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 1 を読み込んでいく。

ソースコード 1　my_library_v3.h

```
1  /*L1-ノルム*/
2  double vector_norm1(double a[N + 1])
3  {
4      int i = 0;
5      double norm = 0.0;
6      for (i = 1; i <= N; i++)
7      {
8          norm += fabs(a[i]);
9      }
10     return norm;
11 }
12 /*行列の入力*/
13 int input_matrix(double a[N][N])
14 {
15     int n, i, j;
16     char z, zz;
17     while (1)
18     {
19         printf("行列の次数の入力 (1 < n < %d) n = ", N - 1);
20         scanf("%d%c", &n, &zz);
21         if ((n <= 1) || (N - 1 <= n))
22             continue;
23         printf("\n 行列 A の成分を入力します\n\n");
```

```
24          for (i = 1; i <= n; i++)
25          {
26              for (j = 1; j <= n; j++)
27              {
28                  printf("a(%d, %d)=", i, j);
29                  scanf("%lf%c", &a[i][j], &zz);
30              }
31              printf("\n");
32          }
33          printf("正しく入力しましたか？(y/n)");
34          scanf("%c%c", &z, &zz);
35          if (z == 'y')
36              break;
37      }
38      return n;
39  }
40  /*行列の出力*/
41  void print_matrix(double a[N][N], int n)
42  {
43      int i, j;
44      for (i = 1; i <= n; i++)
45      {
46          for (j = 1; j <= n; j++)
47          {
48              printf(" %10.6lf", a[i][j]);
49          }
50          printf("\n");
51      }
52  }
53  /*LU 分解*/
54  int ludecomp(int n, double a[N][N], double l[N][N], double u[N][N])
55  {
56      int i, j, k;
57      double p, q;
58      /*LU 分解とガウス消去*/
59      for (i = 1; i <= n; i++)
60      {
61          p = a[i][i];
62          if (fabs(p) < 1.0e-6)
63          {
64              printf("この行列はLU 分解出来ません. \n");
65              return -1;
```

```
66          }
67          for (j = i; j <= n; j++)
68          {
69              l[j][i] = a[j][i];
70              a[i][j] = a[i][j] / p;
71          }
72          for (k = i + 1; k <= n; k++)
73          {
74              q = a[k][i];
75              for (j = 1; j <= n + 1; j++)
76              {
77                  a[k][j] = a[k][j] - a[i][j] * q;
78              }
79          }
80          for (j = i; j <= n; j++)
81          {
82              u[i][j] = a[i][j];
83          }
84      }
85      return 0;
86  }
87  /*LU 分解による連立一次方程式の解*/
88  int lu_solve(int n, double a[N][N], double b[N], double x[N])
89  {
90      int i, j;
91      double l[N][N] = {0}, u[N][N] = {0};
92      double y[N] = {0};
93      /*LU 分解*/
94      int ret = ludecomp(n, a, l, u);
95      if (ret != 0)
96      {
97          return ret;
98      }
99      /*前進代入*/
100     for (i = 1; i <= n; i++)
101     {
102         double py = b[i];
103         for (j = 1; j < i; j++)
104         {
105             py -= l[i][j] * y[j];
106         }
107         y[i] = py / l[i][i];
```

```
108     }
109     /*後退代入*/
110     for (i = n; i >= 1; i--)
111     {
112         double px = y[i];
113         for (int j = i + 1; j <= n; j++)
114         {
115             px -= u[i][j] * x[j];
116         }
117         x[i] = px / u[i][i];
118     }
119     return 0;
120 }
```

ソースコード 1 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 2 に実装されている。

ソースコード 2　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 2 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 $x = x_\alpha$ での $f_k(x)$ のテイラー展開をすると

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

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