「工学のための数値計算」 5章 章末問題 解答例

□ **1** ヤコビ法の反復行列は

$$H = -D^{-1}(L+U)$$

だから，p.87 の $A = D + L + U$ の関係を利用すると，H の ij 成分 h_{ij} は

$$h_{ij} = \begin{cases} -\dfrac{a_{ij}}{a_{ii}}, & i \neq j \\ 0, & i = j \end{cases}$$

と書くことができる．これより，H の第 i 行の要素の絶対値和は

$$\sum_{j=1}^{n} |h_{ij}| = \frac{1}{|a_{ii}|} \sum_{j=1, j\neq i}^{n} |a_{ij}| \qquad (i = 1, \ldots, n)$$

となり，A が狭義優対角行列であれば

$$\sum_{j=1}^{n} |h_{ij}| < 1 \qquad (i = 1, \ldots, n)$$

すなわち $\|H\| < 1$ が示される．

□ **2, 3, 4** 問題 2,3,4 をまとめて，C のプログラムとして示す．なお，収束判定は p.83 の式 (5.1) を用いた．また，5.4 節の説明では，簡単のために配列は 1 から始まるように記述したが，このプログラムでは C の標準的な仕様に従い，配列はすべて 0 から始まるようにしたことに注意されたい．

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

#define RANDOM_MAX 0x7fffffff /* random 関数の最大値 */
#define N    9                /* 連立一次方程式の次元 */
#define NNE 33                /* 係数行列の非零要素の個数 */
#define EPS 1.0e-8            /* 収束判定定数 */
#define IT_MAX 10000          /* 反復回数の上限 */

/* ------------------------------------------------------ */

int main(void)
{
  /* 関数のプロトタイプ宣言 */
  int gs_method(int, double *, int *, int *,
                double *, double, int);

  /* 係数行列は静的に宣言し，CRS 形式で格納する */
  double val[NNE] =
    {  4, -1, -1,            /* 0 行目 */
```

```
    -1,  4, -1, -1,        /* 1 行目 */
    -1,  4, -1,            /* 2 行目 */
    -1,  4, -1, -1,        /* 3 行目 */
    -1, -1,  4, -1, -1,    /* 4 行目 */
    -1, -1,  4, -1,        /* 5 行目 */
    -1,  4, -1,            /* 6 行目 */
    -1, -1,  4, -1,        /* 7 行目 */
    -1, -1,  4 };          /* 8 行目 */
int col[NNE] =
  {  0, 1, 3,              /* 0 行目 */
     0, 1, 2, 4,           /* 1 行目 */
     1, 2, 5,              /* 2 行目 */
     0, 3, 4, 6,           /* 3 行目 */
     1, 3, 4, 5, 7,        /* 4 行目 */
     2, 4, 5, 8,           /* 5 行目 */
     3, 6, 7,              /* 6 行目 */
     4, 6, 7, 8,           /* 7 行目 */
     5, 7, 8 };            /* 8 行目 */
int ptr[N+1] =
  { 0, 3, 7, 10, 14, 19, 23, 26, 30, 33 };

/* 右辺ベクトル b の設定 */
double b[N] =
  { 1, 1, 2, 0, 0, 1, 0, 0, 1 };

/* 近似解を格納するベクトル
 * 右辺ベクトルを入力としてガウス・ザイデル法を計算する
 * 関数に渡す */
double x[N] =
  { 1, 1, 2, 0, 0, 1, 0, 0, 1 };

/* ローカル変数の宣言 */
double res, tmp;
int i, j, itnum;

if((itnum =
    gs_method(N, val, col, ptr, x, EPS, IT_MAX)) < 0) {
  /* IT_MAX 回の反復では収束しなかった場合 */
  fprintf( stderr ,"NOT convergence\n");
} else {
  /* 収束した場合の結果の表示 */
  /* 計算結果の表示 */
  for(i = 0; i < N; i++) {
    printf("%d  %e\n", i, x[i]);
  }
```

```
    /* 反復回数の表示 */
    printf("Iteration = %d\n", itnum);
    /* 残差ノルム |Ax - b| の計算 */
    for(res = 0.0, i = 0; i < N; i++) {
      /* A x の計算 */
      for(tmp = 0.0, j = ptr[i]; j < ptr[i+1]; j++) {
        tmp += val[j]*x[col[j]];
      }
      if(fabs(tmp - b[i]) > res) {
        res = fabs(tmp - b[i]);
      }
    }
    /* 近似解の残差ノルムの表示 */
    printf("Residual = %e\n", res);
  }
  return 0;
}

/* ------------------------------------------------------ */

int gs_method(int n, double *val, int *col, int *ptr,
      double *b, double eps, int it_max)
     /* ガウス・ザイデル法による連立一次方程式の解法 */
{
  double *x_k, *x_kp1;
  double r, xm, sum, diag;
  int    i, j, it;

  /* 近似解用の配列は動的に領域を確保する */
  x_k   = (double *) malloc((size_t) n * sizeof(double));
  x_kp1 = (double *) malloc((size_t) n * sizeof(double));

  /* 初期ベクトルの設定
   * ここでは，random 関数を使用した
   * [0,1] 区間上の一様乱数を用いる */
  for(i = 0; i < n; i++) {
    x_k[i] = (double) random() / RANDOM_MAX;
  }

  /* ガウス・ザイデル法 */
  for(it = 1, r = eps + 100.0, xm = 1.0; /* 各種パラメータの設定 */
      r > eps*xm && it <= it_max; /* 収束判定と反復回数の上限のチェック */
      it++) {
```

```
    /* ガウス・ザイデル法の一反復 */
    for(i = 0; i < n; i++) {
      for(sum = 0.0, j = ptr[i]; j < ptr[i+1]; j++) {
        if(col[j] < i) {
          sum += val[j] * x_kp1[col[j]];
        } else if(col[j] > i) {
          sum += val[j] * x_k[col[j]];
        } else {
          diag = val[j];
        }
      }
      x_kp1[i] = (b[i] - sum) / diag;
    }
    /* 収束判定のために，一反復前の近似解との
     * 最大値ノルムを求める */
    for(r = xm = 0.0, i = 0; i < n; i++) {
      if(fabs(x_kp1[i] - x_k[i]) > r) {
        r = fabs(x_kp1[i] - x_k[i]);
      }
      /* x_k の最大値ノルムを求める */
      if(fabs(x_k[i]) > xm) {
        xm = fabs(x_k[i]);
      }
      /* 近似解ベクトルの更新 */
      x_k[i] = x_kp1[i];
    }
  }

  /* b に近似解を代入する */
  for(i = 0; i < n; i++) {
    b[i] = x_k[i];
  }

  /* 配列の領域の解放 */
  free(x_k);
  free(x_kp1);

  if(it < it_max) {
    /* 反復回数の上限以下ならば正常終了 */
    return it;
  } else {
    /* it_max 回で収束しなかった場合は -1 を返す */
    return -1;
  }
}
```

プログラム実行例

```
0  5.000000e-01
1  7.142857e-01
2  8.571429e-01
3  2.857143e-01
4  5.000000e-01
5  7.142857e-01
6  1.428571e-01
7  2.857143e-01
8  5.000000e-01
Iteration = 21
Residual = 1.107929e-08
```