Introduction of Fast Fourier Transform [1]

Tobramycin

NCUT

2022.7

南開大學
Nankai University

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. of Computation, vol. 19, pp. 297–301, 1965

Contents

FFT Experiment

FFT Input and Output

真实的函数：

$$S = 2 + 3 * cos(2 * \pi * 50 * t - \pi * 30/180) + 1.5 * cos(2 * \pi * 75 * t + \pi * 90/180) \quad (1)$$

采样点数：设 1s 内采样点为 256 个，则采样频率为 256Hz，分辨 1Hz 频率

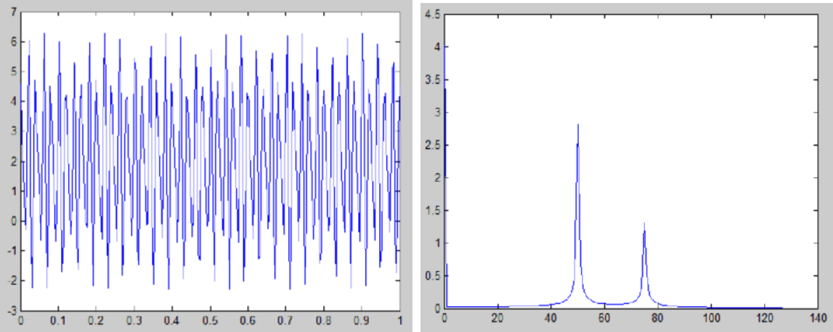


Figure: Origin and Spectrum

Output 计算公式

FFT 幅度：毕达哥拉斯公式

$$\text{Magnitude} = \sqrt{\text{Real}^{2} + \text{Imaginary}^{2}} \tag{2}$$

FFT 相位：反正切规则

$$\text{Phase} = \tan^{-1}\left(\frac{\text{Imaginary}}{\text{Real}}\right) \tag{3}$$

Output 计算结果

fft input: [256,], fft output: [256,], 采样定理：采样频率 $\leq 2*$ 信号频率

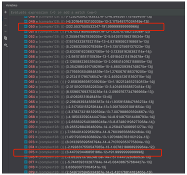


Figure: FFT Result

- 0Hz 分量复数表示：511.999
- 50Hz 分量复数表示：$332 - 191j$
- 75Hz 分量复数表示：$3.44e^{12} + 191j$
- 0Hz 模值表示：$511.999 * \frac{2}{N} = 4$
- 50Hz 模值表示：$384 * \frac{2}{N} = 3$
- 75Hz 模值表示：$192 * \frac{2}{N} = 1.5$

类似，相位计算得：0, -30, 90

因此，原函数

$S = 2 + 3 * cos(2 * \pi * 50 * t - \pi * 30/180) + 1.5 * cos(2 * \pi * 75 * t + \pi * 90/180)$

FFT Introduction

FFT Introduction

傅立叶变换 (FT)：连续时间 & 连续频率

$$X(f) = \int_{-\infty}^{+\infty} x(t) \cdot e^{-i2\pi ft} \cdot dt \tag{4}$$

离散傅立叶变换 (DFT)：离散时间 & 离散频率

$$X(k) = \sum_{n=0}^{N-1} x(n) \times e^{-i2\pi \frac{k}{N} n}, k = 0, 1, ..., N-1 \tag{5}$$

n 表示离散时间，k 表示离散频率

1. Vandermonde Matrix

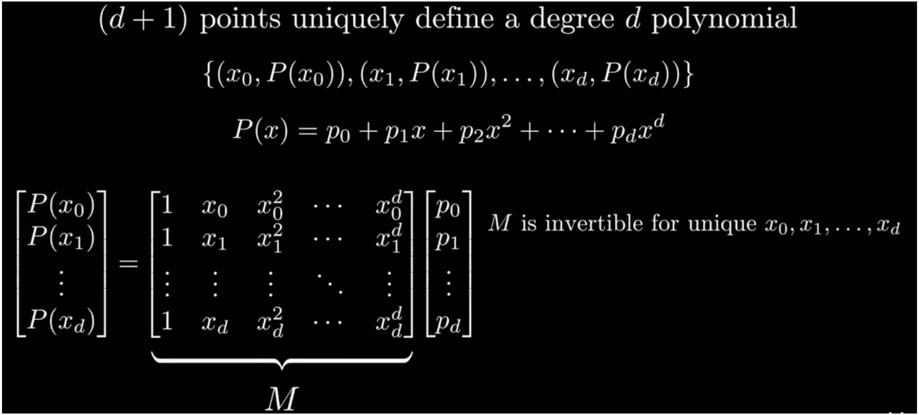


Figure: Vandermonde Matrix[2]

[2] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb

1. Vandermonde Matrix

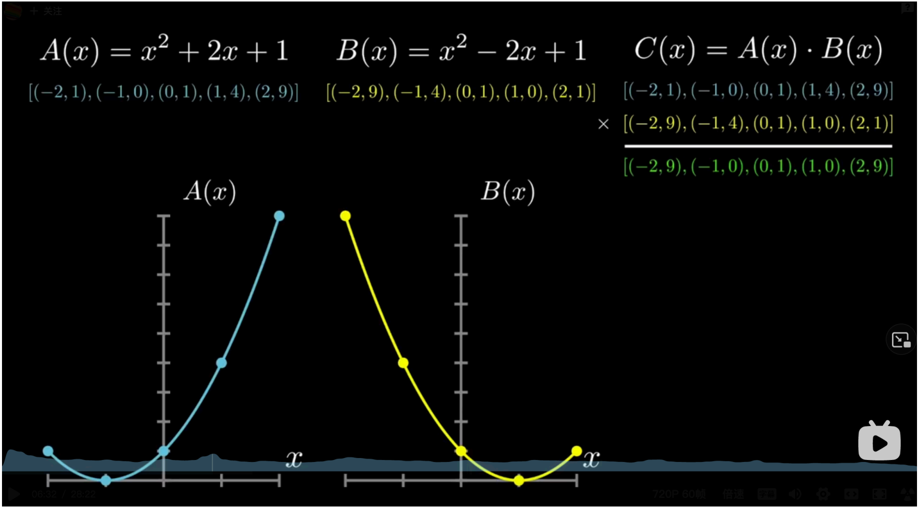


Figure: 点值表示法[3]

2. 对称结构

$$P(x) = p_0 + p_1 x + p_2 x^2 + \cdots + p_{n-1} x^{n-1}$$

Evaluate at n points $\pm x_1, \pm x_2, \ldots, \pm x_{n/2}$

$$P(x) = P_e(x^2) + x P_o(x^2)$$

$$\left.\boxed{\begin{aligned} P(x_i) &= P_e(x_i^2) + x_i P_o(x_i^2) \\ P(-x_i) &= P_e(x_i^2) - x_i P_o(x_i^2) \end{aligned}}\right\} \text{Lot of overlap!}$$

$P_e(x^2)$ and $P_o(x^2)$ have degree $n/2 - 1$!

$$\underbrace{\text{Evaluate } P_e(x^2) \text{ and } P_o(x^2) \text{ each at } x_1^2, x_2^2, \ldots, x_{n/2}^2 \ (n/2 \text{ points})}_{\text{Same process on simpler problem}}$$

Figure: 对称结构[4]

[4] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb_logo

2. 对称结构

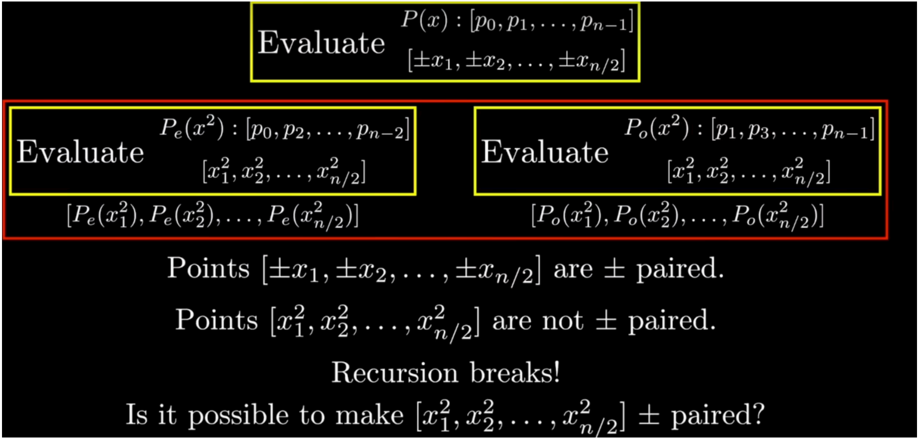


Figure: 对称结构[5]

[5] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb_logo

3. 复数引入

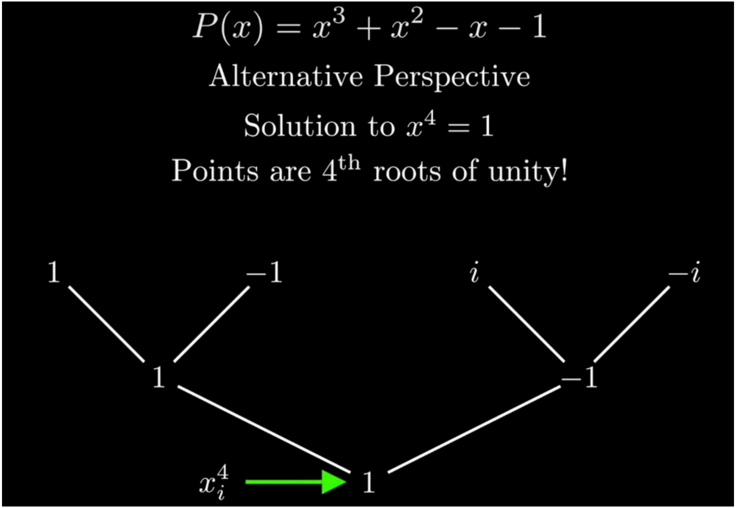


Figure: 复数引入[6]

3. 复数引入

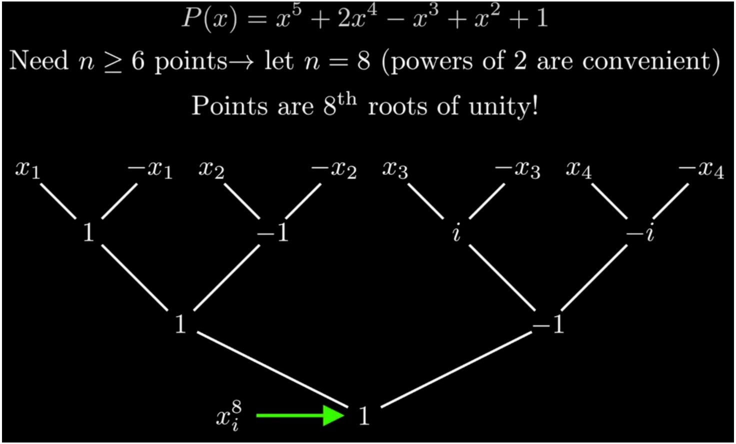


Figure: 复数引入[7]

[7] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb$_logo$

4. 如何选取复数采样点

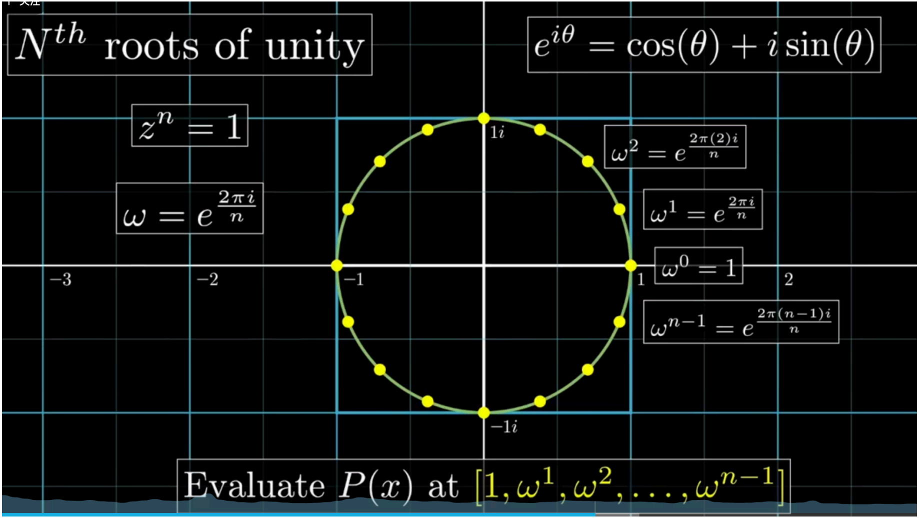
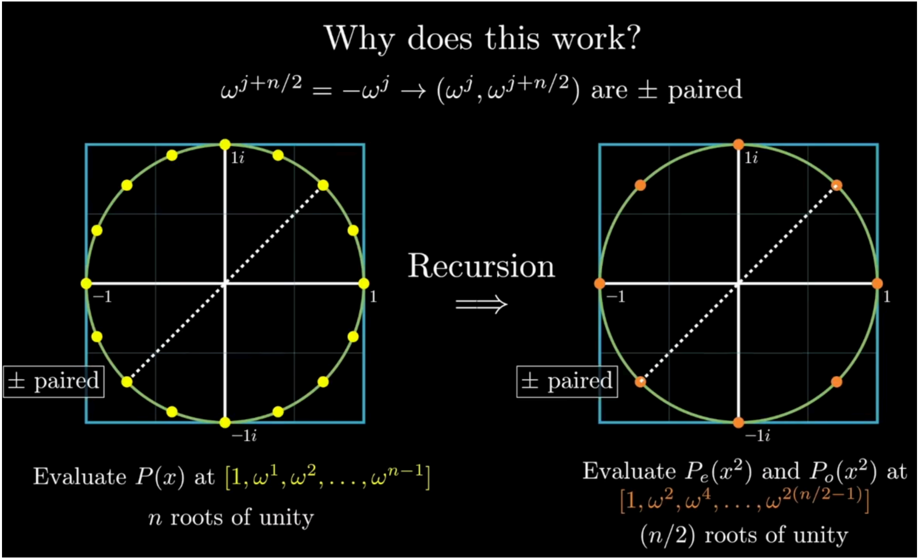


Figure: 复数引入[8]

4. 如何选取复数采样点


Why does this work?
$\omega^{j+n/2} = -\omega^j \rightarrow (\omega^j, \omega^{j+n/2})$ are $\pm$ paired
1i
−1
1
−1i
± paired
Recursion
$\Longrightarrow$
1i
−1
1
−1i
± paired
Evaluate $P(x)$ at $[1, \omega^1, \omega^2, \ldots, \omega^{n-1}]$
n roots of unity
Evaluate $P_e(x^2)$ and $P_o(x^2)$ at
$[1, \omega^2, \omega^4, \ldots, \omega^{2(n/2-1)}]$
$(n/2)$ roots of unity

FFT Pseudocode

```
def FFT(P):
    # P - [p_0, p_1, ..., p_{n-1}] coeff representation
    n = len(P) # n is a power of 2
    if n == 1:
        return P
    ω = e^(2πi/n)
    P_e, P_o = [p_0, p_2, ..., p_{n-2}], [p_1, p_3, ..., p_{n-1}]
    y_e, y_o = FFT(P_e), FFT(P_o)
    y = [0] * n
    for j in range(n/2):
        y[j] = y_e[j] + ω^j y_o[j]
        y[j + n/2] = y_e[j] - ω^j y_o[j]
    return y
```

Figure: FFT Pseudocode[10]

[10] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb_logo

IFFT Vandermonde Matrix

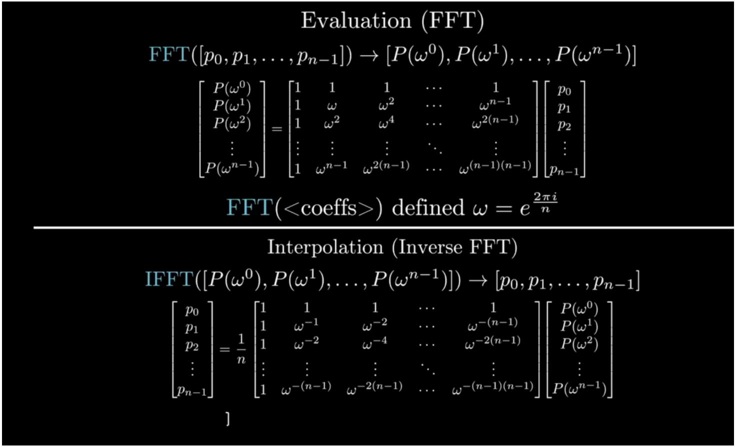


Figure: IFFT Vandermonde Matrix [11]

[11] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb$_{logo}$

IFFT Pseudocode

+ 关注

$$\text{IFFT}(<\text{values}>) \Leftrightarrow \text{FFT}(<\text{values}>) \text{ with } \omega = \frac{1}{n} e^{\frac{-2\pi i}{n}}$$

```
def FFT(P):
    # P - [p_0, p_1, ..., p_{n-1}] coeff rep
    n = len(P) # n is a power of 2
    if n == 1:
        return P
    ω = e^{2πi/n}
    P_e, P_o = P[::2], P[1::2]
    y_e, y_o = FFT(P_e), FFT(P_o)
    y = [0] * n
    for j in range(n/2):
        y[j] = y_e[j] + ω^j y_o[j]
        y[j + n/2] = y_e[j] - ω^j y_o[j]
    return y
```

```
def IFFT(P):
    # P - [P(ω^0), P(ω^1), ..., P(ω^{n-1})] value rep
    n = len(P) # n is a power of 2
    if n == 1:
        return P
    ω = (1/n) * e^{-2πi/n}
    P_e, P_o = P[::2], P[1::2]
    y_e, y_o = IFFT(P_e), IFFT(P_o)
    y = [0] * n
    for j in range(n/2):
        y[j] = y_e[j] + ω^j y_o[j]
        y[j + n/2] = y_e[j] - ω^j y_o[j]
    return y
```

Figure: FFT Pseudocode[12]

[12] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb

复数采样点的另一个优势

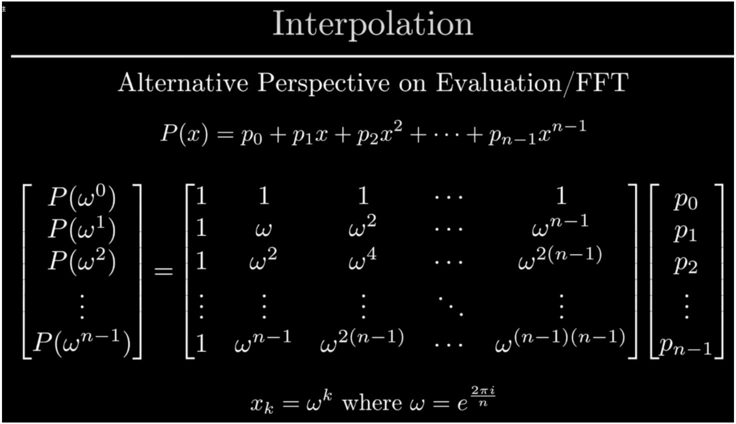


Figure: 另一个优势[13]

[13] https://www.youtube.com/watch?v=h7apO7q16V0feature=emb_logo

Thank you!