連続系アルゴリズム
レポート課題8

連絡先：mail@myuuuuun.com

平成 27 年 12 月 14 日

レポート内で使用したプログラムは GitHub にアップロードしています。 `https://github.com/myuuuuun/various/tree/master/ContinuousAlgorithm/HW8/`

問題 1

$||A||_2 = \sqrt{max\ \lambda(A^{\mathrm{T}}A)}$ を示せ

定義から、

$$||A||_2 = \sup_{x \neq \mathbf{0}} \frac{||Ax||_2}{||x||_2}$$

であり、

$$\begin{aligned}\frac{||Ax||_2}{||x||_2} &= \frac{\sqrt{x^{\mathrm{T}}A^{\mathrm{T}}Ax}}{\sqrt{x^{\mathrm{T}}x}} \\ &= \frac{\sqrt{x^{\mathrm{T}}\lambda x}}{\sqrt{x^{\mathrm{T}}x}} \\ &= \sqrt{\lambda}\end{aligned}$$

※ただし、λ は $A^{\mathrm{T}}A$ の任意の固有値で、

$$\begin{aligned}\lambda &= \left(\frac{||Ax||_2}{||x||_2}\right)^2 \\ &\geq 0\end{aligned}$$

となるので、

$$\begin{aligned}||A||_2 &= \sup_{x \neq \mathbf{0}} \frac{||Ax||_2}{||x||_2} \\ &= \sqrt{max\ \lambda}\end{aligned}$$

となって、行列の 2 ノルムが $\sqrt{\text{スペクトル半径}}$ であることが示された。

問題 2

A を $n \times n$ の狭義対角優位行列（ならばスペクトル半径は 1 未満で反復法は収束する）として、$Ax = b$ を Jacobi 法で解け

$$A = \begin{pmatrix} 6 & 1 & 2 \\ 1 & 5 & 3 \\ 2 & 3 & 7 \end{pmatrix}, \qquad x = \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}, \qquad b = \begin{pmatrix} 10 \\ 0 \\ 17 \end{pmatrix}$$

とすれば、$Ax = b$ が成り立つ。この A, b から、Jacobi 法を用いて x の近似解を求める。

ソースコード: hw8-1.cpp、重要部のみ転記

```
#include <iostream>
#include <array>
#include <cmath>
#include <limits>
#include <iomanip>
using namespace std;

template <size_t row, size_t col>
  using array_2d = std::array<std::array<double, col>, row>;

// matrix class
template <size_t r_num, size_t c_num> class matrix.... （略）

// jacobi method
// coef_mat * x = b_v を解く
template <size_t size>
const matrix<size, 1> jacobi(
  const matrix<size, size> &coef_mat,
  const matrix<size, 1> &b_v,
  const matrix<size, 1> &initial_v
){
  int loop_max = 10000;
  int loop = 0;
  int col = size, row = size;
  matrix<size, size> inversed_diag;
  matrix<size, 1> old_solution, new_solution, residual;

  old_solution = initial_v;

  // 対角行列の逆行列を求める
  for(int i=0; i<size; i++){
    inversed_diag.mat[i][i] = 1.0 / coef_mat.mat[i][i];
  }

  // 反復
  // x_new = D^-1 * (b - (E+F) * x_old), E+F:=A-D
  bool flag = false; // ループ終了 flag
  double row_sum;
  while(loop < loop_max){
    for(int i=0; i<size; i++){
      row_sum = 0;
      for(int j=0; j<size; j++){
        if(i!=j) row_sum += coef_mat.mat[i][j] * old_solution[j][0];
      }
      new_solution.mat[i][0]
        = inversed_diag.mat[i][i] * (b_v.mat[i][0] - row_sum);
```

```
    }

    // 終了判定
    for(int i=0; i<size; i++){
      row_sum = 0;
      for(int j=0; j<size; j++){
        row_sum += coef_mat.mat[i][j] * new_solution[j][0];
      }
      residual.mat[i][0] = b_v.mat[i][0] - row_sum;
    }

    // 残差を出力
    print(residual);

    for(int i=0; i<size; i++){
      if(abs(residual[i][0]) > EPS) break;
      if(i==size-1) flag = true;
    }

    if(flag) break;
    old_solution = new_solution;
    loop++;
  }
  // 反復回数を出力
  // cout << loop+1 << endl;

  return new_solution;
}


int main(){
  std::array<double, 3> x_v={1, -2, 3}, initial_v={0, 0, 0};
  array_2d<3, 3> A_base={{{6, 1, 2}, {1, 5, 3}, {2, 3, 7}}};
  matrix<3, 3> A(A_base);
  matrix<3, 1> x(x_v), initial(initial_v), jacobi_solution, b;

  b = A*x;
  jacobi_solution = jacobi(A, b, initial);

  cout << 元の掛け算": " << endl;
  cout << "A=" << endl;
  print(A);
  cout << "x=" << endl;
  print(x);
  cout << "b=" << endl;
  print(b);

  cout << "jacobi 法の近似解 x: " << endl;
  print(jacobi_solution);

  return 0;
}
```

出力結果:

```
元の掛け算:
A=
         6          1          2
         1          5          3
         2          3          7

x=
         1
        -2
         3

b=
        10
         0
        17

jacobi法の近似解 x:
         1
        -2
         3
```

図 1: Jacobi 法の計算結果

となって、元の x と近似解は一致した。反復回数は 80 回であった。残差を plot すると、

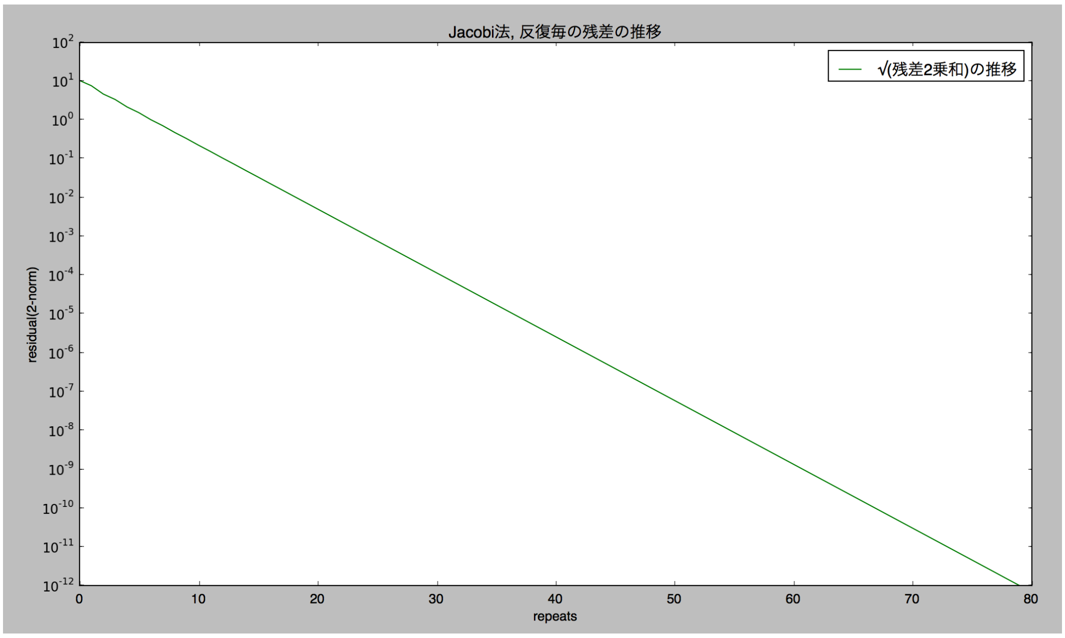


図 2: 反復毎の残差の推移

となった（グラフ出力のソースコード: 8-2.py)。

オプション1

共役勾配法について調査せよ

共役勾配法は、係数行列 A に対して、線形方程式 $Ax = b$ の解を求める問題を、正定値2次形式

$$S(\xi) = \frac{1}{2} < x - \xi, A(x - \xi) >= \frac{1}{2}(x - \xi)^T A(x - \xi)$$

の最小化問題に帰着させて解く、逐次最小化法の1つ（$< A, B >$ はAとBの内積）。

2次形式の関数Sは、A が正定値対称行列であれば狭義凸関数となり、局所最適解が大域的最適解に一致する。A の正定値性から、Sが最小値をとるのは $x = \xi$ の時で、このとき $S = 0$ となる。共役勾配法（を含む逐次最小化法）はこのSの性質を用いて、Sを順次小さくするように近似解を修正することで、真の解に近づけていく（山登り法の発想）。

したがって、共役勾配法で近似解が真の解に収束するためには、係数行列 A が正定値対称である必要がある。

具体的には、適当に探索方向ベクトルを決め、初期ベクトルから探索方向に進んだ時にもっともSが小さくなるような点を選んで、その解を次のステップの初期ベクトルとする、というステップを繰り返すことで、解を真値に近づける。

共役勾配法の特徴は、探索方向ベクトルを1次独立な方向に次々と取ることによって、誤差の影響を考えなければ最大 $n :=$ (A の行数) 反復の内に解が収束することを保証した点にある。探索方向ベクトルの生成には Gram-Schmidt の正規直交化を用いる。

更に共役勾配法は、係数行列 A の固有値に重複がある場合に、その重複分だけ短い最大反復回数で収束することが証明できる。そこで係数行列 A をなんらかの正則行列によって相似変換した行列を用いて、元の方程式と同値な方程式に対し共役勾配法を適用することで、より高速に大規模な線形方程式を解くことが出来る。これを前処理付き共役勾配法という。

アルゴリズム

r を残差ベクトル、p を探索方向ベクトルとする。

1. 適当な初期ベクトル x^0 を選んでつぎの計算を行う

$$r^0 = b - Ax^0$$
$$p^0 = r^0$$

2. k=0, 1, 2 ... について次の計算を繰り返す

$$\alpha_k = \frac{< r^k, r^k >}{< p^k, Ap^k >}$$
$$x^{k+1} = x^k + \alpha_k p^k$$
$$r^{k+1} = r^k - \alpha_k Ap^k$$
$$||r^{k+1}|| \leq \epsilon ||b|| \text{ なら終了}$$
$$\beta_k = \frac{< r^{k+1}, r^{k+1} >}{< r^k, r^k >}$$
$$p^{k+1} = r^{k+1} + \beta_k p^k$$

参考: 森『数値解析』, 杉原・室田『線形計算の数理』

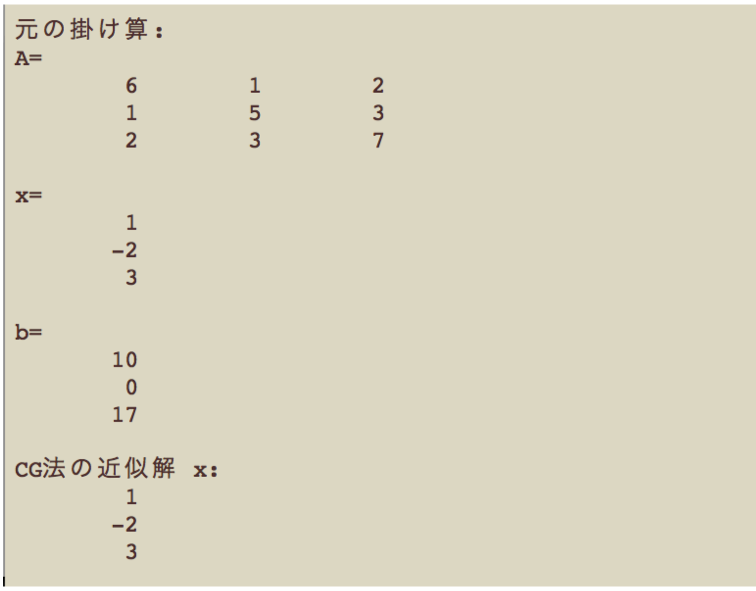
ソースコード: hw8-3.cpp、重要部のみ転記

```
// conjugate gradient method
// coef_mat * x = b_v を解く
template <size_t size>
const matrix<size, 1> conjugate_gradient(
  const matrix<size, size> &coef_mat,
  const matrix<size, 1> &b_v,
  const matrix<size, 1> &initial_v
){
  int loop_max = 10000;
  int loop = 0;
  int col = size, row = size;
  matrix<size, size> inversed_diag;
  matrix<size, 1> solution, old_residual, new_residual, direction;

  // 初期残差
  double row_sum;
  for(int i=0; i<size; i++){
    for(int j=0; j<size; j++){
      row_sum = coef_mat.mat[i][j] * initial_v.mat[j][0];
    }
    old_residual.mat[i][0] = b_v.mat[i][0] - row_sum;
  }

  // 初期ベクトル
  solution = initial_v;

  // 初期探索方向
  direction = old_residual;

  // 反復
  bool flag = false; ループ終了// flag
  double alpha, beta;
  while(loop < loop_max){
    // テンプレートの特殊化タイミングの話を理解したら行列演算に直すetc
    double u=0, l=0;
    for(int i=0; i<size; i++){
      u += old_residual.mat[i][0] * old_residual.mat[i][0];
```

```
      row_sum = 0;
      for(int j=0; j<size; j++){
        row_sum += coef_mat.mat[i][j] * direction.mat[j][0];
      }
      l += direction.mat[i][0] * row_sum;
    }
    alpha = u / l;

    for(int i=0; i<size; i++){
      solution.mat[i][0] += alpha * direction[i][0];
    }

    for(int i=0; i<size; i++){
      row_sum = 0;
      for(int j=0; j<size; j++){
        row_sum += coef_mat.mat[i][j] * direction.mat[j][0];
      }
      new_residual.mat[i][0] = old_residual.mat[i][0] - alpha * row_sum;
    }

    // 終了判定
    for(int i=0; i<size; i++){
      row_sum = 0;
      for(int j=0; j<size; j++){
        row_sum += coef_mat.mat[i][j] * solution.mat[j][0];
      }
      new_residual.mat[i][0] = b_v.mat[i][0] - row_sum;
    }
    // 残差を出力
    print(new_residual);
    for(int i=0; i<size; i++){
      if(abs(new_residual[i][0]) > EPS) break;
      if(i==size-1) flag = true;
    }
    if(flag) break;

    // 次の方向ベクトルを設定
    u=0, l=0;
    for(int i=0; i<size; i++){
      u += new_residual.mat[i][0] * new_residual.mat[i][0];
      l += old_residual.mat[i][0] * old_residual.mat[i][0];
    }
    beta = u / l;

    for(int i=0; i<size; i++){
      direction.mat[i][0] *=  beta;
      direction.mat[i][0] += new_residual.mat[i][0];
    }

    old_residual = new_residual;
    loop++;
  }

  return solution;
}


int main(){
```

```
  std::array<double, 3> x_v={1, -2, 3}, initial_v={0, 0, 0};
  array_2d<3, 3> A_base={{{6, 1, 2}, {1, 5, 3}, {2, 3, 7}}};
  matrix<3, 3> A(A_base);
  matrix<3, 1> x(x_v), initial(initial_v), cg_solution, b;

  b = A*x;
  cg_solution = conjugate_gradient(A, b, initial);

  cout << 元の掛け算": " << endl;
  cout << "A=" << endl;
  print(A);
  cout << "x=" << endl;
  print(x);
  cout << "b=" << endl;
  print(b);

  cout << "CG 法の近似解 x: " << endl;
  print(cg_solution);

  return 0;
}
```

これを用いて問題 2 と同じ問題を解くと、
出力結果:



```
元の掛け算:
A=
        6        1        2
        1        5        3
        2        3        7

x=
        1
       -2
        3

b=
       10
        0
       17

CG法の近似解 x:
        1
       -2
        3
```

図 3: 共役勾配法の計算結果

となって、元の x と近似解は一致した。反復回数は理論通り 3 回であった。

残差:
1 回目: $[-1.0705419, -7.1840751, 0.62973055]$, 2 回目: $[-1.2082795, 0.24235499, 0.71075262]$, 3 回目: $[5.3290705e^{-15}, 3.5527137e^{-15}, 7.1054274e^{-15}]$