13 Fast Fourier Transform (FFT)

The fast Fourier transform (FFT) is an algorithm for the efficient implementation of the discrete Fourier transform. We begin our discussion once more with the continuous Fourier transform.

13.1 Continuous and Discrete Fourier Transforms Revisited

Let E_k be the complex exponential defined by

$$E_k(x) := e^{ikx}$$

and consider the complex inner product

$$\langle f, g\rangle = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)\overline{g(x)}dx.$$

Then

$$\langle E_k, E_k\rangle = \frac{1}{2\pi}\int_{-\pi}^{\pi} \underbrace{e^{ikx}e^{-ikx}}_{=1}dx = 1,$$

and for $k \neq \ell$ we have

$$\begin{aligned}
\langle E_k, E_\ell\rangle &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{ikx}e^{-i\ell x}dx \\
&= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{i(k-\ell)x}dx \\
&= \frac{1}{2\pi}\left.\frac{e^{i(k-\ell)x}}{i(k-\ell)}\right|_{-\pi}^{\pi} \\
&= \frac{1}{2\pi i(k-\ell)}\left[e^{i(k-\ell)\pi} - e^{-i(k-\ell)\pi}\right] \\
&= \frac{1}{2\pi i(k-\ell)}e^{i(k-\ell)\pi}\left[1 - \underbrace{e^{2\pi(\ell-k)i}}_{=1}\right] = 0.
\end{aligned}$$

As we just saw, the complex exponentials are orthonormal on the interval $[-\pi, \pi]$. In fact, they are used as the basis functions for complex Fourier series

$$f(x) \sim \sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

with Fourier coefficients

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)e^{-ikt}dt = \langle f, E_k\rangle = \hat{f}(k).$$

As mentioned earlier, the coefficients $c_k = \hat{f}(k)$ are also known as the *Fourier transform* of f.

Remark Note that the notation used here is a bit different from that used in Chapter 11. In particular, earlier we used the interval $(-\infty, \infty)$.

In order to get to the discrete Fourier transform we first truncate the Fourier series and obtain

$$P(x) = \sum_{k=0}^{n} c_k E_k(x)$$

— a *trigonometric polynomial.* This can be verified by writing

$$P(x) = \sum_{k=0}^{n} c_k e^{ikx} = \sum_{k=0}^{n} c_k \left(e^{ix}\right)^k$$

so that P is a polynomial of degree n in the complex exponential e^{ix}.

We now also consider equally spaced data in the interval $[0, 2\pi)$, i.e.,

$$(x_j, f(x_j)) \quad \text{with} \quad x_j = \frac{2\pi j}{N},\ j = 0, 1, \ldots, N-1, \tag{122}$$

and introduce a discrete (pseudo-)inner product

$$\langle f, g\rangle_N = \frac{1}{N} \sum_{j=0}^{N-1} f\left(\frac{2\pi j}{N}\right) \overline{g\left(\frac{2\pi j}{N}\right)}. \tag{123}$$

Remark This is only a pseudo-inner product since $\langle f, f\rangle_N = 0$ implies $f = 0$ only at the discrete nodes $\frac{2\pi j}{N}$, $j = 0, 1, \ldots, N-1$.

Theorem 13.1 *The set $\{E_k\}_{k=0}^{N-1}$ of complex exponentials is orthonormal with respect to the inner product $\langle\ ,\ \rangle_N$ defined in (123).*

Proof First,

$$\begin{aligned}
\langle E_k, E_k\rangle_N &= \frac{1}{N} \sum_{j=0}^{N-1} E_k\left(\frac{2\pi j}{N}\right) \overline{E_k\left(\frac{2\pi j}{N}\right)} \\
&= \frac{1}{N} \sum_{j=0}^{N-1} \underbrace{e^{ik2\pi j/N} e^{-ik2\pi j/N}}_{=1} = 1.
\end{aligned}$$

Next, for $k \neq \ell$ we have

$$\begin{aligned}
\langle E_k, E_\ell\rangle_N &= \frac{1}{N} \sum_{j=0}^{N-1} E_k\left(\frac{2\pi j}{N}\right) \overline{E_\ell\left(\frac{2\pi j}{N}\right)} \\
&= \frac{1}{N} \sum_{j=0}^{N-1} e^{ik2\pi j/N} e^{-i\ell 2\pi j/N} \\
&= \frac{1}{N} \sum_{j=0}^{N-1} \left(e^{i\frac{2\pi(k-\ell)}{N}}\right)^j.
\end{aligned}$$

Now, since $k \neq \ell$,

$$e^{i\frac{2\pi(k-\ell)}{N}} \neq 1$$

we can apply the formula for the sum of a finite geometric series to get

$$\langle E_k, E_\ell \rangle_N = \frac{1}{N} \frac{\left(e^{i\frac{2\pi(k-\ell)}{N}}\right)^N - 1}{e^{i\frac{2\pi(k-\ell)}{N}} - 1}.$$

Finally, since

$$\left(e^{i\frac{2\pi(k-\ell)}{N}}\right)^N = 1$$

the orthogonality result follows. ■

This gives rise to a best approximation result (just as for standard Fourier series):

Corollary 13.2 *Let data be given as in (122) and consider* $G = \operatorname{span}\{E_k\}_{k=0}^n$ *with* $n < N$*. Then the best least squares approximation to the data from G (with respect to the inner product (123)) is given by the discrete Fourier series*

$$g(x) = \sum_{k=0}^{n} c_k E_k(x)$$

with

$$c_k = \langle f, E_k \rangle_N = \frac{1}{N} \sum_{j=0}^{N-1} f\left(\frac{2\pi j}{N}\right) e^{-ik\frac{2\pi j}{N}}, \qquad k = 0, 1, \ldots, n.$$

Remark

1. The coefficients c_k of the discrete Fourier series are called the *discrete Fourier transform* (DFT) of f.
2. It is our goal to compute the DFT (and also evaluate g) via the fast Fourier transform (FFT).
3. Note that, even if the data is real, the DFT will in general be complex. If, e.g., $N = 4$, and the values $f\left(\frac{2\pi j}{N}\right)$, $j = 0, 1, 2, 3$ are $4, 0, 3, 6$ respectively, then the coefficients c_k are

$$\begin{aligned} c_0 &= 13 \\ c_1 &= 1 + 6i \\ c_2 &= 1 \\ c_3 &= 1 - 6i. \end{aligned}$$

13.2 The FFT Algorithm

We start with the trigonometric polynomial

$$P(x) = \sum_{k=0}^{N-1} c_k E_k(x)$$

with DFT coefficients

$$c_k = \frac{1}{N} \sum_{j=0}^{N-1} f\left(\frac{2\pi j}{N}\right) e^{-ik\frac{2\pi j}{N}}$$

$$= \frac{1}{N}\sum_{j=0}^{N-1} f(x_j)(\lambda_k)^j$$

where $x_j = \frac{2\pi j}{N}$ and we set

$$\lambda_k = e^{-i\frac{2\pi k}{N}}.$$

Remark The values λ_k (or more naturally λ_{-k}) are often referred to as the N-*th roots of unity*. This is illustrated in Figure 8 for $N = 5$.

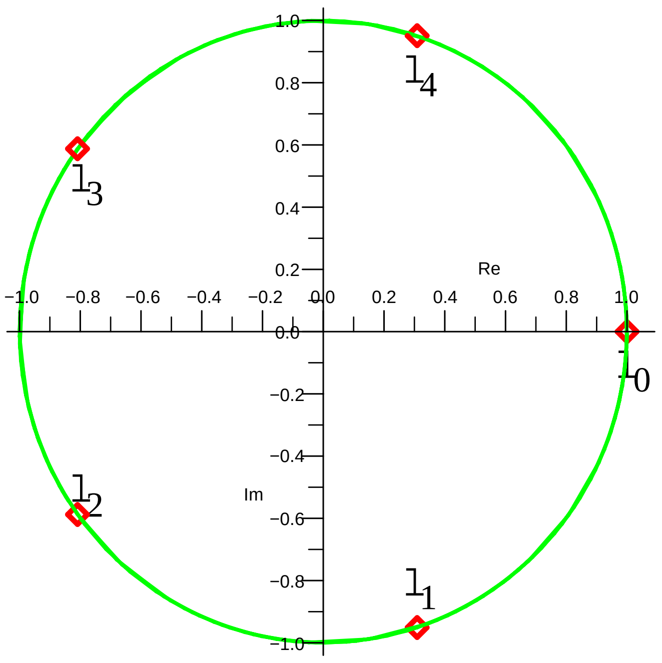

Figure 8: Roots of unity λ_k, $k = 0, 1, \dots, N-1$, for $N = 5$.

The computation of each coefficient c_k corresponds to evaluation of a polynomial of degree $N-1$ in λ_k. Using nested multiplication (Horner's method) this can be done in $\mathcal{O}(N)$ operations. Since we have N coefficients, the polynomial P can be constructed in $\mathcal{O}(N^2)$ operations.

The major benefit of the fast Fourier transform is that it reduces the amount of work to $\mathcal{O}(N \log_2 N)$ operations.

The FFT was discovered by Cooley and Tukey in 1965. However, Gauss seemed to already be aware of similar ideas. One of the most popular modern references is the "DFT Owners Manual" by Briggs and Henson (published by SIAM in 1995). It is the dramatic reduction in computational complexity that earned the FFT a spot on the list of "Top 10 Algorithms of the 20th Century" (see handout).

The FFT is based on the following observation:

Theorem 13.3 *Let data* $(x_k, f(x_k))$, $x_k = \frac{k\pi}{n}$, $k = 0, 1, \dots, 2n-1$, *be given, and assume that* p *and* q *are exponential polynomials of degree at most* $n-1$ *which interpolate* *part of the data according to*

$$p(x_{2j}) = f(x_{2j}), \qquad q(x_{2j}) = f(x_{2j+1}), \qquad j = 0, 1, \ldots, n-1,$$

then the exponential polynomial of degree at most $2n-1$ *which interpolates all the given data is*

$$P(x) = \frac{1}{2}\left(1 + e^{inx}\right) p(x) + \frac{1}{2}\left(1 - e^{inx}\right) q\left(x - \frac{\pi}{n}\right). \tag{124}$$

Proof Since $e^{inx} = \left(e^{ix}\right)^n$ has degree n, and p and q have degree at most $n-1$ it is clear that P has degree at most $2n-1$.

We need to verify the interpolation claim, i.e., show that

$$P(x_k) = f(x_k), \qquad k = 0, 1, \ldots, 2n-1.$$

By assumption we have

$$P(x_k) = \frac{1}{2}\left(1 + e^{inx_k}\right) p(x_k) + \frac{1}{2}\left(1 - e^{inx_k}\right) q\left(x_k - \frac{\pi}{n}\right)$$

where

$$e^{inx_k} = e^{in\frac{k\pi}{n}} = \left(e^{i\pi}\right)^k = (-1)^k.$$

Therefore

$$P(x_k) = \begin{cases} p(x_k) & \text{if } k \text{ even,} \\ q\left(x_k - \frac{\pi}{n}\right) & \text{if } k \text{ odd.} \end{cases}$$

Let k be even, i.e., $k = 2j$. Then, by the assumption on p

$$P(x_{2j}) = p(x_{2j}) = f(x_{2j}), \qquad j = 0, 1, \ldots, n-1.$$

On the other hand, for $k = 2j+1$ (odd), we have (using the assumption on q)

$$P(x_{2j+1}) = q\left(x_{2j+1} - \frac{\pi}{n}\right) = q(x_{2j}) = f(x_{2j+1}), \qquad j = 0, 1, \ldots, n-1.$$

This is true since

$$x_{2j+1} - \frac{\pi}{n} = \frac{(2j+1)\pi}{n} - \frac{\pi}{n} = \frac{2j\pi}{n} = x_{2j}.$$

■

The second useful fact tells us how to obtain the coefficients of P from those of p and q.

Theorem 13.4 *Let*

$$p = \sum_{j=0}^{n-1} \alpha_j E_j, \qquad q = \sum_{j=0}^{n-1} \beta_j E_j, \qquad \textit{and} \qquad P = \sum_{j=0}^{2n-1} \gamma_j E_j.$$

Then, for $j = 0, 1, \ldots, n-1$,

$$\begin{aligned} \gamma_j &= \frac{1}{2}\alpha_j + \frac{1}{2}e^{-ij\pi/n}\beta_j \\ \gamma_{j+n} &= \frac{1}{2}\alpha_j - \frac{1}{2}e^{-ij\pi/n}\beta_j. \end{aligned}$$

Proof In order to use (124) we need to rewrite

$$\begin{aligned} q\left(x-\frac{\pi}{n}\right) &= \sum_{j=0}^{n-1} \beta_j E_j\left(x-\frac{\pi}{n}\right) \\ &= \sum_{j=0}^{n-1} \beta_j e^{ij\left(x-\frac{\pi}{n}\right)} \\ &= \sum_{j=0}^{n-1} \beta_j e^{ijx} e^{-ij\pi/n}. \end{aligned}$$

Therefore

$$\begin{aligned} P(x) &= \frac{1}{2}\left(1+e^{inx}\right)p(x) + \frac{1}{2}\left(1-e^{inx}\right) q\left(x-\frac{\pi}{n}\right) \\ &= \frac{1}{2}\sum_{j=0}^{n-1}\left(1+e^{inx}\right)\alpha_j e^{ijx} + \frac{1}{2}\sum_{j=0}^{n-1}\left(1-e^{inx}\right)\beta_j e^{ijx}e^{-ij\pi/n} \\ &= \frac{1}{2}\sum_{j=0}^{n-1}\left[\left(\alpha_j+\beta_j e^{-ij\pi/n}\right)e^{ijx} + \left(\alpha_j-\beta_j e^{-ij\pi/n}\right)e^{i(n+j)x}\right]. \end{aligned}$$

However, the terms in parentheses (together with the factor 1/2) lead precisely to the formulæ for the coefficients γ. ■

Example We consider the data

x	0	π
y	$f(0)$	$f(\pi)$

.

Now we apply the DFT to find the interpolating polynomial P. We have $n = 1$ and

$$P(x) = \sum_{j=0}^{1} \gamma_j E_j(x) = \gamma_0 e^0 + \gamma_1 e^{ix}.$$

According to Theorem 13.4 the coefficients are given by

$$\begin{aligned} \gamma_0 &= \frac{1}{2}\left(\alpha_0+\beta_0 e^0\right), \\ \gamma_1 &= \frac{1}{2}\left(\alpha_0-\beta_0 e^0\right). \end{aligned}$$

Therefore, we still need to determine α_0 and β_0. They are found via interpolation at only one point (cf. Theorem 13.3):

$$\begin{aligned} p(x) &= \sum_{j=0}^{0}\alpha_j E_j(x) = \alpha_0 \quad \text{such that} \quad p(x_0) = f(x_0) \quad \Longrightarrow \quad \alpha_0 = f(x_0) \\ q(x) &= \sum_{j=0}^{0}\beta_j E_j(x) = \beta_0 \quad \text{such that} \quad q(x_0) = f(x_1) \quad \Longrightarrow \quad \beta_0 = f(x_1). \end{aligned}$$

With $x_0 = 0$ and $x_1 = \pi$ we obtain

$$P(x) = \frac{1}{2}\left(f(0)+f(\pi)\right) + \frac{1}{2}\left(f(0)-f(\pi)\right)e^{ix}.$$

Example We now refine the previous example, i.e., we consider the data

x	0	$\frac{\pi}{2}$	π	$\frac{3\pi}{2}$
y	$f(0)$	$f\left(\frac{\pi}{2}\right)$	$f(\pi)$	$f\left(\frac{3\pi}{2}\right)$

.

Now $n = 2$, and P is of the form

$$P(x) = \sum_{j=0}^{3} \gamma_j E_j(x).$$

According to Theorem 13.4 the coefficients are given by

$$\begin{aligned}
\gamma_0 &= \frac{1}{2}\left(\alpha_0 + \beta_0\right), \\
\gamma_1 &= \frac{1}{2}\Big(\alpha_1 + \beta_1 \underbrace{e^{-i\pi/2}}_{=-i}\Big), \\
\gamma_2 &= \frac{1}{2}\left(\alpha_0 - \beta_0\right), \\
\gamma_3 &= \frac{1}{2}\Big(\alpha_1 - \beta_1 \underbrace{e^{-i\pi/2}}_{=-i}\Big).
\end{aligned}$$

This leaves the coefficients α_0, β_0 and α_1, β_1 to be determined. They are found via interpolation at two points (cf. Theorem 13.3):

$$\begin{aligned}
p(x) &= \sum_{j=0}^{1} \alpha_j E_j(x) \quad \text{such that} \quad p(x_0) = f(x_0),\ p(x_2) = f(x_2), \\
q(x) &= \sum_{j=0}^{1} \beta_j E_j(x) \quad \text{such that} \quad q(x_0) = f(x_1),\ q(x_2) = f(x_3).
\end{aligned}$$

These are both problems of the form dealt with in the previous example, so a recursive strategy is suggested.

A general recursive algorithm for the FFT is given in the Kincaid/Cheney textbook on pages 455 and 456.

In order to figure out the *computational complexity* of the FFT algorithm we assume $N(= 2n) = 2^m$ so that the recursive strategy for the FFT can be directly applied. (If N is not a power of 2, then the data can be padded appropriately with zeros). In order to evaluate the interpolating exponential polynomial

$$P(x) = \sum_{k=0}^{N-1} c_k E_k(x) = \sum_{j=0}^{2n-1} \gamma_j E_j(x)$$

we can use the FFT to compute the coefficients c_k (or γ_j).

Instead of giving an exact count of the arithmetic operations involved in this task, we estimate the minimum number of multiplications required to compute the coefficients of an exponential polynomial with N terms.

Lemma 13.5 *Let $R(N)$ be the minimum number of multiplications required to compute the coefficients of an exponential polynomial with $N = 2n$ terms. Then*

$$R(N) \leq 2R(n) + 2n.$$

Proof There are N coefficients γ_j computed recursively via the formulas

$$\begin{aligned} \gamma_j &= \frac{1}{2}\alpha_j + \frac{1}{2}e^{-ij\pi/n}\beta_j \\ \gamma_{j+n} &= \frac{1}{2}\alpha_j - \frac{1}{2}e^{-ij\pi/n}\beta_j \end{aligned}$$

listed in Theorem 13.4. If the factors $\frac{1}{2}e^{-ij\pi/n}$, $n = 0, 1, \ldots, n-1$, are pre-computed, then each of these formulas involves 2 multiplications for a total of $2n$ multiplications. Moreover, the coefficients α_j and β_j are determined recursively using $R(n)$ multiplications each. ■

Theorem 13.6 *Under the same assumptions as in Lemma 13.5 we have*

$$R(2^m) \leq m2^m.$$

Proof We use induction on m. For $m = 0$ there are no multiplications since $P(x) = c_0$. Now, assume the inequality holds for m. Then, by Lemma 13.5 (with $n = 2^m$)

$$R\left(2^{m+1}\right) = R\left(2 \cdot 2^m\right) \leq 2R\left(2^m\right) + 2 \cdot 2^m.$$

By the induction hypothesis this is less than or equal to

$$2m2^m + 2 \cdot 2^m = (m+1)2^{m+1}.$$

■

Setting $N = 2^m \Leftrightarrow m = \log_2 N$ in Theorem 13.6 implies

$$R(N) \leq N \log_2 N,$$

i.e., the fast Fourier transform reduces the amount of computation required to evaluate an exponential polynomial from $\mathcal{O}(N^2)$ (if done directly) to $\mathcal{O}(N \log_2 N)$. For $N = 1000$ this means roughly 10^4 instead of 10^6 operations.

Remark

1. Application of the classical fast Fourier transform is limited to the case where the data is equally spaced, periodic, and of length 2^m. The often suggested strategy of padding with zeros (to get a data vector of length 2^m) or the application of the FFT to non-periodic data may produce unwanted noise.

2. Various version of the FFT for non-equally spaced data have also been proposed in recent years (e.g., NFFT by Daniel Potts, Gabriele Steidl and Manfred Tasche).

Remark Given a vector $a = [a_0, a_1, \ldots, a_{N-1}]^T$ of discrete data, Corollary 13.2 tells us that the discrete Fourier transform $\hat{a}$ of a is computed componentwise as

$$\hat{a}_k = \frac{1}{N} \sum_{j=0}^{N-1} a_j e^{-ik\frac{2\pi j}{N}}, \qquad k = 0, 1, \ldots, N-1.$$

Of course, the FFT can be used to compute these coefficients. There is an analogous formula for the inverse DFT, i.e.,

$$a_j = \sum_{k=0}^{N-1} \hat{a}_k e^{ik\frac{2\pi j}{N}}, \qquad j = 0, 1, \ldots, N-1.$$

We close with a few applications of the FFT. Use of the FFT for many applications benefits from the fact that convolution in the function domain corresponds to multiplication in the transform domain, i.e.,

$$\widehat{f * g} = \hat{f}\hat{g}$$

so that we have

$$f * g = \left(\hat{f}\hat{g}\right)^{\vee}.$$

Some applications are now

- Filters in signal or image processing.
- Compression of audio of video signals. E.g., the MP3 file format makes use of the discrete cosine transform.
- De-noising of data contaminated with measurement error (see, e.g., the Maple worksheet `478578_FFT.mws`).
- Multiplication with cyclic matrices. A cyclic matrix is of the form

$$C = \begin{bmatrix} a_0 & a_1 & \ldots & a_{n-1} & a_n \\ a_n & a_0 & a_1 & \ldots & a_{n-1} \\ a_{n-1} & a_n & a_0 & a_1 & \ldots \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ a_1 & \ldots & a_{n-1} & a_n & a_0 \end{bmatrix}$$

  and multiplication by C corresponds to a discrete convolution, i.e.,

$$(Cy)_i = \sum_{j=0}^{n} C_{ij} y_j = \sum_{j=0}^{n} a_{j-i} y_j, \qquad i = 0, \ldots, n,$$

  where the subscript on a is to be interpreted cyclically, i.e., $a_{n+1+i} \equiv a_i$ or $a_{-i} \equiv a_{n+1-i}$. Thus, a coupled system of linear equations with cyclic system matrix can be decoupled (and thus trivially solved) by applying the discrete Fourier transform.

- High precision integer arithmetic also makes use of the FFT.
- Numerical solution of differential equations.

Remark Another transform technique that has some similarities to (but also some significant differences from) the FFT is the fast wavelet transform. Using so-called scaling functions and associated wavelets as orthogonal basis functions one can also perform best approximation of functions (or data). An advantage of wavelets over the trigonometric basis used for the FFT is the fact that not only do we have localization in the transform domain, but also in the function domain.