

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

一、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。

二、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。

三、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。

四、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]：

$$\begin{cases} x_1 + 2x_2 + x_3 = 2 \\ -2x_1 - 2x_2 - x_3 = -3 \\ 2x_1 - 3x_2 - 2x_3 = -1 \end{cases}$$

解答：

程序代码

```

1  # filename: 3.1 ColumnPivotMethod.py
2
3  class Matrix:
4      """Abatrct class representing a Matrix.
5
6      The internal structure is two dimension list.
7      """
8      def __init__(self,m,n,mainCol):
9          """
10             self.row:          the row of the Matrix
11             self.col:          the collumn of the Matrix
12             self.CheckedRow:    if one row has been checked,
13                                it should not be checked again
14             self.body:          the internal storing structure
15             self.mainCol:       the coefficient matrix
16          """
17             self.row = m
18             self.col = n
19             self.CheckedRow = set(range(self.row))
20             self.body = [[0 for i in range(n)] for i in range(m)]
21             self.mainCol = mainCol
22
23     def getVal(self):
24         """Giving value to each element of the matrix.
25
26         Overwrite the original value zeros.
27         """
28         for i in range(self.row):
29             for j in range(self.col):
30                 self.body[i][j] = int(input())
31
32     def valid(self,e):
33         """If the two matrix are not in the same form, return false,
34         else return true.
35
36         It is useful in the next functions.
37         """
38         if self.row != e.row or self.col != e.col:
39             return False
40         else:
41             return True
42
43     def add(self,e):
44         """A methon does not used in the pivot PCA algorithm.
45
46         Maybe it will be used in other programs.
47         """
48         self.valid(e)
49         tmp = Matrix(self.row,self.col,self.mainCol)

```

```

50     for i in range(self.row): # deep copy
51         for j in range(self.col):
52             tmp.body[i][j] = self.body[i][j]
53
54     for i in range(self.row):
55         for j in range(self.col):
56             tmp.body[i][j] += e.body[i][j]
57     return tmp
58
59     def constMulti(self,const):
60         """A constant number multiple a matrix."""
61         tmp = Matrix(self.row,self.col,self.mainCol)
62         for i in range(self.row): # deep copy
63             for j in range(self.col):
64                 tmp.body[i][j] = self.body[i][j]
65
66         for i in range(self.row):
67             for j in range(self.col):
68                 tmp.body[i][j] *= const
69         return tmp
70
71     def matrixTransform(self,target_row_Number,source_row_Number,times=None):
72         """There is a big problem, every decimal number we see is stored in the
73         RAM with the binary platform.
74
75         I can import the decimal lib to solve this problem, but I did not!
76         """
77         if times == None: # special case of matrixTransform(TarRow,times)
78
79             times_tmp = source_row_Number
80             for j in range(self.col):
81                 self.body[target_row_Number][j] *= times_tmp
82             return
83         elif times == 'exchange':
84             for i in range(self.col):
85                 self.body[target_row_Number][i],\
86                 self.body[source_row_Number][i] \
87                 = self.body[source_row_Number][i],\
88                 self.body[target_row_Number][i]
89             return
90         else:
91             for i in range(self.col):
92                 self.body[target_row_Number][i] += \
93                 times * self.body[source_row_Number][i]
94
95     def Peel(self,layer):
96         """Getting part of the matrix."""
97         B = Matrix(self.row-layer,self.col-layer,self.mainCol)
98         for i in range(self.row-layer):

```

```

99         for j in range(self.col-layer):
100             B.body[i][j] = A.body[i + layer][j + layer]
101         return B
102
103     def combine(self,B,layer):
104         """A long with method Peel."""
105         for i in range(self.row-layer):
106             for j in range(self.col-layer):
107                 A.body[i + layer][j + layer] = B.body[i][j]
108
109     def GaussJordanCol_col(A):
110         Max = 0 # initialize the variable
111         position = 0 # the row number of the maximum value
112         for i in range(min(A.row,A.mainCol)):
113
114             for j in A.CheckedRow: # getting the max and position by a flow
115                 if abs(Max) <= abs(A.body[j][i]):
116                     Max = A.body[j][i]
117                     position = j
118
119             A.CheckedRow.remove(position) # this row should not be checked again
120
121             A.matrixTransform(position, 1 / Max)
122
123             for j in range(A.row):
124                 if j != position:
125                     A.matrixTransform(j,position,-1 * A.body[j][i])
126
127             Max = 0 # reinitialize the variables
128             position = 0
129
130         begin = 0
131         for j in range(A.mainCol):
132             for i in range(A.row):
133                 if A.body[i][j] == 1:
134                     A.matrixTransform(begin,i,'exchange')
135                     begin += 1
136         return A
137
138     """-----my Main Function-----"""
139
140     A = Matrix(3,6,3)
141     A.body[0] = [1,2,3,1,0,0]
142     A.body[1] = [2,4,5,0,1,0]
143     A.body[2] = [3,5,6,0,0,1]
144
145     print('The original matrix is shown below:\n')
146     for i in range(A.row):
147         for j in range(A.col):

```

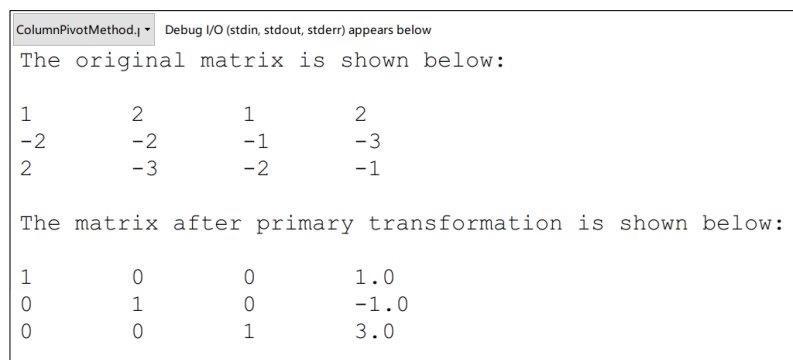
```

148     print(int(A.body[i][j]), '\t', end='')
149     print('')
150
151 A = GaussJordanCol_col(A)
152
153 print('\nThe matrix after primary transformation is shown below:\n')
154 for i in range(A.row):
155     for j in range(A.col):
156         if A.body[i][j] == 0.0:
157             A.body[i][j] = 0
158         if A.body[i][j] == -0.0:
159             A.body[i][j] = 0
160         if A.body[i][j] - round(A.body[i][j]) == 0:
161             A.body[i][j] = round(A.body[i][j])
162         print(round(A.body[i][j],4), '\t', end='')
163     print('')

```

Code Box 1 列主元消元法求线性方程组的解

输出结果



```

ColumnPivotMethod.py Debug I/O (stdin, stdout, stderr) appears below
The original matrix is shown below:

1      2      1      2
-2     -2     -1     -3
2      -3     -2     -1

The matrix after primary transformation is shown below:

1      0      0      1.0
0      1      0     -1.0
0      0      1      3.0

```

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]：

$$B = \begin{bmatrix} 2 & 1 & -3 & -1 \\ 3 & 1 & 0 & 7 \\ -1 & 2 & 4 & -2 \\ 1 & 0 & -1 & 5 \end{bmatrix}$$

程序代码

```

1  # filename: 3.2 CompletePivotMethod.py
2
3  class Matrix:
4      """Abatrct class representing a Matrix.
5
6      The internal structure is two dimension list.
7      """
8      def __init__(self,m,n,mainCol):
9          """
10             self.row:          the row of the Matrix
11             self.col:          the collumn of the Matrix
12             self.CheckedRow:    if one row has been checked,
13                                it should not be checked again
14             self.body:         the internal storing structure
15             self.mainCol:      the coefficient matrix
16          """
17             self.row = m
18             self.col = n
19             self.CheckedRow = set(range(self.row))
20             self.body = [[0 for i in range(n)] for i in range(m)]
21             self.mainCol = mainCol
22
23     def getVal(self):
24         """Giving value to each element of the matrix.
25
26         Overwrite the original value zeros.
27         """
28         for i in range(self.row):
29             for j in range(self.col):
30                 self.body[i][j] = int(input())
31
32     def valid(self,e):
33         """If the two matrix are not in the same form, return false,
34         else return true.
35
36         It is useful in the next functions.
37         """
38         if self.row != e.row or self.col != e.col:
39             return False
40         else:
41             return True
42

```

```

43     def add(self,e):
44         """A methon does not used in the pivot PCA algorithm.
45
46         Maybe it will be used in other programs.
47         """
48         self.valid(e)
49         tmp = Matrix(self.row,self.col,self.mainCol)
50         for i in range(self.row): # deep copy
51             for j in range(self.col):
52                 tmp.body[i][j] = self.body[i][j]
53
54         for i in range(self.row):
55             for j in range(self.col):
56                 tmp.body[i][j] += e.body[i][j]
57         return tmp
58
59     def constMulti(self,const):
60         """A constant number multiple a matrix."""
61         tmp = Matrix(self.row,self.col,self.mainCol)
62         for i in range(self.row): # deep copy
63             for j in range(self.col):
64                 tmp.body[i][j] = self.body[i][j]
65
66         for i in range(self.row):
67             for j in range(self.col):
68                 tmp.body[i][j] *= const
69         return tmp
70
71     def matrixTransform(self,target_row_Number,source_row_Number,times=None):
72         """There is a big problem, every decimal number we see is stored in the
73         RAM with the binary platform.
74
75         I can import the decimal lib to solve this problem, but I did not!
76         """
77         if times == None: # special case of matrixTransform(TarRow,times)
78
79             times_tmp = source_row_Number
80             for j in range(self.col):
81                 self.body[target_row_Number][j] *= times_tmp
82             return
83         elif times == 'exchange':
84             for i in range(self.col):
85                 self.body[target_row_Number][i],\
86                 self.body[source_row_Number][i] \
87                 = self.body[source_row_Number][i],\
88                 self.body[target_row_Number][i]
89             return
90         else:
91             for i in range(self.col):

```

```

92         self.body[target_row_Number][i] += \
93         times * self.body[source_row_Number][i]
94
95     def Peel(self,layer):
96         """Getting part of the matrix."""
97         B = Matrix(self.row-layer,self.col-layer,self.mainCol)
98         for i in range(self.row-layer):
99             for j in range(self.col-layer):
100                 B.body[i][j] = A.body[i + layer][j + layer]
101         return B
102
103     def combine(self,B,layer):
104         """A long with method Peel."""
105         for i in range(self.row-layer):
106             for j in range(self.col-layer):
107                 A.body[i + layer][j + layer] = B.body[i][j]
108
109     def GaussJordanCol_complete(A):
110         Max = 0 # initialize the variable
111         position_row = 0 # the row number of the maximum value
112         position_col = 0
113
114         for i in range(A.row):
115
116             for j in A.CheckedRow:
117                 for k in range(A.mainCol): # not in all the columns
118                     if abs(Max) <= abs(A.body[j][k]):
119                         Max = A.body[j][k]
120                         position_row = j
121                         position_col = k
122
123             A.matrixTransform(position_row, 1 / Max)
124
125             A.CheckedRow.remove(position_row)
126
127             for j in range(A.row):
128                 if j != position_row:
129                     A.matrixTransform(j,position_row,-1 * A.body[j][position_col])
130
131         Max = 0
132         position_row = 0
133         position_col = 0
134
135         begin = 0
136         for j in range(A.mainCol):
137             for i in range(A.row):
138                 if A.body[i][j] == 1:
139                     A.matrixTransform(begin,i,'exchange')
140                     begin += 1

```



```

141     return A
142
143     """-----my Main Function-----"""
144
145     A = Matrix(4,8,4)
146     A.body[0] = [2,1,-3,1,1,0,0,0]
147     A.body[1] = [3,1,0,7,0,1,0,0]
148     A.body[2] = [-1,2,4,-2,0,0,1,0]
149     A.body[3] = [1,0,-1,5,0,0,0,1]
150
151     print('The original matrix is shown below:\n')
152     for i in range(A.row):
153         for j in range(A.col):
154             print(int(A.body[i][j]),'\t',end='')
155         print('')
156
157     A = GaussJordanCol_complete(A)
158
159     print('\nThe matrix after primary transformation is shown below:\n')
160     for i in range(4):
161         for j in range(8):
162             if A.body[i][j] == 0.0:
163                 A.body[i][j] = 0
164             if A.body[i][j] == -0.0:
165                 A.body[i][j] = 0
166             if A.body[i][j] - round(A.body[i][j]) == 0:
167                 A.body[i][j] = round(A.body[i][j])
168             print(round(A.body[i][j],4),'\t',end='')
169     print('')

```

Code Box 2 全主元消元法求矩阵的逆

输出结果

CompletePivotMetho Debug I/O (stdin, stdout, stderr) appears below

The original matrix is shown below:

2	1	-3	1	1	0	0	0
3	1	0	7	0	1	0	0
-1	2	4	-2	0	0	1	0
1	0	-1	5	0	0	0	1

The matrix after primary transformation is shown below:

1	0	0	0	-0.0506	0.5823	-0.2658	-0.9114
0	1	0	0	0.4177	-0.3038	0.443	0.519
0	0	1	0	-0.2405	0.2658	-0.0127	-0.3291
0	0	0	1	-0.038	-0.0633	0.0506	0.3165

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 $[x_1, x_2, x_3, x_4]^T$ 的数值解。

$$\begin{bmatrix} 12 & -3 & 3 & 4 \\ -18 & 3 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ 3 & 1 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 15 \\ -15 \\ 6 \\ 2 \end{bmatrix}$$

程序代码

```

1  # filename: 3.3 LU Decomposition.py
2
3  class Matrix:
4      """Abatrct class representing a Matrix.
5
6      The internal structure is two dimension list.
7      """
8      def __init__(self,m,n,mainCol):
9          """
10             self.row:          the row of the Matrix
11             self.col:          the collumn of the Matrix
12             self.CheckedRow:    if one row has been checked,
13                               it should not be checked again
14             self.body:         the internal storing structure
15             self.mainCol:      the coefficient matrix
16          """
17             self.row = m
18             self.col = n
19             self.CheckedRow = set(range(self.row))
20             self.body = [[0 for i in range(n)] for i in range(m)]
21             self.mainCol = mainCol
22
23     def getVal(self):
24         """Giving value to each element of the matrix.
25
26         Overwrite the original value zeros.
27         """
28         for i in range(self.row):
29             for j in range(self.col):
30                 self.body[i][j] = int(input())
31
32     def valid(self,e):
33         """If the two matrix are not in the same form, return false,
34         else return true.
35
36         It is useful in the next functions.
37         """
38         if self.row != e.row or self.col != e.col:
39             return False
40         else:
41             return True
42

```

```

43     def add(self,e):
44         """A methon does not used in the pivot PCA algorithm.
45
46         Maybe it will be used in other programs.
47         """
48         self.valid(e)
49         tmp = Matrix(self.row,self.col,self.mainCol)
50         for i in range(self.row): # deep copy
51             for j in range(self.col):
52                 tmp.body[i][j] = self.body[i][j]
53
54         for i in range(self.row):
55             for j in range(self.col):
56                 tmp.body[i][j] += e.body[i][j]
57         return tmp
58
59     def constMulti(self,const):
60         """A constant number multiple a matrix."""
61         tmp = Matrix(self.row,self.col,self.mainCol)
62         for i in range(self.row): # deep copy
63             for j in range(self.col):
64                 tmp.body[i][j] = self.body[i][j]
65
66         for i in range(self.row):
67             for j in range(self.col):
68                 tmp.body[i][j] *= const
69         return tmp
70
71     def matrixTransform(self,target_row_Number,source_row_Number,times=None):
72         """There is a big problem, every decimal number we see is stored in the
73         RAM with the binary platform.
74
75         I can import the decimal lib to solve this problem, but I did not!
76         """
77         if times == None: # special case of matrixTransform(TarRow,times)
78
79             times_tmp = source_row_Number
80             for j in range(self.col):
81                 self.body[target_row_Number][j] *= times_tmp
82             return
83         elif times == 'exchange':
84             for i in range(self.col):
85                 self.body[target_row_Number][i],\
86                 self.body[source_row_Number][i] \
87                 = self.body[source_row_Number][i],\
88                 self.body[target_row_Number][i]
89             return
90         else:
91             for i in range(self.col):

```

```

92         self.body[target_row_Number][i] += \
93         times * self.body[source_row_Number][i]
94
95     def Peel(self, layer):
96         """Getting part of the matrix."""
97         B = Matrix(self.row-layer, self.col-layer, self.mainCol)
98         for i in range(self.row-layer):
99             for j in range(self.col-layer):
100                 B.body[i][j] = A.body[i + layer][j + layer]
101         return B
102
103     def combine(self, B, layer):
104         """A long with method Peel."""
105         for i in range(self.row-layer):
106             for j in range(self.col-layer):
107                 A.body[i + layer][j + layer] = B.body[i][j]
108
109     def LU_Decompose_Doolittle(A):
110         """Decompose matrix A with Doolittle method.
111
112         In this function, I use divide and conquer method.
113         """
114         for i in range(A.col-1):
115             if i == 0:
116                 for j in range(1, A.row):
117                     A.body[j][i] = A.body[j][i] / A.body[0][0]
118             else:
119                 for j in range(A.col-i):
120                     for k in range(i):
121                         A.body[i][i+j] = A.body[i][i+j] - \
122                         A.body[i][k] * A.body[k][i+j]
123                 for j in range(A.row-i-1):
124                     for k in range(i):
125                         A.body[i+j+1][i] = A.body[i+j+1][i] - \
126                         A.body[i+j+1][k] * A.body[k][i]
127                 A.body[i+j+1][i] = A.body[i+j+1][i] / A.body[i][i]
128         return A
129
130     """-----my Main Function-----"""
131
132     A = Matrix(4, 5, 4)
133     A.body[0] = [12, -3, 3, 4, 15]
134     A.body[1] = [-18, 3, -1, -1, -15]
135     A.body[2] = [1, 1, 1, 1, 6]
136     A.body[3] = [3, 1, -1, 1, 2]
137
138     print('The original matrix is shown below:\n')
139     for i in range(A.row):
140         for j in range(A.col):

```

```

141     print(int(A.body[i][j]), '\t', end='')
142     print('')
143
144 A = LU_Decompose_Doolittle(A)
145
146 B = Matrix(4,4,4)
147 for i in range(A.row):
148     for j in range(A.col):
149         B.body[i][j] = A.body[i][j]
150
151 L1 = Matrix(4,4,4)
152
153 for i in range(L1.col):
154     for j in range(L1.row-i):
155         L1.body[j+i][i] = B.body[j+i][i]
156 for i in range(L1.row):
157     L1.body[i][i] = 1
158
159 U = L1.constMulti(-1).add(B)    # I am the best! professional!
160 for i in range(L1.row):
161     U.body[i][i] += 1
162
163 print('-----\nA:\n')
164 for i in range(4):
165     for j in range(4):
166         if B.body[i][j] == 0.0:
167             B.body[i][j] = 0
168         if B.body[i][j] == -0.0:
169             B.body[i][j] = 0
170         if B.body[i][j] - round(B.body[i][j]) == 0:
171             B.body[i][j] = round(B.body[i][j])
172         print(round(B.body[i][j],2), '\t', end='')
173     print('')
174
175 print('-----\nU:\n')
176
177 for i in range(4):
178     for j in range(4):
179         if U.body[i][j] == 0.0:
180             U.body[i][j] = 0
181         if U.body[i][j] == -0.0:
182             U.body[i][j] = 0
183         if U.body[i][j] - round(U.body[i][j]) == 0:
184             U.body[i][j] = round(U.body[i][j])
185         print(round(U.body[i][j],2), '\t', end='')
186     print('')
187
188 print('-----\nL:\n')
189

```

```
190 for i in range(4):
191     for j in range(4):
192         if L1.body[i][j] == 0.0:
193             L1.body[i][j] = 0
194         if L1.body[i][j] == -0.0:
195             L1.body[i][j] = 0
196         if L1.body[i][j] - round(L1.body[i][j]) == 0:
197             L1.body[i][j] = round(L1.body[i][j])
198         print(round(L1.body[i][j],2),'\t',end='')
199     print('')
```

Code Box 3 LU 分解

输出结果

3.3 LU Decomposition.py (pid 17808) (no process) - Debug process terminated

```
The original matrix is shown below:

12      -3      3      4      15
-18      3     -1     -1     -15
1         1         1         1         6
3         1        -1         1         2
-----
A:

12      -3      3      4
-1.5    -1.5    3.5    5
0.08    -0.83   3.67   4.83
0.25    -1.17   0.64   2.76
-----
U:

12      -3      3      4
0       -1.5    3.5    5
0        0      3.67   4.83
0        0        0     2.76
-----
L:

1         0         0         0
-1.5      1         0         0
0.08     -0.83      1         0
0.25     -1.17     0.64      1
```

输出结果 3

通过生成的 LU 矩阵，可以轻易得到， $[x_1, x_2, x_3, x_4]^T$ 的数值解为 $[1, 2, 3, 0]$ ，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 $O(n)$ ，其中 n 为矩阵的行数。

$$\begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & -1 & 4 & -1 & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 100 \\ 200 \\ 200 \\ 200 \\ 100 \end{bmatrix}$$

程序代码

```

1  # filename: 3.4 Seize Method.py
2
3  class ThreeTriangleMatrix:
4      def __init__(self,a,b,c,m):
5          self.a = a
6          self.b = b
7          self.c = c
8          self.row = m
9          self.col = m
10
11     def calculate(self):
12         alpha = []
13         beta = []
14         gamma = self.a
15
16         alpha.append(self.b)
17         beta.append(self.c / alpha[0])
18
19         for i in range(1,self.row-1):
20             alpha.append(self.b - gamma * beta[i-1])
21             beta.append(self.c / alpha[i])
22
23         alpha.append(self.b - gamma * beta[-1])
24
25         return [gamma,alpha,beta]
26
27 A = ThreeTriangleMatrix(-1,4,-1,5)
28 A.calculate()
29
30 L = [[0 for i in range(A.row)] for j in range(A.col)]
31
32 for i in range(A.row-1):
33     L[i+1][i] = A.calculate()[0]
34     L[i][i] = A.calculate()[1][i]
35
36 L[A.row-1][A.row-1] = (A.calculate()[1])[A.row-1]
37
38 U1 = [[0 for i in range(A.row)] for j in range(A.col)]
39 for i in range(A.row):

```

```

40     U1[i][i] = 1
41
42     for i in range(A.row-1):
43         U1[i][i+1] = A.calculate()[2][i]
44
45     print('-----\nU1:\n')
46     for i in range(A.row):
47         for j in range(A.col):
48             print(round(U1[i][j],2), '\t', end=' ')
49         print(' ')
50
51     print('-----\nL:]\n')
52     for i in range(A.row):
53         for j in range(A.col):
54             print(round(L[i][j],2), '\t', end=' ')
55         print(' ')

```

Code Box 4 追赶公式求三对角线矩阵的 LU 分解

输出结果

```

3.4 Seize Method.py (pid 16484) (no process) - Debug I/O (stdin, stdout, stderr) appears below
-----
U1:
1      -0.25    0      0      0
0      1      -0.27    0      0
0      0      1     -0.27    0
0      0      0      1    -0.27
0      0      0      0      1
-----
L:]
4      0      0      0      0
-1     3.75    0      0      0
0     -1     3.73    0      0
0      0     -1     3.73    0
0      0      0     -1     3.73

```

输出结果 4

做得了 LU 分解，将数据导入 MATLAB，可以迅速求出 $[x_1, x_2, x_3, x_4, x_5]^T$ 的数值解为：

$$[46.15, 84.61, 92.30, 84.61, 46.15]^T$$

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。

五、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。

六、参考文献

- [1] 金一庆, 陈越, 王冬梅. 数值方法 [M]. 北京: 机械工业出版社, 2000.2.
- [2] GOODRICH M T, TAMASSIA R, GOLDWASSER M H. Data Structures and Algorithms in Python [M]. U.S: Wiley & Sons, Inc., 2013.