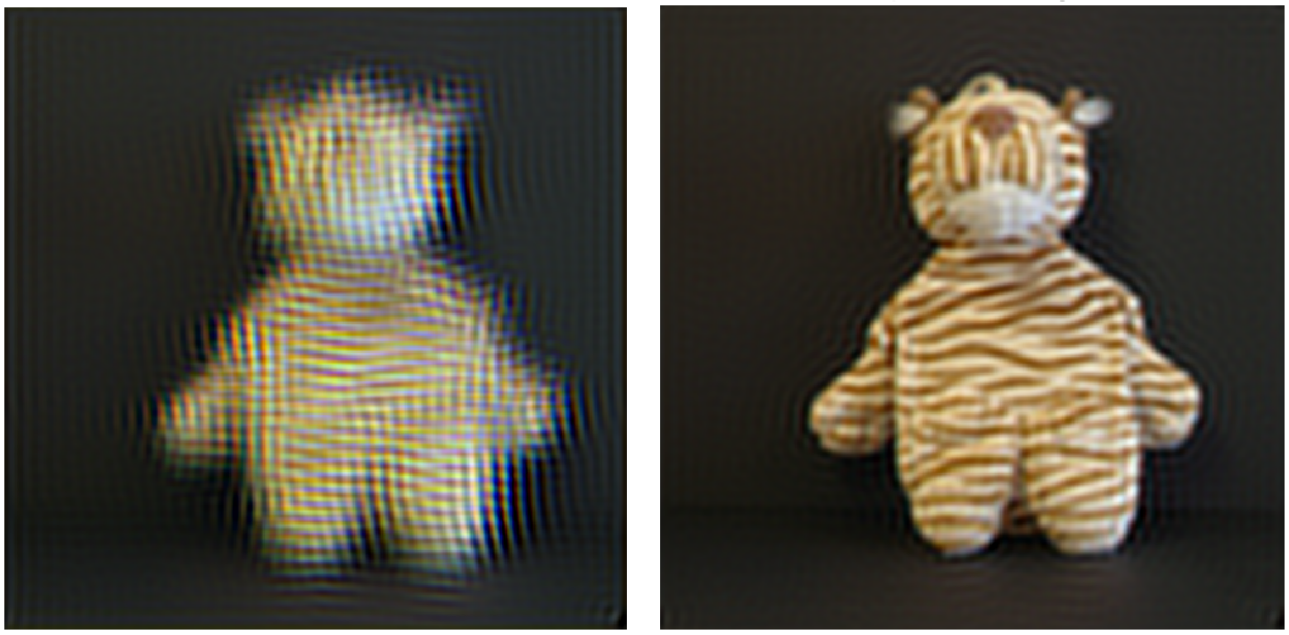


Figure 15: Super-resolved image with no thresholds (left) and with optimal threshold (right)