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 |

# 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。

# 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。

# 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。

# 实验记录与实验结果分析

## 1题

对下列矩阵计算，，[1] ：

（1） （2） （3）

**解答：**

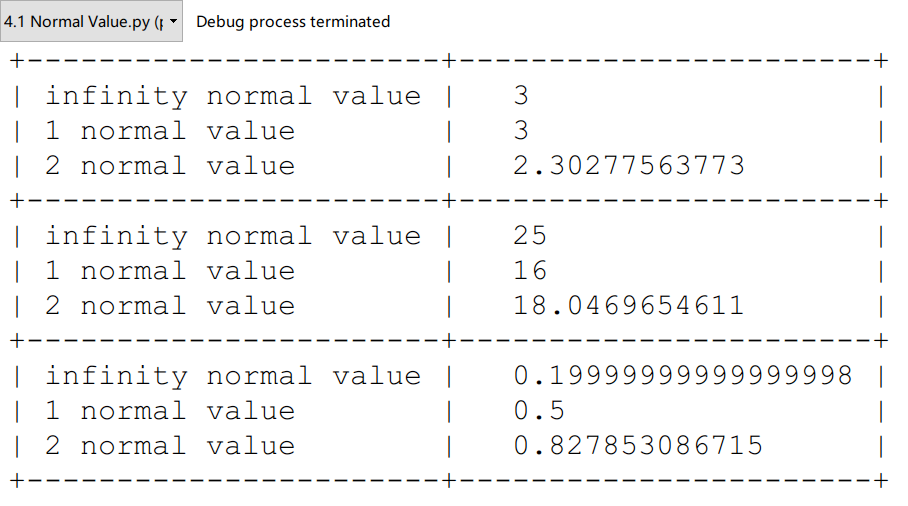
在一般的无大型模块导入的情况下，对于行范数与列范数而言，求解都是比较简单的，但是对于谱范数却并不是很容易，首先一点，难以做到的是特征方程的求解，这个多项式方程，当阶数特别大的时候几乎不能通过公式法求解，而且写一个字符串识别程序也不见得容易。如果能写成，那么在无公式的情况下，用数值方法求解也是不容易的，数值方法的计算量本身就很大，而且得到的一般不是精确解。所以综合来看，利用特征方程求解的这一做法基本放弃。这里直接调用库函数。

### 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233 | """filename: 4.1 Normal Value.py"""  """This is a universally usable code to get different  kind of normal value of an matrix. And the matrix is  a instance of Class Matrix.  Word is clean, this is the code!  """  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is a two dimension list.  """  **def** \_\_init\_\_**(**self**,**m**,**n**,**mainCol**):**  """  self.row: the row of the Matrix  self.col: the collumn of the Matrix  self.CheckedRow: if one row has been checked,  it should not be checked again  self.body: the internal storing structure  self.mainCol: the coefficient matrix  """  self**.**row **=** m  self**.**col **=** n  self**.**CheckedRow **=** set**(**range**(**self**.**row**))**  self**.**body **=** **[[**0 **for** i **in** range**(**n**)]** **for** i **in** range**(**m**)]**  self**.**mainCol **=** mainCol  **def** getVal**(**self**):**  """Giving value to each element of the matrix.    Overwrite the original value zeros.  """  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  self**.**body**[**i**][**j**]** **=** int**(**input**())**  **def** valid**(**self**,**e**,**kind **=** **None):**  """If these two matrix are not in the same form, return false,  else return true.    It is useful in the next functions.  """  **if** kind **==** 'multi'**:**  # SELF \* E  **return** self**.**col **==** e**.**row    **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** matrixAdd**(**self**,**e**):**  """A methon does not used in the pivot PCA algorithm.    Maybe it will be used in other programs.  """  self**.**valid**(**e**)**  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **+=** e**.**body**[**i**][**j**]**  **return** tmp  **def** matrixConstMulti**(**self**,**const**):**  """A constant number multiple a matrix."""  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **\*=** const  **return** tmp  **def** matrixMulti**(**self**,**e**):**  """Return the multiplication of two matrix."""  self**.**valid**(**e**,**'multi'**)**  ans **=** Matrix**(**self**.**row**,**e**.**col**,**e**.**col**)** # e.col has no meaning  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**e**.**col**):**  tmp **=** 0  **for** k **in** range**(**self**.**col**):**  tmp **+=** self**.**body**[**i**][**k**]** **\*** e**.**body**[**k**][**j**]**  ans**.**body**[**i**][**j**]** **=** tmp    **return** ans  **def** matrixTransform**(**self**,**target\_row\_Number**,**source\_row\_Number**,**times**=None):**  """There is a big problem, every decimal number we see is stored in the  RAM with the binary platform.    I can import the decimal lib to solve this problem, but I did not!  """  **if** times **==** **None:** # special case of matrixTransform(TarRow,times)  times\_tmp **=** source\_row\_Number  **for** j **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**j**]** **\*=** times\_tmp  **return**  **elif** times **==** 'exchange'**:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**],**\  self**.**body**[**source\_row\_Number**][**i**]** \  **=** self**.**body**[**source\_row\_Number**][**i**],**\  self**.**body**[**target\_row\_Number**][**i**]**  **return**  **else:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**]** **+=** \  times **\*** self**.**body**[**source\_row\_Number**][**i**]**  **def** matrixTranspose**(**self**):**  ans **=** Matrix**(**self**.**col**,**self**.**row**,**self**.**row**)**  # main column of ans is meaningless  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**j**][**i**]** **=** self**.**body**[**i**][**j**]**  **return** ans  **def** Quick\_Sort**(**L**):**  """A simple code to order a list with quicksort algorithm."""  **if** len**(**L**)** **<=** 1**:**  **return** L    **else:**  less **=** **[]**  equal **=** **[]**  bigger **=** **[]**  pivot **=** **(**L**[**0**]** **+** L**[-**1**]** **+** L**[**len**(**L**)//**2**])** **/** 3  # pivot should be taken seriously!  **for** i **in** range**(**len**(**L**)):**  **if** L**[**i**]** **<** pivot**:**  less**.**append**(**L**[**i**])**  **elif(**L**[**i**]** **==** pivot**):**  equal**.**append**(**L**(**i**))**  **else:**  bigger**.**append**(**L**[**i**])**  less **=** Quick\_Sort**(**less**)**  bigger **=** Quick\_Sort**(**bigger**)**  **return** less **+** equal **+** bigger  **def** GetNormalValue**(**M**,**kind**=**1**):**  """M is a matrix.    Return the Normal value of M.  """  **if** kind **==** 'inf'**:**  """NorVal is the maximum value among the sums of every row. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**row**):**  **for** j **in** range**(**M**.**col**):**  tmp **+=** M**.**body**[**i**][**j**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 1**:**  """NorVal is the maximum value among the sums of every column. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**col**):**  **for** j **in** range**(**M**.**row**):**  tmp **+=** M**.**body**[**j**][**i**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 2**:**  Multi **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  """The solve of elg is a little bit difficult,  I will change the code after learning.    Now I use the numpy package to solve it.  """  **import** numpy **as** np    MUL **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  tmp **=** np**.**array**(**MUL**.**body**)**  a**,**b **=** np**.**linalg**.**eig**(**tmp**)**  **return** np**.**sqrt**(**max**(**a**))**    **else:**  **raise** ValueError**(**"""Bad input!\n"""**)**  """-------------------------my Main Function-------------------------"""  a **=** Matrix**(**2**,**2**,**2**)**  **print(**'+-----------------------+-----------------------+'**)**  a**.**body **=** **[[**1**,-**1**],[**2**,**1**]]**  b **=** GetNormalValue**(**a**,**'inf'**)**  **print(**'| infinity normal value | '**,**b**,**' |'**)**  b **=** GetNormalValue**(**a**,**1**)**  **print(**'| 1 normal value | '**,**b**,**' |'**)**  b **=** GetNormalValue**(**a**,**2**)**  **print(**'| 2 normal value | '**,**b**,**' |'**)**  **print(**'+-----------------------+-----------------------+'**)**  a**.**body **=** **[[**10**,**15**],[**0**,**1**]]**  b **=** GetNormalValue**(**a**,**'inf'**)**  **print(**'| infinity normal value | '**,**b**,**' |'**)**  b **=** GetNormalValue**(**a**,**1**)**  **print(**'| 1 normal value | '**,**b**,**' |'**)**  b **=** GetNormalValue**(**a**,**2**)**  **print(**'| 2 normal value | '**,**b**,**' |'**)**  **print(**'+-----------------------+-----------------------+'**)**  a**.**body **=** **[[**0.6**,-**0.5**],[-**0.1**,**0.3**]]**  b **=** GetNormalValue**(**a**,**'inf'**)**  **print(**'| infinity normal value | '**,**b**,**'|'**)**  b **=** GetNormalValue**(**a**,**1**)**  **print(**'| 1 normal value | '**,**b**,**' |'**)**  b **=** GetNormalValue**(**a**,**2**)**  **print(**'| 2 normal value | '**,**b**,**' |'**)**  **print(**'+-----------------------+-----------------------+'**)** |

Code Box 1 矩阵的三种范数

### 输出结果



输出结果 1

### 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。

## 2题

设，计算的条件数及。

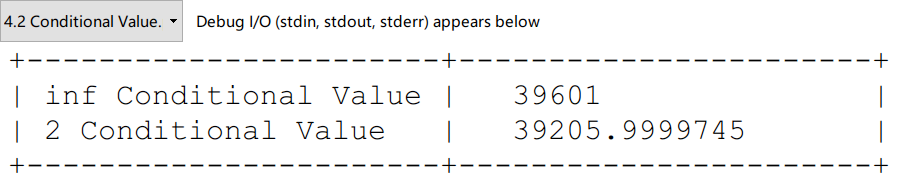
**解答：**

### 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224 | """filename: 4.2 Conditional Value.py"""  """This is a universally usable code to get different  kind of normal value of an matrix. And the matrix is  a instance of Class Matrix.  Word is clean, this is the code!  """  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is a two dimension list.  """  **def** \_\_init\_\_**(**self**,**m**,**n**,**mainCol**):**  """  self.row: the row of the Matrix  self.col: the collumn of the Matrix  self.CheckedRow: if one row has been checked,  it should not be checked again  self.body: the internal storing structure  self.mainCol: the coefficient matrix  """  self**.**row **=** m  self**.**col **=** n  self**.**CheckedRow **=** set**(**range**(**self**.**row**))**  self**.**body **=** **[[**0 **for** i **in** range**(**n**)]** **for** i **in** range**(**m**)]**  self**.**mainCol **=** mainCol  **def** getVal**(**self**):**  """Giving value to each element of the matrix.    Overwrite the original value zeros.  """  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  self**.**body**[**i**][**j**]** **=** int**(**input**())**  **def** valid**(**self**,**e**,**kind **=** **None):**  """If these two matrix are not in the same form, return false,  else return true.    It is useful in the next functions.  """  **if** kind **==** 'multi'**:**  # SELF \* E  **return** self**.**col **==** e**.**row    **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** matrixAdd**(**self**,**e**):**  """A methon does not used in the pivot PCA algorithm.    Maybe it will be used in other programs.  """  self**.**valid**(**e**)**  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **+=** e**.**body**[**i**][**j**]**  **return** tmp  **def** matrixConstMulti**(**self**,**const**):**  """A constant number multiple a matrix."""  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **\*=** const  **return** tmp  **def** matrixMulti**(**self**,**e**):**  """Return the multiplication of two matrix."""  self**.**valid**(**e**,**'multi'**)**  ans **=** Matrix**(**self**.**row**,**e**.**col**,**e**.**col**)** # e.col has no meaning  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**e**.**col**):**  tmp **=** 0  **for** k **in** range**(**self**.**col**):**  tmp **+=** self**.**body**[**i**][**k**]** **\*** e**.**body**[**k**][**j**]**  ans**.**body**[**i**][**j**]** **=** tmp    **return** ans  **def** matrixTransform**(**self**,**target\_row\_Number**,**source\_row\_Number**,**times**=None):**  """There is a big problem, every decimal number we see is stored in the  RAM with the binary platform.    I can import the decimal lib to solve this problem, but I did not!  """  **if** times **==** **None:** # special case of matrixTransform(TarRow,times)  times\_tmp **=** source\_row\_Number  **for** j **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**j**]** **\*=** times\_tmp  **return**  **elif** times **==** 'exchange'**:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**],**\  self**.**body**[**source\_row\_Number**][**i**]** \  **=** self**.**body**[**source\_row\_Number**][**i**],**\  self**.**body**[**target\_row\_Number**][**i**]**  **return**  **else:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**]** **+=** \  times **\*** self**.**body**[**source\_row\_Number**][**i**]**  **def** matrixTranspose**(**self**):**  ans **=** Matrix**(**self**.**col**,**self**.**row**,**self**.**row**)**  # main column of ans is meaningless  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**j**][**i**]** **=** self**.**body**[**i**][**j**]**  **return** ans  **def** Quick\_Sort**(**L**):**  """A simple code to order a list with quicksort algorithm."""  **if** len**(**L**)** **<=** 1**:**  **return** L    **else:**  less **=** **[]**  equal **=** **[]**  bigger **=** **[]**  pivot **=** **(**L**[**0**]** **+** L**[-**1**]** **+** L**[**len**(**L**)//**2**])** **/** 3  # pivot should be taken seriously!  **for** i **in** range**(**len**(**L**)):**  **if** L**[**i**]** **<** pivot**:**  less**.**append**(**L**[**i**])**  **elif(**L**[**i**]** **==** pivot**):**  equal**.**append**(**L**(**i**))**  **else:**  bigger**.**append**(**L**[**i**])**  less **=** Quick\_Sort**(**less**)**  bigger **=** Quick\_Sort**(**bigger**)**  **return** less **+** equal **+** bigger  **def** GetNormalValue**(**M**,**kind**=**2**):**  """M is a matrix.    Return the Normal value of M.  """  **if** kind **==** 'inf'**:**  """NorVal is the maximum value among the sums of every row. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**row**):**  **for** j **in** range**(**M**.**col**):**  tmp **+=** M**.**body**[**i**][**j**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 1**:**  """NorVal is the maximum value among the sums of every column. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**col**):**  **for** j **in** range**(**M**.**row**):**  tmp **+=** M**.**body**[**j**][**i**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 2**:**  Multi **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  """  The solving of eig is a little difficult for me,  I will change the code after learning more later.    Now I use the numpy package to solve it.  """  **import** numpy **as** np    MUL **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  tmp **=** np**.**array**(**MUL**.**body**)**  a**,**b **=** np**.**linalg**.**eig**(**tmp**)**  **return** np**.**sqrt**(**max**(**a**))**    **else:**  **raise** ValueError**(**"""Bad input!\n"""**)**  **def** GetCondValue**(**M**,**kind**=**2**):**  **if** kind **==** 'inf'**:**  **return** GetNormalValue**(**M**,**'inf'**)** **\*** GetNormalValue**(**M**.**matrixTranspose**(),**'inf'**)**  **if** kind **==** 1**:**  **return** GetNormalValue**(**M**,**1**)** **\*** GetNormalValue**(**M**.**matrixTranspose**(),**1**)**  **if** kind **==** 2**:**  **return** GetNormalValue**(**M**,**2**)** **\*** GetNormalValue**(**M**.**matrixTranspose**(),**2**)**  """-------------------------my Main Function-------------------------"""  A **=** Matrix**(**2**,**2**,**2**)**  A**.**body **=** **[[**100**,**99**],[**99**,**98**]]**  ans\_inf **=** GetCondValue**(**A**,**'inf'**)**  ans\_2 **=** GetCondValue**(**A**,**2**)**  **print(**'+-----------------------+-----------------------+'**)**  **print(**'| inf Conditional Value\t| '**,**ans\_inf**,**'\t\t|'**)**  **print(**'| 2 Conditional Value\t| '**,**ans\_2**,**'\t|'**)**  **print(**'+-----------------------+-----------------------+'**)** |

Code Box 2

### 输出结果



输出结果 2

### 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。

## 3题

用高斯-赛德尔迭代法解下列线性方程组，要求当时迭代终止。

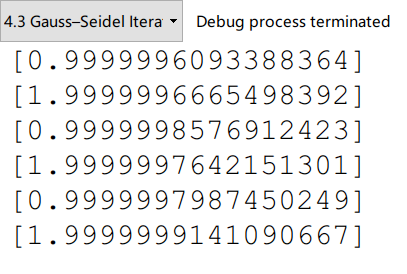
**解答：**

### 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298 | """filename: 4.3 Gauss-Seidel Method.py"""  """filename: 4.4 Plot.py"""  **import** matplotlib**.**pyplot **as** pl  **import** numpy **as** np  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is a two dimension list.  """  **def** \_\_init\_\_**(**self**,**m**,**n**,**mainCol**):**  """  self.row: the row of the Matrix  self.col: the collumn of the Matrix  self.CheckedRow: if one row has been checked,  it should not be checked again  self.body: the internal storing structure  self.mainCol: the coefficient matrix  """  self**.**row **=** m  self**.**col **=** n  self**.**CheckedRow **=** set**(**range**(**self**.**row**))**  self**.**body **=** **[[**0 **for** i **in** range**(**n**)]** **for** i **in** range**(**m**)]**  self**.**mainCol **=** mainCol  **def** getVal**(**self**):**  """Giving value to each element of the matrix.    Overwrite the original value zeros.  """  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  self**.**body**[**i**][**j**]** **=** int**(**input**())**  **def** valid**(**self**,**e**,**kind **=** **None):**  """If these two matrix are not in the same form, return false,  else return true.    It is useful in the next functions.  """  **if** kind **==** 'multi'**:**  # SELF \* E  **return** self**.**col **==** e**.**row    **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** matrixAdd**(**self**,**e**):**  """A methon does not used in the pivot PCA algorithm.    Maybe it will be used in other programs.  """  self**.**valid**(**e**)**  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **+=** e**.**body**[**i**][**j**]**  **return** tmp  **def** matrixConstMulti**(**self**,**const**):**  """A constant number multiple a matrix."""  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **\*=** const  **return** tmp  **def** matrixMulti**(**self**,**e**):**  """Return the multiplication of two matrix."""  self**.**valid**(**e**,**'multi'**)**  ans **=** Matrix**(**self**.**row**,**e**.**col**,**e**.**col**)** # e.col has no meaning  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**e**.**col**):**  tmp **=** 0  **for** k **in** range**(**self**.**col**):**  tmp **+=** self**.**body**[**i**][**k**]** **\*** e**.**body**[**k**][**j**]**  ans**.**body**[**i**][**j**]** **=** tmp    **return** ans  **def** matrixTransform**(**self**,**target\_row\_Number**,**source\_row\_Number**,**times**=None):**  """There is a big problem, every decimal number we see is stored in the  RAM with the binary platform.    I can import the decimal lib to solve this problem, but I did not!  """  **if** times **==** **None:** # special case of matrixTransform(TarRow,times)  times\_tmp **=** source\_row\_Number  **for** j **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**j**]** **\*=** times\_tmp  **return**  **elif** times **==** 'exchange'**:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**],**\  self**.**body**[**source\_row\_Number**][**i**]** \  **=** self**.**body**[**source\_row\_Number**][**i**],**\  self**.**body**[**target\_row\_Number**][**i**]**  **return**  **else:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**]** **+=** \  times **\*** self**.**body**[**source\_row\_Number**][**i**]**  **def** matrixTranspose**(**self**):**  ans **=** Matrix**(**self**.**col**,**self**.**row**,**self**.**row**)**  # main column of ans is meaningless  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**j**][**i**]** **=** self**.**body**[**i**][**j**]**  **return** ans  **def** Quick\_Sort**(**L**):**  """A simple code to order a list with quicksort algorithm."""  **if** len**(**L**)** **<=** 1**:**  **return** L    **else:**  less **=** **[]**  equal **=** **[]**  bigger **=** **[]**  pivot **=** **(**L**[**0**]** **+** L**[-**1**]** **+** L**[**len**(**L**)//**2**])** **/** 3  # pivot should be taken seriously!  **for** i **in** range**(**len**(**L**)):**  **if** L**[**i**]** **<** pivot**:**  less**.**append**(**L**[**i**])**  **elif(**L**[**i**]** **==** pivot**):**  equal**.**append**(**L**[**i**])**  **else:**  bigger**.**append**(**L**[**i**])**  less **=** Quick\_Sort**(**less**)**  bigger **=** Quick\_Sort**(**bigger**)**  **return** less **+** equal **+** bigger  **def** GetNormalValue**(**M**,**kind**=**2**):**  """M is a matrix.    Return the Normal value of M.  """  **if** kind **==** 'inf'**:**  """NorVal is the maximum value among the sums of every row. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**row**):**  **for** j **in** range**(**M**.**col**):**  tmp **+=** M**.**body**[**i**][**j**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 1**:**  """NorVal is the maximum value among the sums of every column. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**col**):**  **for** j **in** range**(**M**.**row**):**  tmp **+=** M**.**body**[**j**][**i**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 2**:**  Multi **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  """  The solving of eig is a little difficult for me,  I will change the code after learning more later.    Now I use the numpy package to solve it.  """  **import** numpy **as** np    MUL **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  tmp **=** np**.**array**(**MUL**.**body**)**  a**,**b **=** np**.**linalg**.**eig**(**tmp**)**  **return** np**.**sqrt**(**max**(**a**))**    **else:**  **raise** ValueError**(**"""Bad input!\n"""**)**  **def** vectorNormalValue**(**a**,**kind**=**2**):**  """Get the normal value of a vector."""  **if** kind **==** 1**:**  ans **=** 0  **for** i **in** a**:**  ans **+=** abs**(**i**)**  **return** ans    **if** kind **==** 2**:**  **from** math **import** sqrt **as** sq  ans **=** 0  **for** i **in** a**:**  ans **+=** i **\*** i  **return** sq**(**ans**)**    **if** kind **==** 'inf'**:**  tmp **=** list**()**  **for** i **in** a**:**  tmp**.**append**(**abs**(**i**))**  Q **=** Quick\_Sort**(**tmp**)**  **return** Q**[-**1**]**  **def** iterFormat**(**A**,**b**):**  """Get the iter matrix."""  **if** b**.**col **!=** 1**:**  **return**  **for** i **in** range**(**A**.**row**):**  b**.**body**[**i**][**0**]** **=** b**.**body**[**i**][**0**]** **/** A**.**body**[**i**][**i**]**  **for** i **in** range**(**A**.**row**):**  A**.**matrixTransform**(**i**,**1 **/** A**.**body**[**i**][**i**])**  B **=** Matrix**(**A**.**row**,**A**.**col**,**A**.**mainCol**)**  **for** i **in** range**(**B**.**row**):**  **for** j **in** range**(**B**.**col**):**  **if** i **==** j**:**  B**.**body**[**i**][**j**]** **=** 0  **else:**  B**.**body**[**i**][**j**]** **=** **-**1 **\*** A**.**body**[**i**][**j**]**  ans **=** **(**B**,**b**)**  **return** ans  **def** GS\_op1**(**B**,**b**,**x**):**  """simple function."""  **for** i **in** range**(**x**.**row**):**  tmp **=** 0  **for** j **in** range**(**B**.**col**):**  tmp **+=** B**.**body**[**i**][**j**]** **\*** x**.**body**[**j**][**0**]**  x**.**body**[**i**][**0**]** **=** tmp **+** b**.**body**[**i**][**0**]**  **return** x  **def** GS**(**times**):**  **for** i **in** range**(**times**-**1**):**  GS\_op1**(**B**,**b**,**x**)**    tmp **=** Matrix**(**6**,**1**,**1**)**  **for** i **in** range**(**x**.**row**):**  tmp**.**body**[**i**][**0**]** **=** x**.**body**[**i**][**0**]** # deep copy    GS\_op1**(**B**,**b**,**x**)** # do it once more    step3 **=** x**.**matrixConstMulti**(-**1**)**  there **=** tmp**.**matrixAdd**(**step3**)**    c **=** list**()**    **for** i **in** range**(**there**.**row**):**  c**.**append**(**there**.**body**[**i**][**0**])**    **return** vectorNormalValue**(**c**)**  """-------------------------my Main Function-------------------------"""  A **=** Matrix**(**6**,**6**,**6**)**  A**.**body **=** **[[**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**]]**  b **=** Matrix**(**6**,**1**,**1**)**  b**.**body **=** **[[**0**],[**5**],[**0**],[**6**],[-**2**],[**6**]]**  x **=** Matrix**(**6**,**1**,**1**)**  x**.**body **=** **[[**1**],[**1**],[**1**],[**1**],[**1**],[**1**]]**  tmp **=** iterFormat**(**A**,**b**)**  B**,**b **=** tmp**[**0**],**tmp**[**1**]**  guide **=** 10  times **=** 5  **while** guide **>** 1e-5**:**  guide **=** GS**(**times**)**  **for** i **in** x**.**body**:**  **print(**i**)** |

Code Box 3

### 输出结果



输出结果 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。

**解答：**

### 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362 | """filename: 4.4 Plot.py"""  **import** matplotlib**.**pyplot **as** pl  **import** numpy **as** np  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is a two dimension list.  """  **def** \_\_init\_\_**(**self**,**m**,**n**,**mainCol**):**  """  self.row: the row of the Matrix  self.col: the collumn of the Matrix  self.CheckedRow: if one row has been checked,  it should not be checked again  self.body: the internal storing structure  self.mainCol: the coefficient matrix  """  self**.**row **=** m  self**.**col **=** n  self**.**CheckedRow **=** set**(**range**(**self**.**row**))**  self**.**body **=** **[[**0 **for** i **in** range**(**n**)]** **for** i **in** range**(**m**)]**  self**.**mainCol **=** mainCol  **def** getVal**(**self**):**  """Giving value to each element of the matrix.    Overwrite the original value zeros.  """  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  self**.**body**[**i**][**j**]** **=** int**(**input**())**  **def** valid**(**self**,**e**,**kind **=** **None):**  """If these two matrix are not in the same form, return false,  else return true.    It is useful in the next functions.  """  **if** kind **==** 'multi'**:**  # SELF \* E  **return** self**.**col **==** e**.**row    **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** matrixAdd**(**self**,**e**):**  """A methon does not used in the pivot PCA algorithm.    Maybe it will be used in other programs.  """  self**.**valid**(**e**)**  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **+=** e**.**body**[**i**][**j**]**  **return** tmp  **def** matrixConstMulti**(**self**,**const**):**  """A constant number multiple a matrix."""  tmp **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp**.**body**[**i**][**j**]** **\*=** const  **return** tmp  **def** matrixMulti**(**self**,**e**):**  """Return the multiplication of two matrix."""  self**.**valid**(**e**,**'multi'**)**  ans **=** Matrix**(**self**.**row**,**e**.**col**,**e**.**col**)** # e.col has no meaning  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**e**.**col**):**  tmp **=** 0  **for** k **in** range**(**self**.**col**):**  tmp **+=** self**.**body**[**i**][**k**]** **\*** e**.**body**[**k**][**j**]**  ans**.**body**[**i**][**j**]** **=** tmp    **return** ans  **def** matrixTransform**(**self**,**target\_row\_Number**,**source\_row\_Number**,**times**=None):**  """There is a big problem, every decimal number we see is stored in the  RAM with the binary platform.    I can import the decimal lib to solve this problem, but I did not!  """  **if** times **==** **None:** # special case of matrixTransform(TarRow,times)  times\_tmp **=** source\_row\_Number  **for** j **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**j**]** **\*=** times\_tmp  **return**  **elif** times **==** 'exchange'**:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**],**\  self**.**body**[**source\_row\_Number**][**i**]** \  **=** self**.**body**[**source\_row\_Number**][**i**],**\  self**.**body**[**target\_row\_Number**][**i**]**  **return**  **else:**  **for** i **in** range**(**self**.**col**):**  self**.**body**[**target\_row\_Number**][**i**]** **+=** \  times **\*** self**.**body**[**source\_row\_Number**][**i**]**  **def** matrixTranspose**(**self**):**  ans **=** Matrix**(**self**.**col**,**self**.**row**,**self**.**row**)**  # main column of ans is meaningless  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**j**][**i**]** **=** self**.**body**[**i**][**j**]**  **return** ans  **def** Quick\_Sort**(**L**):**  """A simple code to order a list with quicksort algorithm."""  **if** len**(**L**)** **<=** 1**:**  **return** L    **else:**  less **=** **[]**  equal **=** **[]**  bigger **=** **[]**  pivot **=** **(**L**[**0**]** **+** L**[-**1**]** **+** L**[**len**(**L**)//**2**])** **/** 3  # pivot should be taken seriously!  **for** i **in** range**(**len**(**L**)):**  **if** L**[**i**]** **<** pivot**:**  less**.**append**(**L**[**i**])**  **elif(**L**[**i**]** **==** pivot**):**  equal**.**append**(**L**[**i**])**  **else:**  bigger**.**append**(**L**[**i**])**  less **=** Quick\_Sort**(**less**)**  bigger **=** Quick\_Sort**(**bigger**)**  **return** less **+** equal **+** bigger  **def** GetNormalValue**(**M**,**kind**=**2**):**  """M is a matrix.    Return the Normal value of M.  """  **if** kind **==** 'inf'**:**  """NorVal is the maximum value among the sums of every row. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**row**):**  **for** j **in** range**(**M**.**col**):**  tmp **+=** M**.**body**[**i**][**j**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 1**:**  """NorVal is the maximum value among the sums of every column. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**M**.**col**):**  **for** j **in** range**(**M**.**row**):**  tmp **+=** M**.**body**[**j**][**i**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 2**:**  Multi **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  """  The solving of eig is a little difficult for me,  I will change the code after learning more later.    Now I use the numpy package to solve it.  """  **import** numpy **as** np    MUL **=** M**.**matrixMulti**(**M**.**matrixTranspose**())**  tmp **=** np**.**array**(**MUL**.**body**)**  a**,**b **=** np**.**linalg**.**eig**(**tmp**)**  **return** np**.**sqrt**(**max**(**a**))**    **else:**  **raise** ValueError**(**"""Bad input!\n"""**)**  **def** vectorNormalValue**(**a**,**kind**=**2**):**  """Get the normal value of a vector."""  **if** kind **==** 1**:**  ans **=** 0  **for** i **in** a**:**  ans **+=** abs**(**i**)**  **return** ans    **if** kind **==** 2**:**  **from** math **import** sqrt **as** sq  ans **=** 0  **for** i **in** a**:**  ans **+=** i **\*** i  **return** sq**(**ans**)**    **if** kind **==** 'inf'**:**  tmp **=** list**()**  **for** i **in** a**:**  tmp**.**append**(**abs**(**i**))**  Q **=** Quick\_Sort**(**tmp**)**  **return** Q**[-**1**]**  **def** iterFormat**(**A**,**b**):**  """Get the iter matrix."""  **if** b**.**col **!=** 1**:**  **return**  **for** i **in** range**(**A**.**row**):**  b**.**body**[**i**][**0**]** **=** b**.**body**[**i**][**0**]** **/** A**.**body**[**i**][**i**]**  **for** i **in** range**(**A**.**row**):**  A**.**matrixTransform**(**i**,**1 **/** A**.**body**[**i**][**i**])**  B **=** Matrix**(**A**.**row**,**A**.**col**,**A**.**mainCol**)**  **for** i **in** range**(**B**.**row**):**  **for** j **in** range**(**B**.**col**):**  **if** i **==** j**:**  B**.**body**[**i**][**j**]** **=** 0  **else:**  B**.**body**[**i**][**j**]** **=** **-**1 **\*** A**.**body**[**i**][**j**]**  ans **=** **(**B**,**b**)**  **return** ans  **def** GS\_op1**(**B**,**b**,**x**):**  """simple function."""  **for** i **in** range**(**x**.**row**):**  tmp **=** 0  **for** j **in** range**(**B**.**col**):**  tmp **+=** B**.**body**[**i**][**j**]** **\*** x**.**body**[**j**][**0**]**  x**.**body**[**i**][**0**]** **=** tmp **+** b**.**body**[**i**][**0**]**  # x is changed during one iteration  **return** x  **def** relax\_op1**(**B**,**b**,**x**):**  """simple function."""  **for** i **in** range**(**x**.**row**):**  tmp **=** 0  **for** j **in** range**(**B**.**col**):**  tmp **+=** B**.**body**[**i**][**j**]** **\*** x**.**body**[**j**][**0**]**  x**.**body**[**i**][**0**]** **=** tmp **+** b**.**body**[**i**][**0**]**  # x is changed during one iteration  **return** x  **class** **Plot:**  """Just for testing."""  **def** \_\_init\_\_**(**self**):**  """Initialize this class.    As you can see, this function could not be universally used.  """  A **=** Matrix**(**6**,**6**,**6**)**  A**.**body **=** **[[**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**]]**    b **=** Matrix**(**6**,**1**,**1**)**  b**.**body **=** **[[**0**],[**5**],[**0**],[**6**],[-**2**],[**6**]]**  tmp **=** iterFormat**(**A**,**b**)**    '''  self.B: the matrix to iterate  self.b: the matrix with only one column  self.x: the original value we give,  Default set it [0,0,0,0,0,0]'  '''  self**.**B **=** tmp**[**0**]**  self**.**b **=** tmp**[**1**]**  self**.**x **=** Matrix**(**6**,**1**,**1**)**  **def** Jacobi**(**self**,**times**):**  self**.**\_\_init\_\_**()**  **for** i **in** range**(**times**-**1**):**  step1 **=** self**.**B**.**matrixMulti**(**self**.**x**)**  step2 **=** step1**.**matrixAdd**(**self**.**b**)**  self**.**x **=** step2    lastStep **=** Matrix**(**self**.**x**.**row**,**self**.**x**.**col**,**self**.**x**.**mainCol**)**  **for** i **in** range**(**self**.**x**.**row**):**  lastStep**.**body**[**i**][**0**]** **=** self**.**x**.**body**[**i**][**0**]**    step1 **=** self**.**B**.**matrixMulti**(**lastStep**)**  lastStep **=** step1**.**matrixAdd**(**self**.**b**)**  step3 **=** lastStep**.**matrixConstMulti**(-**1**)**  there **=** step2**.**matrixAdd**(**step3**)**    c **=** list**()**  **for** i **in** range**(**there**.**row**):**  c**.**append**(**there**.**body**[**i**][**0**])**    **return** vectorNormalValue**(**c**)**  **def** GS**(**self**,**times**):**  self**.**\_\_init\_\_**()**    **for** i **in** range**(**times**-**1**):**  GS\_op1**(**self**.**B**,**self**.**b**,**self**.**x**)**    tmp **=** Matrix**(**6**,**1**,**1**)**  **for** i **in** range**(**self**.**x**.**row**):**  tmp**.**body**[**i**][**0**]** **=** self**.**x**.**body**[**i**][**0**]** # deep copy    GS\_op1**(**self**.**B**,**self**.**b**,**self**.**x**)** # do it once more    step3 **=** self**.**x**.**matrixConstMulti**(-**1**)**  there **=** tmp**.**matrixAdd**(**step3**)**    c **=** list**()**    **for** i **in** range**(**there**.**row**):**  c**.**append**(**there**.**body**[**i**][**0**])**    **return** vectorNormalValue**(**c**)**    **def** plot**(**self**,**n**=**100**):**  """Default iteration deepth is 100."""    x\_jacobi **=** **[]**  y\_jacobi **=** **[]**  **for** i **in** range**(**4**,**n**,**4**):**  x\_jacobi**.**append**(**i**)**  y\_jacobi**.**append**(**self**.**Jacobi**(**i**))**    x\_gs **=** **[]**  y\_gs **=** **[]**  **for** i **in** range**(**4**,**n**,**4**):**  x\_gs**.**append**(**i**)**  y\_gs**.**append**(**self**.**GS**(**i**))**    pl**.**grid**()**  pl**.**title**(**'Relation: Iteration deepth and Normal Value'**,**fontsize**=**16**)**  pl**.**plot**(**x\_jacobi**,**y\_jacobi**,**'o-g'**,**label**=**'Jacobi Method'**)**  pl**.**plot**(**x\_gs**,**y\_gs**,**'o-r'**,**label**=**'Gauss-Seidel Method'**)**  pl**.**legend**()**  pl**.**xlabel**(**'Iteration Deepth'**)**  pl**.**ylabel**(**'Normal Value'**)**    pl**.**show**()**  M **=** Plot**()**  M**.**plot**()** |

Code Box 4

### 输出结果



输出结果 4

### 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。

## 5题

用松弛法解

分别取，，。要求当时迭代终止，并对每个值确定迭代次数（初值）。

**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。

### 程序代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  564 | # -\*- coding: utf-8 -\*-  """  Created on Tue Nov 28 18:30:53 2017  @author: Newton  """  """filename: 4.5 Relaxation Method.py"""  **import** matplotlib**.**pyplot **as** pl  **import** numpy **as** np  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is a two dimension list.  """  **def** \_\_init\_\_**(**self**,**m**,**n**,**mainCol **=** **None):**  """  self.row: the row of the Matrix  self.col: the collumn of the Matrix  self.CheckedRow: if one row has been checked,  it should not be checked again  self.body: the internal storing structure  self.mainCol: the coefficient matrix  """  self**.**row **=** m  self**.**col **=** n  self**.**CheckedRow **=** set**(**range**(**self**.**row**))**  self**.**body **=** **[[**0 **for** i **in** range**(**n**)]** **for** i **in** range**(**m**)]**  self**.**mainCol **=** mainCol  **def** getVal**(**self**):**  """Giving value to each element of the matrix.    Overwrite the original value zeros.  """  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  self**.**body**[**i**][**j**]** **=** input**()**  **def** valid**(**self**,**e**,**kind **=** **None):**  """If these two matrices are not in the right form, return false,  else return true.    It is useful in the next functions.  """  **if** kind **==** 'multi'**:**  # SELF \* E  **return** self**.**col **==** e**.**row    **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** matrixAdd**(**self**,**e**):**  """One matrix add to another and generate a new one.    The original one would not change.  """  self**.**valid**(**e**)**  # if e and self are not in the same form, return false.    ans **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**    **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]** **+** e**.**body**[**i**][**j**]**  **return** ans  **def** matrixConstMulti**(**self**,**const**):**  """A constant number multiple a matrix."""    ans **=** Matrix**(**self**.**row**,**self**.**col**,**self**.**mainCol**)**    **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]** **\*** const  **return** ans  **def** matrixMulti**(**self**,**e**):**  """Return the multiplication of two matrices.    Attention! ans = self \* e, not e \* self.  """    self**.**valid**(**e**,**'multi'**)**  # if e and self could not make multiplication, return false    ans **=** Matrix**(**self**.**row**,** e**.**col**,** e**.**col**)**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**e**.**col**):**  tmp **=** 0  **for** k **in** range**(**self**.**col**):**  tmp **+=** self**.**body**[**i**][**k**]** **\*** e**.**body**[**k**][**j**]**  ans**.**body**[**i**][**j**]** **=** tmp    **return** ans  **def** matrixTransform**(**self**,**target\_row**,**source\_row**,**times**=None):**  """  case 1: two coefficient, make coe(1) row times coe(2)  case 2: exchange the two rows with 'exchange' reminding  case 3: coe(1) rows add the coe(3) times of coe(2) row.    """    ans **=** Matrix**(**self**.**row**,** self**.**col**,** self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):** # deep copy  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**    **if** times **==** **None:** # special case of matrixTransform(TarRow,times)  times\_tmp **=** source\_row  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**target\_row**][**j**]** **=** \  ans**.**body**[**target\_row**][**j**]** **\*** times\_tmp  **elif** times **==** 'exchange'**:**  **for** i **in** range**(**self**.**col**):**  ans**.**body**[**target\_row**][**i**],** ans**.**body**[**source\_row**][**i**]** \  **=** ans**.**body**[**source\_row**][**i**],** ans**.**body**[**target\_row**][**i**]**    **else:**  **for** i **in** range**(**self**.**col**):**  ans**.**body**[**target\_row**][**i**]** **+=** \  times **\*** ans**.**body**[**source\_row**][**i**]**    **return** ans  **def** matrixTranspose**(**self**):**  """Generate a new matrix which is the transpose of the old one."""    ans **=** Matrix**(**self**.**col**,**self**.**row**,**self**.**row**)**  # main column of ans is meaningless    **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**j**][**i**]** **=** self**.**body**[**i**][**j**]**  **return** ans  **def** matrixNormVal**(**self**,** kind **=** 2**):**  """  Return the Normal value of this matrix.  """    **if** kind **==** 'inf'**:**  """NorVal is the maximum value among the sums of every row. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  tmp **+=** self**.**body**[**i**][**j**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)** # quick sort algorithm  **return** sums**[-**1**]**    **elif** kind **==** 1**:**  """NorVal is the maximum value among the sums of every column. """  sums **=** list**()**  tmp **=** 0  **for** i **in** range**(**self**.**col**):**  **for** j **in** range**(**self**.**row**):**  tmp **+=** self**.**body**[**j**][**i**]**  sums**.**append**(**tmp**)**  tmp **=** 0    sums **=** Quick\_Sort**(**sums**)**  **return** sums**[-**1**]**    **elif** kind **==** 2**:**  """  The solving of eig is a little difficult for me,  I will change the code after learning more later.    Now I use the numpy package to solve it.  """  **import** numpy **as** np    MUL **=** self**.**matrixMulti**(**self**.**matrixTranspose**())**  tmp **=** np**.**array**(**MUL**.**body**)**  a**,**b **=** np**.**linalg**.**eig**(**tmp**)**  **return** np**.**sqrt**(**max**(**a**))**    **else:**  **raise** ValueError**(**"""Bad input!\n"""**)**    **def** matrixInversion**(**self**):**  """Generate a new matrix which is the old one's inversion."""    Max **=** 0 # initialize the variable  position\_row **=** 0 # the row number of the maximum value  position\_col **=** 0    ans **=** Matrix**(**self**.**row**,** self**.**col**,** self**.**mainCol**)**  **for** i **in** range**(**self**.**row**):**  **for** j **in** range**(**self**.**col**):**  ans**.**body**[**i**][**j**]** **=** self**.**body**[**i**][**j**]**  eyes **=** Matrix**(**self**.**row**,** self**.**col**,** self**.**mainCol**)**  **for** i **in** range**(**eyes**.**row**):**  eyes**.**body**[**i**][**i**]** **=** 1  ans**.**body**[**i**]** **+=** eyes**.**body**[**i**]**    ans**.**mainCol **=** ans**.**col  ans**.**col **\*=** 2    **for** i **in** range**(**ans**.**row**):**  **for** j **in** ans**.**CheckedRow**:**  **for** k **in** range**(**ans**.**mainCol**):** # not in all the columns  **if** abs**(**Max**)** **<=** abs**(**ans**.**body**[**j**][**k**]):**  Max **=** ans**.**body**[**j**][**k**]**  position\_row **=** j  position\_col **=** k    ans **=** ans**.**matrixTransform**(**position\_row**,** 1 **/** Max**)**    ans**.**CheckedRow**.**remove**(**position\_row**)**    **for** j **in** range**(**ans**.**row**):**  **if** j **!=** position\_row**:**  ans **=** ans**.**matrixTransform\  **(**j**,** position\_row**,-**1 **\*** ans**.**body**[**j**][**position\_col**])**    Max **=** 0  position\_row **=** 0  position\_col **=** 0    begin **=** 0  **for** j **in** range**(**ans**.**mainCol**):**  **for** i **in** range**(**ans**.**row**):**  **if** ans**.**body**[**i**][**j**]** **==** 1**:**  ans **=** ans**.**matrixTransform**(**begin**,**i**,**'exchange'**)**  begin **+=** 1    new **=** Matrix**(**self**.**row**,** self**.**row**,** **None)**  **for** i **in** range**(**new**.**row**):**  **for** j **in** range**(**new**.**col**):**  new**.**body**[**i**][**j**]** **=** ans**.**body**[**i**][**j**+**ans**.**row**]**  ans **=** new  **return** ans  **def** Quick\_Sort**(**L**):**  """A simple code to order a list with quicksort algorithm."""    **if** len**(**L**)** **<=** 1**:**  **return** L    **else:**  less **=** **[]**  equal **=** **[]**  bigger **=** **[]**  pivot **=** **(**L**[**0**]** **+** L**[-**1**]** **+** L**[**len**(**L**)//**2**])** **/** 3  # pivot should be taken seriously!  **for** i **in** range**(**len**(**L**)):**  **if** L**[**i**]** **<** pivot**:**  less**.**append**(**L**[**i**])**  **elif(**L**[**i**]** **==** pivot**):**  equal**.**append**(**L**[**i**])**  **else:**  bigger**.**append**(**L**[**i**])**  less **=** Quick\_Sort**(**less**)**  bigger **=** Quick\_Sort**(**bigger**)**  **return** less **+** equal **+** bigger  #---------------------------new class---------------------------  **class** **MatrixIterMethods:**  """This class includes three methods which could be used in the  solving of linear equations.    Method 1: Jabobi method  Method 2: Gauss-Seidel method  Method 3: Relaxation method    All these three methods will return x and the iteration deepth.  """    **def** \_\_init\_\_**(**self**,**A**,**b**,**x0**,**omega**=**1**):**  """Initialize this class.    A: coefficient matrix  b: where Ax = b  x0: the original value of x we choose    These global variables would not change after init operation.  """    **if** omega **==** 1**:**  **if** b**.**col **!=** 1**:**  **raise** ValueError**(**'Bad inputs, please Check it!'**)**    tmp\_b **=** Matrix**(**b**.**row**,** b**.**col**,** b**.**mainCol**)**  **for** i **in** range**(**b**.**row**):**  **for** j **in** range**(**b**.**col**):**  tmp\_b**.**body**[**i**][**j**]** **=** b**.**body**[**i**][**j**]**    **for** i **in** range**(**tmp\_b**.**row**):**  tmp\_b**.**body**[**i**][**0**]** **=** **(**tmp\_b**.**body**[**i**][**0**])** **/** A**.**body**[**i**][**i**]**    tmp\_A **=** Matrix**(**A**.**row**,** A**.**col**,** A**.**mainCol**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  tmp\_A**.**body**[**i**][**j**]** **=** A**.**body**[**i**][**j**]**    **for** i **in** range**(**A**.**row**):**  tmp\_A **=** tmp\_A**.**matrixTransform**(**i**,**1 **/** A**.**body**[**i**][**i**])**    B **=** Matrix**(**A**.**row**,** A**.**col**,** A**.**mainCol**)**  **for** i **in** range**(**B**.**row**):**  **for** j **in** range**(**B**.**col**):**  **if** i **==** j**:**  B**.**body**[**i**][**j**]** **=** 0  **else:**  B**.**body**[**i**][**j**]** **=** **-**1 **\*** tmp\_A**.**body**[**i**][**j**]**  ans **=** **(**B**,**tmp\_b**)**    **else:**  L **=** Matrix**(**A**.**row**,** A**.**col**,** A**.**mainCol**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  L**.**body**[**i**][**j**]** **=** A**.**body**[**i**][**j**]**  **if** i **<=** j**:**  L**.**body**[**i**][**j**]** **=** 0    U **=** Matrix**(**A**.**row**,** A**.**col**,** A**.**mainCol**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  U**.**body**[**i**][**j**]** **=** A**.**body**[**i**][**j**]**  **if** i **>=** j**:**  U**.**body**[**i**][**j**]** **=** 0    D **=** Matrix**(**A**.**row**,** A**.**col**,** A**.**mainCol**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  D**.**body**[**i**][**j**]** **=** A**.**body**[**i**][**j**]**  **if** i **!=** j**:**  D**.**body**[**i**][**j**]** **=** 0    step1 **=** L**.**matrixConstMulti**(**omega**)**  step2 **=** D**.**matrixAdd**(**step1**)**  step3 **=** step2**.**matrixInversion**()** # mark    step4 **=** D**.**matrixConstMulti**(-**1 **\*** omega**)**  step5 **=** U**.**matrixConstMulti**(-**1 **\*** omega**)**  step6 **=** D**.**matrixAdd**(**step4**)**  step7 **=** step6**.**matrixAdd**(**step5**)** # mark    step8 **=** b**.**matrixConstMulti**(**omega**)**    end\_B **=** step3**.**matrixMulti**(**step7**)**  end\_b **=** step3**.**matrixMulti**(**step8**)**    ans **=** **(**end\_B**,** end\_b**)**    self**.**B **=** ans**[**0**]**  self**.**b **=** ans**[**1**]**  self**.**x **=** Matrix**(**x0**.**row**,** x0**.**col**,** x0**.**mainCol**)**    **for** i **in** range**(**x0**.**row**):**  **for** j **in** range**(**x0**.**col**):**  self**.**x**.**body**[**i**][**j**]** **=** x0**.**body**[**i**][**j**]**    **def** GS\_op1**(**self**):**  """Do one time iteration with GS method or Relaxation method.    self.x will be changed after this operation.  """  **for** i **in** range**(**self**.**x**.**row**):**  tmp **=** 0  **for** j **in** range**(**self**.**B**.**col**):**  tmp **+=** self**.**B**.**body**[**i**][**j**]** **\*** self**.**x**.**body**[**j**][**0**]**  self**.**x**.**body**[**i**][**0**]** **=** tmp **+** self**.**b**.**body**[**i**][**0**]**    **def** jacobiMethod**(**self**,**accuracy**):**  """The iteration would not stop by iter times.    It will stop while the accuracy gets.  """  self**.**\_\_init\_\_**(**A**,**b**,**x0**)** # omega uses the default setting    save\_tmp **=** self**.**x**.**matrixConstMulti**(-**1**)**  # save the original value for comparing.    step1 **=** self**.**B**.**matrixMulti**(**self**.**x**)**  step2 **=** step1**.**matrixAdd**(**self**.**b**)**    self**.**x **=** step2  # update the value of self.x    p **=** step2**.**matrixAdd**(**save\_tmp**)**  accu **=** p**.**matrixNormVal**()**    iter\_deepth **=** 1    **while(**accu **>** accuracy**):**  save\_tmp **=** self**.**x**.**matrixConstMulti**(-**1**)**  # save the original value for comparing.    step1 **=** self**.**B**.**matrixMulti**(**self**.**x**)**  step2 **=** step1**.**matrixAdd**(**self**.**b**)**    self**.**x **=** step2  # update the value of self.x    p **=** step2**.**matrixAdd**(**save\_tmp**)**  accu **=** p**.**matrixNormVal**()**    iter\_deepth **+=** 1    **return** **(**self**.**x**,** iter\_deepth**)**    **def** gsMethod**(**self**,** accuracy**):**  """Iteration will stop when the accuracy gets."""    self**.**\_\_init\_\_**(**A**,**b**,**x0**)**    save\_tmp **=** self**.**x**.**matrixConstMulti**(-**1**)**  # save the original value for comparing.    self**.**GS\_op1**()**  # self.x has been changed auto    p **=** self**.**x**.**matrixAdd**(**save\_tmp**)**  accu **=** p**.**matrixNormVal**()**    iter\_deepth **=** 1    **while(**accu **>** accuracy**):**  save\_tmp **=** self**.**x**.**matrixConstMulti**(-**1**)**  # save the original value for comparing.    self**.**GS\_op1**()**  # self.x has been changed    p **=** self**.**x**.**matrixAdd**(**save\_tmp**)**  accu **=** p**.**matrixNormVal**()**    iter\_deepth **+=** 1    **return(**self**.**x**,** iter\_deepth**)**    **def** relaxMethod**(**self**,** omega**,** accuracy**):**  """Iteration will stop when the accuracy gets."""    self**.**\_\_init\_\_**(**A**,**b**,**x0**,** omega**)** # omega uses the default setting    save\_tmp **=** self**.**x**.**matrixConstMulti**(-**1**)**  # save the original value for comparing.    step1 **=** self**.**B**.**matrixMulti**(**self**.**x**)**  step2 **=** step1**.**matrixAdd**(**self**.**b**)**    self**.**x **=** step2  # update the value of self.x    p **=** step2**.**matrixAdd**(**save\_tmp**)**  accu **=** p**.**matrixNormVal**()**    iter\_deepth **=** 1    **while(**accu **>** accuracy**):**  save\_tmp **=** self**.**x**.**matrixConstMulti**(-**1**)**  # save the original value for comparing.    step1 **=** self**.**B**.**matrixMulti**(**self**.**x**)**  step2 **=** step1**.**matrixAdd**(**self**.**b**)**    self**.**x **=** step2  # update the value of self.x    p **=** step2**.**matrixAdd**(**save\_tmp**)**  accu **=** p**.**matrixNormVal**()**    iter\_deepth **+=** 1  **return** **(**self**.**x**,** iter\_deepth**)**  **def** Plot**(**A**,** b**,** x0**,** e**=**1e-16**):**  """A pure function for ploting.    e is the target accuracy.  """    ERROR **=** list**()**  tmp **=** 1  **while** tmp **>** e**:**  tmp **/=** 10  ERROR**.**append**(**tmp**)**    deepth\_1 **=** list**()**  **for** i **in** ERROR**:**  J **=** MatrixIterMethods**(**A**,** b**,** x0**)**  M\_1 **=** J**.**relaxMethod**(**1.0**,** i**)**  # Relaxation factor is 1.0, equal to GSM    deepth\_1**.**append**(**M\_1**[**1**])**    deepth\_2 **=** list**()**  **for** i **in** ERROR**:**  M\_1 **=** J**.**relaxMethod**(**1.03**,** i**)**  # Relaxation factor is 1.03  deepth\_2**.**append**(**M\_1**[**1**])**    deepth\_3 **=** list**()**  **for** i **in** ERROR**:**  M\_1 **=** J**.**relaxMethod**(**1.1**,**i**)**  # Relaxation factor is 1.1  deepth\_3**.**append**(**M\_1**[**1**])**    **import** math  **for** i **in** range**(**len**(**ERROR**)):**  ERROR**[**i**]** **=** **-**1 **\*** math**.**log10**(**ERROR**[**i**])**    pl**.**grid**()**  pl**.**title**(**"Same Accuracy: Iteration deepth and Omega's Value"**,**fontsize**=**16**)**  pl**.**plot**(**ERROR**,** deepth\_1**,** 'o-r'**,**label **=** 'omega is 1.00'**)**  pl**.**plot**(**ERROR**,** deepth\_2**,** 'o-g'**,**label **=** 'omega is 1.03'**)**  pl**.**plot**(**ERROR**,** deepth\_3**,** 'o-b'**,**label **=** 'omega is 1.10'**)**  pl**.**legend**()**  pl**.**xlabel**(**'Level of Accuracy'**)**  pl**.**ylabel**(**'Iteration Deepth'**)**  pl**.**show**()**  **if** \_\_name\_\_ **==** "\_\_main\_\_"**:**    A **=** Matrix**(**6**,** 6**)**  A**.**body **=** **[[**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**]]**    b **=** Matrix**(**6**,** 1**)**  b**.**body **=** **[[**0**],[**5**],[**0**],[**6**],[-**2**],[**6**]]**    x0 **=** Matrix**(**6**,** 1**)**  x0**.**body **=** **[[**1**],[**1**],[**1**],[**1**],[**1**],[**1**]]**    Plot**(**A**,** b**,** x0**,** 1e-20**)**    A **=** Matrix**(**3**,** 3**)**  A**.**body **=** **[[**4**,** **-**1**,** 0**]**\  **,[-**1**,** 4**,** **-**1**]**\  **,[** 0**,** **-**1**,** 4**]]**  b **=** Matrix**(**3**,** 1**)**  b**.**body **=** **[[**1**],[**4**],[-**3**]]**  x0 **=** Matrix**(**3**,** 1**)**  x0**.**body **=** **[[**1**],[**1**],[**1**]]**  Plot**(**A**,** b**,** x0**,** 1e-20**)** |

Code Box 5

### 输出结果



输出结果 5 三阶矩阵

其实，还可以对第4题进行作图比较：



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。

### 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。

# 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。

# 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.