Lecture 18

Fast Fourier Transform

Overview

Using the Discrete Fourier Transform (DFT) we can represent an *N*-point signal $x(n)$ as a superposition of complex exponentials, or equivalently as a superposition of sinusoids.

However, on the order of N^2 operations are required to compute the DFT, and for large *N* this presents a substantial computational burden. What makes the DFT practical for DSP applications is that we can exploit the properties of its complex-exponential factors to reduce the computational requirements to on the order of $N \log_2 N$. The corresponding speed up is on the order of $N/\log_2 N$. For $N=1024$ this is about a factor of 100, a tremendous improvement. An algorithm for this more efficient calculation of the DFT is called a *Fast Fourier Transform* (FFT). In this lecture we develop the Cooley-Tukey FFT algorithm.

DFT computation requirements

The DFT of a signal $x(n), 0 \le n \le N-1$ is

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, 0 \le k \le N-1 \qquad (1)$$

Consider what is involved in calculating this. For each of the *N* values of *k*, and then for each of the *N* values of *n* we have to

- calculate $e^{-j\frac{2\pi}{N}kn} = \cos(2\pi kn/N) - j\sin(2\pi kn/N)$ (two trig function evaluations)
- perform the multiplication $x(n)e^{-j\frac{2\pi}{N}kn} = x(n)\cos(2\pi kn/N) - jx(n)\sin(2\pi kn/N)$ (which takes two real multiplications, assuming *x* is real)
- add the result to $X(k)$ (which takes two real additions, one for the real part and one for the imaginary part)

A brute-force calculation of (1) therefore requires approximately $2N^2$ trig function evaluations, $2N^2$ real multiplications, and $2N^2$ real additions.

We could, in principle, precalculate the trig functions and store the *N* distinct values of $e^{-j\frac{2\pi}{N}kn}$ in an array instead of calculating trig functions for every term of (1). But even then, we would still have $O(N^2)$ multiplications and additions to perform. In the appendix is a Scilab function `dft1()` that calculates (1) directly. For $N=1024$ it requires about 10 seconds to execute. Suppose this was applied to an audio signal with sampling frequency $F_s = 8\,\text{kHz}$. Since $1024/8000 = 0.128$ these 1024 samples would occupy a 128 ms window. If a 128-ms segment of data requires 10 sec to process there is no way we could use the DFT to perform realtime audio DSP with Scilab.

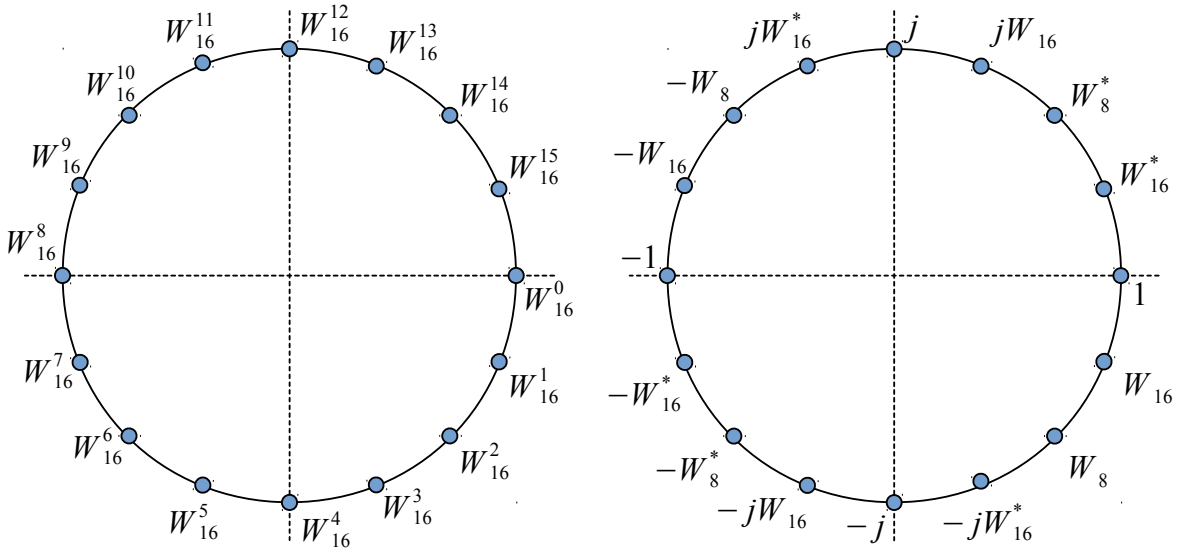

Fig. 1: The 16 distinct values of W_{16}^m can be expressed in terms of $1, j, W_{16}, W_8$.

Also in the appendix is a Scilab function `fft1()` that implements the FFT algorithm we will derive in this lecture. For $N=1024$ it requires only about 23 ms to execute. This is much less than the duration of the window, implying that we could use this routine for realtime DSP. It's the potential for such a dramatic speed up that we want to understand.

Properties of complex exponential factors

Using the short-hand notation

$$W_N = e^{-j\frac{2\pi}{N}} \tag{2}$$

(1) can be written

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \tag{3}$$

Since k and n are integers, $m = kn$ is also, and we can write $W_N^{kn} = W_N^m$. It's the special properties of the complex exponentials W_N^m that enable us to develop the FFT algorithm.

First let's see how many distinct values of W_N^m there are. Since

$$W_N^{m\pm N} = W_N^{\pm N} W_N^m = e^{-j\frac{2\pi}{N}(\pm N)} W_N^m = W_N^m \tag{4}$$

W_N^m is periodic in m with period N, so it has only N distinct values

$$W_N^m, 0 \le m \le N-1 \tag{5}$$

Because $\left|e^{j\phi}\right| = 1$ these N values fall on the unit circle of the complex plane (Fig. 1).

$$W_N^0 = 1, W_N^{N/2} = -1, W_N^{N/4} = -j \tag{6}$$

$$W_N^{2m} = W_{N/2}^m \tag{7}$$

$$\Delta\theta = \frac{2\pi}{N} = \frac{360^o}{N}$$

Key to the FFT is the fact that W_N^{kn} can always be expressed in terms of the factors W_N^k, $W_{N/2}^k$, $W_{N/4}^k$, etc. To do this we express n as a sum of powers of 2, separate factors of the form $W_N^{k2^p}$ and use

$$W_N^{k2^p} = W_{N/2^p}^k \tag{8}$$

For example, suppose we want to calculate W_{16}^{k7}. We write $7 = 4 + 2 + 1$ to obtain

$$W_{16}^{k7} = W_{16}^{k4} W_{16}^{k2} W_{16}^{k1} = W_4^k W_8^k W_{16}^k \tag{9}$$

$$\begin{aligned} z &= x + jy \\ z^* &= x - jy \\ -z^* &= -x + jy \end{aligned} \tag{10}$$

Cooley-Tukey algorithm

The first and probably still the most commonly applied FFT algorithm was published by Cooley and Tukey in 1965 during the early years of DSP development. We will limit consideration to the case where N is a power of 2. The algorithm can be extended to other values of N at the cost of greater complexity. To motivate the algorithm we will explicitly work out the cases $N = 2, 4, 8$ and then generalize to the case $N = 2^p$.

N = 2

For $N = 2$ (3) becomes

$$X(k) = x(0) + x(1) W_2^k \tag{11}$$

There are two distinct values of W_2^k,

$$W_2^0 = 1, W_2^1 = -1 \tag{12}$$

which can be expressed as

$$W_2^k = (-1)^k \tag{13}$$

Therefore (11) can be written

$$X(k) = x(0) + (-1)^k x(1) \tag{14}$$

and we have

$$\begin{aligned} X(0) &= x(0) + x(1) \\ X(1) &= x(0) - x(1) \end{aligned} \tag{15}$$

Let's call (15) the "2-point DFT." It can be represented as a signal-flow diagram as shown at left in Fig. 2. This operation is sometimes called a *butterfly* due to the appearance of its signal-flow diagram.

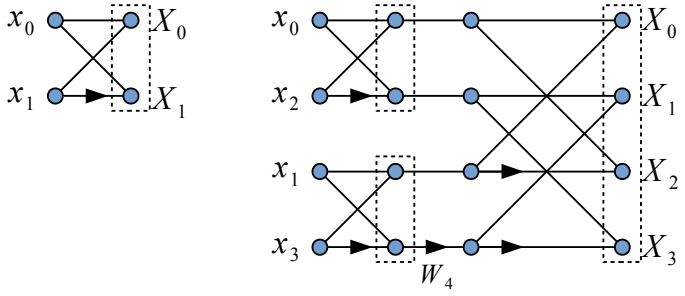

Fig. 2: 2-point (left) and 4-point (right) FFTs. Computation flows from left to right. Unlabeled arrows denote negation (multiplication by −1). Labeled arrows denote multiplication by the listed factor. Two branches joining at a node are summed.

N = 4

For $N=4$ (3) is

$$X(k)=x(0)+x(1)W_4^k+x(2)W_4^{k2}+x(3)W_4^{k3} \tag{16}$$

Decomposing the W factors as

$$\begin{aligned} W_4^{k2} &= W_2^k=(-1)^k \\ W_4^{k3} &= W_4^{k2}W_4^k=W_2^kW_4^k=(-1)^kW_4^k \end{aligned} \tag{17}$$

we have

$$X(k)=x(0)+x(1)W_4^k+(-1)^kx(2)+(-1)^kx(3)W_4^k \tag{18}$$

Two terms have a common factor W_4^k. Combining them we arrive at

$$X(k)=\left[x(0)+(-1)^kx(2)\right]+W_4^k\left[x(1)+(-1)^kx(3)\right] \tag{19}$$

for $0\le k\le 3$. The expressions in brackets are butterflies (2-point DFTs). We see that a 4-point DFT can be written as a combination of 2, 2-point DFTs with *twiddle factors* W_4^k. The first butterfly operates on the even indices 0 and 2 while the second operates on the odd indices 1 and 3.

$$\begin{aligned} x_e(0)&=x(0) \;,\; x_e(1)=x(2) \\ x_o(0)&=x(1) \;,\; x_o(1)=x(3) \end{aligned}$$

$$\begin{aligned} X_e(k)&=x_e(0)+(-1)^kx_e(1) \\ X_o(k)&=x_o(0)+(-1)^kx_o(1) \end{aligned}$$

For $0\le k\le 1$

$$\begin{aligned} X(k)&=X_e(k)+W_4^kX_o(k) \\ X(k+2)&=X_e(k)-W_4^kX_o(k) \end{aligned}$$

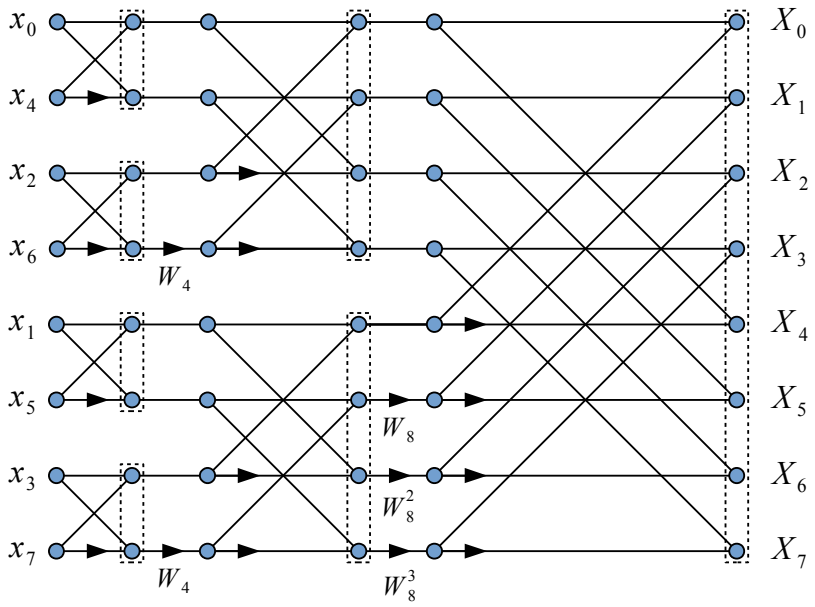

Fig. 3: Eight-point FFT.

$N = 8$

$$X(k) = x(0) + x(1)W_8^k + x(2)W_8^{k2} + x(3)W_8^{k3} + x(4)W_8^{k4} + x(5)W_8^{k5} + x(6)W_8^{k6} + x(7)W_8^{k7}$$

$$W_8^{k2} = W_4^k$$
$$W_8^{k3} = W_8^{k2}W_8^k = W_4^k W_8^k$$
$$W_8^{k4} = (-1)^k$$
$$W_8^{k5} = W_8^{k4}W_8^k = (-1)^k W_8^k$$
$$W_8^{k6} = W_8^{k4}W_8^{k2} = (-1)^k W_4^k$$
$$W_8^{k7} = W_8^{k4}W_8^{k3} = (-1)^k W_4^k W_8^k$$

$$X(k) = x(0) + x(1)W_8^k + x(2)W_4^k + x(3)W_4^k W_8^k + (-1)^k\left[x(4) + x(5)W_8^k + x(6)W_4^k + x(7)W_4^k W_8^k\right]$$

Four terms have a common factor W_8^k. Combining these we find

$$X(k) = \left[x(0) + x(2)W_4^k + (-1)^k x(4) + (-1)^k x(6)W_4^k\right] + W_8^k\left[x(1) + x(3)W_4^k + (-1)^k x(5) + (-1)^k W_4^k x(7)\right] \quad (20)$$

Comparing each of the bracketed expressions to (18) we see that an 8-point DFT can be expressed as a combination of 2, 4-point DFTs. The first involves the even indices while the second involves the odd indices. Those 4-point DFTs in turn can be expressed as combinations of 2-point DFTs.

$$x_e(0) = x(0)\ ,\ x_e(1) = x(2)\ ,\ x_e(2) = x(4)\ ,\ x_e(3) = x(6)$$
$$x_o(0) = x(1)\ ,\ x_o(1) = x(3)\ ,\ x_o(2) = x(5)\ ,\ x_o(3) = x(7)$$

$$X(k) = X_e(k) + W_4^k X_o(k)$$
$$X(k+4) = X_e(k) - W_8^k X_o(k)$$

We are seeing a pattern that suggests that an N-point DFT can be calculated by combining the 2, $N/2$ -point DFTs of the even and odd index terms with the twiddle factors W_N^k .

$N = 2^p$

For the general case $N = 2^p$ the DFT is

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

Let's define the sequence of even-index and odd-index elements of $x(n)$ as

$$x_e(m) = x(2m)$$
$$x_o(m) = x(2m+1)$$

with $0 \le m \le N/2 - 1$. Using these definitions we can put the DFT in the form

$$X(k) = \sum_{m=0}^{N/2-1} x_e(m) W_N^{k(2m)} + \sum_{m=0}^{N/2-1} x_o(m) W_N^{k(2m+1)}$$
$$= \sum_{m=0}^{N/2-1} x_e(m) W_{N/2}^{km} + W_N^k \sum_{m=0}^{N/2-1} x_o(m) W_{N/2}^{km}$$

The summations are each $N/2$ -point DFTs. Let's denote these by

$$X_e(k) = \sum_{m=0}^{N/2-1} x_e(m) W_{N/2}^{km}$$
$$X_o(k) = \sum_{m=0}^{N/2-1} x_o(m) W_{N/2}^{km}$$

These are periodic; $X_e(k+N/2) = X_e(k)$ and $X_o(k+N/2) = X_o(k)$.

$$X(k) = X_e(k) + W_N^k X_o(k) \quad , \quad 0 \le k \le N-1$$

Since $W_N^{k+N/2} = W_N^{N/2} W_N^k = -W_N^k$ for $0 \le k \le N/2 - 1$

$$X(k) = X_e(k) + W_N^k X_o(k)$$
$$X(k+N/2) = X_e(k) - W_N^k X_o(k)$$

References

1. https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

Appendix – Scilab code

```
//all times using Scilab on Windows 10 with Intel i5-3210M @ 2.5GHz

//brute-force DFT
//takes about 10 sec for N=1024
function X = dft1(x)
  N = length(x);
  for k=0:N-1
    X(k+1) = x(1);
    q = -%i*2*%pi*k/N;
    for n=1:N-1
      X(k+1) = X(k+1)+exp(q*n)*x(n+1);
    end
  end
endfunction

//recursive FFT
//takes about 23 millisec for N=1024
function X = fft1(x)
  N = length(x);
  if (N==2)
    X = [x(1)+x(2),x(1)-x(2)];
  else
    Xe = fft1(x(1:2:N-1));
    Xo = fft1(x(2:2:N));
    k = [0:N/2-1];
    Wk = exp(-%i*2*%pi*k/N);
    WkXo = Wk.*Xo;
    X = [Xe+WkXo,Xe-WkXo];
  end
endfunction

//built-in FFT
//takes about 92 microsec for N=1024
X = fft(x);
```