応用数理実験

連立 1 次方程式に対する CG 法の数値実験

桂田 祐史

2003 年 5 月 8 日

1 連絡事項

パソコンの設定については、

```
http://www.math.meiji.ac.jp/mmo/mathcomp/
```

2 講義の内容の復習

代表的な反復法として、共役勾配法 (**conjugate gradient method**, 略して **CG 法**) を取り上げる[1]。これは A が正定値対称行列である場合に利用可能な方法である[2]。

CG 法のアルゴリズム:

初期ベクトル $\mathbf{x_0}$ をとる ; 目標とする相対残差 ε を決める ;
$\mathbf{r_0} := \mathbf{b} - A\mathbf{x_0}$; $\mathbf{p_0} := \mathbf{r_0}$;
for $k := 0, 1, \cdots$ **until** $\|\mathbf{r_k}\| \le \varepsilon\|\mathbf{b}\|$ **do**
begin

$$\alpha_k := \frac{(\mathbf{r_k}, \mathbf{p_k})}{(\mathbf{p_k}, A\mathbf{p_k})};$$

$\mathbf{x_{k+1}} := \mathbf{x_k} + \alpha_k \mathbf{p_k}$;
$\mathbf{r_{k+1}} := \mathbf{r_k} - \alpha_k A\mathbf{p_k}$;

$$\beta_k := -\frac{(\mathbf{r_{k+1}}, A\mathbf{p_k})}{(\mathbf{p_k}, A\mathbf{p_k})};$$

$\mathbf{p_{k+1}} := \mathbf{r_{k+1}} + \beta_k \mathbf{p_k}$;
end

ここに掲げた「古典的な」CG 法では、実際に解ける問題がかなり限定される。近年は前処理 (**preconditioning**) と呼ばれるテクニックを併用した前処理つき共役勾配法 (**preconditioned CG method**, 略して **PCG 法**) が広く使われている。前処理でやっていることは、与えられた問題を、固有値が密集している係数行列を持つ問題に変換することであると言えるが、具体的にどのような前処理を採用すべきかは、係数行列 A の性質に強く依存する。

---

[1] CG 法では丸め誤差がなければ有限 (未知数の個数 N 以下) 回の反復で真の解が得られるので、直接法的な性格も持っている。しかし実際上扱う問題では、N よりかなり小さな回数の反復で、十分な精度の解が得られる (またそれ以上反復しても精度は改善されない) ので、反復法的な性格の方が強い、とする意見がやや優勢である。

[2] 係数行列 A が正定値でない場合にも適用できる同様の方法が色々と開発されているが、決定版と言えるものは見つかっていない (そんなものはないのかもしれない)。

3　本日すべきこと

「古典的」CG 法で連立 1 次方程式を解くプログラムを使って簡単な実験をしてもらう。

(1) サンプル・プログラム `cg.C` を入手して、コンパイル&実行してみる (プログラムの内容について、ざっと解説する)。実行を開始すると、`c1,c2,N=` と尋ねてくるが、`c1`, `c2` は行列 A を

$$A = \begin{pmatrix} c_1 A' & O \\ O & c_2 A' \end{pmatrix}$$

と表したときの c_1, c_2 を指す。例えば、`1 1 100` と入力すると、以下の (2-i) を解くことができ、`1 10 100` と入力すると、(2-ii) を $c = 10$ で解くことができる。

(2) 対角線上の成分の値が 4, その両隣りの成分が 1 である 50 次の三重対角行列を A' とする:

$$A' = \begin{pmatrix} 4 & 1 & & & 0 \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ 0 & & & 1 & 4 \end{pmatrix}.$$

さらに適当な解ベクトル $x^* \in \mathbf{R}^{100}$ を選んで、以下の実験をして、その結果を分析する。

(2-i)

$$A^{(1)} \equiv \begin{pmatrix} A' & O \\ O & A' \end{pmatrix}, \qquad b^{(1)} \equiv A^{(1)} x^*$$

として方程式 $A^{(1)}x = b^{(1)}$ を解け。

(2-ii) (2-i) の方程式のうち、最初の 50 行を c 倍 ($c = 10$, あるいは 100) した問題、すなわち

$$A^{(2)} \equiv \begin{pmatrix} cA' & O \\ O & A' \end{pmatrix}, \qquad b^{(2)} \equiv A^{(2)} x^*$$

として作った方程式 $A^{(2)}x = b^{(2)}$ を解け。

(2-iii) (2-i) の方程式のうち、すべての行を c 倍 ($c = 10, 100$) した問題、すなわち

$$A^{(3)} \equiv \begin{pmatrix} cA' & O \\ O & cA' \end{pmatrix}, \qquad b^{(3)} \equiv A^{(3)} x^*$$

として作った方程式 $A^{(3)}x = b^{(3)}$ を解け。

解説

実は上記の実験 (2) は、最も原始的な前処理の一つであるスケーリング[3]の効用を納得するためのものである。もともと (2-i) のような方程式は固有値があまり散らばっていないため、

[3] これが有効な場合があることは古く (1970 年代以前) から分かっていた。

CG 法にとっては比較的扱い易い問題であると言えるが[4]、(2-ii) のような問題に変換してしまうと、固有値の存在範囲が広がってしまい、収束が悪くなる (反復回数が増える) はずである。一方で (2-iii) になると、固有値の広がり具合はまた (2-i) と同等なものに戻り、反復回数も同程度になると予想される。

4 MATLAB, Octave でやってみる？

- http://www.math.meiji.ac.jp/~mk/lecture/ouyousuurijikken/lesson1/ 『応用数理実験課題 (1) MATLAB 入門』
- ttp://www.math.meiji.ac.jp/~mk/lecture/ouyousuurijikken/private-matlab-notebook/ 『MATLAB 手習い』
- ttp://www.math.meiji.ac.jp/~mk/lecture/ouyousuurijikken/intro-matlab-poisson/ 『応用数理実験課題 (X) Poisson 方程式, -$\triangle$ の固有値問題 』

A CG 法の計算手順について補足

以下、A は N 次の正値対称行列, $b \in \mathbf{R}^N$ で、ともに与えられているものとする。

記号の定義

x_k を $x^* \stackrel{\text{def.}}{=} A^{-1}b$ の近似と考えるとき、

$$r_k \stackrel{\text{def.}}{=} b - Ax_k = A(x^* - x_k)$$

を残差 (**residual**) と呼ぶ。

以下現れるベクトル p_i, r_i については、

$$\text{(直交性)} \quad (r_i, r_j) = 0 \quad (i \neq j),$$

$$\text{(共役直交性)} \quad \langle p_i, p_j \rangle = 0 \quad (i \neq j)$$

が成り立つ。

CG 法にはいくつかのバージョンがあるが、大きく

(I) 2 項漸化式 (Hestenes 版に代表される)

(II) 3 項漸化式 (Rutishauser 版に代表される)

の二つに分類される。この講義では 2 項漸化式版を取り上げる。

細かい工夫だが、次のアルゴリズムは演算回数が少ない。

[4] とはいえ、このような三項方程式で Gauss の消去法と競争したら勝てないであろう。あくまで実験用の問題である。

CG 法のアルゴリズム (高橋秀俊版)

初期ベクトル $\mathbf{x_0}$ をとる ; 目標とする相対残差 ε を決める ;
$\boldsymbol{r}_0 := \boldsymbol{b} - A\mathbf{x_0}$; $\beta_0 := 1/(\boldsymbol{r}_0, \boldsymbol{r}_0)$; $\mathbf{p_0} := \beta_0 \boldsymbol{r}_0$;
for $k := 0, 1, \cdots$ **until** $\|\boldsymbol{r}_k\| \leq \varepsilon \|\boldsymbol{b}\|$ **do**
begin

$$\alpha_k := \frac{1}{(\mathbf{p_k}, A\mathbf{p_k})};$$

$$\mathbf{x_{k+1}} := \mathbf{x_k} + \alpha_k \mathbf{p_k};$$

$$\boldsymbol{r}_{k+1} := \boldsymbol{r}_k - \alpha_k A\mathbf{p_k};$$

$$\beta_k := \frac{1}{(\boldsymbol{r}_{k+1}, \boldsymbol{r}_{k+1})};$$

$$\mathbf{p_{k+1}} := \mathbf{p_k} + \beta_k \boldsymbol{r}_{k+1};$$

end

B 余裕があれば以下のようなことにチャレンジして欲しい

以下の中から出来ることを二つ三つやってみよう。

- 叩き台プログラム `cg.c` を高橋版のアルゴリズムを用いるように書き換えてみよ (難易度小 ~ 中)。
- 誤差、残差がそれぞれどのように減少するか観察せよ。単調減少するか？(難易度小)
- (未知数の個数 N が小さい場合で良いから) 残差ベクトル $\{r_i\}$ の直交性、$\{p_i\}$ の共役直交性を数値計算で確かめよ (難易度中)。
- 係数行列の固有値について調べよ。厳密に求めることも可能であるし、先日紹介した Octave を用いて近似計算することも出来る (難易度中)。
- CG 法の収束の速さについて

$$\frac{\|x^* - x_k\|}{\|x^* - x_0\|} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k, \quad \kappa \stackrel{\text{def.}}{=} \frac{\max\{\lambda; \lambda \in \sigma(A)\}}{\min\{\lambda; \lambda \in \sigma(A)\}}, \quad \sigma(A) = A \text{ の固有値全体}$$

  が成り立つと言われているが、検証せよ (難易度中 ~ 大)。
- 係数行列は三重対角行列なので、行列とベクトルの掛け算の計算量はかなり節約できるはずである。これを遂行せよ (難易度中)。
- 講義で説明した SD 法 (steepest descent method — 探索方向 $\mathbf{p_{k+1}}$ として $\mathbf{r_{k+1}}$ を用いる) のプログラムを作り、CG 法と比較せよ。

C　叩き台プログラムのリスト

以下に cg.C を掲げる。

こうしてコンパイル、実行できる

```
oyabun% g++ -O -o cg -I/usr/local/include cg.C -L/usr/local/lib -lmatrix
oyabun% ./cg
```

```
 1  /*
 2   * cg.C --- 入出力に iostream を使う
 3   *    g++ -O -o cg cg.C  -L/usr/local/lib -lmatrix
 4   */
 5
 6  #include <iostream.h>
 7  #include <cmath>
 8  extern "C" {
 9  #include <matrix.h>
10  }
11
12  // m 行 n 列の零行列を作って返す
13  matrix zeros(int m, int n)
14  {
15    int i, j;
16    matrix tmp;
17    tmp = new_matrix(m, n);
18    for (i = 0; i < m; i++)
19      for (j = 0; j < n; j++)
20        tmp[i][j] = 0.0;
21    return tmp;
22  }
23
24  // m 行 n 列の行列 z を零行列にする
25  void zeros(int m, int n, matrix z)
26  {
27    int i, j;
28    for (i = 0; i < m; i++)
29      for (j = 0; j < n; j++)
30        z[i][j] = 0.0;
31  }
32
33  // n 次元の零ベクトルを作って返す
34  vector zeros(int n)
35  {
36    int i;
37    vector tmp;
38    tmp = new_vector(n);
39    for (i = 0; i < n; i++)
40      tmp[i] = 0.0;
41    return tmp;
42  }
43
44  // n 次元のベクトル z を零ベクトルにする
45  void zeros(int n, vector z)
46  {
47    int i;
48    for (i = 0; i < n; i++)
```

```
49      z[i] = 0.0;
50  }
51
52  // n 次元ベクトルの内容を表示する
53  void print_vector(int n, vector x)
54  {
55    int i;
56    for (i = 0; i < n; i++) {
57      cout << " " << x[i];
58      if (i % 5 == 4)
59        cout << endl;
60    }
61    if (i % 5 != 0)
62      cout << endl;
63  }
64
65  // m 行 n 列の行列の内容を表示する
66  void print_matrix(int m, int n, matrix a)
67  {
68    int i, j;
69    for (i = 0; i < m; i++) {
70      for (j = 0; j < n; j++) {
71        cout << a[i][j];
72        if (j % 5 == 4)
73          cout << endl;
74      }
75      if (j % 5 != 0)
76        cout << endl;
77    }
78  }
79
80  // ベクトル x, y の内積 (x,y) を計算する
81  double dotproduct(int n, vector x, vector y)
82  {
83    int i;
84    double sum;
85    sum = 0;
86    for (i = 0; i < n; i++)
87      sum += x[i] * y[i];
88    return sum;
89  }
90
91  // ベクトル x のノルム ||x|| を計算する
92  double norm(int n, vector x)
93  {
94    return sqrt(dotproduct(n, x, x));
95  }
96
97  // 行列 A, ベクトル b の積 A b を計算して、ベクトル Ab に書き込む
98  void multiply_mv(int m, int n, vector Ab, matrix A, vector b)
99  {
100   int i;
101   for (i = 0; i < m; i++)
102       Ab[i] = dotproduct(n, A[i], b);
103 }
104
105 // ベクトル x をベクトル y にコピーする
106 void copy_vector(int n, vector y, vector x)
```

```
107 {
108   int i;
109   for (i = 0; i < n; i++)
110     y[i] = x[i];
111 }
112
113 // CG 法で n 元連立 1 次方程式 A x=b を解く。
114 // 呼び出すときの x は初期ベクトル，戻ってきたときの x は解となる。
115 // ||残差|| < ε ||b|| となったら反復を停止する。
116 // 反復回数を maxiter に返す。
117 void cg(int n, matrix A, vector b, vector x, double eps, int *maxiter)
118 {
119   int i, k;
120   double eps_b, pAp, alpha, beta;
121   vector r, p, Ap;
122
123   r = new_vector(n);
124   p = new_vector(n);
125   Ap = new_vector(n);
126   /* eps_b := ε ||b|| */
127   eps_b = eps * norm(n, b);
128   /* r := A x */
129   multiply_mv(n, n, r, A, x);
130   /* r := b - A x */
131   for (i = 0; i < n; i++)
132     r[i] = b[i] - r[i];
133   /* p := r */
134   copy_vector(n, p, r);
135   /* 反復 */
136   for (k = 0; k < n; k++) {
137     /* Ap := A * p */
138     multiply_mv(n, n, Ap, A, p);
139     /* pAp := (p, Ap) */
140     pAp = dotproduct(n, p, Ap);
141     /* α = (r,p)/(p,Ap) */
142     alpha = dotproduct(n, r, p) / pAp;
143     /* x, r の更新 */
144     for (i = 0; i < n; i++) {
145       /* x = x + α *p */
146       x[i] += alpha * p[i];
147       /* r = r - α *A*p */
148       r[i] -= alpha * Ap[i];
149     }
150     /* ||r||<ε ||b|| ならば反復を終了 */
151     if (norm(n, r) < eps_b)
152       break;
153     /* β := - (r, Ap) / (p, Ap) */
154     beta = - dotproduct(n, r, Ap) / pAp;
155     /* p := r + β *p */
156     for (i = 0; i < n; i++)
157       p[i] = r[i] + beta * p[i];
158   }
159   if (maxiter != NULL)
160     *maxiter = k;
161 }
162
163 int main()
164 {
```

```
165   int n, N, i, niter;
166   matrix a;
167   vector x, b;
168   double c1, c2;
169   /* 問題のサイズを決める */
170   cout << "c1, c2, N=";
171   cin >> c1 >> c2 >> N;
172   n = N / 2;
173   N = 2 * n;
174   /* 行列 A, ベクトル x, b を記憶する変数の準備 */
175   a = zeros(N, N);
176   x = new_vector(N);
177   b = new_vector(N);
178   /* 係数行列を決める */
179   for (i = 0; i < N; i++) {
180     a[i][i] = 4.0;
181     if (i != 0)
182       a[i][i-1] = -1.0;
183     if (i != N-1)
184       a[i][i+1] = -1.0;
185   }
186   a[n-1][n] = 0.0;
187   a[n][n-1] = 0.0;
188   /* */
189   for (i = 0; i < n; i++) {
190     if (i != 0)
191       a[i][i-1] *= c1;
192     a[i][i] *= c1;
193     a[i][i+1] *= c1;
194   }
195   for (i = n; i < N; i++) {
196     a[i][i-1] *= c2;
197     a[i][i] *= c2;
198     if (i != N-1)
199       a[i][i+1] *= c2;
200   }
201   /* 係数行列を表示する */
202   if (N < 10) {
203     cout << "a=" << endl;
204     print_matrix(N, N, a);
205   }
206   /* 解 x を決める */
207   for (i = 0; i < N; i++)
208     x[i] = i;
209   /* 解に対応する右辺 b を計算する */
210   multiply_mv(N, N, b, a, x);
211   /* b を表示する */
212   if (N < 10) {
213     cout << "b=" << endl;
214     print_vector(N, b);
215   }
216   /* 初期ベクトル x を零ベクトルにする */
217   zeros(N, x);
218   /* CG 法で解く */
219   cg(N, a, b, x, 1e-12, &niter);
220   /* 解を表示する */
221   cout << "x=" << endl;
222   print_vector(N, x);
```

```
223    cout << "反復回数=" << niter << endl;
224
225    return 0;
226  }
```