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： 数值积分	**学号：** 20151910042	**上机实践日期：** 2017-12-18
上机实践编号： No.05	**组号：**	**最后修改时间：** 20:25

一、 实验目的

1. 通过对所学的数值积分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. 通过编程，检验学习水平。

二、 实验内容

1. 编制辛普森公式的相关程序；
2. 编程实现用符合梯形公式与复合辛普森公式求积分。

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

分别用复合梯形公式和复合辛普森公式计算下列积分，并比较结果。[1]

$$\int_0^1 \frac{x}{4+x^2}\mathrm{d}x \qquad (n=8)$$

解答：

4.1.1 程序代码

```
# -*- coding: utf-8 -*-
"""
```

```
Created on Sat Dec  9 22:25:09 2017

@author: Newton
"""

"""filename: 1. Numerical Integration.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
```

```python

    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

class Integrate:
    """This class aims to compute the numerical integration of a function, or
    just some discrete points.
    """
    def __init__(self, x_min, x_max, function_name=None, step=None):
```

```
        """
        function_name:       the function needs to be calculated
        x_min:              the beginning of the range of x
        x_max:              the end of the range of x

        If the input is in this format, we can generate a list which represents
        the value of the function under step.

        if the number of inputs is 2, it means two list, x and y.
        """
        if function_name == None and step == None:
            self.x = x_min
            self.y = x_max
        else:
            from numpy import arange
            self.x = list(arange(x_min, x_max, step))      # this is a list
            self.y = list()
            for i in range(len(self.x)):
                self.y.append(function_name(self.x[i]))

    def Linear(self):
        ans = 0
        for i in range(len(self.x)-1):
            ans += (self.y[i] + self.y[i+1]) * (self.x[i+1] - self.x[i]) / 2
        return ans

    def Simpson(self):
        ans = 0
        point_m = list()
        for i in range(1, len(self.y), 2):
            point_m.append(self.y[i])
        s_m = sum(point_m)

        point_double = list()
        for i in range(2, len(self.y), 2):
            point_double.append(self.y[i])
        s_d = sum(point_double)

        ans = (self.y[0] + self.y[-1] + 4 * s_m + 2 * s_d) * (self.x[1] - self.x[0]) / 3
        return ans

if __name__ == '__main__':
    from math import sin as sin

    def func(x):
        y = x / (4 + x*x)
        return y

    c = Integrate(0, 1, func, .1/.8)
```

```
    print('+-----------+--------------------+')
    print('|  Method    |  Value               |')
    print('+-----------+--------------------+')
    print('|  Linear    | ', c.Linear(), '   |')
    print('+-----------+--------------------+')
    print('|  Simpson   | ', c.Simpson(), '     |')
    print('+-----------+--------------------+')
```

Code Box 1

4.1.2 输出结果

1. Numerical Integrati▾ Debug I/O (stdin, stdout, stderr) appears below

```
+-----------+--------------------+
|  Method   |  Value             |
+-----------+--------------------+
|  Linear   |  0.0874269446935   |
+-----------+--------------------+
|  Simpson  |  0.11088932243     |
+-----------+--------------------+
```

输出结果 1

4.1.3 代码分析

这段代码可以做两种输入，一种是给出积分上下界，同时给出函数名、步长，另一种是仅仅给出已经对应好的$(x,\ y)$点对 list，进行数值积分。积分方法有两种，第一是梯形公式积分，另一种是辛普森公式积分。

程序的输出，是调用相应方法所得的积分值，是一个数值。

五、 实验体会

这个章节的实验报告内容比较少。所以难度不大。

在实验过程中，针对辛普森公式进行编程。因为课本上给出的辛普森公式是针对等距分割的集合，所以当单纯输入两组数字的时候，一旦数字不等距，那么就无法使用课本给出的方法。针对这种问题，我做了一个新的方法，依据的原理还是插值算法。

在非等距节点中，进行分组：每两个相邻的区间划分为一组。在这个分组上，用拥有的三个点进行二次插值，然后从这个小区间组中找出 100 个或者更多的点，利用二次插值多项式进行求差值，将得到的 100 组点，进行线性积分。对整个区间完成计算，得到总的值。

在上述算法中，基本思想还是辛普森公式，不同的是，课本的算法是利用公式，将小区间组上的积分值用三个点的函数值表示了出来同时给出了误差估计，而这个算法却需要另想办法进行计算（在这里是采取插值，之后进行梯形公式积分）。不过遗憾的是这个方法的精度比线性积分还要差。可能导致的结果是龙格现象，这导致在这个区间上，100 个点对太过密集，积分的次数太高，在区间两侧不准确了。

而采用了教科书上的方法，进行计算之后，社区了一个 5 阶高阶无穷小量，尽管分割小区间，会在每个小区间上造成一定的误差，但是，误差却并不是很大，因为$\lim\limits_{n\to\infty} n(\frac{b-a}{n})^5=0$。

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.