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： 插值法	**学号：** 20151910042	**上机实践日期：** 2017-11-29
上机实践编号： No.03	**组号：**	**最后修改时间：** 15:13

一、实验目的

1. 通过对所学的插值法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. 检验教材知识的理解与掌握程度。

二、实验内容

1. 编制用拉格朗日插值方法进行插值的程序；
2. 编制用牛顿插值方法进行插值的程序；
3. 要求牛顿插值方法在等距与不等距下两种情况下，程序可以进行自行选择，降低计算量。

三、实验平台

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

四、实验记录与实验结果分析

1 题

已知正弦函数表：

x_k	0.5	0.7	0.9	1.1	1.3	1.5	1.7	1.9
$\sin x_k$	0.4794	0.6442	0.7833	0.8912	0.9636	0.9975	0.9917	0.9463

编写程序，分别用拉格朗日插值和牛顿插值多项式计算$x_0=0.6$，0.8，1.0处的函数值$\sin(0.6)$、$\sin(0.8)$、$\sin(1.0)$的近似值$f(0.6)$、$f(0.8)$和$f(1.0)$。

解答：

程序代码：

```
1    # -*- coding: utf-8 -*-
2    """
3    Created on Thu Dec  7 12:14:49 2017
4
5    @author: Newton
```

```
6   """
7
8   """filename: 1.Interpolation Methods.py"""
9
10  class Interp:
11      """This class aims to make the interpolation method combined. each method
12      member of this class represents a method of interpolation.
13
14      +---------------+-----------+
15      |     Name      |   Method  |
16      +---------------+-----------+
17      |  Newton       |   Newton  |
18      |  Lagrange     |   Lagr    |
19      +---------------+-----------+
20
21      """
22
23      def __init__(self, x_known, y_known, x_unknown):
24          """The (x, y) points we have already known is essential to the
25          interpolation."""
26          self.x = x_known    # x_known is a list
27          self.y = y_known    # y_known is a list
28          self.ux = x_unknown # need to be computed
29
30          if len(self.x) != len(self.y):
31              raise ValueError("Bad input, len(x) should equal to len(y)")
32
33      def getDiffQuotientTab(self):
34          """Generate a matrix which represents the difference quotient table
35          of (x_known, y_known).
36          """
37          n = len(self.x) - 1
38
39          ans = [[None for i in range(n)] for i in range(n)]
40          # initialize it with default setting None.
41
42          for i in range(n):          # column
43              for j in range(i, n):   # row
44                  if i == 0:
45                      ans[j][i] = (self.y[j+1] - self.y[j]) \
46                      / (self.x[j+1] - self.x[j])
47                  else:
48                      ans[j][i] = (ans[j][i-1] - ans[j-1][i-1]) \
49                      / (self.x[j+1] - self.x[j-1])
50
51          return ans
52
53      def Newton(self):
54          """Need self.getDiffQuotientTab method.
```

```
55
56
57          """
58          step0 = self.getDiffQuotientTab()
59          step1 = list()
60          for i in range(len(self.x)-1):
61              step1.append(step0[i][i])
62
63          ans = [0 for i in range(len(self.ux))]
64
65          for i in range(len(self.ux)):       # generate a list of y we needed
66              for j in range(len(self.x)):    # a long polynomial function
67                  if j == 0:
68                      ans[i] += self.y[j]
69                  else:
70                      tmp = 1
71                      for k in range(j):
72                          tmp *= (self.ux[i] - self.x[k])
73                      tmp *= step1[j-1]
74
75                      ans[i] += tmp
76
77          return ans
78
79      def Lagr(self):
80          n = len(self.x)
81          m = len(self.ux)
82
83          ans = []
84
85          for i in range(m):          # all the x unknown
86              s = 0
87              for k in range(n):      # sum
88                  p = 1
89                  for j in range(n):  # multi
90                      if j != k:
91                          p = p * ((self.ux[i] - self.x[j]) / (self.x[k] - self.x[j]))
92                  s = s + p * self.y[k]
93              ans.append(s)
94          return ans
95
96  if __name__ == '__main__':
97
98      x = [0.5, 0.7, 0.9, 1.1, 1.3, 1.5, 1.7, 1.9]
99      y = [0.4794, 0.6442, 0.7833, 0.8912, 0.9636, 0.9975, 0.9917, 0.9463]
100     m = [0.6, 0.8, 1.0]
101     c = Interp(x, y, m)
102     ans_newton = c.Newton()
103     ans_lagr = c.Lagr()
```

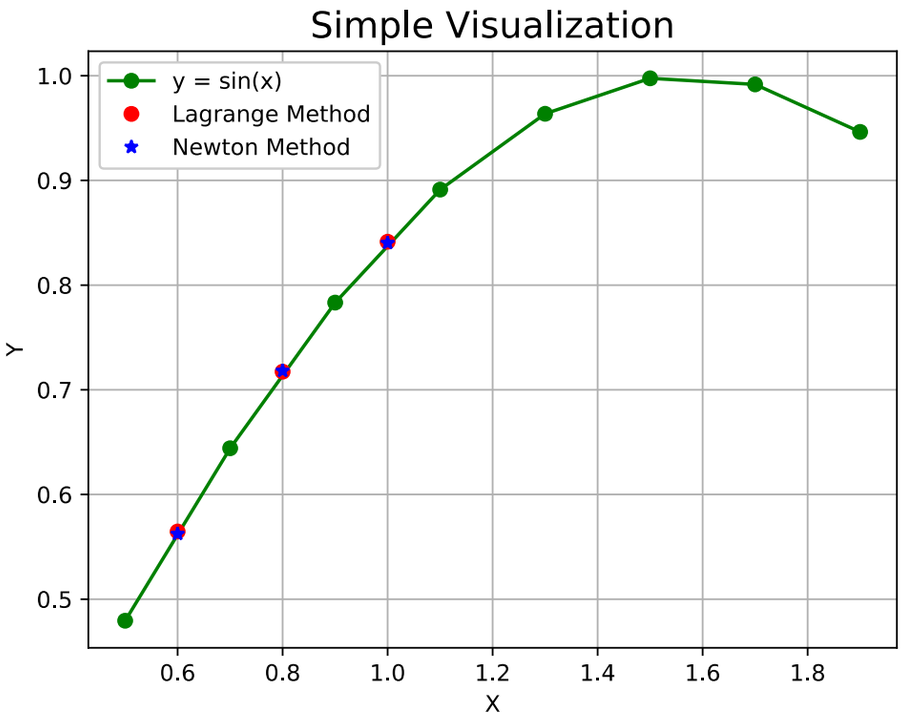
```
104
105     ans_n = list()
106     for i in ans_newton:
107         ans_n.append(round(i, 4))
108
109     ans_l = list()
110     for i in ans_lagr:
111         ans_l.append(round(i, 4))
112
113     print('+----------+---------+--------------+')
114     print('|  Method  |    x    |      y       |')
115     print('+----------+---------+--------------+')
116     print('|   Lagr   |  ',m[0], '  |   ', ans_l[0], '   |')
117     print('|          |  ',m[1], '  |   ', ans_l[1], '   |')
118     print('|          |  ',m[2], '  |   ', ans_l[2], '   |')
119     print('+----------+---------+--------------+')
120     print('|  Newton  |  ',m[0], '  |   ', ans_n[0], '   |')
121     print('|          |  ',m[1], '  |   ', ans_n[1], '   |')
122     print('|          |  ',m[2], '  |   ', ans_n[2], '   |')
123     print('+----------+---------+--------------+')
124
125     import matplotlib.pyplot as pl
126
127     pl.grid()
128     pl.title("Simple Visualization",fontsize=16)
129     pl.plot(x, y, 'o-g', label = 'y = sin(x)')
130     pl.plot(m, ans_l, 'ro', label = 'Lagrange Method')
131     pl.plot(m, ans_n, 'b*', label = 'Newton Method')
132     pl.legend()
133     pl.xlabel('X')
134     pl.ylabel('Y')
135     pl.show()
```

Code Box 1

输出结果：

```
1. Interpolation Meth  Debug process terminated
+----------+---------+--------------+
|  Method  |    x    |      y       |
+----------+---------+--------------+
|   Lagr   |   0.6   |    0.5646    |
|          |   0.8   |    0.7173    |
|          |   1.0   |    0.8414    |
+----------+---------+--------------+
|  Newton  |   0.6   |    0.5624    |
|          |   0.8   |    0.7181    |
|          |   1.0   |    0.8402    |
+----------+---------+--------------+
```

输出结果 1



输出结果 2

代码分析：

本段代码，是构建了一个简单的封装，可以将已知点作为参数输入，然后输入作为第三个参数的要估计的点的横坐标列表，最终返回一个列表，它储存着根据相应方法得到的插值数值。

2 题

已知[1]

x	0.4	0.5	0.6	0.7	0.8	0.9
$\ln x$	−0.916 291	−0.693 147	−0.510 826	−0.357 765	−0.223 144	−0.105 361

用牛顿后插公式求$\ln 0.78$的近似值，并根据 5 阶差分估计 4 阶公式的误差。

解答：

程序代码：

```
# -*- coding: utf-8 -*-
"""
Created on Sat Dec  9 19:18:27 2017

@author: Newton
"""

"""filename: 2. Newton left-interp Method.py"""

class Interp:
    """This class aims to make the interpolation method combined. each method
    member of this class represents a method of interpolation.

    +---------------+-----------+
    |     Name      |   Method  |
    +---------------+-----------+
    |  Newton       |   Newton  |
    |  Lagrange     |   Lagr    |
    +---------------+-----------+

    """

    def __init__(self, x_known, y_known, x_unknown):
        """The (x, y) points we have already known is essential to the
        interpolation."""
        self.x = x_known    # x_known is a list
        self.y = y_known    # y_known is a list
        self.ux = x_unknown # need to be computed

        if len(self.x) != len(self.y):
            raise ValueError("Bad input, len(x) should equal to len(y)")

    def getDiffTab(self):

        n = len(self.x) - 1

        ans = [[None for i in range(n)] for i in range(n)]
```

```
38
39          for i in range(n):          # column
40              for j in range(i, n):   # row
41                  if i == 0:
42                      ans[j][i] = self.y[j+1] - self.y[j]
43                  else:
44                      ans[j][i] = ans[j][i-1] - ans[j-1][i-1]
45          return ans
46
47      def getDiffQuotientTab(self):
48          """Generate a matrix which represents the difference quotient table
49          of (x_known, y_known).
50          """
51
52          equidistant = False     # equidistant is false by defualt
53
54          t = self.x[1] - self.x[0]
55          for i in range(1, len(self.x)-1):
56              if round(t, 1) == round(self.x[i+1] - self.x[i], 1):
57                  equidistant = True
58              else:
59                  equidistant = False
60                  break
61
62          if equidistant == False:
63              n = len(self.x) - 1
64
65              ans = [[None for i in range(n)] for i in range(n)]
66              # initialize it with default setting None.
67
68              for i in range(n):          # column
69                  for j in range(i, n):   # row
70                      if i == 0:
71                          ans[j][i] = (self.y[j+1] - self.y[j]) \
72                          / (self.x[j+1] - self.x[j])
73                      else:
74                          ans[j][i] = (ans[j][i-1] - ans[j-1][i-1]) \
75                          / (self.x[j+1] - self.x[j-1])
76              pass
77              return ans
78
79          else:
80              from math import factorial as fc
81              from math import pow as pow
82
83              n = len(self.x) - 1
84
85              ans = [[None for i in range(n)] for i in range(n)]
86
```

```
87              diffTab = self.getDiffTab()
88
89              for i in range(n):
90                  low = fc(i+1) * pow(t, i+1)
91                  up = diffTab[i][i]
92                  ans[i][i] = up/low
93
94              return ans
95
96      def Newton(self):
97          """Need self.getDiffQuotientTab method.
98
99
100         """
101         step0 = self.getDiffQuotientTab()
102         step1 = list()
103         for i in range(len(self.x)-1):
104             step1.append(step0[i][i])
105
106         ans = [0 for i in range(len(self.ux))]
107
108         for i in range(len(self.ux)):       # generate a list of y we needed
109             for j in range(len(self.x)):    # a long polynomial function
110                 if j == 0:
111                     ans[i] += self.y[j]
112                 else:
113                     tmp = 1
114                     for k in range(j):
115                         tmp *= (self.ux[i] - self.x[k])
116                     tmp *= step1[j-1]
117
118                     ans[i] += tmp
119
120         return ans
121
122     def Lagr(self):
123         n = len(self.x)
124         m = len(self.ux)
125
126         ans = []
127
128         for i in range(m):          # all the x unknown
129             s = 0
130             for k in range(n):      # sum
131                 p = 1
132                 for j in range(n):  # multi
133                     if j != k:
134                         p = p * ((self.ux[i] - self.x[j]) / (self.x[k] - self.x[j]))
135                 s = s + p * self.y[k]
```

```
            ans.append(s)
        return ans

if __name__ == '__main__':

    x = [0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
    y = [-0.916291, -0.693147, -0.510826, -0.357765, -0.223144, -0.105361]
    m = [0.41, 0.51, 0.61, 0.71, 0.81, 0.89]
    c = Interp(x, y, m)

    ans_newton = c.Newton()
    ans_lagr = c.Lagr()

    ans_n = list()
    for i in ans_newton:
        ans_n.append(round(i, 4))

    ans_l = list()
    for i in ans_lagr:
        ans_l.append(round(i, 4))

    print('+----------+----------+---------------+')
    print('|  Method  |    x     |       y       |')
    print('+----------+----------+---------------+')
    print('|   Lagr   |  ',m[0], '  |   ', ans_l[0], '   |')
    print('|          |  ',m[1], '  |   ', ans_l[1], '   |')
    print('|          |  ',m[2], '  |   ', ans_l[2], '   |')
    print('|          |  ',m[3], '  |   ', ans_l[3], '   |')
    print('|          |  ',m[4], '  |   ', ans_l[4], '   |')
    print('|          |  ',m[5], '  |   ', ans_l[5], '   |')
    print('+----------+----------+---------------+')
    print('|  Newton  |  ',m[0], '  |   ', ans_n[0], '   |')
    print('|          |  ',m[1], '  |   ', ans_n[1], '   |')
    print('|          |  ',m[2], '  |   ', ans_n[2], '   |')
    print('|          |  ',m[3], '  |   ', ans_n[3], '   |')
    print('|          |  ',m[4], '  |   ', ans_n[4], '   |')
    print('|          |  ',m[5], '  |   ', ans_n[5], '   |')
    print('+----------+----------+---------------+')

    import matplotlib.pyplot as pl

    pl.grid()
    pl.title("Simple Visualization",fontsize=16)
    pl.plot(x, y, 'o-g', label = 'y = sin(x)')
    pl.plot(m, ans_l, 'ro', label = 'Lagrange Method')
    pl.plot(m, ans_n, 'b-', label = 'Newton Method')
    pl.legend()
    pl.xlabel('X')
    pl.ylabel('Y')
```

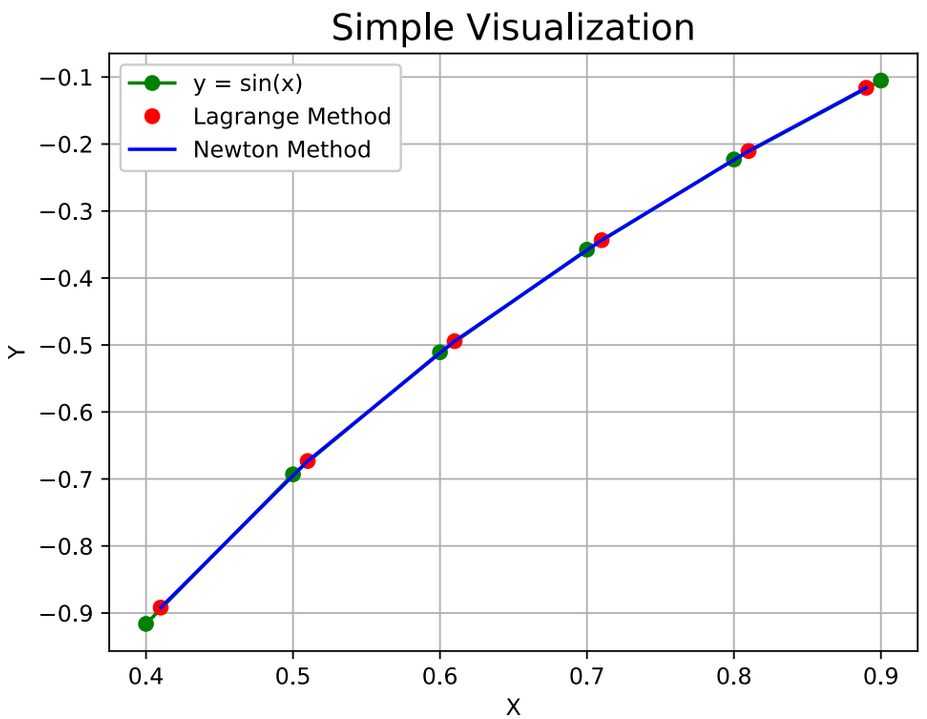
```
185     pl.show()
```

Code Box 2

输出结果：

```
2. Newton left-interp | Debug I/O (stdin, stdout, stderr) appears below
+----------+----------+---------------+
|  Method  |    x     |       y       |
+----------+----------+---------------+
|   Lagr   |   0.41   |    -0.8919    |
|          |   0.51   |    -0.6732    |
|          |   0.61   |    -0.4944    |
|          |   0.71   |    -0.3436    |
|          |   0.81   |    -0.2105    |
|          |   0.89   |    -0.1161    |
+----------+----------+---------------+
|  Newton  |   0.41   |    -0.8919    |
|          |   0.51   |    -0.6732    |
|          |   0.61   |    -0.4944    |
|          |   0.71   |    -0.3436    |
|          |   0.81   |    -0.2105    |
|          |   0.89   |    -0.1161    |
+----------+----------+---------------+
```

输出结果 3



输出结果 4

代码分析：

牛顿后插公式，就是在等距节点的基础上，简化了差商的计算，但是需要计算一个独立的差分表。差分表的构建需要依据前插还是后插，其中前插表得到的是一个上三角矩阵，后插公式得到的是一个下三角矩阵。得到了差分表，就可以根据对角线元素，构建差商表。然后根据题目 1 的已有代码，即可做出图像。

五、实验体会

本次实验难度较小，代码量不算大。

之前想过用 MATLAB 做 03 号实验，但是后来还是决定继续采用 Python3，首先是平台比较开放，其次是本次基本不涉及矩阵，就算是涉及到矩阵运算，也已经有了一个由我独立设计的比较完善的 Python 包，所以坚持 Python 可能是一个比较好的选择。

插值算法的核心在于解方程组。而在牛顿插值多项式中，引入差商概念，用差商推导出了一般的插值计算公式，同时给出了很明确的算法与误差估计。其形式与泰勒展开式非常相似，余项也和泰勒公式非常像。

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.