数値計算 Class-5 演習

21T2166D 渡辺大樹

2023 月 5 月 10 日

1　演習内容

Class-5 では Class-2 で演習したニュートン法を非線形の連立方程式の解決のために拡張し、実際に連立非線形方程式の解を見つけていく。

非線形の連立方程式を解くためには前回演習した LU 分解による連立方程式の解法を用いたいのでライブラリとしてソースコード my_library_v3.h を読み込んでいく。

ソースコード my_library_v3.h にはベクトルのノルムを求めるための vector_norm1() や LU 分解で連立一次方程式を解く lu_solve() が実装されている。

$A\mathbf{x} = \mathbf{b}$ について解くとき、lu_solve() では前回実装した ludecomp() 関数で行列 A を LU 分解したのち、$L\mathbf{y} = \mathbf{b}$ と $U\mathbf{x} = \mathbf{y}$ についてそれぞれを前進代入、後進代入で解いている。

では以降では実際に非線形連立方程式を解く段階を説明していく。

アルゴリズムは以下ソースコード 1 に実装されている。

ソースコード 1　nonlinear_system.c

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4  #define EPS pow(10.0, -8.0) /* epsilon の設定*/
5  #define KMAX 100 /* 最大反復回数*/
6  #define N 4
7  #include "my_library_v3.h"
8  #include "nonlinear_system.h"
9  int main(void)
10 {
11     double x, y, z;
12     printf("初期値x0, y0, z0 を入力してください----> x0 y0 z0\n");
13     scanf("%lf %lf %lf", &x, &y, &z);
14     int i, k = 0;
15     double xk[N], d[N], J[N][N];
16     double Jx[N]; /* ヤコビ行列の解 */
```

```
17      xk[1] = x;
18      xk[2] = y;
19      xk[3] = z;
20      do
21      {
22          /*右辺ベクトルの作成*/
23          d[1] = -f(xk[1], xk[2], xk[3]);
24          d[2] = -g(xk[1], xk[2], xk[3]);
25          d[3] = -h(xk[1], xk[2], xk[3]);
26          /*ヤコビ行列の作成*/
27          J[1][1] = f_x(xk[1], xk[2], xk[3]);
28          J[1][2] = f_y(xk[1], xk[2], xk[3]);
29          J[1][3] = f_z(xk[1], xk[2], xk[3]);
30          J[2][1] = g_x(xk[1], xk[2], xk[3]);
31          J[2][2] = g_y(xk[1], xk[2], xk[3]);
32          J[2][3] = g_z(xk[1], xk[2], xk[3]);
33          J[3][1] = h_x(xk[1], xk[2], xk[3]);
34          J[3][2] = h_y(xk[1], xk[2], xk[3]);
35          J[3][3] = h_z(xk[1], xk[2], xk[3]);
36          printf("Call lu_solve k= %d\n", k);
37          lu_solve(3, J, d, Jx);
38          for (i = 1; i <= 3; i++)
39          {
40              xk[i] += Jx[i];
41          }
42          k++;
43      } while (vector_norm1(Jx) > EPS && k < KMAX);
44
45      if (k == KMAX)
46      {
47          printf("答えが見つかりませんでした\n");
48      }
49      else
50      {
51          printf("回数 %d で，答えは x=%f, y=%f, z=%f です\n", k + 1, xk[1], xk
                [2], xk[3]);
52      }
53
54      return 0;
55  }
```

このソースコード 1 の中で実装されている拡張されたニュートン法についてコードを交えながら説明する。

まず、以下の一意解を持つ n 元連立非線形方程式

$$\begin{aligned} f_1(x_1, x_2, \cdots, x_n) &= 0 \\ f_2(x_1, x_2, \cdots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \cdots, x_n) &= 0 \end{aligned}$$

を考える。

$$\begin{aligned} x &= (x_1, x_2, \cdots, x_n) \\ \mathbf{f}(x) &= [f_1(x), f_2(x), \cdots, f_n(x)]^t \end{aligned}$$

とすると、

$$\mathbf{f}(x) = 0$$

を求めればよい。

この式は非線形ではあるが一意の解を持つ連立方程式であるから、線形の連立方程式で用いたニュートン法を非線形に拡張し用いる。

線形一次の連立方程式では

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

の数列が解に収束するまで繰り返していた。これは解を α とすると $x = \alpha$ についてのテイラー展開を元に導出した式である。

これを非線形でもこの考え方を応用し、解 α の近くの点 $x = x^{(0)}$ での $f_k(x)$ のテイラー展開をすると

$$f_k(\alpha) \approx f_k(x^{(0)}) + \sum_{i=1}^{n} \frac{\partial f_k}{\partial x_i}(x^{(0)})(\alpha_i - {x^{(0)}}_i)$$

の式が得られる。この式がゼロになる点を解けばよい。

ここでこれを $\mathbf{f}(x)$ にまで拡張させると上記の式は

$$\begin{bmatrix} f_1(x^{(0)}) \\ f_2(x^{(0)}) \\ \vdots \\ f_n(x^{(0)}) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x^{(0)}) & \frac{\partial f_1}{\partial x_2}(x^{(0)}) & \cdots & \frac{\partial f_1}{\partial x_n}(x^{(0)}) \\ \frac{\partial f_2}{\partial x_1}(x^{(0)}) & \frac{\partial f_2}{\partial x_2}(x^{(0)}) & \cdots & \frac{\partial f_2}{\partial x_n}(x^{(0)}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x^{(0)}) & \frac{\partial f_n}{\partial x_2}(x^{(0)}) & \cdots & \frac{\partial f_n}{\partial x_n}(x^{(0)}) \end{bmatrix} \begin{bmatrix} \alpha_1 - {x_1}^{(0)} \\ \alpha_2 - {x_2}^{(0)} \\ \vdots \\ \alpha_n - {x_n}^{(0)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

となる。

ここで $\alpha - x^{(0)}$ の係数行列は $x^{(0)}$ におけるヤコビアンであり、

$$J(x^{(0)}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x^{(0)}) & \frac{\partial f_1}{\partial x_2}(x^{(0)}) & \cdots & \frac{\partial f_1}{\partial x_n}(x^{(0)}) \\ \frac{\partial f_2}{\partial x_1}(x^{(0)}) & \frac{\partial f_2}{\partial x_2}(x^{(0)}) & \cdots & \frac{\partial f_2}{\partial x_n}(x^{(0)}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x^{(0)}) & \frac{\partial f_n}{\partial x_2}(x^{(0)}) & \cdots & \frac{\partial f_n}{\partial x_n}(x^{(0)}) \end{bmatrix}$$

で表す。

$\alpha - x^{(0)}$ を Δx で表すとこの式は

$$\mathbf{f}(x^{(0)}) + J(x^{(0)})\Delta x = 0$$

となる。

この式を Δx について解けば

$$\Delta x = -[J(x^{(0)})]^{-1}\mathbf{f}(x^{(0)})$$

となる。

この Δx は回数を重ねるごとに解 α に近付く数列としてとらえれば k 回目において $\Delta x = x^{k+1} - x^k$ となるので式は

$$x^{k+1} = x^k - [J(x^{(k)})]^{-1}\mathbf{f}(x^{(k)})$$

となる。

ここで

$$d = -[J(x^{(k)})]^{-1}\mathbf{f}(x^{(k)})$$

となる d を、

$$J(x^{(k)})d = -\mathbf{f}(x^{(k)})$$

の連立方程式で解いてあげることで

$$x^{k+1} = x^k + d$$

という数列の収束を計算すればよくなる。これが拡張されたニュートン法の式となる。

ここまでの話より、このニュートン法で必要になるのは連立非線形方程式とその偏微分から導かれるヤコビアン、また適切な初期値となる。プログラム的に言うと反復回数の上限や d の閾値などが必要になる。

ソースコード 1 では 22-35 行目においてあらかじめ設定された非線形方程式から右辺ベクトルとヤコビアンを作成し、これらを先ほど述べた lu_solve() に入れて連立方程式の解を Jx[] に入れる。この連立方程式の解を足し合わせていくことで解に近付いていく。

ソースコード 1 の 43 行目ではこの繰り返しを終了する条件として Jx があらかじめ設定した閾値より小さくなるかこの反復回数が設定した上限を超えるかで設定されている。

反復回数が上限を超えてしまうと求解は失敗といえるが、Jx が閾値より小さくなったといえるなら、その時の Jx の足し合わせが解となり、それを出力するプログラムとなっている。

2 演習結果

以下にこのプログラムを動かしたときの結果を示す。

まず初期値 (x, y, z) を $(1.1, 1.2, 1.3)$ としたときは初期値 x0, y0, z0 を入力してください—-¿ x0 y0 z0 1.1 1.2 1.3 Call lu_solve k= 0 Call lu_solve k= 1 Call lu_solve k= 2 Call lu_solve k= 3

Call lu_solve k= 4 Call lu_solve k= 5 回数 7 で，答えは x=0.972127, y=1.302878, z=0.650619 です