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

```
/*L1-ノルム*/
double vector_norm1(double a[N + 1])
{
    int i = 0;
    double norm = 0.0;
    for (i = 1; i <= N; i++)
    {
        norm += fabs(a[i]);
    }
    return norm;
}
/*行列の入力*/
int input_matrix(double a[N][N])
{
    int n, i, j;
    char z, zz;
    while (1)
    {
        printf("行列の次数の入力 (1 < n < %d) n = ", N - 1);
        scanf("%d%c", &n, &zz);
        if ((n <= 1) || (N - 1 <= n))
            continue;
        printf("\n 行列 A の成分を入力します\n\n");
```

```
        for (i = 1; i <= n; i++)
        {
            for (j = 1; j <= n; j++)
            {
                printf("a(%d, %d)=", i, j);
                scanf("%lf%c", &a[i][j], &zz);
            }
            printf("\n");
        }
        printf("正しく入力しましたか？(y/n)");
        scanf("%c%c", &z, &zz);
        if (z == 'y')
            break;
    }
    return n;
}
/*行列の出力*/
void print_matrix(double a[N][N], int n)
{
    int i, j;
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf(" %10.6lf", a[i][j]);
        }
        printf("\n");
    }
}
/*LU 分解*/
int ludecomp(int n, double a[N][N], double l[N][N], double u[N][N])
{
    int i, j, k;
    double p, q;
    /*LU 分解とガウス消去*/
    for (i = 1; i <= n; i++)
    {
        p = a[i][i];
        if (fabs(p) < 1.0e-6)
        {
            printf("この行列はLU 分解出来ません．\n");
            return -1;
```

```c
        }
        for (j = i; j <= n; j++)
        {
            l[j][i] = a[j][i];
            a[i][j] = a[i][j] / p;
        }
        for (k = i + 1; k <= n; k++)
        {
            q = a[k][i];
            for (j = 1; j <= n + 1; j++)
            {
                a[k][j] = a[k][j] - a[i][j] * q;
            }
        }
        for (j = i; j <= n; j++)
        {
            u[i][j] = a[i][j];
        }
    }
    return 0;
}
/*LU 分解による連立一次方程式の解*/
int lu_solve(int n, double a[N][N], double b[N], double x[N])
{
    int i, j;
    double l[N][N] = {0}, u[N][N] = {0};
    double y[N] = {0};
    /*LU 分解*/
    int ret = ludecomp(n, a, l, u);
    if (ret != 0)
    {
        return ret;
    }
    /*前進代入*/
    for (i = 1; i <= n; i++)
    {
        double py = b[i];
        for (j = 1; j < i; j++)
        {
            py -= l[i][j] * y[j];
        }
        y[i] = py / l[i][i];
```

```
    }
    /*後退代入*/
    for (i = n; i >= 1; i--)
    {
        double px = y[i];
        for (int j = i + 1; j <= n; j++)
        {
            px -= u[i][j] * x[j];
        }
        x[i] = px / u[i][i];
    }
    return 0;
}
```

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

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

```
    {
        /*右辺ベクトルの作成*/
        d[1] = -f(xk[1], xk[2], xk[3]);
        d[2] = -g(xk[1], xk[2], xk[3]);
        d[3] = -h(xk[1], xk[2], xk[3]);
        /*ヤコビ行列の作成*/
        J[1][1] = f_x(xk[1], xk[2], xk[3]);
        J[1][2] = f_y(xk[1], xk[2], xk[3]);
        J[1][3] = f_z(xk[1], xk[2], xk[3]);
        J[2][1] = g_x(xk[1], xk[2], xk[3]);
        J[2][2] = g_y(xk[1], xk[2], xk[3]);
        J[2][3] = g_z(xk[1], xk[2], xk[3]);
        J[3][1] = h_x(xk[1], xk[2], xk[3]);
        J[3][2] = h_y(xk[1], xk[2], xk[3]);
        J[3][3] = h_z(xk[1], xk[2], xk[3]);
        printf("Call lu_solve k= %d\n", k);
        lu_solve(3, J, d, Jx);
        for (i = 1; i <= 3; i++)
        {
            xk[i] += Jx[i];
        }
        k++;
    } while (vector_norm1(Jx) > EPS && k < KMAX);

    if (k == KMAX)
    {
        printf("答えが見つかりませんでした\n");
    }
    else
    {
        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]);
    }

    return 0;
}
```

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

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

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

$$
\begin{bmatrix} f_1(x_\alpha) \\ f_2(x_\alpha) \\ \vdots \\ f_n(x_\alpha) \end{bmatrix}
+
\begin{bmatrix}
\frac{\partial f_1}{\partial x_1}(x_\alpha) & \frac{\partial f_1}{\partial x_2}(x_\alpha) & \cdots & \frac{\partial f_1}{\partial x_n}(x_\alpha) \\
\frac{\partial f_2}{\partial x_1}(x_\alpha) & \frac{\partial f_2}{\partial x_2}(x_\alpha) & \cdots & \frac{\partial f_2}{\partial x_n}(x_\alpha) \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_1}(x_\alpha) & \frac{\partial f_n}{\partial x_2}(x_\alpha) & \cdots & \frac{\partial f_n}{\partial x_n}(x_\alpha)
\end{bmatrix}
\begin{bmatrix} \alpha_1 - x_{\alpha 1} \\ \alpha_2 - x_{\alpha 2} \\ \vdots \\ \alpha_n - x_{\alpha n} \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}
$$

となる。

ここで $\alpha - x_\alpha$ の係数行列は x_α におけるヤコビアンであり、

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

で表す。

　$\alpha - x_\alpha$ を Δx で表すとこの式は

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

となる。

　この式を Δx について解けば

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

となり、