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

4.1 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$f(0.6)$，$f(0.8)$，$f(1.0)$。

解答：

4.1.1 程序代码

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Dec  7 12:14:49 2017

@author: Newton
"""

"""filename: 1.Interpolation Methods.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 getDiffQuotientTab(self):
        """Generate a matrix which represents the difference quotient table
        of (x_known, y_known).
        """
        n = len(self.x) - 1

        ans = [[None for i in range(n)] for i in range(n)]
        # initialize it with default setting None.

        for i in range(n):          # column
            for j in range(i, n):   # row
                if i == 0:
                    ans[j][i] = (self.y[j+1] - self.y[j]) \
                    / (self.x[j+1] - self.x[j])
```

```
                else:
                    ans[j][i] = (ans[j][i-1] - ans[j-1][i-1]) \
                    / (self.x[j+1] - self.x[j-1])

        return ans

    def Newton(self):
        """Need self.getDiffQuotientTab method.


        """
        step0 = self.getDiffQuotientTab()
        step1 = list()
        for i in range(len(self.x)-1):
            step1.append(step0[i][i])

        ans = [0 for i in range(len(self.ux))]

        for i in range(len(self.ux)):      # generate a list of y we needed
            for j in range(len(self.x)):    # a long polynomial function
                if j == 0:
                    ans[i] += self.y[j]
                else:
                    tmp = 1
                    for k in range(j):
                        tmp *= (self.ux[i] - self.x[k])
                    tmp *= step1[j-1]

                    ans[i] += tmp

        return ans

    def Lagr(self):
        n = len(self.x)
        m = len(self.ux)

        ans = []

        for i in range(m):          # all the x unknown
            s = 0
            for k in range(n):      # sum
                p = 1
                for j in range(n):  # multi
                    if j != k:
                        p = p * ((self.ux[i] - self.x[j]) / (self.x[k] - self.x[j]))
                s = s + p * self.y[k]
            ans.append(s)
        return ans

```

```python
if __name__ == '__main__':

    x = [0.5, 0.7, 0.9, 1.1, 1.3, 1.5, 1.7, 1.9]
    y = [0.4794, 0.6442, 0.7833, 0.8912, 0.9636, 0.9975, 0.9917, 0.9463]
    m = [0.6, 0.8, 1.0]
    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('+----------+---------+----------------+')
    print('|  Newton  |  ',m[0], '  |   ', ans_n[0], '   |')
    print('|          |  ',m[1], '  |   ', ans_n[1], '   |')
    print('|          |  ',m[2], '  |   ', ans_n[2], '   |')
    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')
    pl.show()
```

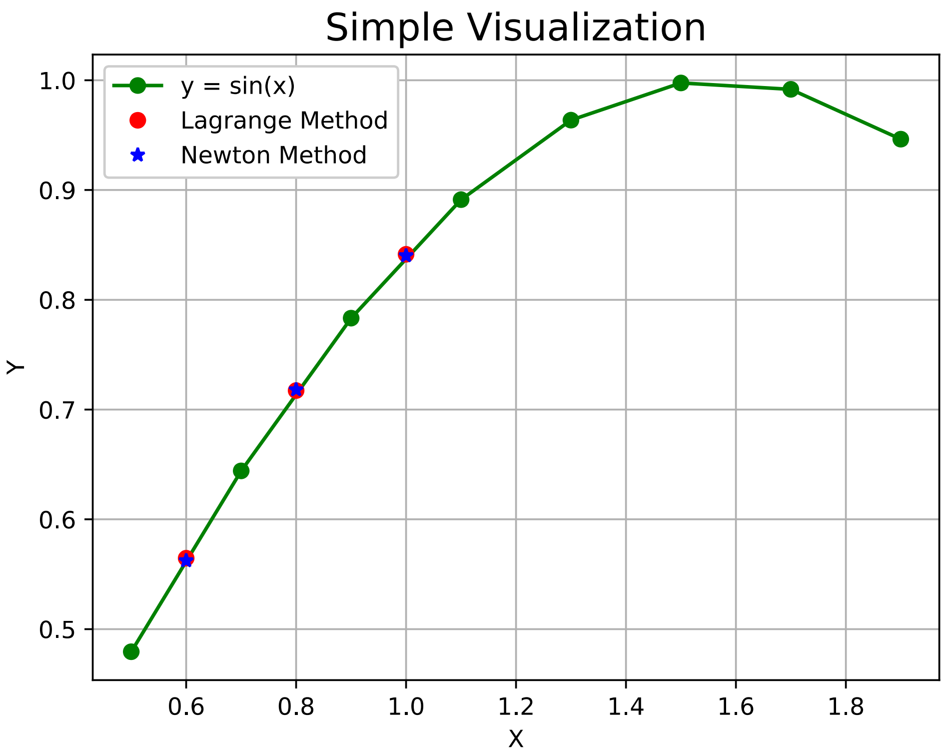
Code Box 1

4.1.2 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

4.1.3 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。

4.2 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：

4.2.1 程序代码

```python
# -*- 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)]

        for i in range(n):          # column
            for j in range(i, n):   # row
                if i == 0:
                    ans[j][i] = self.y[j+1] - self.y[j]
```

```
                else:
                    ans[j][i] = ans[j][i-1] - ans[j-1][i-1]
        return ans

    def getDiffQuotientTab(self):
        """Generate a matrix which represents the difference quotient table
        of (x_known, y_known).
        """

        equidistant = False     # equidistant is false by defualt

        t = self.x[1] - self.x[0]
        for i in range(1, len(self.x)-1):
            if round(t, 1) == round(self.x[i+1] - self.x[i], 1):
                equidistant = True
            else:
                equidistant = False
                break

        if equidistant == False:
            n = len(self.x) - 1

            ans = [[None for i in range(n)] for i in range(n)]
            # initialize it with default setting None.

            for i in range(n):          # column
                for j in range(i, n):   # row
                    if i == 0:
                        ans[j][i] = (self.y[j+1] - self.y[j]) \
                        / (self.x[j+1] - self.x[j])
                    else:
                        ans[j][i] = (ans[j][i-1] - ans[j-1][i-1]) \
                        / (self.x[j+1] - self.x[j-1])
            pass
            return ans

        else:
            from math import factorial as fc
            from math import pow as pow

            n = len(self.x) - 1

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

            diffTab = self.getDiffTab()

            for i in range(n):
                low = fc(i+1) * pow(t, i+1)
                up = diffTab[i][i]
```

```python
                ans[i][i] = up/low

            return ans

    def Newton(self):
        """Need self.getDiffQuotientTab method.


        """
        step0 = self.getDiffQuotientTab()
        step1 = list()
        for i in range(len(self.x)-1):
            step1.append(step0[i][i])

        ans = [0 for i in range(len(self.ux))]

        for i in range(len(self.ux)):       # generate a list of y we needed
            for j in range(len(self.x)):    # a long polynomial function
                if j == 0:
                    ans[i] += self.y[j]
                else:
                    tmp = 1
                    for k in range(j):
                        tmp *= (self.ux[i] - self.x[k])
                    tmp *= step1[j-1]

                    ans[i] += tmp

        return ans

    def Lagr(self):
        n = len(self.x)
        m = len(self.ux)

        ans = []

        for i in range(m):          # all the x unknown
            s = 0
            for k in range(n):      # sum
                p = 1
                for j in range(n):  # multi
                    if j != k:
                        p = p * ((self.ux[i] - self.x[j]) / (self.x[k] - self.x[j]))
                s = s + p * self.y[k]
            ans.append(s)
        return ans

if __name__ == '__main__':

```

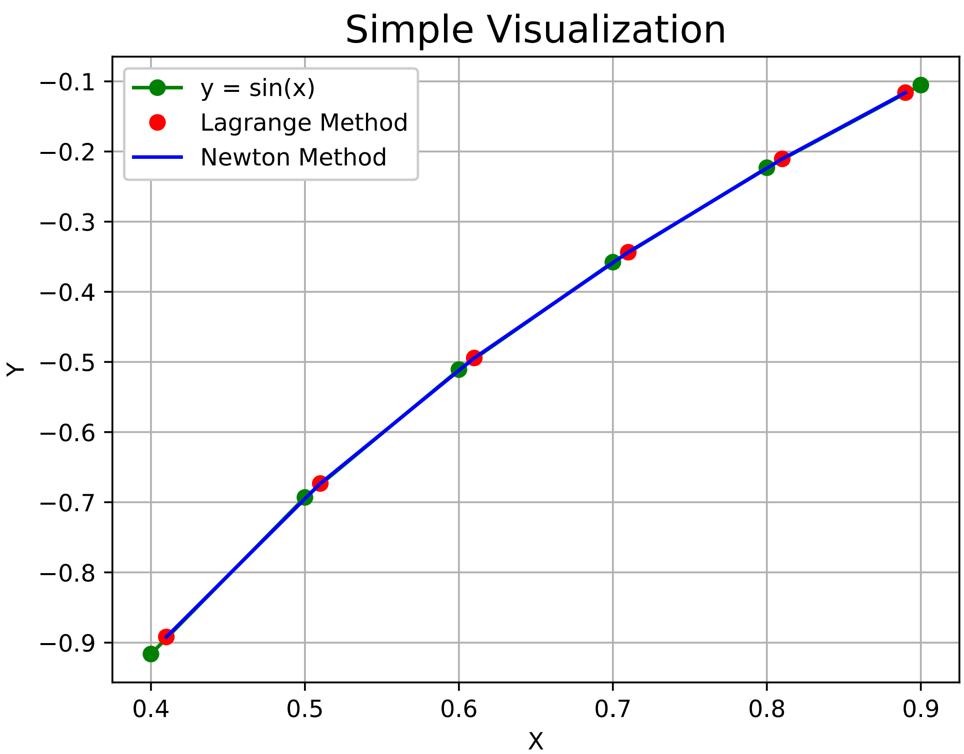
```python
    x = [0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
    y = [-0.916291, -0.693147, -0.510826, -0.357765, -0.223144, -0.105361]
    m = [0.41, 0.51, 0.61, 0.71, 0.81, 0.89]
    c = Interp(x, y, m)

    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')
    pl.show()
```

Code Box 2

4.2.2 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

4.2.3 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.