FFT: Fast Fourier Transform

arnaucube

August 2022

Abstract

Usually while reading papers and books I take handwritten notes, this document contains some of them re-written to $LaTeX$.

The notes are not complete, don't include all the steps neither all the proofs. I use these notes to revisit the concepts after some time of reading the topic.

This document are notes done while reading about the topic from [1], [2], [3].

Contents

1 Discrete & Fast Fourier Transform

1.1 Discrete Fourier Transform (DFT)

Continuous:

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

Discrete: The k^{th} frequency, evaluating at n of N samples.

$$\hat{f}_k = \sum_{n=0}^{n-1} f_n e^{\frac{-j\pi k n}{N}}$$

where we can group under $b_n = \frac{\pi k n}{N}$. The previous expression can be expanded into:

$$x_k = x_0 e^{-b_0 j} + x_1 e^{-b_1 j} + ... + x_n e^{-b_n j}$$

By the *Euler's formula* we have $e^{jx} = cos(x) + j \cdot sin(x)$, and using it in the previous x_k, we obtain

$$x_k = x_0[cos(-b_0) + j \cdot sin(-b_0)] + \dots$$

Using $\hat{f}_k$ we obtained

$$\{f_0, f_1, \dots, f_N\} \xrightarrow{DFT} \{\hat{f}_0, \hat{f}_1, \dots, \hat{f_N}\}$$

To reverse the $\hat{f}_k$ back to f_k:

$$f_k = \left(\sum_{n=0}^{n-1} \hat{f}_n e^{\frac{-j\pi k n}{N}} \right) \cdot \frac{1}{N}$$

$$DFT = \begin{pmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_n \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w_n & w_n^2 & \dots & w_n^{N-1} \\ 1 & w_n^2 & w_n^4 & \dots & w_n^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & \dots & w_n^{(N-1)^2} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

1.2 Fast Fourier Transform (FFT)

While DFT is $O(n)$, FFT is $O(nlog(n))$

Here you can find a simple implementation of the these concepts in Rust: arnaucube/fft-rs [4]

2 FFT over finite fields, roots of unity, and polynomial multiplication

FFT is very useful when working with polynomials. [TODO poly multiplication]

An implementation of the FFT over finite fields using the Vandermonde matrix approach can be found at [5].

2.1 Intro

Let $A(x)$ be a polynomial of degree $n-1$,

$$A(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{n-1} \cdot x^{n-1} = \sum_{i=0}^{n-1} a_i \cdot x^i$$

We can represent $A(x)$ in its evaluation form,

$$(x_0, A(x_0)), (x_1, A(x_1)), \cdots, (x_{n-1}, A(x_{n-1})) = (x_i, A(x_i))$$

We can evaluate A(x) at n given points $(x_0, x_1, ..., x_{n-1})$:

$$\begin{pmatrix} A(x_0) \\ A(x_1) \\ A(x_2) \\ \vdots \\ A(x_{n-1}) \end{pmatrix} = \begin{pmatrix} x_0^0 & x_0^1 & x_0^2 & \dots & x_0^{n-1} \\ x_1^0 & x_1^1 & x_1^2 & \dots & x_1^{n-1} \\ x_2^0 & x_2^1 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n-1}^0 & x_{n-1}^1 & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

This is known by the Vandermonde matrix.

But this will not be too efficient. Instead of random x_i values, we use *roots of unity*, where $\omega_n^n = 1$. We denote ω as a primitive n^{th} root of unity:

$$\begin{pmatrix} A(1) \\ A(\omega) \\ A(\omega^2) \\ \vdots \\ A(\omega^{n-1}) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \dots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Which we can see as

$$\hat{A} = F_n \cdot A$$

This matches our system of equations:

- at $x = 0$, $a_0 + a_1 + \cdots + a_{n-1} = A_0 = A(1)$
- at $x = 1$, $a_0 \cdot 1 + a_1 \cdot \omega + a_2 \cdot \omega^2 + \cdots + a_{n-1} \cdot \omega^{n-1} = A_1 = A(\omega)$
- at $x = 2$, $a_0 \cdot 1 + a_1 \cdot \omega^2 + a_2 \cdot \omega^4 + \cdots + a_{n-1} \cdot \omega^{2(n-1)} = A_2 = A(\omega^2)$
- $\cdots$
- at $x = n-1$, $a_0 \cdot 1 + a_1 \cdot \omega^{n-1} + a_2 \cdot \omega^{2(n-1)} + \cdots + a_{n-1} \cdot \omega^{(n-1)(n-1)} = A_2 = A(\omega^{n-1})$

We denote the F_n as the Fourier matrix, with j rows and k columns, where each entry can be expressed as $F_{jk} = \omega^{jk}$.

To find the a_i values, we use the inverted $F_n = F_n^{-1}$

2.2 Roots of unity

todo

2.3 FFT over finite fields

todo

2.4 Polynomial multiplication with FFT

todo

References

[1] Linear algebra and its applications, by gilbert strang (chapter 3.5). https://archive.org/details/linearalgebrait00stra.

[2] Thomas Pornin mathoverflow answer. https://crypto.stackexchange.com/a/63616.

[3] notes by Prof. R. Fateman. https://www.csee.umbc.edu/~phatak/691a/fft-lnotes/fftnotes.pdf.

[4] fft-rs. https://github.com/arnaucube/fft-rs.

[5] fft-sage. https://github.com/arnaucube/math/blob/master/fft.sage.