

Interpolation :Lagrange's method and Newton's method

Chien-De Lee

NTNU PHY ,TAIWAN

scientific-computing

<https://goo.gl/UW3tCn>

Interpolation(插值法)

① 定義

由已知的離散數據求取未知數據的方法

簡單的例子 (線性內插法)

$$\begin{aligned}x_1 &= 1, y_1 = 2 \\x_2 &= 2, y_2 = 4 \\x_3 &= 5, y_3 = 10\end{aligned}\tag{1}$$

$$f(x=3) = ??,$$

簡單的例子 (線性內插法)

$$x_1 = 1, y_1 = 2$$

$$x_2 = 2, y_2 = 4$$

$$x_3 = 5, y_3 = 10$$

(1)

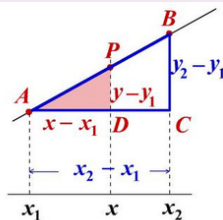
$$f(x = 3) = ??,$$

5. 利用相似三角形邊長
成等比例可知

$$\frac{\overline{PD}}{\overline{BC}} = \frac{\overline{AD}}{\overline{AC}}$$

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

$$y = y_1 + \frac{x - x_1}{x_2 - x_1} (y_2 - y_1)$$



Linear interpolation

```
120 def Linear(data, x):
121     length = len(data)
122     for i in range(1, length):
123         X = data[i, 0]
124         Y = data[i, 1]
125         pX = data[i-1, 0]
126         pY = data[i-1, 1]
127         if x == X:
128             return Y
129         elif X > x:
130             return pY+(Y-pY)*(x-pX)/(X-pX)
131     return pY+(Y-pY)/(X-pX)*(x-pX)
```

Lagrange's interpolation

$$L(x) = \sum_{j=0}^k y_j l_j(x_j) \quad (2)$$

$$l_i(x) = \prod_{0 \leq m \leq k, m \neq j} \frac{x - x_m}{x_j - x_m} \quad (3)$$

Lagrange's interpolation

$$L(x) = \sum_{j=0}^k y_j l_j(x_j) \quad (2)$$

$$l_i(x) = \prod_{0 \leq m < k, m \neq j} \frac{x - x_m}{x_j - x_m} \quad (3)$$

$$\begin{aligned} x_1 &= 1, y_1 = 1 \\ x_2 &= 2, y_2 = 4 \\ x_3 &= 3, y_3 = 9 \end{aligned} \quad (4)$$

$$L(x) = 1 \cdot \frac{x-2}{1-2} \cdot \frac{x-3}{1-3} + 4 \cdot \frac{x-1}{2-1} \cdot \frac{x-3}{2-3} + 9 \cdot \frac{x-1}{3-1} \cdot \frac{x-2}{3-2} = x^2 \quad (5)$$

Lagrange's interpolation

```
70 def Lagrange(data, x):
71     foo = 0
72     for j, xy in enumerate(data):
73         foo += Lagrange_basis(data, x, j)*xy[1]
74     return foo
75
76
77 def Lagrange_basis(data, x, j):
78     foo = 1
79     xj = data[j, 0]
80     for m, xy in enumerate(data):
81         if m == j:
82             continue
83         xm = xy[0]
84         foo *= (x-xm)/(xj-xm)
85     return foo
```

Newton's interpolation

$$N(x) = \sum_{j=0}^k a_j n_j(x_j) \quad (6)$$

$$n_j(x) = \prod_{i=0}^{j-1} (x - x_i) \quad (7)$$

$$a_j = f[y_0, y_1, \dots, y_j] \quad (8)$$

$$f[y_v, \dots, y_{v+j}] = \frac{f[y_{v+1}, \dots, y_{v+j}] - f[y_v, \dots, y_{v+j-1}]}{x_{v+j} - x_v} \quad (9)$$

Newton's interpolation

```
88 def Newton(data, x):
89     foo = 0
90     for j, xy in enumerate(data):
91         foo += Newton_basis(data, x, j)*Divided_difference(
92             data[:j+1]
93         )
94     return foo
95
96
97 def Divided_difference(data):
98     N = len(data)
99     if N == 1:
100         return data[0, 1]
101     else:
102         foo = Divided_difference(data[1:])-Divided_difference(data[:-1])
103         foo /= (data[-1, 0]-data[0, 0])
104         return foo
105
106
107 def Newton_basis(data, x, j):
108     foo = 1
109     for i, xy in enumerate(data[:j]):
110         xi = xy[0]
111         foo *= x-xi
112     return foo
```