数值分析：多项式插值

张王优

2025-10-11

SCHOOL OF
ARTIFICIAL INTELLIGENCE
上海交通大学人工智能学院

目录

CONTENTS

十进制与二进制表示

- 十进制到二进制的转换

十进制 $$\sum_{i=-\infty}^{+\infty} b_i \cdot 2^i = \cdots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} + 2^{-2} + \cdots$$

二进制 $$\cdots\ b_2\ b_1\ b_0\ .\ b_{-1}\ b_{-2}\ \cdots$$

- 转换规则
  - ❑ 整数部分：连续除以 2 记录余项。从小数点向左记录来记录余项 0 或 1
  - ❑ 小数部分：连续乘以 2 记录余项。从小数点向右记录来记录余项 0 或 1

十进制与二进制表示

例

将十进制数 98.3 转换为相应的二进制数：

$$98 \xrightarrow{\div 2} 49余0 \xrightarrow{\div 2} 24余1 \xrightarrow{\div 2} 12余0 \xrightarrow{\div 2} 6余0 \xrightarrow{\div 2} 3余0 \xrightarrow{\div 2} 1余1 \xrightarrow{\div 2} 0余1$$

$$0.3 \xrightarrow{\times 2} 0.6余0 \xrightarrow{\times 2} \underline{0.2余1 \xrightarrow{\times 2} 0.4余0 \xrightarrow{\times 2} 0.8余0 \xrightarrow{\times 2} 0.6余1} \xrightarrow{\times 2} \underline{0.2余1 \xrightarrow{\times 2} \cdots\cdots}$$

故 $(98.3)_{10} \rightarrow (1100010.0\ 1001\ 1001\ 1001\cdots)_2 = (1100010.0\overline{1001})_2$

- 转换规则
  - 整数部分：连续除以 2 记录余项。从小数点向左记录来记录余项 0 或 1
  - 小数部分：连续乘以 2 记录余项。从小数点向右记录来记录余项 0 或 1

十进制与二进制表示

- 二进制到十进制的转换

十进制 $\sum_{i=-\infty}^{+\infty} b_i \cdot 2^i = \cdots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} + 2^{-2} + \cdots$

二进制 $\cdots\ b_2\ b_1\ b_0\ .\ b_{-1}\ b_{-2}\ \cdots$

例

将以下二进制数转换为相应的十进制数：

$(10101)_2 =$ $1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^4 = (21)_{10}$

$(.1011)_2 =$ $1 \cdot 2^{-1} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} = \left(\frac{11}{16}\right)_{10}$

十进制与二进制表示

- 二进制到十进制的转换

例

将二进制数 $(.\overline{1011})_2$ 转换为十进制

令 $x = (.\overline{1011})_2$

$2^4 \cdot x = (1011.\overline{1011})_2$

$2^4 \cdot x - x = (1011)_2$

$15 \cdot x = 11$

$x = \left(\frac{11}{15}\right)_{10}$

例

将二进制数 $(.10\overline{101})_2$ 转换为十进制

令 $x = (.10\overline{101})_2$

$2^2 \cdot x = (10.\overline{101})_2$

$2^5 \cdot x = (10101.\overline{101})_2$

$2^5 \cdot x - 2^2 \cdot x = (10101)_2 - (10)_2$

$28 \cdot x = 19$

$x = \left(\frac{19}{28}\right)_{10}$

整数在计算机中的二进制表示

C++整数类型	比特位 n
short	≥16
int	≥16
long	≥32
long long	≥64

- 无符号整数（n-bit）
  - 所有位均用于表示数值
  - 取值范围：$0 \sim 2^n-1$
- 有符号整数（n-bit）
  - 最高位表示符号位（±），其余位表示数值
  - 取值范围：$-2^{n-1} \sim 2^{n-1}-1$
- Python3 中的整数类型 *int*
  - 取值范围：任意大！（只要内存放得下）
  - 拓展阅读：https://rushter.com/blog/python-integer-implementation/

实数在计算机中的浮点表示（Floating-point）

- Python 中的浮点数
  - 默认类型 *float*：双精度（FP64）
  - 所见非所得

15 个 0

```
>>> print(0.1, 0.10000000000000001)
输出: 0.1      0.1
```

16 个 0

```
>>> print(0.1000000000000000000555 == 0.1)
输出: True
```

  - 拓展阅读：https://docs.python.org/zh-cn/3.13/tutorial/floatingpoint.html

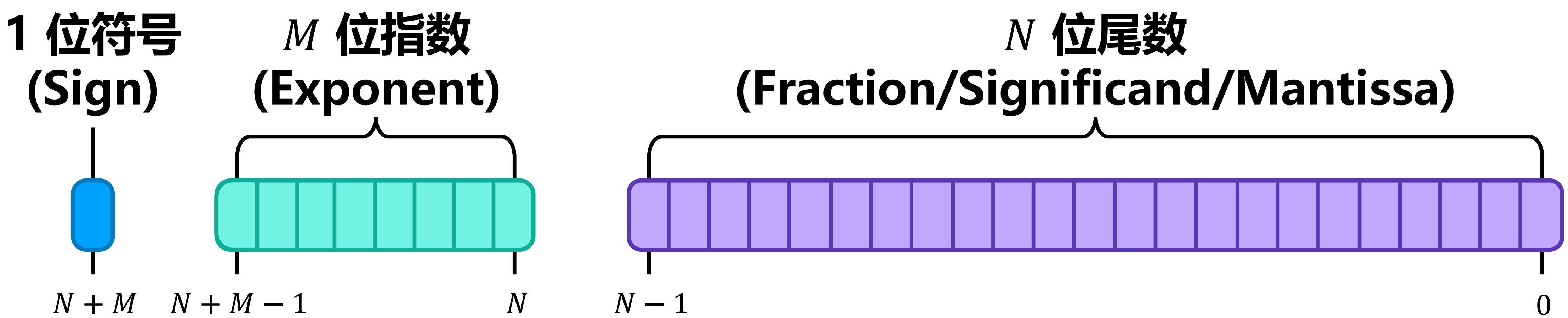
实数在计算机中的浮点表示（Floating-point)

- Python 中的浮点数

```
>>> a = 0.1 + 0.1 + 0.1
>>> print(a == 0.3)
输出: False

>>> import math
>>> print(
...     math.isclose(a, 0.3, rel_tol=0, abs_tol=0.5e-15),
...     math.isclose(a, 0.3, rel_tol=0, abs_tol=0.5e-16),
... )
输出: True     False
```

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 正规化的浮点数表示：$\pm 1.b_1 b_2 b_3 b_4 \cdots b_N \times 2^p$
  - 其中尾数 $b_2, b_3, b_4 \cdots b_N \in \{0,1\}$；　指数项 p 是有符号整数

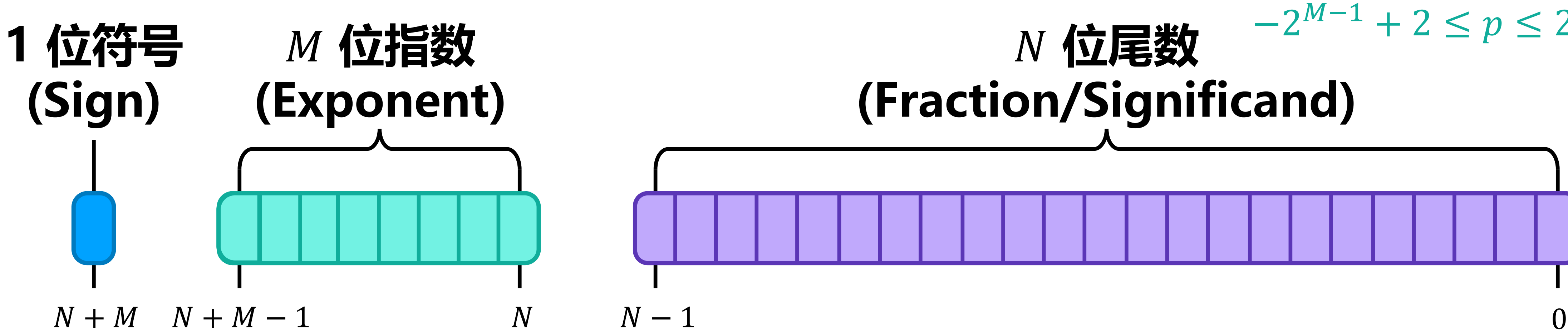


精度	符号	指数	尾数	可表示的最大/最小数值	机器精度	有效数字
半精度 (16 bit, FP16)	1 bit	5 bit	10 bit	$\pm(2-2^{-10})\cdot 2^{15}=\pm 65504$	2^{-10}	约3~4
单精度 (32 bit, FP32)	1 bit	8 bit	23 bit	$\pm(2-2^{-23})\cdot 2^{127}$	2^{-23}	约6~9
双精度 (64 bit, FP64)	1 bit	11 bit	52 bit	$\pm(2-2^{-52})\cdot 2^{1023}$	2^{-52}	约15~17

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 正规化的浮点数表示：$\pm 1.b_1b_2b_3b_4\cdots b_N \times 2^p$
  - 其中尾数 $b_2, b_3, b_4 \cdots b_N \in \{0,1\}$；　　指数项 p 是有符号整数

$-2^{M-1}+2 \le p \le 2^{M-1}-1$



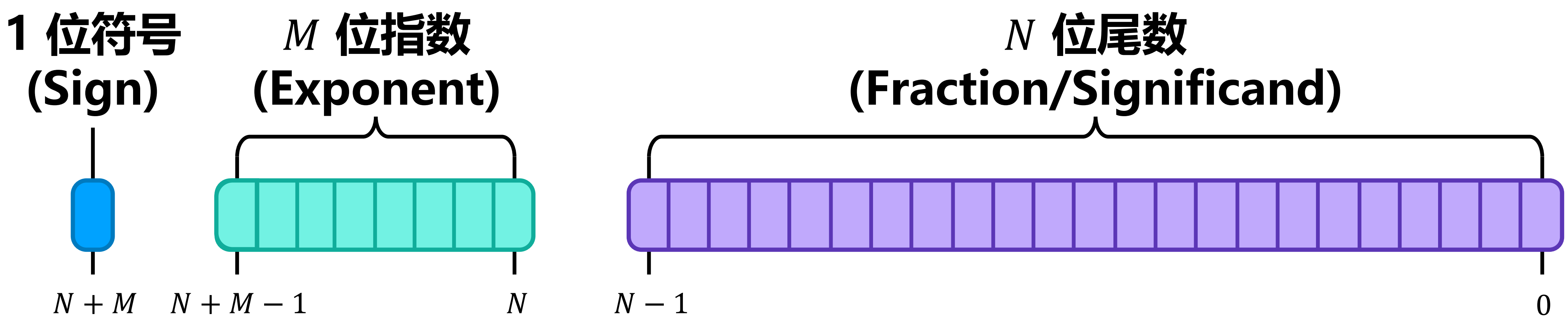
当 $p=-2^{M-1}+1$ 和 $p=2^{M-1}$，即 $c=0$ 和 $c=2^M-1$ 时有特殊含义

  - 实际存储时指数部分为无符号整数 c，数值范围为 $[1, 2^M-2]$，为此需要在 p 上额外**加上**一项指数偏置：$2^{M-1}-1$　⇨ 移码

↳ 便于比较大小，可直接将不同浮点数看作有符号整数来比较大小

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - ❑ 正规化的浮点数表示：$\pm 1.b_1 b_2 b_3 b_4 \cdots b_N \times 2^p$
  - ❑ 其中尾数 $b_2, b_3, b_4 \cdots b_N \in \{0,1\}$；　　指数项 p 是有符号整数



  - ❑ 浮点数中同时存在正 0 和负 0：$\pm 1.0 \times 2^p$，　$p = -2^{M-1} + 1$
    - ▪ 它们数值相等，但在一些特殊运算中能带来便利

```
import math; math.copysign(x, -0.)
```

`1 / (1 / -inf) ⇨ -inf`

实数在计算机中的浮点表示（Floating-point）

FP64 的 $(0.1)_{10} = (0.0\,0011\,0011\cdots0011\,010\,)_2 = \sum_{i=1}^{13}\left(2^{-4i}+2^{-4i-1}\right)+2^{-55}$

$= \frac{3602879701896397}{36028797018963968} = 0.1000000000000000055511151231257827021181583404541015625$

- 对于数字 0.1，它的 FP64 浮点表示为

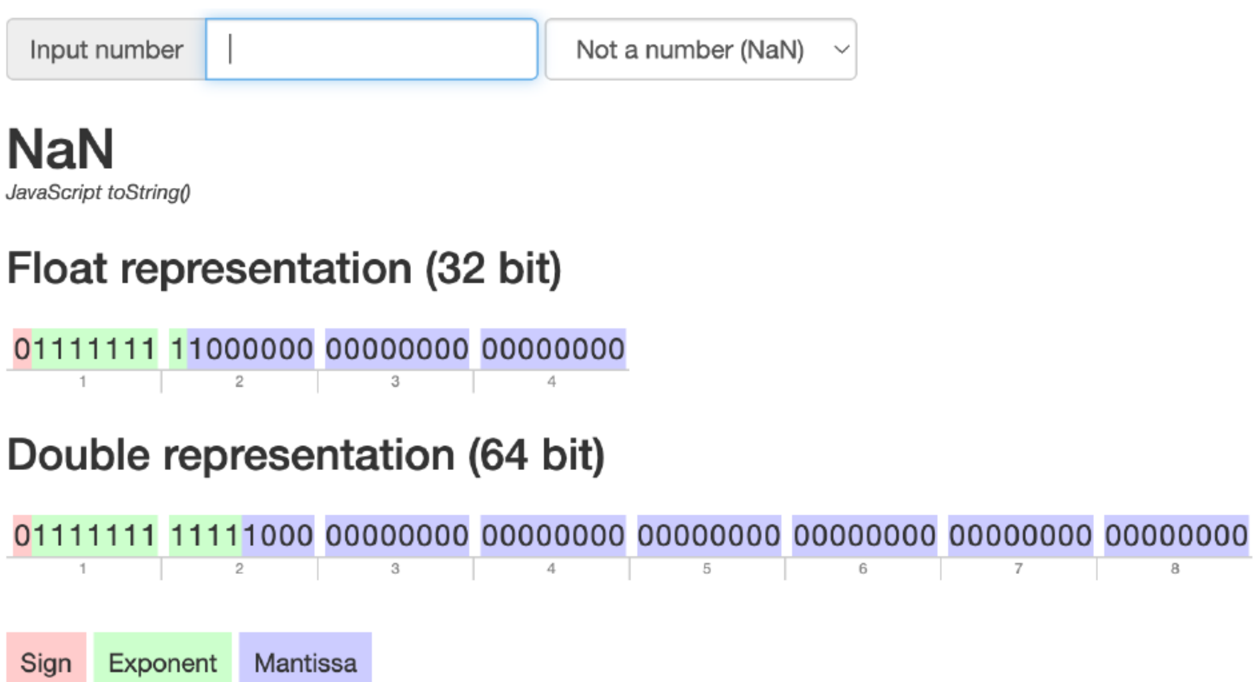
1. 首先转换为二进制表示： $(0.1)_{10} = (0.0\overline{0011})_2$
2. 表示为归一化浮点数形式： $+1.\overline{1001}\times 2^{-4}$
3. 将尾数部分截断至 $N=52$ 位： $+1.\underbrace{\boxed{1001\;1001\;\cdots1010}}_{N=52}1001\cdots\;\times 2^{-4}$ 　$b_{53}=1 \rightarrow b_{52}$进位
4. 加上指数偏置 $2^{M-1}-1=2^{10}-1=1023$： $+1.\boxed{1001\;1001\;\cdots1010}\times 2^{1019}$
5. 最后将各部分的二进制表示拼在一起： $\boxed{0}\boxed{0111\;1111\;011}\boxed{1001\;1001\cdots1010}$

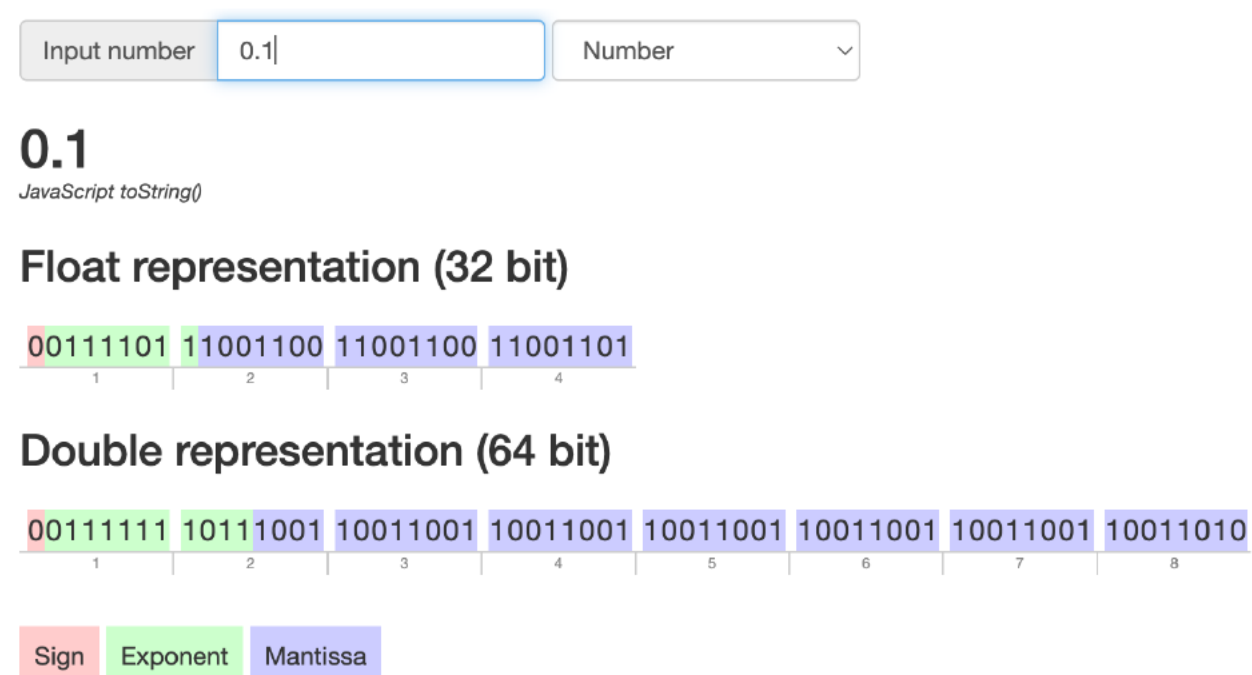
实数在计算机中的浮点表示（Floating-point)

```
>>> # https://stackoverflow.com/a/16444778
>>> import struct
>>> from decimal import Decimal

>>> def fp64_binary(num):
...     return ''.join(
...         '{:0>8b}'.format(c)
...         for c in struct.pack('!d', num)
...     )

>>> print(fp64_binary(0.1))
    0011111110111001100110011001100110011001100110011001100
    11001100110011010
>>> print(Decimal.from_float(0.1))
    0.1000000000000000055511151231257827021181583404541015625
```

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 在线可视化工具
    - https://tools.timodenk.com/ieee-754-floating-point





实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 对于数字 0.1
    - 它的 FP64 浮点表示为 $0\ 0111\ 1111\ 0111001\ 1001 \cdots 1010$
    - 它的 FP64 近似数值为 0.1000000000000000055511151231257827021181583404541015625
      - 误差限 $\varepsilon_r^* < 0.6\cdots\times 10^{-17} < \frac{1}{2}\times 10^{-16}$
      - 有效数字有 16 位

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 对于数字 100 0000 0000.1 = 0.1 + 10^{10}
    - 它的 FP64 浮点表示为 0 1000 0100 0000010101000000010 1111100100000000000 0 0011 0011 0011 0011 01
    - 它的 FP64 近似数值为 10000000000.100000381469726562 5
      - 误差限 $\varepsilon_r^* < 0.4\times10^{-6} < \frac{1}{2}\times10^{-6}$
      - 有效数字有 17 位

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 浮点数加法（不考虑溢出） $c \leftarrow a + b$

$$a = (-1)^{s_1}(1 + f_1) \times 2^{p_1} \qquad b = (-1)^{s_2}(1 + f_2) \times 2^{p_2} \qquad 0 < f_1, f_2 < 1$$

1. 对齐阶码（对阶）

$$p = \max(p_1, p_2) \qquad a' = (-1)^{s_1} g_1' \times 2^{p} \qquad b' = (-1)^{s_2} g_2' \times 2^{p}$$

2. 尾数求和

$$c' = [(-1)^{s_1} g_1' + (-1)^{s_2} g_2'] \times 2^{p}$$

3. 正规化

$$s_3 = \mathrm{CopySign}(c') \qquad c' = (-1)^{s_3}(1 + f_3') \times 2^{p'} \qquad f_3 = \mathrm{RoundOff}(f_3')$$

$$\Rightarrow \quad c = (-1)^{s_3}(1 + f_3) \times 2^{p'}$$

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 浮点数加法（不考虑溢出） $c \leftarrow a + b$

例

计算 $1 + 3\times2^{-53}$：

$$
\begin{aligned}
& \left(1 + 3 \times 2^{-53}\right)_{10} \\
= \; & (1.00\cdots0)_2 \times 2^0 + (11.0\cdots0)_2 \times 2^{-53}
\end{aligned}
$$

① 对阶

$$
\begin{aligned}
= \; & 1.\boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000} \quad \times 2^0 \\
& +0.\boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001}\,1 \times 2^0
\end{aligned}
$$

② 尾数求和

$$
= 1.\boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001}\,1 \times 2^0
$$

③ 正规化

$$
\begin{aligned}
\rightarrow \; & 1.\boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010} \quad \times 2^0 \\
= \; & (1 + 2^{-51})_{10}
\end{aligned}
$$

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 浮点数乘法（不考虑溢出） $c \leftarrow a \times b$

$$a = (-1)^{s_1}(1+f_1)\times 2^{p_1} \qquad b = (-1)^{s_2}(1+f_2)\times 2^{p_2} \qquad 0 < f_1, f_2 < 1$$

1. 尾数相乘，指数相加，符号位异或

$$g_3 = (1+f_1)\times(1+f_2)$$

$$p_3 = p_1 + p_2$$

$$s_3 = s_1 \text{ XOR } s_2$$

2. 正规化

$$c = (-1)^{s_3}(1+f_3)\times 2^{p_3'}$$

实数在计算机中的浮点表示（Floating-point）

- IEEE 754 二进位浮点数算术标准
  - 最近舍入准则（Round to nearest）
  - 机器的有限字长 ⇨ 大多数浮点数本身存在舍入误差
    ⇨ 每步算术运算都将带入舍入误差
  - **例：** 计算 $(1+3\times2^{-53})-1$

$$
\begin{aligned}
& (1+3\times2^{-53})_{10} \\
= \ & (1.00\cdots0)_2\times2^0+(11.0\cdots0)_2\times2^{-53} \\
= \ & 1.\ \boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000}\quad\ \ \times2^0 \\
+ & 0.\ \boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001}\ 1\ \times2^0 \\
= \ & 1.\ \boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001}\ 1\ \times2^0 \\
\to \ & 1.\ \boxed{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010}\quad\ \ \times2^0 \\
= \ & (1+2^{-51})_{10}
\end{aligned}
$$

浮点数运算中的反直觉事实

- 算术运算往往不再遵循结合律
  - $(a+b)+c$ vs. $a+(b+c)$
  - 大数“吃”小数
- 数学上等价的不同计算公式，其实际计算结果可能相差甚远
  - $1-\cos(x)$ vs. $2*\sin^2(x/2)$ 当 x 靠近 2π 整数倍时
  - $(1+x)^2-1$ vs. $x*(2+x)$ 当 $x \ll 1$ 时
  - $\sqrt{x+1}-\sqrt{x}$ vs. $1/\left(\sqrt{x+1}+\sqrt{x}\right)$ 当 $x \ll 1$ 时
  - e^x-1 vs. $expm1(x)$ 当 $x \ll 1$ 时
  - $\ln(1+x)$ vs. $Log1p(x)$ 当 $x \ll 1$ 时

如何避免有效数字的丢失?

- 避免 $|x| \gg |y|$ 时进行除法 x/y
  - 可通过调整运算顺序避免
- 避免 $x \approx y$ 时进行减法 $x - y$
  - 可尝试用其他等价的运算替代，如

$$\log_{10} x - \log_{10} y \ \Rightarrow\ \log_{10}\left(\frac{x}{y}\right) \qquad \ln\left(x - \sqrt{x^2-1}\right) \Rightarrow -\ln\left(x + \sqrt{x^2-1}\right)$$

- 当运算涉及数量级相差很大的多个数字时，应注意运算顺序，防止“大数吃小数”
  - 优先处理数量级相近的数字间的运算
- 简化计算步骤，减少运算次数（如秦九韶算法、FFT）

如何避免有效数字的丢失?

例

在 3 位有效数字的计算机上计算 $\sqrt{9.01}-3$:

注: $\sqrt{9.01} \approx 3.0016662$

- 直接计算: $\sqrt{9.01}-3 \approx 3.00-3=0.00$
- 平方差公式:

$$
\begin{aligned}
\sqrt{9.01}-3 &= \frac{(\sqrt{9.01}-3)(\sqrt{9.01}+3)}{\sqrt{9.01}+3} \\
&= \frac{9.01-9}{\sqrt{9.01}+3} \approx \frac{0.01}{3.00+3} \\
&\approx 0.00166667 \approx 0.00167 = 1.67\times 10^{-3}
\end{aligned}
$$

如何避免有效数字的丢失?

例

二次方程 $x^2+9^{12}x-3=0$ 求根:

- 直接计算: $x_1, x_2=\frac{-b\pm\sqrt{b^2-4ac}}{2a}=\frac{-9^{12}\pm\sqrt{9^{24}+4\times 3}}{2}$ $\Rightarrow$ $x_1^*=0$
- 平方差公式:

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}=\frac{\left(-b+\sqrt{b^2-4ac}\right)\left(b+\sqrt{b^2-4ac}\right)}{2a\left(b+\sqrt{b^2-4ac}\right)}$$

$$=\frac{-4ac}{2a\left(b+\sqrt{b^2-4ac}\right)}=\frac{-2c}{b+\sqrt{b^2-4ac}}$$

$$=\frac{-2\times(-3)}{9^{12}+\sqrt{9^{24}+4\times 3}}=\frac{6}{9^{12}+\sqrt{9^{24}+12}}\approx 1.0622\times 10^{-11}$$

如何避免有效数字的丢失?

例

在 5 位有效数字的计算机上求解 $f(x) = e^x - x - 1$ 在 $x = 0.01$ 处的值:

- 直接计算: $f(0.01) = e^{0.01} - 0.01 - 1 \approx 1.0101 - 0.01 - 1 = 0.0001$
- 泰勒展开:

$$f(x) = e^x - x - 1 = \left(1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}\xi^4\right) - x - 1 \approx \frac{1}{2!}x^2 + \frac{1}{3!}x^3$$

$$f(0.01) \approx 0.50000 \times 0.01^2 + 0.16667 \times 0.01^3$$

$$= (0.50000 + 0.0016667) \times 10^{-4}$$

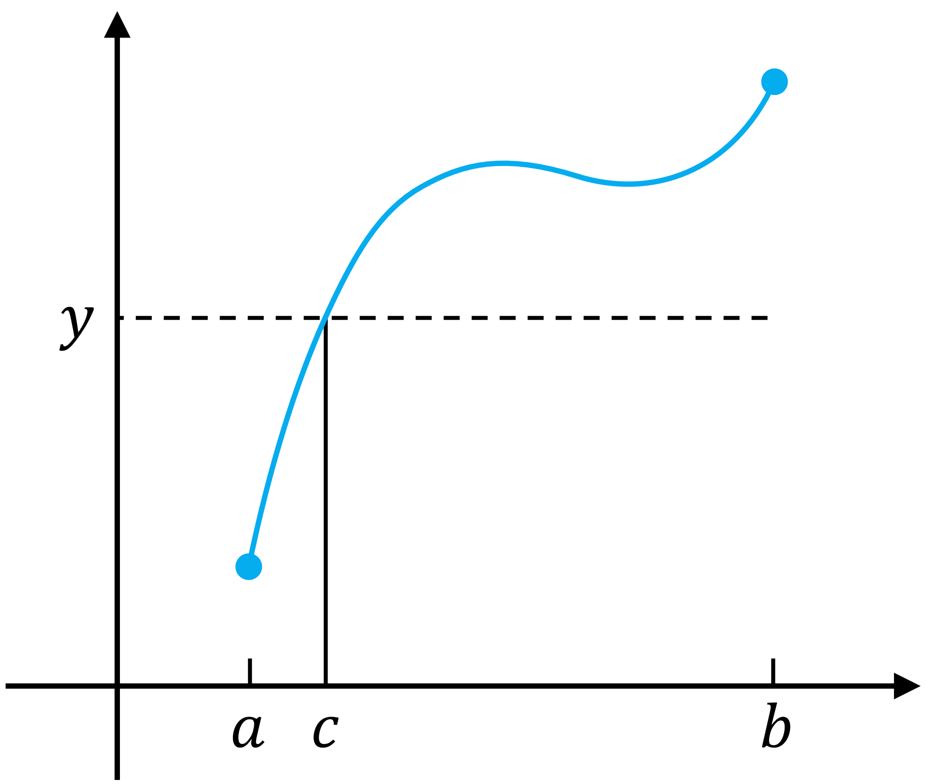
$$\approx 5.0167 \times 10^{-5}$$

目录

CONTENTS

微积分基础

- 介值定理

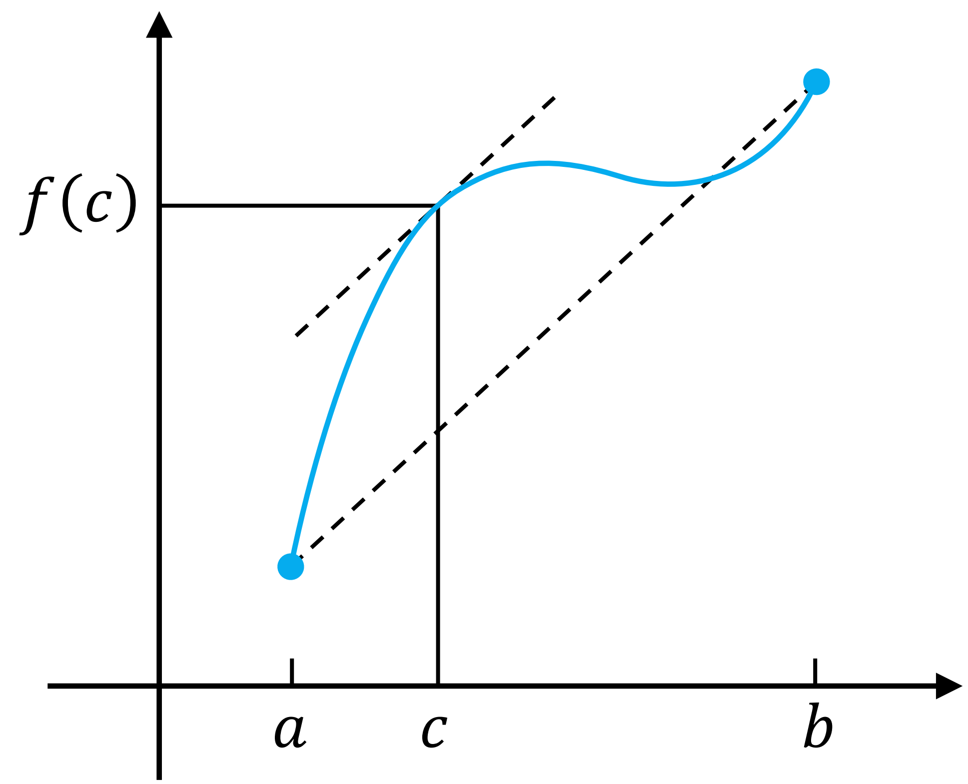


介值定理（Intermediate Value Theorem）

令 f 是区间 $[a,b]$ 上的一个连续函数，则 f 可以得到 $f(a)$ 和 $f(b)$ 之间的所有值。更准确地说，如果 y 是 $f(a)$ 和 $f(b)$ 之间的数，则存在数字 c，$a \le c \le b$，使得 $f(c) = y$.

微积分基础

- 介值定理
- 均值定理

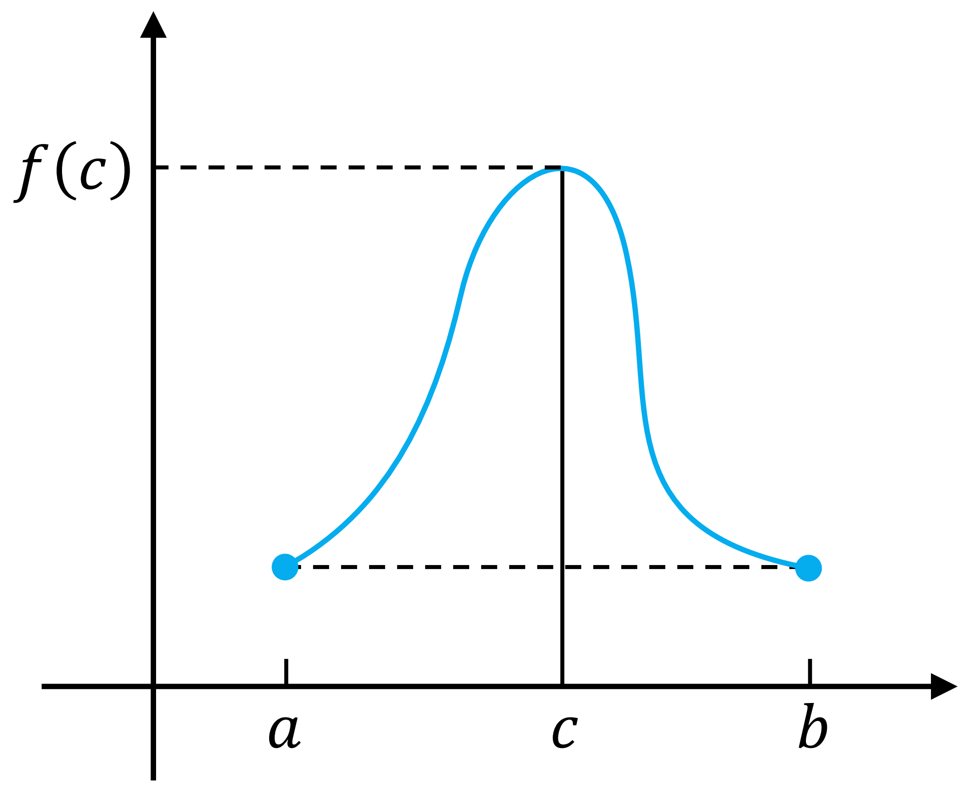


均值定理（Mean Value Theorem）

令 f 是区间 $[a,b]$ 上的连续可微函数，则在 a 和 b 之间存在数 c 使得 $f'(c) = \frac{f(b)-f(a)}{b-a}$.

微积分基础

- 介值定理
- 均值定理
- 罗尔定理

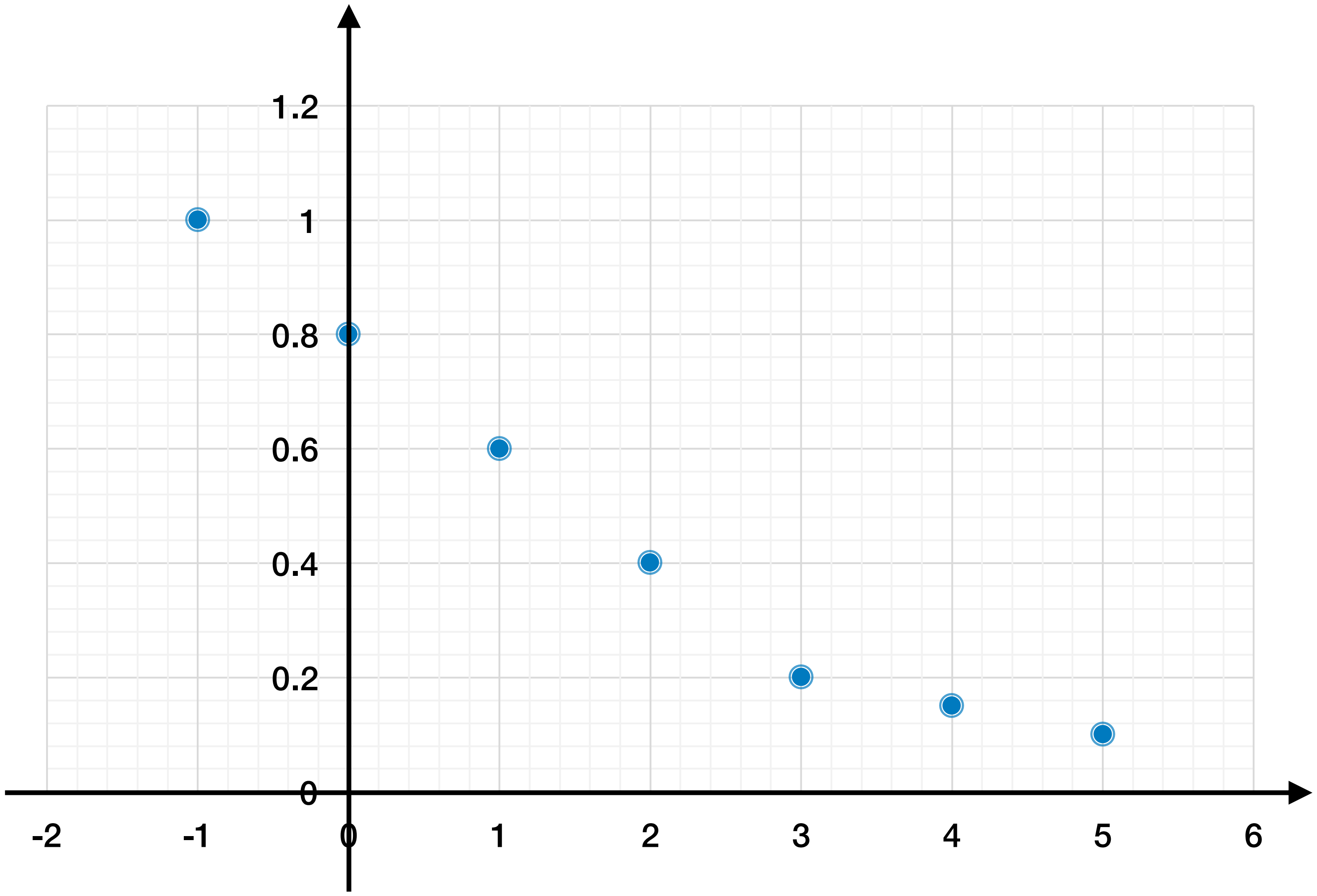


罗尔定理（Rolle's Theorem）

令 f 是区间 $[a, b]$ 上的连续可微函数，并假设 $f(a) = f(b)$，则在 a 和 b 之间存在数 c 使得 $f'(c) = 0$.

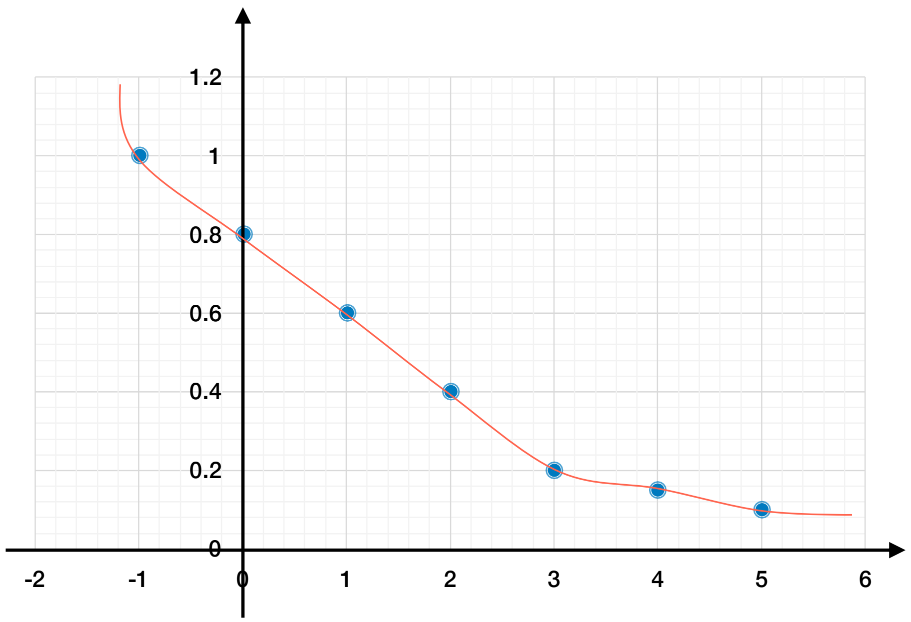
插值（Interpolation）的概念

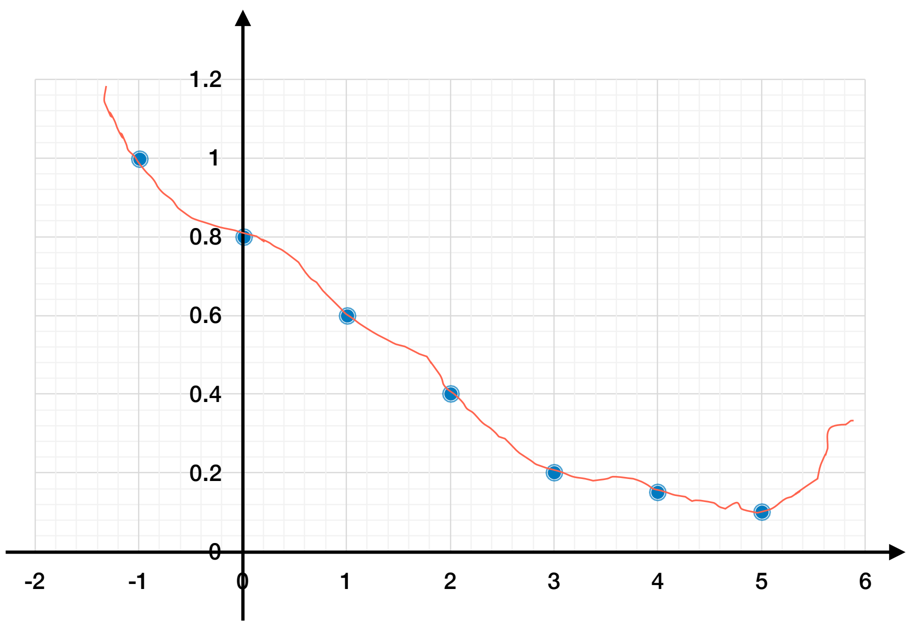
- 已知一系列数据点 $\{(x_i,\ y_i)\}$，如何找到函数满足 $P(x_i) = y_i$?

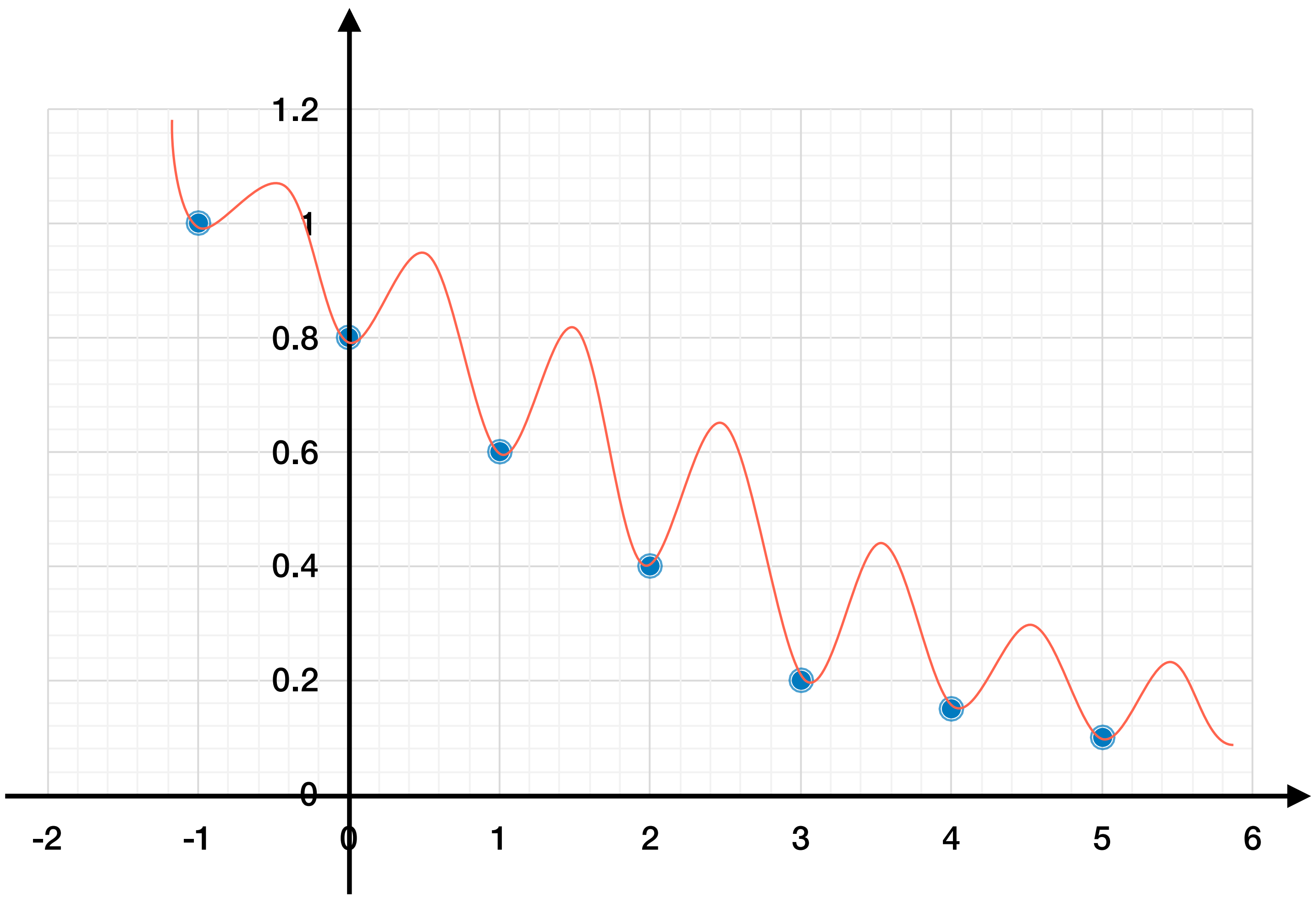


插值（Interpolation）的概念

- 已知一系列数据点 $\{(x_i,\ y_i)\}$，如何找到函数满足 $P(x_i) = y_i$?







插值（Interpolation）的概念

- 为什么需要插值?
  - 许多实际问题都可以用某个函数 $f(x)$ 表示其内在变化规律
  - $f(x)$ 通常没有解析形式，只能通过实验/观测得到数据表
  - 如何根据这些数据估计其他点的函数值?　　⇨　　插值

定义

设函数 $f(x)$ 在区间 $[a,b]$ 上有定义，且已知在点 $a \le x_0 \le x_1 < \cdots < x_n \le b$ 上的值 $y_0, y_1, \cdots, y_n$，若存在一简单函数 $p(x)$，使得

$$p(x_i) = y_i \qquad (i = 0,1,\cdots,n)$$

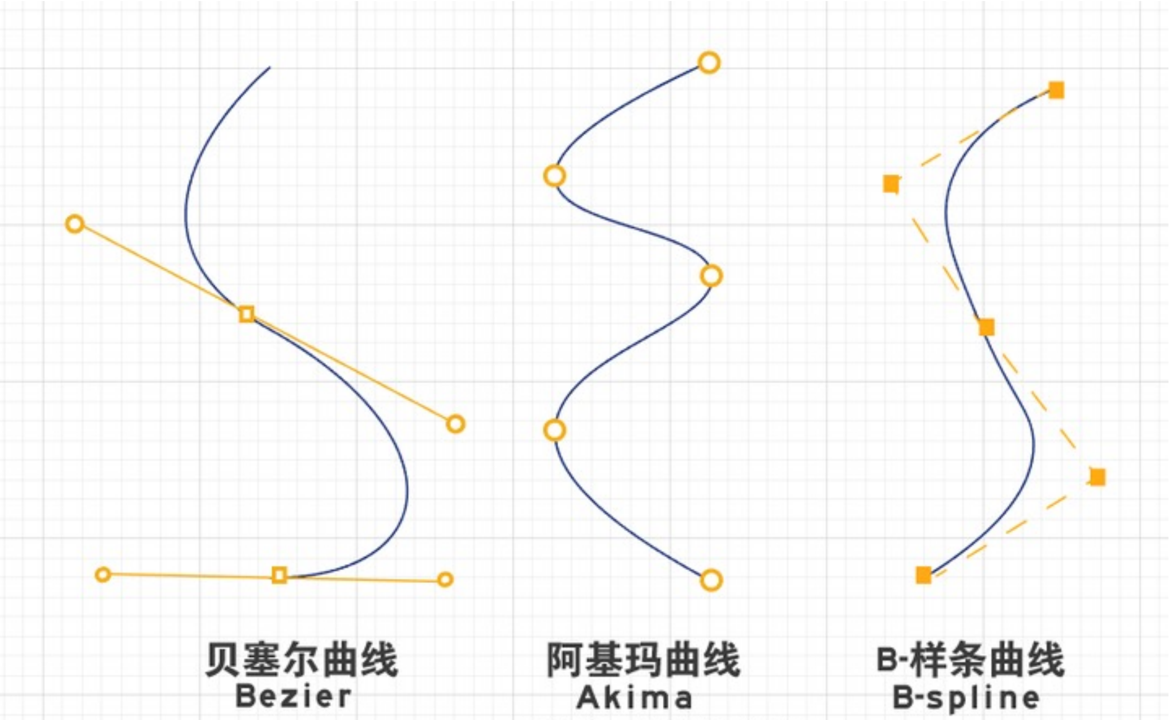
成立，则称 $p(x)$ 为 $f(x)$ 的插值函数。求 $p(x)$ 的方法称为插值法。

插值法的发展

- 在我国古代，插值法的发展与天文历法息息相关
  - 东汉时期（约公元 206 年），刘洪在《乾象历》中使用了一次插值公式计算月行度数
  - 隋朝时期（约公元 600 年），刘焯在《皇极历》中使用等间距节点二次插值公式推算出五星位置和日、月食的起运时刻
  - 唐朝时期（约公元 727 年），僧一行在《大衍历》中将插值法推广到了二次不等间距的情形
  - 元朝时期（约公元 1280 年），王恂、郭守敬等在《授时历》中提出“招差法”（类似差分插值）的插值法
  - 元朝时期（约公元 1303 年），朱世杰在《四元玉鉴》中提出了一个四次插值公式

插值法的发展

- 系统的插值理论是在 17 世纪微积分产生之后才逐渐发展起来的
  - 1795 年，法国数学家拉格朗日在其著作《师范学校数学基础教程》中提出了一种插值方法，后人称之为拉格朗日插值法
  - 20 世纪 50-60 年代，样条函数被用于解决早期计算机辅助设计（CAD）和制造（CAM）中的曲线绘制问题，进而发展为样条插值



常见插值方法

- 多项式插值：$p(x)$ 为多项式，最简单和常用的插值函数
  - 线性插值：$p(x)$ 为一次多项式
  - 抛物线插值：$p(x)$ 为二次多项式
- 分段插值： $p(x)$ 为分段多项式
  - 样条插值：$p(x)$ 为样条函数（一种特殊的分段多项式）
- 三角插值：$p(x)$ 为三角函数
- 有理插值：$p(x)$ 为有理函数
- ……

多项式插值的定义

（插值区间）（插值节点）

- 已知函数 $y = f(x)$ 在区间 $[a, b]$ 上的 $n+1$ 个点

$$a \le x_0 < x_1 < \cdots < x_n \le b$$

处的函数值为 $y_0 = f(x_0), y_1 = f(x_1), \cdots, y_n = f(x_n)$

求次数不超过 n 的多项式

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n \qquad (c_0, c_1, c_2, \cdots, c_n \in \mathbb{R})$$

使得

$$p(x_i) = y_i \qquad i = 0, 1, \cdots, n$$

定理（插值多项式的存在唯一性）

满足上述条件的多项式 $p(x)$ 存在且唯一。

多项式插值的定义

- 证明

满足上述条件的多项式 $p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$ 存在且唯一

$\Updownarrow$

以下线性方程组的解存在且唯一

$$\begin{cases} c_0 + c_1 x_0 + c_2 x_0^2 + \cdots + c_n x_0^n & = y_0 \\ c_0 + c_1 x_1 + c_2 x_1^2 + \cdots + c_n x_1^n & = y_1 \\ \qquad \vdots & \quad \vdots \\ c_0 + c_1 x_n + c_2 x_n^2 + \cdots + c_n x_n^n & = y_n \end{cases} \Longleftrightarrow$$

Cramer 法则

Vandermonde 行列式

$$c_i = \frac{D_i}{D} \Rightarrow D = \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{vmatrix} \neq 0$$

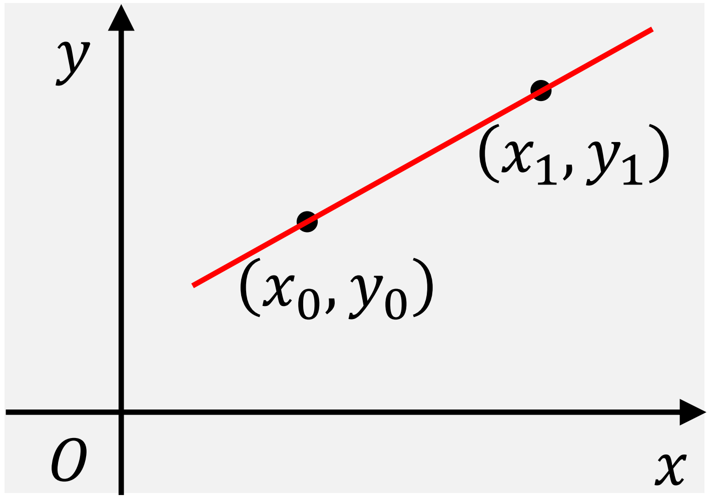
$\Uparrow$

已知 $a \le x_0 < x_1 < \cdots < x_n \le b \quad \Rightarrow \quad x_0 \neq x_1 \neq \cdots \neq x_n$

多项式插值的一些特例

- 一般形式
$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n$$
其中 $c_0, c_1, c_2, \cdots, c_n \in \mathbb{R}$
- 线性插值（$n = 1$）
$$p(x) = c_0 + c_1 x$$
- 抛物线插值（$n = 2$）
$$p(x) = c_0 + c_1 x + c_2 x^2$$
- 三次插值（$n = 3$）
$$p(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$$
- ……

多项式插值的一些特例



- 线性插值（$n = 1$）

$$p(x) = c_0 + c_1 x$$

❑ 已知区间 $[x_0, x_1]$ 端点处的函数值 $y_0 = f(x_0)$ 和 $y_1 = f(x_1)$，则通过这两点的线性插值多项式为

$$p(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) \qquad \text{(点斜式)}$$

$$p(x) = y_0 \underbrace{\frac{x - x_1}{x_0 - x_1}}_{l_0} + y_1 \underbrace{\frac{x - x_0}{x_1 - x_0}}_{l_1} \qquad \text{(两点式)}$$

$$\Downarrow$$

$$p(x) = y_0 l_0 + y_1 l_1$$

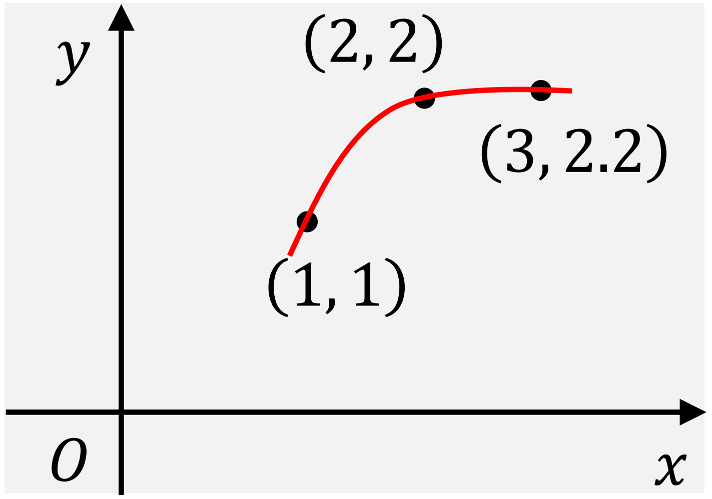
其中 l_0 和 l_1 为一次多项式

$$l_0(x_0) = 1,\ l_0(x_1) = 0$$
$$l_1(x_0) = 0,\ l_1(x_1) = 1$$

多项式插值的一些特例

- 抛物线插值（$n=2$）

$$p(x) = c_0 + c_1 x + c_2 x^2$$



例

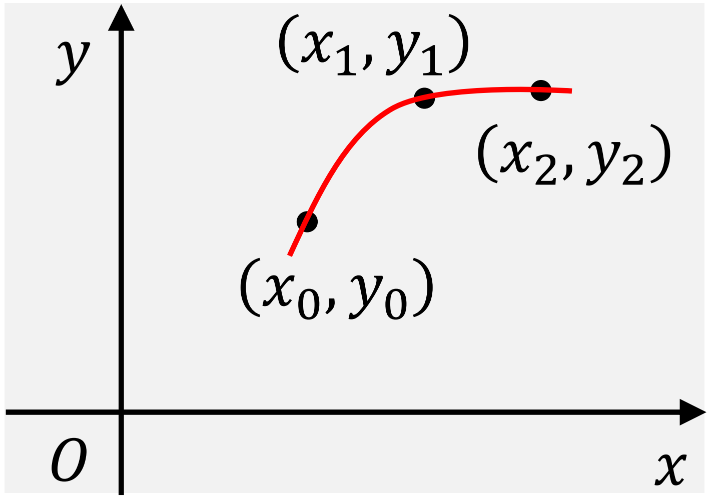
已知三个插值节点 $(1,1),(2,2),(3,2.2)$，求通过这三点的抛物线插值多项式。

列方程组求解

$$\begin{cases} c_0 + c_1 \cdot 1 + c_2 \cdot 1^2 = 1 \\ c_0 + c_1 \cdot 2 + c_2 \cdot 2^2 = 2 \\ c_0 + c_1 \cdot 3 + c_2 \cdot 3^2 = 2.2 \end{cases} \Rightarrow \begin{cases} c_0 + c_1 + c_2 = 1 \\ c_0 + 2c_1 + 4c_2 = 2 \\ c_0 + 3c_1 + 9c_2 = 2.2 \end{cases}$$

$$\Rightarrow \begin{cases} c_0 = -0.8 \\ c_1 = 2.2 \\ c_2 = -0.4 \end{cases} \Rightarrow \quad p(x) = -0.8 + 2.2x - 0.4x^2$$

多项式插值的一些特例



- 抛物线插值（$n = 2$）

$$p(x) = c_0 + c_1 x + c_2 x^2$$

- 已知区间 $[x_0, x_2]$ 内**不共线**的三点处的函数值 $y_0 = f(x_0)$，$y_1 = f(x_1)$ 和 $y_2 = f(x_2)$，则通过这三点的抛物线插值多项式为

$$p(x) = y_0 \underbrace{\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}}_{l_0} + y_1 \underbrace{\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}}_{l_1} + y_2 \underbrace{\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}}_{l_2}$$

$$\Downarrow$$

$$p(x) = y_0 l_0 + y_1 l_1 + y_2 l_2$$

其中 l_0，l_1 和 l_2 为二次多项式

$$l_0(x_0) = 1,\ l_0(x_1) = 0,\ l_0(x_2) = 0$$
$$l_1(x_0) = 0,\ l_1(x_1) = 1,\ l_1(x_2) = 0$$
$$l_2(x_0) = 0,\ l_2(x_1) = 0,\ l_2(x_2) = 1$$

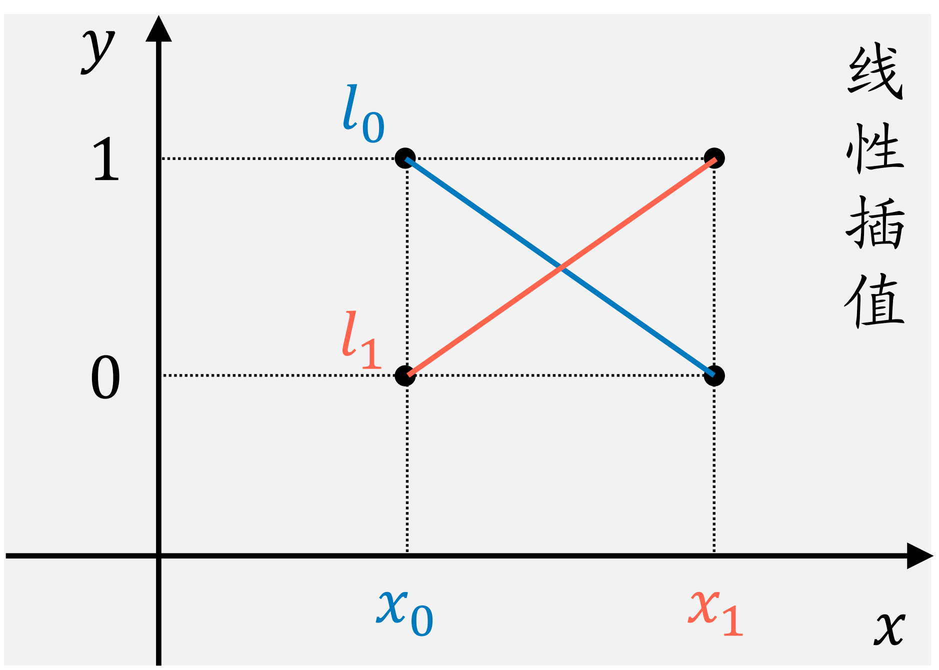
多项式插值的一些特例

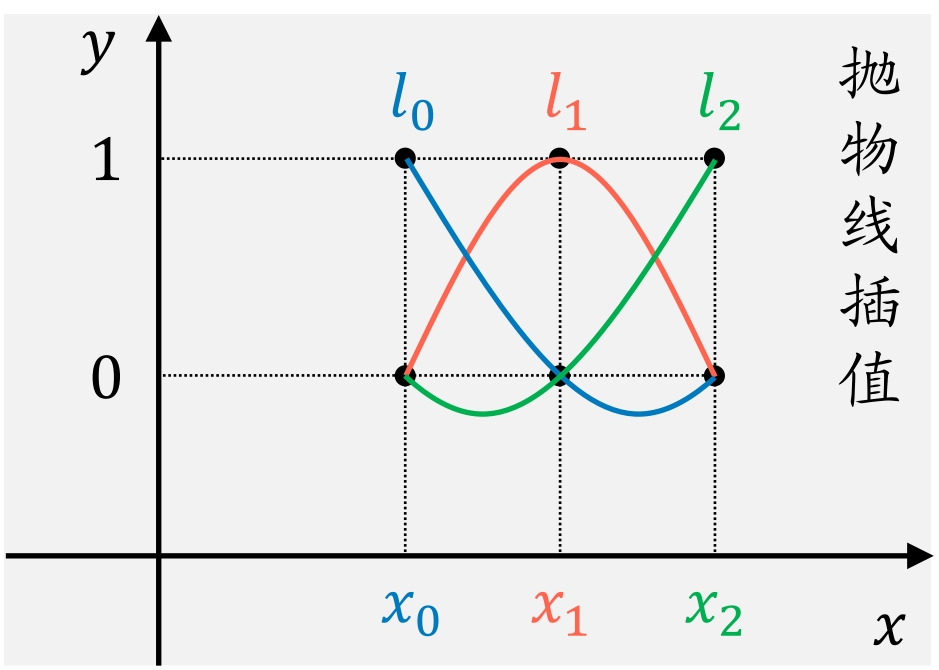
- 线性插值（$n = 1$）

$$p(x) = y_0 l_0 + y_1 l_1 \quad \Rightarrow \quad l_0, l_1 \text{ 称为一次插值基函数}$$

- 抛物线插值（$n = 2$）

$$p(x) = y_0 l_0 + y_1 l_1 + y_2 l_2 \quad \Rightarrow \quad l_0, l_1, l_2 \text{ 称为二次插值基函数}$$





多项式插值的一些特例

- 将上述方法推广至一般情况（n 次多项式）
  - $p(x) = c_0 z_0(x) + c_1 z_1(x) + \cdots + c_n z_n(x)$

    其中 $z_0(x), z_1(x), \cdots, z_n(x)$ 是 $\mathcal{H}_n$ 的一组基　　基函数插值法

    $\mathcal{H}_n$ 表示次数不超过 n 的所有多项式的集合
  - 如何寻找合适的插值基函数?
    1. 分别以 $1, x, x^2, \cdots, x^n$ 作为插值基函数
    2. 分别以 $l_0(x), l_1(x), l_2(x), \cdots, l_n(x)$ 作为插值基函数
    3. ……

    Lagrange 插值法

Lagrange 插值法

- 用 Lagrange 插值基函数来计算插值多项式的方法

定义

若 n 次多项式 $l_i(x)$（$i = 0, 1, \cdots, n$）在 $n+1$ 个节点 $x_0 < x_1 < \cdots < x_n$ 上满足条件

$$l_i(x_k) = \begin{cases} 1, & k = i, \\ 0, & k \neq i \end{cases} \qquad (i, k = 0, 1, \cdots, n)$$

则称 $l_0(x), l_1(x), \cdots, l_n(x)$ 为节点 $x_0, x_1, \cdots, x_n$ 上的 n 次 Lagrange 插值基函数。

- 如何计算 $l_0(x), l_1(x), \cdots, l_n(x)$ 的具体数学形式?

Lagrange 插值法

- 如何计算 $l_0(x), l_1(x), \cdots, l_n(x)$ 的具体数学形式?

$$l_i(x_k) = \begin{cases} 1, & k = i, \\ 0, & k \neq i \end{cases} \qquad (i, k = 0, 1, \cdots, n)$$

❑ 根据定义，可设

$$l_i(x) = a_i(x - x_0)(x - x_1)\cdots(x - x_{i-1})(x - x_{i+1})\cdots(x - x_n)$$

❑ 将 $l_i(x_i) = 1$ 代入可求得

$$a_i = \frac{1}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}$$

$$l_i(x) = \prod_{k=0,\, k\neq i}^{n} \frac{x - x_k}{x_i - x_k} \qquad (i = 0, 1, 2, \cdots, n)$$

Lagrange 插值法

- Lagrange 插值多项式

$$p(x) = c_0 l_0(x) + c_1 l_1(x) + c_2 l_2(x) + \cdots + c_n l_n(x)$$

将插值条件 $p(x_0) = y_0, p(x_1) = y_1, \cdots, p(x_n) = y_n$ 代入可得

$$c_i = y_i \qquad (i = 0, 1, \cdots, n)$$

⬇

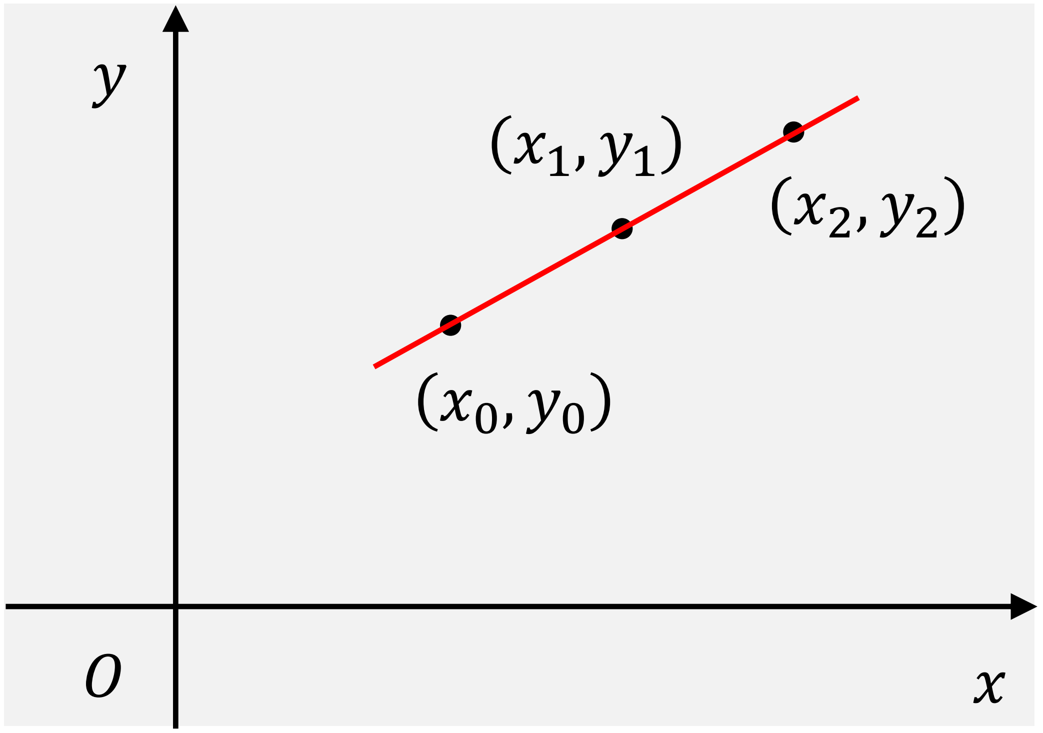
$$p(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) + \cdots + y_n l_n(x) \triangleq L_n(x)$$

$$= \sum_{i=0}^{n} y_i l_i(x) = \sum_{i=0}^{n} y_i \prod_{k=0,\, k\neq i}^{n} \frac{x - x_k}{x_i - x_k}$$

Lagrange 插值多项式

Lagrange 插值法

- n 次插值多项式的次数可能会低于 n
  - 比如在抛物线插值（$n = 2$）中，若三点共线，则 $L_2(x)$ 为直线，插值多项式退化为线性插值多项式



- 参考阅读：用 Python 实现 Lagrange 插值

Lagrange 插值法

```
>>> from scipy.interpolate import lagrange
>>> Ln = lagrange([1, 2], [0.0175, 0.0349])
>>> print(Ln(1.2))     # 0.020980000000000002
```

例

计算角 $\alpha = 1.2^\circ$ 的正弦函数值.

角度（°）	sin	cos	tan
0	0.0000	1.0000	0.0000
1	0.0175	0.9998	0.0175
2	0.0349	0.9994	0.0349
3	0.0523	0.9986	0.0524
4	0.0698	0.9976	0.0699
…	…	…	…

查表法 + Lagrange 插值

一次：$L_1(\alpha) = 0.0175 \cdot \frac{\alpha - 2^\circ}{1^\circ - 2^\circ} + 0.0349 \cdot \frac{\alpha - 1^\circ}{2^\circ - 1^\circ}$

$L_1(1.2^\circ) = 0.0175 \cdot 0.8 + 0.0349 \cdot 0.2 \approx 0.0210$

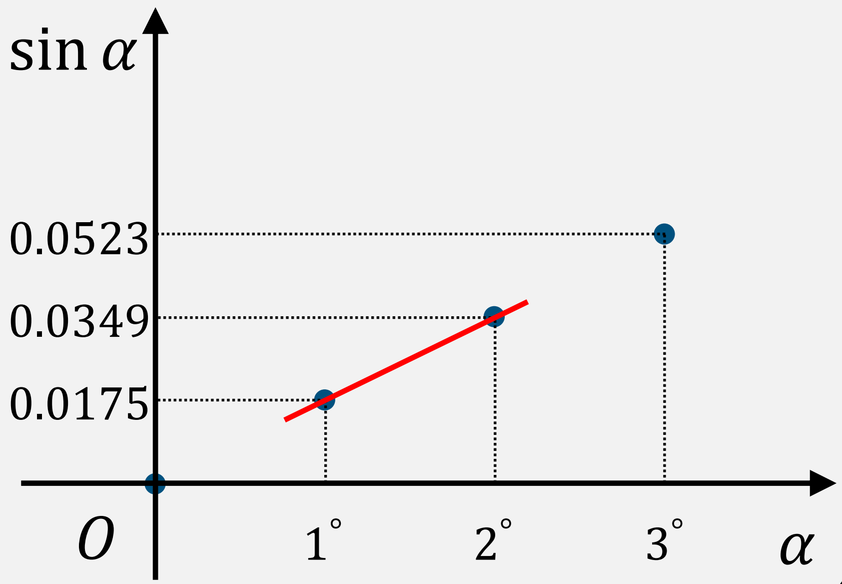
二次：$L_2(\alpha) = 0.0175 \cdot \frac{(\alpha - 0^\circ)(\alpha - 2^\circ)}{(1^\circ - 0^\circ)(1^\circ - 2^\circ)} + 0.0349 \cdot \frac{(\alpha - 0^\circ)(\alpha - 1^\circ)}{(2^\circ - 0^\circ)(2^\circ - 1^\circ)}$

$L_2(1.2^\circ) = 0.0175 \cdot 0.96 + 0.0349 \cdot 0.12 \approx 0.0210$

十二次：$L_{12}(1.2^\circ) \approx 0.020928439522237215 \approx 0.0209$

二十次：$L_{20}(1.2^\circ) \approx 0.016093000057113692 \approx 0.0161$

三十次：$L_{30}(1.2^\circ) \approx -5.186944966913506 \approx -5.187$



```
>>> from scipy.interpolate import lagrange
>>> Ln = lagrange([1, 2], [0.0175, 0.0349])
>>> print(Ln(1.2))     # 0.020980000000000002
```

Lagrange 插值法

例

计算角 $\alpha = 1.2^\circ$ 的正弦函数值.

角度（°）	sin	cos	tan
...			
-2	-0.0349	0.9994	-0.0349
-1	-0.0175	0.9998	-0.0175
0	0.0000	1.0000	0.0000
1	0.0175	0.9998	0.0175
2	0.0349	0.9994	0.0349
...	...	...	...

查表法 + Lagrange 插值

一次：$L_1(\alpha) = 0.0175 \cdot \frac{\alpha-2^\circ}{1^\circ-2^\circ} + 0.0349 \cdot \frac{\alpha-1^\circ}{2^\circ-1^\circ}$

$L_1(1.2^\circ) = 0.0175 \cdot 0.8 + 0.0349 \cdot 0.2 \approx 0.0210$

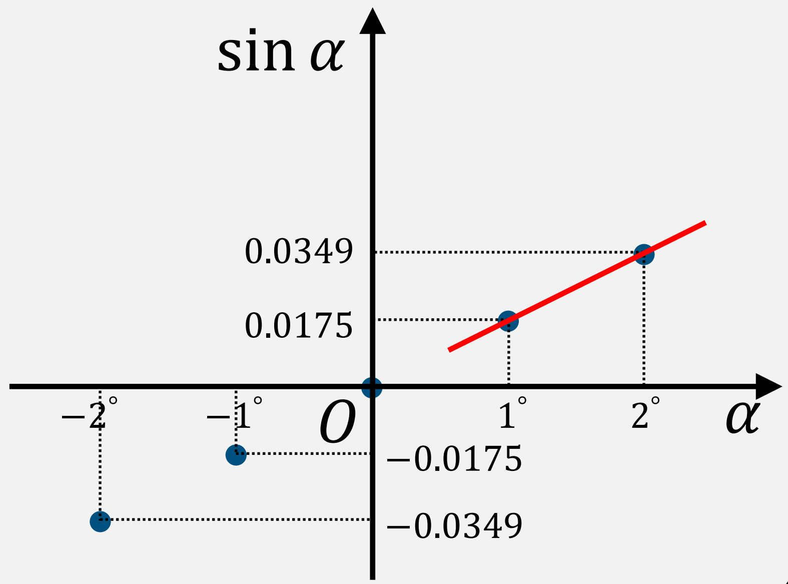
二次：$L_2(\alpha) = 0.0175 \cdot \frac{(\alpha-0^\circ)(\alpha-2^\circ)}{(1^\circ-0^\circ)(1^\circ-2^\circ)} + 0.0349 \cdot \frac{(\alpha-0^\circ)(\alpha-1^\circ)}{(2^\circ-0^\circ)(2^\circ-1^\circ)}$

$L_2(1.2^\circ) = 0.0175 \cdot 0.96 + 0.0349 \cdot 0.12 \approx 0.0210$

十二次：$L_{12}(1.2^\circ) \approx 0.02098781757130344 8 \approx 0.0210$

二十次：$L_{20}(1.2^\circ) \approx 0.0209881012676337 2 \approx 0.0210$

三十次：$L_{30}(1.2^\circ) \approx 0.0209882413942554 84 \approx 0.0210$



Lagrange 插值法的误差分析

- 若在 $[a,b]$ 上用 $L_n(x)$ 近似 $f(x)$，则其截断误差为
$$R_n(x) = f(x) - L_n(x),$$
也称为插值余项（插值多项式的余项）。

定理（证明见教材第 2.2.4 节）

设 $f^{(n)}(x)$ 在 $[a,b]$ 上连续， $f^{(n+1)}(x)$ 在 $[a,b]$ 内存在，节点 $a \le x_0 < x_1 < \cdots < x_n \le b$，$L_n(x)$ 是 Lagrange 插值多项式，则对于任何 $x \in [a,b]$，插值余项
$$R_n(x) = f(x) - L_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x).$$
这里 $\xi \in (a,b)$ 且依赖于 x，$\omega_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$.

Lagrange 插值法的误差分析

- 插值余项 $R_n(x) = f(x) - L_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x)$

$$\omega_{n+1}(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$$

- 在实际分析中，ξ 在区间 $[a, b]$ 中的具体位置通常难以得到
- 如果可以求出

$$M_{n+1} \overset{\text{def}}{=} \max_{a\le x\le b}\left|f^{(n+1)}(x)\right|$$

则插值多项式 $L(x)$ 逼近 $f(x)$ 的截断误差限是

$$|R_n(x)| = |f(x) - L_n(x)| \le \frac{M_{n+1}}{(n+1)!}|\omega_{n+1}(x)|$$

Lagrange 插值法的误差分析

$$|R_n(x)| = |f(x) - L_n(x)| \le \frac{M_{n+1}}{(n+1)!}|\omega_{n+1}(x)|$$

例

已知 $\sin 0^\circ = 0$， $\sin 1^\circ = 0.0175$， $\sin 2^\circ = 0.0349$，用线性插值和抛物线插值分别计算 $\sin 1.2^\circ$ 的值，并估计插值余项的最大值。

$f(\alpha) = \sin\alpha^\circ = \sin r$， 弧度 $r = \frac{\alpha^\circ}{180^\circ}\cdot\pi$

$f'(\alpha) = \frac{df}{dr}\cdot\frac{dr}{d\alpha} = \cos r\cdot\frac{\pi}{180}$

$f''(\alpha) = \frac{df'}{dr}\cdot\frac{dr}{d\alpha} = -\sin r\cdot\left(\frac{\pi}{180}\right)^2$

$f''(\alpha) = \frac{df''}{dr}\cdot\frac{dr}{d\alpha} = -\cos r\cdot\left(\frac{\pi}{180}\right)^3$

线性插值（$n = 1$，取 $x_0 = 1^\circ$，$x_1 = 2^\circ$）

$$L_1(\alpha) = 0.0175\cdot\frac{\alpha - 2^\circ}{1^\circ - 2^\circ} + 0.0349\cdot\frac{\alpha - 1^\circ}{2^\circ - 1^\circ}$$

$$L_1(1.2^\circ) = 0.0175\cdot 0.8 + 0.0349\cdot 0.2 = 0.02098 \approx 0.0210$$

$R_1(x) \le \frac{M_2}{2!}|(x-1^\circ)(x-2^\circ)|$，其中 $M_2 = \max\limits_{1^\circ\le x\le 2^\circ}|f''(x)| = \max\limits_{1^\circ\le x\le 2^\circ}|\sin x|\cdot\left(\frac{\pi}{180}\right)^2$

$\Rightarrow$ $M_2 \le 1.1\times10^{-5}$ $\Rightarrow$ $R_1(1.2^\circ) \le \frac{1}{2}\times1.1\times10^{-5}\times|(1.2^\circ - 1^\circ)(1.2^\circ - 2^\circ)| = 8.8\times10^{-7}$

Lagrange 插值法的误差分析

$$|R_n(x)| = |f(x) - L_n(x)| \le \frac{M_{n+1}}{(n+1)!}|\omega_{n+1}(x)|$$

例

已知 $\sin 0^\circ = 0$， $\sin 1^\circ = 0.0175$， $\sin 2^\circ = 0.0349$，用线性插值和抛物线插值分别计算 $\sin 1.2^\circ$ 的值，并估计插值余项的最大值。

抛物线插值（$n=2$，取 $x_0 = 0^\circ$，$x_1 = 1^\circ$，$x_2 = 2^\circ$）

$$L_2(\alpha) = 0.0175 \cdot \frac{(\alpha - 0^\circ)(\alpha - 2^\circ)}{(1^\circ - 0^\circ)(1^\circ - 2^\circ)} + 0.0349 \cdot \frac{(\alpha - 0^\circ)(\alpha - 1^\circ)}{(2^\circ - 0^\circ)(2^\circ - 1^\circ)}$$

$$L_2(1.2^\circ) = 0.0175 \cdot 0.96 + 0.0349 \cdot 0.12 = 0.020988 \approx 0.0210$$

$R_2(x) \le \frac{M_3}{3!}\left|x(x-1^\circ)(x-2^\circ)\right|$，其中 $M_3 = \max\limits_{0^\circ \le x \le 2^\circ} |f'''(x)| = \max\limits_{0^\circ \le x \le 2^\circ} |\cos x| \cdot \left(\frac{\pi}{180}\right)^3$

$\Rightarrow\ M_3 = 5.3\times10^{-6}\ \Rightarrow\ R_2(1.2^\circ) \le \frac{1}{6}\times5.3\times10^{-6}\times\left|1.2^\circ \cdot (1.2^\circ - 1^\circ)\cdot(1.2^\circ - 2^\circ)\right| = 1.7\times10^{-7}$

Lagrange 插值法的缺点

1. 当已知的数据节点数由 n 增加到 $n+1$ 时, Lagrange 插值公式

$$p(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) + \cdots + y_n l_n(x) \triangleq L_n(x)$$

$$= \sum_{i=0}^{n} y_i l_i(x) = \sum_{i=0}^{n} y_i \prod_{k=0,\, k \neq i}^{n} \frac{x - x_k}{x_i - x_k}$$

需要重新计算所有基函数 $l_i(x)$，这会带来大量计算开销

- 有没有一种方法能够有效地复用在过去的数据点上计算得到的基函数，提高计算效率呢？ ⇨ 逐次线性插值、牛顿插值

2. 插值多项式 $L_n(x)$ 的次数越高，逼近 $f(x)$ 的精度并非越好

⇨ 分段多项式插值、样条插值

课后习题

课后习题（10.22 课间将作业交给助教）

1. （教材第 12、13 页）习题 6、12

6. 设 $Y_0=28$，按递推公式

$$Y_n = Y_{n-1} - \frac{1}{100}\sqrt{783} \quad (n=1,2,\cdots)$$

计算到 Y_{100}. 若取 $\sqrt{783}\approx 27.982$（五位有效数字），试问计算 Y_{100} 将有多大误差.

12. 计算 $(\sqrt{2}-1)^6$，取 $\sqrt{2}\approx 1.4$，利用下式计算，哪一个得到的结果最好？

$$\frac{1}{(\sqrt{2}+1)^6}, \quad (3-2\sqrt{2})^3, \quad \frac{1}{(3+2\sqrt{2})^3}, \quad 99-70\sqrt{2}.$$

2. （教材第 43 页）习题 3、5

3. 给出 $f(x)=\ln x$ 的数值表（见表 2.9），用线性插值及二次插值计算 $\ln 0.54$ 的近似值.

表 2.9

x	0.4	0.5	0.6	0.7	0.8
$\ln x$	−0.916 291	−0.693 147	−0.510 826	−0.357 765	−0.223 144

5. 设 $x_k=x_0+kh$，$k=0,1,2,3$，求 $\max\limits_{x_0\leqslant x\leqslant x_3} |l_2(x)|$.