

数值分析第一次大作业

傅锡豪, 15091082

2017 年 4 月 12 日

1 算法设计

本程序的算法包括以下四个方面

一、存储： 由于题目的矩阵是稀疏的带状矩阵，所以通过把矩阵逆时针旋转储存，大大减少存储空间。原矩阵和被压缩矩阵之间元素的对应关系是： $a_{i,j} = c_{i-j+3,j}$ 。原本是 501x501 的矩阵，经过压缩后是 5x501 的矩阵，而且只有 $c_{1,1}$ $c_{1,2}$ $c_{2,1}$ $c_{501,501}$ $c_{501,500}$ $c_{500,501}$ 是不存储数据的，其余空间都存放了原矩阵的非 0 数据。所以后面使用幂法和反幂法求解特征值的时候都是使用针对**带状矩阵**的算法。

二、计算特征值： 对于该题，需要分别求解最大、最小、按模最小、和距离特定 39 个值最近的特征值，分别由以下算法给定：

(1) **求解按模最大特征值：**

直接对矩阵 A 使用幂法，可直接求解出按模最大的特征值 λ_{max} 。

(2) **通过平移求出另一特征值：**

用 (1) 求的 λ_{max} 的绝对值对矩阵 A 做平移，对 $(A - |\lambda_{max}|I)$ 进行反幂法操作得到 $1/\beta$ ，计算 $\lambda = 1/\beta + |\lambda_{max}|$ 。通过比较 λ_{max} 和 λ 的值的大小，把数值大的赋给 λ_{501} ，数值小的赋给 λ_1 。

(3) **求按模最小特征值**

对矩阵 A 用反幂法直接求出按模最小的特征值，即得到 λ_s 。

(4) **求和 μ_k 相近的特征值**

进入循环：k: $1 \rightarrow 39$ ，依次执行以下操作：

1. 求解 μ_k : $\mu_k = \lambda_1 + k * (\lambda_{501} - \lambda_1)/40$;

2. 对用 μ_k 的值矩阵 A 平移，对 $(A - \mu_k I)$ 做反幂法操作，得到 $1/\beta_k$;

3. 距离 μ_k 最近的特征值由下式给定： $\lambda_{\mu_k} = \mu_k + 1/\beta_k$

三、A 的条件数

由非奇异对称矩阵的条件数由该式给定： $cond(A)_2 = |\lambda_1/\lambda_n|$ ，其中 λ_1 和 λ_n 分别是该矩阵的模最大和模最小的特征值。

在该题中， $cond(A)_2 = |\lambda_{max}/\lambda_s|$ 。

四、A 的行列式

本题中 A 的阶数过大，无法直接求解行列式。采用将 A 进行 Doolittle 分解，则 $|A| = |L||U|$ 。由于 L 是对角线为 1 的下三角阵，而 U 是三上角阵，所以 $|A| = |L||U| = \prod_{i=1}^{501} u_{i,i}$ 。

在本题中，由于使用反幂法时对 A 做了 Doolittle 分解，所以在对矩阵 A 用反幂法求出 λ_s 后，把被处理过的 A 第 3 行（相当于未被压缩矩阵的对角线）各元素相乘，即 $det(A) = \prod_{i=1}^{501} c_{3,i}$ 。

2 源程序

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <iostream>
4  #include <float.h>
5  #include <iomanip>
6  #include <cmath>
7  #define MEle A->Elements
8  #define VEle b->Elements
9  #define SEle Solution->Elements
10 #define YEle y->Elements
11 #define UEle u->Elements
12 #define TEle tp->Elements
13 #define AREle A_remain->Elements
14 using namespace std;
15
16 int main()
17 {
18     double Epthron = 10e-12;
19     double c, b;
20     double Miu[39];
21     double a[501];
22     double DET_A = 1;
23     double lambda_s;
24     double lambda_max;
25     double lambda1;
26     double lambda501;
27     double cond;
28     int s = 2;
29     Matrix* A = new Matrix(5, 501);
30     //用于保护矩阵

```

```

31     Matrix* A_remain = new Matrix(5, 501);
32     A->setR(2);
33     A->setS(2);
34
35     for (int j = 0; j < 501; j++){
36         a[j] = (1.64-0.024*(j+1))*sin(0.2*(j+1))-0.64*exp(0.1/(j+1));
37     }
38     b = 0.16;
39     c = -0.064;
40     //init the MATRIX A
41     for (int j = 0; j < 501; j++){
42         if (j > 1) {
43             MEle[0][j] = c;
44             AREle[0][j] = c;
45         }
46         if (j > 0){
47             MEle[1][j] = b;
48             AREle[1][j] = b;
49         }
50         if (j < 500){
51             MEle[3][j] = b;
52             AREle[3][j] = b;
53         }
54         if (j < 499){
55             MEle[4][j] = c;
56             AREle[4][j] = c;
57         }
58         MEle[2][j] = a[j];
59         AREle[2][j] = a[j];
60     }
61
62     //get the one with max module
63     lambda_max = PowerMethod(A, Epthron);
64
65     //translate to get the other
66     A->Translate(abs(lambda_max));
67     lambda1 = InversePowerMethod(A, Epthron) + abs(lambda_max);
68
69     //compare lambda1 and lambda501
70     if (lambda1 > lambda_max){
71         lambda501 = lambda1;
72         lambda1 = lambda_max;
73     }
74     else {
75         lambda501 = lambda_max;
76     }
77
78     //get lambda_s
79     Recover(A_remain, A);
80     lambda_s = InversePowerMethod(A, Epthron);
81

```

```

82      //A stands for L+U, thus the products of the diagonal is detA
83      for (int i = 0; i < 501; ++i)
84      {
85          DET_A*=MEle[s][i];
86      }
87
88      cond = abs(lambda_max/lambda_s);
89
90      //translate to get those close to Miu
91      for (int i = 0; i < 39; ++i)
92      {
93          Miu[i] = lambda1 + (i+1)*(lambda501 - lambda1)/40;
94          Recover(A_remain, A);
95          //translate
96          A->Translate(Miu[i]);
97
98          cout << "lambda_miu" << i+1 << " = " << setprecision(11) << scientific << InversePowerMethod(A,
99              Epthron)+Miu[i] << endl;
100      }
101
102      cout << "lambda501 = " << lambda501 << endl;
103      cout << "lambda1 = " << lambda1 << endl;
104      cout << "lambda_s = " << lambda_s << endl;
105      cout << "cond_A = " << cond << endl;
106      cout << "det_A = " << DET_A << endl;
107
108      return 0;
109  }
110
111  int Minimum(int m, int n){
112      if (m > n) {
113          return n;
114      }
115      else {
116          return m;
117      }
118  }
119
120  int Maximum(int m, int n){
121      if (m > n) {
122          return m;
123      }
124      else {
125          return n;
126      }
127  }
128
129  int Maximum(int o, int p, int q){
130      if (o > p)
131      {
132          if (o > q)
133          {
134              return o;

```

```

132         }
133         else {
134             return q;
135         }
136     }
137     else if (p > q)
138     {
139         return p;
140     }
141     else {
142         return q;
143     }
144 }
145 class Matrix{
146 private:
147     int row;
148     int column;
149     int s, r;
150 public:
151     double **Elements;
152     Matrix(int row, int column){
153         this->row = row;
154         this->column = column;
155         Elements = (double**)malloc(sizeof(double)*row);
156         for (int i = 0; i < row; ++i)
157         {
158             Elements[i] = (double*)malloc(sizeof(double)*column);
159         }
160     }
161     void setValue(){
162         this->r = 0;
163         this->s = 0;
164         double **p = Elements;
165         cout << "Input Matrix:\n";
166         for (int i = 0; i < this->row; ++i)
167         {
168             for (int j = 0; j < this->column; ++j)
169             {
170                 cin >> p[i][j];
171             }
172         }
173     }
174
175     //for saving storage
176     //s means the number of diagonals that is above the central diagonal
177     void setValue(int row, int column, int s, int r){
178         double **p = Elements;
179         double temp;
180         this->s = s;
181         this->r = r;
182         cout << "Input Matrix:\n";

```

```

183         for (int i = 0; i < row; ++i)
184         {
185             for (int j = 0; j < column; ++j)
186             {
187                 cin >> temp;
188                 if (temp != 0.0)
189                 {
190                     p[i-j+s][j] = temp;
191                 }
192             }
193         }
194     }
195
196     void setR(int r){
197         this->r = r;
198     }
199     void setS(int s){
200         this->s = s;
201     }
202     int getR(){
203         return this->r;
204     }
205     int getS(){
206         return this->s;
207     }
208     int getRow(){
209         return this->row;
210     }
211     int getColumn(){
212         return this->column;
213     }
214
215     void Translate(double distance){
216         if (this->getR() == this->getS() == 0){
217             for (int i = 0; i < this->getColumn(); ++i)
218             {
219                 Elements[i][i] -= distance;
220             }
221         }
222         else{
223             for (int i = 0; i < this->getColumn(); ++i){
224                 Elements[this->s][i] -= distance;
225             }
226         }
227     }
228 };
229
230 class Vector{
231 private:
232     int size;
233 public:

```

```

234     double* Elements;
235     Vector(int size){
236         this->size = size;
237         Elements = (double*)malloc(sizeof(double)*size);
238     }
239     void setValue(){
240         double *p = Elements;
241         cout << "Input Vector:\n";
242         for (int i = 0; i < this->size; ++i)
243             {
244                 cin >> p[i];
245             }
246     }
247
248     int getSize(){
249         return this->size;
250     }
251     double getNorm2(){
252         double norm = 0;
253         for (int i = 0; i < this->size; i++){
254             norm+=Elements[i]*Elements[i];
255         }
256         return sqrt(norm);
257     }
258 };
259 class EquationGroup{
260 private:
261     Matrix* A;
262     Vector* b;
263     Vector* Solution;
264 public:
265     //in order to make storage more economical
266     void initEquation(Matrix* A, Vector* b){
267         this->A = A;
268         this->b = b;
269         Solution = new Vector(A->getColumn());
270     }
271
272     Vector* getSolution(){
273         return Solution;
274     }
275
276     void TrianglarDecomposition(){
277         int n = A->getRow();
278         double u, l;
279         for(int k = 0; k < n; k++){
280
281             for(int j = k; j < n; j++){
282                 u = MEle[k][j];
283                 for(int t = 0; t <= k-1; t++){
284                     u-=MEle[k][t]*MEle[t][j];

```

```

285         }
286
287         MEle[k][j] = u;
288     }
289     for(int i = k+1; i < n; i++){
290         l = MEle[i][k];
291         for(int t = 0; t <= k-1; t++){
292             l-=MEle[i][t]*MEle[t][k];
293         }
294         l/=MEle[k][k];
295
296         MEle[i][k] = l;
297     }
298 }
299 }
300 void Substitution(){
301     int n = A->getRow();
302     double y[n];
303     y[0] = VEle[0];
304     for(int i = 1; i < n; i++){
305         y[i] = VEle[i];
306         for(int t = 0; t <= i-1; t++){
307             y[i]-=MEle[i][t]*y[t];
308         }
309     }
310     SEle[n-1] = y[n-1]/MEle[n-1][n-1];
311     for(int i = n-2; i >= 0; i--){
312         SEle[i] = y[i];
313         for(int t = i+1; t < n; t++){
314             SEle[i]-=MEle[i][t]*SEle[t];
315         }
316         SEle[i]/=MEle[i][i];
317     }
318 }
319
320 void TrianglarDecompStrip(){
321     int s = A->getS();
322     int r = A->getR();
323     int n = A->getColumn();
324     int temp1;
325     int temp2;
326     for (int k = 0; k < n; k++){
327         temp1 = Minimun(k+s, n-1);
328         for (int j = k; j <= temp1; j++){
329             temp2 = Maximum(0, k-r, j-s);
330             for (int t = temp2; t < k; t++){
331                 MEle[k-j+s][j] -= MEle[k-t+s][t]*MEle[t-j+s][j];
332             }
333         }
334         temp1 = Minimun(k+r, n-1);
335         for (int i = k+1; i <= temp1; i++){

```



```

336         temp2 = Maximum(0, i-r, k-s);
337         for (int t = temp2; t < k; t++){
338             MEle[i-k+s][k] -= MEle[i-t+s][t]*MEle[t-k+s][k];
339         }
340         MEle[i-k+s][k] = MEle[i-k+s][k]/MEle[s][k];
341     }
342 }
343 }
344
345 void SubstitutionStrip(){
346     int temp;
347     int s = A->getS();
348     int r = A->getR();
349     int n = b->getSize();
350
351     //保护列向量
352     Vector* tp = new Vector(n);
353     for (int i = 0; i < n; ++i){
354         TEle[i] = VEle[i];
355     }
356     for (int i = 1; i < n; ++i){
357         temp = Maximum(0, i-r);
358         for (int t = temp; t < i; t++){
359             TEle[i] -= MEle[i-t+s][t]*TEle[t];
360         }
361     }
362     SEle[n-1] = TEle[n-1]/MEle[s][n-1];
363     for (int i = n-2; i >= 0; i--){
364         SEle[i] = TEle[i];
365         temp = Minimum(i+s, n-1);
366         for (int t = i+1; t <= temp; t++){
367             SEle[i] -= MEle[i-t+s][t]*SEle[t];
368         }
369         SEle[i] = SEle[i]/MEle[s][i];
370     }
371     delete tp;
372 }
373
374 };
375
376 double DotProduct(Vector* X, Vector* Y){
377     double product = 0;
378     for (int i = 0; i < X->getSize(); i++){
379         product+=(X->Elements[i]*Y->Elements[i]);
380     }
381     return product;
382 }
383
384 double PowerMethod(Matrix* A, double Error){
385     double Yita;
386     double Beta;

```

```

387     double FormerBeta;
388     double LocalError;
389     int column = A->getColumn();
390     int row = A->getRow();
391     //the s of Matrix_A
392     int s = A->getS();
393     Vector* y = new Vector(column);
394     Vector* u = new Vector(column);
395
396     //generate u0
397     for (int i = 0; i < column; i++){
398         UEle[i] = 0.1;
399     }
400
401     //begin loop, until the error is smaller the given one
402     do {
403         Yita = u->getNorm2();
404         for (int i = 0; i < u->getSize(); i++){
405             YEle[i] = UEle[i]/Yita;
406         }
407
408         //modify u
409         if (A->getR() == A->getS() == 0){
410             for (int i = 0; i < row; i++){
411                 UEle[i] = 0;
412                 for (int j = 0; j < column; j++){
413                     UEle[i] += MEle[i][j]*YEle[j];
414                 }
415             }
416         }
417         else {
418             for (int i = 0; i < column; i++){
419                 UEle[i] = 0;
420                 for (int j = 0; j < column; j++){
421                     if (i-j+s >= 0 && i-j+s < row)
422                     {
423                         UEle[i] += MEle[i-j+s][j]*YEle[j];
424                     }
425                 }
426             }
427
428         }
429
430         FormerBeta = Beta;
431         Beta = DotProduct(y, u);
432         LocalError = abs(Beta - FormerBeta)/abs(Beta);
433     } while (LocalError > Error);
434
435     return Beta;
436 }
437 double InversePowerMethod(Matrix* A, double Error){

```

```

438     double Beta;
439     double FormerBeta;
440     double LocalError;
441     double Yita;
442     int column = A->getColumn();
443     int row = A->getRow();
444
445     Vector* y = new Vector(column);
446     Vector* u = new Vector(column);
447
448     EquationGroup* E = new EquationGroup();
449     E->initEquation(A, y);
450
451     //do the decomposition
452     if (A->getR() == A->getS() == 0){
453         E->TrianglarDecomposition();
454     }
455     else {
456         E->TrianglarDecompStrip();
457     }
458
459     //generate u0
460     for (int i = 0; i < column; i++){
461         UEle[i] = 0.1;
462     }
463     //begin loop, until the error is smaller the given one
464     do {
465         Yita = u->getNorm2();
466         for (int i = 0; i < column; i++){
467             YEle[i] = UEle[i]/Yita;
468         }
469
470         //solve equation to get u
471         E->initEquation(A, y);
472
473         if (A->getR() == A->getS() == 0){
474             E->Substitution();
475         }
476         else {
477             E->SubstitutionStrip();
478         }
479
480         for (int i = 0; i < column; i++){
481             UEle[i] = E->getSolution()->Elements[i];
482         }
483
484         FormerBeta = Beta;
485         Beta = DotProduct(y, u);
486         LocalError = abs(1/Beta - 1/FormerBeta)/abs(1/Beta);
487     } while (LocalError > Error);
488

```

```

489     return 1/Beta;
490 }
491
492 void Recover(Matrix* A_for_copying, Matrix* A_for_targeting){
493     int row = A_for_targeting->getRow();
494     int column = A_for_targeting->getColumn();
495     for (int i = 0; i < row; ++i)
496     {
497         for (int j = 0; j < column; ++j)
498         {
499             A_for_targeting->Elements[i][j] = A_for_copying->Elements[i][j];
500         }
501     }
502 }

```

3 打印结果

```

1  lambda_miu1 = -1.01829340331e+01
2  lambda_miu2 = -9.58570742506e+00
3  lambda_miu3 = -9.17267242393e+00
4  lambda_miu4 = -8.65228400790e+00
5  lambda_miu5 = -8.09348380867e+00
6  lambda_miu6 = -7.65940540769e+00
7  lambda_miu7 = -7.11968464869e+00
8  lambda_miu8 = -6.61176433940e+00
9  lambda_miu9 = -6.06610322659e+00
10 lambda_miu10 = -5.58510105263e+00
11 lambda_miu11 = -5.11408352981e+00
12 lambda_miu12 = -4.57887217687e+00
13 lambda_miu13 = -4.09647092626e+00
14 lambda_miu14 = -3.55421121575e+00
15 lambda_miu15 = -3.04109001813e+00
16 lambda_miu16 = -2.53397031113e+00
17 lambda_miu17 = -2.00323076956e+00
18 lambda_miu18 = -1.50355761123e+00
19 lambda_miu19 = -9.93558606008e-01
20 lambda_miu20 = -4.87042673885e-01
21 lambda_miu21 = 2.23173624957e-02
22 lambda_miu22 = 5.32417474207e-01
23 lambda_miu23 = 1.05289896269e+00
24 lambda_miu24 = 1.58944588188e+00
25 lambda_miu25 = 2.06033046027e+00
26 lambda_miu26 = 2.55807559707e+00
27 lambda_miu27 = 3.08024050931e+00
28 lambda_miu28 = 3.61362086769e+00
29 lambda_miu29 = 4.09137851045e+00
30 lambda_miu30 = 4.60303537828e+00
31 lambda_miu31 = 5.13292428390e+00
32 lambda_miu32 = 5.59490634808e+00

```

```
33 lambda_miu33 = 6.08093385703e+00
34 lambda_miu34 = 6.68035409211e+00
35 lambda_miu35 = 7.29387744813e+00
36 lambda_miu36 = 7.71711171424e+00
37 lambda_miu37 = 8.22522001405e+00
38 lambda_miu38 = 8.64866606519e+00
39 lambda_miu39 = 9.25420034458e+00
40 lambda501 = 9.72463409966e+00
41 lambda1 = -1.07001136138e+01
42 lambda_s = -5.55791079436e-03
43 cond_A = 1.92520427364e+03
44 det_A = 2.77278614175e+118
```

4 讨论

对于使用反幂法之后矩阵 A 改变的考虑

由于使用反幂法时，需要对矩阵 A 和一个列向量求解线性方程组，而在使用 Doolittle 分解求解的时候，为了节省存储空间，会把分解后的 L 和 U 矩阵存在原本的矩阵 A 的位置，导致在每次使用反幂法之后，矩阵 A 都会被改变。本题中使用一个矩阵来保留原矩阵 A 的值，在每次进行反幂法之后都使用一个函数（源程序 492 行），把被修改过的 A 进行恢复。这样既需要多出一个矩阵的存储空间，还需要多次恢复该矩阵，增加了操作次数（尽管和内部求解方程组的时间复杂度 $O(n^3)$ 相比，并不会改变时间复杂度的量级）。而如果在 Doolittle 分解的时候不把 L 和 U 存在 A 的位置，而是开辟另一块内存空间，多次回带结束之后释放该空间，这样重复使用反幂法的时候也最多只有一个矩阵的空间占用，并且在时间上不需要像上面一样进行一个二重循环，也可以节省算法的操作时间。所以**对于本题来说**，或许可以修改 Doolittle 算法来避免这个恢复矩阵 A 的过程。