2. MSRI Graduate Summer School Notes
Jeff Blanchard, 2018
Discrete Orthogonal Systems and Sparse Representations

A continuous function can represent an analogue signal while a vector in $\mathbb{R}^n$ can represent a discrete or digital signal. In 2009 all television stations ceased broadcasting analogue signals and now exclusively transmit digital signals? This was mandated by Congress. Why? Do you remember analogue TV? The difference in quality in the DTV signals and the analogue signals was considerable. In fact, these same stations started broadcasting extra channels. By transmitting digital signals, they can send out a larger quantity of content with higher quality images. You might also recall a time, or have at least heard your parents describe a time, when people used to get film developed from a camera! Now everyone, even professional photographers, utilize digital imagery. The reason for all of this is the processing power of microchips (think Noyce '49) and numerical linear algebra (think you).

An analogue signal is converted to a digital signal in a number of ways. Mathematically, we model this as sampling the function at a finite number of discrete points. For example, if we want to plot the sine function, we can simply take 1000 evenly spaced points $x_i \in [0, 2\pi]$, compute $y_i = \sin(x_i)$, and plot the points (x_i, y_i) in the plane. Even a linear interpolation of these points will look like a nice smooth curve. Try it!

So, the moral of this introduction is that we live in a digital world and this is very powerful. The mathematics of the analogue world is beautiful and rich but the direct applications are shrinking in signal processing. However, the discrete mathematics of the digital world are messy and often complicated. The discrete transforms we discuss here are all derived from the continuous (analogue) spaces from which we sample.

2.1. Discrete Fourier Transform.

The Discrete Fourier Transform (DFT) is our finite dimensional equivalent to the Fourier transform. A function's Fourier Series is given by

$$f(x) = \lim_{N\to\infty} \sum_{n=-N}^{N} c_n e^{2\pi i n x}; \quad c_n =< f(x), e^{2\pi i n x} > .$$

Definition 2.1. A complex number z is an nth root of unity if $z^n = 1$. A root of unity is a *primitive* nth root of unity if it is an nth root of unity but is not a kth root of unity for any $k < n$.

Exercise 2.1. Show that $\omega_n = e^{-2\pi i/n}$ is a primitive nth root of unity.

Exercise 2.2. Prove the following identity: Let ω be a primitive nth root of unity and let k be any integer. Then

$$\sum_{j=0}^{n-1} \omega^{jk} = \begin{cases} n & \text{if } k/n \text{ is an integer} \\ 0 & \text{otherwise} \end{cases}.$$

(Hint: What type of sum are we dealing with? What if it were a series?)

For simplicity of our notation in this subsection, we will temporarily redefine how we index a vector. We will use zero-based indexing, namely

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} \in \mathbb{R}^n.$$

Definition 2.2 Discrete Fourier Transform (DFT). The DFT of a vector $x = [x_0, x_1, \ldots, x_{n-1}]^T \in \mathbb{R}^n$ is the vector $y = [y_0, y_1, \ldots, y_{n-1}]^T \in \mathbb{C}^n$ where $\omega = e^{-2\pi i/n}$ and

$$y_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j \omega^{jk}.$$

This is often better described as a matrix multiplication. The normalization by $\frac{1}{\sqrt{n}}$ is not always standard, but we will use it in our discussion in order to impose that the transform matrix is unitary. Define the $n \times n$

Discrete Fourier matrix F_n as

$$F_n = \frac{1}{\sqrt{n}} \begin{bmatrix} w^0 & w^0 & w^0 & \dots & w^0 \\ w^0 & w^1 & w^2 & \dots & w^{n-1} \\ w^0 & w^2 & w^4 & \dots & w^{2(n-1)} \\ w^0 & w^3 & w^6 & \dots & w^{3(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w^0 & w^{n-1} & w^{2(n-1)} & \dots & w^{(n-1)^2} \end{bmatrix}. \tag{1}$$

Exercise 2.3. Show that if y is the DFT of $x \in \mathbb{R}^n$ according to Definition 2.2, then $y = F_n x$.

Exercise 2.4. Prove that F_n is unitary.

The DFT of a real valued vector can certainly be complex. Now, if we actually construct the matrix F_n and perform the matrix multiplication, this requires $O(n^2)$ operations. Cooley and Tukey developed the Fast Fourier Transform (FFT) which performs the same transformation in $O(n \log_2 n)$ operations, a considerable savings. (Let $n = 2^{20}$ and compare n^2 to $n \log_2 n$.) The importance and role of the FFT in computational science can not be understated. If you are unfamiliar with the Cooley-Tukey algorithm for the FFT, you should look it up and implement it yourself (when you get home). We'll use built-in functions.

The MATLAB command[3] to compute a DFT using the FFT is `fft`. In MATLAB, our $1/\sqrt{n}$ scaling is not employed so the FFT is not a unitary transform matrix. Instead, if we let $M_n = \sqrt{n} F_n$ so that M_n is the unscaled transform matrix (1), then $y = \texttt{fft}(x)$ computes $y = M_n x$. The inverse transform, $x = \texttt{ifft}(y)$ is then computed via the matrix

$$M_n^{-1} = \left(\sqrt{n} F_n\right)^{-1} = \frac{1}{\sqrt{n}} F_n^* = \frac{1}{n} M_n^* = \frac{1}{n} \overline{M_n}.$$

Again, other than a scaling, this has no real implication for applications.

2.2. **Discrete Wavelet Transform.** In general, every wavelet system has a special function, the *scaling function* φ, and a wavelet function ψ that satisfy two refinement equations

$$\varphi(x) = \sum_{k \in \mathbb{N}} a_k \varphi(2x - k) \text{ and } \psi(x) = \sum_{k \in \mathbb{N}} b_k \varphi(2x - k)$$

which also give rise to low and high pass filters $h(\omega)$ and $g(\omega)$ via the continuous Fourier Transform. The discrete equivalents of these filters describe how to process the data via the wavelet transform. The simplest wavelet is a piecewise constant function call the Haar Wavelet. It satisfies the refinement equations

$$\varphi(x) = \varphi(2x) + \varphi(2x - 1) \text{ and } \psi(x) = \varphi(2x) - \varphi(2x - 1).$$

The associated low and high pass filters for the Haar Wavelet are $h(\omega) = \frac{1}{2}\left(1 + e^{-i\omega}\right)$ and $g(\omega) = \frac{1}{2}\left(1 - e^{-i\omega}\right)$.

2.2.1. *Discrete Haar Transform.* For the Haar wavelet, the low pass filter h is a trigonometric polynomial with coefficients $\frac{1}{2}$. This tells us that to process a signal $x \in \mathbb{R}^n$ via the low pass filter by averaging every two adjacent terms. To fully develop this mathematically requires the notion of convolution. Instead, I will describe the operation in its raw form. Since the filter has length two, we pass the filter along the signal and apply the coefficients to the elements of x. Let us call this operator H. Then,

$$H(x) = \left[\frac{1}{2}x_1 + \frac{1}{2}x_2, \frac{1}{2}x_3 + \frac{1}{2}x_4, \dots, \frac{1}{2}x_{n-1} + \frac{1}{2}x_n\right] \in \mathbb{R}^{n/2}.$$

Now the high pass filter g is a trigonometric polynomial with coefficients $\frac{1}{2}$ and $-\frac{1}{2}$. The operator G takes differences of every two adjacent elements in x:

$$G(x) = \left[\frac{1}{2}x_1 - \frac{1}{2}x_2, \frac{1}{2}x_3 - \frac{1}{2}x_4, \dots, \frac{1}{2}x_{n-1} - \frac{1}{2}x_n\right] \in \mathbb{R}^{n/2}.$$

[3]Python users should figure out the FFT too including how it is scaled.

So, after a single pass through the filters H and G we can construct a vector in $\mathbb{R}^n$ by concatenating $H(x)$ and $G(x)$ so that

$$\begin{aligned} W_1(x) &= [H(x), G(x)] \\ &= \frac{1}{2}\left[x_1 + x_2, x_3 + x_4, \ldots, x_{n-1} + x_n, x_1 - x_2, x_3 - x_4, \ldots, x_{n-1} - x_n\right]. \end{aligned}$$

Notice that we can easily recover the entire vector from this new vector since

$$W_1(x)_i = \frac{1}{2}\left(x_{2i-1} + x_{2i}\right),\ W_1(x)_{\frac{n}{2}+i} = \frac{1}{2}\left(x_{2i-1} - x_{2i}\right),\ i = 1, 2, \ldots, n/2$$

is a fully determined system of equations. Therefore, after one pass through the filters, we have a filter of averages and differences which completely define the original signal.

Subsequent passes through a filter only process the portion of the signal which has been passed though the low pass filter. So, after the second pass, we would have

$$\begin{aligned} W_2(x) &= [H(H((x)), G(H(x)), G(x)] \\ &= \frac{1}{2}\left[\frac{x_1 + x_2 + x_3 + x_4}{2}, \ldots, \frac{x_{n-3} + x_{n-2} + x_{n-1} + x_n}{2}, \frac{(x_1 + x_2) - (x_3 + x_4)}{2}, \ldots,\right. \\ &\qquad \left. \ldots, \frac{(x_{n-3} + x_{n-2}) - (x_{n-1} + x_n)}{2}, x_1 - x_2, x_3 - x_4, \ldots, x_{n-1} - x_n\right]. \end{aligned}$$

Again, reconstruction is possible since we can first apply the reconstruction scheme to the first half of $W_2(x)$ to recover $W_1(x)$ and then apply the reconstruction scheme to recover x from $W_1(x)$. If, $x \in \mathbb{R}^n$ and $n = 2^p$ for an integer p, we can apply the filtering process p times to obtain the full p scale wavelet transform

$$W(x) = W_p(x) = \left[H^p(x), G(H^{p-1}(x)), G(H^{p-2}(x)), \ldots, G(H^{p-r}(x)), \ldots, G(H(x)), G(x)\right] \in \mathbb{R}^n.$$

Notice that after p iterations, we have that $W(x)_1$ is simply an average of all the elements in x, the second entry is the difference between the first half of the vector and the second half of the vector, the next two entries are the difference of adjacent fourths of the vectors, and so on.

Exercise 2.5. Compute $W(x)$ for the vector $x = [1, 1, 2, 2.05, 2.95, 3, 3.05, 3.05]$. (After the first iteration you should have $W_1(x) = [1, 2.025, 2.975, 3.05, 0, -0.025, -0.025, 0]$.)

Notice that the first vector had several "big" coefficients while the wavelet transform did not. In fact, if you decide that you are not concerned with coefficients with magnitudes smaller than 0.05, then your reconstruction process will get to a point where $\hat{W}_1(x) = [1, 2.025, 2.975, 3.05, 0, 0, 0, 0]$. So, if we represent our original signal in the wavelet basis, we have half as many nonzero entries.

Exercise 2.6. Reconstruct the vector $\hat{x}$ from $\hat{W}_1(x)$. Does it differ significantly from x? (Use a plotting tool to really see the difference.)

In this formulation, the Haar Wavelet transform is not unitary in that it is altering the norm of the vector in each iteration. Fortunately, there is an easy fix to the problem; we simply need to normalize the operators by altering the coefficients to $\frac{1}{\sqrt{2}}$. In other words, if H is the vector of coefficients in $h(\omega)$, we want to normalize H so that $\|H\|_2 = 1$.

2.2.2. *Daubechies Wavelets.* If you don't know who Ingrid Daubechies is, you should find out as soon as possible. She has done many things, but is most famous for finding compactly supported, smooth wavelets. Daubechies wavelets have finite low pass filters which are trigonometric polynomials. The fact that the filters are finite is due to the fact that they have compact support in the time domain; they are also smooth enough to have excellent frequency localization. The Haar wavelet is the simplest wavelet in the class of Daubechies wavelets and has two nonzero coefficients. The next Daubechies wavelet has four nonzero coefficients in the filters and is usually called the D_4 (or Daubechies 4 tap wavelet). The number of coefficients in the finite trigonometric polynomial filter is directly related to the smoothness of the associated system. The low and

high pass filters for D_4 are

$$h(\omega) = \alpha + \beta e^{-i\omega} + \gamma e^{-2i\omega} + \delta e^{-3i\omega}$$
$$g(\omega) = \delta - \gamma e^{-i\omega} + \beta e^{-2i\omega} - \alpha e^{-3i\omega}$$
$$\alpha = \frac{1+\sqrt{3}}{4\sqrt{2}},\ \beta = \frac{3+\sqrt{3}}{4\sqrt{2}},\ \gamma = \frac{3-\sqrt{3}}{4\sqrt{2}},\ \delta = \frac{1-\sqrt{3}}{4\sqrt{2}}.$$

The coefficients for D_n for any even n can be found online. Try searching for them. You do have to realize that not everyone uses the same scaling of the coefficients. The way we have written the coefficients above, the DWT matrix will be unitary, but this is not always the convention.

It is possible to define a discrete wavelet transform in general for any length of finite filter and therefore for all Daubechies wavelets. The greater the number of coefficients, the smoother the wavelet, the better the wavelet system at sparsely representing vectors. There is much to know about wavelets, but the main point is that they are great at sparse representation and also have fast discrete transforms.

2.3. **Sparse Representations.** In the previous sections we have developed three bases for vectors in $\mathbb{R}^n$, the discrete Fourier basis, the Haar wavelet basis, and the Daubechies 4 wavelet basis. Why would we need so many different bases. The following plots show different sparse approximations for some relatively simple functions. The following process develops plots showing a 95% compression in the sense that the number of nonzeros in the vector representation for each basis is only 5% the number of nonzeros in the original vector. These plots were generated with the following process:

1. 256 samples of the function are taken on $[-1, 1]$.
2. The sample vector is transformed into each of the bases.
3. The magnitude of the 95th percentile coefficient, T_{95}, is computed.
4. All coefficients whose magnitude is smaller than T_{95} are set to zero (hard thresholding).
5. A sparse approximation to the original signal is reconstructed by inverting the thresholded coefficient vector.
6. The original function is plotted with all three sparse approximations.

It is clear from these figures that certain functions have a nice, sparse approximation in some basis even when the other bases fail to accurately approximate the original function. In all cases, however, the sparse approximations still have the same generic "shape" as the original function. The functions under consideration are

$$f_1(x) = \begin{cases} 1-|x| & x \in [-1,1] \\ 0 & \text{otherwise} \end{cases}$$
$$f_2(x) = 13x^3 + 8x$$
$$f_3(x) = \begin{cases} \frac{1}{2}(x+x^2) & x \in [-1,0) \\ \frac{1}{2}(x-x^2) & x \in [0,1] \\ 0 & \text{otherwise} \end{cases} \tag{2}$$
$$f_4(x) = \begin{cases} 1 & x \in [-1,-1/4) \\ 2 & x \in [-1/4,1/4) \\ -\frac{1}{2} & x \in [1/4,1/2] \\ -1 & x \in [1/2,1] \\ 0 & \text{otherwise} \end{cases}$$

Let x denote the original signal and let the sparse approximations be x_f, x_h, x_d for the FFT, Haar Wavelet, and D_4 reconstructions. In Fig. 1, we have a visual interpretation of the accuracy of approximation. However, it is also useful to consider the approximation error which we present in Table 1 in terms of the ℓ_2 norm.

Function	$\|x - x_f\|_2$	$\|x - x_h\|_2$	$\|x - x_d\|_2$
f_1	0.0444	0.7659	0.0472
f_2	55.4721	14.2418	7.5374
f_3	0.0044	.1536	0.0370
f_4	4.5851	0.000	2.9185

TABLE 1. The ℓ_2 error of the sparse approximations x_f (FFT), x_h (Haar), and x_d (Daubechies 4) compared with the samples x from the functions defined by (2).

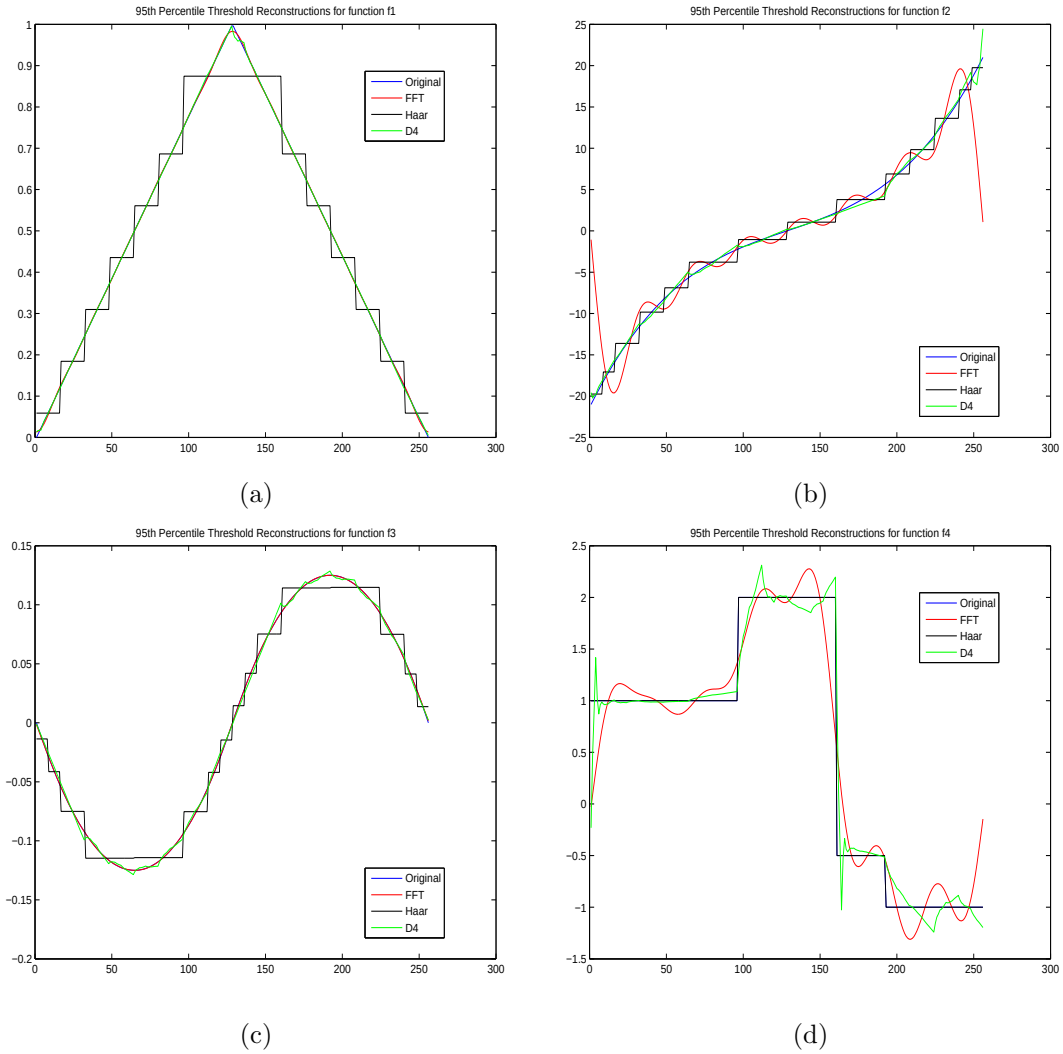

FIGURE 1. Sparse Approximation with roughly 5% as many coefficients as the original function samples from the standard basis. (a) f_1, the tent function, (b) f_2, a cubic, odd polynomial, (c) f_3, a piecewise defined, quadratic with one continuous derivative, and (d) f_4, a step function.

2.4. **Additional FFT and Haar Wavelet Exercises.**

Exercise 2.7. Prove that the Inverse Discrete Fourier Transform (IDFT) of $y \in \mathbb{C}^n$ is the vector $x = F_n^* y$. Observing that F_n is also symmetric, develop a computational efficient method for computing $x = F_n^* y$. (Hint: the efficiency is obtained by conjugating vectors rather than matrices.)

From the previous discussion, it is clear that we want to make a unitary transform matrix W so that Wx is the full wavelet transform of x. For the Haar wavelet, this is straightforward. The first entry should be the average of all elements, the second entry the difference of the two halves of the vector, the third and fourth entry the difference of the first and second quarters and the third and fourth quarters, respectively, and so on.

Exercise 2.8. In MATLAB, create a function haarmatrix.m that takes a dimension of a vector n, and creates an $N \times N$ matrix W where $N = 2^p \geq n > 2^{p-1}$. The first row should be all ones, the second row should have all ones in the first half with -1 in the second half, the third row should have 1 in the first fourth, -1 in the second fourth, and zeros elsewhere; the fourth row should have zeros in the first half, 1 in the third fourth, -1 in the last fourth, etc. Notice that you can build the first row and the second row, then subsample the second row by selecting every other element and place this is the appropriate location for the third and fourth rows. Repeat the subsampling and proper placement in rows 5 through 8, then again in 9 through 16, again in 17 through 32, etc.

Exercise 2.9. The matrix you just created is not unitary, but fear not. We simply need to normalize all of the rows. The naive method would be to build the matrix as above, compute the norm of each row, and scale the rows. But, we already know what the norm of the rows will be so we should just make a minor adjustment to our code from the previous exercise. Determine the appropriate scaling of each row and alter your code so that after subsampling, you scale the coefficients you are assign to the appropriate shifts and placement in the matrix.

Exercise 2.10. Create a function which computes the Haar Wavelet transform a vector $x \in \mathbb{R}^n$.

Exercise 2.11. Since your matrix is unitary, it is easy to compute the inverse. Write a function that computes the inverse Haar wavelet transform of a vector $w \in \mathbb{R}^{2^p}$.

Exercise 2.12. Find a sound file, load it, perform a Haar wavelet transform, retain only the top 50% of the coefficients, and apply the inverse Haar transform. Compare the two sound files. (Matlab has handel.mat in the examples so that `load handel.mat` will put a vector y in your workspace so that the command `sound(y)` would play Handel's Messiah.)

Exercise 2.13. Suppose you had signals that were all the sum of a piecewise constant function and a few different sinusoids with different periods. These certainly would not be sparse in the Haar basis nor in the Fourier basis. Can you think of a set of functions that might not be a basis, but would be very good at sparsely representing these types of functions?