多次元 Newton 法の例

桂田 祐史

2000 年 11 月 22 日, 2009 年 7 月 9 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。

$$-u''(x) = u(x)^2 \qquad (x \in (0,1)),$$
$$u(0) = u(1) = 0.$$

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。

$$-u''(x) = f(x) \quad (x \in (0,1)), \tag{1}$$
$$u(0) = u(1) = 0. \tag{2}$$

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

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

とおくと、差分方程式は

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

になる ($u''(x) \fallingdotseq \dfrac{u(x+h) - 2u(x) + u(x-h)}{h^2}$ 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。

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

(3) 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、(3) は

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

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

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

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

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

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

(5) 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

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

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、$A_k := \vec{F}'(\vec{U}^{(k)}) = A - \mathrm{diag}[U_1^{(k)}, \ldots, U_{N-1}^{(k)}]$ 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。

```
/*
 * 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 <math.h>
#include <matrix.h>
#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()
{
  int N, i, k;
  vector al,ad,au,akl,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);
  akl = 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("NEWTON", 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++) {
      akl[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, akl+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;
}
```

コンパイル&実行

```
oyabun% ccmg Newton-glsc.c trid-lu.c
oyabun% ./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
oyabun%
```

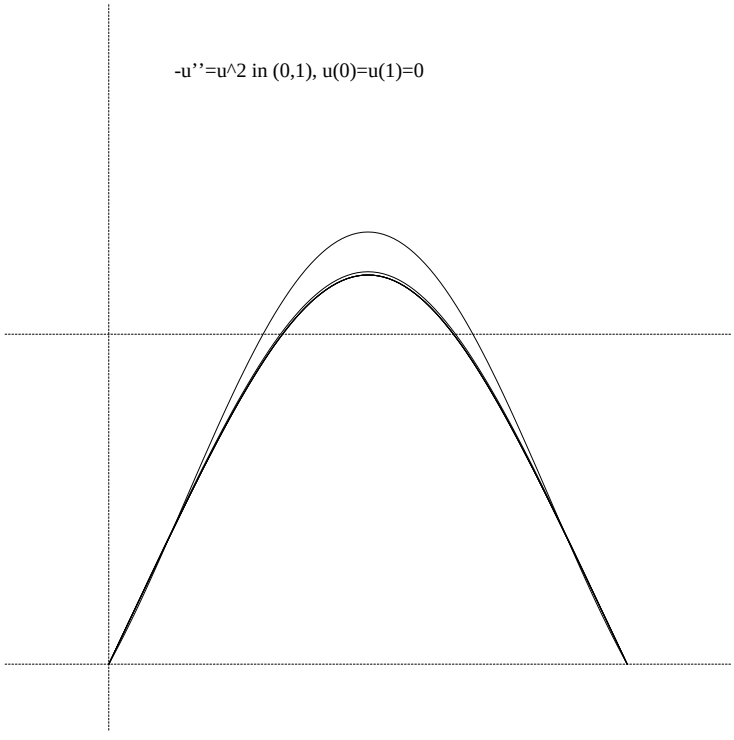

図 1: Newton 法の近似の様子

A trid-lu.c

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

#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、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++) ad[i + 1] -= au[i] * al[i + 1] / ad[i];
}


/* 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] / ad[i];
  /* 後退代入 (backward substitution) */
  b[nm1] /= ad[nm1];
  for (i = n - 2; i >= 0; i--) b[i] = (b[i] - au[i] * b[i + 1]) / ad[i];
}
```