



BEIJING UNIVERSITY OF CHEMICAL TECHNOLOGY

COURSE HOMEWORKS

矩阵的谱半径

计科 1703 张俊峰 2017040330

目录

第 1 章 矩阵的谱半径	1
1.1 简介	1
1.2 理论分析	1
1.3 算法描述	2
1.4 实例分析	3
1.5 总结	11

创建日期: 2019 年 11 月 19 日

更新日期: 2019 年 11 月 20 日

第 1 章 矩阵的谱半径

1.1 简介

设 A 是 $n \times n$ 矩阵, $\lambda_1, \dots, \lambda_n$ 是其特征值, 则 A 的谱半径 $\rho(A) = \max \{|\rho_1|, \dots, |\rho_n|\}$.

(1) 矩阵的谱半径等于矩阵的特征值绝对值的最大值。

(2) 在数学中, 矩阵的谱半径是指其特征值绝对值集合的上确界。

1.2 理论分析

定理 1. 设 $A \in R^{n \times n}$ 为 n 阶方阵, 则对任意矩阵范数 $\|\cdot\|$ 都有 $\rho(A) \leq \|\cdot\|$.

定理 2. 若 A 为 n 阶正规矩阵, 则 $\rho(A) \leq \|A\|_2$.

证明. 因 A 是正规矩阵, 故存在幺正矩阵 P , 使得

$$P^H A P = \begin{bmatrix} \lambda_1 & & \\ & \dots & \\ & & \lambda_n \end{bmatrix} \quad (1.1)$$

□

由此可得:

$$P^H A^H P = \begin{bmatrix} \bar{\lambda}_1 & & \\ & \dots & \\ & & \bar{\lambda}_n \end{bmatrix} \quad (1.2)$$

从而:

$$P^H A^H A P = \begin{bmatrix} |\lambda_1|^2 & & \\ & \dots & \\ & & |\lambda_n|^2 \end{bmatrix} \quad (1.3)$$

又显然有:

$$\lambda_A H_A = \max \{|\lambda_1|^2, |\lambda_2|^2, \dots, |\lambda_n|^2\} = |\lambda_t|^2.$$

这里 t 是 $\{1, 2, \dots, n\}$ 中的某一值, 因此有:

$$\|A\|_2 = \sqrt{\lambda_A H_A} = |\lambda_t|.$$

又因为:

$$\rho(A) = \max \{|\lambda_1|, |\lambda_2|, \dots, |\lambda_n|\} = |\lambda_t|$$

所以, 可以证明:

$$\rho(A) \leq \|A\|_2$$

1.3 算法描述

Algorithm 1: 算法

Input: 要进行计算你的矩阵的行与列，并且按照行的顺序依次输入矩阵元素;
Output: 该矩阵的所有特征值和矩阵的谱半径 (绝对值最大的特征值);

- 1 输入矩阵的行与列以及矩阵内的元素;
- 2 *bool InitMatrix(Matrix *matrix, int row, int column)*
- 3 定义一个结构体, 用来表示二维矩阵;
- 4 *typedef struct*
- 5 {
- 6 *int row;*
- 7 *int column;*
- 8 *double *data;*
- 9 }*Matrix;*
- 10 使用 QR 分解求矩阵特征值
- 11 *for k = 0 → NUM*
- 12 {
- 13 *QR(temp, temp_Q, temp_R);*
- 14 *MatrixMulMatrix(temp, temp_R, temp_Q);* 分解后两个矩阵相乘
- 15 }
- 16 获取特征值, 将特征值存储在 *eValue* 中
- 17 *for k = 0 → temp.column*
- 18 {
- 19 *eValue.data[k] = temp.data[k * temp.column + k];*
- 20 *result[k] = fabs(temp.data[k * temp.column + k]);*
- 21 }
- 22 输出特征值以及矩阵的谱半径
- 23 *float maxn = -1*
- 24 *for k = 0 → temp.column*
- 25 {
- 26 *if (result[k] > maxn)*
- 27 *maxn = result[k];*
- 28 }
- 29 *PrintMatrix(eValue);*
- 30 *printf* 矩阵的谱半径 (即特征值绝对值最大) 为: *maxn*;

1.4 实例分析

```

D:\Learn_Files\CalculateMethod\Tests\bin\Debug\Tests.exe
请输入要进行计算的矩阵行与列(以逗号隔开): 4, 4
按照以行的输入顺序依次输入矩阵内的元素, 一共输入16个元素: 1 2 5 4 2 1 3 -3 5 3 -6 -4 4 -3 -4 2
输入矩阵如下:
    1      2      5      4
    2      1      3     -3
    5      3     -6     -4
    4     -3     -4      2

特征值:
   -10.9
   -3.033
    5.39
    6.547

矩阵的谱半径(即特征值绝对值最大)为:    10.903543
    
```

图 1.1: sample

```

1  #include<stdio.h>
2  #include<stdlib.h>
3  #include <math.h>
4  #include <stdbool.h>
5
6  //定义一个结构体, 用来表示一个二维的矩阵
7  typedef struct
8  {
9      int row;
10     int column;
11     double *data; //用来存放矩阵的元素
12 }Matrix;
13
14 /*****
15 函数功能: 初始化一个矩阵
16 输入: 要初始化的矩阵matrix、矩阵的行row、矩阵的列column
17 输出: 初始化成功: true; 初始化失败: false
18 *****/
19 bool InitMatrix(Matrix *matrix, int row, int column)
20 {
21     int matrix_size = row*column*sizeof(double);
22     if (matrix_size <= 0)
23         return false;
24     matrix->data = (double*)malloc(matrix_size); //给矩阵分配空间
25     if (matrix->data)
26     {
27         matrix->row = row;
28         matrix->column = column;
29         return true;
    
```

```

30     }
31     else
32     {
33         matrix->row = 0;
34         matrix->column = 0;
35         return false;
36     }
37 }
38
39 /*****
40 函数功能：打印出一个矩阵
41 输入：一个矩阵matrix
42 输出：无
43 *****/
44 void PrintMatrix(Matrix *matrix)
45 {
46     int matrix_num = matrix->row*matrix->column;
47     for (int i = 0; i < matrix_num; i++)
48     {
49         printf("%12.4g ", matrix->data[i]);
50         if ((i + 1) % (matrix->column) == 0)
51             printf("\n");
52     }
53     printf("\n");
54 }
55
56 /*****
57 函数功能：获取一个矩阵的大小
58 输入：一个矩阵matrix
59 输出：矩阵的大小size
60 *****/
61 int GetMatrixSize(Matrix *matrix)
62 {
63     return matrix->row*matrix->column;
64 }
65
66 /*****
67 函数功能：清零，使矩阵每个元素为0
68 输入：需要清零的矩阵matrix
69 输出：无
70 *****/
71 void SetMatrixZeros(Matrix *matrix)
72 {
73     int matrix_num = GetMatrixSize(matrix);
74     for (int i = 0; i < matrix_num; i++)
75         matrix->data[i] = 0;
76 }
77

```

```

78  /*****
79  函数功能：判断一个矩阵是否为空
80  输入：一个矩阵matrix
81  输出：为空则true，否则为false
82  *****/
83  bool IsNullMatrix(Matrix *matrix)
84  {
85      int matrix_num = GetMatrixSize(matrix);
86      if ((matrix_num <= 0) || (matrix->data == NULL))
87          return true;
88      else
89          return false;
90  }
91
92  /*****
93  函数功能：释放掉一个矩阵
94  输入：一个矩阵matrix
95  输出：无
96  *****/
97  void DestroyMatrix(Matrix *matrix)
98  {
99      if (!IsNullMatrix(matrix))
100      {
101          matrix->data = NULL;
102          matrix->row = 0;
103          matrix->column = 0;
104          free(matrix->data);
105      }
106  }
107
108  /*****
109  函数功能：计算一个矩阵的2范数，即求模
110  输入：一个矩阵matrix
111  输出：所求的范数结果norm2_ans
112  *****/
113  double MatrixNorm2(Matrix *matrix)
114  {
115      double norm2_ans = 0.0;
116      int matrix_num = GetMatrixSize(matrix);
117      for (int i = 0; i < matrix_num; i++)
118          norm2_ans += (matrix->data[i]) * (matrix->data[i]);
119      norm2_ans = (double)sqrt(norm2_ans);
120      return norm2_ans;
121  }
122
123  /*****
124  函数功能：把一个矩阵复制
125  输入：需要进行复制的矩阵matrix_A，复制得到的一个矩阵matrix_B

```

```

126 输出：无
127 ******/
128 void CopyMatrix(Matrix *matrix_A, Matrix *matrix_B)
129 {
130     if (matrix_B->row != matrix_A->row)
131         matrix_B->row = matrix_A->row;
132     if (matrix_B->column != matrix_A->column)
133         matrix_B->column = matrix_A->column;
134     int size_A = GetMatrixSize(matrix_A);
135     for (int i = 0; i < size_A; i++)
136         matrix_B->data[i] = matrix_A->data[i];
137 }
138
139 /*****
140 函数功能：对一个方阵A进行QR分解
141 输入：需要分解的矩阵A、分解后的正交矩阵Q和上三角矩阵R
142 输出：无
143 *****/
144 void QR(Matrix *A, Matrix *Q, Matrix *R)
145 {
146     Matrix col_A, col_Q;
147     InitMatrix(&col_A, A->row, 1);
148     SetMatrixZeros(&col_A); //用来存A的每一列
149     InitMatrix(&col_Q, A->row, 1);
150     SetMatrixZeros(&col_Q); //用来存Q的每一列
151
152     if (A->row != A->column)
153         printf("A is not a square matrix!");
154
155     int A_size = GetMatrixSize(A);
156     int Q_size = GetMatrixSize(Q);
157     int R_size = GetMatrixSize(R);
158
159     if (Q_size != A_size)
160     {
161         DestroyMatrix(Q);
162         InitMatrix(Q, A->row, A->column);
163         SetMatrixZeros(Q);
164     }
165     else
166     {
167         Q->row = A->row;
168         Q->column = A->column;
169         SetMatrixZeros(Q);
170     }
171
172     if (R_size != A_size)
173     {

```

```

174     DestroyMatrix(R);
175     InitMatrix(R, A->row, A->column);
176     SetMatrixZeros(R);
177 }
178 else
179 {
180     R->row = A->row;
181     R->column = R->column;
182     SetMatrixZeros(R);
183 }
184
185 //施密特正交化
186 for (int j = 0; j < A->column; j++)
187 {
188     for (int i = 0; i < A->column; i++)//把A的第j列存入col_A中
189     {
190         col_A.data[i] = A->data[i * A->column + j];
191         col_Q.data[i] = A->data[i * A->column + j];
192     }
193     for (int k = 0; k < j; k++)//计算第j列以前
194     {
195         R->data[k * R->column + j] = 0;
196         for (int i1 = 0; i1 < col_A.row; i1++)
197             { //R=Q'A(Q'即Q的转置) 即Q的第k列和A的第j列做内积
198                 R->data[k * R->column + j] += col_A.data[i1] * Q->data[i1 * Q->column
199                     + k]; //Q的第k列
200             }
201         for (int i2 = 0; i2 < A->column; i2++)
202         {
203             col_Q.data[i2] -= R->data[k * R->column + j] * Q->data[i2 * Q->
204                 column + k];
205         }
206     }
207     double temp = MatrixNorm2(&col_Q);
208     R->data[j * R->column + j] = temp;
209     for (int i3 = 0; i3 < Q->column; i3++)
210     {
211         //单位化Q
212         Q->data[i3 * Q->column + j] = col_Q.data[i3] / temp;
213     }
214 }
215 DestroyMatrix(&col_A);
216 DestroyMatrix(&col_Q);
217 }
218
219 /*****

```



```

220  函数功能：给特征值排序，当flag=1时，则升序，当flag=0，则降序
221  输入：需要排序的序列eValue，升序还是降序的选择flag
222  输出：排序成功则返回true，否则返回false
223  *****/
224  bool SortEigenValues(Matrix *eValue, int flag)
225  {
226      int size = GetMatrixSize(eValue);
227
228      for (int i = 0; i < size - 1; i++)
229      {
230          int k = i;
231          for (int j = i + 1; j < size; j++)
232          {
233              if (flag == 1)
234              {
235                  if (eValue->data[k] > eValue->data[j])
236                  {
237                      k = j;
238                  }
239              }
240              else if (flag == 0)
241              {
242                  if (eValue->data[k] < eValue->data[j])
243                  {
244                      k = j;
245                  }
246              }
247              else
248                  return false;
249          }
250          if (k != i)
251          {
252              double temp;
253              temp = eValue->data[i];
254              eValue->data[i] = eValue->data[k];
255              eValue->data[k] = temp;
256          }
257      }
258      return true;
259  }
260
261  *****/
262  函数功能：计算两个矩阵相乘C=A*B
263  输入：用来存计算结果的矩阵C、需要进行乘法计算的两个矩阵A和B
264  输出：计算成功则输出true，失败则false
265  *****/
266  bool MatrixMulMatrix(Matrix *C, Matrix *A, Matrix *B)
267  {

```

```

268     if ((IsNullMatrix(A)) || (IsNullMatrix(B)))
269         return false;
270
271     int A_col = A->column;
272     int B_row = B->row;
273     InitMatrix(C, A->row, B->column);
274     SetMatrixZeros(C);
275
276     if (A_col != B_row)
277     {
278         printf("A_col!=B_row!");
279         return false;
280     }
281
282     for (int i = 0; i < A->row; i++)
283     {
284         for (int j = 0; j < B->column; j++)
285         {
286             for (int k = 0; k < A->column; k++)
287                 C->data[i*C->row + j] += A->data[i*A->row + k] * B->data[k*B->
                    column + j];
288         }
289     }
290
291     return true;
292 }
293
294 int main()
295 {
296     const unsigned NUM = 50; //最大迭代次数, 让数据更准确
297     Matrix mymatrix,temp,temp_Q,temp_R, eValue;
298     int row,col;
299     while (1)
300     {
301         printf("请输入要进行计算的矩阵行与列(以逗号隔开): ");
302         scanf("%d,%d", &row, &col);
303
304         InitMatrix(&mymatrix, row, col);
305         InitMatrix(&temp, row, col);
306         SetMatrixZeros(&temp);
307         SetMatrixZeros(&mymatrix);
308
309         int num = row*col;
310         printf("按照以行的输入顺序依次输入矩阵内的元素, 一共输入%d个元素: ", num);
311         int data;
312         for (int i = 0; i < num; i++)
313         {
314             scanf("%d", &data);

```

```

315     mymatrix.data[i] = data;
316 }
317 printf("输入矩阵如下: \n");
318 PrintMatrix(&mymatrix);
319
320 CopyMatrix(&mymatrix, &temp);
321
322 InitMatrix(&temp_Q, mymatrix.row, mymatrix.column);
323 InitMatrix(&temp_R, mymatrix.row, mymatrix.column);
324 InitMatrix(&eValue, mymatrix.row, 1);
325
326 //使用QR分解求矩阵特征值
327 for (int k = 0; k < NUM; ++k)
328 {
329     QR(&temp, &temp_Q, &temp_R);
330     MatrixMulMatrix(&temp, &temp_R, &temp_Q); //R*Q
331 }
332
333 float result[temp.column];
334 //获取特征值, 将之存储于eValue
335 for (int k = 0; k < temp.column; ++k)
336 {
337     eValue.data[k] = temp.data[k * temp.column + k];
338     result[k] = fabs(temp.data[k * temp.column + k]);
339 }
340
341 SortEigenValues(&eValue, 1); //给特征值排序, 1为升序, 0为降序
342 printf("特征值: \n");
343 PrintMatrix(&eValue);
344
345 float maxn = -1;
346 for (int k = 0; k < temp.column; ++k)
347 {
348     if (result[k] > maxn)
349         maxn = result[k];
350 }
351 printf("矩阵的谱半径 (即特征值绝对值最大) 为: %lf\n", maxn);
352
353 DestroyMatrix(&eValue);
354 DestroyMatrix(&mymatrix);
355 DestroyMatrix(&temp);
356 DestroyMatrix(&temp_Q);
357 DestroyMatrix(&temp_R);
358 }
359 return 0;
360 }

```

1.5 总结

经过这次文章的撰写，我学到了很多知识。首先，对于题目而言，我学到了矩阵的特征值的计算，以及之后得到矩阵的谱半径，这让我又回忆起来线性代数的知识。其次，通过写文章，我对矩阵的谱半径有了更深入的认识，而且，经过程序的编写，我又掌握了矩阵特征值的求法，这个看着简单，但实际写程序是很困难的。最后，我学习了怎么运用 TeXworks 软件写出漂亮的文章，它的排版和对数学公式的美化真的让我很喜欢。

参考文献

- [1] Author. *Wiki*. <https://zh.wikipedia.org/wiki/%E8%B0%B1%E5%8D%8A%E5%BE%84>, 2019.
- [2] Author. *百度百科*. <https://baike.baidu.com/item/%E8%B0%B1%E5%8D%8A%E5%BE%84>, 2019.
- [3] Author. *CSDN*. https://blog.csdn.net/qq_36417014/article/details/83901715, 2019.
- [4] Author. *CSDN*. <https://blog.csdn.net/gsw404/article/details/78684278>, 2019.