科学計算研究室 Pythonゼミ

～2. Bairstow法～

2021-02-10 福田　浩

1　原理

1.1　前提条件

n 次方程式

$$x^n + a_1 x^{n-1} + a_2 x^{n-2} + a_3 x^{n-3} \cdots + a_{n-2} x^2 + a_{n-1} x + a_n = 0$$

が,

$$(x^{n-2} + b_1 x^{n-3} + b_2 x^{n-4} + \cdots b_{n-4} x^2 + b_{n-3} x + b_{n-2})(x^2 + px + q) + Rx + S = 0$$

のように変形できるとする

1.2　アルゴリズム

上式で，$R = 0, S = 0$ を満たす p, q を見つけることが出来れば，2 次方程式の解の公式から

$$x = \frac{-p \pm \sqrt{p^2 - 4q}}{2}$$

が解となる．同様の求根手順を

$$x^{n-2} + b_1 x^{n-3} + b_2 x^{n-4} + \cdots b_{n-4} x^2 + b_{n-3} x + b_{n-2} = 0$$

に適用すれば，n 次方程式の最高次数は 2 つづつ下がり，ついにはすべての解を求めることが出来る．

一般に，R と S は p, q の関数なので，$R(p, q), S(p, q)$ と書ける．ここで，Newton 法の原理から，p, q の微小変化量 $\Delta p, \Delta q$ の近傍で，以下の式が成り立てば，$R(p, q) = 0, S(p, q) = 0$ が言える．

$$\begin{cases} R(p_i + \Delta p, q_i + \Delta q) & \simeq R(p_i, q_i) + \frac{\partial R(p_i, q_i)}{\partial p_i} \Delta p_i + \frac{\partial R(p_i, q_i)}{\partial q_i} \Delta q_i \\ S(p_i + \Delta p, q_i + \Delta q) & \simeq S(p_i, q_i) + \frac{\partial S(p_i, q_i)}{\partial p_i} \Delta p_i + \frac{\partial S(p_i, q_i)}{\partial q_i} \Delta q_i \end{cases} \tag{1}$$

ここで，$\Delta p, \Delta q$ はそれぞれ,

$$\begin{cases} \Delta p & = p_{i+1} - p_i \\ \Delta q & = q_{i+1} - q_i \end{cases} \tag{2}$$

である．

もとの n 次方程式と，それを変形した方程式の係数を比較する．

$$\begin{cases} a_1 &= b_1 + p_i \\ a_2 &= b_2 + p_i b_1 + q_i \\ a_3 &= b_3 + p_i b_2 + q_i b_1 \\ & \vdots \\ a_k &= b_k + p_i b_{k-1} + q_i b_{k-2} \\ & \vdots \\ a_{n-2} &= b_{n-2} + p_i b_{n-3} + q_i b_{n-4} \\ a_{n-1} &= p_i b_{n-2} + q_i b_{n-3} + R \\ a_n &= q_i b_{n-2} + S \end{cases} \tag{3}$$

これを $b_k = \cdots, R = \cdots, S = \cdots$ の形に整理すると，

$$\begin{cases} b_1 &= a_1 - p_i \\ b_2 &= a_2 - p_i b_1 - q_i \\ b_3 &= a_3 - p_i b_2 - q_i b_1 \\ & \vdots \\ b_k &= a_k - p_i b_{k-1} - q_i b_{k-2} \\ & \vdots \\ b_{n-2} &= a_{n-2} - p_i b_{n-3} - q_i b_{n-4} \\ R &= a_{n-1} - p_i b_{n-2} - q_i b_{n-3} \\ S &= a_n - q_i b_{n-2} \end{cases} \tag{4}$$

ここで，

$$b_k = a_k - p_i b_{k-1} - q_i b_{k-2} \tag{5}$$
$$b_{n-1} = a_{n-1} - p_i b_{n-2} - q_i b_{n-3} \tag{6}$$
$$b_n = a_n - p_i b_{n-1} - q_i b_{n-2} \tag{7}$$

であるから，

$$\begin{cases} R &= & b_{n-1} \\ S &= b_n & + p_i b_{n-1} \end{cases} \tag{8}$$

である．

$$\begin{matrix} \frac{\partial R}{\partial p_i} &= \frac{\partial b_{n-1}}{\partial p_i} & \frac{\partial S}{\partial p_i} &= \frac{\partial b_n}{\partial p_i} + b_{n-1} + p_i \frac{\partial b_{n-1}}{\partial p_i} \\ \frac{\partial R}{\partial q_i} &= \frac{\partial b_n}{\partial q_i} & \frac{\partial S}{\partial q_i} &= \frac{\partial b_n}{\partial q_i} + p_i \frac{\partial b_{n-1}}{\partial q_i} \end{matrix} \tag{9}$$

の偏微分係数を求めるために，以下の漸化式を導入する．

$$\begin{aligned} c_1 &= \frac{\partial b_1}{\partial p_i} = \frac{\partial}{\partial p_i}(a_1 - p_i) = -1 \\ c_2 &= \frac{\partial b_2}{\partial p_i} = \frac{\partial}{\partial p_i}(a_2 - p_i b_1 - q_i) = -b_1 - p_i \frac{\partial b_1}{\partial p_i} \\ &= -b_1 - p_i c_1 \\ c_k &= \frac{\partial b_k}{\partial p_i} = \frac{\partial}{\partial p_i}(a_k - p_i b_{k-1} - q_i b_{k-2}) \\ &= -b_{k-1} - p_i \frac{\partial b_{k-1}}{\partial p_i} - q_i \frac{\partial b_{k-2}}{\partial p_i} \\ &= -b_{k-1} - p_i c_{k-1} - q_i c_{k-2} \end{aligned} \tag{10}$$

$$
\begin{aligned}
d_1 &= \frac{\partial b_1}{\partial q_i} = \frac{\partial}{\partial q_i}(a_1 - p_i) = 0 \\
d_2 &= \frac{\partial b_2}{\partial q_i} = \frac{\partial}{\partial q_i}(a_2 - p_i b_1 - q_i) - 1 \\
d_k &= \frac{\partial b_k}{\partial q_i} = \frac{\partial}{\partial q_i}(a_k - p_i b_{k-1} - q_i b_{k-2}) \\
&= -b_{k-2} - p_i \frac{\partial b_{k-1}}{\partial q_i} - q_i \frac{\partial b_{k-2}}{\partial q_i} \\
&= -b_{k-2} - p_i d_{k-1} - q_i d_{k-2}
\end{aligned} \tag{11}
$$

漸化式 c_k, d_k を用いると,

$$
\begin{aligned}
\frac{\partial R}{\partial p_i} &= c_{n-1} & \frac{\partial S}{\partial p_i} &= c_n + b_{n-1} + p_i c_{n-1} \\
\frac{\partial R}{\partial q_i} &= d_{n-1} & \frac{\partial S}{\partial q_i} &= d_n + p_i d_{n-1}
\end{aligned} \tag{12}
$$

であるから，以下の $\Delta p, \Delta q$ に関する 2 元 1 次の連立方程式を解く問題に帰着できる.

$$
\begin{array}{cccccc}
c_{n-1}\Delta p & & +d_{n-1}\Delta q & & = & -b_{n-1} \\
(c_n + b_{n-1} & +p_i c_{n-1})\Delta p & +(d_n & +p_i d_{n-1})\Delta q & = -b_n & -p_i b_{n-1}
\end{array} \tag{13}
$$

上式 $\times p_i -$ 下式より，連立方程式は以下のように簡単になる.

$$
\begin{array}{llll}
c_{n-1}\Delta p & +d_{n-1}\Delta q & = & -b_{n-1} \\
(c_n + b_{n-1})\Delta p & +d_n \Delta q & = & -b_n
\end{array} \tag{14}
$$

2 元連立方程式の解は，各係数を行列の要素とする行列式によって，以下のように書き下すことが出来る.

$$
\Delta p = \frac{\begin{vmatrix} -b_{n-1} & d_{n-1} \\ -b_n & d_n \end{vmatrix}}{\begin{vmatrix} c_{n-1} & d_{n-1} \\ c_n + b_{n-1} & d_n \end{vmatrix}} \quad \Delta q = \frac{\begin{vmatrix} c_{n-1} & -b_{n-1} \\ c_n + b_{n-1} & -b_n \end{vmatrix}}{\begin{vmatrix} c_{n-1} & d_{n-1} \\ c_n + b_{n-1} & d_n \end{vmatrix}} \tag{15}
$$

この式を用いて，$\Delta p, \Delta q$ を計算し，逐次 p_i, q_i を修正することで，p_i, q_i を収束に導く.

2 課題

Bairstow 法を用いて高次方程式の解を求めるプログラムを Python で書き，動作を検証せよ.

- 条件
  - 高次方程式の，次数と係数を標準入力から取得する機能を備えること
  - 解を標準出力へ出力する機能を備えること
- 課題
  - 作成したプログラムを用いて $x^4 + 4x^3 + 8x^2 + 8x - 5 = 0$ を解け.
  - 作成したプログラムを用いて $x^4 + 4x^3 + 8x^2 + 8x - 6 = 0$ を解け.
  - 作成したプログラムを用いて $x^5 + x^4 + x^3 + x^2 + x + 1 = 0$ を解け.

2.1 参考： C++のソースコード例

高次方程式の次数と係数を標準入力から読み取り，その解を Bairstow 法により求めて，標準出力に出力する．アルゴリズムの参考例であり，エラー処理は実装していない．

```
#include <iostream>
#include <math.h>
#define EPS 0.00001
using namespace std;
/* Quadratic equation solver*/
void quadratic(double p, double q){
    if(p*p-4*q>0){
        cout << -p/2 + sqrt(p*p-4*q)/2 << ", ";
        cout << -p/2 - sqrt(p*p-4*q)/2 << endl;
    }
    else{
        cout << -p/2 << "+" << sqrt(-p*p+4*q)/2 << "i, ";
        cout << -p/2 << "-" << sqrt(-p*p+4*q)/2 << "i" << endl;
    }
}
/* Bairstow method */
void bs(int n, double a[], double *p, double *q){
    double b[n+1], c[n+1], d[n+1];
    double dp, dq;
    *p = rand(); // initialize p as a random value
    *q = rand(); // initialize q as a random value
    while(1){
        b[0] = 1; // !!!!!!!!!! warning !!!!!!!!!! potential BUG
        b[1] = a[1]-(*p)*a[0];
        for(int i=2; i<n+1; i++){
            b[i] =  a[i]   -(*p)*b[i-1] -(*q)*b[i-2];
        }
        c[1] = -1;
        c[2] = -b[1]-(*p)*c[1];
        for(int i=3; i<n+1; i++){
            c[i] = -b[i-1] -(*p)*c[i-1] -(*q)*c[i-2];
        }
        d[1] = 0;
        d[2] = -1;
        for(int i=3; i<n+1; i++){
            d[i] = -b[i-2] -(*p)*d[i-1] -(*q)*d[i-2];
        }
        dp = (-b[n-1]*d[n]+b[n]*d[n-1])       /(c[n-1]*d[n]-d[n-1]*(c[n]+b[n-1]));
        dq = (-b[n]*c[n-1]+b[n-1]*(c[n]+b[n-1]))/(c[n-1]*d[n]-d[n-1]*(c[n]+b[n-1]));
        if((fabs(dp)>EPS) || (fabs(dq)>EPS)){
            *p += dp;
            *q += dq;
        }
        else break;
    }
    for (int i=0; i<n; i++) a[i]=b[i];
}
```

```
int main(void){
    double a[10];
    double p, q;
    int n;
    cin >> n;
    for (int i=0;i<n;i++) cin >> a[i];
    cout << "Eqn: ";
    for (int i=0;i<n-1;i++) cout << a[i] << "*x^" << n-i-1 << " + ";
    cout << a[n-1] << "=0" << endl;

    n--;
    while(1){
        if(n==0) break;
        if(n==1){
            cout << -a[1]/a[0] << endl;
            break;
        }
        if(n==2){
            quadratic(a[1], a[2]);// !!!!!!!!!! warning !!!!!!!!!! potential BUG
            break;
        }
        if(n>2){
            bs(n, a, &p, &q);
            quadratic(p, q);// !!!!!!!!!! warning !!!!!!!!!! potential BUG
            n += -2;
        }
    }
}
```

2.2 更なる検討

- 上記 C++ソースコードにはバグがある．Hack せよ．
- p_i, q_i の初期値の設定の仕方次第では，解は収束しない．解が収束する場合，p_i, q_i と解はどのような関係を満たす必要があるか考察せよ．

A 復習：行列式

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc \tag{16}$$

B 復習：連立方程式の解析的解法

$$\begin{cases} ax + by & = & r \\ cx + dy & = & s \end{cases} \tag{17}$$

$$x = \frac{\begin{vmatrix} r & b \\ s & d \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}} \quad y = \frac{\begin{vmatrix} a & r \\ c & s \end{vmatrix}}{\begin{vmatrix} a & b \\ c & d \end{vmatrix}} \tag{18}$$