

多次元 Newton 法の例

桂田 祐史

2000年11月22日, 2009年7月9日, 2019年1月15日

1 ターゲット問題

次の非線形 2 点境界値問題の解を差分法で求める。

$$\begin{aligned} -u''(x) &= u(x)^2 \quad (x \in (0, 1)), \\ u(0) &= u(1) = 0. \end{aligned}$$

2 差分法の復習

対応する線形問題は、1 次元 Poisson 方程式の Dirichlet 境界値問題である。

$$\begin{aligned} (1) \quad & -u''(x) = f(x) \quad (x \in (0, 1)), \\ (2) \quad & u(0) = u(1) = 0. \end{aligned}$$

$h = 1/N$, $x_i = ih$, $u_i = u(x_i)$, $f_i = f(x_i)$ ($i = 0, 1, 2, \dots, N$) とおき、 u_i の近似値 U_i を差分法で求めることにする。

$$A = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & & \mathbf{0} \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ \mathbf{0} & & & -1 & 2 \end{pmatrix}, \quad \vec{U} = \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{N-1} \end{pmatrix}, \quad \vec{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix}$$

とおくと、差分方程式は

$$A\vec{U} = \vec{f}$$

になる ($u''(x) \doteq \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$ がポイント)。

3 ターゲット問題の差分近似

これは次の方程式で \vec{U} を定めるのが自然だろう。

$$(3) \quad A\vec{U} = \begin{pmatrix} (U_1)^2 \\ (U_2)^2 \\ \vdots \\ (U_{N-1})^2 \end{pmatrix}.$$

(??) は非線形方程式である。

4 Newton 法

$$\vec{F}(\vec{U}) := A\vec{U} - \begin{pmatrix} (U_1)^2 \\ (U_2)^2 \\ \vdots \\ (U_{N-1})^2 \end{pmatrix}$$

とおくと、(??) は

$$(4) \quad \vec{F}(\vec{U}) = \vec{0}$$

となる。

この場合に、Newton 法とは、初期値 $\vec{U}^{(0)}$ を適当に選んで、後は漸化式

$$(5) \quad \vec{U}^{(k+1)} = \vec{U}^{(k)} - \left(\vec{F}'(\vec{U}^{(k)}) \right)^{-1} \vec{F}(\vec{U}^{(k)})$$

でベクトル列 $\{\vec{U}^{(k)}\}_{k \in \mathbb{N}}$ を定めるというものである。

$$\vec{F}'(\vec{U}) = A - 2 \begin{pmatrix} U_1 & & & \mathbf{0} \\ & U_2 & & \\ & & \ddots & \\ \mathbf{0} & & & U_{N-1} \end{pmatrix} = A - \text{diag}[U_1, \dots, U_{N-1}]$$

であるから、ヤコビ行列 $\vec{F}'(\vec{U})$ は三重対角行列である。

(??) において逆行列が現れるが、逆行列を計算せずに、連立1次方程式を解く形で計算を遂行すべきことに注意しよう。

5 サンプル・プログラムの実行例

(2009年7月9日 グラフィックス・ライブラリを GLSC に乗り換える。)

(2019年1月15日 現象数理学科 Mac ですぐに試せるように書き換えた。)

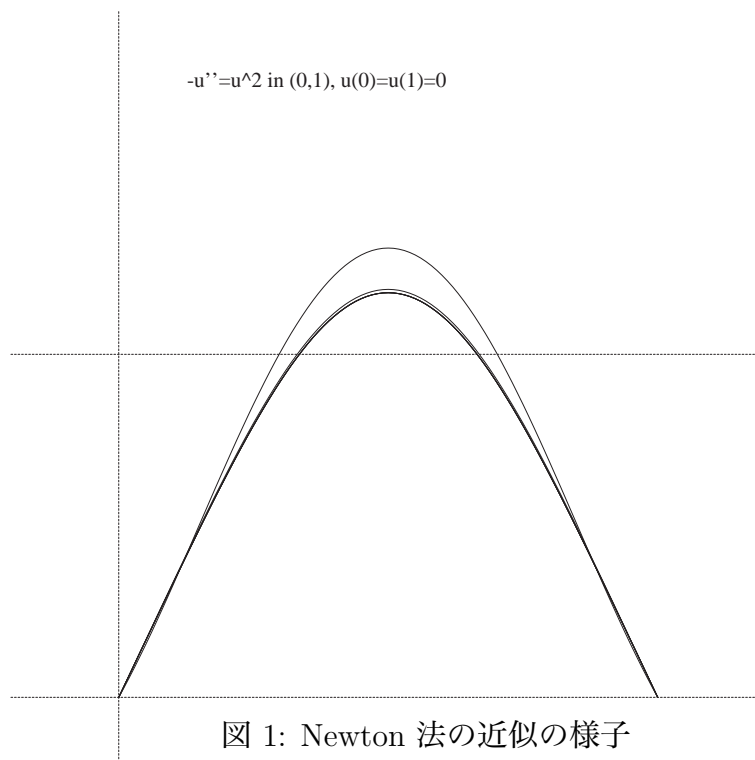
A は計算しておく。 $\vec{F}'(U^{(k)})^{-1}\vec{F}(\vec{U}^{(k)})$ を計算するため、 $A_k := \vec{F}'(\vec{U}^{(k)}) = A - \text{diag}[U_1^{(k)}, \dots, U_{N-1}^{(k)}]$ とおき、連立1次方程式 $A_k \vec{x} = \vec{F}(\vec{U}^{(k)})$ を解く。 A_k が三重対角行列なので、Gauss の消去法を使うと非常に効率的に解ける。

—— 入手してコンパイル (GLSC) ——

```
% curl -O http://nalab.mind.meiji.ac.jp/~mk/program/misc/Newton-glsc.c
% curl -O http://nalab.mind.meiji.ac.jp/~mk/program/linear/trid-lu.c
% curl -O http://nalab.mind.meiji.ac.jp/~mk/program/linear/trid-lu.h
% gclsc Newton-glsc.c trid-lu.c
```

—— 実行 (GLSC) ——

```
% ./Newton-glsc
H (10 位で OK)=10
du=46.0345
du=7.2523
du=0.688334
du=0.00573817
du=4.06013e-07
du=2.50143e-14
min=0, max=11.7958
%
```



(2022/12/10 追記) Newton-glsc.c の GLSC3D バージョン Newton-glsc3d.c を作った。

入手してコンパイル (GLSC3D)

```
curl -O http://nalab.mind.meiji.ac.jp/~mk/program/misc/Newton-glsc3d.c
curl -O http://nalab.mind.meiji.ac.jp/~mk/program/linear/trid-lu.c
curl -O http://nalab.mind.meiji.ac.jp/~mk/program/linear/trid-lu.h
curl -O http://nalab.mind.meiji.ac.jp/~mk/program/misc/ccg2
chmod +x ccg2 ; mv ccg2 ~/bin
ccg2 Newton-glsc3d.c trid-lu.c
```

実行 (GLSC3D)

```
% ./Newton-glsc
H (10 位で OK)=10
du=46.0345
du=7.2523
du=0.688334
du=0.00573817
du=4.06013e-07
du=2.50143e-14
min=0, max=11.7958
%
```

A ソース・プログラム

A.1 Newton-glsc.c

```
/*
 * Newton-glsc.c
 * 非線形 2 点境界値問題
 *  $-u''=u^2$  in  $(0,1)$ ,  $u(0)=u(1)=0$ 
 * を差分法で離散化して得られる非線形方程式を Newton 法で解く。
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef double *vector;

vector new_vector(int n)
{
    return malloc(sizeof(double) * n);
}

double dotprod(int n, vector x, vector y)
{
```

```

int i;
double s;
s=0;
for (i=0; i< n; i++)
    s += x[i] * y[i];
return s;
}

#define G_DOUBLE
#include <glsc.h>
#include "trid-lu.h"

/* 三重対角行列とベクトルのかけ算 */
void mul_mv(int n,
            vector ab,
            vector al, vector ad, vector au,
            vector b)
{
    int i, nm1 = n - 1;
    ab[0] = ad[0] * b[0] + au[0] * b[1];
    for (i = 1; i < nm1; i++)
        ab[i] = al[i] * b[i-1] + ad[i] * b[i] + au[i] * b[i+1];
    ab[nm1] = al[nm1] * b[nm1-1] + ad[nm1] * b[nm1];
}

double norm(int n, vector x)
{
    return sqrt(dotprod(n, x, x));
}

int main(void)
{
    int N, i, k;
    vector al, ad, au, ak1, akd, aku;
    vector U, x;
    double h, h2, du, H;
    double win_width, win_height, w_margin, h_margin;

    N = 100;
    h = 1.0 / N;
    h2 = h * h;
    al = new_vector(N+1); ad = new_vector(N+1); au = new_vector(N+1);
    ak1 = new_vector(N+1); akd = new_vector(N+1); aku = new_vector(N+1);
    U = new_vector(N+1);
    x = new_vector(N+1);

    /* 初期値 */
    printf("H (10 位で OK)="); scanf("%lg", &H);
    for (i = 0; i <= N; i++)
        U[i] = H;
    U[0] = U[N] = 0.0;
    /* A */
    for (i = 1; i < N; i++) {
        al[i] = - 1.0 / h2; ad[i] = 2.0 / h2; au[i] = - 1.0 / h2;
    }
}

```

```

/* GLSC 初期化 */
win_width = 150.0; win_height = 150.0; w_margin = 5.0; h_margin = 5.0;
g_init("NewtonMeta", win_width + 2 * w_margin, win_height + 2 * h_margin);
/* 画面とメタファイルの両方に記録する */
g_device(G_BOTH);
/* 座標系の定義: [-0.2,1.2] × [-2.0, 20.0] という閉領域を表示する */
g_def_scale(0,
            -0.2, 1.2, -2.0, 20.0,
            w_margin, h_margin, win_width, win_height);
/* 線を二種類用意する */
g_def_line(0, G_BLACK, 0, G_LINE_SOLID);
g_def_line(1, G_BLACK, 0, G_LINE_DOTS);
/* 表示するための文字列の属性を定義する */
g_def_text(0, G_BLACK, 3);
/* 定義したものを選択する */
g_sel_scale(0); g_sel_line(0); g_sel_text(0);

/* タイトル表示 */
g_text(40.0, 20.0, "-u''=u^2 in (0,1), u(0)=u(1)=0");
/* 点線 */
g_sel_line(1);
/* 座標軸 */
g_move(-0.2, 0.0); g_plot(1.2, 0.0); g_move(0.0, -2.0); g_plot(0.0, 20.0);
/* 高さ 10 */
g_move(-0.2, 10.0); g_plot(1.2, 10.0);
for (k = 1; k < 100; k++) {
    /* A U^k の計算 */
    mul_mv(N - 1, x+1, al+1, ad+1, au+1, U+1);
    /* A_k=F(U^k) の計算 */
    for (i = 1; i < N; i++)
        x[i] -= U[i] * U[i];
    /* F'(U^k) の計算 */
    for (i = 1; i < N; i++) {
        ak1[i] = al[i]; akd[i] = ad[i] - 2 * U[i]; aku[i] = au[i];
    }
    /* F'(U^k)^{-1} U(U^k) の計算 */
    trid(N-1, ak1+1, akd+1, aku+1, x+1);
    /* */
    du = norm(N-1, x+1);
    printf("du=%g\n", du);
    /* U^{k+1} の計算 */
    for (i = 1; i < N; i++)
        U[i] -= x[i];
    /* */
#ifdef NONE
    for (i = 0; i <= N; i++)
        printf("U[%d]=%g\n", i, U[i]);
#endif
    g_sel_line(0);
    g_move(0.0, U[0]);
    for (i = 1; i <= N; i++)
        g_plot(i * h, U[i]);
    if (du < 1.0e-12)
        break;
}
{

```

```

double min, max;
max = U[0]; min = U[0];
for (i = 1; i <= N; i++) {
    if (U[i] > max)
        max = U[i];
    else if (U[i] < min)
        min = U[i];
}
printf("min=%g, max=%g\n", min, max);
}
g_sleep(-1.0);
g_term();
return 0;
}

```

A.2 trid-lu.c

```

/*
 * trid-lu.c -- 3項方程式を Gauss の消去法で解く
 *
 * http://nalab.mind.meiji.ac.jp/~mk/program/ から入手可能
 * curl -O http://nalab.mind.meiji.ac.jp/~mk/program/linear/trid-lu.c
 * curl -O http://nalab.mind.meiji.ac.jp/~mk/program/linear/trid-lu.h
 *
 */

#include "trid-lu.h"

/* 3項方程式（係数行列が三重対角である連立1次方程式のこと）Ax=b を解く
 *
 * 入力
 *   n: 未知数の個数
 *   al,ad,au: 連立1次方程式の係数行列
 *   (al: 対角線の下側 i.e. 下三角部分 (lower part)
 *    ad: 対角線      i.e. 対角部分 (diagonal part)
 *    au: 対角線の上側 i.e. 上三角部分 (upper part)
 *    つまり
 *
 *      ad[0] au[0]  0  ..... 0
 *      al[1] ad[1] au[1]  0  ..... 0
 *           0  al[2] ad[2] au[2]  0 ..... 0
 *
 *                                     .....
 *                                 al[n-2] ad[n-2] au[n-2]
 *                                 0      al[n-1] ad[n-1]
 *
 *   al[i] = A_{i,i-1}, ad[i] = A_{i,i}, au[i] = A_{i,i+1},
 *   al[0], au[n-1] は意味がない)
 *
 *   b: 連立1次方程式の右辺の既知ベクトル
 *   (添字は 0 から。i.e. b[0],b[1],...,b[n-1] にデータが入っている。)
 *
 * 出力
 *   al,ad,au: 入力した係数行列を LU 分解したもの
 *   b: 連立1次方程式の解
 *
 * 能書き
 *   一度 call すると係数行列を LU 分解したものが返されるので、

```

```

*   以後は同じ係数行列に関する連立1次方程式を解くために、
*   関数 trisol() が使える。
*   注意
*   Gauss の消去法を用いているが、ピボットの選択等はしていな
*   いので、ピボットの選択をしていないので、係数行列が正定値である
*   などの適切な条件がない場合は結果が保証できない。
*/

void trid(int n, double *al, double *ad, double *au, double *b)
{
    trilu(n,al,ad,au);
    trisol(n,al,ad,au,b);
}

/* 三重対角行列の LU 分解 (pivoting なし) */
void trilu(int n, double *al, double *ad, double *au)
{
    int i, nm1 = n - 1;
    /* 前進消去 (forward elimination) */
    for (i = 0; i < nm1; i++) {
        al[i + 1] /= ad[i];
        ad[i + 1] -= au[i] * al[i + 1];
    }
}

/* LU 分解済みの三重対角行列を係数に持つ3項方程式を解く */
void trisol(int n, double *al, double *ad, double *au, double *b)
{
    int i, nm1 = n - 1;
    /* 前進消去 (forward elimination) */
    for (i = 0; i < nm1; i++) b[i + 1] -= b[i] * al[i + 1];
    /* 後退代入 (backward substitution) */
    b[nm1] /= ad[nm1];
    for (i = nm1 - 1; i >= 0; i--) b[i] = (b[i] - au[i] * b[i + 1]) / ad[i];
}

```

A.3 trid-lu.h

```

/* trid-lu.h -- 3項方程式を Gauss の消去法で解く */

void trid(int, double *, double *, double *, double *);
void trilu(int, double *, double *, double *);
void trisol(int, double *, double *, double *, double *);

```