シミュレーション工学　ラグランジュ補間

三浦夢生

2020 年 12 月 15 日

1　目的

連立方程式の解を関数補間によって求める方法について学び，理解を深める．
ラグランジュ補間を実装する．

2　用語

今回用いた手法などについて簡単に説明を行う．

2.1　関数補間

ある離散的なデータが与えられているときに，そのデータ点を通る多項式を求めることでその間に存在するであろうデータを予測し，近似することを関数補間という．いくつかの方法があるうち，今回はラグランジュ補間についてまとめる．

2.2　ラグランジュ補間

2 点のデータ点が与えられると一次式が定まる．同様に 3 点が与えられると二次式が定まる．一般に，x 座標がそれぞれ異なる n 個のデータ点が与えられた時 n-1 次式が定まる．

この考えのもと，以下の式を用いて多項式補間をする方法を，ラグランジュ補間という．

$$L(x) = \sum_{i=0}^{n} f(x_i) l_i(x) \tag{1}$$

$$l_i(x) = \prod_{j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{2}$$

一次式 $y = a_1 x + a_0$ に対して $(x_0, y_0), (x_1, y_1)$ を代入し，それぞれの係数について解くと

$$y = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1 \tag{3}$$

となる．同様に二次，三次と拡張し，係数部と全体の式を総和及び総乗の記号を用いることで上記の一般式が導出できる．

3　課題

3.1　課題 1

$(x, y) = (1, 1), (3, 2), (4, 5)$ を通る曲線をラグランジュ補間によって求め，区間 $[-1, 5]$ のグラフを示す．

3.2　課題 2

$f(x) = (1 + 25x^2)^{-1}$ を考える．区間 $[-1, 1]$ において等間隔のデータを 5 点，11 点，21 点生成しそれぞれにラグランジュ補間を適用，そのグラフを示す．

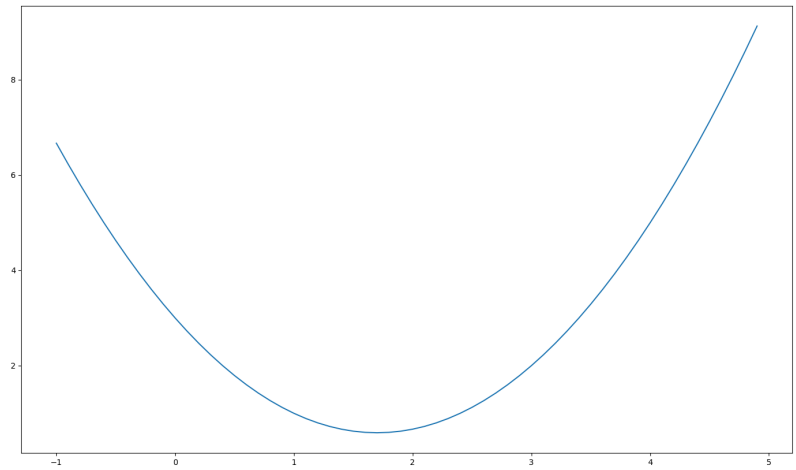

図 1　ラグランジュ補間によって求めた関数の区間 [-1,5] におけるグラフ

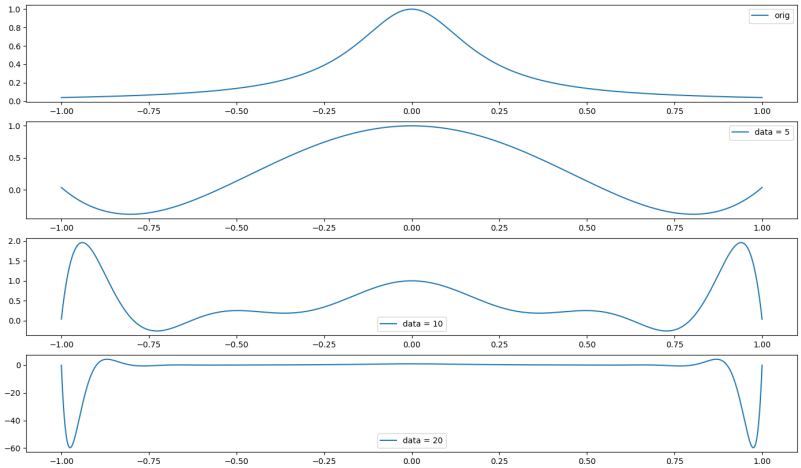

図 2　上からデータ点が 1000 点，5 点，11 点，21 点のときのグラフ

4　付録

今回用いたソースコードを以下に示す.

4.1　課題 1

```
1 import numpy as np
```

```
import matplotlib.pyplot as plt

def lagrange(_x, _dataX, _dataY):
    l = 0
    L = 0
    for i in range(len(_dataX)):
        l = base(i, _x, _dataX) / base(i, _dataX[i], _dataX)
        L += _dataY[i] * l
    return L

def base(_i, _x, _dataX):
    l = 1
    for k in range(len(_dataX)):
        if _i != k:
            l *= _x - _dataX[k]
    return l

def main():
    dataX = np.array([1.0, 3.0, 4.0])
    dataY = np.array([1.0, 2.0, 5.0])

    x = np.arange(-1, 5, 0.1)
    y = lagrange(x, dataX, dataY)

    plt.plot(x, y)
    plt.show()

if __name__ == '__main__':
    main()
```

4.2 課題 2

```
import numpy as np
import matplotlib.pyplot as plt

def func(_x):
    return 1 / (1 + 25 * (_x ** 2))

def lagrange(_x, _dataX, _dataY):
    l = 0
    L = 0
    for i in range(len(_dataX)):
        l = base(i, _x, _dataX) / base(i, _dataX[i], _dataX)
        L += _dataY[i] * l
    return L
```

```
14
15 def base(_i, _x, _dataX):
16     l = 1
17     for k in range(len(_dataX)):
18         if _i != k:
19             l *= _x - _dataX[k]
20     return l
21
22 def main():
23     xa = -1.0
24     xb = 1.0
25
26     step = (xb - xa) / 1000
27     x = np.arange(xa, xb + step, step)
28     y = func(x)
29
30     step = (xb - xa) / 4
31     dataX1 = np.arange(xa, xb + step, step)
32     dataY1 = func(dataX1)
33     L1 = lagrange(x, dataX1, dataY1)
34
35     step = (xb - xa) / 10
36     dataX2 = np.arange(xa, xb + step, step)
37     dataY2 = func(dataX2)
38     L2 = lagrange(x, dataX2, dataY2)
39
40     step = (xb - xa) / 20
41     dataX3 = np.arange(xa, xb + step, step)
42     dataY3 = func(dataX3)
43     L3 = lagrange(x, dataX3, dataY3)
44
45     plt.subplot(411)
46     plt.plot(x, y, label="orig")
47     plt.legend()
48     plt.subplot(412)
49     plt.plot(x, L1, label='data = 5')
50     plt.legend()
51     plt.subplot(413)
52     plt.plot(x, L2, label='data = 10')
53     plt.legend()
54     plt.subplot(414)
55     plt.plot(x, L3, label='data = 20')
56     plt.legend()
57     plt.show()
58
59 if __name__ == '__main__':
```

```
60      main()
```