

# 云南大学数学与统计学院

## 上机实践报告

课程名称：数值计算实验	年级：2015 级	上机实践成绩：
指导教师：朱娟萍	姓名：刘鹏	
上机实践名称：解线性方程组的迭代法	学号：20151910042	上机实践日期：2017-11-28
上机实践编号：No.02	组号：	最后修改时间：19:51

### 一、实验目的

1. 通过对所学的线性方程组迭代求解的理论方法进行编程，提升程序编写水平；
2. 通过对理论方法的编程实验，进一步掌握理论方法的每一个细节；
3. 通过数值法求解，掌握判断循环终结的条件，理解矩阵范数的存在意义。

### 二、实验内容

1. 编制求矩阵的各种范数的程序；
2. 编程实现用雅可比迭代法求线性方程组的数值解；
3. 编程实现用高斯-塞德尔方法求线性方程组的数值解；
4. 编程实现用 SOR 方法求线性方程组的数值解。

### 三、实验平台

Windows 10 1709 Enterprise 中文版；  
Python 3.6.0；  
Wing IDE Professional 6.0.5-1 集成开发环境；  
MATLAB R2017b win64；  
AxMath 公式编辑器；  
EndNote X8 文献管理。

### 四、实验记录与实验结果分析

#### 4.1 1 题

对下列矩阵计算  $\|\cdot\|_\infty$ ,  $\|\cdot\|_1$ ,  $\|\cdot\|_2^{[1]}$ ：

$$(1) A = \begin{bmatrix} 1 & -1 \\ 2 & 1 \end{bmatrix} \quad (2) B = \begin{bmatrix} 10 & 15 \\ 0 & 1 \end{bmatrix} \quad (3) C = \begin{bmatrix} 0.6 & -0.5 \\ -0.1 & 0.3 \end{bmatrix}$$

解答：

在一般的无大型模块导入的情况下，对于行范数与列范数而言，求解都是比较简单的，但是对于谱范数却并不是很容易，首先一点，难以做到的是特征方程的求解，这个多项式方程，当阶数特别大的时候几乎不能通过公式法求解，而且写一个字符串识别程序也不见得容易。如果能写成，那么在无公式的情况下，用数值方法求解也是不容易的，数值方法的计算量本身就很大，而且得到的一般不是精确解。所以综合来

看，利用特征方程求解的这一做法基本放弃。这里直接调用库函数。

#### 4.1.1 程序代码

```

1  """filename: 4.1 Normal Value.py"""
2
3  """This is a universally usable code to get different
4  kind of normal value of an matrix. And the matrix is
5  a instance of Class Matrix.
6
7  Word is clean, this is the code!
8  """
9
10 class Matrix:
11     """Abatrct class representing a Matrix.
12
13     The internal structure is a two dimension list.
14     """
15     def __init__(self,m,n,mainCol):
16         """
17         self.row:          the row of the Matrix
18         self.col:          the collumn of the Matrix
19         self.CheckedRow:   if one row has been checked,
20                             it should not be checked again
21         self.body:         the internal storing structure
22         self.mainCol:      the coefficient matrix
23         """
24         self.row = m
25         self.col = n
26         self.CheckedRow = set(range(self.row))
27         self.body = [[0 for i in range(n)] for i in range(m)]
28         self.mainCol = mainCol
29
30     def getVal(self):
31         """Giving value to each element of the matrix.
32
33         Overwrite the original value zeros.
34         """
35         for i in range(self.row):
36             for j in range(self.col):
37                 self.body[i][j] = int(input())
38
39     def valid(self,e,kind = None):
40         """If these two matrix are not in the same form, return false,
41         else return true.
42
43         It is useful in the next functions.
44         """
45         if kind == 'multi':
46             # SELF * E

```

```

47         return self.col == e.row
48
49     if self.row != e.row or self.col != e.col:
50         return False
51     else:
52         return True
53
54     def matrixAdd(self,e):
55         """A methon does not used in the pivot PCA algorithm.
56
57         Maybe it will be used in other programs.
58         """
59         self.valid(e)
60         tmp = Matrix(self.row,self.col,self.mainCol)
61         for i in range(self.row): # deep copy
62             for j in range(self.col):
63                 tmp.body[i][j] = self.body[i][j]
64
65         for i in range(self.row):
66             for j in range(self.col):
67                 tmp.body[i][j] += e.body[i][j]
68         return tmp
69
70     def matrixConstMulti(self,const):
71         """A constant number multiple a matrix."""
72         tmp = Matrix(self.row,self.col,self.mainCol)
73         for i in range(self.row): # deep copy
74             for j in range(self.col):
75                 tmp.body[i][j] = self.body[i][j]
76
77         for i in range(self.row):
78             for j in range(self.col):
79                 tmp.body[i][j] *= const
80         return tmp
81
82     def matrixMulti(self,e):
83         """Return the multiplication of two matrix."""
84         self.valid(e,'multi')
85
86         ans = Matrix(self.row,e.col,e.col) # e.col has no meaning
87
88         for i in range(self.row):
89             for j in range(e.col):
90                 tmp = 0
91                 for k in range(self.col):
92                     tmp += self.body[i][k] * e.body[k][j]
93                 ans.body[i][j] = tmp
94
95         return ans

```

```

96
97     def matrixTransform(self,target_row_Number,source_row_Number,times=None):
98         """There is a big problem, every decimal number we see is stored in the
99         RAM with the binary platform.
100
101         I can import the decimal lib to solve this problem, but I did not!
102         """
103         if times == None:    # special case of matrixTransform(TarRow,times)
104
105             times_tmp = source_row_Number
106             for j in range(self.col):
107                 self.body[target_row_Number][j] *= times_tmp
108             return
109         elif times == 'exchange':
110             for i in range(self.col):
111                 self.body[target_row_Number][i],\
112                 self.body[source_row_Number][i] \
113                 = self.body[source_row_Number][i],\
114                 self.body[target_row_Number][i]
115             return
116         else:
117             for i in range(self.col):
118                 self.body[target_row_Number][i] += \
119                 times * self.body[source_row_Number][i]
120
121     def matrixTranspose(self):
122         ans = Matrix(self.col,self.row,self.row)
123         # main column of ans is meaningless
124         for i in range(self.row):
125             for j in range(self.col):
126                 ans.body[j][i] = self.body[i][j]
127         return ans
128
129     def Quick_Sort(L):
130         """A simple code to order a list with quicksort algorithm."""
131         if len(L) <= 1:
132             return L
133
134         else:
135             less = []
136             equal = []
137             bigger = []
138
139             pivot = (L[0] + L[-1] + L[len(L)//2]) / 3
140             # pivot should be taken seriously!
141
142             for i in range(len(L)):
143                 if L[i] < pivot:
144                     less.append(L[i])

```

```

145         elif(L[i] == pivot):
146             equal.append(L(i))
147         else:
148             bigger.append(L[i])
149     less = Quick_Sort(less)
150     bigger = Quick_Sort(bigger)
151
152     return less + equal + bigger
153
154 def GetNormalValue(M,kind=1):
155     """M is a matrix.
156
157     Return the Normal value of M.
158     """
159     if kind == 'inf':
160         """NorVal is the maximum value among the sums of every row. """
161         sums = list()
162         tmp = 0
163         for i in range(M.row):
164             for j in range(M.col):
165                 tmp += M.body[i][j]
166             sums.append(tmp)
167             tmp = 0
168
169         sums = Quick_Sort(sums)
170         return sums[-1]
171
172     elif kind == 1:
173         """NorVal is the maximum value among the sums of every column. """
174         sums = list()
175         tmp = 0
176         for i in range(M.col):
177             for j in range(M.row):
178                 tmp += M.body[j][i]
179             sums.append(tmp)
180             tmp = 0
181
182         sums = Quick_Sort(sums)
183         return sums[-1]
184
185     elif kind == 2:
186         Multi = M.matrixMulti(M.matrixTranspose())
187         """The solve of elg is a little bit difficult,
188         I will change the code after learning.
189
190         Now I use the numpy package to solve it.
191         """
192         import numpy as np
193

```

```

194     MUL = M.matrixMulti(M.matrixTranspose())
195     tmp = np.array(MUL.body)
196
197     a,b = np.linalg.eig(tmp)
198
199     return np.sqrt(max(a))
200
201     else:
202         raise ValueError("""Bad input!\n""")
203
204     """-----my Main Function-----"""
205
206     a = Matrix(2,2,2)
207     print('+-----+')
208     a.body = [[1,-1],[2,1]]
209     b = GetNormalValue(a,'inf')
210     print('| infinity normal value | ',b,' |')
211     b = GetNormalValue(a,1)
212     print('| 1 normal value | ',b,' |')
213     b = GetNormalValue(a,2)
214     print('| 2 normal value | ',b,' |')
215     print('+-----+')
216
217     a.body = [[10,15],[0,1]]
218     b = GetNormalValue(a,'inf')
219     print('| infinity normal value | ',b,' |')
220     b = GetNormalValue(a,1)
221     print('| 1 normal value | ',b,' |')
222     b = GetNormalValue(a,2)
223     print('| 2 normal value | ',b,' |')
224     print('+-----+')
225
226     a.body = [[0.6,-0.5],[-0.1,0.3]]
227     b = GetNormalValue(a,'inf')
228     print('| infinity normal value | ',b,' |')
229     b = GetNormalValue(a,1)
230     print('| 1 normal value | ',b,' |')
231     b = GetNormalValue(a,2)
232     print('| 2 normal value | ',b,' |')
233     print('+-----+')

```

Code Box 1 矩阵的三种范数

#### 4.1.2 输出结果

4.1 Normal Value.py ( Debug process terminated

infinity normal value	3
1 normal value	3
2 normal value	2.30277563773
infinity normal value	25
1 normal value	16
2 normal value	18.0469654611
infinity normal value	0.19999999999999998
1 normal value	0.5
2 normal value	0.827853086715

输出结果 1

### 4.1.3 代码分析

矩阵范数是迭代过程的核心，是判断迭代精度的标尺。

行、列范数都比较简单，但是谱范数比较复杂，参考了书后的 Jacobi 方法之类的迭代法，也没有给出很详细的稳定算法，MATLAB 的代码不开源，只能借用 Python3 的 numpy 进行计算，把 numpy 的方法嵌入到了 GetNormalValue 函数里面做了一个简单的封装。等到以后写出优质的稳定算法，再保留接口替换一下就可以了。

在排序过程中，自建了一个快速排序算法，其中每次的比较数值是三平均数，稳定性比较可靠。无论是随机序列还是等差序列，都可以比较好地进行递归。

## 4.2 2 题

设  $A = \begin{bmatrix} 100 & 99 \\ 99 & 98 \end{bmatrix}$ ，计算  $A$  的条件数  $\text{cond}(A)_\infty$  及  $\text{cond}(A)_2$ 。

解答：

### 4.2.1 程序代码

```

1  """filename: 4.2 Conditional Value.py"""
2
3  """This is a universally usable code to get different
4  kind of normal value of an matrix. And the matrix is
5  a instance of Class Matrix.
6
7  Word is clean, this is the code!
8  """
9
10 class Matrix:
11     """Abatrct class representing a Matrix.
12
13     The internal structure is a two dimension list.
14     """
15     def __init__(self,m,n,mainCol):
16         """

```

```

17         self.row:          the row of the Matrix
18         self.col:          the column of the Matrix
19         self.CheckedRow:   if one row has been checked,
20                             it should not be checked again
21         self.body:         the internal storing structure
22         self.mainCol:      the coefficient matrix
23         """
24         self.row = m
25         self.col = n
26         self.CheckedRow = set(range(self.row))
27         self.body = [[0 for i in range(n)] for i in range(m)]
28         self.mainCol = mainCol
29
30     def getVal(self):
31         """Giving value to each element of the matrix.
32
33         Overwrite the original value zeros.
34         """
35         for i in range(self.row):
36             for j in range(self.col):
37                 self.body[i][j] = int(input())
38
39     def valid(self,e,kind = None):
40         """If these two matrix are not in the same form, return false,
41         else return true.
42
43         It is useful in the next functions.
44         """
45         if kind == 'multi':
46             # SELF * E
47             return self.col == e.row
48
49         if self.row != e.row or self.col != e.col:
50             return False
51         else:
52             return True
53
54     def matrixAdd(self,e):
55         """A methon does not used in the pivot PCA algorithm.
56
57         Maybe it will be used in other programs.
58         """
59         self.valid(e)
60         tmp = Matrix(self.row,self.col,self.mainCol)
61         for i in range(self.row): # deep copy
62             for j in range(self.col):
63                 tmp.body[i][j] = self.body[i][j]
64
65         for i in range(self.row):

```



```

66         for j in range(self.col):
67             tmp.body[i][j] += e.body[i][j]
68         return tmp
69
70     def matrixConstMulti(self,const):
71         """A constant number multiple a matrix."""
72         tmp = Matrix(self.row,self.col,self.mainCol)
73         for i in range(self.row): # deep copy
74             for j in range(self.col):
75                 tmp.body[i][j] = self.body[i][j]
76
77         for i in range(self.row):
78             for j in range(self.col):
79                 tmp.body[i][j] *= const
80         return tmp
81
82     def matrixMulti(self,e):
83         """Return the multiplication of two matrix."""
84         self.valid(e,'multi')
85
86         ans = Matrix(self.row,e.col,e.col) # e.col has no meaning
87
88         for i in range(self.row):
89             for j in range(e.col):
90                 tmp = 0
91                 for k in range(self.col):
92                     tmp += self.body[i][k] * e.body[k][j]
93                 ans.body[i][j] = tmp
94
95         return ans
96
97     def matrixTransform(self,target_row_Number,source_row_Number,times=None):
98         """There is a big problem, every decimal number we see is stored in the
99         RAM with the binary platform.
100
101         I can import the decimal lib to solve this problem, but I did not!
102         """
103         if times == None: # special case of matrixTransform(TarRow,times)
104
105             times_tmp = source_row_Number
106             for j in range(self.col):
107                 self.body[target_row_Number][j] *= times_tmp
108             return
109         elif times == 'exchange':
110             for i in range(self.col):
111                 self.body[target_row_Number][i],\
112                 self.body[source_row_Number][i] \
113                 = self.body[source_row_Number][i],\
114                 self.body[target_row_Number][i]

```

```

115         return
116     else:
117         for i in range(self.col):
118             self.body[target_row_Number][i] += \
119                 times * self.body[source_row_Number][i]
120
121     def matrixTranspose(self):
122         ans = Matrix(self.col,self.row,self.row)
123         # main column of ans is meaningless
124         for i in range(self.row):
125             for j in range(self.col):
126                 ans.body[j][i] = self.body[i][j]
127         return ans
128
129     def Quick_Sort(L):
130         """A simple code to order a list with quicksort algorithm."""
131         if len(L) <= 1:
132             return L
133
134         else:
135             less = []
136             equal = []
137             bigger = []
138
139             pivot = (L[0] + L[-1] + L[len(L)//2]) / 3
140             # pivot should be taken seriously!
141
142             for i in range(len(L)):
143                 if L[i] < pivot:
144                     less.append(L[i])
145                 elif(L[i] == pivot):
146                     equal.append(L[i])
147                 else:
148                     bigger.append(L[i])
149             less = Quick_Sort(less)
150             bigger = Quick_Sort(bigger)
151
152             return less + equal + bigger
153
154     def GetNormalValue(M,kind=2):
155         """M is a matrix.
156
157         Return the Normal value of M.
158         """
159         if kind == 'inf':
160             """NorVal is the maximum value among the sums of every row. """
161             sums = list()
162             tmp = 0
163             for i in range(M.row):

```

```

164         for j in range(M.col):
165             tmp += M.body[i][j]
166             sums.append(tmp)
167             tmp = 0
168
169         sums = Quick_Sort(sums)
170         return sums[-1]
171
172     elif kind == 1:
173         """NorVal is the maximum value among the sums of every column. """
174         sums = list()
175         tmp = 0
176         for i in range(M.col):
177             for j in range(M.row):
178                 tmp += M.body[j][i]
179                 sums.append(tmp)
180                 tmp = 0
181
182         sums = Quick_Sort(sums)
183         return sums[-1]
184
185     elif kind == 2:
186         Multi = M.matrixMulti(M.matrixTranspose())
187         """
188         The solving of eig is a little difficult for me,
189         I will change the code after learning more later.
190
191         Now I use the numpy package to solve it.
192         """
193         import numpy as np
194
195         MUL = M.matrixMulti(M.matrixTranspose())
196         tmp = np.array(MUL.body)
197
198         a,b = np.linalg.eig(tmp)
199
200         return np.sqrt(max(a))
201
202     else:
203         raise ValueError("""Bad input!\n""")
204
205 def GetCondValue(M,kind=2):
206     if kind == 'inf':
207         return GetNormalValue(M,'inf') * GetNormalValue(M.matrixTranspose(),'inf')
208     if kind == 1:
209         return GetNormalValue(M,1) * GetNormalValue(M.matrixTranspose(),1)
210     if kind == 2:
211         return GetNormalValue(M,2) * GetNormalValue(M.matrixTranspose(),2)
212

```

```

213
214 """-----my Main Function-----"""
215
216 A = Matrix(2,2,2)
217 A.body = [[100,99],[99,98]]
218
219 ans_inf = GetCondValue(A,'inf')
220 ans_2 = GetCondValue(A,2)
221 print('+-----+')
222 print('| inf Conditional Value\t| ',ans_inf,'\t\t|')
223 print('| 2 Conditional Value\t| ',ans_2,'\t|')
224 print('+-----+')

```

Code Box 2

#### 4.2.2 输出结果

4.2 Conditional Value. Debug I/O (stdin, stdout, stderr) appears below

```

+-----+
| inf Conditional Value | 39601 |
| 2 Conditional Value | 39205.9999745 |
+-----+

```

输出结果 2

#### 4.2.3 代码分析

条件数是矩阵的范数与矩阵的转置的范数的乘积。

#### 4.3 3 题

用高斯-赛德尔迭代法解下列线性方程组，要求当 $\|x^{(K+1)} - x^{(K)}\| \leq 10^{-5}$ 时迭代终止。

$$\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 \\ -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 6 \\ -2 \\ 6 \end{bmatrix}$$

解答：

##### 4.3.1 程序代码

```

1  """filename: 4.3 Gauss-Seidel Method.py"""
2
3  """filename: 4.4 Plot.py"""
4
5  import matplotlib.pyplot as pl
6  import numpy as np
7

```

```

8  class Matrix:
9      """Abatrct class representing a Matrix.
10
11      The internal structure is a two dimension list.
12      """
13      def __init__(self,m,n,mainCol):
14          """
15          self.row:          the row of the Matrix
16          self.col:          the collumn of the Matrix
17          self.CheckedRow:   if one row has been checked,
18                          it should not be checked again
19          self.body:         the internal storing structure
20          self.mainCol:      the coefficient matrix
21          """
22          self.row = m
23          self.col = n
24          self.CheckedRow = set(range(self.row))
25          self.body = [[0 for i in range(n)] for i in range(m)]
26          self.mainCol = mainCol
27
28      def getVal(self):
29          """Giving value to each element of the matrix.
30
31          Overwrite the original value zeros.
32          """
33          for i in range(self.row):
34              for j in range(self.col):
35                  self.body[i][j] = int(input())
36
37      def valid(self,e,kind = None):
38          """If these two matrix are not in the same form, return false,
39          else return true.
40
41          It is useful in the next functions.
42          """
43          if kind == 'multi':
44              # SELF * E
45              return self.col == e.row
46
47          if self.row != e.row or self.col != e.col:
48              return False
49          else:
50              return True
51
52      def matrixAdd(self,e):
53          """A methon does not used in the pivot PCA algorithm.
54
55          Maybe it will be used in other programs.
56          """

```

```

57     self.valid(e)
58     tmp = Matrix(self.row,self.col,self.mainCol)
59     for i in range(self.row): # deep copy
60         for j in range(self.col):
61             tmp.body[i][j] = self.body[i][j]
62
63     for i in range(self.row):
64         for j in range(self.col):
65             tmp.body[i][j] += e.body[i][j]
66     return tmp
67
68     def matrixConstMulti(self,const):
69         """A constant number multiple a matrix."""
70         tmp = Matrix(self.row,self.col,self.mainCol)
71         for i in range(self.row): # deep copy
72             for j in range(self.col):
73                 tmp.body[i][j] = self.body[i][j]
74
75         for i in range(self.row):
76             for j in range(self.col):
77                 tmp.body[i][j] *= const
78     return tmp
79
80     def matrixMulti(self,e):
81         """Return the multiplication of two matrix."""
82         self.valid(e,'multi')
83
84         ans = Matrix(self.row,e.col,e.col) # e.col has no meaning
85
86         for i in range(self.row):
87             for j in range(e.col):
88                 tmp = 0
89                 for k in range(self.col):
90                     tmp += self.body[i][k] * e.body[k][j]
91                 ans.body[i][j] = tmp
92
93         return ans
94
95     def matrixTransform(self,target_row_Number,source_row_Number,times=None):
96         """There is a big problem, every decimal number we see is stored in the
97         RAM with the binary platform.
98
99         I can import the decimal lib to solve this problem, but I did not!
100        """
101         if times == None: # special case of matrixTransform(TarRow,times)
102
103             times_tmp = source_row_Number
104             for j in range(self.col):
105                 self.body[target_row_Number][j] *= times_tmp

```

```

106         return
107     elif times == 'exchange':
108         for i in range(self.col):
109             self.body[target_row_Number][i],\
110             self.body[source_row_Number][i] \
111             = self.body[source_row_Number][i],\
112             self.body[target_row_Number][i]
113         return
114     else:
115         for i in range(self.col):
116             self.body[target_row_Number][i] += \
117             times * self.body[source_row_Number][i]
118
119     def matrixTranspose(self):
120         ans = Matrix(self.col,self.row,self.row)
121         # main column of ans is meaningless
122         for i in range(self.row):
123             for j in range(self.col):
124                 ans.body[j][i] = self.body[i][j]
125         return ans
126
127     def Quick_Sort(L):
128         """A simple code to order a list with quicksort algorithm."""
129         if len(L) <= 1:
130             return L
131
132         else:
133             less = []
134             equal = []
135             bigger = []
136
137             pivot = (L[0] + L[-1] + L[len(L)//2]) / 3
138             # pivot should be taken seriously!
139
140             for i in range(len(L)):
141                 if L[i] < pivot:
142                     less.append(L[i])
143                 elif(L[i] == pivot):
144                     equal.append(L[i])
145                 else:
146                     bigger.append(L[i])
147             less = Quick_Sort(less)
148             bigger = Quick_Sort(bigger)
149
150             return less + equal + bigger
151
152     def GetNormalValue(M,kind=2):
153         """M is a matrix.

```

```

155     Return the Normal value of M.
156     """
157     if kind == 'inf':
158         """NorVal is the maximum value among the sums of every row. """
159         sums = list()
160         tmp = 0
161         for i in range(M.row):
162             for j in range(M.col):
163                 tmp += M.body[i][j]
164             sums.append(tmp)
165             tmp = 0
166
167         sums = Quick_Sort(sums)
168         return sums[-1]
169
170     elif kind == 1:
171         """NorVal is the maximum value among the sums of every column. """
172         sums = list()
173         tmp = 0
174         for i in range(M.col):
175             for j in range(M.row):
176                 tmp += M.body[j][i]
177             sums.append(tmp)
178             tmp = 0
179
180         sums = Quick_Sort(sums)
181         return sums[-1]
182
183     elif kind == 2:
184         Multi = M.matrixMulti(M.matrixTranspose())
185         """
186         The solving of eig is a little difficult for me,
187         I will change the code after learning more later.
188
189         Now I use the numpy package to solve it.
190         """
191         import numpy as np
192
193         MUL = M.matrixMulti(M.matrixTranspose())
194         tmp = np.array(MUL.body)
195
196         a,b = np.linalg.eig(tmp)
197
198         return np.sqrt(max(a))
199
200     else:
201         raise ValueError("""Bad input!\n""")
202
203 def vectorNormalValue(a,kind=2):

```



```

204     """Get the normal value of a vector."""
205     if kind == 1:
206         ans = 0
207         for i in a:
208             ans += abs(i)
209         return ans
210
211     if kind == 2:
212         from math import sqrt as sq
213         ans = 0
214         for i in a:
215             ans += i * i
216         return sq(ans)
217
218     if kind == 'inf':
219         tmp = list()
220         for i in a:
221             tmp.append(abs(i))
222         Q = Quick_Sort(tmp)
223         return Q[-1]
224
225 def iterFormat(A,b):
226     """Get the iter matrix."""
227     if b.col != 1:
228         return
229     for i in range(A.row):
230         b.body[i][0] = b.body[i][0] / A.body[i][i]
231
232     for i in range(A.row):
233         A.matrixTransform(i,1 / A.body[i][i])
234     B = Matrix(A.row,A.col,A.mainCol)
235     for i in range(B.row):
236         for j in range(B.col):
237             if i == j:
238                 B.body[i][j] = 0
239             else:
240                 B.body[i][j] = -1 * A.body[i][j]
241     ans = (B,b)
242     return ans
243
244 def GS_op1(B,b,x):
245     """simple function."""
246     for i in range(x.row):
247         tmp = 0
248         for j in range(B.col):
249             tmp += B.body[i][j] * x.body[j][0]
250         x.body[i][0] = tmp + b.body[i][0]
251     return x
252

```

```

253 def GS(times):
254     for i in range(times-1):
255         GS_op1(B,b,x)
256
257         tmp = Matrix(6,1,1)
258         for i in range(x.row):
259             tmp.body[i][0] = x.body[i][0] # deep copy
260
261         GS_op1(B,b,x) # do it once more
262
263         step3 = x.matrixConstMulti(-1)
264         there = tmp.matrixAdd(step3)
265
266         c = list()
267
268         for i in range(there.row):
269             c.append(there.body[i][0])
270
271         return vectorNormalValue(c)
272
273 """-----my Main Function-----"""
274
275 A = Matrix(6,6,6)
276 A.body = [[4,-1, 0,-1, 0, 0]\
277           ,[-1, 4,-1, 0,-1, 0]\
278           ,[ 0,-1, 4, 0, 0,-1]\
279           ,[-1, 0, 0, 4,-1, 0]\
280           ,[ 0,-1, 0,-1, 4,-1]\
281           ,[ 0, 0,-1, 0,-1, 4]]
282
283 b = Matrix(6,1,1)
284 b.body = [[0],[5],[0],[6],[-2],[6]]
285
286 x = Matrix(6,1,1)
287 x.body = [[1],[1],[1],[1],[1],[1]]
288
289 tmp = iterFormat(A,b)
290 B,b = tmp[0],tmp[1]
291
292 guide = 10
293 times = 5
294 while guide > 1e-5:
295     guide = GS(times)
296
297 for i in x.body:
298     print(i)

```

Code Box 3

### 4.3.2 输出结果

```
4.3 Gauss-Seidel Itera  Debug process terminated
[0.99999996093388364]
[1.9999999665498392]
[0.9999998576912423]
[1.9999997642151301]
[0.9999997987450249]
[1.9999999141090667]
```

输出结果 3

### 4.3.3 代码分析

高斯-塞德尔方法，无非是在计算过程中，使实时的结果及时反馈，从而提高迭代效率。

## 4.4 4 题

用雅可比迭代法、高斯-赛德尔迭代法解下列线性方程组，比较在不同迭代深度下，两种迭代法的差异。

$$\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 \\ -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 6 \\ -2 \\ 6 \end{bmatrix}$$

解答：

### 4.4.1 程序代码

```
1  """filename: 4.4 Plot.py"""
2
3  import matplotlib.pyplot as pl
4  import numpy as np
5
6  class Matrix:
7      """Abatrct class representing a Matrix.
8
9      The internal structure is a two dimension list.
10     """
11     def __init__(self,m,n,mainCol):
12         """
13         self.row:          the row of the Matrix
14         self.col:          the collumn of the Matrix
15         self.CheckedRow:   if one row has been checked,
16                             it should not be checked again
17         self.body:         the internal storing structure
18         self.mainCol:      the coefficient matrix
19         """
20         self.row = m
```

```

21     self.col = n
22     self.CheckedRow = set(range(self.row))
23     self.body = [[0 for i in range(n)] for i in range(m)]
24     self.mainCol = mainCol
25
26     def getVal(self):
27         """Giving value to each element of the matrix.
28
29         Overwrite the original value zeros.
30         """
31         for i in range(self.row):
32             for j in range(self.col):
33                 self.body[i][j] = int(input())
34
35     def valid(self,e,kind = None):
36         """If these two matrix are not in the same form, return false,
37         else return true.
38
39         It is useful in the next functions.
40         """
41         if kind == 'multi':
42             # SELF * E
43             return self.col == e.row
44
45         if self.row != e.row or self.col != e.col:
46             return False
47         else:
48             return True
49
50     def matrixAdd(self,e):
51         """A methon does not used in the pivot PCA algorithm.
52
53         Maybe it will be used in other programs.
54         """
55         self.valid(e)
56         tmp = Matrix(self.row,self.col,self.mainCol)
57         for i in range(self.row): # deep copy
58             for j in range(self.col):
59                 tmp.body[i][j] = self.body[i][j]
60
61         for i in range(self.row):
62             for j in range(self.col):
63                 tmp.body[i][j] += e.body[i][j]
64         return tmp
65
66     def matrixConstMulti(self,const):
67         """A constant number multiple a matrix."""
68         tmp = Matrix(self.row,self.col,self.mainCol)
69         for i in range(self.row): # deep copy

```

```

70         for j in range(self.col):
71             tmp.body[i][j] = self.body[i][j]
72
73     for i in range(self.row):
74         for j in range(self.col):
75             tmp.body[i][j] *= const
76     return tmp
77
78 def matrixMulti(self,e):
79     """Return the multiplication of two matrix."""
80     self.valid(e,'multi')
81
82     ans = Matrix(self.row,e.col,e.col) # e.col has no meaning
83
84     for i in range(self.row):
85         for j in range(e.col):
86             tmp = 0
87             for k in range(self.col):
88                 tmp += self.body[i][k] * e.body[k][j]
89             ans.body[i][j] = tmp
90
91     return ans
92
93 def matrixTransform(self,target_row_Number,source_row_Number,times=None):
94     """There is a big problem, every decimal number we see is stored in the
95     RAM with the binary platform.
96
97     I can import the decimal lib to solve this problem, but I did not!
98     """
99     if times == None: # special case of matrixTransform(TarRow,times)
100
101         times_tmp = source_row_Number
102         for j in range(self.col):
103             self.body[target_row_Number][j] *= times_tmp
104         return
105     elif times == 'exchange':
106         for i in range(self.col):
107             self.body[target_row_Number][i],\
108             self.body[source_row_Number][i] \
109             = self.body[source_row_Number][i],\
110             self.body[target_row_Number][i]
111         return
112     else:
113         for i in range(self.col):
114             self.body[target_row_Number][i] += \
115             times * self.body[source_row_Number][i]
116
117 def matrixTranspose(self):
118     ans = Matrix(self.col,self.row,self.row)

```

```

119     # main column of ans is meaningless
120     for i in range(self.row):
121         for j in range(self.col):
122             ans.body[j][i] = self.body[i][j]
123     return ans
124
125 def Quick_Sort(L):
126     """A simple code to order a list with quicksort algorithm."""
127     if len(L) <= 1:
128         return L
129
130     else:
131         less = []
132         equal = []
133         bigger = []
134
135         pivot = (L[0] + L[-1] + L[len(L)//2]) / 3
136         # pivot should be taken seriously!
137
138         for i in range(len(L)):
139             if L[i] < pivot:
140                 less.append(L[i])
141             elif(L[i] == pivot):
142                 equal.append(L[i])
143             else:
144                 bigger.append(L[i])
145         less = Quick_Sort(less)
146         bigger = Quick_Sort(bigger)
147
148         return less + equal + bigger
149
150 def GetNormalValue(M,kind=2):
151     """M is a matrix.
152
153     Return the Normal value of M.
154     """
155     if kind == 'inf':
156         """NorVal is the maximum value among the sums of every row. """
157         sums = list()
158         tmp = 0
159         for i in range(M.row):
160             for j in range(M.col):
161                 tmp += M.body[i][j]
162             sums.append(tmp)
163             tmp = 0
164
165         sums = Quick_Sort(sums)
166         return sums[-1]
167

```

```

168     elif kind == 1:
169         """NorVal is the maximum value among the sums of every column. """
170         sums = list()
171         tmp = 0
172         for i in range(M.col):
173             for j in range(M.row):
174                 tmp += M.body[j][i]
175             sums.append(tmp)
176             tmp = 0
177
178         sums = Quick_Sort(sums)
179         return sums[-1]
180
181     elif kind == 2:
182         Multi = M.matrixMulti(M.matrixTranspose())
183         """
184         The solving of eig is a little difficult for me,
185         I will change the code after learning more later.
186
187         Now I use the numpy package to solve it.
188         """
189         import numpy as np
190
191         MUL = M.matrixMulti(M.matrixTranspose())
192         tmp = np.array(MUL.body)
193
194         a,b = np.linalg.eig(tmp)
195
196         return np.sqrt(max(a))
197
198     else:
199         raise ValueError("""Bad input!\n""")
200
201 def vectorNormalValue(a,kind=2):
202     """Get the normal value of a vector."""
203     if kind == 1:
204         ans = 0
205         for i in a:
206             ans += abs(i)
207         return ans
208
209     if kind == 2:
210         from math import sqrt as sq
211         ans = 0
212         for i in a:
213             ans += i * i
214         return sq(ans)
215
216     if kind == 'inf':

```

```

217     tmp = list()
218     for i in a:
219         tmp.append(abs(i))
220     Q = Quick_Sort(tmp)
221     return Q[-1]
222
223 def iterFormat(A,b):
224     """Get the iter matrix."""
225     if b.col != 1:
226         return
227     for i in range(A.row):
228         b.body[i][0] = b.body[i][0] / A.body[i][i]
229
230     for i in range(A.row):
231         A.matrixTransform(i,1 / A.body[i][i])
232     B = Matrix(A.row,A.col,A.mainCol)
233     for i in range(B.row):
234         for j in range(B.col):
235             if i == j:
236                 B.body[i][j] = 0
237             else:
238                 B.body[i][j] = -1 * A.body[i][j]
239     ans = (B,b)
240     return ans
241
242 def GS_op1(B,b,x):
243     """simple function."""
244     for i in range(x.row):
245         tmp = 0
246         for j in range(B.col):
247             tmp += B.body[i][j] * x.body[j][0]
248         x.body[i][0] = tmp + b.body[i][0]
249         # x is changed during one iteration
250     return x
251
252 def relax_op1(B,b,x):
253     """simple function."""
254     for i in range(x.row):
255         tmp = 0
256         for j in range(B.col):
257             tmp += B.body[i][j] * x.body[j][0]
258         x.body[i][0] = tmp + b.body[i][0]
259         # x is changed during one iteration
260     return x
261
262 class Plot:
263     """Just for testing."""
264     def __init__(self):
265         """Initialize this class.

```



```

266
267     As you can see, this function could not be universally used.
268     """
269     A = Matrix(6,6,6)
270     A.body = [[4,-1, 0,-1, 0, 0]\
271               ,[-1, 4,-1, 0,-1, 0]\
272               ,[ 0,-1, 4, 0, 0,-1]\
273               ,[-1, 0, 0, 4,-1, 0]\
274               ,[ 0,-1, 0,-1, 4,-1]\
275               ,[ 0, 0,-1, 0,-1, 4]]
276
277     b = Matrix(6,1,1)
278     b.body = [[0],[5],[0],[6],[-2],[6]]
279     tmp = iterFormat(A,b)
280
281     ...
282     self.B:           the matrix to iterate
283     self.b:           the matrix with only one column
284     self.x:           the original value we give,
285                       Default set it [0,0,0,0,0,0]'
286     ...
287     self.B = tmp[0]
288     self.b = tmp[1]
289     self.x = Matrix(6,1,1)
290
291     def Jacobi(self,times):
292         self.__init__()
293         for i in range(times-1):
294             step1 = self.B.matrixMulti(self.x)
295             step2 = step1.matrixAdd(self.b)
296             self.x = step2
297
298         lastStep = Matrix(self.x.row,self.x.col,self.x.mainCol)
299         for i in range(self.x.row):
300             lastStep.body[i][0] = self.x.body[i][0]
301
302         step1 = self.B.matrixMulti(lastStep)
303         lastStep = step1.matrixAdd(self.b)
304
305         step3 = lastStep.matrixConstMulti(-1)
306         there = step2.matrixAdd(step3)
307
308         c = list()
309         for i in range(there.row):
310             c.append(there.body[i][0])
311
312         return vectorNormalValue(c)
313
314     def GS(self,times):

```

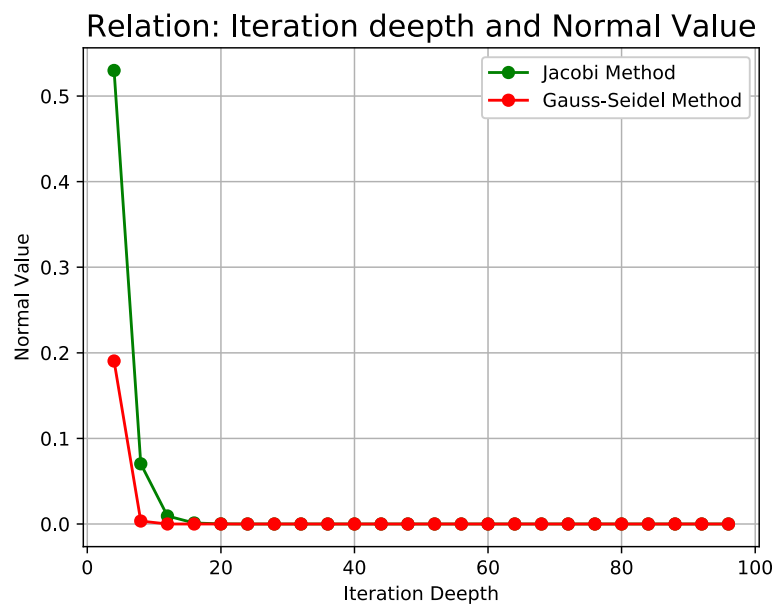
```

315     self.__init__()
316
317     for i in range(times-1):
318         GS_op1(self.B,self.b,self.x)
319
320     tmp = Matrix(6,1,1)
321     for i in range(self.x.row):
322         tmp.body[i][0] = self.x.body[i][0] # deep copy
323
324     GS_op1(self.B,self.b,self.x) # do it once more
325
326     step3 = self.x.matrixConstMulti(-1)
327     there = tmp.matrixAdd(step3)
328
329     c = list()
330
331     for i in range(there.row):
332         c.append(there.body[i][0])
333
334     return vectorNormalValue(c)
335
336 def plot(self,n=100):
337     """Default iteration depth is 100."""
338
339     x_jacobi = []
340     y_jacobi = []
341     for i in range(4,n,4):
342         x_jacobi.append(i)
343         y_jacobi.append(self.Jacobi(i))
344
345     x_gs = []
346     y_gs = []
347     for i in range(4,n,4):
348         x_gs.append(i)
349         y_gs.append(self.GS(i))
350
351     pl.grid()
352     pl.title('Relation: Iteration depth and Normal Value',fontsize=16)
353     pl.plot(x_jacobi,y_jacobi,'o-g',label='Jacobi Method')
354     pl.plot(x_gs,y_gs,'o-r',label='Gauss-Seidel Method')
355     pl.legend()
356     pl.xlabel('Iteration Depth')
357     pl.ylabel('Normal Value')
358
359     pl.show()
360
361 M = Plot()
362 M.plot()

```

Code Box 4

#### 4.4.2 输出结果



输出结果 4

#### 4.4.3 代码分析

可以发现，在相同的迭代次数下，高斯-赛德尔方法的精度更高。

#### 4.5 5 题

用松弛法解

$$\begin{cases} 4x_1 - x_2 = 1 \\ -x_1 + 4x_2 - x_3 = 4 \\ -x_2 + 4x_3 = -3 \end{cases}$$

分别取 $\omega = 1.03$ ,  $\omega = 1$ ,  $\omega = 1.1$ 。要求当 $\|x^{(K)} - x^{(K-1)}\| < 5 \times 10^{-6}$ 时迭代终止，并对每个 $\omega$ 值确定迭代次数（初值 $x^{(0)} = (0, 0, 0)^T$ ）。

解答：

可以从更高的观点来看这个问题。下面编程作图以比较。

##### 4.5.1 程序代码

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Tue Nov 28 18:30:53 2017
4
5  @author: Newton
6  """
7
8  """filename: 4.5 Relaxation Method.py"""
9
10 import matplotlib.pyplot as pl

```

```

11 import numpy as np
12
13 class Matrix:
14     """Abatrct class representing a Matrix.
15
16     The internal structure is a two dimension list.
17     """
18     def __init__(self,m,n,mainCol = None):
19         """
20         self.row:         the row of the Matrix
21         self.col:         the collumn of the Matrix
22         self.CheckedRow:   if one row has been checked,
23                           it should not be checked again
24         self.body:         the internal storing structure
25         self.mainCol:      the coefficient matrix
26         """
27         self.row = m
28         self.col = n
29         self.CheckedRow = set(range(self.row))
30         self.body = [[0 for i in range(n)] for i in range(m)]
31         self.mainCol = mainCol
32
33     def getVal(self):
34         """Giving value to each element of the matrix.
35
36         Overwrite the original value zeros.
37         """
38         for i in range(self.row):
39             for j in range(self.col):
40                 self.body[i][j] = input()
41
42     def valid(self,e,kind = None):
43         """If these two matrices are not in the right form, return false,
44         else return true.
45
46         It is useful in the next functions.
47         """
48         if kind == 'multi':
49             # SELF * E
50             return self.col == e.row
51
52         if self.row != e.row or self.col != e.col:
53             return False
54         else:
55             return True
56
57     def matrixAdd(self,e):
58         """One matrix add to another and generate a new one.
59

```

```

60         The original one would not change.
61         """
62         self.valid(e)
63         # if e and self are not in the same form, return false.
64
65         ans = Matrix(self.row,self.col,self.mainCol)
66
67         for i in range(self.row):
68             for j in range(self.col):
69                 ans.body[i][j] = self.body[i][j] + e.body[i][j]
70
71         return ans
72
73     def matrixConstMulti(self,const):
74         """A constant number multiple a matrix."""
75
76         ans = Matrix(self.row,self.col,self.mainCol)
77
78         for i in range(self.row):
79             for j in range(self.col):
80                 ans.body[i][j] = self.body[i][j] * const
81
82         return ans
83
84     def matrixMulti(self,e):
85         """Return the multiplication of two matrices.
86
87         Attention! ans = self * e, not e * self.
88         """
89
90         self.valid(e,'multi')
91         # if e and self could not make multiplication, return false
92
93         ans = Matrix(self.row, e.col, e.col)
94
95         for i in range(self.row):
96             for j in range(e.col):
97                 tmp = 0
98                 for k in range(self.col):
99                     tmp += self.body[i][k] * e.body[k][j]
100                 ans.body[i][j] = tmp
101
102         return ans
103
104     def matrixTransform(self,target_row,source_row,times=None):
105         """
106         case 1:    two coefficient, make coe(1) row times coe(2)
107         case 2:    exchange the two rows with 'exchange' reminding
108         case 3:    coe(1) rows add the coe(3) times of coe(2) row.

```

```

109
110     """
111
112     ans = Matrix(self.row, self.col, self.mainCol)
113     for i in range(self.row): # deep copy
114         for j in range(self.col):
115             ans.body[i][j] = self.body[i][j]
116
117     if times == None: # special case of matrixTransform(TarRow,times)
118         times_tmp = source_row
119         for j in range(self.col):
120             ans.body[target_row][j] = \
121                 ans.body[target_row][j] * times_tmp
122
123     elif times == 'exchange':
124         for i in range(self.col):
125             ans.body[target_row][i], ans.body[source_row][i] \
126                 = ans.body[source_row][i], ans.body[target_row][i]
127
128     else:
129         for i in range(self.col):
130             ans.body[target_row][i] += \
131                 times * ans.body[source_row][i]
132
133     return ans
134
135 def matrixTranspose(self):
136     """Generate a new matrix which is the transpose of the old one."""
137
138     ans = Matrix(self.col,self.row,self.row)
139     # main column of ans is meaningless
140
141     for i in range(self.row):
142         for j in range(self.col):
143             ans.body[j][i] = self.body[i][j]
144     return ans
145
146 def matrixNormVal(self, kind = 2):
147     """
148     Return the Normal value of this matrix.
149     """
150
151     if kind == 'inf':
152         """NorVal is the maximum value among the sums of every row. """
153         sums = list()
154         tmp = 0
155         for i in range(self.row):
156             for j in range(self.col):
157                 tmp += self.body[i][j]

```

```

158         sums.append(tmp)
159         tmp = 0
160
161         sums = Quick_Sort(sums)      # quick sort algorithm
162         return sums[-1]
163
164     elif kind == 1:
165         """NorVal is the maximum value among the sums of every column. """
166         sums = list()
167         tmp = 0
168         for i in range(self.col):
169             for j in range(self.row):
170                 tmp += self.body[j][i]
171             sums.append(tmp)
172             tmp = 0
173
174         sums = Quick_Sort(sums)
175         return sums[-1]
176
177     elif kind == 2:
178         """
179         The solving of eig is a little difficult for me,
180         I will change the code after learning more later.
181
182         Now I use the numpy package to solve it.
183         """
184         import numpy as np
185
186         MUL = self.matrixMulti(self.matrixTranspose())
187         tmp = np.array(MUL.body)
188
189         a,b = np.linalg.eig(tmp)
190
191         return np.sqrt(max(a))
192
193     else:
194         raise ValueError("""Bad input!\n""")
195
196     def matrixInversion(self):
197         """Generate a new matrix which is the old one's inversion."""
198
199         Max = 0                # initialize the variable
200         position_row = 0       # the row number of the maximum value
201         position_col = 0
202
203
204         ans = Matrix(self.row, self.col, self.mainCol)
205         for i in range(self.row):
206             for j in range(self.col):

```

```

207         ans.body[i][j] = self.body[i][j]
208
209     eyes = Matrix(self.row, self.col, self.mainCol)
210     for i in range(eyes.row):
211         eyes.body[i][i] = 1
212         ans.body[i] += eyes.body[i]
213
214     ans.mainCol = ans.col
215     ans.col *= 2
216
217     for i in range(ans.row):
218         for j in ans.CheckedRow:
219             for k in range(ans.mainCol): # not in all the columns
220                 if abs(Max) <= abs(ans.body[j][k]):
221                     Max = ans.body[j][k]
222                     position_row = j
223                     position_col = k
224
225     ans = ans.matrixTransform(position_row, 1 / Max)
226
227     ans.CheckedRow.remove(position_row)
228
229     for j in range(ans.row):
230         if j != position_row:
231             ans = ans.matrixTransform\
232                 (j, position_row, -1 * ans.body[j][position_col])
233
234     Max = 0
235     position_row = 0
236     position_col = 0
237
238     begin = 0
239     for j in range(ans.mainCol):
240         for i in range(ans.row):
241             if ans.body[i][j] == 1:
242                 ans = ans.matrixTransform(begin, i, 'exchange')
243                 begin += 1
244
245     new = Matrix(self.row, self.row, None)
246     for i in range(new.row):
247         for j in range(new.col):
248             new.body[i][j] = ans.body[i][j+ans.row]
249     ans = new
250     return ans
251
252 def Quick_Sort(L):
253     """A simple code to order a list with quicksort algorithm."""
254
255     if len(L) <= 1:

```



```

256         return L
257
258     else:
259         less = []
260         equal = []
261         bigger = []
262
263         pivot = (L[0] + L[-1] + L[len(L)//2]) / 3
264         # pivot should be taken seriously!
265
266         for i in range(len(L)):
267             if L[i] < pivot:
268                 less.append(L[i])
269             elif(L[i] == pivot):
270                 equal.append(L[i])
271             else:
272                 bigger.append(L[i])
273         less = Quick_Sort(less)
274         bigger = Quick_Sort(bigger)
275
276         return less + equal + bigger
277
278     #-----new class-----
279
280     class MatrixIterMethods:
281         """This class includes three methods which could be used in the
282         solving of linear equations.
283
284         Method 1:    Jabobi method
285         Method 2:    Gauss-Seidel method
286         Method 3:    Relaxation method
287
288         All these three methods will return x and the iteration depth.
289         """
290
291         def __init__(self,A,b,x0,omega=1):
292             """Initialize this class.
293
294             A:    coefficient matrix
295             b:    where Ax = b
296             x0:    the original value of x we choose
297
298             These global variables would not change after init operation.
299             """
300
301             if omega == 1:
302                 if b.col != 1:
303                     raise ValueError('Bad inputs, please Check it!')
304

```

```

305     tmp_b = Matrix(b.row, b.col, b.mainCol)
306     for i in range(b.row):
307         for j in range(b.col):
308             tmp_b.body[i][j] = b.body[i][j]
309
310     for i in range(tmp_b.row):
311         tmp_b.body[i][0] = (tmp_b.body[i][0]) / A.body[i][i]
312
313     tmp_A = Matrix(A.row, A.col, A.mainCol)
314     for i in range(A.row):
315         for j in range(A.col):
316             tmp_A.body[i][j] = A.body[i][j]
317
318     for i in range(A.row):
319         tmp_A = tmp_A.matrixTransform(i, 1 / A.body[i][i])
320
321     B = Matrix(A.row, A.col, A.mainCol)
322     for i in range(B.row):
323         for j in range(B.col):
324             if i == j:
325                 B.body[i][j] = 0
326             else:
327                 B.body[i][j] = -1 * tmp_A.body[i][j]
328     ans = (B, tmp_b)
329
330 else:
331     L = Matrix(A.row, A.col, A.mainCol)
332     for i in range(A.row):
333         for j in range(A.col):
334             L.body[i][j] = A.body[i][j]
335             if i <= j:
336                 L.body[i][j] = 0
337
338     U = Matrix(A.row, A.col, A.mainCol)
339     for i in range(A.row):
340         for j in range(A.col):
341             U.body[i][j] = A.body[i][j]
342             if i >= j:
343                 U.body[i][j] = 0
344
345     D = Matrix(A.row, A.col, A.mainCol)
346     for i in range(A.row):
347         for j in range(A.col):
348             D.body[i][j] = A.body[i][j]
349             if i != j:
350                 D.body[i][j] = 0
351
352     step1 = L.matrixConstMulti(omega)
353     step2 = D.matrixAdd(step1)

```

```

354         step3 = step2.matrixInversion() # mark
355
356         step4 = D.matrixConstMulti(-1 * omega)
357         step5 = U.matrixConstMulti(-1 * omega)
358         step6 = D.matrixAdd(step4)
359         step7 = step6.matrixAdd(step5) # mark
360
361         step8 = b.matrixConstMulti(omega)
362
363         end_B = step3.matrixMulti(step7)
364         end_b = step3.matrixMulti(step8)
365
366         ans = (end_B, end_b)
367
368     self.B = ans[0]
369     self.b = ans[1]
370     self.x = Matrix(x0.row, x0.col, x0.mainCol)
371
372     for i in range(x0.row):
373         for j in range(x0.col):
374             self.x.body[i][j] = x0.body[i][j]
375
376     def GS_op1(self):
377         """Do one time iteration with GS method or Relaxation method.
378
379         self.x will be changed after this operation.
380         """
381         for i in range(self.x.row):
382             tmp = 0
383             for j in range(self.B.col):
384                 tmp += self.B.body[i][j] * self.x.body[j][0]
385             self.x.body[i][0] = tmp + self.b.body[i][0]
386
387     def jacobiMethod(self, accuracy):
388         """The iteration would not stop by iter times.
389
390         It will stop while the accuracy gets.
391         """
392         self.__init__(A, b, x0) # omega uses the default setting
393
394         save_tmp = self.x.matrixConstMulti(-1)
395         # save the original value for comparing.
396
397         step1 = self.B.matrixMulti(self.x)
398         step2 = step1.matrixAdd(self.b)
399
400         self.x = step2
401         # update the value of self.x
402

```

```

403     p = step2.matrixAdd(save_tmp)
404     accu = p.matrixNormVal()
405
406     iter_depth = 1
407
408     while(accu > accuracy):
409         save_tmp = self.x.matrixConstMulti(-1)
410         # save the original value for comparing.
411
412         step1 = self.B.matrixMulti(self.x)
413         step2 = step1.matrixAdd(self.b)
414
415         self.x = step2
416         # update the value of self.x
417
418         p = step2.matrixAdd(save_tmp)
419         accu = p.matrixNormVal()
420
421         iter_depth += 1
422
423     return (self.x, iter_depth)
424
425 def gsMethod(self, accuracy):
426     """Iteration will stop when the accuracy gets."""
427
428     self.__init__(A,b,x0)
429
430     save_tmp = self.x.matrixConstMulti(-1)
431     # save the original value for comparing.
432
433     self.GS_op1()
434     # self.x has been changed auto
435
436     p = self.x.matrixAdd(save_tmp)
437     accu = p.matrixNormVal()
438
439     iter_depth = 1
440
441     while(accu > accuracy):
442         save_tmp = self.x.matrixConstMulti(-1)
443         # save the original value for comparing.
444
445         self.GS_op1()
446         # self.x has been changed
447
448         p = self.x.matrixAdd(save_tmp)
449         accu = p.matrixNormVal()
450
451         iter_depth += 1

```

```

452
453     return(self.x, iter_depth)
454
455 def relaxMethod(self, omega, accuracy):
456     """Iteration will stop when the accuracy gets."""
457
458     self.__init__(A,b,x0, omega)  # omega uses the default setting
459
460     save_tmp = self.x.matrixConstMulti(-1)
461     # save the original value for comparing.
462
463     step1 = self.B.matrixMulti(self.x)
464     step2 = step1.matrixAdd(self.b)
465
466     self.x = step2
467     # update the value of self.x
468
469     p = step2.matrixAdd(save_tmp)
470     accu = p.matrixNormVal()
471
472     iter_depth = 1
473
474     while(accu > accuracy):
475         save_tmp = self.x.matrixConstMulti(-1)
476         # save the original value for comparing.
477
478         step1 = self.B.matrixMulti(self.x)
479         step2 = step1.matrixAdd(self.b)
480
481         self.x = step2
482         # update the value of self.x
483
484         p = step2.matrixAdd(save_tmp)
485         accu = p.matrixNormVal()
486
487         iter_depth += 1
488     return (self.x, iter_depth)
489
490 def Plot(A, b, x0, e=1e-16):
491     """A pure function for plotting.
492
493     e is the target accuracy.
494     """
495
496     ERROR = list()
497     tmp = 1
498     while tmp > e:
499         tmp /= 10
500         ERROR.append(tmp)

```

```

501
502     depth_1 = list()
503     for i in ERROR:
504         J = MatrixIterMethods(A, b, x0)
505         M_1 = J.relaxMethod(1.0, i)
506         # Relaxation factor is 1.0, equal to GSM
507
508         depth_1.append(M_1[1])
509
510     depth_2 = list()
511     for i in ERROR:
512         M_1 = J.relaxMethod(1.03, i)
513         # Relaxation factor is 1.03
514         depth_2.append(M_1[1])
515
516     depth_3 = list()
517     for i in ERROR:
518         M_1 = J.relaxMethod(1.1, i)
519         # Relaxation factor is 1.1
520         depth_3.append(M_1[1])
521
522     import math
523     for i in range(len(ERROR)):
524         ERROR[i] = -1 * math.log10(ERROR[i])
525
526     pl.grid()
527     pl.title("Same Accuracy: Iteration depth and Omega's Value", fontsize=16)
528     pl.plot(ERROR, depth_1, 'o-r', label = 'omega is 1.00')
529     pl.plot(ERROR, depth_2, 'o-g', label = 'omega is 1.03')
530     pl.plot(ERROR, depth_3, 'o-b', label = 'omega is 1.10')
531     pl.legend()
532     pl.xlabel('Level of Accuracy')
533     pl.ylabel('Iteration Depth')
534     pl.show()
535
536 if __name__ == "__main__":
537
538     A = Matrix(6, 6)
539     A.body = [[4, -1, 0, -1, 0, 0]\
540              ,[-1, 4, -1, 0, -1, 0]\
541              ,[ 0, -1, 4, 0, 0, -1]\
542              ,[-1, 0, 0, 4, -1, 0]\
543              ,[ 0, -1, 0, -1, 4, -1]\
544              ,[ 0, 0, -1, 0, -1, 4]]
545
546     b = Matrix(6, 1)
547     b.body = [[0],[5],[0],[6],[-2],[6]]
548
549     x0 = Matrix(6, 1)

```

```

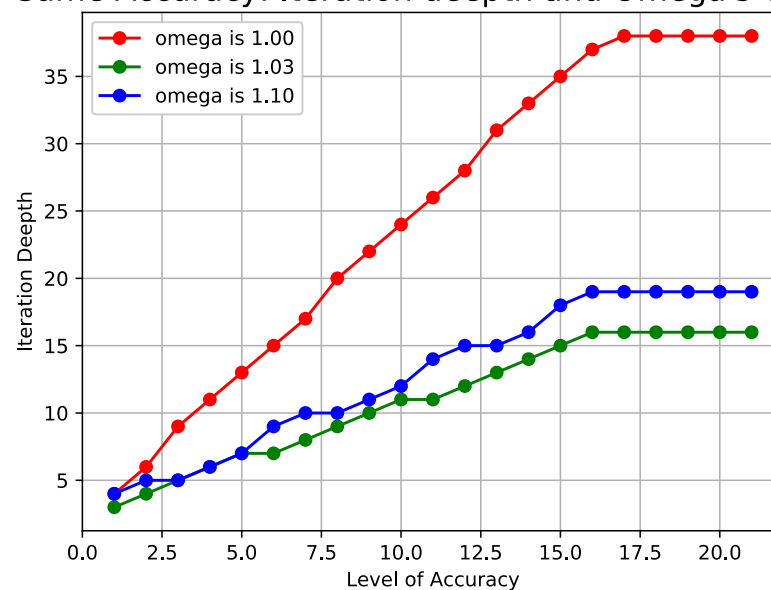
550 x0.body = [[1],[1],[1],[1],[1],[1]]
551
552 Plot(A, b, x0, 1e-20)
553
554 A = Matrix(3, 3)
555 A.body = [[4, -1, 0]\
556           ,[-1, 4, -1]\
557           ,[ 0, -1, 4]]
558
559 b = Matrix(3, 1)
560 b.body = [[1],[4],[-3]]
561
562 x0 = Matrix(3, 1)
563 x0.body = [[1],[1],[1]]
564
564 Plot(A, b, x0, 1e-20)

```

Code Box 5

#### 4.5.2 输出结果

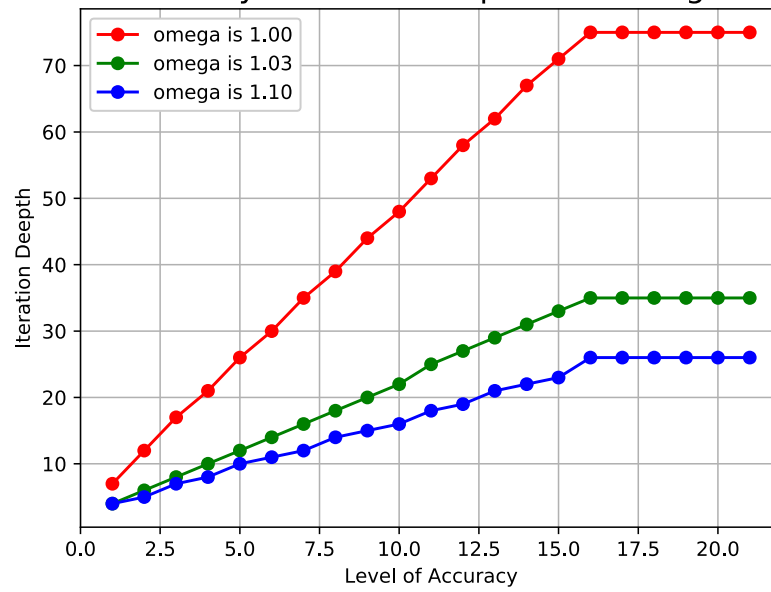
Same Accuracy: Iteration depth and Omega's Value



输出结果 5 三阶矩阵

其实，还可以对第 4 题进行作图比较：

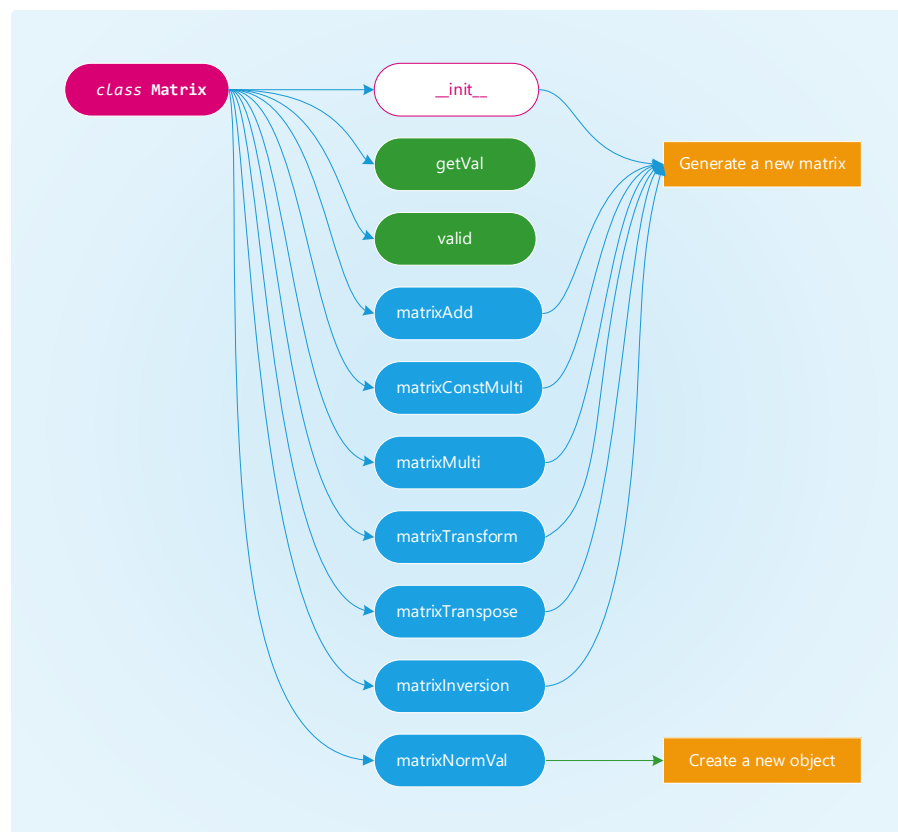
Same Accuracy: Iteration depth and Omega's Value



输出结果 6 六阶矩阵

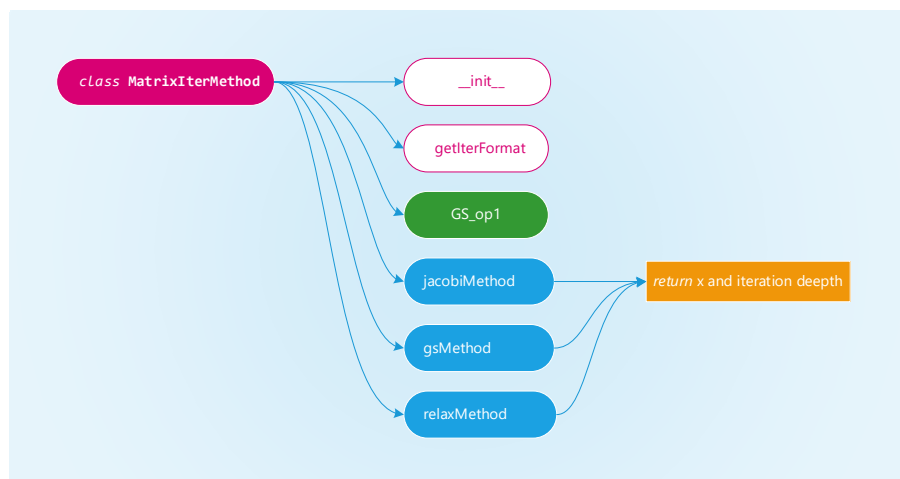
可以发现，收敛速度最快的松弛因子，对于不同矩阵来说是不一样的。

### 4.5.3 代码分析



Code Analysis 1





Code Analysis 2

本题代码行数比较多，而且涉及到了多个类的交互，所以对于参数的设计、传递，都有着巨大的考验，为了能够顺利构建出最终的图像，必须把 `class Matrix` 的对外行为设计得合乎规范，这样才能保证在调用矩阵的方法时毫无差错。可以从 Code Analysis 1 中看到，矩阵的所有方法，都没有改变该矩阵，换言之，这个类是不可更改的。这要求调用的时候，必须参考这一原则，写清楚赋值语句。而第二个类 `class MatrixIterMethod` 中，三个方法都列入了其中，虽然本题从实质上并没有调用雅可比方法，不过从先前的实验中可以看到，相同的精度下，它需要的迭代次数更多。

本段代码的更改经历了很长的时间，最终发现只有深刻理解才能写出程序。超松弛迭代法有着很深刻的数学经验在里面，而且在这里，超松弛迭代也经历了一个从一般分量形式到矩阵形式的转换。从时间统筹角度看，采用 MATLAB 进行编程会更加方便，也可以留出更多的时间来进行理论学习。

## 五、实验体会

从某种意义上讲，本次实验选错了语言，可能用基于矩阵的 MATLAB 会更加方便，而 Python 的 numpy 并不支持原生运算符，所以还是存在一定的局限性。

本试验报告的所有数据都经过 MATLAB 的验证，俱无问题。

如果有可能，在以后的实验报告中我将采用 MATLAB 进行编程。

## 六、参考文献

- [1] 金一庆, 陈越, 王冬梅. 数值方法[M]. 北京: 机械工业出版社; 2000.2.
- [2] 黄金伟. 矩阵的特征值与特征向量的简易求法[J]. 福建信息技术教育. 2006.