

Notes from AMSC 661 05/05/05 (neat date!)
Notes from Dr. O'Leary's lecture on Transforms and Wavelets:

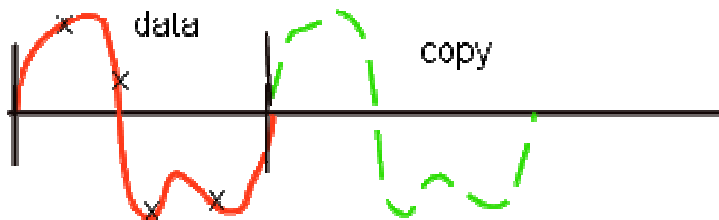
<http://www.cs.umd.edu/~oleary/c661/661fourierhand.pdf>
and a few slides from
<http://www.cs.umd.edu/~oleary/c460/old/460matrixhand.pdf>

Joanna R. Pressley

Last lecture we were introduced to the Continuous Fourier Transform in two forms, the Discrete Fourier Transform, the Discrete Sine Transform, the Haar Transform, and the Discrete Haar Transform. Here **are a few additional points about the Transforms.**

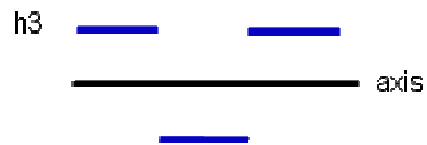
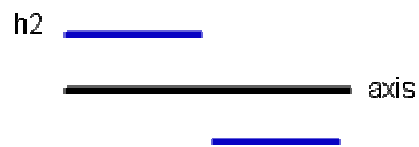
Fourier Transform - $f(x)$ must go to zero fast enough so that $F(s)$ does not go to infinity. Also, x can be complex.

Discrete Fourier Transform – we are using a discrete set of x , but we are assuming that the true function continues outside of our set and is just copies of the set. In this way, we are assuming that the function is periodic.



Discrete Sine Transform - When we use the DST we are also assuming the data is periodic. It is good to use the DST when the data starts at zero and ends at zero (zero boundary conditions), like the picture above. Also, remember $e^{2\pi i s x / n} = \cos(2\pi s x / n) + i \sin(2\pi s x / n)$. This helps us understand that the DST comes from the imaginary part of the Fourier Transform. The DST allows you to work with real numbers and real arithmetic, where as the Fast Fourier Transform (which we'll get to later) uses complex.

Discrete Haar Transform – We contract $h(x)$ to get the h_i functions. You get more oscillations on $[0,1]$ as you contract more.



An important property of the Fourier Transform –

When functions are periodic – $f(x+p) = f(x)$, then $F(s)$ will almost always be zero. Then we can write the integral as a sum.

$$f(x) = \sum_{k=-\infty}^{\infty} F[k]e^{2\pi kx/p}.$$

Now we move on the fast methods. Regularly, it would take $O(n^2)$ computations to compute the discrete transforms. These methods help us compute the discrete transforms with much fewer computations. They were first developed by Gauss and rediscovered by Cooley and Tukey. We will try to understand the algorithms using a matrix factorization representation.

Fast Methods (mostly from the 460 notes)

Let's start with the **Fast Fourier Transform (FFT)**. We want to compute:

$$F(s) = \sum_{x=0}^{n-1} f(x)e^{-2\pi isx/n}$$

for $s = 0, \dots, n-1$.

We can write this as a matrix-vector product. For example, let $n = 4$. Then we can write it as $F_4 * f(x) = F(s)$ where $f(x)$ is the vector $[f(0) f(1) f(2) f(3)]'$, $F(s)$ is the vector $[F(0) F(1) F(2) F(3)]'$ and F_4 is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4 & \omega_4^2 & \omega_4^3 \\ 1 & \omega_4^2 & \omega_4^4 & \omega_4^6 \\ 1 & \omega_4^3 & \omega_4^6 & \omega_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

where

$$\omega_4 = e^{-2\pi i/4} = -i,$$

In general,

$$\omega_n = e^{-2\pi i/n}$$

and $i = \sqrt{-1}$. Thus, $\omega = \cos(2\pi/n) - i \sin(2\pi/n)$

F_4 comes from plugging in the values for s and x into $e^{-2\pi isx/n}$. For instance, the first element in the matrix is when $s=0$ and $x=0$. The second element in the first row is when $s=0$ and $x=1$. The second element in the first column is when $s=1$ and $x=0$.

So $F_4(k,j) = e^{2\pi i k j / n}$.

The matrix is symmetric, so to compute the IDFT you just multiply by the transpose of the matrix.

When n gets large, the matrix is too big to store. So we use the fast methods to lessen storage and computation time. To see how this works, let's look at a large problem. Let $n = 8$.

Let's define z :

$$z \equiv \omega_8 = e^{-2\pi i/8}$$

we know the following 2 facts

$$z^2 = \omega_4.$$

$$z^8 = 1.$$

Let's look at the entire matrix for F_8

$$F_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & z & z^2 & z^3 & z^4 & z^5 & z^6 & z^7 \\ 1 & z^2 & z^4 & z^6 & 1 & z^2 & z^4 & z^6 \\ 1 & z^3 & z^6 & z^1 & z^4 & z^7 & z^2 & z^5 \\ 1 & z^4 & 1 & z^4 & 1 & z^4 & 1 & z^4 \\ 1 & z^5 & z^2 & z^7 & z^4 & z & z^6 & z^3 \\ 1 & z^6 & z^4 & z^2 & 1 & z^6 & z^4 & z^2 \\ 1 & z^7 & z^6 & z^5 & z^4 & z^3 & z^2 & z^1 \end{bmatrix}$$

We can rearrange the columns as long as we also rearrange $f(x)$ and $F(s)$ too. Move the odd columns (purple) to the front in order and the even columns to the back. Then we get

$$F_8^{reordered} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & z^2 & z^4 & z^6 & z & z^3 & z^5 & z^7 \\ 1 & z^4 & 1 & z^4 & z^2 & z^6 & z^2 & z^6 \\ 1 & z^6 & z^4 & z^2 & z^3 & z & z^7 & z^5 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & z^2 & z^4 & z^6 & -z & -z^3 & -z^5 & -z^7 \\ 1 & z^4 & 1 & z^4 & -z^2 & -z^6 & -z^2 & -z^6 \\ 1 & z^6 & z^4 & z^2 & -z^3 & -z & -z^7 & -z^5 \end{bmatrix} = \begin{bmatrix} F_4 & DF_4 \\ F_4 & -DF_4 \end{bmatrix}$$

where

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & z & 0 & 0 \\ 0 & 0 & z^2 & 0 \\ 0 & 0 & 0 & z^3 \end{bmatrix}$$

The top blue block is F_4 and is repeated underneath. Then the second top block is F_4 multiplied by the matrix D . The bottom right block is just the block above it multiplied by minus 1. So storage is much reduced.

So if we were multiplying F_8 by a vector x , we would need to rearrange x also to use this algorithm. Then we would get:

$$\begin{aligned}
 F_8 x &= F_8^{reordered} x^{reordered} \\
 &= \begin{bmatrix} \mathbf{F}_4 & D\mathbf{F}_4 \\ F_4 & -D\mathbf{F}_4 \end{bmatrix} \begin{bmatrix} x(1:2:8) \\ x(2:2:8) \end{bmatrix} \\
 &= \begin{bmatrix} I & D \\ I & -D \end{bmatrix} \begin{bmatrix} F_4 x(1:2:8) \\ F_4 x(2:2:8) \end{bmatrix}.
 \end{aligned}$$

Due to the regrouping above, we have lessened the computation cost greatly! We now only do 2*the cost for $n = 4$ plus 12 multiplications (we count for complex arithmetic).

If we start with $n = 128$, we can reduce that to 2*the cost of $n = 64$, which we can reduce to 2*the cost of $n = 32$, which we can reduce to 2*the cost of $n = 16$, which we can reduce to 2*the cost of $n = 8$, which we just saw we could reduce to 2*the cost of $n = 4$ (all of this plus some additions).

For $n = 128$, the slow FT costs 214665 flops ($O(n^2)$). For the FFT the cost is 5053 flops ($O(n \log_2(n))$). This is the radix 2 FFT. It works for powers of 2. If n is not a power of 2 there are other fast algorithms. The work is approximately $O(np)$ where p is small if n has small prime factors.

Fast Haar

Now let's look at fast methods for the Haar Transform. Let's define H_1 :

$$\mathbf{H}_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The Haar transform for a 2-dim x is $H_1 * x$. Now for powers of radix 2,

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{H}_{k-1} & \mathbf{H}_{k-1} \\ \mathbf{H}_{k-1} & -\mathbf{H}_{k-1} \end{bmatrix}$$

So for instance, H_2 is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

For H_1 , the first row is like h_1 , the second row is like h_2 . For H_2 the first row is like h_1 the second row is like h_4 , the third row is like h_2 , and the fourth row is also like h_2 but shifted.

Now let's look at H_3 :

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Here, row 4 and 7 are shifted versions of rows 3 and 5 respectively. So the rows are the Haar wavelets with a couple of strange functions added in. These Haar Transforms are examples of wavelet transforms. So let's look at wavelet transforms.

Wavelets

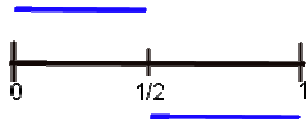
We want to decompose the function into frequency and locality, so it is a 2D breakup. Using Fourier analysis, we broke the signal up into its frequency components. With wavelets we will break the function "up into spatial components scaled to the frequency".

We will use the mother wavelet, the analyzing wavelet and the father wavelet, the scaling function.

The mother wavelet is

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1/2 \\ -1 & \text{if } 1/2 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

and it looks like:

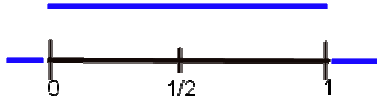


Important properties of the mother wavelet are that if you take the integral over the reals of the mother wavelet, this equals zero. Also, it has width 1.

The father wavelet is

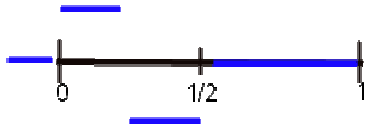
$$\phi(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

And it looks like:

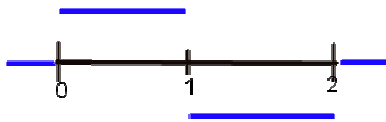


We can also look at Dilations and translations of the mother wavelet.

For instance $\psi(2x)$ has width $1/2$ and looks like:



and $\psi(x/2)$ has width 2 and looks like:



Next time we will be talking about wavelet expansions.