

Sequential Fast Fourier Transform

(PSC §3.1–3.2)

Applications of Fourier analysis

- ▶ **Fourier analysis** studies the decomposition of functions into their frequency components.
- ▶ Piano Concerto no. 9 by Mozart: enhance high frequencies.
- ▶ Chest picture by Computerised Tomography: reconstruct your interior without slicing you up.
- ▶ Star picture by pre-repair Hubble Space Telescope: remove blur.



Fourier series

- ▶ Let $f : \mathbf{R} \rightarrow \mathbf{C}$ be a T -periodic function:
 $f(t + T) = f(t)$ for all $t \in \mathbf{R}$.
- ▶ **Fourier series** associated with f :

$$\tilde{f}(t) = \sum_{k=-\infty}^{\infty} c_k e^{2\pi i k t / T}.$$

- ▶ **Fourier coefficients** c_k are given by

$$c_k = \frac{1}{T} \int_0^T f(t) e^{-2\pi i k t / T} dt.$$

- ▶ i is the complex number with $i^2 = -1$.
- ▶ Series converges if f is piecewise **smooth** (continuously differentiable).

Fourier series for real-valued function

- Complex Fourier coefficients c_k and corresponding real coefficients a_k, b_k for T -periodic $f : \mathbf{R} \rightarrow \mathbf{R}$ are given by

$$c_k = a_k - ib_k = \frac{1}{T} \int_0^T f(t) \left(\cos \frac{2\pi kt}{T} - i \sin \frac{2\pi kt}{T} \right) dt.$$

- Since $c_{-k} = \overline{c_k}$, $a_k = (c_k + \overline{c_k})/2$, and $b_k = (c_k - \overline{c_k})i/2$:

$$\begin{aligned} \tilde{f}(t) &= \sum_{k=-\infty}^{\infty} c_k e^{2\pi ikt/T} = \sum_{k=1}^{\infty} \overline{c_k} e^{-2\pi ikt/T} + c_0 + \sum_{k=1}^{\infty} c_k e^{2\pi ikt/T} \\ &= c_0 + \sum_{k=1}^{\infty} (\overline{c_k} + c_k) \cos \frac{2\pi kt}{T} + \sum_{k=1}^{\infty} (-\overline{c_k} + c_k) i \sin \frac{2\pi kt}{T} \\ &= a_0 + 2 \sum_{k=1}^{\infty} a_k \cos \frac{2\pi kt}{T} + 2 \sum_{k=1}^{\infty} b_k \sin \frac{2\pi kt}{T}. \end{aligned}$$

It's a discrete world

One second of audio on a compact disc contains 44,100 function values $f(t_j)$ in regularly spaced sample points

$$t_j = \frac{jT}{n}, \quad 0 \leq j < n.$$

Approximation of Fourier coefficients

- ▶ Trapezoidal rule on interval $[t_j, t_{j+1}] = \left[\frac{jT}{n}, \frac{(j+1)T}{n} \right]$:

$$\int_{t_j}^{t_{j+1}} f(t) dt \approx \frac{f(t_j) + f(t_{j+1})}{2} \cdot \frac{T}{n}.$$

- ▶ On the whole interval $[0, T]$:

$$\begin{aligned} c_k &= \frac{1}{T} \int_0^T f(t) e^{-2\pi i k t / T} dt \\ &\approx \frac{1}{T} \cdot \frac{T}{n} \left(\frac{f(0)}{2} + \sum_{j=1}^{n-1} f(t_j) e^{-2\pi i k t_j / T} + \frac{f(T)}{2} \right) \\ &= \frac{1}{n} \sum_{j=0}^{n-1} f(t_j) e^{-2\pi i j k / n} \quad (\text{since } f(0) = f(T) = f(t_0)). \end{aligned}$$

Discrete Fourier transform

- ▶ The **discrete Fourier transform** (DFT) of a vector $\mathbf{x} = (x_0, \dots, x_{n-1})^T$ is the vector $\mathbf{y} = (y_0, \dots, y_{n-1})^T$ with

$$y_k = \sum_{j=0}^{n-1} x_j e^{-2\pi i j k / n} = \sum_{j=0}^{n-1} x_j \omega_n^{jk}, \text{ for } 0 \leq k < n.$$

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

- ▶ Compare:

$$c_k \approx \frac{1}{n} \sum_{j=0}^{n-1} f(t_j) e^{-2\pi i j k / n}$$

Thus $\mathbf{c} \approx \text{DFT}(\mathbf{x})$, where $x_j = f(t_j)/n$.

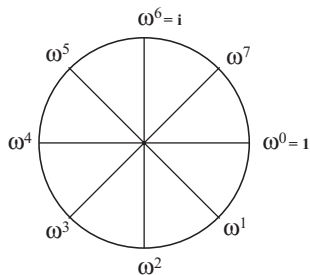
Inverse DFT

- ▶ Easy to prove: the **inverse DFT** (IDFT) of a vector $\mathbf{x} = (x_0, \dots, x_{n-1})^T$ is the vector $\mathbf{y} = (y_0, \dots, y_{n-1})^T$ with

$$y_k = \frac{1}{n} \sum_{j=0}^{n-1} x_j e^{+2\pi i j k / n}, \text{ for } 0 \leq k < n.$$

Same as DFT formula, except for the scaling $1/n$ and the sign of the exponent.

Roots of unity



- ▶ $\omega_8 = e^{-2\pi i/8} = e^{-\pi i/4} = \frac{1}{2}\sqrt{2} - \frac{1}{2}\sqrt{2}i.$
- ▶ $\omega_n^n = e^{-2\pi i n/n} = e^{-2\pi i} = 1.$
- ▶ $\omega_n^{n/2} = e^{-2\pi i(n/2)/n} = e^{-\pi i} = -1.$
- ▶ $\omega_n^2 = e^{-4\pi i/n} = e^{-2\pi i/(n/2)} = \omega_{n/2}.$



Matrix–vector multiplication

- Define the $n \times n$ **Fourier matrix** F_n by

$$(F_n)_{jk} = \omega_n^{jk}, \quad \text{for } 0 \leq j, k < n.$$

- Hence $F_n \mathbf{x} = \text{DFT}(\mathbf{x})$:

$$(F_n \mathbf{x})_j = \sum_{k=0}^{n-1} (F_n)_{jk} x_k = \sum_{k=0}^{n-1} x_k \omega_n^{jk} = (\text{DFT}(\mathbf{x}))_j.$$

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

$$F_4 = \begin{bmatrix} \omega_4^0 & \omega_4^0 & \omega_4^0 & \omega_4^0 \\ \omega_4^0 & \omega_4^1 & \omega_4^2 & \omega_4^3 \\ \omega_4^0 & \omega_4^2 & \omega_4^4 & \omega_4^6 \\ \omega_4^0 & \omega_4^3 & \omega_4^6 & \omega_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}.$$

Cost of straightforward DFT

- ▶ Complex multiplication

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i$$

requires 1 real addition, 1 real subtraction, 4 real multiplications, hence a total of 6 flops.

- ▶ Complex addition

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

requires 2 real additions.

- ▶ To compute y_k , we need n complex multiplications and $n - 1$ complex additions, so $6n + 2(n - 1) = 8n - 2$ flops.
- ▶ To compute the n components of \mathbf{y} , we need $8n^2 - 2n$ flops.



Splitting into even and odd components

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk} = \sum_{j=0}^{n/2-1} x_{2j} \omega_n^{2jk} + \sum_{j=0}^{n/2-1} x_{2j+1} \omega_n^{(2j+1)k}.$$

Using $\omega_n^2 = \omega_{n/2}$ gives

$$y_k = \sum_{j=0}^{n/2-1} x_{2j} \omega_{n/2}^{jk} + \omega_n^k \sum_{j=0}^{n/2-1} x_{2j+1} \omega_{n/2}^{jk}, \quad \text{for } 0 \leq k < n.$$

- ▶ Each sum is a DFT of length $n/2$, for $0 \leq k < n/2$.
- ▶ Thus, we can compute the first half of the DFT by a DFT on the **even components** of \mathbf{x} and a DFT on the **odd components**, and then **combining** the results.
- ▶ Cost is $2 \cdot [8(n/2)^2 - 2(n/2)] + 8(n/2) = 4n^2 + 2n$ flops.



Computing the second half of the DFT

Let $n/2 \leq k < n$. Substituting $k = k' + n/2$ into

$$y_k = \sum_{j=0}^{n/2-1} x_{2j} \omega_{n/2}^{jk} + \omega_n^k \sum_{j=0}^{n/2-1} x_{2j+1} \omega_{n/2}^{jk}$$

gives $0 \leq k' < n/2$ and

$$\begin{aligned} y_{k'+n/2} &= \sum_{j=0}^{n/2-1} x_{2j} \omega_{n/2}^{j(k'+n/2)} + \omega_n^{k'+n/2} \sum_{j=0}^{n/2-1} x_{2j+1} \omega_{n/2}^{j(k'+n/2)} \\ &= \sum_{j=0}^{n/2-1} x_{2j} \omega_{n/2}^{jk'} - \omega_n^{k'} \sum_{j=0}^{n/2-1} x_{2j+1} \omega_{n/2}^{jk'}, \end{aligned}$$

because $\omega_{n/2}^{n/2} = 1$ and $\omega_n^{n/2} = -1$. Now drop the primes.

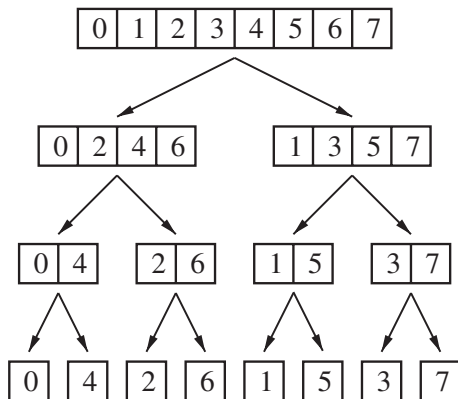


Cost reduction of one split

$$y_{k+n/2} = \sum_{j=0}^{n/2-1} x_{2j} \omega_{n/2}^{jk} - \omega_n^k \sum_{j=0}^{n/2-1} x_{2j+1} \omega_{n/2}^{jk}, \quad \text{for } 0 \leq k < n/2.$$

- ▶ This is the same formula as for the first half, except for the subtraction.
- ▶ Thus, we can compute the second half of the DFT almost **without extra work**, performing just $n/2$ complex subtractions, i.e., n flops.
- ▶ The total cost for the whole DFT with one split is $4n^2 + 3n$ flops, thus saving about half the flops from the original $8n^2 - 2n$.

Recursive computation of DFT



The problem is split repeatedly, until the problem size is 1.



Recursive fast Fourier transform (FFT) algorithm

input: \mathbf{x} : vector of length n .
output: \mathbf{y} : vector of length n , $\mathbf{y} = F_n \mathbf{x}$.
call: $\mathbf{y} := \text{FFT}(\mathbf{x}, n)$.

if $n \bmod 2 = 0$ **then**

$\mathbf{x}^e := x(0 : 2 : n - 1)$; $\mathbf{y}^e := \text{FFT}(\mathbf{x}^e, n/2)$;

$\mathbf{x}^o := x(1 : 2 : n - 1)$; $\mathbf{y}^o := \text{FFT}(\mathbf{x}^o, n/2)$;

for $k := 0$ **to** $n/2 - 1$ **do**

$\tau := \omega_n^k y_k^o$;

$y_k := y_k^e + \tau$;

$y_{k+n/2} := y_k^e - \tau$;

else $\mathbf{y} := \text{DFT}(\mathbf{x}, n)$;

$$y_k = \sum_{j=0}^{n/2-1} x_{2j} \omega_{n/2}^{jk} + \omega_n^k \sum_{j=0}^{n/2-1} x_{2j+1} \omega_{n/2}^{jk}, \quad \text{for } 0 \leq k < n/2.$$

Sequential FFT



Cost of fast Fourier transform

- ▶ Loop has complex multiplication, addition, subtraction, together $6 + 2 + 2 = 10$ flops.
- ▶ $n/2$ iterations of loop, hence a total of $n/2 \cdot 10 = 5n$ flops.
- ▶ Perform 2 FFT($n/2$) operations and $5n$ flops for FFT(n):

$$\begin{aligned}T(n) &= 2T\left(\frac{n}{2}\right) + 5n \\&= 2\left(2T\left(\frac{n}{4}\right) + 5\frac{n}{2}\right) + 5n = 4T\left(\frac{n}{4}\right) + 2 \cdot 5n \\&= \dots = nT(1) + (\log_2 n) \cdot 5n = 5n \log_2 n.\end{aligned}$$

- ▶ Much faster than $8n^2$ time for direct computation of DFT.
- ▶ For $n = 2^{27} = 134,217,728$ (50 min 43 s audio, an average CD), an FFT can be done in 18 s on a 1 Gflop/s PC, but it would take over 4.5 years using the straightforward DFT.



Summary

- ▶ The fast Fourier transform (FFT) idea was discovered by Gauss (1805), rediscovered by Danielson and Lanczos (1942), and is commonly attributed to [Cooley and Tukey](#) (1965), who rediscovered it in the digital era.
- ▶ The FFT is the [computational workhorse](#) in many applications, from weather forecasting to signal and image processing. Without the FFT, modern medicine would be impossible.
- ▶ The cost of an FFT of length n is $5n \log_2 n$ flops.
- ▶ We have derived a [recursive](#) FFT algorithm, i.e., an algorithm that calls itself with a smaller problem size.

