Interpolation :Lagrange's method and Newton's method

Chien-De Lee

NTNU PHY ,TAIWAN

scientific-computing

https://goo.gl/UW3tCn

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)

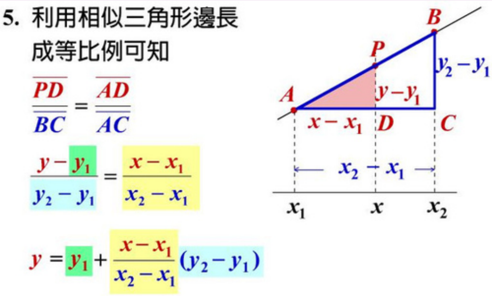
$$\begin{aligned} x_1 &= 1, y_1 = 2 \\ x_2 &= 2, y_2 = 4 \\ x_3 &= 5, y_3 = 10 \end{aligned} \tag{1}$$

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

簡單的例子 (線性內插法)

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

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



Linear interpolation

```python
def Linear(data, x):
    length = len(data)
    for i in range(1, length):
        X = data[i, 0]
        Y = data[i, 1]
        pX = data[i-1, 0]
        pY = data[i-1, 1]
        if x == X:
            return Y
        elif X > x:
            return pY+(Y-pY)*(x-pX)/(X-pX)
    return pY+(Y-pY)/(X-pX)*(x-pX)
```

Lagrange's interpolation

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

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

Lagrange's interpolation

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

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

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

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

Lagrange's interpolation

```
def Lagrange(data, x):
    foo = 0
    for j, xy in enumerate(data):
        foo += Lagrange_basis(data, x, j)*xy[1]
    return foo


def Lagrange_basis(data, x, j):
    foo = 1
    xj = data[j, 0]
    for m, xy in enumerate(data):
        if m == j:
            continue
        xm = xy[0]
        foo *= (x-xm)/(xj-xm)
    return foo
```

Newton's interpolation

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

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

$$a_j = f[y_0, y_1 \ldots\ldots y_j] \tag{8}$$

$$f[y_v \ldots, y_{v+j}] = \frac{f[y_{v+1}, \ldots y_{v+j}] - f[y_v, \ldots y_{v+j-1}]}{x_{v+j} - x_v} \tag{9}$$

Newton's interpolation

```python
def Newton(data, x):
    foo = 0
    for j, xy in enumerate(data):
        foo += Newton_basis(data, x, j)*Divided_difference(
            data[:j+1]
        )
    return foo


def Divided_difference(data):
    N = len(data)
    if N == 1:
        return data[0, 1]
    else:
        foo = Divided_difference(data[1:])-Divided_difference(data[:-1])
        foo /= (data[-1, 0]-data[0, 0])
        return foo


def Newton_basis(data, x, j):
    foo = 1
    for i, xy in enumerate(data[:j]):
        xi = xy[0]
        foo *= x-xi
    return foo
```