科学計算研究室 Python ゼミ

～ 5. Lagrange 補間～

2021-03-10 福田　浩

1　原理

1.1　補間とは

内挿（ないそう、英: interpolation）や補間（ほかん）とは、ある既知の数値データ列を基にして、そのデータ列の各区間の範囲内を埋める数値を求めること、またはそのような関数を与えること。またその手法を内挿法（英: interpolation method）や補間法という。対義語は外挿や補外。(Wikipedia)

要はこう言うこと．実験で以下のデータが得られたとする．上司が「`x=2.0，3.0，4.0，5.0` の時の値を報告せよ」と御無体なことを言う．テキトーに答えると説教が始まるのが目に見えるので，補間式を算出して，根拠ある値を報告しなければならない．Lagrange 補間はこんな場合に使う方法．

Table 1: 実験で得られたデータ

x	y
1.3	0.0
1.8	-3.0
2.5	-1.0
3.4	1.0
4.6	0.4
5.5	4.0
6.0	2.0

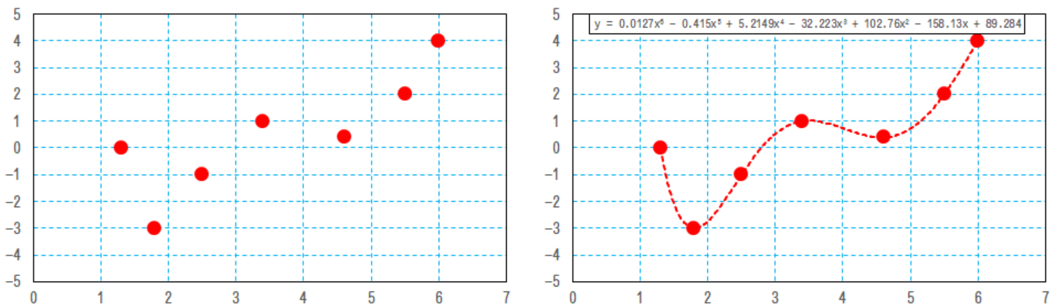

Figure 1: 実験で得られたデータ (左) とその Lagrange 補間.(右)

1.2　考え方

3 点 $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ を通過する曲線は，x の 2 次式で表すことが出来る．その式を以下のように表しても一般性を失わない．

$$y = y_0 z_0 + y_1 z_1 + y_2 z_2 \tag{1}$$

ここで z_0, z_1, z_2 はそれぞれ

$$\begin{cases} z_0 &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} \\ z_1 &= \frac{(x-x_0)\qquad(x-x_2)}{(x_1-x_0)\qquad(x_1-x_2)} \\ z_2 &= \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} \end{cases} \tag{2}$$

である．式 2 は，z_0, z_1, z_2 がいずれも x の 2 次式であり，かつ 3 点 $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ を満たすので，これら 3 点を通過することがわかる．よって，与えられた 3 点を通過する 2 次式を表現していると言える．

以上のことから，任意の n 点 $(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)$ を通過する $n-1$ 次式は

$$y = \sum_{i=0}^{n} y_i z_i \tag{3}$$

$$\begin{cases} z_0 &= \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)} \\ z_1 &= \frac{(x-x_0)\qquad(x-x_2)\cdots(x-x_n)}{(x_1-x_0)\qquad(x_1-x_2)\cdots(x_1-x_n)} \\ z_2 &= \frac{(x-x_0)(x-x_1)\qquad\cdots(x-x_n)}{(x_2-x_0)(x_2-x_1)\qquad\cdots(x_2-x_n)} \\ &\vdots \\ z_n &= \frac{(x-x_0)(x-x_1)(x-x_2)\cdots}{(x_n-x_0)(x_n-x_1)(x_n-x_2)\cdots} \end{cases} \tag{4}$$

$$z_k = \prod_{\substack{i=0 \\ (i \neq k)}}^{n} \frac{x - x_i}{x_k - x_i} \tag{5}$$

で表現することが出来る．

1.3　アルゴリズム

C++ソースコードの解説を参照.

2　課題

Lagrange 補間プログラムを Python で実装せよ

- 通過する点の数と，その座標 (x, y) を標準入力から入力する機能を備えること
- 通過する点の数は，最大 10 点とし，その x 座標は昇順で与えられるとしてよい
- 上記補間の結果通過する x 座標の最小値から最大値までの区間を 400 等分し，その時の x 座標と，y 座標を標準出力に出力する機能を備えること
- 以下の点を通過する点を補間し，x 座標の最小値から最大値までの区間を 400 等分して，グラフで示せ
  - $(-2, 1), (0, -1), (2, 5)$
- 以下の点を通過する点を補間し，x 座標の最小値から最大値までの区間を 400 等分して，グラフで示せ
  - $(1, 0), (1.5, -14.77), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0)$

3　参考：C++ソースコード

解説

- xx は，補間後の x 座標 (倍精度小数点)
- xstep は，補間する x 座標の刻みで，今回は $(x_{max} - x_{min})/400$
- b[i] は，式 4 の分母
  - 17 行目で 1 に初期化して，19 行目で演算
- a[i] は，式 4 の分子
  - 26 行目で 1 に初期化して，30 行目で演算
- 最後に 30 行目で式 3 を計算

```
 1  #include <iostream>
 2  #define N 401
 3  using namespace std;
 4
 5  int main(void){
 6
 7      int n,i,j,k;
 8      double x[10], y[10], z[N], a[10], b[10], xx,xstep;
 9
10      cin >> n;
11      for(i=0;i<n;i++) cin >> x[i] >> y[i];
12      fstep = (x[n-1]-x[0])/(N-1);
13
14      for(i=0;i<N;i++) z[i] = 0.0;
15
16      for(i=0;i<n;i++){
17          b[i]=1.0;
18          for(j=0;j<n;j++){
19              if(i!=j) b[i] *= (x[i]-x[j]);
20          }
21      }
22
23      xx = x[0];
24      for(k=0;k<N;k++){
25          for(i=0;i<n;i++){
26              a[i] = 1.0;
27              for(j=0;j<n;j++){
28                  if(i!=j) a[i] *= (fk-x[j]);
29              }
30              z[k] += y[i]*a[i]/b[i];
31          }
32          cout << xx << " " << z[k] << endl;
33          xx += xstep;
34      }
35      return 0;
36  }
```

4　更なる検討

- 高次の多項式で多項式補間する際に発生する問題として「Runge 現象」がある．
- 以下の補間グラフは，上記ソースコードを用いて $y = \frac{1}{1+25x^2}$ をサンプリングし，補間したものである．
- Runge 現象が見出した課題とその解決策を考察せよ．

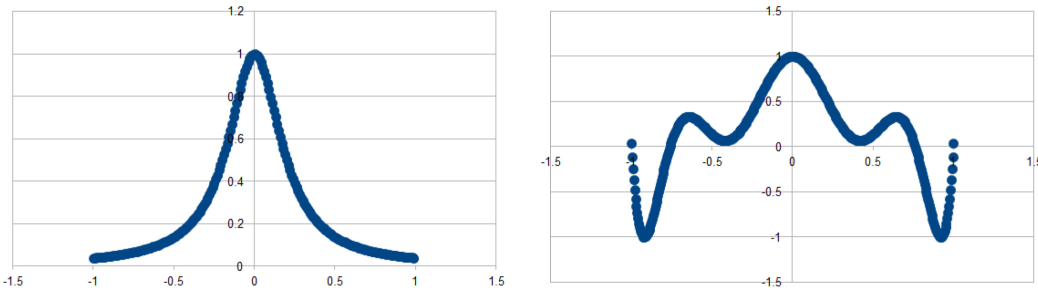

Figure 2: $y = \frac{1}{1+25x^2}$ のグラフ (左) とそれを等間隔 9 点でサンプリングしたのちに Lagrange 補間したグラフ (右)

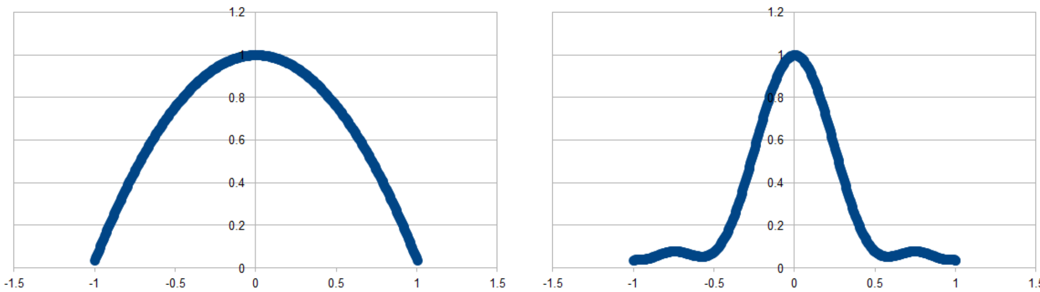

Figure 3: 等間隔 3 点で Lagrange 補間したグラフ (左)，不等間隔 11 点で Lagrange 補間したグラフ (右)