数理解析特論

連立 1 次方程式に対する CG 法の数値実験

桂田 祐史

2005 年 5 月 24 日

1 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。

2 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] 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)。

2.1 課題 2 (必修部分)

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:

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

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。

(1)

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

として方程式 $A^{(1)}x = b^{(1)}$ を解け。

(2) (1) 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。

(3) (1) 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。

(なお、最初は計算の効率をあまり考えなくても構わない。)

2.2 解説

実は上記の実験は、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 (1) 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、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) 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 (3) になると、固有値の広がり具合はまた (1) 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。

[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。

[4] とはいえ、このような 3 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。

2.3 課題 2 (努力目標部分)

以下の中から出来ることを二つ三つやってみよう。「難易度」は大まかな目安である (MATLAB を使うか、C を使うかでも異なる)。

- プログラムを C で書いた場合: 付録で説明した高橋版アルゴリズムを用いるようにプログラムを書き直してみよ。
- 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、もちろん MATLAB を使ってもよい (C, C++ で書いた場合もここだけは MATLAB を使って構わない)。
- 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。

2.4 方法の説明

今回の課題は MATLAB で解いても良いし、C または C++ で解いてもよい。

2.4.1 MATLAB の場合

MATLAB の場合は前回の資料[5]で説明してある程度の機能で十分のはずである。行列のブロックに小さな行列を埋め込むことだけ説明しておく。

```
>> n=5
>> n2=2*n
>> at=(1:n)'*(1:n)
>> a=zeros(n2,n2);
>> a(1:n,1:n)=at;
>> a(n+1:n2,n+1:n2)=10*at
```

とすると、次のようになる。

[5] http://www.math.meiji.ac.jp/~mk/lecture/suurikaisekitokuron/

```
a =

     1     2     3     4     5     0     0     0     0     0
     2     4     6     8    10     0     0     0     0     0
     3     6     9    12    15     0     0     0     0     0
     4     8    12    16    20     0     0     0     0     0
     5    10    15    20    25     0     0     0     0     0
     0     0     0     0     0    10    20    30    40    50
     0     0     0     0     0    20    40    60    80   100
     0     0     0     0     0    30    60    90   120   150
     0     0     0     0     0    40    80   120   160   200
     0     0     0     0     0    50   100   150   200   250
```

2.4.2 C でプログラムを作る

ベクトルとベクトルの内積、行列とベクトルの積などを計算する関数を作成すれば、後は比較的簡単にプログラムが書ける。

参考プログラム cg.c[6] を公開しておく。コンパイルした後、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、課題 2 の必修部分 (1) の問題を解くことができ、`10 1 100` と入力すると、(2) の問題を $c = 10$ で解くことができる。

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、

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

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

[6] http://www.math.meiji.ac.jp/~mk/lecture/suurikaisekitokuron/cg.c

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。

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 叩き台プログラムのリスト

以下に cg.c を掲げる。

```
 1  /*
 2   * cg.c --- CG 法のサンプル・プログラム
 3   *   コンパイル: gcc -O -o cg cg.c -lm
 4   */
 5
 6  #include <stdio.h>
 7  #include <math.h>
 8  #include <stdlib.h>
 9
10  typedef double *vector;
11  typedef vector *matrix;
12
13  vector new_vector(int n)   { return malloc(sizeof(double) * n); }
14  void free_vector(vector v) { free(v); }
15
16  matrix new_matrix(int nrow, int ncol)
17  {
18      int i;
19      vector ap;
20      matrix a;
21
22      if ((a = malloc(nrow * sizeof(vector))) == NULL)
```

```
23          return NULL;
24      if ((ap = malloc(nrow * ncol * sizeof(double))) == NULL) {
25          free(a); return NULL;
26      }
27      for (i = 0; i < nrow; i++)
28          a[i] = ap + (i * ncol);
29      return a;
30  }
31
32  void free_matrix(matrix a)
33  {
34      free(a[0]);
35      free(a);
36  }
37
38  /* m 行 n 列の零行列を作って返す */
39  matrix zeros(int m, int n)
40  {
41    int i, j;
42    matrix tmp;
43    tmp = new_matrix(m, n);
44    for (i = 0; i < m; i++)
45      for (j = 0; j < n; j++)
46        tmp[i][j] = 0.0;
47    return tmp;
48  }
49
50  /* n 次元のベクトル z を零ベクトルにする */
51  void clear_vector(int n, vector z)
52  {
53    int i;
54    for (i = 0; i < n; i++)
55      z[i] = 0.0;
56  }
57
58  // n 次元ベクトルの内容を表示する
59  void print_vector(int n, vector x)
60  {
61    int i;
62    for (i = 0; i < n; i++) {
63      printf("  %g", x[i]);
64      if (i % 5 == 4)
65        printf("\n");
66    }
67    if (i % 5 != 0)
68      printf("\n");
69  }
70
71  // m 行 n 列の行列の内容を表示する
72  void print_matrix(int m, int n, matrix a)
73  {
74    int i;
75    for (i = 0; i < m; i++)
76      print_vector(n, a[i]);
77  }
78
79  // ベクトル x, y の内積 (x,y) を計算する
80  double dotproduct(int n, vector x, vector y)
```

```
 81  {
 82    int i;
 83    double sum;
 84    sum = 0;
 85    for (i = 0; i < n; i++)
 86      sum += x[i] * y[i];
 87    return sum;
 88  }
 89
 90  // ベクトル x のノルム ||x|| を計算する
 91  double norm(int n, vector x)
 92  {
 93    return sqrt(dotproduct(n, x, x));
 94  }
 95
 96  // 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
 97  void multiply_mv(int m, int n, vector Ab, matrix A, vector b)
 98  {
 99    int i;
100    for (i = 0; i < m; i++)
101        Ab[i] = dotproduct(n, A[i], b);
102  }
103
104  // ベクトル x をベクトル y にコピーする
105  void copy_vector(int n, vector y, vector x)
106  {
107    int i;
108    for (i = 0; i < n; i++)
109      y[i] = x[i];
110  }
111
112  /* CG 法で n 元連立 1 次方程式 A x=b を解く。
113   * 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。
114   * ||残差|| < ε  ||b|| となったら反復を停止する。
115   * 反復回数を maxiter に返す。
116   */
117
118  void cg(int n, matrix A, vector b, vector x, double eps, int *maxiter)
119  {
120    int i, k;
121    double eps_b, pAp, alpha, beta;
122    vector r, p, Ap;
123
124    r = new_vector(n);
125    p = new_vector(n);
126    Ap = new_vector(n);
127    /* eps_b := ε  ||b|| */
128    eps_b = eps * norm(n, b);
129    /* r := A x */
130    multiply_mv(n, n, r, A, x);
131    /* r := b - A x */
132    for (i = 0; i < n; i++)
133      r[i] = b[i] - r[i];
134    /* p := r */
135    copy_vector(n, p, r);
136    /* 反復 */
137    for (k = 0; k < n; k++) {
138      /* Ap := A * p */
```

```
139       multiply_mv(n, n, Ap, A, p);
140       /* pAp := (p, Ap) */
141       pAp = dotproduct(n, p, Ap);
142       /* α  = (r,p)/(p,Ap) */
143       alpha = dotproduct(n, r, p) / pAp;
144       /* x, r の更新 */
145       for (i = 0; i < n; i++) {
146         /* x = x + α *p */
147         x[i] += alpha * p[i];
148         /* r = r - α *A*p */
149         r[i] -= alpha * Ap[i];
150       }
151       /* ||r||<ε ||b|| ならば反復を終了 */
152       if (norm(n, r) < eps_b)
153         break;
154       /* β  := - (r, Ap) / (p, Ap) */
155       beta = - dotproduct(n, r, Ap) / pAp;
156       /* p := r + β *p */
157       for (i = 0; i < n; i++)
158         p[i] = r[i] + beta * p[i];
159     }
160     if (maxiter != NULL)
161       *maxiter = k;
162   }
163
164   int main()
165   {
166     int n, N, i, niter;
167     matrix a;
168     vector x, b;
169     double c1, c2;
170     /* 問題のサイズを決める */
171     printf("c1, c2, N=");
172     scanf("%lf %lf %d", &c1, &c2, &N);
173     n = N / 2;
174     N = 2 * n;
175     /* 行列 A, ベクトル x, b を記憶する変数の準備 */
176     a = zeros(N, N);
177     x = new_vector(N);
178     b = new_vector(N);
179     /* 係数行列を決める */
180     for (i = 0; i < N; i++) {
181       a[i][i] = 4.0;
182       if (i != 0)
183         a[i][i-1] = -1.0;
184       if (i != N-1)
185         a[i][i+1] = -1.0;
186     }
187     a[n-1][n] = 0.0;
188     a[n][n-1] = 0.0;
189     /* */
190     for (i = 0; i < n; i++) {
191       if (i != 0)
192         a[i][i-1] *= c1;
193       a[i][i] *= c1;
194       a[i][i+1] *= c1;
195     }
196     for (i = n; i < N; i++) {
```

```
197       a[i][i-1] *= c2;
198       a[i][i] *= c2;
199       if (i != N-1)
200         a[i][i+1] *= c2;
201     }
202     /* 係数行列を表示する */
203     if (N < 10) {
204       printf("a=\n");
205       print_matrix(N, N, a);
206     }
207     /* 解 x を決める */
208     for (i = 0; i < N; i++)
209       x[i] = i;
210     /* 解に対応する右辺 b を計算する */
211     multiply_mv(N, N, b, a, x);
212     /* b を表示する */
213     if (N < 10) {
214       printf("b=\n");
215       print_vector(N, b);
216     }
217     /* 初期ベクトル x を零ベクトルにする */
218     clear_vector(N, x);
219     /* CG 法で解く */
220     cg(N, a, b, x, 1e-12, &niter);
221     /* 解を表示する */
222     printf("x=\n");
223     print_vector(N, x);
224     printf("反復回数=%d\n", niter);
225
226     return 0;
227   }
```