MATH3X76/4076: Mathematical Computing

09. Fast Fourier Transformation

Lindon Roberts

Semester 1, 2023

Office: Carslaw Building F07, Room 683
Email: lindon.roberts@sydney.edu.au

Based on lecture notes written by S. Roberts, L. Stals, Q. Jin, M. Hegland, K. Duru, A. Paganini, G. Vasil, F. Roosta, C. Cartis.

THE UNIVERSITY OF SYDNEY

Signal Processing

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

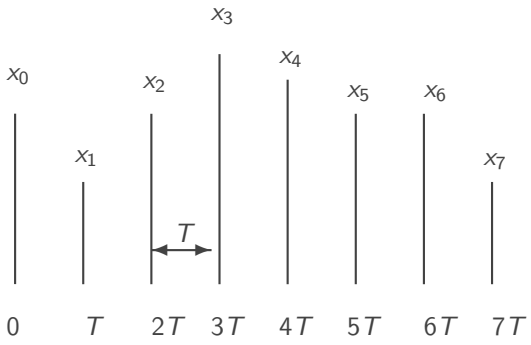
- Image compression
- Noise removal
- Classification
- Feature detection

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

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

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:

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

Then, we only have to worry about the finite vector

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

This is an object we can more easily study.

Periodic Signals

Simple example: musical instruments playing a single note

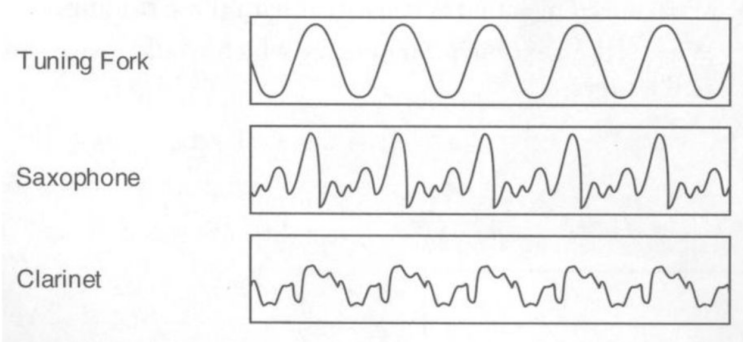


Figure 1. Sound waves from different instruments.

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:

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

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:

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

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

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

Fourier Series

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

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:

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

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.

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:

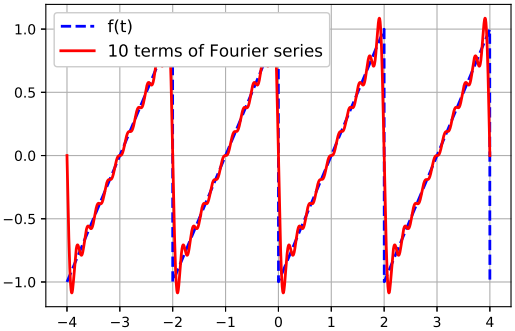


Figure 2. $f(t)$ vs. first 10 terms of Fourier series for sawtooth function.

Complex Numbers: reminder

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

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

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 $e^{\pm i\theta}$:

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

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

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

Complex Fourier Series

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

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

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:

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

Complex Fourier Series

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

The coefficients are

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

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 $[-L, L]$.

Discrete Fourier Transform

We have

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

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

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

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

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

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

Discrete Fourier Transform

Definition

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

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

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 $1/n$): $c_j = \hat{x}_j/n$.

It is the discrete equivalent of the Fourier transform ($\approx$ Fourier series for non-periodic functions)

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

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

Discrete Fourier Transform

The DFT has some nice properties (the continuous case is similar):

Original signal $\boldsymbol{z}$	DFT $\hat{\boldsymbol{z}}$	
$a\boldsymbol{x} + b\boldsymbol{y}$	$a\hat{\boldsymbol{x}} + b\hat{\boldsymbol{y}}$	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 $\mathbb{C}^n \to \mathbb{C}^n$:

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

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 of 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).

Inverse DFT

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 $\hat{\boldsymbol{x}} \to F_n^{-1}\hat{\boldsymbol{x}} = \frac{1}{n}\overline{F}_n\hat{\boldsymbol{x}}$. That is,

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

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

Convolution

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

Definition

The convolution of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is the vector $\boldsymbol{z}$ 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 $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$.

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

Convolution

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

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

We return the nonzero entries: $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y} = (5, 17, 23, 30, 25, 12)$. 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. $x_0 y_0 = x_n y_n$).

Definition

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

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

where indexing is modulo n. We typically use the same notation $\boldsymbol{z} = \boldsymbol{x} * \boldsymbol{y}$ (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 $\boldsymbol{x} * \boldsymbol{y}$ if $\boldsymbol{x} = (5, 7, 4, 0)$ and $\boldsymbol{y} = (1, 2, 1, 3)$.

$$
\begin{aligned}
z_0 &= x_0y_0 + x_1y_{-1} + x_2y_{-2} + x_3y_{-3} = x_0y_0 + x_1y_3 + x_2y_2 + x_3y_1 &&= 30,\\
z_1 &= x_0y_1 + x_1y_0 + x_2y_{-1} + x_3y_{-2} = x_0y_1 + x_1y_0 + x_2y_3 + x_3y_2 &&= 29,\\
z_2 &= x_0y_2 + x_1y_1 + x_2y_0 + x_3y_{-1} = x_0y_2 + x_1y_1 + x_2y_0 + x_3y_3 &&= 23,\\
z_3 &= x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0 &&= 30.
\end{aligned}
$$

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

Circular Convolution

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:

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

Applying specific convolutions to input signals is sometimes called applying a linear filter in engineering.

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,

$$\boldsymbol{x} * \boldsymbol{y} \to \hat{\boldsymbol{x}} \odot \hat{\boldsymbol{y}},$$
$$\boldsymbol{x} \odot \boldsymbol{y} \to \frac{1}{n} \hat{\boldsymbol{x}} * \hat{\boldsymbol{y}},$$

where $\odot$ represents element-wise multiplication ("Hadamard product" of vectors).

Computing DFTs

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

How can we compute the DFT?

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

Easy: matrix-vector multiplication!

Computing DFTs

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

$$\begin{aligned}
\hat{x}_0 &= \omega_n^0 x_0 + \omega_n^0 x_1 + \omega_n^0 x_2 + \omega_n^0 x_3, \\
\hat{x}_1 &= \omega_n^0 x_0 + \omega_n^1 x_1 + \omega_n^2 x_2 + \omega_n^3 x_3, \\
\hat{x}_2 &= \omega_n^0 x_0 + \omega_n^2 x_1 + \omega_n^4 x_2 + \omega_n^6 x_3, \\
\hat{x}_3 &= \omega_n^0 x_0 + \omega_n^3 x_1 + \omega_n^6 x_2 + \omega_n^9 x_3,
\end{aligned}$$

where

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

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

$$\begin{aligned}
\hat{x}_0 &= \omega_4^0 x_0 + \omega_4^0 x_1 + \omega_4^0 x_2 + \omega_4^0 x_3, \\
\hat{x}_1 &= \omega_4^0 x_0 + \omega_4^1 x_1 + \omega_4^2 x_2 + \omega_4^3 x_3, \\
\hat{x}_2 &= \omega_4^0 x_0 + \omega_4^2 x_1 + \omega_4^4 x_2 + \omega_4^6 x_3, \\
\hat{x}_3 &= \omega_4^0 x_0 + \omega_4^3 x_1 + \omega_4^6 x_2 + \omega_4^9 x_3.
\end{aligned}$$

Rearranging, we have

$$\begin{aligned}
\hat{x}_0 &= (\omega_4^0 x_0 + \omega_4^0 x_2) + \omega_4^0(\omega_4^0 x_1 + \omega_4^0 x_3), \\
\hat{x}_1 &= (\omega_4^0 x_0 + \omega_4^2 x_2) + \omega_4^1(\omega_4^0 x_1 + \omega_4^2 x_3), \\
\hat{x}_2 &= (\omega_4^0 x_0 + \omega_4^4 x_2) + \omega_4^2(\omega_4^0 x_1 + \omega_4^4 x_3), \\
\hat{x}_3 &= (\omega_4^0 x_0 + \omega_4^6 x_2) + \omega_4^3(\omega_4^0 x_1 + \omega_4^6 x_3).
\end{aligned}$$

Fast Fourier Transform: Example

$$\begin{aligned}
\hat{x}_0 &= (\omega_4^0 x_0 + \omega_4^0 x_2) + \omega_4^0(\omega_4^0 x_1 + \omega_4^0 x_3),\\
\hat{x}_1 &= (\omega_4^0 x_0 + \omega_4^2 x_2) + \omega_4^1(\omega_4^0 x_1 + \omega_4^2 x_3),\\
\hat{x}_2 &= (\omega_4^0 x_0 + \omega_4^4 x_2) + \omega_4^2(\omega_4^0 x_1 + \omega_4^4 x_3),\\
\hat{x}_3 &= (\omega_4^0 x_0 + \omega_4^6 x_2) + \omega_4^3(\omega_4^0 x_1 + \omega_4^6 x_3).
\end{aligned}$$

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

$$\begin{aligned}
\hat{x}_0 &= (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_1 &= (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1(\omega_2^0 x_1 + \omega_2^1 x_3),\\
\hat{x}_2 &= (\omega_2^0 x_0 + \omega_2^2 x_2) + \omega_4^2(\omega_2^0 x_1 + \omega_2^2 x_3),\\
\hat{x}_3 &= (\omega_2^0 x_0 + \omega_2^3 x_2) + \omega_4^3(\omega_2^0 x_1 + \omega_2^3 x_3).
\end{aligned}$$

Fast Fourier Transform: Example

$$\begin{aligned}
\hat{x}_0 &= (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_1 &= (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1(\omega_2^0 x_1 + \omega_2^1 x_3),\\
\hat{x}_2 &= (\omega_2^0 x_0 + \omega_2^2 x_2) + \omega_4^2(\omega_2^0 x_1 + \omega_2^2 x_3),\\
\hat{x}_3 &= (\omega_2^0 x_0 + \omega_2^3 x_2) + \omega_4^3(\omega_2^0 x_1 + \omega_2^3 x_3).
\end{aligned}$$

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 $\omega_2^2 = 1$).

$$\begin{aligned}
\hat{x}_0 &= (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_1 &= (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1(\omega_2^0 x_1 + \omega_2^1 x_3),\\
\hat{x}_2 &= (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^2(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_3 &= (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^3(\omega_2^0 x_1 + \omega_2^1 x_3).
\end{aligned}$$

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

Fast Fourier Transform: Example

$$\begin{aligned}
\hat{x}_0 &= (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_1 &= (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1(\omega_2^0 x_1 + \omega_2^1 x_3),\\
\hat{x}_2 &= (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^2(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_3 &= (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^3(\omega_2^0 x_1 + \omega_2^1 x_3).
\end{aligned}$$

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 $\omega_4^2 = -1$):

$$\begin{aligned}
\hat{x}_0 &= (\omega_2^0 x_0 + \omega_2^0 x_2) + \omega_4^0(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_1 &= (\omega_2^0 x_0 + \omega_2^1 x_2) + \omega_4^1(\omega_2^0 x_1 + \omega_2^1 x_3),\\
\hat{x}_2 &= (\omega_2^0 x_0 + \omega_2^0 x_2) - \omega_4^0(\omega_2^0 x_1 + \omega_2^0 x_3),\\
\hat{x}_3 &= (\omega_2^0 x_0 + \omega_2^1 x_2) - \omega_4^1(\omega_2^0 x_1 + \omega_2^1 x_3).
\end{aligned}$$

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

FFT: General Case

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

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

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

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 $j = 0, \dots, m-1$:

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

That is,

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

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

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

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 Algorithm

FFT Steps

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

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 exactly how depends on the prime factors of n): a widely used package that implements this is FFTW.

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 $\boldsymbol{x}$ 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 n complex adds and $n/2$ complex multiplies:

$$C(n) = 2C(n/2) + 5n.$$

FFT: Computational Cost

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

$$\begin{aligned}
C(n) &= 2C(n/2) + 5n, \\
&= 2\left[2C(n/4) + \frac{5n}{2}\right] + 5n = 4C(n/4) + 10n, \\
&= 4\left[2C(n/8) + \frac{5n}{4}\right] + 10n = 8C(n/8) + 15n, \\
&\quad \dots \\
&= 2^{r-1}\left[2C(n/2^r) + \frac{5n}{2^{r-1}}\right] + 5(r-1)n = 2^r C(n/2^r) + 5rn.
\end{aligned}$$

Eventually, $C(1) = 0$ since $F_1 = \begin{bmatrix} 1 \end{bmatrix}$. To get to $C(1)$ we do $r = \log_2 n$ steps, so the cost is

$$C(n) = 0 + 5(\log_2 n + 1)n = \mathcal{O}(n \log_2 n).$$

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 $\approx$ 262Hz).

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

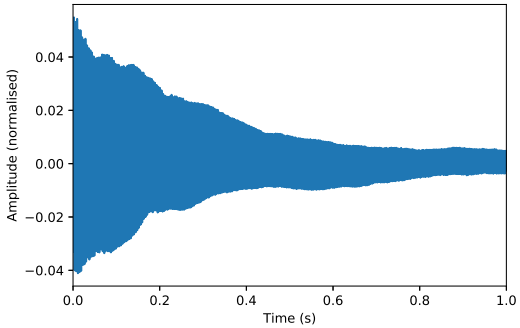


Figure 3. Piano sound wave (middle C).

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

Audio Analysis Example

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

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

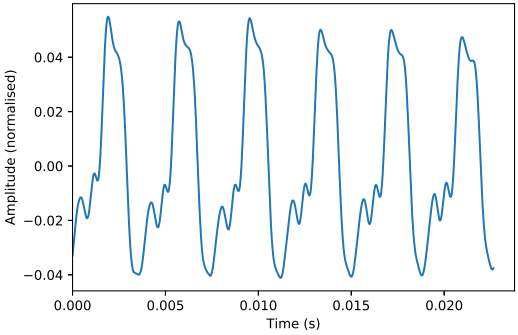


Figure 4. Piano sound wave (middle C).

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

Audio Analysis Example

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

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

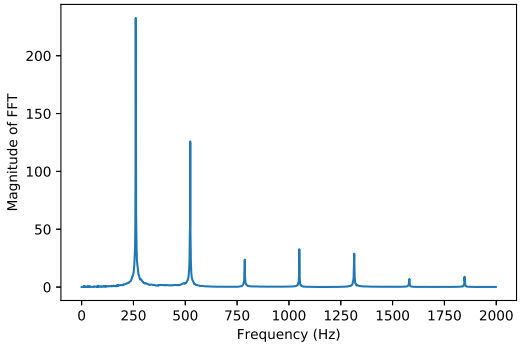


Figure 5. FFT of piano sound wave (middle C).

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).

In mathematics, the Fourier Transform is fundamental to harmonic analyis, an important field related to PDEs, functional analysis, approximation theory, etc.