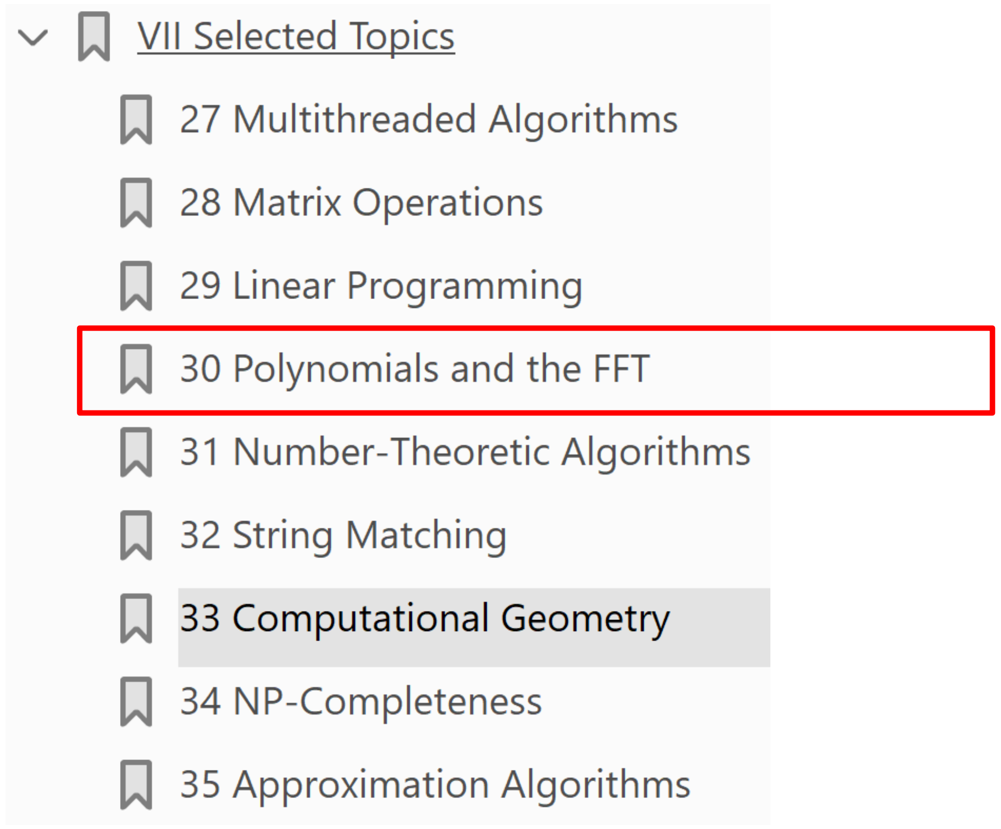
Chapter 30

Polynomials, Convolution and the FFT

songyou@buaa.edu.cn

VII Selected Topics



30 Polynomials, Convolution and the FFT

¥245.64
Princeton Lectures in Analysis/Stein 复分

¥69.20
傅里叶分析 傅里叶分析Fourier变换级数

¥80.40
傅里叶分析（英文版） 傅里叶分析（英文

¥76.90
快速傅里叶变换：算法与应用 从原理到案

¥41.70
小波与傅里叶分析基础（第二版） 小波与

¥218.00
傅里叶变换及其应用 （美）布雷斯韦尔

¥124.00
傅里叶变换 正版书籍，支持七天无理由，

¥114.00
快速傅里叶变换的计算框架 正版图书,有

¥25.00
离散时间信号处理与MATLAB仿真 离散时

¥47.00
傅里叶变换红外光谱分析第三版 【正版书

Chapter outline

- Introduction: Polynomials adding and multiplying, Convolution, Fourier Transform
- 30.1, presents two ways to represent polynomials:
    - coefficient representation
    - point-value representation
- Adding polynomials
- The straightforward methods for multiplying polynomials
    - $\Theta(n^2)$, polynomials are represented in coefficient form
    - $\Theta(n)$, when they are represented in point-value form
- Multiply polynomials using the coefficient representation in only $\Theta(n\lg n)$ time?
- FFT and its inverse
- 30.2, DFT and FFT
- 30.3, how to implement the FFT quickly

30 Polynomials, Convolution and the FFT

$$A(x) = a_0x^0 + a_1x^1 + \ldots + a_{n-1}x^{n-1} = \sum_{j=0}^{n-1} a_j x^j \quad \text{and} \quad B(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$C(x) = A(x) + B(x) \ ?$$

$$C(x) = A(x)*B(x) = A(x)\cdot B(x) = A(x)B(x) \ ?$$

- Straightforward method of adding two polynomials of degree n takes $\Theta(n)$ time
- Straightforward method of multiplying them takes $\Theta(n^2)$ time
- Fast Fourier Transform (FFT, 〔快速傅里叶变换算法〕, can reduce the time to multiply polynomials to $\Theta(n\lg n)$
- Why? How?

Polynomials

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \qquad x \in F, \qquad (R, C)$$

$$(a_0, a_1, \cdots, a_{n\text{-}1})$$

- polynomial *coefficients* : $a_0, a_1, \cdots, a_{n\text{-}1}, \in F$.
- polynomial *degree*〔维数〕: $A(x)$ is said to have degree k if its highest nonzero coefficient is a_k .
- polynomial *degree-bound*〔维界〕: any integer strictly greater than the degree.

- Therefore, the degree of a polynomial of degree-bound n may be any integer between 0 and n-1, inclusive.

Polynomials addition 〔多项式相加〕

- $A(x)$, $B(x)$, degree-bound n
- $C(x) = A(x) + B(x)$, also degree-bound n

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad \text{and} \quad B(x) = \sum_{j=0}^{n-1} b_j x^j$$

$$\text{then} \quad C(x) = \sum_{j=0}^{n-1} c_j x^j$$

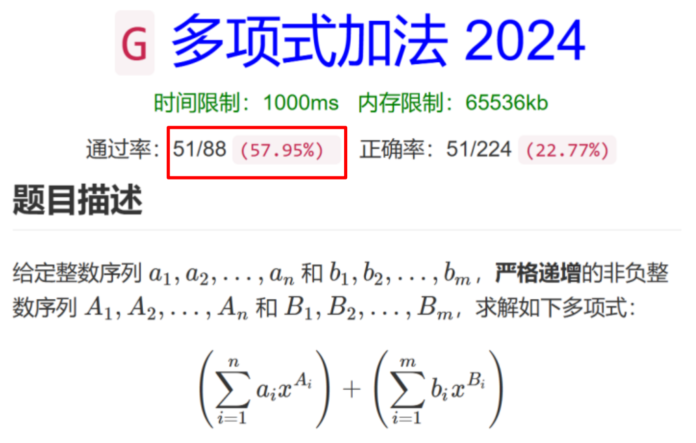
where $c_j = a_j + b_j$ for $j = 0, 1, \cdots, n - 1$.

For example, if the polynomials $A(x) = 6x^3+7x^2-10x+9$

and $B(x) = -2x^3+4x-5$,

then $C(x) = 4x^3+7x^2-6x+4$.

- Running time: $T(n) = \Theta(n)$



G 多项式加法 2024

时间限制：1000ms 内存限制：65536kb

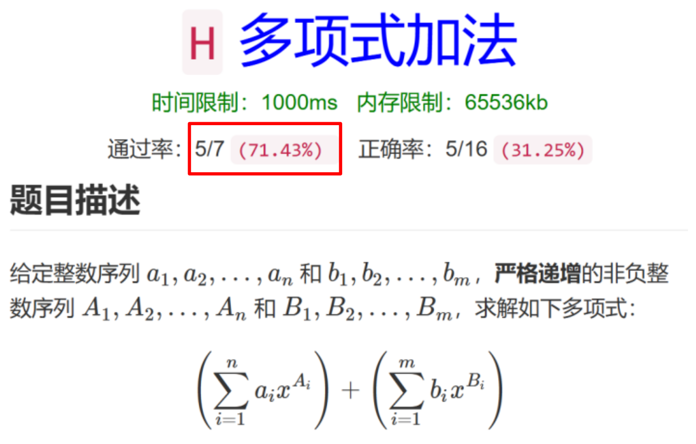
通过率：51/88 (57.95%) 正确率：51/224 (22.77%)

题目描述

给定整数序列 $a_1, a_2, \ldots, a_n$ 和 $b_1, b_2, \ldots, b_m$，**严格递增**的非负整数序列 $A_1, A_2, \ldots, A_n$ 和 $B_1, B_2, \ldots, B_m$，求解如下多项式：

$$\left(\sum_{i=1}^{n} a_i x^{A_i}\right) + \left(\sum_{i=1}^{m} b_i x^{B_i}\right)$$

2024秋程设-C6，1300+人



H 多项式加法

时间限制：1000ms 内存限制：65536kb

通过率：5/7 (71.43%) 正确率：5/16 (31.25%)

题目描述

给定整数序列 $a_1, a_2, \ldots, a_n$ 和 $b_1, b_2, \ldots, b_m$，**严格递增**的非负整数序列 $A_1, A_2, \ldots, A_n$ 和 $B_1, B_2, \ldots, B_m$，求解如下多项式：

$$\left(\sum_{i=1}^{n} a_i x^{A_i}\right) + \left(\sum_{i=1}^{m} b_i x^{B_i}\right)$$

2024秋航C-C6，1300+人

Polynomials multiplication（多项式相乘）

- $A(x)$, $B(x)$, degree-bound n
- $C(x) = A(x)*B(x)$,   degree-bound $2n$-1

$$= \sum_{j=0}^{2n-2} c_j x^j$$

For example, multiply $A(x) = 6x^3 + 7x^2 - 10x + 9$

and $B(x) = -2x^3 + 4x - 5$

$$\begin{array}{rrrrrrr}
 & & & 6x^3 & +\ 7x^2 & -\ 10x & +\ 9 \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 \\
\hline
 & & & -\ 30x^3 & -\ 35x^2 & +\ 50x & -\ 45 \\
 & & 24x^4 & +\ 28x^3 & -\ 40x^2 & +\ 36x & \\
-\ 12x^6 & -\ 14x^5 & +\ 20x^4 & -\ 18x^3 & & & \\
\hline
-\ 12x^6 & -\ 14x^5 & +\ 44x^4 & -\ 20x^3 & -\ 75x^2 & +\ 86x & -\ 45
\end{array}$$

- Running time: $T(n) = \Theta(n^2)$

Polynomials multiplication（多项式相乘）

- $A(x)$, $B(x)$, degree-bound n $\qquad A(x)=\sum_{j=0}^{n-1} a_j x^j$ and $B(x)=\sum_{j=0}^{n-1} b_j x^j$
- $C(x) = A(x)*B(x)$, degree-bound $2n$-1

Another way to express the product $C(x)$ is

$$C(x)=\sum_{j=0}^{2n-2} c_j x^j \qquad (30.1)\ \text{, where}\quad c_j=\sum_{k=0}^{j} a_k b_{j-k} \qquad (30.2)$$

Convolution
卷积

$$a \otimes b = (a_0, a_1, \cdots, a_{n\text{-}1}) \otimes (b_0, b_1, \cdots, b_{n\text{-}1}) = (c_0, c_1, \cdots, c_{2n\text{-}2})$$

where, $c_j = a_0 b_j + a_1 b_{j\text{-}1} + \ldots + a_k b_{j\text{-}k} + \ldots + a_j b_0$

Note that degree(C) = degree(A) + degree(B), implying

degree-bound(C) = degree-bound(A) + degree-bound(B) -1
$\leq$ degree-bound(A) + degree-bound(B).

We also say: degree-bound(C) = degree-bound(A)+degree-bound(B)

Polynomials multiplication & Convolution〔多项式相乘与卷积〕

$$A(x)=\sum_{j=0}^{n-1} a_j x^j \quad \text{and} \quad B(x)=\sum_{j=0}^{n-1} b_j x^j \quad \Rightarrow \quad C(x) = A(x)*B(x), \text{ degree-bound } 2n\text{-}1$$

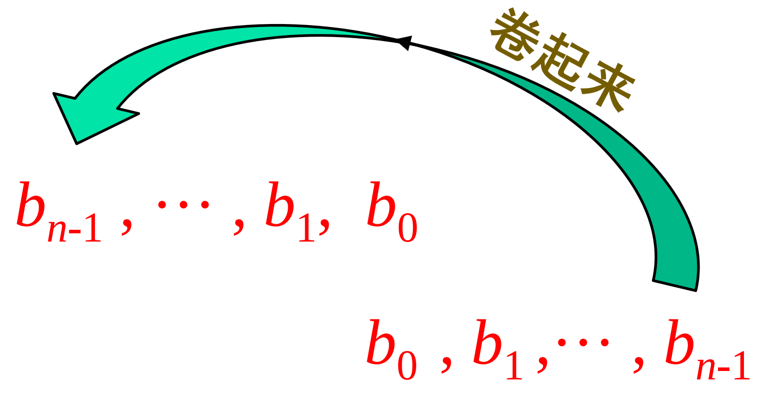
$$C(x)=\sum_{j=0}^{2n-2} c_j x^j \qquad (30.1) \qquad \text{, where} \quad c_j=\sum_{k=0}^{j} a_k b_{j-k} \qquad (30.2)$$

Convolution
卷积

$$a \otimes b = \quad (a_0\,, a_1\,,\cdots\,, a_{n\text{-}1}) \otimes (b_0\,, b_1\,,\cdots\,, b_{n\text{-}1}) = (c_0\,, c_1\,,\cdots\,, a_{2n\text{-}2})$$

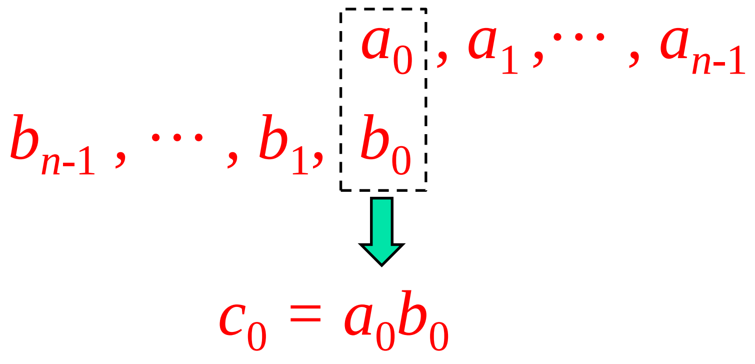
where, $c_j = a_0b_j + a_1b_{j\text{-}1} + \ldots + a_kb_{j\text{-}k} + \ldots + a_jb_0$

卷积的物理意义 ： 逆向（卷起来）→ 加权和（平均）→ 滑动， （逆向加权，滑动平均）



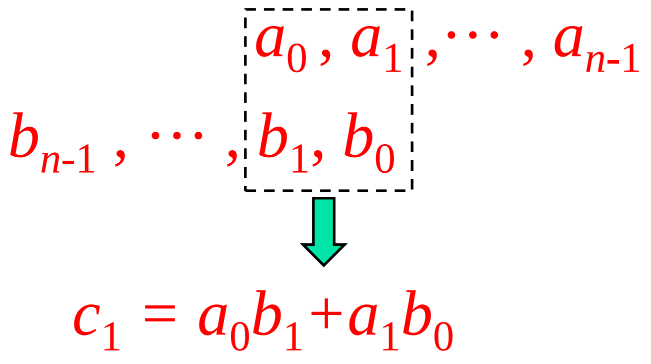
$b_{n\text{-}1}\,,\cdots\,, b_1,\; b_0$

$b_0\,, b_1\,,\cdots\,, b_{n\text{-}1}$



$$\begin{array}{lll} & \boxed{a_0} & , a_1\,,\cdots\,, a_{n\text{-}1} \\ b_{n\text{-}1}\,,\cdots\,, b_1, & \boxed{b_0} & \end{array}$$

$c_0 = a_0b_0$



$$\begin{array}{lll} & \boxed{a_0\,, a_1} & ,\cdots\,, a_{n\text{-}1} \\ b_{n\text{-}1}\,,\cdots\,, & \boxed{b_1, b_0} & \end{array}$$

$c_1 = a_0b_1 + a_1b_0$

Polynomials multiplication & Convolution 〔多项式相乘与卷积〕

- $A(x)$, $B(x)$, degree-bound n $\qquad A(x)=\sum_{j=0}^{n-1} a_j x^j$ and $B(x)=\sum_{j=0}^{n-1} b_j x^j$

- $C(x) = A(x)*B(x)$, degree-bound $2n$-1
  Another way to express the product $C(x)$ is

$$C(x)=\sum_{j=0}^{2n-2} c_j x^j \qquad (30.1) \quad \text{, where} \quad c_j=\sum_{k=0}^{j} a_k b_{j-k} \qquad (30.2)$$

Convolution
卷积

$$a \otimes b = (a_0, a_1, \cdots, a_{n\text{-}1}) \otimes (b_0, b_1, \cdots, b_{n\text{-}1}) = (c_0, c_1, \cdots, a_{2n\text{-}2})$$

where, $c_j = a_0 b_j + a_1 b_{j\text{-}1} + \ldots + a_k b_{j\text{-}k} + \ldots + a_j b_0$

- Running time: $T(n) = \Theta(n^2)$, since each coefficient in A must be multiplied by each coefficient in B.
- Can we accelerate the computation? Yes!
- Using Fast Fourier Transform (FFT,〔快速傅里叶变换算法〕, can reduce the time to multiply polynomials to $\Theta(n\lg n)$.

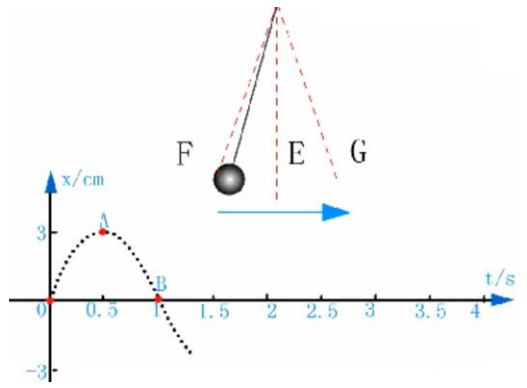
Fourier Transform (FT) and FFT

FT: the most common use is in signal processing

- *s*(*t*): *time domain* 〔信号的时域表示〕
- *S*(*ω*): *frequency domain* 〔频域〕

$$S(\omega)=\int e^{-i\omega t}s(t)dt$$

$$S(f)=\int e^{-i2\pi f\cdot t}s(t)dt$$



$$s(t)=\sin(\omega t)=\sin(2\pi\cdot f\cdot t)=\sin(60\pi\cdot t)$$

$$\omega=2\pi f,\ \ f=\frac{1}{T}=\frac{\omega}{2\pi}=\frac{60\pi}{2\pi}=30$$

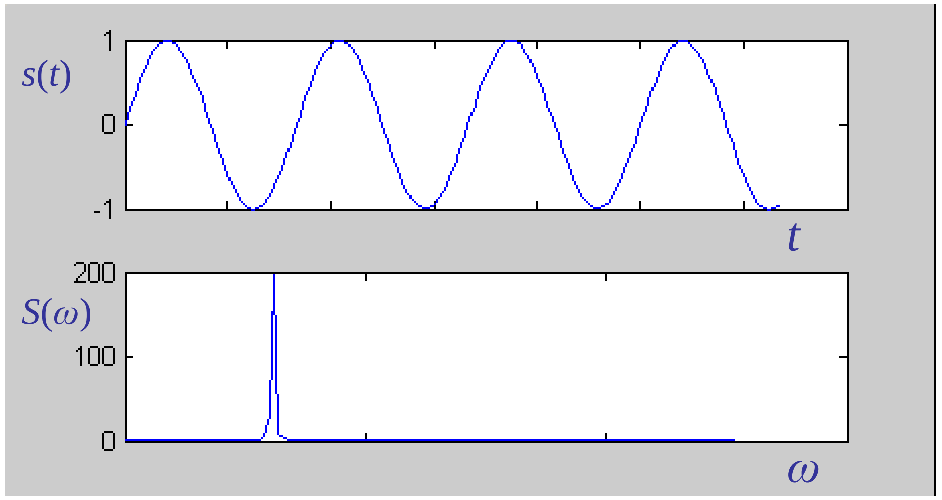
t = 1/30 s 就达到一个周期（重复）

f: 频率，每秒转多少圈（重复）（如：1秒转30圈）
T: 周期，转一圈所需的时间（转1圈花1/30秒）
ω: 角频率，每秒转多少（弧）度（1秒转2π*30弧度）

$$s(t)=\sin(60\pi t)$$

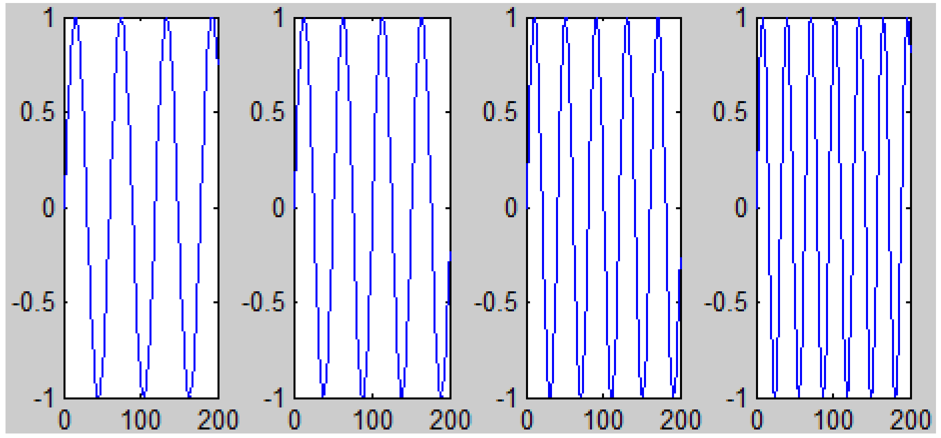
$$s(t)=\sin(60\pi t)+\sin(30\pi t)?$$

$$s(t)=\sin(60\pi t)+\sin(30\pi t)+\sin(90\pi t)?$$

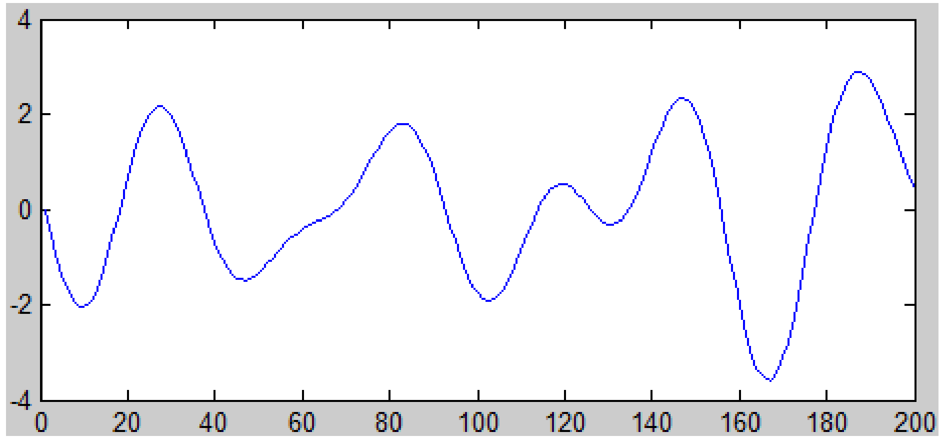


Fourier Transform (FT) and FFT

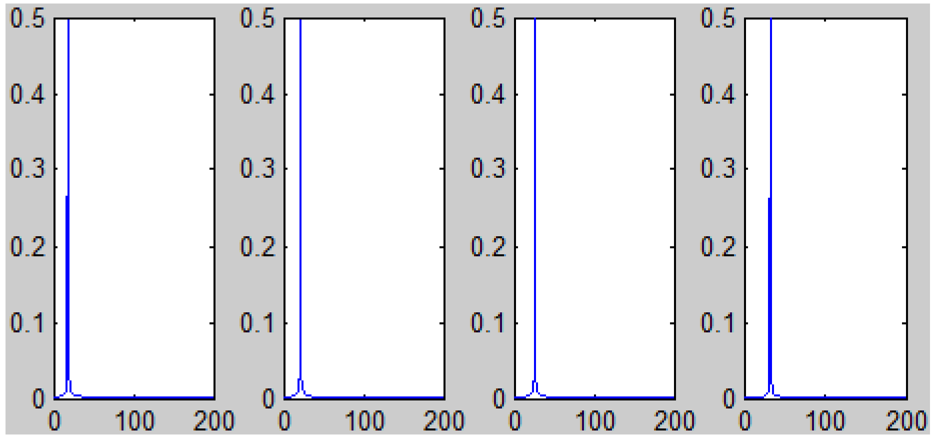
$s_i(t) = \sin(w_i t)$



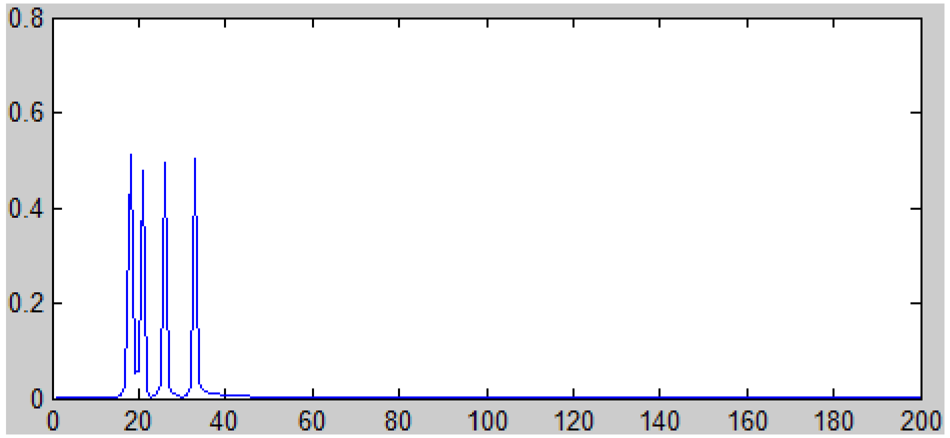
$s(t) = \sum a_i \cdot s_i(t)$



$S_i(\omega) = F(s_i(t))$



$S(\omega) = F(s(t))$



Fourier Transform (FT) and FFT

FT: the most common use is in signal processing

计算科学中最重要的32个算法

- $s(t)$: *time domain* 〔信号的时域表示〕
- $S(\omega)$: *frequency domain* 〔频域〕

$$s(t) = \sin(\omega t) = \sin(2\pi \cdot f \cdot t) = \sin(60\pi \cdot t)$$

$$\omega = 2\pi f, \quad f = \frac{1}{T} = \frac{\omega}{2\pi} = \frac{60\pi}{2\pi} = 30$$

f: 频率，每秒转多少圈（重复）（如：1秒转30圈）
T: 周期，转一圈所需的时间（转1圈花1/30秒）
ω: 角频率，每秒转多少（弧）度（1秒转2π*30弧度）

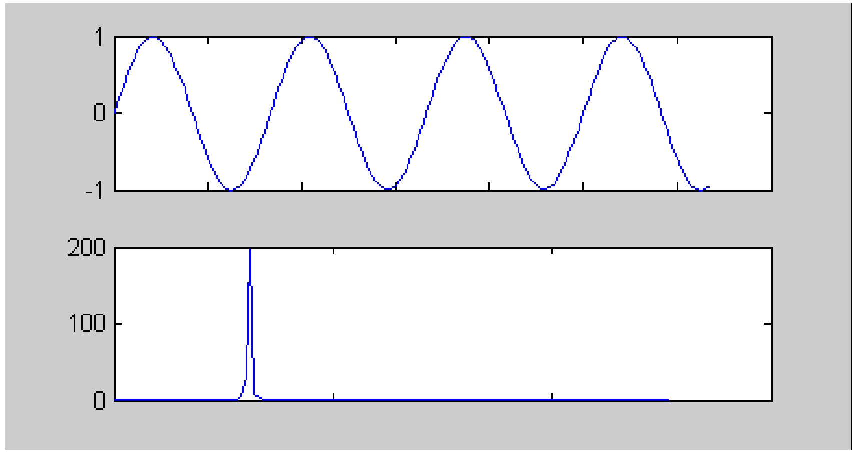
$$S(\omega) = \int e^{-i\omega t} s(t) dt$$

$$S(f) = \int e^{-i2\pi f \cdot t} s(t) dt$$

$$s(t) = \sin(60\pi t)$$

$$s(t) = \sin(60\pi t) + \sin(30\pi t)?$$

$$s(t) = \sin(60\pi t) + \sin(30\pi t) + \sin(90\pi t)?$$



Fourier Transform

让·巴普蒂斯·约瑟夫·傅里叶（Baron Jean Baptiste Joseph Fourier，1768-1830），男爵，法国数学家、物理学家，1768年3月21日生于欧塞尔，1830年5月16日卒于巴黎。

1817年当选为科学院院士。主要贡献是在研究《热的传播》和《热的分析理论》时创立了一套数学理论，对19世纪的数学和物理学的发展都产生了深远影响。

$s(t) = \sin(60\pi t) + \sin(30\pi t)$？

$s(t) = \sin(60\pi t) + \sin(30\pi t) + \sin(90\pi t)$？

$$S(\omega) = \int \mathrm{e}^{-i\omega t} s(t)\mathrm{d}t$$

- FT的基本思想首先由傅里叶提出，所以以其名字来命名以示纪念。傅里叶变换是一种特殊的积分变换。它能将满足一定条件的某个函数表示成正弦基函数的线性组合或者积分。在不同的研究领域，傅里叶变换具有多种不同的变体形式，如连续傅里叶变换和离散傅里叶变换。
- 数学上看，用简单表示来对复杂函数的深入研究。正弦函数在物理上是被充分研究而相对简单的函数类，这一想法跟化学上的原子论想法何其相似！奇妙的是，现代数学发现傅里叶变换具有非常好的性质，使得它如此的好用和有用，让人不得不感叹造物的神奇。
- 哲学上看，"分析主义"和"还原主义"，就是要通过对事物内部适当的分析达到增进对其本质理解的目的。比如近代原子论试图把世界上所有物质的本源分析为原子，而原子不过数百种而已，相对物质世界的无限丰富，这种分析和分类无疑为认识事物的各种性质提供了很好的手段。

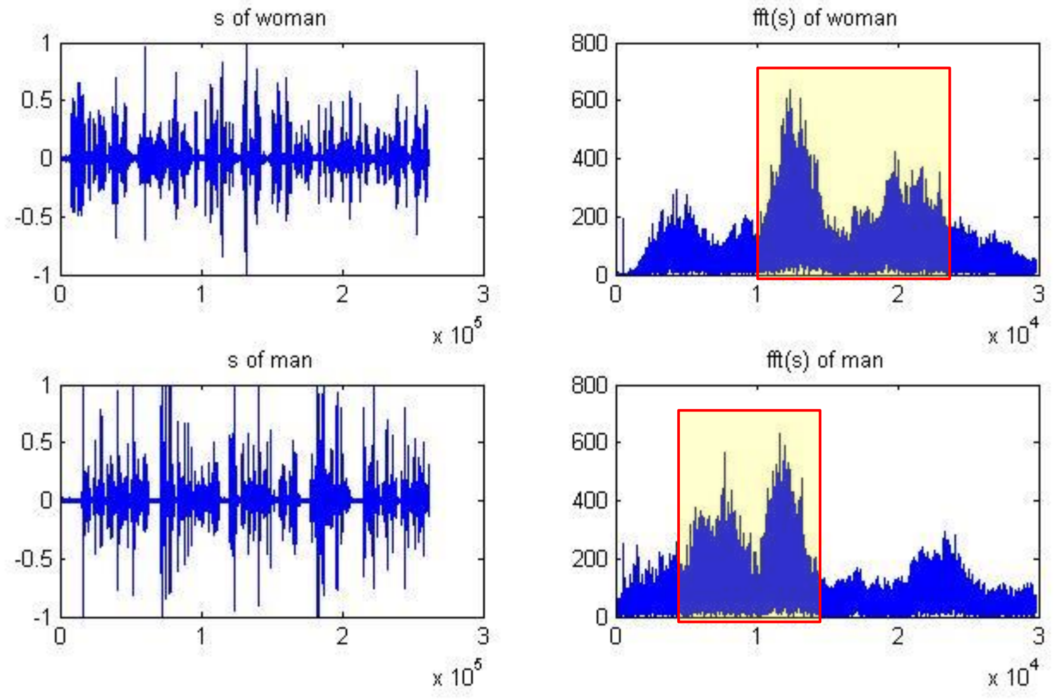
30 Polynomials, Convolution and the FFT

- FT: the most common use is in signal processing
  - *s*(*t*): *time domain* 〔信号的时域表示〕
  - *S*(*w*): *frequency domain* 〔频域〕

$$S(\omega) = \int e^{-i\omega t} s(t)\mathrm{d}t$$

$$S(f) = \int e^{-i2\pi f\cdot t} s(t)\mathrm{d}t$$

- An example of FFT (Fast Fourier Transform)



30.1 Representation of polynomials

- equivalence : The coefficient and point-value representations of polynomials are in a sense equivalent.

- That is, a polynomial in point-value form has a unique counterpart in coefficient form.

30.1.1 Coefficient representation

- A *coefficient representation* of a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$

  $a = (a_0, a_1, \cdots, a_{n-1})^T$, column vectors

秦九韶（1208年 - 1268年），汉族，出生于普州（今四川安岳县）。南宋著名数学家，与李冶、杨辉、朱世杰并称宋元数学四大家。

- Convenient for certain operations. For example,
  - *evaluating* $A(x)$ at a given point x_0 , takes time $\Theta(n)$ using *Horner's rule*:

    $A(x_0) = a_0+x_0(a_1+x_0(a_2+ \cdots +x_0(a_{n-2}+x_0(a_{n-1}))))$. 秦九韶算法

  - *adding* two polynomials represented $a = (a_0, a_1, \cdots, a_{n-1})^T$ and $b = (b_0, b_1, \cdots, b_{n-1})^T$, takes $\Theta(n)$ time: we just produce the coefficient vector $c = (c_0, c_1, \cdots, c_{n-1})$, where $c_j = a_j+b_j$ for $j = 0, 1, \cdots, n - 1$.

30.1.1 Coefficient representation

- multiplication of $A(x)$ and $B(x)$

$$A(x)=\sum_{j=0}^{n-1}a_j x^j,\quad a=(a_1,a_2,\cdots,a_{n-1})^T;\qquad B(x)=\sum_{j=0}^{n-1}b_j x^j,\quad b=(b_1,b_2,\cdots,b_{n-1})^T$$

$$C(x)=\sum_{j=0}^{2n-2}c_j x^j \qquad (30.1)$$

where $$c_j=\sum_{k=0}^{j}a_k b_{j-k} \qquad (30.2)$$

$$\begin{array}{rrrrrrr}
 & & & 6x^3 & +\ 7x^2 & -\ 10x & +\ 9 \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 \\
\hline
 & & & -\ 30x^3 & -\ 35x^2 & +\ 50x & -\ 45 \\
 & & 24x^4 & +\ 28x^3 & -\ 40x^2 & +\ 36x & \\
-\ 12x^6 & -\ 14x^5 & +\ 20x^4 & -\ 18x^3 & & & \\
\hline
-\ 12x^6 & -\ 14x^5 & +\ 44x^4 & -\ 20x^3 & -\ 75x^2 & +\ 86x & -\ 45
\end{array}$$

- takes time $\Theta(n^2)$, why?
- more difficult than adding two polynomials.
- vector c, given by equation (30.2), is also called the *convolution*(卷积) of a and b, denoted by $c = a \odot b$ ($a \otimes b$) .
- Multiplying polynomials and computing convolutions are fundamental computational problems, and important.

30.1.2 Point-value representation

- a *point-value representation* of a polynomial

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$\{(x_0, y_0), (x_1, y_1), \cdots, (x_{n\text{-}1}, y_{n\text{-}1})\}$, a set of n point-value pairs such that all of the x_k are distinct and

$$y_k = A(x_k),\ k = 0, 1, \cdots, n - 1 \qquad (30.3)$$

- any set of n distinct points $x_0, x_1, \cdots, x_{n\text{-}1}$ can be used as a basis for a polynomial representation, which is not unique. 〔多项式的点值表示不唯一〕

30.1.2 Point-value representation

- polynomial representation

  $a = (a_0, a_1, \cdots, a_{n-1})^T$, column vectors

  $$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

  $\{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$, a set of n point-value pairs

$$\begin{tikzcd}[column sep=huge]
\boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_k, y_k)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"]
\end{tikzcd}$$

- Evaluation
  - select n distinct points $x_0, x_1, \cdots, x_{n-1}$,
  - evaluate $A(x_k)$ for $k = 0, 1, \cdots, n - 1$.

  $$\Rightarrow \quad y_k = A(x_k) = \sum_{j=0}^{n-1} a_j (x_k)^j$$

  - the n-point evaluation takes time $\Theta(n^2)$. (Horner method)
  - if choose the x_k cleverly, the computation only needs $\Theta(n\lg n)$.

30.1.2 Point-value representation

- polynomial representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

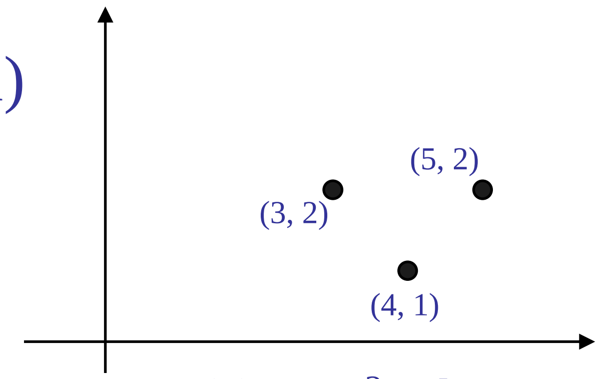
  $a = (a_0, a_1, \cdots, a_{n-1})^T$, column vectors

  $\{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$, a set of n point-value pairs

$$\begin{tikzcd} \boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"] \end{tikzcd}$$

- Interpolation〔插值〕, The inverse of evaluation
- *Theorem* 30.1 (Uniqueness of an interpolating polynomial)

  For any set $\{(x_k, y_k)\}$ of n point-value pairs, all the x_k values are distinct, there is a unique polynomial $A(x)$ of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \cdots, n - 1$.



$A(x) = ax^2 + bx + c$?

30.1.2 Point-value representation

- polynomial representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

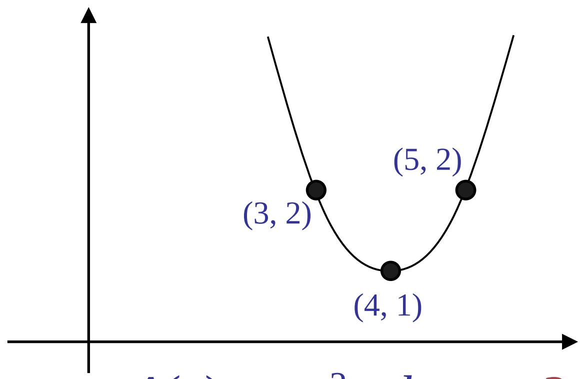
  $a = (a_0, a_1, \cdots, a_{n-1})^T$, column vectors

  $\{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$, a set of n point-value pairs

$$\begin{tikzcd} \boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"] \end{tikzcd}$$

- Interpolation〔插值〕, The inverse of evaluation
- *Theorem* 30.1 (Uniqueness of an interpolating polynomial)

  For any set $\{(x_k , y_k)\}$ of n point-value pairs, all the x_k values are distinct, there is a unique polynomial $A(x)$ of degree-bound n such that $y_k = A(x_k)$ for $k = 0, 1, \cdots, n - 1$.



$A(x) = ax^2 + bx + c$?

$\Rightarrow$ $a = 1, b = -8, c = 17$

30.1.2 Point-value representation

- polynomial representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"]
\end{tikzcd}$$

- *Theorem* 30.1 For any set $\{(x_k, y_k)\}$ of n point-value pairs, all the x_k values are distinct, there is a unique polynomial $A(x)$ of degree-bound n, s.t. $y_k = A(x_k)$ for $k = 0, 1, \cdots, n$-1.

  Proof Equation (30.3) "$y_k=A(x_k)$, $k = 0, 1, \cdots, n$-1" is equivalent to

$$\begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ & & \ldots\ldots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, \cdots, x_{n-1}) \cdot a \qquad (30.4)$$

V is Vander-monde matrix, has determinant $\prod_{0 \le j < k \le n-1} (x_k - x_j)$, therefore, V^{-1} exists. Thus,

$a = V(x_0, x_1, \cdots, x_{n-1})^{-1} \cdot y$. (chap28, LU decomposition, $O(n^3)$)

30.1.2 Point-value representation

- polynomial representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd} \boxed{\begin{array}{c}\text{coefficient form}\\ a=(a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"] \end{tikzcd}$$

- *Theorem* 30.1 For any set $\{(x_k, y_k)\}$ of n point-value pairs, all the x_k values are distinct, there is a unique polynomial $A(x)$ of degree-bound n, s.t. $y_k = A(x_k)$ for $k = 0, 1, \cdots, n\text{-}1$.

- *Lagrange's formula*: A faster algorithm for n-point interpolation

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}$$

Computing a of A using Lagrange’s formula takes time $\Theta(n^2)$.

30.1.2 Point-value representation

- polynomial representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"]
\end{tikzcd}$$

- quite convenient for many operations on polynomials,
  - *addition*, if $C(x) = A(x) + B(x)$, then $C(x_k) = A(x_k) + B(x_k)$ for any point x_k.
  - More precisely, if we have a point-value representation for $A : \{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$, and for $B : \{(x_0, z_0), (x_1, z_1), \cdots, (x_{n-1}, z_{n-1})\}$, then

    $$C : \{(x_0, y_0+z_0), (x_1, y_1+z_1), \cdots, (x_{n-1}, y_{n-1}+z_{n-1})\}$$
  - takes time $\Theta(n)$.

30.1.2 Point-value representation

- polynomial representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n\text{-}1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"]
\end{tikzcd}$$

- quite convenient for many operations on polynomials,
  - *multiplying*, if $C(x) = A(x)B(x)$, then $C(x_k) = A(x_k)B(x_k)$ for any point x_k, however
  - degree-bound(C) = degree-bound(A) + degree-bound(B)
  - multiplying A and B gives n point-value pairs for C,
  - but, degree-bound(C) = $2n$
  - need $2n$ point-value pairs for C
  - must "extended" point-value for A and B consisting of $2n$ point-value pairs each.

$$\begin{array}{rrrrrrr}
 & & & 6x^3 & +\ 7x^2 & -\ 10x & +\ 9 \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 \\
\hline
 & & & -\ 30x^3 & -\ 35x^2 & +\ 50x & -\ 45 \\
 & & 24x^4 & +\ 28x^3 & -\ 40x^2 & +\ 36x & \\
-\ 12x^6 & -\ 14x^5 & +\ 20x^4 & -\ 18x^3 & & & \\
\hline
-\ 12x^6 & -\ 14x^5 & +\ 44x^4 & -\ 20x^3 & -\ 75x^2 & +\ 86x & -\ 45
\end{array}$$

30.1.2 Point-value representation

- polynomial representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j\,, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"]
\end{tikzcd}$$

- quite convenient for many operations on polynomials,
  - *multiplying*, $C(x) = A(x)B(x) \Rightarrow C(x_k) = A(x_k)B(x_k)$ for any x_k
  - "extended" point-value for A and B
  - Given an extended point-value representation for A and B,

    $A : \{(x_0, y_0), (x_1, y_1), \cdots, (x_{2n-1}, y_{2n-1})\}$, and

    $B : \{(x_0, z_0), (x_1, z_1), \cdots, (x_{2n-1}, z_{2n-1})\}$, then

    $C : \{(x_0, y_0z_0), (x_1, y_1z_1), \cdots, (x_{2n-1}, y_{2n-1}z_{2n-1})\}$
  - takes time $\Theta(n)$, much less than the time required to multiply polynomials in coefficient form.

30.1.2 Point-value representation

- polynomial representation $$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd} \boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j , y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"] \end{tikzcd}$$

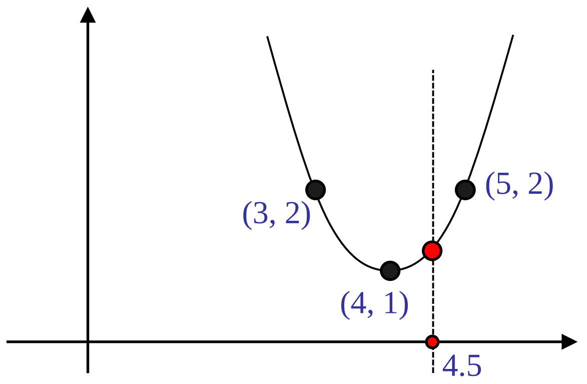
- How to evaluate a polynomial given in point-value form at a new point (How to extended point-value for A and B)?

  给定多项式的点值表示形式，如何求多项式在新点处的值？

- There is apparently no approach that is simpler than converting the polynomial to coefficient form first, and then evaluating it at the new point.

  先将多项式转换为系数表示形式，再求多项式在新点处的值。

$A(3) = 2$
$A(4) = 1$
$A(5) = 2$
$A(4.5) = ?$



$A(x) = ax^2 + bx + c$?

⇨ $a = 1$, $b = -8$, $c = 17$

⇨ $A(4.5) = 4.5^2 - 8*4.5 + 17 = 1.25$

30.1.3 Fast multiplication of polynomials

- polynomial representation $$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n\text{-}1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"]
\end{tikzcd}$$

- *Multiplying polynomial* $C(x) = A(x)B(x)$
  - point-value form, $\Theta(n)$, very quick!
  - coefficient form, $\Theta(n^2)$
- Can we use the linear-time multiplication method for polynomials in point-value form to expedite(加速) polynomial multiplication in coefficient form?

  对于多项式乘法，能否依据点值形式的线性乘法来改进系数形式的乘法？
- The answer hinges on our ability to convert a polynomial quickly from coefficient form to point-value and vice-versa.

  该问题依赖于将多项式从系数表示快速转换为点值表示的能力，或相反。

30.1.3 Fast multiplication of polynomials

Polynomial-multiplication problem

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{coefficient form}\\ a = (a_0, a_1, \cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\begin{array}{c}Time1\\ \text{evaluation}\end{array}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\begin{array}{c}\text{interpolation}\\ Time2\end{array}"]
\end{tikzcd}$$

n^2

time of polynomial-multiplication

n

time of polynomial-multiplication

If $\quad Time1 + Time2 + n < n^2$

30.1.3 Fast multiplication of polynomials

- polynomial representation $$A(x)=\sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd}[column sep=huge] \boxed{\begin{array}{c}\text{coefficient form}\\ a=(a_0, a_1,\cdots, a_{n-1})^T\end{array}} \arrow[r, shift left, "\begin{array}{c}\text{(DFT)}\\ \text{evaluation}\end{array}"] & \boxed{\begin{array}{c}\text{point-value form}\\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\begin{array}{c}\text{interpolation}\\ \text{(inverse DFT)}\end{array}"] \end{tikzcd}$$

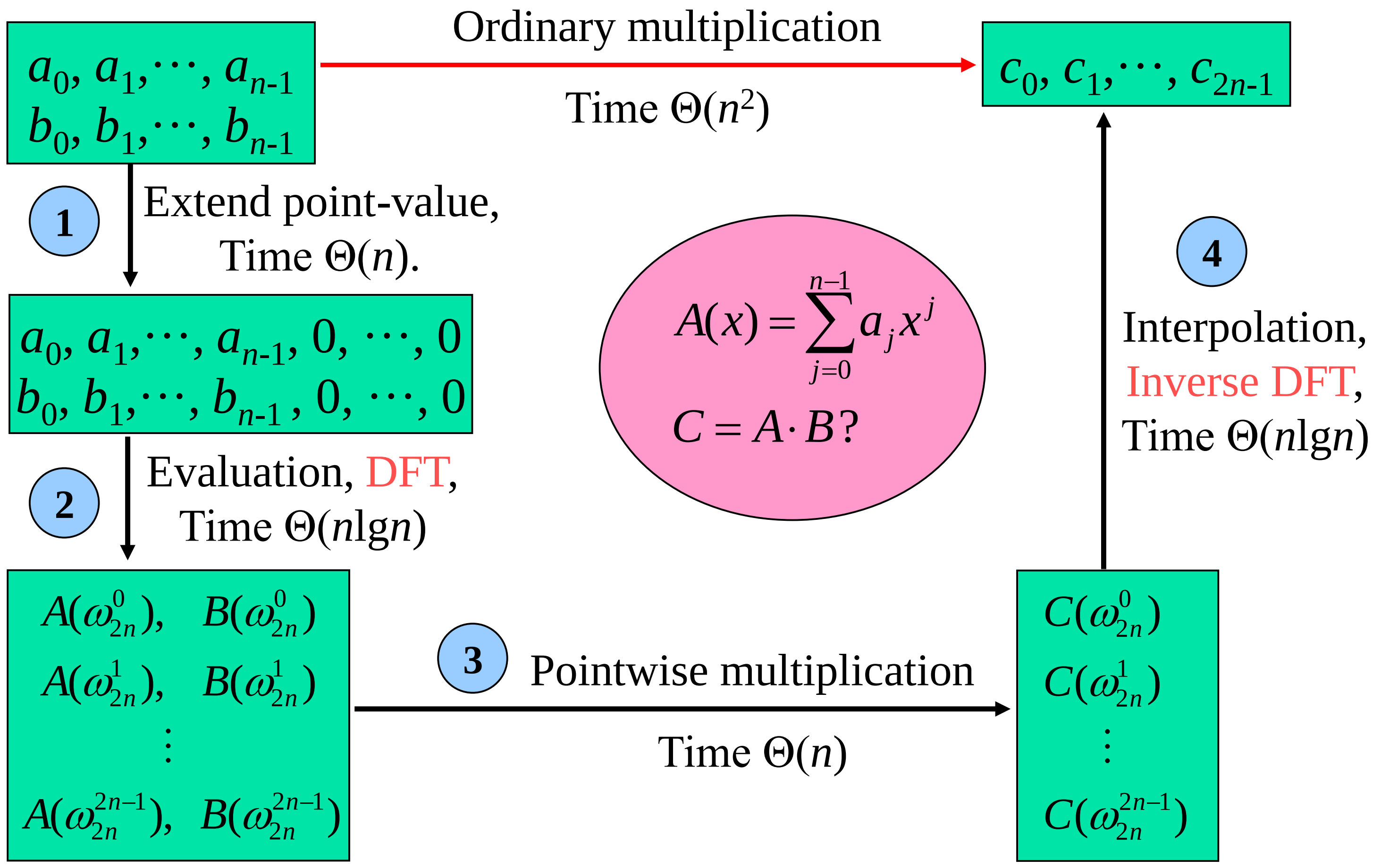
- We can use any points we want as evaluation points, but by choosing "**complex roots of unity**" as the evaluation points, we can convert between representations in only $\Theta(n\lg n)$ time.

  选单位复数根作为求值点，求值(evaluation)和插值(interpolation)能快速实现！

- 30.2, FFT performs the DFT and inverse DFT operations in $\Theta(n\lg n)$ time.

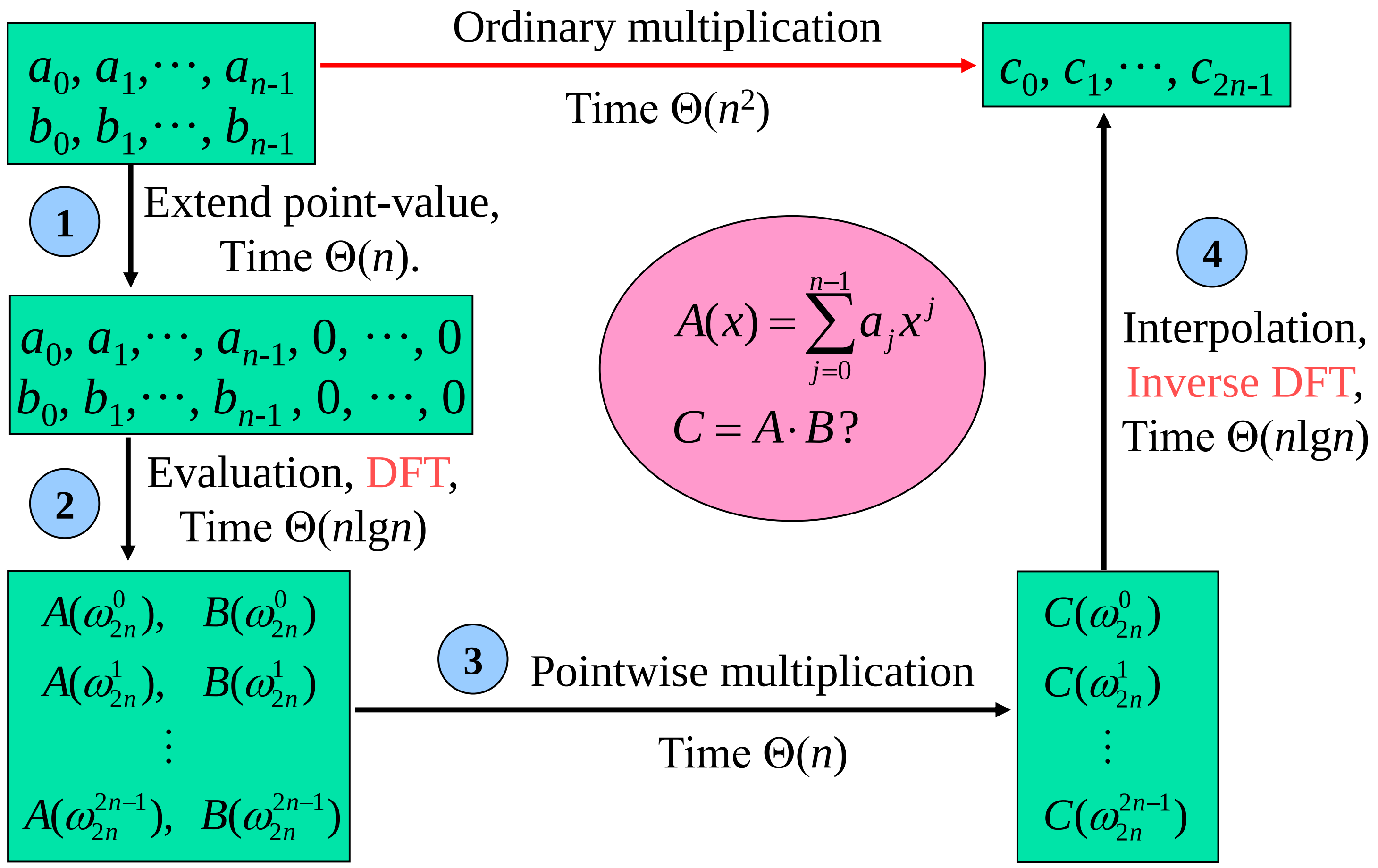
30.1.3 Fast multiplication of polynomials

A graphical outline of an efficient polynomial-multiplication process


$a_0, a_1, \cdots, a_{n-1}$
$b_0, b_1, \cdots, b_{n-1}$
Ordinary multiplication
Time $\Theta(n^2)$
$c_0, c_1, \cdots, c_{2n-1}$
1
Extend point-value,
Time $\Theta(n)$.
$a_0, a_1, \cdots, a_{n-1}, 0, \cdots, 0$
$b_0, b_1, \cdots, b_{n-1}, 0, \cdots, 0$
$A(x) = \sum_{j=0}^{n-1} a_j x^j$
$C = A \cdot B$?
4
Interpolation,
Inverse DFT,
Time $\Theta(n\lg n)$
2
Evaluation, DFT,
Time $\Theta(n\lg n)$
$A(\omega_{2n}^0), \quad B(\omega_{2n}^0)$
$A(\omega_{2n}^1), \quad B(\omega_{2n}^1)$
$\vdots$
$A(\omega_{2n}^{2n-1}), \quad B(\omega_{2n}^{2n-1})$
3
Pointwise multiplication
Time $\Theta(n)$
$C(\omega_{2n}^0)$
$C(\omega_{2n}^1)$
$\vdots$
$C(\omega_{2n}^{2n-1})$

已知多项式的系数表示，快速实现多项式相乘的计算框架

30.1.3 Fast multiplication of polynomials


$a_0, a_1, \cdots, a_{n-1}$
$b_0, b_1, \cdots, b_{n-1}$
Ordinary multiplication
Time $\Theta(n^2)$
$c_0, c_1, \cdots, c_{2n-1}$
1
Extend point-value,
Time $\Theta(n)$.
$a_0, a_1, \cdots, a_{n-1}, 0, \cdots, 0$
$b_0, b_1, \cdots, b_{n-1}, 0, \cdots, 0$
$A(x) = \sum_{j=0}^{n-1} a_j x^j$
$C = A \cdot B?$
4
Interpolation,
Inverse DFT,
Time $\Theta(n\lg n)$
2
Evaluation, DFT,
Time $\Theta(n\lg n)$
$A(\omega_{2n}^0), \quad B(\omega_{2n}^0)$
$A(\omega_{2n}^1), \quad B(\omega_{2n}^1)$
$\vdots$
$A(\omega_{2n}^{2n-1}), \quad B(\omega_{2n}^{2n-1})$
3
Pointwise multiplication
Time $\Theta(n)$
$C(\omega_{2n}^0)$
$C(\omega_{2n}^1)$
$\vdots$
$C(\omega_{2n}^{2n-1})$

***Theorem* 30.2** The product of two polynomials of degree-bound n can be computed in time $\Theta(n\lg n)$, with both the input and output representations in coefficient form.

30.1 Polynomial

- representation

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{coefficient form} \\ a = (a_0, a_1, \cdots, a_{n\text{-}1})^T\end{array}} \arrow[r, shift left, "\text{evaluation}"] & \boxed{\begin{array}{c}\text{point-value form} \\ \{(x_j, y_j)\}\end{array}} \arrow[l, shift left, "\text{interpolation}"]
\end{tikzcd}$$

- operations
  - Addition
  - ***Multiplying***
  - Evaluation
  - Interpolation
  - …

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad \text{and} \quad B(x) = \sum_{j=0}^{n-1} b_j x^j$$

then $C(x) = \sum_{j=0}^{n-1} c_j x^j$ $\qquad c_j = a_j + b_j$

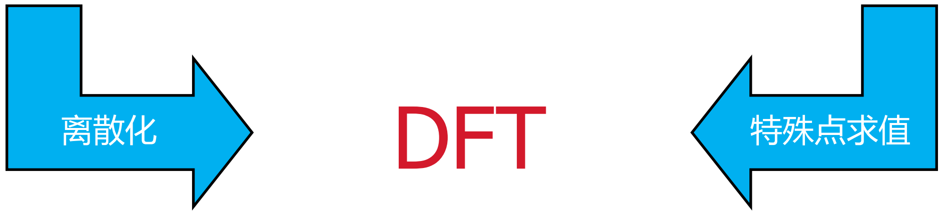
$$C(x) = \sum_{j=0}^{2n-2} c_j x^j \tag{30.1}$$

where $c_j = \sum_{k=0}^{j} a_k b_{j-k}$ (30.2)

$$\begin{array}{r}
6x^3 + 7x^2 - 10x + 9 \\
- 2x^3 + 4x - 5 \\
\hline
- 30x^3 - 35x^2 + 50x - 45 \\
24x^4 + 28x^3 - 40x^2 + 36x \\
- 12x^6 - 14x^5 + 20x^4 - 18x^3 \\
\hline
- 12x^6 - 14x^5 + 44x^4 - 20x^3 - 75x^2 + 86x - 45
\end{array}$$

30.2 The DFT and FFT

$$S(\omega) = \int e^{-i\omega t} s(t)\, dt \qquad\qquad A(x) = \sum_{j=0}^{n-1} a_j x^j$$



$$y_0 = A(x_0) = \sum_{j=0}^{n-1} a_j (x_0)^j$$

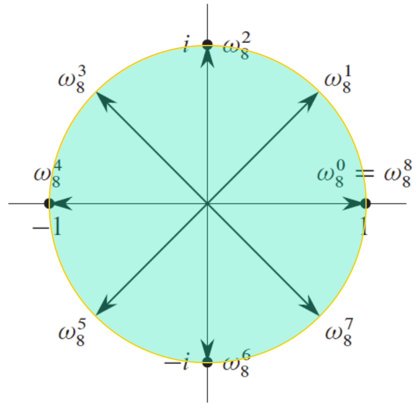
$$\vdots$$

$$y_k = A(x_k) = \sum_{j=0}^{n-1} a_j (x_k)^j$$

- DFT (Discrete Fourier Transform, 离散傅里叶变换)
- how the FFT (Fast Fourier Transform, 快速傅里叶变换) computes the DFT and its inverse in just $\Theta(n\lg n)$ time
- complex roots of unity and their properties 单位复数根及其性质

30.2.1 Complex roots of unity

- $\omega^n = 1$, complex number ω is a complex *n*th root of unity
  〔ω 称为 1 的 n 次复根，或单位 n 次复根，一般也表示为 $\omega_n = (\omega_n)^1$〕
- $\omega_n = e^{2\pi i/n}$: principal *n*th root of unity 〔单位 n 次复根的基元〕
- $$\omega_n^k = (\omega_n)^k = (e^{\frac{2\pi}{n}i})^k = e^{2\pi ik/n},\ k = 0,1,\cdots,n-1$$
  exactly *n* complex *n*th roots of unity 〔单位 1 有 n 个 n 次复根〕
- $e^{iu} = \cos(u) + i\sin(u)$:
  the exponential of a complex number
  $$\omega_n^k = e^{2\pi ik/n} = \cos(2\pi k/n) + i\sin(2\pi k/n),$$
  $$k = 0,1,\cdots,n-1$$
- All of the other complex *n*th roots of unity are powers of ω_n . The *n* complex *n*th roots of unity $\{\omega_n^k,\ k = 0,\cdots,n-1\}$ form a group.
  〔n 个单位 n 次复根组成一个群（伽罗瓦，法，1811~1832）〕

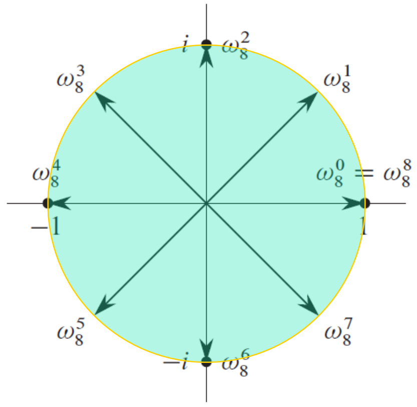


30.2.1 Complex roots of unity

- $\omega_n = e^{2\pi i/n}$: principal *n*th root of unity 〔单位 *n* 次复根的基元〕
- $e^{iu} = \cos(u) + i\sin(u)$ 〔欧拉公式〕
  the exponential of a complex number

$$\omega_n^k = e^{2\pi ik/n} = \cos(2\pi k/n) + i\sin(2\pi k/n),$$
$$k = 0, 1, \cdots, n-1$$

- All of the other complex *n*th roots of unity are powers of ω_n . The *n* complex *n*th roots of unity $\{\omega_n^k,\ k = 0, \cdots, n-1\}$ form a group.



群的快读
集合S以及定义在S上的运算，满足性质：
①封闭性，②单位元，③结合律，④逆元。
如，整数及加法构成一个群。
进一步定义，交换群，有限群等。

埃瓦里斯特•伽罗瓦（1811年10月25日-1832年5月31日），1811年10月25日生，法国数学家。现代数学中的分支学科群论的创立者。用群论彻底解决了根式求解代数方程的问题，而且由此发展了一整套关于群和域的理论，人们称之为伽罗瓦理论，并把其创造的“群”叫作伽罗瓦群（Galois Group）。在世时在数学上研究成果的重要意义没被人们所认识，曾呈送科学院3篇学术论文，均被退回或遗失。后转向政治，支持共和党，曾两次被捕。21岁时死于一次决斗。

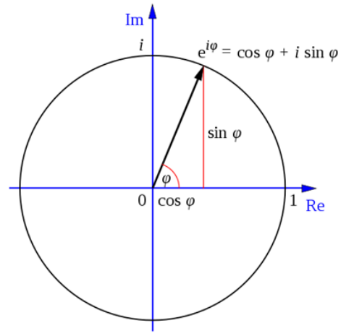
个人成就：使用群论的想法去讨论方程式的可解性，整套想法现称为伽罗瓦理论，是当代代数与数论的基本支柱之一。他系统化地阐释了为何五次以上之方程式没有公式解，而四次以下有公式解。他解决了古代三大尺规作图问题中的两个：“三等分任意角不可能”，“倍立方不可能”。

30.2.1 Complex roots of unity

最美丽的公式：欧拉公式

$$e^{iu} = \cos(u) + i\sin(u)$$

$$e^{i\pi} + 1 = 0$$



理由如下：

1. 最重要的自然常数的 e 含于其中。自然对数的底，大到飞船的速度，小至蜗牛的螺线，谁能够离开它?
2. 最重要的常数 π 含于其中。世界上最完美的平面对称图形是圆。“最伟大的公式”能够离开圆周率吗? (还有 π 和 e 是两个最重要的无理数！)
3. 最重要的运算符号 + 含于其中。加号最重要：因为其余符号都是由加号派生而来，减号是加法的逆运算，乘法是累计的加法……
4. 最重要的两个元在里面。零元 0，单位 1，是构造群，环，域的基本元素。如果你看了有关《近世代数》的书，你就会体会到它的重要性。
5. 最重要的虚单位 i 也在其中。虚单位 i 使数轴上的问题扩展到了平面，而在哈密尔的 4 元数与 凯莱的 8 元数中也离开不了它。
6. 这个公式的美在于其精简。她没有多余的字符，却联系着几乎所有的数学知识。有了加号，可以得到其余运算符号；有了 0，1，就可以得到其他的数字；有了 π 就有了圆函数，也就是三角函数；有了 i 就有了虚数，平面向量与其对应，也就有了哈密尔的 4 元数，现实的空间与其对应；有了 e 就有了微积分，就有了和工业革命时期相适宜的数学。

30.2.1 Complex roots of unity

some properties

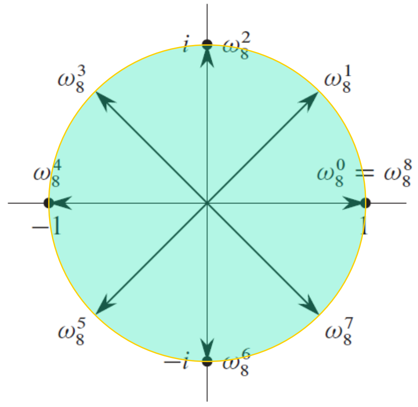
(1) $\omega_n^n = \omega_n^0 = 1$

($\omega_n^n = (e^{2\pi i/n})^n = \cos 2\pi + i \sin 2\pi$)

(2) $\omega_n^j \omega_n^k = \omega_n^{j+k} = \omega_n^{(j+k)\bmod n}$

(if $j+k = mn+r$, then

$\omega_n^{j+k} = \omega_n^{mn+r} = \omega_n^{mn}\omega_n^r = (\omega_n^n)^m \omega_n^r = \omega_n^r = \omega_n^{(j+k)\bmod n}$)



(3) $\omega_n^{-1} = \omega_n^{n-1}$

(proof by using $\omega_n^k = e^{2\pi ik/n}$, $k = 0, 1, \cdots, n-1$)

30.2.1 Complex roots of unity

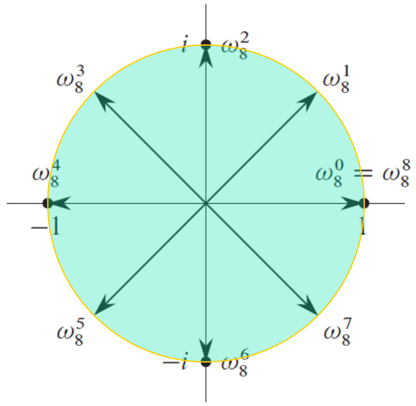
- *Lemma* 30.3 (Cancellation lemma, 可约（相消）引理)

  For any integers $n \geq 0$, $k \geq 0$, and $d > 0$,

$$\omega_{dn}^{dk} = \omega_n^k. \qquad (30.7)$$

Proof $\omega_{dn}^{dk} = (e^{2\pi i/dn})^{dk} = (e^{2\pi i/n})^k = \omega_n^k.$

$$(\ \omega_8^2 = \omega_4^1 \ , \quad \omega_8^4 = \omega_4^2 \ , \quad \omega_8^6 = \omega_4^3 \ , \quad \omega_8^8 = \omega_4^4 \ )$$



- *Corollary* 30.4 For any even integer $n > 0$,

$$\omega_n^{n/2} = \omega_2 = -1.$$

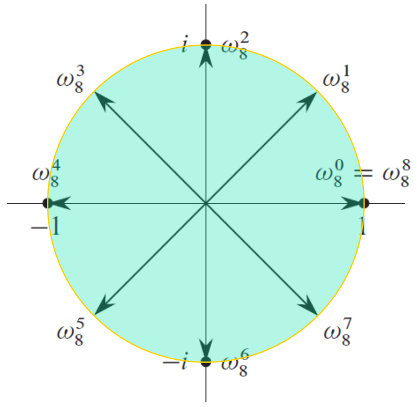
$$\omega_n^{n/2} = \omega_{(n/2)\cdot 2}^{n/2} = \omega_2 = e^{2\pi i/2} = \cos\pi + i\sin\pi = -1.$$

30.2.1 Complex roots of unity

Lemma 30.5 (Halving lemma, 等分引理)

If $n > 0$ is even, then the squares of the n complex nth roots of unity are the $n/2$ complex $(n/2)$th roots of unity.

〔n 个单位 n 次复方根的平方是 $n/2$ 个 $n/2$ 次复方根〕



Proof idea

$$(\omega_n^k)^2 = \omega_{2(n/2)}^{2k} = \omega_{n/2}^k \ , \ k = 0,1,\cdots,\frac{n}{2}-1$$

$$(\omega_n^{k+n/2})^2 = \omega_n^{2k+n} = \omega_n^{2k}\omega_n^n = \omega_n^{2k} = (\omega_n^k)^2 = \omega_{n/2}^k \ , \ k = 0,1,\cdots,\frac{n}{2}-1$$

$$\boxed{\begin{matrix} (\omega_n^0)^2 & ,(\omega_n^1)^2 & ,\cdots & ,(\omega_n^{n/2-1})^2 \\ (\omega_n^{0+n/2})^2 & ,(\omega_n^{1+n/2})^2 & ,\cdots & ,(\omega_n^{n/2-1+n/2})^2 \end{matrix}}$$

$$\omega_{n/2}^0 \quad ,\omega_{n/2}^1 \quad , \ \cdots \ ,\omega_{n/2}^{n/2-1}$$

$\boxed{-3, -2, -1, 1, 2, 3}^{\boxed{2}}$

$\boxed{9, 4, 1}$

30.2.1 Complex roots of unity

Lemma 30.6 (Summation lemma, 求和引理)

For any integer $n \geq 1$ and nonnegative integer k not divisible by n , ($k \neq m \cdot n$), we have

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = 0.$$

Proof

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1} = \frac{(\omega_n^n)^k - 1}{\omega_n^k - 1} = \frac{(1)^k - 1}{\omega_n^k - 1} = 0,\ k \neq mn.$$

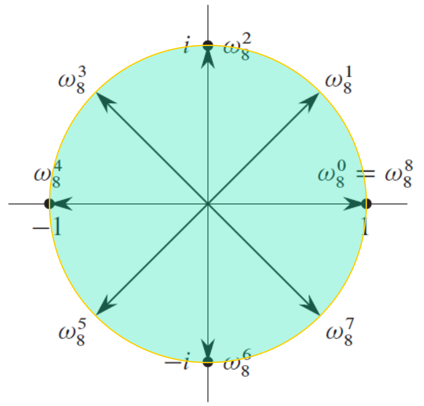
证明思想：利用实数的等幂级数求和性质。

30.2.2 The DFT

$$x_k = (\omega_n)^k = \omega_n^k = \left(e^{\frac{2\pi}{n}i}\right)^k = e^{\frac{2\pi k}{n}i} = \cos(2\pi k/n) + i\sin(2\pi k/n),$$

$$k = 0, 1, \cdots, n-1$$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \qquad \begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ & & \ldots\ldots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, \cdots, x_{n-1}) \cdot a \qquad (30.4)$$



- wish to evaluate a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ at $x = \omega_n^0, \omega_n^1, \omega_n^2, \cdots, \omega_n^{n-1}$
- without loss of generality, assume that $n = 2^m$, if not, let $a_{n+k} = 0$
- Discrete Fourier Transform (DFT): let A is given in coefficient form: $a = (a_0, a_1, \cdots, a_{n-1})^T$, let $x_k = \omega_n^k$, define y_k , for $k = 0, 1, \cdots , n-1$, by

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \ , \qquad \begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \ldots\ldots & & \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix}$$

y = DFT$_n$(a)： 数学意义上，多项式在特殊点的取值（单位复根）

Application of DFT

$$x_k = (\omega_n)^k = \omega_n^k = \left(e^{\frac{2\pi}{n}i}\right)^k = e^{\frac{2\pi k}{n}i} = \cos(2\pi k/n) + i\sin(2\pi k/n),$$

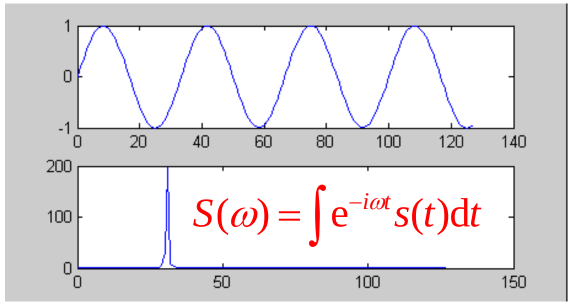
$$k = 0, 1, \cdots, n-1$$

- wish to evaluate a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ of degree-bound n at $x = \omega_n^0, \omega_n^1, \omega_n^2, \cdots, \omega_n^{n-1}$.
- Discrete Fourier Transform (DFT, 离散傅里叶变换)

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \ ,$$

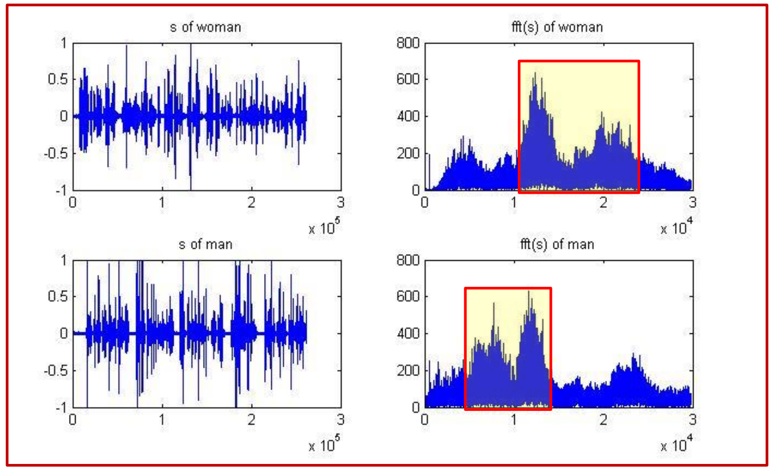
$$\begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ & & \ldots\ldots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \ldots\ldots & & \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix}$$

y = DFT$_n(a)$： 数学意义上，多项式在特殊点的取值（单位复根）



Signal $s(t)$: discrete, $a_i = s(t_i)$

Spcetrum $S(\omega)$: $S(\omega)$ = DFT$_n(s)$



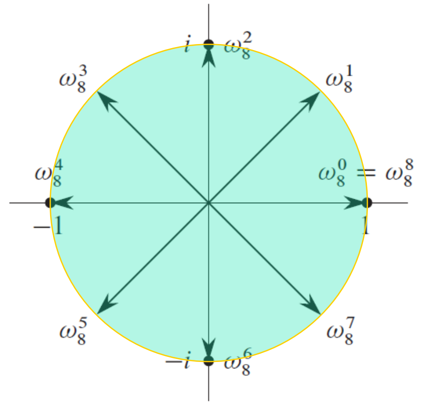
y = DFT$_n(a)$： 物理意义上，从一个矢量（时域）变换到另一个矢量（频域）

30.2.2 The DFT

$$x_k = (\omega_n)^k = \omega_n^k = \left(e^{\frac{2\pi}{n}i}\right)^k = e^{\frac{2\pi k}{n}i} = \cos(2\pi k/n) + i\sin(2\pi k/n),$$
$$k = 0, 1, \cdots, n-1$$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ & & \dots\dots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, \cdots, x_{n-1}) \cdot a \qquad (30.4)$$



- wish to evaluate a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ at $x = \omega_n^0, \omega_n^1, \omega_n^2, \cdots, \omega_n^{n-1}$
- without loss of generality, assume that $n = 2^m$, if not, let $a_{n+k} = 0$
- Discrete Fourier Transform (DFT): let A is given in coefficient form: $a = (a_0, a_1, \cdots, a_{n-1})^T$, let $x_k = \omega_n^k$, define y_k , for $k = 0, 1, \cdots, n-1$, by

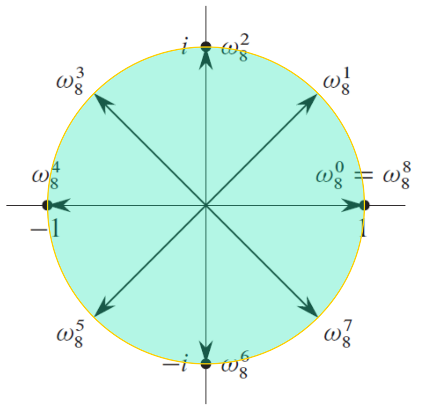
$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \ , \qquad \begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \dots\dots & & \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix}$$

$y = \mathbf{DFT}_n(a)$： 数学意义上，多项式在特殊点的取值（单位复根）

30.2.2 The DFT

$$x_k = (\omega_n^{(1)})^k = (\omega_n)^k = \omega_n^{(k)} = \omega_n^k = \left(e^{\frac{2\pi}{n}i}\right)^k = e^{\frac{2\pi k}{n}i} = \cos(2\pi k/n) + i\sin(2\pi k/n),$$
$$k = 0,1,\cdots,n-1$$

$$y = A(x) = \sum_{j=0}^{n-1} a_j x^j \Rightarrow \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ & & \ldots\ldots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, x_2, \cdots, x_{n-1}) \cdot a$$



$$y_k = A(x_k) = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} ,$$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^{(2)\cdot 1} & \omega_n^{(2)\cdot 2} & \cdots & \omega_n^{(2)\cdot(n-1)} \\ & & \cdots & & \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \cdots \\ a_{n-1} \end{pmatrix}$$

```
// DFT_real_input.c
void testDFT(struct Complex *y, struct Complex *a, int n)
{
    if(n == 1)
    {
        (*y).real = (*a).real;
        (*y).imag = (*a).imag;
        return;
    }
    struct Complex wn = {cos(2*pi/n), sin(2*pi/n)};
    struct Complex w = {1.0, 0.0}, wnk;

    for(int k=0; k < n; k++)
    {
        wnk = w;
        y[k] = ComAdd(a[0], ComMulti(wnk, a[1])); //y[k]的前两项
        for(int j=2; j < n; j++)
        {
            wnk = ComMulti(wnk, w);
            y[k] = ComAdd(y[k], ComMulti(wnk, a[j]));
        }
        w = ComMulti(w, wn);
    }
}
```

一个实例:

$$y = A(x) = 1 + 3x$$

$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & \omega_2^1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \end{pmatrix} \Rightarrow \begin{array}{l} y_0 = A(x_0) = A(1) = 1 + 3 * 1 = 4 \\ y_1 = A(x_1) = A(-1) = 1 + 3 * (-1) = -2 \end{array}$$

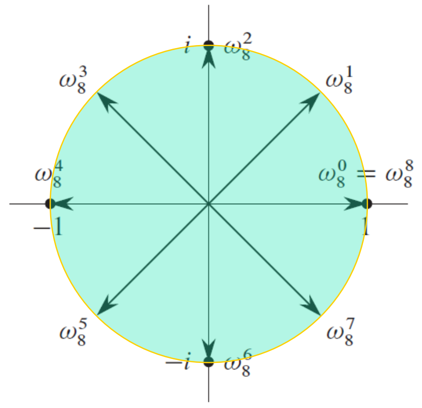
y = DFT$_n(a)$：数学意义上，多项式在特殊点的取值（单位复根）

30.2.2 The DFT

$$x_k = (\omega_n)^k = \omega_n^k = \left(e^{\frac{2\pi}{n}i}\right)^k = e^{\frac{2\pi k}{n}i} = \cos(2\pi k/n) + i\sin(2\pi k/n),$$
$$k = 0, 1, \cdots, n-1$$

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$\begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ & & \ldots\ldots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, \cdots, x_{n-1}) \cdot a \qquad (30.4)$$



- wish to evaluate a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ at $x = \omega_n^0, \omega_n^1, \omega_n^2, \cdots, \omega_n^{n-1}$
- without loss of generality, assume that $n = 2^m$, if not, let $a_{n+k} = 0$
- Discrete Fourier Transform (DFT): let A is given in coefficient form: $a = (a_0, a_1, \cdots, a_{n-1})^T$, let $x_k = \omega_n^k$, define y_k , for $k = 0, 1, \cdots , n-1$, by

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \ , \qquad \begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \ldots\ldots & & \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix}$$

y = DFT$_n$(a)： 数学意义上，多项式在特殊点的取值（单位复根）

Take time $\Theta(n^2)$ to compute straightforward? 慢！如何快速计算？

Exercises

30.1-2

求多项式相除的商多项式和余项。

$$A(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} = \sum_{j=0}^{n-1} a_j x^j = Q(x)(x - x_0) + r$$

输入：多项式 A 的系数 $(a_0, a_1, \ldots, a_{n\text{-}1})$，定点 x_0

输出：求满足如上条件的商多项式 $Q(x) = (q_0, q_1, \ldots, q_{n\text{-}2})$，余项 r

Exercises

对下列输入向量 $\boldsymbol{a}$，求 $\boldsymbol{y} = \mathrm{DFT}(\boldsymbol{a})$

$\boldsymbol{a} = (0, 1)$

$\boldsymbol{a} = (1, 0)$

$\boldsymbol{a} = (1, 1, 0, 1)$

$\boldsymbol{a} = (0, 1, 2, 3)$

$$\begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ & & \dots\dots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \cdots\cdots & & \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix}$$