

MATH3511/6111: Scientific Computing

13. Fast Fourier Transformation

Lindon Roberts

Semester 1, 2022

Office: Hanna Neumann Building #145, Room 4.87

Email: lindon.roberts@anu.edu.au

Based on lecture notes written by S. Roberts, L. Stals, Q. Jin, M. Hegland, K. Duru.



Australian
National
University

Signal processing is the analysis of signals such as:

- Images (e.g. biology, medicine, astronomy)
- Video (e.g. drones, CCTV)
- Sound (e.g. music, speech)

Techniques from signal processing can be used to perform useful tasks like

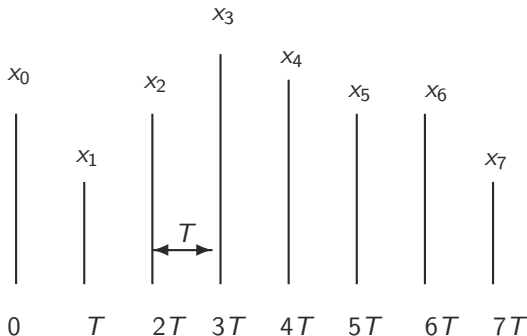
- Image compression
- Noise removal
- Classification
- Feature detection

Continuous vs. Discrete

Most signals are **continuous** in reality (e.g. sound, vision).

However, most digital equipment samples the signals at **discrete intervals**

- Pixels in a photo
- Bitrate for sound (# samples of audio intensity per second)



Periodic Signals

In reality, a discrete sampled signal is a sequence of values

$$\dots, x_{-2}, x_{-1}, x_0, x_1, x_2, \dots$$

This is not feasible to work with (infinite in size, how do you capture infinitely many pieces of information?). A common technique is to assume the signal is **periodic**: it repeats after every n values:

$$\dots, x_{n-2}, x_{n-1}, x_0, x_1, x_2, \dots, x_{n-2}, x_{n-1}, x_0, x_1, \dots$$

Then, we only have to worry about the finite vector

$$\mathbf{x} = [x_0, x_1, \dots, x_{n-1}]^T \in \mathbb{R}^n$$

This is an object we can more easily study.

Periodic Signals

Simple example: musical instruments playing a single note

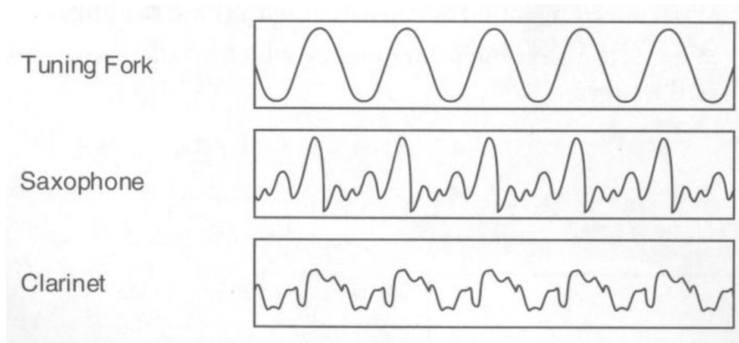


Figure 1. Sound waves from different instruments.

Image source: <https://slideplayer.com/slide/14412059/>

Fourier Series

First, let's look at periodic functions (the continuous case).

Suppose $f(t)$ is periodic with period L :

$$\dots = f(t - 2L) = f(t - L) = f(t) = f(t + L) = f(t + 2L) = \dots$$

for all $t \in [0, L]$, and continuous in each repeating interval $[-L, 0)$, $[0, L)$, $[L, 2L)$, etc. For example, the sound waves on the previous slide.

Then it is possible to write $f(t)$ as a linear combination of sines and cosines:

$$f(t) = \frac{A_0}{2} + \sum_{n=1}^{\infty} \left[A_n \cos\left(\frac{2\pi nt}{L}\right) + B_n \sin\left(\frac{2\pi nt}{L}\right) \right]$$

This is called the **Fourier series** for $f(t)$. This gives $f(t)$ in terms of the basis (for periodic functions)

$$\left\{ 1, \cos\left(\frac{2\pi t}{L}\right), \sin\left(\frac{2\pi t}{L}\right), \cos\left(\frac{4\pi t}{L}\right), \sin\left(\frac{4\pi t}{L}\right), \dots \right\}$$

$$f(t) = \frac{A_0}{2} + \sum_{n=1}^{\infty} \left[A_n \cos\left(\frac{2\pi nt}{L}\right) + B_n \sin\left(\frac{2\pi nt}{L}\right) \right]$$

Fortunately, our basis is actually an orthogonal basis, given the inner product (on the vector space of periodic functions)

$$(f, g) = \int_0^L f(t)g(t)dt.$$

Because of this, it is easy to calculate the coefficients of $f(t)$ in this basis:

$$A_n = \frac{2}{L} \int_0^L f(t) \cos\left(\frac{2\pi nt}{L}\right) dt, \quad B_n = \frac{2}{L} \int_0^L f(t) \sin\left(\frac{2\pi nt}{L}\right) dt.$$

History: Jean-Baptiste Joseph Fourier (1822), formalised by Peter Gustav Lejeune Dirichlet (1829) and Bernhard Riemann (1854), but decomposing curves into periodic functions goes back to ancient Greece.

Fourier Series

Example: if $f(t) = t - 1$ for $t \in [0, 2)$, then $A_n = 0$ and $B_n = -\frac{2}{n\pi}$.

Comparing $f(t)$ with the first 10 terms of the Fourier series, we get:

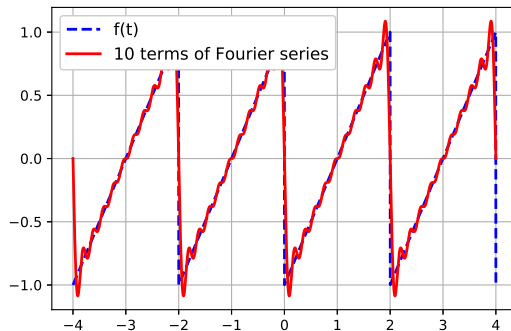


Figure 2. $f(t)$ vs. first 10 terms of Fourier series for sawtooth function.

Complex Numbers: reminder

Remember **Euler's identity** (Leonhard Euler, 1748, but possibly known to Johann Bernoulli and/or Roger Cotes earlier)

$$e^{i\theta} = \cos(\theta) + i \sin(\theta) = \text{cis}(\theta)$$

This implies

$$e^{-i\theta} = \cos(-\theta) + i \sin(-\theta) = \cos(\theta) - i \sin(\theta).$$

Therefore we can change between $(\sin(\theta), \cos(\theta))$ and $e^{\pm i\theta}$:

$$\cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}, \quad \text{and} \quad \sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}.$$

The **primitive n -th root of unity** is $\omega_n = e^{-2\pi i/n} = \text{cis}(-2\pi/n)$. The n -th roots of unity are

$$\{1, \omega_n, \omega_n^2, \dots, \omega_n^{n-1}\} = \{\omega_n^k : k = 0, \dots, n-1\}.$$

Complex Fourier Series

$$f(t) = \frac{A_0}{2} + \sum_{n=1}^{\infty} \left[A_n \cos\left(\frac{2\pi nt}{L}\right) + B_n \sin\left(\frac{2\pi nt}{L}\right) \right]$$

The Fourier series writes $f(t)$ in the basis $\{1, \cos(\frac{2\pi t}{L}), \sin(\frac{2\pi t}{L}), \dots\}$.

We can also change basis from cos and sin to complex exponentials:

$$f(t) = c_0 + \sum_{n=1}^{\infty} \left(c_n e^{2\pi i n t / L} + c_{-n} e^{-2\pi i n t / L} \right) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i n t / L}.$$

This is the **complex form of the Fourier series**.

The coefficients c_n are also complex:

$$c_0 = \frac{A_0}{2}, \quad c_n = \frac{A_n + iB_n}{2}, \quad c_{-n} = \frac{A_n - iB_n}{2}.$$

Complex Fourier Series

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i n t / L}$$

The coefficients are

$$c_0 = \frac{A_0}{2}, \quad c_n = \frac{A_n + iB_n}{2}, \quad c_{-n} = \frac{A_n - iB_n}{2}.$$

Using Euler's identity and the definition of A_n , B_n we get

$$c_n = \frac{1}{L} \int_0^L f(t) e^{-2\pi i n t / L} dt.$$

Note: we assumed $f(t)$ was periodic around $[0, L)$, but any other interval is fine too (some constants change, but the basic ideas are identical). The most common alternative is $[-L, L]$.

Fourier Transform

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i n t / L}, \quad \text{where} \quad c_n = \frac{1}{L} \int_0^L f(t) e^{-2\pi i n t / L} dt$$

Question

What if $f(t)$ is not periodic on any interval?

Idea: look at the ‘interval’ $(-\infty, \infty)$, and allow n to take non-integer values (sum \rightarrow integral).

The **Fourier transform** of a function $f(t)$ is

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \omega t} dt.$$

Given $\hat{f}(\omega)$, we can go back to $f(t)$ using the **inverse** Fourier Transform

$$f(t) = \int_{-\infty}^{\infty} \hat{f}(\omega) e^{2\pi i \omega t} d\omega.$$

Fourier Transform

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-2\pi i\omega t} dt$$

The Fourier transform has several nice properties.

Original function $h(t)$	Fourier transform $\hat{h}(\omega)$	
$af(t) + bg(t)$	$a\hat{f}(\omega) + b\hat{g}(\omega)$	Linearity
$f(at)$	$\frac{1}{ a }\hat{f}\left(\frac{\omega}{a}\right)$	Time scaling
$\frac{1}{ b }f\left(\frac{t}{b}\right)$	$\hat{f}(b\omega)$	Frequency scaling
$f(t - t_0)$	$\hat{f}(\omega)e^{2\pi i\omega t_0}$	Time shifting
$f(t)e^{-2\pi i\omega_0 t}$	$\hat{f}(\omega - \omega_0)$	Frequency shifting
$f'(t)$	$2\pi i\omega\hat{f}(\omega)$	Differentiation

Fourier Transform

Let's prove the time scaling result.

Suppose $h(t) = f(at)$ for some $a \neq 0$, and use the change of variables $u = at$.

$$\begin{aligned}\hat{h}(\omega) &= \int_{-\infty}^{\infty} f(at) e^{-2\pi i \omega t} dt, \\ &= \begin{cases} \int_{-\infty}^{\infty} f(u) e^{-2\pi i \omega (u/a)} \frac{1}{a} du, & \text{if } a > 0, \\ \int_{\infty}^{-\infty} f(u) e^{-2\pi i \omega (u/a)} \frac{1}{a} du, & \text{if } a < 0, \end{cases} \\ &= \int_{-\infty}^{\infty} f(u) e^{-2\pi i (\omega/a) u} \frac{1}{|a|} du, \\ &= \frac{1}{|a|} \hat{f}\left(\frac{\omega}{a}\right).\end{aligned}$$

Discrete Fourier Transform

Going back to periodic functions on $[0, L)$, we had

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{2\pi i n t / L}, \quad \text{where} \quad c_n = \frac{1}{L} \int_0^L f(t) e^{-2\pi i n t / L} dt$$

Now suppose we have a **discrete periodic signal** based on $\mathbf{x} \in \mathbb{R}^n$:

$$\dots, x_{n-2}, x_{n-1}, x_0, x_1, x_2, \dots, x_{n-2}, x_{n-1}, x_0, x_1, \dots$$

It turns out that we can also represent \mathbf{x} in terms of complex exponentials:

$$x_k = \sum_{j=0}^{n-1} c_j e^{2\pi i j k / n}, \quad \text{where} \quad c_j = \frac{1}{n} \sum_{k=0}^{n-1} x_k e^{-2\pi i j k / n}.$$

This is basically the same as the continuous case (integral \rightarrow sum).

Discrete Fourier Transform

Definition

If $\mathbf{x} \in \mathbb{C}^n$, the **Discrete Fourier Transform** (DFT) of \mathbf{x} is the vector $\hat{\mathbf{x}} \in \mathbb{C}^n$ defined by

$$\hat{x}_k = \sum_{j=0}^{n-1} x_j e^{-2\pi i j k / n} = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \quad \forall k = 0, \dots, n-1.$$

where $\omega_n = e^{-2\pi i / n}$ is the primitive n -th root of unity.

The DFT calculates the coefficients c_j (up to a constant factor of $1/n$): $c_j = \hat{x}_j / n$.

This is also the discrete equivalent of the (continuous) Fourier transform

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \omega t} dt.$$

History: Alexis Clairaut (sine-only, 1754), Joseph Louis Lagrange (cos-only 1759, complex 1770), Carl Friedrich Gauss (1805)

Discrete Fourier Transform

The DFT has properties similar to the continuous case:

Original signal \mathbf{z}	DFT $\hat{\mathbf{z}}$	
$a\mathbf{x} + b\mathbf{y}$	$a\hat{\mathbf{x}} + b\hat{\mathbf{y}}$	Linearity
$z_j = x_{j-j_0}$	$\hat{z}_k = e^{2\pi i j_0 k/n} \hat{x}_k$	Time shifting
$z_j = e^{-2\pi i j k_0/n} x_j$	$\hat{z}_k = \hat{x}_{k-k_0}$	Frequency shifting

Note that all indexing is done modulo n .

DFT matrix

Since the DFT is linear, we can write it as a matrix transformation from $\mathbb{C}^n \rightarrow \mathbb{C}^n$:

$$\hat{\mathbf{x}} = F_n \mathbf{x}, \quad \text{where} \quad [F_n]_{j,k} = \omega_n^{jk}, \quad \forall j, k = 0, \dots, n-1,$$

where $\omega_n = e^{-2\pi i/n}$. By definition, the matrix $F_n \in \mathbb{C}^{n \times n}$ is symmetric: $F_n^T = F_n$.

Discrete Fourier Transform

For example, if $n = 1$, then $\omega_n = 1$ so

$$F_1 = \begin{bmatrix} 1 \end{bmatrix}.$$

If $n = 2$, then $\omega_n = e^{-i\pi} = -1$ so

$$F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

If $n = 4$, then $\omega_n = e^{-i\pi/2} = -i$ so

$$F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}.$$

(we will often assume n is a power of 2, but other choices n are allowable too)

Gauss Relationship

Theorem

We have the Gauss relationship:

$$\sum_{j=0}^{n-1} \omega_n^{jk} = \begin{cases} n, & \omega_n^k = 1, \\ 0, & \omega_n^k \neq 1. \end{cases}$$

Proof: Case 1: If $\omega_n^k = 1$ then $\sum_{j=0}^{n-1} \omega_n^{jk} = \sum_{j=0}^{n-1} 1^j = n$.

Case 2: If $\omega_n^k \neq 1$, then we remember $\sum_{j=0}^{n-1} a^j = \frac{1-a^n}{1-a}$ and calculate

$$\sum_{j=0}^{n-1} \omega_n^{jk} = \frac{1 - \omega_n^{kn}}{1 - \omega_n^k} = \frac{1 - (\omega_n^n)^k}{1 - \omega_n^k} = \underbrace{\frac{1 - 1}{1 - \omega_n^k}}_{\neq 0} = 0,$$

since $\omega_n^n = 1$ (n -th root of unity).

The Gauss identity allows us to prove:

Theorem

If $F_n \in \mathbb{C}^{n \times n}$ is the DFT matrix, then

$$F_n \overline{F}_n = nI,$$

where \overline{F}_n denotes the complex conjugate (of each entry). Hence F_n is invertible.

Proof: The (j, k) entry of $F_n \overline{F}_n$ is

$$[F_n \overline{F}_n]_{j,k} = \sum_{l=0}^{n-1} \omega_n^{jl} \overline{\omega}_n^{lk} = \sum_{l=0}^{n-1} \omega_n^{jl} \omega_n^{-lk} = \sum_{l=0}^{n-1} \omega_n^{l(j-k)} = \begin{cases} n, & \text{if } j - k = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Inverse DFT

The inverse DFT (IDFT) is given by the linear transformation $\hat{\mathbf{x}} \rightarrow F_n^{-1} \hat{\mathbf{x}} = \frac{1}{n} \overline{F}_n \hat{\mathbf{x}}$. That is,

$$\mathbf{x} = \frac{1}{n} \overline{F}_n \hat{\mathbf{x}}, \quad \text{or} \quad x_j = \frac{1}{n} \sum_{k=0}^{n-1} \hat{x}_k \overline{\omega}_n^{jk} = \frac{1}{n} \sum_{k=0}^{n-1} \hat{x}_k \omega_n^{-jk}$$

Warning! Different software/textbooks may use different conventions to deal with the n factor in $F_n \overline{F}_n = nI$.

- Sometimes the DFT is defined as $\frac{1}{\sqrt{n}} F_n$, so the inverse is $\frac{1}{\sqrt{n}} \overline{F}_n$ (matrices where $A \overline{A}^T = I$ are called [Hermitian](#), they are the complex equivalent of orthogonal matrices)
- Sometimes the DFT is defined as $\frac{1}{n} F_n$ so the inverse is \overline{F}_n .

SciPy by uses our convention by default, but optionally allows both other choices.

Alternative Perspective

Another way of thinking about the DFT is that it **represents all linear transformations (“filters”) which are translationally invariant**.

Given our signal is periodic, it doesn't matter where we choose our 'starting point' x_0 :

$$\dots, x_{n-2}, x_{n-1}, x_0, x_1, x_2, \dots, x_{n-2}, x_{n-1}, x_0, x_1, \dots$$

If S is the **circulant matrix** that shifts the data forward by one time period

$$S\mathbf{x} = \begin{bmatrix} x_{n-1} & x_0 & x_1 & \cdots & x_{n-2} \end{bmatrix}^T,$$

then a linear transformation A is **translationally invariant** if $AS = SA$.

- i.e. Applying A then shifting the data is the same as shifting the data then applying A .

For instance, the 3-point moving average: $A = \frac{1}{3}(S^{-1} + I + S)$.

Alternative Perspective

Theorem

If A is a translationally invariant filter, then the eigenvectors of A are the columns of F_n .

That is, we have the eigendecomposition

$$A = F_n \Lambda F_n^{-1}.$$

So A is completely determined by the diagonal eigenvalue matrix Λ .

In fact, every such matrix A can be written as a polynomial in S ,

$$A = a_0 I + a_1 S + a_2 S^2 + \cdots + a_{n-1} S^{n-1},$$

and the eigenvalues Λ can be calculated by applying a DFT to the coefficients $[a_0, \dots, a_{n-1}]$!

Note that $S^n = I$ (n shifts of a length- n vector gets you back to where you started).

Basically, **any linear transform that is time-independent is based on the DFT.**

Convolution

An important operation is the **convolution** of two (discrete or continuous) signals.

Definition

The convolution of two vectors \mathbf{x} and \mathbf{y} is the vector \mathbf{z} defined by

$$z_k = \sum_{j=-\infty}^{\infty} x_j y_{k-j},$$

where indexing starts at zero and all undefined elements are assumed to be zero. We use the notation $\mathbf{z} = \mathbf{x} * \mathbf{y}$.

We generally only return the vector \mathbf{z} of the nonzero entries.

Convolution

Example: compute $\mathbf{x} * \mathbf{y}$ if $\mathbf{x} = (5, 7, 4)$ and $\mathbf{y} = (1, 2, 1, 3)$.

$$\begin{aligned}z_{-1} &= \cdots + x_{-1}y_0 + x_0y_{-1} + x_1y_{-2} + x_2y_{-3} + x_3y_{-4} + \cdots &= 0, \\z_0 &= \cdots + x_{-1}y_1 + x_0y_0 + x_1y_{-1} + x_2y_{-2} + x_3y_{-3} + \cdots &= 5, \\z_1 &= \cdots + x_{-1}y_2 + x_0y_1 + x_1y_0 + x_2y_{-1} + x_3y_{-2} + \cdots &= 17, \\z_2 &= \cdots + x_{-1}y_3 + x_0y_2 + x_1y_1 + x_2y_0 + x_3y_{-1} + \cdots &= 23, \\z_3 &= \cdots + x_{-1}y_4 + x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0 + \cdots &= 30, \\z_4 &= \cdots + x_{-1}y_5 + x_0y_4 + x_1y_3 + x_2y_2 + x_3y_1 + \cdots &= 25, \\z_5 &= \cdots + x_{-1}y_6 + x_0y_5 + x_1y_4 + x_2y_3 + x_3y_2 + \cdots &= 12, \\z_6 &= \cdots + x_{-1}y_7 + x_0y_6 + x_1y_5 + x_2y_4 + x_3y_3 + \cdots &= 0.\end{aligned}$$

We return the nonzero entries: $\mathbf{z} = \mathbf{x} * \mathbf{y} = (5, 17, 23, 30, 25, 12)$. See `numpy.convolve`.

Convolution

You have calculated convolutions before!

Multiplying polynomials:

$$(5, 7, 4) * (1, 2, 1, 3) = (5, 17, 23, 30, 25, 12),$$
$$(5 + 7x + 4x^2)(1 + 2x + x^2 + 3x^3) = 5 + 17x + 23x^2 + 30x^3 + 25x^4 + 12x^5.$$

Moving averages:

$$(5, 7, 4, 10, 3) * (0.5, 0.5) = (2.5, 6, 5.5, 7, 6.5, 1.5)$$

Pairwise differences:

$$(5, 7, 4, 10, 3) * (1, -1) = (5, 2, -3, 6, -7, -3)$$

Circular Convolution

If we **assume our vectors are periodic** and have the same length, then (by doing indexing modulo n) every term in the convolution is defined. We avoid infinite sums by dropping repeated terms (e.g. $x_0y_0 = x_ny_n$).

Definition

The **circular convolution** of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ is the vector $\mathbf{z} \in \mathbb{R}^n$ defined by

$$z_k = \sum_{j=0}^{n-1} x_j y_{k-j}, \quad k = 0, \dots, n-1,$$

where **indexing is modulo n** . We typically use the same notation $\mathbf{z} = \mathbf{x} * \mathbf{y}$ (be careful!).

In signal processing, often one vector is shorter (e.g. moving averages). We make it the same length by adding zeros.

Circular Convolution

Example: compute the circular convolution $\mathbf{x} * \mathbf{y}$ if $\mathbf{x} = (5, 7, 4, 0)$ and $\mathbf{y} = (1, 2, 1, 3)$.

$$z_0 = x_0y_0 + x_1y_{-1} + x_2y_{-2} + x_3y_{-3} = x_0y_0 + x_1y_3 + x_2y_2 + x_3y_1 = 30,$$

$$z_1 = x_0y_1 + x_1y_0 + x_2y_{-1} + x_3y_{-2} = x_0y_1 + x_1y_0 + x_2y_3 + x_3y_2 = 29,$$

$$z_2 = x_0y_2 + x_1y_1 + x_2y_0 + x_3y_{-1} = x_0y_2 + x_1y_1 + x_2y_0 + x_3y_3 = 23,$$

$$z_3 = x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0 = 30.$$

So $\mathbf{z} = \mathbf{x} * \mathbf{y} = (30, 29, 23, 30)$.

Moving averages and pairwise differences also work for periodic signals.

Periodic moving averages:

$$(5, 7, 4, 10, 3) * (0.5, 0.5, 0, 0, 0) = (4, 6, 5.5, 7, 6.5)$$

Periodic pairwise differences:

$$(5, 7, 4, 10, 3) * (1, -1, 0, 0, 0) = (2, 2, -3, 6, -7)$$

DFT of Circular Convolution

How do circular convolutions interact with the DFT?

Theorem

The DFT of a circular convolution is the element-wise products of the individual DFTs. The DFT of an element-wise product is $1/n$ times the circular convolution of the individual DFTs. That is,

$$\begin{aligned}\mathbf{x} * \mathbf{y} &\rightarrow \hat{\mathbf{x}} \odot \hat{\mathbf{y}}, \\ \mathbf{x} \odot \mathbf{y} &\rightarrow \frac{1}{n} \hat{\mathbf{x}} * \hat{\mathbf{y}},\end{aligned}$$

where \odot represents element-wise multiplication (“Hadamard product” of vectors).

DFT of Circular Convolution

Proof 1: Let $\mathbf{z} = \mathbf{x} * \mathbf{y}$. Then

$$\begin{aligned}\hat{z}_k &= \sum_{j=0}^{n-1} (\mathbf{x} * \mathbf{y})_j \omega_n^{jk} = \sum_{j=0}^{n-1} \left(\sum_{l=0}^{n-1} x_l y_{j-l} \right) \omega_n^{jk} = \sum_{j=0}^{n-1} \sum_{l=0}^{n-1} x_l y_{j-l} \omega_n^{jk} = \sum_{j=0}^{n-1} \sum_{l=0}^{n-1} (x_l \omega_n^{lk}) (y_{j-l} \omega_n^{(j-l)k}), \\ &= \sum_{l=0}^{n-1} \left[x_l \omega_n^{lk} \left(\sum_{j=0}^{n-1} y_{j-l} \omega_n^{(j-l)k} \right) \right] = \sum_{l=0}^{n-1} \left[x_l \omega_n^{lk} \left(\sum_{j=0}^{n-1} y_{j-l} \omega_n^{jk} \right) \right] = \left(\sum_{l=0}^{n-1} x_l \omega_n^{lk} \right) \left(\sum_{j=0}^{n-1} y_{j-l} \omega_n^{jk} \right),\end{aligned}$$

and so $\hat{z}_k = \hat{x}_k \hat{y}_k$.

DFT of Circular Convolution

Proof 2: Suppose $\hat{\mathbf{z}} = \frac{1}{n} \hat{\mathbf{x}} * \hat{\mathbf{y}}$. Then

$$\begin{aligned}\hat{z}_k &= \frac{1}{n} \sum_{j=0}^{n-1} \hat{x}_j \hat{y}_{k-j} = \frac{1}{n} \sum_{j=0}^{n-1} \left(\sum_{l=0}^{n-1} x_l \omega_n^{lj} \right) \left(\sum_{m=0}^{n-1} x_m \omega_n^{m(k-j)} \right) = \frac{1}{n} \sum_{j,l,m=0}^{n-1} x_l y_m \omega_n^{(l-m)j + mk}, \\ &= \frac{1}{n} \sum_{l,m=0}^{n-1} x_l y_m \omega_n^{mk} \left(\sum_{j=0}^{n-1} \omega_n^{(l-m)j} \right) = \sum_{l=0}^{n-1} x_l y_l \omega_n^{lk},\end{aligned}$$

where the last equality comes from the Gauss relationship.

Hence $z_l = x_l y_l$, so $\mathbf{z} = \mathbf{x} \odot \mathbf{y}$.

DFTs are useful: give us information about frequencies in a signal, easy to compute convolutions.

How can we compute the DFT?

$$\hat{\mathbf{x}} = F_n \mathbf{x}$$

Easy: matrix-vector multiplication!

Computing DFTs

For example, if $n = 4$, we have

$$\hat{x}_0 = \omega_n^0 x_0 + \omega_n^0 x_1 + \omega_n^0 x_2 + \omega_n^0 x_3,$$

$$\hat{x}_1 = \omega_n^0 x_0 + \omega_n^1 x_1 + \omega_n^2 x_2 + \omega_n^3 x_3,$$

$$\hat{x}_2 = \omega_n^0 x_0 + \omega_n^2 x_1 + \omega_n^4 x_2 + \omega_n^6 x_3,$$

$$\hat{x}_3 = \omega_n^0 x_0 + \omega_n^3 x_1 + \omega_n^6 x_2 + \omega_n^9 x_3,$$

where

$$\omega_n^j = e^{-2\pi i j/n} = \cos\left(\frac{2\pi j}{n}\right) - i \sin\left(\frac{2\pi j}{n}\right).$$

We can pre-compute the powers of ω_n if we want, but this still requires 16 complex multiplications and 12 complex additions.

In general, doing a matrix-vector multiplication requires $\mathcal{O}(n^2)$ flops for an $n \times n$ matrix.

Fast Fourier Transform

We can reduce the cost of calculating a DFT from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$ flops. This algorithm is called the **Fast Fourier Transform (FFT)** [Gauss, 1805; James Cooley and John Tukey, 1965].

- One of the 10 algorithms with “the greatest influence on the development and practice of science and engineering in the 20th century” (IEEE, 2000)
- “The most important algorithm of our lifetime” (G. Strang, 1994)

Makes manipulation of very large signals ($n \gg 1$) practical.

We will look at the case where n is a power of two, but similar ideas work for other n .

Fast Fourier Transform: Example

If $n = 4$ basic matrix-vector multiplication gives us

$$\hat{x}_0 = \omega_4^0 x_0 + \omega_4^0 x_1 + \omega_4^0 x_2 + \omega_4^0 x_3,$$

$$\hat{x}_1 = \omega_4^0 x_0 + \omega_4^1 x_1 + \omega_4^2 x_2 + \omega_4^3 x_3,$$

$$\hat{x}_2 = \omega_4^0 x_0 + \omega_4^2 x_1 + \omega_4^4 x_2 + \omega_4^6 x_3,$$

$$\hat{x}_3 = \omega_4^0 x_0 + \omega_4^3 x_1 + \omega_4^6 x_2 + \omega_4^9 x_3.$$

Rearranging, we have

$$\hat{x}_0 = (\omega_4^0 x_0 + \omega_4^0 x_2) + \omega_4^0 (\omega_4^0 x_1 + \omega_4^0 x_3),$$

$$\hat{x}_1 = (\omega_4^0 x_0 + \omega_4^2 x_2) + \omega_4^1 (\omega_4^0 x_1 + \omega_4^2 x_3),$$

$$\hat{x}_2 = (\omega_4^0 x_0 + \omega_4^4 x_2) + \omega_4^2 (\omega_4^0 x_1 + \omega_4^4 x_3),$$

$$\hat{x}_3 = (\omega_4^0 x_0 + \omega_4^6 x_2) + \omega_4^3 (\omega_4^0 x_1 + \omega_4^6 x_3).$$

Fast Fourier Transform: Example

$$\hat{x}_0 = (\omega_4^0 x_0 + \omega_4^0 x_2) + \omega_4^0 (\omega_4^0 x_1 + \omega_4^0 x_3),$$

$$\hat{x}_1 = (\omega_4^0 x_0 + \omega_4^2 x_2) + \omega_4^1 (\omega_4^0 x_1 + \omega_4^2 x_3),$$

$$\hat{x}_2 = (\omega_4^0 x_0 + \omega_4^4 x_2) + \omega_4^2 (\omega_4^0 x_1 + \omega_4^4 x_3),$$

$$\hat{x}_3 = (\omega_4^0 x_0 + \omega_4^6 x_2) + \omega_4^3 (\omega_4^0 x_1 + \omega_4^6 x_3).$$

Note that since n is even, then $\omega_n^2 = \omega_{n/2}$. Hence

$$\hat{x}_0 = (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0 (\omega_2^0 x_1 + \omega_2^0 x_3),$$

$$\hat{x}_1 = (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1 (\omega_2^0 x_1 + \omega_2^1 x_3),$$

$$\hat{x}_2 = (\omega_2^0 x_0 + \omega_2^2 x_2) + \omega_4^2 (\omega_2^0 x_1 + \omega_2^2 x_3),$$

$$\hat{x}_3 = (\omega_2^0 x_0 + \omega_2^3 x_2) + \omega_4^3 (\omega_2^0 x_1 + \omega_2^3 x_3).$$

Fast Fourier Transform: Example

$$\hat{x}_0 = (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0 (\omega_2^0 x_1 + \omega_2^0 x_3),$$

$$\hat{x}_1 = (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1 (\omega_2^0 x_1 + \omega_2^1 x_3),$$

$$\hat{x}_2 = (\omega_2^0 x_0 + \omega_2^2 x_2) + \omega_4^2 (\omega_2^0 x_1 + \omega_2^2 x_3),$$

$$\hat{x}_3 = (\omega_2^0 x_0 + \omega_2^3 x_2) + \omega_4^3 (\omega_2^0 x_1 + \omega_2^3 x_3).$$

The first two rows are linear combinations of Fourier transforms of size 2 (for (x_0, x_2) and (x_1, x_3))!

What about the last two rows? Use $\omega_2^{2+j} = \omega_2^j$ (since $\omega_2^2 = 1$).

$$\hat{x}_0 = (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0 (\omega_2^0 x_1 + \omega_2^0 x_3),$$

$$\hat{x}_1 = (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1 (\omega_2^0 x_1 + \omega_2^1 x_3),$$

$$\hat{x}_2 = (\omega_2^0 x_0 + \omega_2^2 x_2) + \omega_4^2 (\omega_2^0 x_1 + \omega_2^2 x_3),$$

$$\hat{x}_3 = (\omega_2^0 x_0 + \omega_2^3 x_2) + \omega_4^3 (\omega_2^0 x_1 + \omega_2^3 x_3).$$

The last two rows are also linear combinations of the same (smaller) Fourier transforms!

Fast Fourier Transform: Example

$$\hat{x}_0 = (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0 (\omega_2^0 x_1 + \omega_2^0 x_3),$$

$$\hat{x}_1 = (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1 (\omega_2^0 x_1 + \omega_2^1 x_3),$$

$$\hat{x}_2 = (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^2 (\omega_2^0 x_1 + \omega_2^0 x_3),$$

$$\hat{x}_3 = (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^3 (\omega_2^0 x_1 + \omega_2^1 x_3).$$

The $n = 4$ DFT is a linear combination of 2 DFTs of size $n = 2$. We can also make the weights the same using $\omega_4^{2+j} = -\omega_4^j$ (since $\omega_4^2 = -1$):

$$\hat{x}_0 = (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0 (\omega_2^0 x_1 + \omega_2^0 x_3),$$

$$\hat{x}_1 = (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1 (\omega_2^0 x_1 + \omega_2^1 x_3),$$

$$\hat{x}_2 = (\omega_2^0 x_0 + \omega_2^0 x_2) - \omega_4^0 (\omega_2^0 x_1 + \omega_2^0 x_3),$$

$$\hat{x}_3 = (\omega_2^0 x_0 + \omega_2^1 x_2) - \omega_4^1 (\omega_2^0 x_1 + \omega_2^1 x_3).$$

FFT: General Case

Now suppose we want to do a DFT of size n , where n is even (write $n = 2m$ for some integer m). The k -th entry is

$$\begin{aligned}\hat{x}_k &= x_0 + \omega_n^k x_1 + \omega_n^{2k} x_2 + \cdots + \omega_n^{(n-2)k} x_{n-2} + \omega_n^{(n-1)k} x_{n-1}, \\ &= \left[x_0 + \omega_n^{2k} x_2 + \cdots + \omega_n^{(n-2)k} x_{n-2} \right] + \omega_n^k \left[x_1 + \omega_n^{2k} x_3 + \cdots + \omega_n^{(n-2)k} x_{n-1} \right].\end{aligned}$$

Each term has only even powers of ω_n , so use $\omega_n^{2j} = (\omega_n^2)^j = \omega_m^j$ (since $\omega_n^2 = \omega_m$):

$$\hat{x}_k = \left[x_0 + \omega_m^k x_2 + \cdots + \omega_m^{(m-1)k} x_{n-2} \right] + \omega_n^k \left[x_1 + \omega_m^k x_3 + \cdots + \omega_m^{(m-1)k} x_{n-1} \right].$$

If $k = 0, \dots, m-1$ (first half of the entries), these are just the entries of a DFT of size m (acting on the even/odd-numbered entries of \mathbf{x}).

FFT: General Case

$$\hat{x}_k = \left[x_0 + \omega_m^k x_2 + \cdots + \omega_m^{(m-1)k} x_{n-2} \right] + \omega_n^k \left[x_1 + \omega_m^k x_3 + \cdots + \omega_m^{(m-1)k} x_{n-1} \right]$$

If $k = m, \dots, n-1$, then write $k = j + m$ for $j = 0, \dots, m-1$:

$$\begin{aligned} \hat{x}_{j+m} &= \left[x_0 + \omega_m^{j+m} x_2 + \cdots + \omega_m^{(m-1)j+(m-1)m} x_{n-2} \right] \\ &\quad + \omega_n^{j+m} \left[x_1 + \omega_m^{j+m} x_3 + \cdots + \omega_m^{(m-1)j+(m-1)m} x_{n-1} \right]. \end{aligned}$$

Since $\omega_m^m = 1$ and $\omega_n^m = -1$, we simplify to get

$$\hat{x}_{j+m} = \left[x_0 + \omega_m^j x_2 + \cdots + \omega_m^{(m-1)j} x_{n-2} \right] - \omega_n^j \left[x_1 + \omega_m^j x_3 + \cdots + \omega_m^{(m-1)j} x_{n-1} \right].$$

The two terms are the same as the two terms of \hat{x}_j but with a single sign change.

FFT: General Case

In general, we have for $j = 0, \dots, m-1$:

$$\begin{aligned}\hat{x}_j &= \left[x_0 + \omega_m^j x_2 + \dots + \omega_m^{(m-1)j} x_{n-2} \right] + \omega_n^j \left[x_1 + \omega_m^j x_3 + \dots + \omega_m^{(m-1)j} x_{n-1} \right], \\ \hat{x}_{j+m} &= \left[x_0 + \omega_m^j x_2 + \dots + \omega_m^{(m-1)j} x_{n-2} \right] - \omega_n^j \left[x_1 + \omega_m^j x_3 + \dots + \omega_m^{(m-1)j} x_{n-1} \right].\end{aligned}$$

That is,

$$\begin{aligned}\hat{x}_j &= \hat{y}_j + \omega_n^j \hat{z}_j, \\ \hat{x}_{j+m} &= \hat{y}_j - \omega_n^j \hat{z}_j,\end{aligned}$$

where $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ are the DFTs (of size $m = n/2$) for the vectors

$$\mathbf{y} = (x_0, x_2, \dots, x_{n-2})^T, \quad \text{and} \quad \mathbf{z} = (x_1, x_3, \dots, x_{n-1})^T.$$

Actually, since $F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$, we have $(\hat{x}_j, \hat{x}_{j+m})$ is the DFT of $(\hat{y}_j, \omega_n^j \hat{z}_j)$ (i.e. do $n/2$ small DFTs, of size 2).

FFT: Matrix Form

FFT Steps

1. Permute the entries of \mathbf{x} (get the even and odd-numbered entries \mathbf{y} and \mathbf{z})
2. Two FFTs of size $m = n/2$ (on \mathbf{y} and \mathbf{z})
3. Take linear combinations to get final result ($\hat{y}_j \pm \omega_n^j \hat{z}_j$)

$$F_n = B_n \begin{bmatrix} F_m & 0 \\ 0 & F_m \end{bmatrix} P_n,$$

where P_n is the permutation matrix given by

$$P_n \mathbf{x} = (x_0, x_2, \dots, x_{n-2}, x_1, x_3, \dots, x_{n-1})^T,$$

and B_n is a **butterfly** matrix that gives the linear combinations $\hat{y}_j \pm \omega_n^j \hat{z}_j$ (in the correct order):

$$B_n = \begin{bmatrix} I_{m \times m} & \Theta_m \\ I_{m \times m} & -\Theta_m \end{bmatrix},$$

where I is the identity and $\Theta_m = \text{diag}(1, \omega_n^1, \dots, \omega_n^{m-1})$.

FFT: Matrix Form

Key idea: keep breaking down F_n , $F_{n/2}$, etc., recursively. Then

$$F_n = B_n \begin{bmatrix} F_m & 0 \\ 0 & F_m \end{bmatrix} P_n,$$

becomes

$$F_n = B_n \begin{bmatrix} B_{n/2} & 0 \\ 0 & B_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/4} & 0 & 0 & 0 \\ 0 & F_{n/4} & 0 & 0 \\ 0 & 0 & F_{n/4} & 0 \\ 0 & 0 & 0 & F_{n/4} \end{bmatrix} \begin{bmatrix} P_{n/2} & 0 \\ 0 & P_{n/2} \end{bmatrix} P_n.$$

If n is a power of two, we can keep doing this until we get to a very simple DFT (e.g. F_1 or F_2) which can be easily computed.

If n is not a power of two, it is trickier, but similar recursive tricks can be used to calculate fast DFTs (but depends on the prime factors of n). The same tricks also work for the inverse DFT.

FFT: Implementation

```
def fft(x):
    n = len(x)  # n must be a power of 2
    # Base case
    if n == 1:
        F1 = np.array([1.0])
        return np.array([F1 @ x])  # force to be vector of length 1

    # Recursive case
    omega_n = cmath.exp(-2 * cmath.pi * 1j / n)
    y = x[::2]  # even entries [x0,x2,...] using clever NumPy indexing
    z = x[1::2]  # odd entries [x1,x3,...]
    yhat = fft(y)
    zhat = fft(z)
    xhat = np.zeros((n,), dtype=complex)
    for j in range(n // 2):  # using integer division
        xhat[j] = yhat[j] + omega_n**j * zhat[j]
        xhat[j+n//2] = yhat[j] - omega_n**j * zhat[j]
    return xhat
```

FFT: Computational Cost

Let's check the cost (in flops) of the FFT method, if n is a power of 2.

We assume we have computed ω_n and the powers ω_n^j already (since we can do this 'offline', before knowing what \mathbf{x} will be).

Complex arithmetic: assuming we operate on real and imaginary parts separately,

- Adding two complex numbers takes 2 flops: $(a + bi) + (c + di) = (a + c) + (b + d)i$
- Multiplying two complex numbers takes 6 flops: $(a + bi)(c + di) = (ac - bd) + (ad + bc)i$.

Our FFT cost is the cost of the two smaller FFTs plus $2n$ complex adds and n complex multiplies:

$$C(n) = 2C(n/2) + 10n.$$

FFT: Computational Cost

Let's check the cost (in flops) of the FFT method, if n is a power of 2 (continued).

$$\begin{aligned}C(n) &= 2C(n/2) + 10n, \\&= 2 \left[2C(n/4) + \frac{10n}{2} \right] + 10n = 4C(n/4) + 20n, \\&= 4 \left[2C(n/8) + \frac{10n}{4} \right] + 20n = 8C(n/8) + 30n, \\&\dots \\&= 2^{r-1} \left[2C(n/2^r) + \frac{10n}{2^{r-1}} \right] + 10rn = 2^r C(n/2^r) + 10(r+1)n.\end{aligned}$$

Eventually, $C(1) = 0$ since $F_1 = [1]$. To get to $C(1)$ we do $r = \log_2 n$ steps, so the cost is

$$C(n) = 0 + 10(\log_2 n + 1)n = \mathcal{O}(n \log_2 n).$$

This is far superior to $\mathcal{O}(n^2)$ from simple matrix-vector multiplication.

Audio Analysis Example

Frequency analysis of piano note (middle C, C4 $\approx 262\text{Hz}$).

The sound wave $\mathbf{x} \in \mathbb{R}^{44100}$ is plotted below:

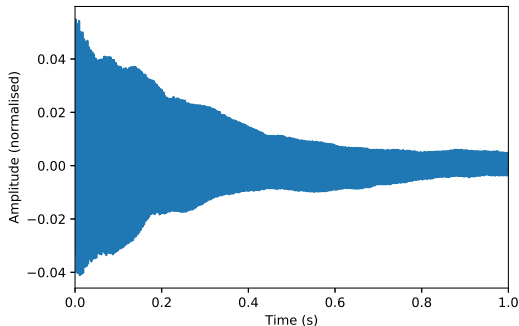


Figure 3. Piano sound wave (middle C).

Audio source: <http://theremin.music.uiowa.edu/MISpiano.html>

Audio Analysis Example

Frequency analysis of piano note (middle C, C4 $\approx 262\text{Hz}$).

The sound wave $\mathbf{x} \in \mathbb{R}^{44100}$ is plotted below, [zoomed in](#):

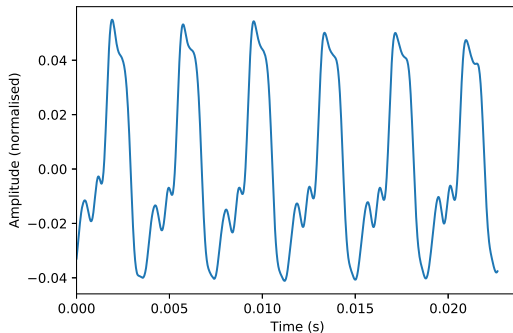


Figure 4. Piano sound wave (middle C).

Audio source: <http://theremin.music.uiowa.edu/MISpiano.html>

Audio Analysis Example

Frequency analysis of piano note (middle C, C4 $\approx 262\text{Hz}$).

Now plot $|\hat{x}_k|$ (magnitude of each element of $\hat{\mathbf{x}}$):

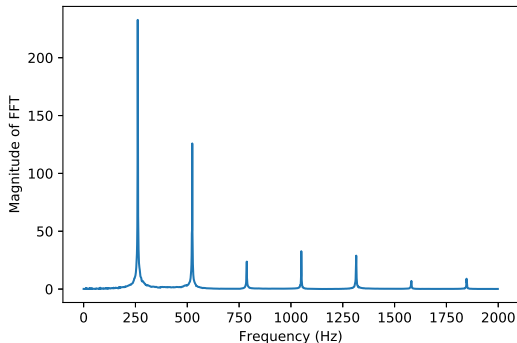


Figure 5. FFT of piano sound wave (middle C).

Alternative Signal Processing Techniques

There are several techniques for signal/image processing similar to DFT/FFT:

- Discrete Cosine Transform: cosine basis functions (Nasir Ahmer, 1972)
- Walsh-Hadamard Transform: piecewise constant basis functions (Joseph Walsh 1923, Jacques Hadamard)
- Wavelet Transform: localised basis functions (Alfred Haar 1909, Stéphane Mallat, Ingrid Daubechies, 1970s–90s)

All of these can be calculated efficiently using similar recursive techniques to the FFT.

The DCT is the most widely used, including for JPEG (images), MP3 (audio) and MP4 (video).