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：线性方程组的直接法 | 学号：20151910042 | 上机实践日期：2017-10-16 |
| 上机实践编号：No.01 | 组号： | 上机实践时间：13:22 |

# 实验目的

1. 通过对所学的线性方程组直接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，发现数值方法与符号方法的区别，并形成专业思维。

# 实验内容

1. 编程实现高斯-若尔当列主元消元法；

2. 编程实现高斯-若尔当全主元消元法；

3. 任选一种方案，Doolittle分解或者Crout分解，编程实现矩阵的LU分解；

4. 编程实现三对角线矩阵的稀疏方式存储，然后对其进行LU分解。

# 实验平台

Windows 10 1703 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（辅助）；

EndNote X8 文献管理。

# 实验记录与实验结果分析

## 1题

编程实现：用高斯-若尔当列主元消元法求下列方程的解[1]：

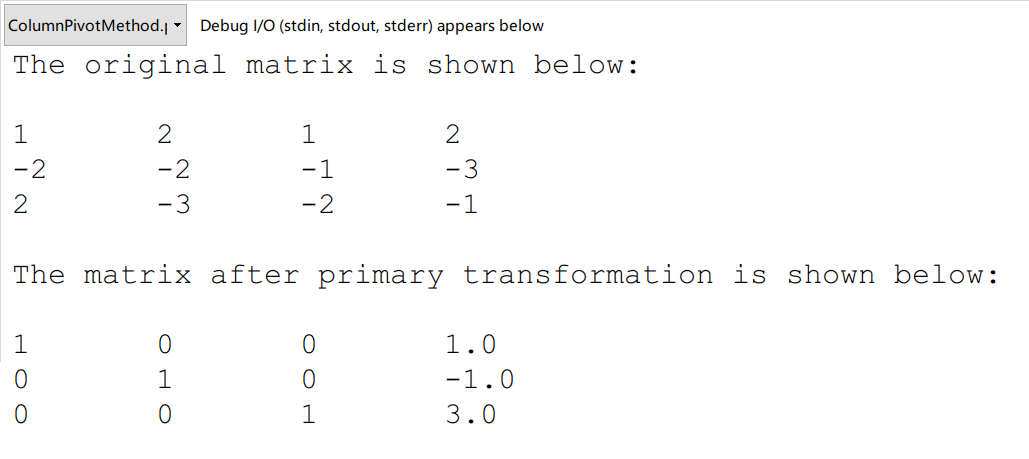
**解答：**

### 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 | # filename: 3.1 ColumnPivotMethod.py  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is 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**):**  """If the two matrix are not in the same form, return false,  else return true.    It is useful in the next functions.  """  **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** add**(**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** constMulti**(**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** 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** Peel**(**self**,**layer**):**  """Getting part of the matrix."""  B **=** Matrix**(**self**.**row**-**layer**,**self**.**col**-**layer**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**-**layer**):**  **for** j **in** range**(**self**.**col**-**layer**):**  B**.**body**[**i**][**j**]** **=** A**.**body**[**i **+** layer**][**j **+** layer**]**  **return** B    **def** combine**(**self**,**B**,**layer**):**  """A long with method Peel."""  **for** i **in** range**(**self**.**row**-**layer**):**  **for** j **in** range**(**self**.**col**-**layer**):**  A**.**body**[**i **+** layer**][**j **+** layer**]** **=** B**.**body**[**i**][**j**]**  **def** GaussJordanCol\_col**(**A**):**  Max **=** 0 # initialize the variable  position **=** 0 # the row number of the maximum value  **for** i **in** range**(**min**(**A**.**row**,**A**.**mainCol**)):**  **for** j **in** A**.**CheckedRow**:** # getting the max and position by a flow  **if** abs**(**Max**)** **<=** abs**(**A**.**body**[**j**][**i**]):**  Max **=** A**.**body**[**j**][**i**]**  position **=** j    A**.**CheckedRow**.**remove**(**position**)** # this row should not be checked again    A**.**matrixTransform**(**position**,** 1 **/** Max**)**  **for** j **in** range**(**A**.**row**):**  **if** j **!=** position**:**  A**.**matrixTransform**(**j**,**position**,-**1 **\*** A**.**body**[**j**][**i**])**  Max **=** 0 # reinitialize the variables  position **=** 0    begin **=** 0  **for** j **in** range**(**A**.**mainCol**):**  **for** i **in** range**(**A**.**row**):**  **if** A**.**body**[**i**][**j**]** **==** 1**:**  A**.**matrixTransform**(**begin**,**i**,**'exchange'**)**  begin **+=** 1  **return** A  """-------------------------my Main Function-------------------------"""  A **=** Matrix**(**3**,**6**,**3**)**  A**.**body**[**0**]** **=** **[**1**,**2**,**3**,**1**,**0**,**0**]**  A**.**body**[**1**]** **=** **[**2**,**4**,**5**,**0**,**1**,**0**]**  A**.**body**[**2**]** **=** **[**3**,**5**,**6**,**0**,**0**,**1**]**  **print(**'The original matrix is shown below:\n'**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  **print(**int**(**A**.**body**[**i**][**j**]),**'\t'**,**end**=**''**)**  **print(**''**)**  A **=** GaussJordanCol\_col**(**A**)**  **print(**'\nThe matrix after primary transformation is shown below:\n'**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  **if** A**.**body**[**i**][**j**]** **==** 0.0**:**  A**.**body**[**i**][**j**]** **=** 0  **if** A**.**body**[**i**][**j**]** **==** **-**0.0**:**  A**.**body**[**i**][**j**]** **=** 0  **if** A**.**body**[**i**][**j**]** **-** round**(**A**.**body**[**i**][**j**])** **==** 0**:**  A**.**body**[**i**][**j**]** **=** round**(**A**.**body**[**i**][**j**])**  **print(**round**(**A**.**body**[**i**][**j**],**4**),**'\t'**,**end**=**''**)**  **print(**''**)** |

Code Box 1 列主元消元法求线性方程组的解

### 输出结果



输出结果 1

### 代码分析

本Python代码采用面向对象[2]的方式写成，首先是定义了一个Matrix类，用来存储矩阵，该类中的所有元素都是public的，这样设置是为了以后的编程调用方便。

Matrix的实例A是一个三行四列的矩阵，其中最后一列是方程组中等号右边的元素，所以这个增广矩阵的计算应该遵循线性代数的法则，即在寻找列主元的时候，不能越界去增广列寻找；同样，一个已经找出过列主元的行，不应该再次存在列主元，所以要在以后的查找中剔除这一行。

最后，输出的结果应该保持美观性，所以按照系数矩阵经过变换之后为单位矩阵的样式进行了重新排序。

值得指出的是，Matrix的matrixTransform成员函数是用多态的方式写的，可以满足行数乘、行交换、行倍加等所有的矩阵初等变换方式。

另外，由于本线性方程组一定存在唯一解，所以在排序与寻找主元的过程中，并没有加以判断，所以这并不是一个通用的程序。

## 2题

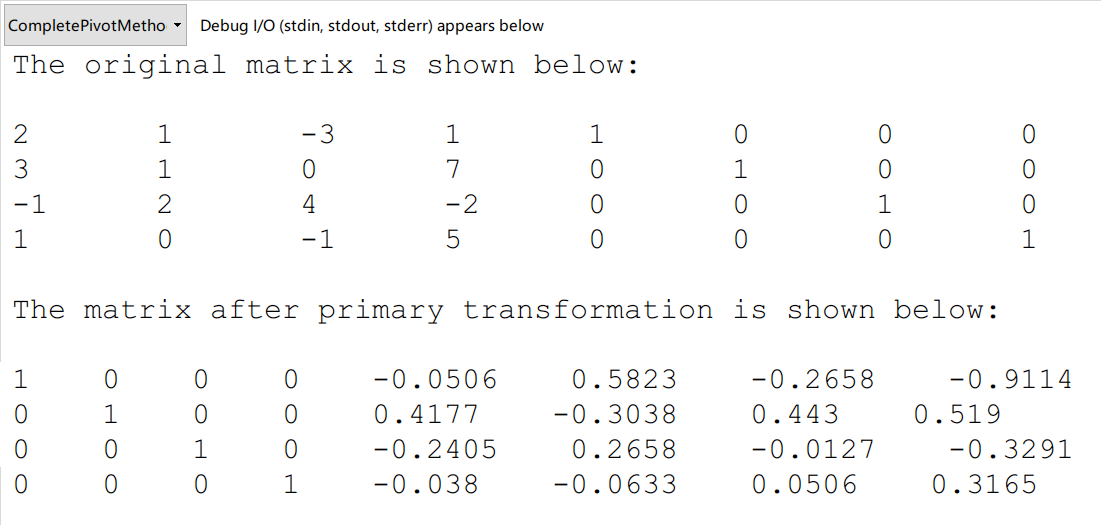
编程实现：请用高斯-若尔当全主元消元法求下列矩阵的逆矩阵[1]：

### 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 | # filename: 3.2 CompletePivotMethod.py  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is 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**):**  """If the two matrix are not in the same form, return false,  else return true.    It is useful in the next functions.  """  **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** add**(**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** constMulti**(**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** 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** Peel**(**self**,**layer**):**  """Getting part of the matrix."""  B **=** Matrix**(**self**.**row**-**layer**,**self**.**col**-**layer**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**-**layer**):**  **for** j **in** range**(**self**.**col**-**layer**):**  B**.**body**[**i**][**j**]** **=** A**.**body**[**i **+** layer**][**j **+** layer**]**  **return** B    **def** combine**(**self**,**B**,**layer**):**  """A long with method Peel."""  **for** i **in** range**(**self**.**row**-**layer**):**  **for** j **in** range**(**self**.**col**-**layer**):**  A**.**body**[**i **+** layer**][**j **+** layer**]** **=** B**.**body**[**i**][**j**]**  **def** GaussJordanCol\_complete**(**A**):**  Max **=** 0 # initialize the variable  position\_row **=** 0 # the row number of the maximum value  position\_col **=** 0    **for** i **in** range**(**A**.**row**):**    **for** j **in** A**.**CheckedRow**:**  **for** k **in** range**(**A**.**mainCol**):** # not in all the columns  **if** abs**(**Max**)** **<=** abs**(**A**.**body**[**j**][**k**]):**  Max **=** A**.**body**[**j**][**k**]**  position\_row **=** j  position\_col **=** k    A**.**matrixTransform**(**position\_row**,** 1 **/** Max**)**    A**.**CheckedRow**.**remove**(**position\_row**)**    **for** j **in** range**(**A**.**row**):**  **if** j **!=** position\_row**:**  A**.**matrixTransform**(**j**,**position\_row**,-**1 **\*** A**.**body**[**j**][**position\_col**])**    Max **=** 0  position\_row **=** 0  position\_col **=** 0    begin **=** 0  **for** j **in** range**(**A**.**mainCol**):**  **for** i **in** range**(**A**.**row**):**  **if** A**.**body**[**i**][**j**]** **==** 1**:**  A**.**matrixTransform**(**begin**,**i**,**'exchange'**)**  begin **+=** 1  **return** A  """-------------------------my Main Function-------------------------"""  A **=** Matrix**(**4**,**8**,**4**)**  A**.**body**[**0**]** **=** **[**2**,**1**,-**3**,**1**,**1**,**0**,**0**,**0**]**  A**.**body**[**1**]** **=** **[**3**,**1**,**0**,**7**,**0**,**1**,**0**,**0**]**  A**.**body**[**2**]** **=** **[-**1**,**2**,**4**,-**2**,**0**,**0**,**1**,**0**]**  A**.**body**[**3**]** **=** **[**1**,**0**,-**1**,**5**,**0**,**0**,**0**,**1**]**  **print(**'The original matrix is shown below:\n'**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  **print(**int**(**A**.**body**[**i**][**j**]),**'\t'**,**end**=**''**)**  **print(**''**)**  A **=** GaussJordanCol\_complete**(**A**)**  **print(**'\nThe matrix after primary transformation is shown below:\n'**)**  **for** i **in** range**(**4**):**  **for** j **in** range**(**8**):**  **if** A**.**body**[**i**][**j**]** **==** 0.0**:**  A**.**body**[**i**][**j**]** **=** 0  **if** A**.**body**[**i**][**j**]** **==** **-**0.0**:**  A**.**body**[**i**][**j**]** **=** 0  **if** A**.**body**[**i**][**j**]** **-** round**(**A**.**body**[**i**][**j**])** **==** 0**:**  A**.**body**[**i**][**j**]** **=** round**(**A**.**body**[**i**][**j**])**  **print(**round**(**A**.**body**[**i**][**j**],**4**),**' '**,**end**=**''**)**  **print(**''**)** |

Code Box 2 全主元消元法求矩阵的逆

### 输出结果



输出结果 2

### 代码分析

此段代码是在Code Box 1的基础上略微修改而得的，由于全主元的位置没有固定性，所以要把每一次找到的全主元的行列坐标都存储下来。其余部分与Code Box 1相同。

## 3题

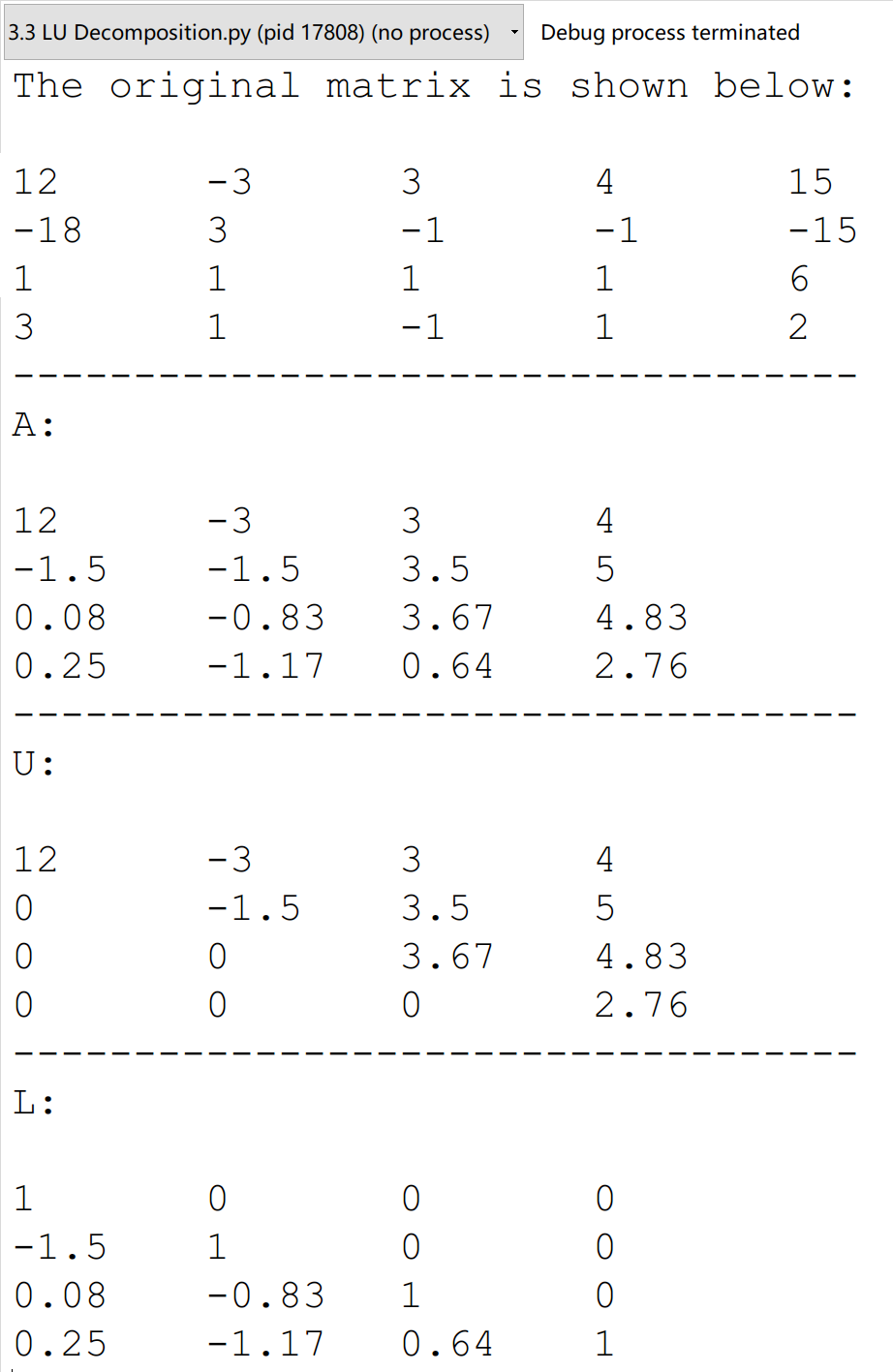
编程实现：利用LU分解法（Doolittle），求的数值解。

### 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 | # filename: 3.3 LU Decomposition.py  **class** **Matrix:**  """Abatrct class representing a Matrix.    The internal structure is 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**):**  """If the two matrix are not in the same form, return false,  else return true.    It is useful in the next functions.  """  **if** self**.**row **!=** e**.**row **or** self**.**col **!=** e**.**col**:**  **return** **False**  **else:**  **return** **True**  **def** add**(**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** constMulti**(**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** 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** Peel**(**self**,**layer**):**  """Getting part of the matrix."""  B **=** Matrix**(**self**.**row**-**layer**,**self**.**col**-**layer**,**self**.**mainCol**)**  **for** i **in** range**(**self**.**row**-**layer**):**  **for** j **in** range**(**self**.**col**-**layer**):**  B**.**body**[**i**][**j**]** **=** A**.**body**[**i **+** layer**][**j **+** layer**]**  **return** B    **def** combine**(**self**,**B**,**layer**):**  """A long with method Peel."""  **for** i **in** range**(**self**.**row**-**layer**):**  **for** j **in** range**(**self**.**col**-**layer**):**  A**.**body**[**i **+** layer**][**j **+** layer**]** **=** B**.**body**[**i**][**j**]**  **def** LU\_Decompose\_Doolittle**(**A**):**  """Decompose matrix A with Dolittle method.    In this function, I use devide and conquer method.  """  **for** i **in** range**(**A**.**col**-**1**):**  **if** i **==** 0**:**  **for** j **in** range**(**1**,**A**.**row**):**  A**.**body**[**j**][**i**]** **=** A**.**body**[**j**][**i**]** **/** A**.**body**[**0**][**0**]**  **else:**  **for** j **in** range**(**A**.**col**-**i**):**  **for** k **in** range**(**i**):**  A**.**body**[**i**][**i**+**j**]** **=** A**.**body**[**i**][**i**+**j**]** **-** \  A**.**body**[**i**][**k**]** **\*** A**.**body**[**k**][**i**+**j**]**  **for** j **in** range**(**A**.**row**-**i**-**1**):**  **for** k **in** range**(**i**):**  A**.**body**[**i**+**j**+**1**][**i**]** **=** A**.**body**[**i**+**j**+**1**][**i**]** **-** \  A**.**body**[**i**+**j**+**1**][**k**]** **\*** A**.**body**[**k**][**i**]**  A**.**body**[**i**+**j**+**1**][**i**]** **=** A**.**body**[**i**+**j**+**1**][**i**]** **/** A**.**body**[**i**][**i**]**  **return** A  """-------------------------my Main Function-------------------------"""  A **=** Matrix**(**4**,**5**,**4**)**  A**.**body**[**0**]** **=** **[**12**,-**3**,**3**,**4**,**15**]**  A**.**body**[**1**]** **=** **[-**18**,**3**,-**1**,-**1**,-**15**]**  A**.**body**[**2**]** **=** **[**1**,**1**,**1**,**1**,**6**]**  A**.**body**[**3**]** **=** **[**3**,**1**,-**1**,**1**,**2**]**  **print(**'The original matrix is shown below:\n'**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  **print(**int**(**A**.**body**[**i**][**j**]),**'\t'**,**end**=**''**)**  **print(**''**)**  A **=** LU\_Decompose\_Doolittle**(**A**)**  B **=** Matrix**(**4**,**4**,**4**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**row**):**  B**.**body**[**i**][**j**]** **=** A**.**body**[**i**][**j**]**  L1 **=** Matrix**(**4**,**4**,**4**)**  **for** i **in** range**(**L1**.**col**):**  **for** j **in** range**(**L1**.**row**-**i**):**  L1**.**body**[**j**+**i**][**i**]** **=** B**.**body**[**j**+**i**][**i**]**  **for** i **in** range**(**L1**.**row**):**  L1**.**body**[**i**][**i**]** **=** 1  U **=** L1**.**constMulti**(-**1**).**add**(**B**)** # I am the best! professional!  **for** i **in** range**(**L1**.**row**):**  U**.**body**[**i**][**i**]** **+=** 1  **print(**'-----------------------------------\nA:\n'**)**  **for** i **in** range**(**4**):**  **for** j **in** range**(**4**):**  **if** B**.**body**[**i**][**j**]** **==** 0.0**:**  B**.**body**[**i**][**j**]** **=** 0  **if** B**.**body**[**i**][**j**]** **==** **-**0.0**:**  B**.**body**[**i**][**j**]** **=** 0  **if** B**.**body**[**i**][**j**]** **-** round**(**B**.**body**[**i**][**j**])** **==** 0**:**  B**.**body**[**i**][**j**]** **=** round**(**B**.**body**[**i**][**j**])**  **print(**round**(**B**.**body**[**i**][**j**],**2**),**'\t'**,**end**=**''**)**  **print(**''**)**  **print(**'-----------------------------------\nU:\n'**)**  **for** i **in** range**(**4**):**  **for** j **in** range**(**4**):**  **if** U**.**body**[**i**][**j**]** **==** 0.0**:**  U**.**body**[**i**][**j**]** **=** 0  **if** U**.**body**[**i**][**j**]** **==** **-**0.0**:**  A**.**body**[**i**][**j**]** **=** 0  **if** U**.**body**[**i**][**j**]** **-** round**(**U**.**body**[**i**][**j**])** **==** 0**:**  U**.**body**[**i**][**j**]** **=** round**(**U**.**body**[**i**][**j**])**  **print(**round**(**U**.**body**[**i**][**j**],**2**),**'\t'**,**end**=**''**)**  **print(**''**)**  **print(**'-----------------------------------\nL:\n'**)**  **for** i **in** range**(**4**):**  **for** j **in** range**(**4**):**  **if** L1**.**body**[**i**][**j**]** **==** 0.0**:**  L1**.**body**[**i**][**j**]** **=** 0  **if** L1**.**body**[**i**][**j**]** **==** **-**0.0**:**  L1**.**body**[**i**][**j**]** **=** 0  **if** L1**.**body**[**i**][**j**]** **-** round**(**L1**.**body**[**i**][**j**])** **==** 0**:**  L1**.**body**[**i**][**j**]** **=** round**(**L1**.**body**[**i**][**j**])**  **print(**round**(**L1**.**body**[**i**][**j**],**2**),**'\t'**,**end**=**''**)**  **print(**''**)** |

Code Box 3 LU分解

### 输出结果



输出结果 3

通过生成的LU矩阵，可以轻易得到，的数值解为，这也是解析解。

### 代码分析

代码仍旧是基于面向对象技术，程序的算法是基于紧凑法求解LU矩阵的形式化做法。

如此编程，相对而言比较节约内存。

## 4题

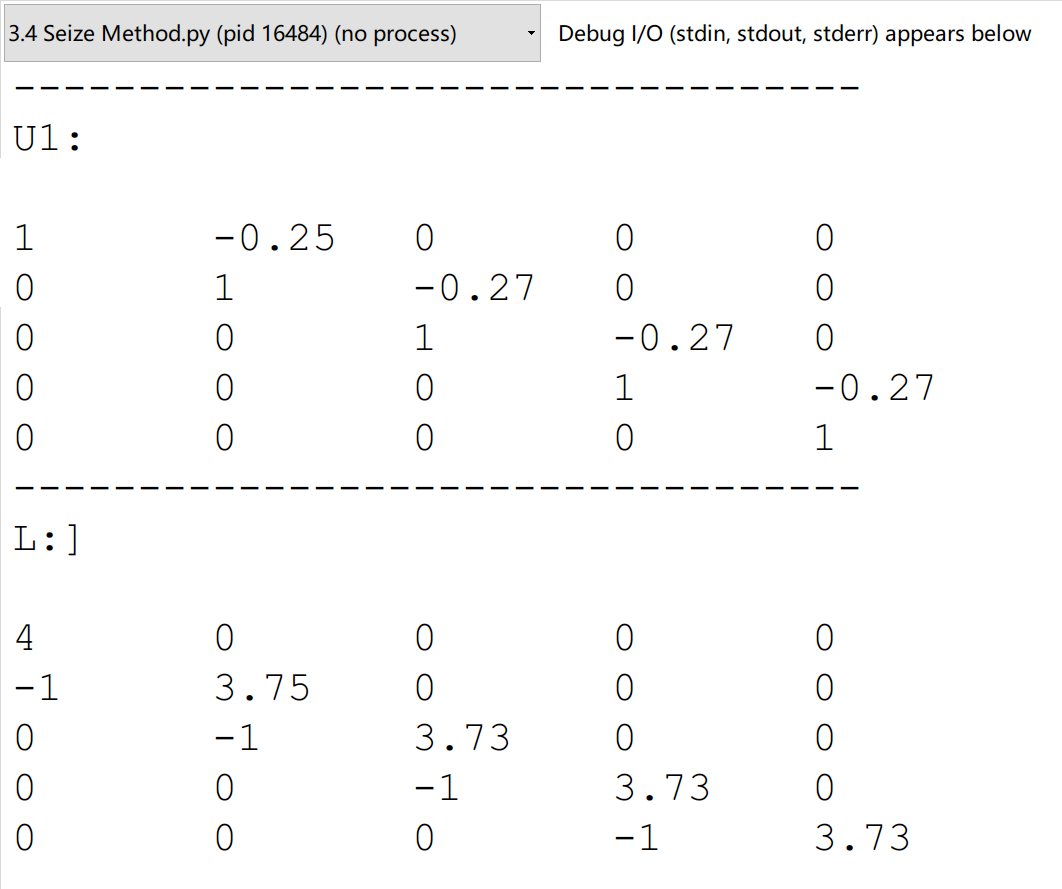
编程实现：用追赶法解下列严格对角优势的三对角线方程组，要求用稀疏格式存储矩阵，主内存占用为，其中为矩阵的行数。

### 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 | # filename: 3.4 Seize Method.py  **class** **ThreeTriangleMatrix:**  **def** \_\_init\_\_**(**self**,**a**,**b**,**c**,**m**):**  self**.**a **=** a  self**.**b **=** b  self**.**c **=** c  self**.**row **=** m  self**.**col **=** m  **def** calculate**(**self**):**  alpha **=** **[]**  beta **=** **[]**  gamma **=** self**.**a    alpha**.**append**(**self**.**b**)**  beta**.**append**(**self**.**c **/** alpha**[**0**])**    **for** i **in** range**(**1**,**self**.**row**-**1**):**  alpha**.**append**(**self**.**b **-** gamma **\*** beta**[**i**-**1**])**  beta**.**append**(**self**.**c **/** alpha**[**i**])**    alpha**.**append**(**self**.**b **-** gamma **\*** beta**[-**1**])**    **return** **[**gamma**,**alpha**,**beta**]**  A **=** ThreeTriangleMatrix**(-**1**,**4**,-**1**,**5**)**  A**.**calculate**()**  L **=** **[[**0 **for** i **in** range**(**A**.**row**)]** **for** j **in** range**(**A**.**col**)]**  **for** i **in** range**(**A**.**row**-**1**):**  L**[**i**+**1**][**i**]** **=** A**.**calculate**()[**0**]**  L**[**i**][**i**]** **=** A**.**calculate**()[**1**][**i**]**  L**[**A**.**row**-**1**][**A**.**row**-**1**]** **=** **(**A**.**calculate**()[**1**])[**A**.**row**-**1**]**  U1 **=** **[[**0 **for** i **in** range**(**A**.**row**)]** **for** j **in** range**(**A**.**col**)]**  **for** i **in** range**(**A**.**row**):**  U1**[**i**][**i**]** **=** 1  **for** i **in** range**(**A**.**row**-**1**):**  U1**[**i**][**i**+**1**]** **=** A**.**calculate**()[**2**][**i**]**  **print(**'----------------------------------\nU1:\n'**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  **print(**round**(**U1**[**i**][**j**],**2**),**'\t'**,**end**=**''**)**  **print** **(**''**)**  **print(**'----------------------------------\nL:]\n'**)**  **for** i **in** range**(**A**.**row**):**  **for** j **in** range**(**A**.**col**):**  **print(**round**(**L**[**i**][**j**],**2**),**'\t'**,**end**=**''**)**  **print(**''**)** |

Code Box 4 追赶公式求三对角线矩阵的LU分解

### 输出结果



输出结果 4

做得了LU分解，将数据导入MATLAB，可以迅速求出的数值解为

### 代码分析

此程序机械化比较高，所以比较简单，就是简单地翻译数学语言。

# 实验体会

在此次实验中，集中利用了MATLAB与Python 3进行程序设计。

主程序是采用Python 3进行编写，验算求解是利用MATLAB进行。当写的程序足够多之后，就会发现，真正重要的东西是数学思维，编程完全决定于数学思维的深度。如果一味追求编程的快感，很容易在这虚无的成就感中迷失自己。所以编程不如不编，能透彻理解数学思维已经很不容易了。

但是从另一个角度看，编程对数学思维又有很强的检验作用，尽管在数学思维之外，高技巧度编程又是另外广阔的一片天地。通过适度的编程，选择一类题目的典型例子进行编程，就可以在一个程序的设计、实现时间里，考验自己的一批数学思维的掌握牢固程度。

MATLAB的强大之处不在于它编程方便，而是两方面：友好的交互式控制台，众多高质量的程序包。正是这两点，使得MATLAB在工业中得到广泛的应用。但是在数值计算的学习上，利用Python或许是更好的选择。Python的数值计算依赖于数组，所以对于循环的考验非常高。Python不像MATLAB的基本运算都是基于矩阵那样地方便，而正是这不方便这一点，使得Python这个看起来更加拙劣的工具更适合新手。

除了工具，更重要的一点是编程。这四个程序的编写基本上没有遇到困难。如果说在理论方面已经理解得足够透彻，那么在编程上不应该存在任何思路困难。数值计算以及数学建模中的程序，很大程度上都是数学语言的直接翻译。这一点与DSA设计有相当大的差别。现在出现的一个问题是，编程趋向IDD模式，即IDE Driven，没有了编译器就不会单步调试，进而很难发现哪里出了差错。这不能说是思路理解得不到位，因为如果单纯是这个原因，那么在纸上演算就会遇到问题。但是很显然不是这样。以后的一个突破，可能就在这里——直接用文本编辑器进行代码书写，然后用IDE进行检验。

最后需要说明，因为有很多东西没有或者不能调用，所以这四个程序基本上是为题目量身定做。虽然面向对象设计里的几个成员函数具有通用性，但是总得来看，一些全局函数还是缺乏相应的判断。当然，如果不能运行，就从侧面说明输入的矩阵不合格。所以，这四个程序的通用性有待进一步挖掘，数据与程序能彻底分开，是判断程序通用性强弱的主要依据之一。

# 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] GOODRICH M T, TAMASSIA R, GOLDWASSER M H. Data Structures and Algorithms in Python [M]. U.S: Wiley & Sons, Inc., 2013.