

## 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, $\lambda_1,...,\lambda_n$  是其特征值,则 A 的谱半径  $\rho(A) = \max\left\{|\rho_1|,...,|\rho_n|\right\}$  .

- (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  $A \in \mathbb{R}^{n \times n}$  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

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

由此可得:

$$P^{H}A^{H}P = \begin{bmatrix} \bar{\lambda_{1}} & & \\ & \dots & \\ & & \bar{\lambda_{n}} \end{bmatrix}$$
 (1.2)

从而:

$$P^{H}A^{H}AP = \begin{bmatrix} |\lambda_{1}|^{2} & & & \\ & \dots & & \\ & & |\lambda_{n}|^{2} \end{bmatrix}$$
 (1.3)

又显然有:

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

这里 t 是  $\{1,2,...,n\}$  中的某一值,因此有:

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

又因为:

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

所以,可以证明:

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

#### 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 {
  int row;
  int column;
  double * data;
9 Matrix;
10 使用 QR 分解求矩阵特征值
11 for k = 0 \rightarrow 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 \rightarrow 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 \rightarrow temp.column
25 {
26 if (result[k] > maxn)
27 \ maxn = result[k];
28 }
29 PrintMatrix(eValue);
30 print f 矩阵的谱半径(即特征值绝对值最大)为: maxn;
```

#### 1.4 实例分析

```
■ D\Lanr. Hes\CalculateMethod\Tests\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Delta\Pi\Sigma\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dishlor\Dis
```

#### 图 1.1: sample

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

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

```
/****************************
78
   函数功能: 判断一个矩阵是否为空
  输入: 一个矩阵 matrix
   输出: 为空则true, 否则为false
81
   **********
                            ***********
  bool IsNullMatrix(Matrix *matrix)
83
84
     int matrix_num =GetMatrixSize(matrix);
85
     if ((matrix_num <= 0) || (matrix->data == NULL))
86
       return true;
87
88
     else
       return false;
89
  }
90
91
92
   函数功能:释放掉一个矩阵
93
   输入: 一个矩阵 matrix
  输出: 无
95
96
   void DestroyMatrix(Matrix *matrix)
97
98
99
     if (!IsNullMatrix(matrix))
100
       matrix->data = NULL;
101
       matrix->row = 0;
102
       matrix->column = 0;
103
       free(matrix->data);
104
     }
105
  }
106
107
108
   函数功能: 计算一个矩阵的2范数, 即求模
109
   输入: 一个矩阵 matrix
110
   输出: 所求的范数结果norm2_ans
111
   double MatrixNorm2(Matrix *matrix)
113
   {
114
115
     double norm2_ans = 0.0;
     int matrix_num = GetMatrixSize(matrix);
116
     for (int i = 0; i < matrix_num; i++)
117
       norm2_ans+=(matrix->data[i]) * (matrix->data[i]);
118
119
     norm2_ans = (double)sqrt(norm2_ans);
     return norm2 ans;
120
  }
121
122
123
   函数功能: 把一个矩阵复制
124
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
```

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

```
DestroyMatrix(R);
174
                                                   InitMatrix(R, A->row, A->column);
175
                                                   SetMatrixZeros(R);
 176
                                    }
177
                                   else
178
179
                                     {
 180
                                                   R->row = A->row;
                                                   R->column = R->column;
181
                                                   SetMatrixZeros(R);
 182
                                   }
183
 184
                                    //施密特正交化
185
                                    for (int j = 0; j < A \rightarrow column; j++)
 186
187
                                                   for (int i = 0; i < A->column; i++)//把A的第j列存入col_A中
 188
 189
                                                                   col_A.data[i] = A->data[i * A->column + j];
 190
                                                                   col_Q.data[i] = A->data[i * A->column + j];
191
 192
                                                   for (int k = 0; k < j; k++)//计算第j列以前
193
 194
195
                                                                   R->data[k * R->column + j] = 0;
                                                                   for (int i1 = 0; i1 < col_A.row; i1++)
 196
                                                                   \{//R=Q'A(Q'即Q的转置)即Q的第k列和A的第j列做内积
197
                                                                                 R->data[k * R->column + j] += col\_A.data[i1] * Q->data[i1 * Q->column + j] += col\_A.data[i1] * Q->data[i2] * Q->column + j] += col\_A.data[i3] * Q->data[i4] * Q->column + j] += col\_A.data[i4] * Q->data[i4] * Q->column + j] += col\_A.data[i5] * Q->data[i5] * Q->data[i5] * Q->column + j] += col\_A.data[i5] * Q->column + j] += col_A.data[i5] * Q->column + j] +=
 198
                                                                                                         + \mathbf{k}];//Q的第k列
                                                                  }
 199
                                                                   for (int i2 = 0; i2 < A \rightarrow column; i2++)
 200
 201
                                                                                 col_Q.data[i2] -= R->data[k * R->column + j] * Q->data[i2 * Q-> data[i2 * Q-> data[i2] + Q->data[i2] + Q->data[i
 202
                                                                                                   column + k];
203
                                                                  }
                                                   }
 204
205
                                                   double temp = MatrixNorm2(&col_Q);
 206
                                                   R->data[j * R->column + j] = temp;
 207
                                                   for (int i3 = 0; i3 < Q \rightarrow column; i3++)
 208
 209
                                                   {
                                                                   //单位化Q
 210
                                                                   Q->data[i3 * Q->column + j] = col_Q.data[i3] / temp;
211
 212
213
                                   }
214
                                    DestroyMatrix(&col_A);
215
                                     DestroyMatrix(&col_Q);
216
217
                    }
218
```

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

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

```
mymatrix.data[i] = data;
315
          }
316
          printf("输入矩阵如下: \n");
317
          PrintMatrix(&mymatrix);
318
319
          CopyMatrix(&mymatrix, &temp);
320
321
322
          InitMatrix(&temp_Q, mymatrix.row, mymatrix.column);
          InitMatrix(&temp_R, mymatrix.row, mymatrix.column);
323
          InitMatrix(&eValue, mymatrix.row, 1);
324
325
          //使用QR分解求矩阵特征值
326
          for (int k = 0; k < NUM; ++k)
327
          {
328
              QR(&temp, &temp_Q, &temp_R);
329
              MatrixMulMatrix(&temp, &temp_R, &temp_Q); //R*Q
330
          }
331
332
          float result[temp.column];
333
          //获取特征值,将之存储于eValue
334
          for (int k = 0; k < temp.column; ++k)
335
          {
336
              eValue.data[k] = temp.data[k * temp.column + k];
337
              result[k] = fabs(temp.data[k * temp.column + k]);
338
          }
339
340
          SortEigenValues(&eValue, 1);//给特征值排序, 1为升序, 0为降序
341
          printf("特征值: \n");
342
          PrintMatrix(&eValue);
343
344
345
          float maxn = -1;
          for (int k = 0; k < temp.column; ++k)
346
          {
347
              if (result[k] > maxn)
348
349
                 maxn = result[k];
          }
350
          printf("矩阵的谱半径(即特征值绝对值最大)为: %lf(n\n", maxn);
351
352
          DestroyMatrix(&eValue);
353
          DestroyMatrix(&mymatrix);
354
          DestroyMatrix(&temp);
355
356
          DestroyMatrix(&temp_Q);
          DestroyMatrix(&temp_R);
357
       }
358
       return 0;
359
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/gsww404/article/details/78684278, 2019.