实验报告一

专业____20 统计实验班_____学号__2020111509__姓名___徐之皓____

一、实验目的

该实验目的是：

1、 利用由梯形公式和辛普森公式诱导出的复合求积公式以及龙贝格加速算法计算积分，并比较各个算法的迭代速度，精确程度

2、 利用不同等距节点的函数值，分别模拟函数的一阶和二阶导数

实验的注意事项

1、 尽量以最简练的语言完成计算和迭代

2、 模拟结果要与真实值做对比，以检验是否合理和需要改进之处

二、实验题目

数值积分实验

1. 用复合梯形公式、复合辛普森公式、龙贝格公式求解下列定积分，要求绝对误差为$\epsilon = 0.5 \times 10^{-8}$，并将计算结果与准确解进行比较：

（1）$e^4 = \int_1^2 \frac{2}{3}x^3 e^{x^2} dx$；　　（2）$\ln 6 = \int_2^3 \frac{2x}{x^2-3} dx$.

2. 利用等距节点的函数值，求下列函数的一阶和二阶导数，分析方法的有效性，并用绘图软件绘出函数的图形，观察其特点。

（1）$y = \frac{1}{20}x^5 - \frac{11}{6}x^3,\ x \in [0,2]$；　　（2）$y = e^{-\frac{1}{x}},\ x \in [-2.5, -0.5]$.

三、实验原理

复合梯形公式

将区间$[a,b]$划分n等分，分点$x_k=a+kh$，$h=\frac{b-a}{n}$，$k=0,1,...,n$，在每个子区间$[x_k,x_k+1]]$上采用梯形公式，则得

$$T_n=\frac{h}{2}\sum_{k=0}^{n-1}[f(x_k)+f(x_{k+1})]$$

复合辛普森公式

将区间$[a,b]$划分n等分，分点$x_k=a+kh$，$h=\frac{b-a}{n}$，$k=0,1,...,n$，在每个子区间$[x_k,x_k+1]]$上采用辛普森公式，并记$x_{k+1/2}=x_k+\frac{1}{2}h$，则得

$$S_n=\frac{h}{6}\sum_{k=0}^{n-1}[f(x_k)+4f(x_{k+1/2}+f(x_{k+1}))]$$

龙贝格序列

设以$T_0^{(k)}$表示二分k次后求得的梯形值，且以$T_m^{(k)}$表示序列的第m次加速值，则依递推公式可得

$$T_m^{(k)}=\frac{4^m}{4^m-1}T_{m-1}^{(k+1)}-\frac{1}{4^m-1}T_{m-1}^{(k)}$$

- **三点公式**：已知三个等距节点$x_0,x_1=x_0+h,x_2=x_0+2h$的函数值为$f(x_0),f(x_1),f(x_2)$

$$P_2(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}f(x_0)+\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f(x_1)+\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}f(x_2)$$

令等距节点$x=x_0+th\ (t=0,1,2)$

$$P_2(x)=P_2(x_0+th)=\frac{1}{2}(t-1)(t-2)f(x_0)-t(t-2)f(x_1)+\frac{1}{2}t(t-1)f(x_2)$$

两端对t求导，那么在**插值节点**x_0,x_1,x_2的导数值为：

$$P_2'(x_0+th)=\frac{1}{2h}[(2t-3)f(x_0)-(4t-4)f(x_1)+(2t-1)f(x_2)]$$

$$\Longrightarrow\begin{cases}f'(x_0)\approx P_2'(x_0)=\frac{1}{2h}[-3f(x_0)+4f(x_1)-f(x_2)], & t=0\\ f'(x_1)\approx P_2'(x_1)=\frac{1}{2h}[-f(x_0)+f(x_2)], & t=1\\ f'(x_2)\approx P_2'(x_2)=\frac{1}{2h}[f(x_0)-4f(x_1)+3f(x_2)], & t=2\end{cases}$$

其中$f'(x_1)=\frac{1}{2h}[-f(x_0)+f(x_2)]$由于少用一个函数值而应用广泛，也就是“**中点公式**”。

更高阶的数值微分公式

令等距的插值节点 $x = x_0 + th\ (t = 0, 1, 2)$，在**插值节点** x_0, x_1, x_2 的二阶导数值为

$$P_2''(x_0 + th) = \frac{1}{h^2}[f(x_0) - 2f(x_1) + f(x_2)]$$

可得到：

$$P_2''(x_1) = \frac{1}{h^2}[f(x_1 - h) - 2f(x_1) + f(x_1 + h)], \quad t = 1$$

四、实验内容

实验软件和语言：利用 Python 语言和集成 Jupyter Notebook 编辑器

实验步骤与方案（自制）：

实验一.

1、数值积分（①②同类型 解决完全一致）.

(1) 复合梯形.

def a, b, h, n=100000(或为不必要的前期接收), f

k 从 1~n $a_i = a + k \cdot h$

No ← $T = \frac{h}{2}\left(f(a) + f(b) + 2\sum_{k=0}^{n-1} x_{k+\frac{1}{2}}\right)$ ← sum

if $eps < 10^{-8}$

↓ Yes

return T n

(2) 复合辛普森.

def a, b, h n=100(,,,), f

k 从 1~n $a_i = a + k \cdot h$

$b_i = a_i - \frac{1}{2}h$ → sum → Sum

N2 ← $T = \frac{h}{6}(f(a) + f(b) + 2 \cdot sum1 + 4 sum2)$

if $eps < 10^{-8}$

↓ yes

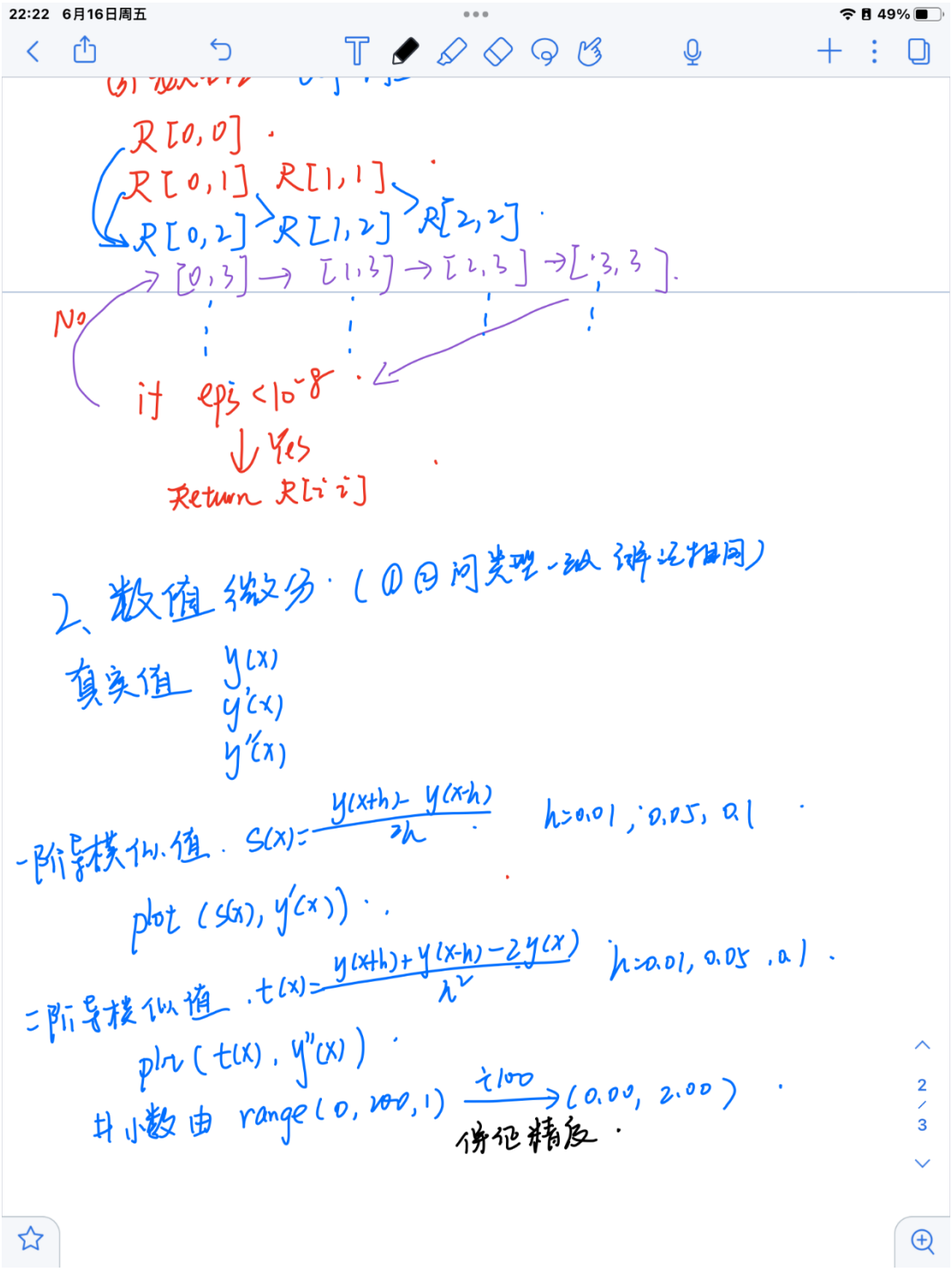
return T, n

(3) 龙贝格 def 同上

R[0,0]

R[0,1] R[1,1]

R[0,2] → R[1,2] → R[2,2]



具体实验内容详见附录

五、实验结果

题 1.1	复合梯形公式	复合辛普森公式	龙贝格积分
迭代值	54.59815003813867	54.598150038143146	54.59815003320592

真实值	54.59815003314423	54.59815003314423	54.59815003314423
迭代次数	163000	248	9
收敛性	收敛	收敛	收敛

题 1.2	复合梯形公式	复合辛普森公式	龙贝格积分
迭代值	1.791759474165676	1.791759474175664	1.7917594693429706
真实值	1.791759469228055	1.791759469228055	1.791759469228055
迭代次数	15001	95	9
收敛性	收敛	收敛	收敛

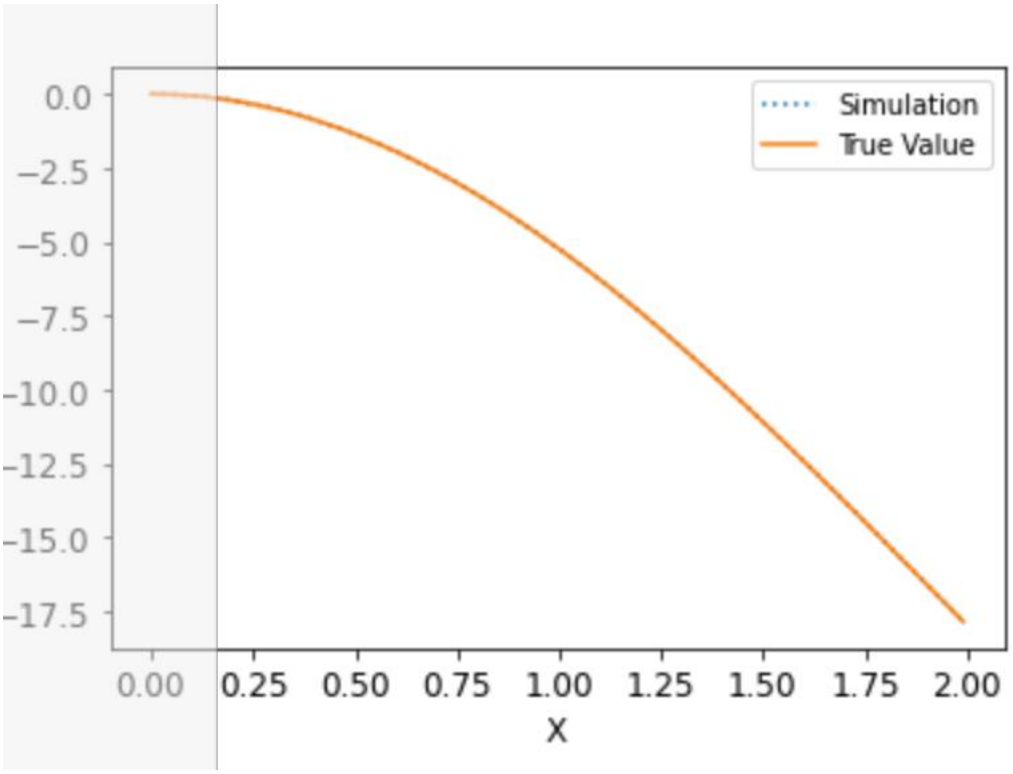


图 1：题 2-1 一阶导模拟

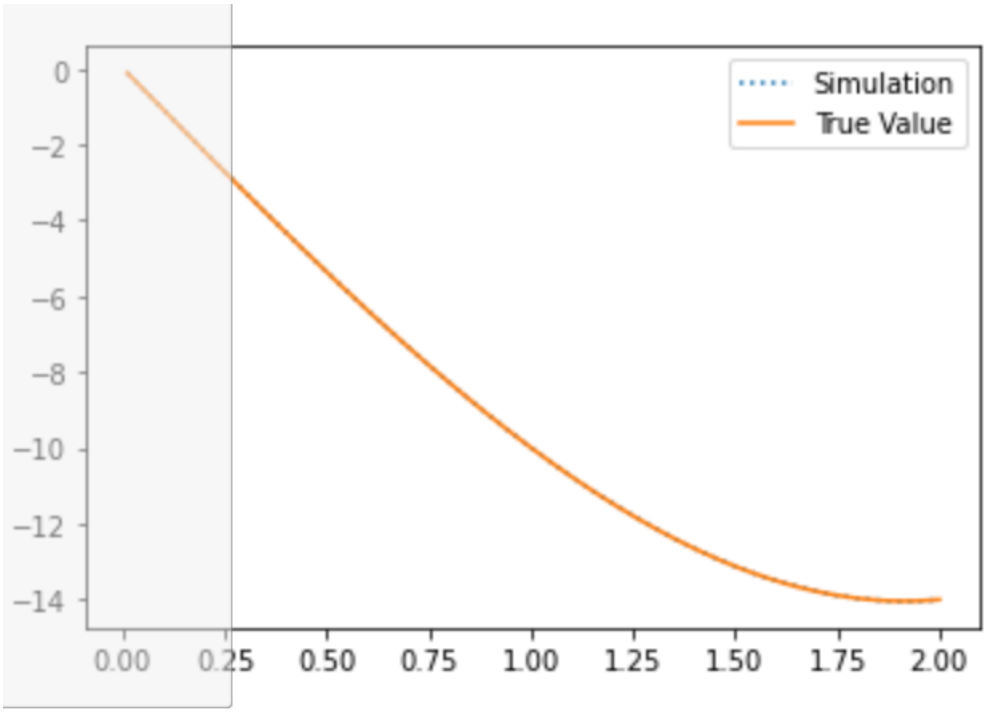


图 2：题 2-1 二阶导模拟

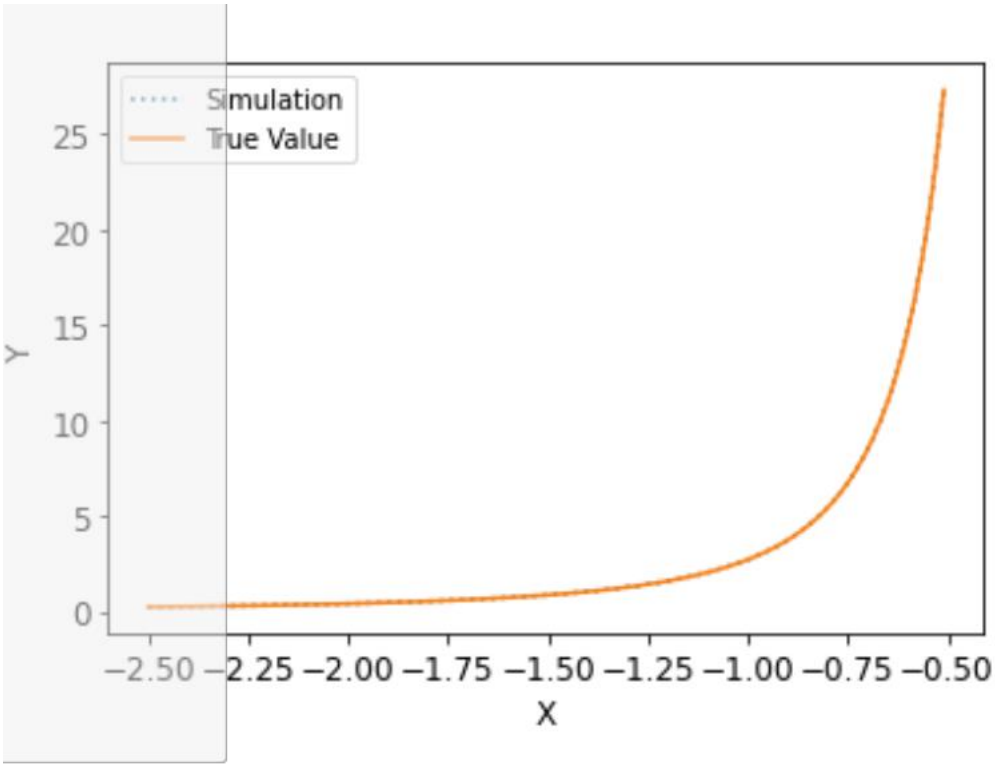


图 3：题 2-2 一阶导模拟

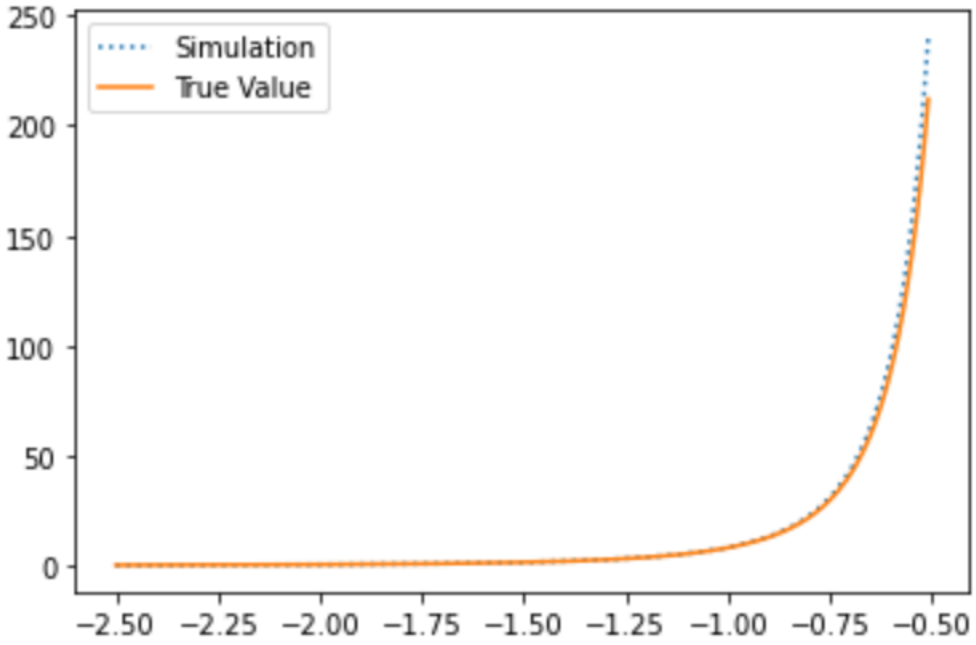


图 4：题 2-2 二阶导模拟

六、实验结果分析

题目 1：复合梯形、复合辛普森公式以及龙贝格积分加速算法都是计算积分近似值的方法，然而从精度和效率上来看：龙贝格积分 》复合辛普森 》复合梯形，在不考虑计算量及其他客观因素时，应采用龙贝格积分求积分，以获得更高的精度。

题目 2：利用二点估计法是，即经过(x-h,f(x-h))和(x+h,f(x+h))二点的割线数值微分对于很小的 h 而言这个值比单边近似还要准确。特别的是公式虽计算 x 点的斜率，但不会用到函数在 x 点的数值。从模拟结果也可以看出这个近似是非常完美的，几乎没有误差；对于二阶导数，采用二阶三点公式，可以看到在解多项式导数时误差更小，但是在 e**(1/x)时，由于 x->0 的 f—>+∞导致了在函数右端点处有些许误差，这是在临界值处高阶导数近似无法避免的

问题（当然也可以通过变换函数避免数值溢出），这也显示了用差值多项式模拟导数并不是非常完美的，特别是在求高阶导和临界值时。

七、附录：

##由于使用 notebook，笔者提供一个 github 地址，里面存放有项目相关的两个 notebook 供查阅，另附件中也有，烦请查看，谢谢。

rqkgHH/Numerical_analysis (github.com)

```
#!/usr/bin/env python
# coding: utf-8

# ### 1. 实验一

# ![%E5%9B%BE%E7%89%87.png](attachment:%E5%9B%BE%E7%89%87.png)

# ## 第一问

# In[247]:


import math
a = 1
b = 2
s = 0
def fun1(x):
    return 2/3*(x**3)*(math.e**(x**2))

def fhtx(): ##复合梯形公式
    n = 100000
    while True:
        n += 1000
        h = (b-a) / n
        sum1 = 0
        for k in range(1,n):
```

```
            ai = a + k * h
            sum1 = sum1 + fun1(ai)

        T = h / 2 * (fun1(a) + fun1(b) + 2 * sum1) #近似值计算

        if abs(T - math.e**4) <= 0.5*10**(-8):
            break ##记录循环次数
        else:
            print(T - math.e**4)
            continue
    return T,n

def fhsim(): ##复合辛普森公式
    n = 100
    while True:
        n += 1
        h = (b-a) / n
        sum1 = 0
        sum2 = 0
        for k in range(1,n+1):
            ai = a + k * h
            sum1 = sum1 + fun1(ai)
            bi = ai - 1/2* h
            sum2 = sum2 + fun1(bi)

        T = h / 6 * (fun1(a) + fun1(b) + 2 * sum1 + 4 * sum2 - 2 *
fun1(b)) #近似值计算

        if abs(T - math.e**4) <= 0.5*10**(-8):
            break ##记录循环次数
        else:
            print(T - math.e**4)
            continue
    return T,n


# In[251]:


def romberg():
    n = 0
    a = 1
    b = 2
    R = [[0] * 10000] * 10000
```

```
    # R = [[0]*(n+1) for_in range(n+1)]
    h = b-a
    R[0][0] = h/2+(f(a)+f(b))
    while True:
        n = n+1
        for i in range(1,n+1):
            h = h/2
            sum3 = 0
            for k in range(1,2**i,2):
                sum3 += f(a+k*h)
            R[i][0] = R[i-1][0]/2 + sum3*h
            for j in range(1,i+1):
                R[i][j] = (4**j*R[i][j-1]-R[i-1][j-1])/(4**j-1)
        if abs(R[n][n]) <= 0.5*10**(-8):
            break
        else:
            print(R[n][n])
            continue

    return R[n][n]+math.e**4,i


# In[252]:


fhtx()


# In[253]:


fhsim()


# In[254]:


romberg()


# In[255]:


math.e**4
```

第二问

In[256]:

```
import math

a = 2
b = 3
s = 0


def fun2(x):
    return 2*x/(x**2-3)

def fhtx2(): ##复合梯形公式
    n = 1
    while True:
        n += 100
        h = (b-a) / n
        sum1 = 0
        for k in range(1,n):
            ai = a + k * h
            sum1 = sum1 + fun2(ai)

        T = h / 2 * (fun2(a) + fun2(b) + 2 * sum1) #近似值计算

        if T - math.log(6) <= 0.5*10**(-8):
            break ##记录循环次数
        else:
            print(T - math.log(6))
            continue
    return T,n

def fhsim2(): ##复合辛普森公式
    n = 1
    while True:
        n += 1
        h = (b-a) / n
        sum1 = 0
        sum2 = 0
```

```
        for k in range(1,n+1):
            ai = a + k * h
            sum1 = sum1 + fun2(ai)
            bi = ai - 1/2* h
            sum2 = sum2 + fun2(bi)

        T = h / 6 * (fun2(a) + fun2(b) + 2 * sum1 + 4 * sum2 - 2 *
fun2(b)) #近似值计算

        if T - math.log(6) <= 0.5*10**(-8):
            break ##记录循环次数
        else:
            print(T - math.log(6))
            continue
    return T,n


def romberg2():
    n = 0
    a = 2
    b = 3
    R = [[0] * 10000] * 10000
    # R = [[0]*(n+1) for_in range(n+1)]
    h = b-a
    R[0][0] = h/2+(fun2(a)+fun2(b))
    while True:
        n = n+1
        for i in range(1,n+1):
            h = h/2
            sum3 = 0
            for k in range(1,2**i,2):
                sum3 += fun2(a+k*h)
            R[i][0] = R[i-1][0]/2 + sum3*h
            for j in range(1,i+1):
                R[i][j] = (4**j*R[i][j-1]-R[i-1][j-1])/(4**j-1)
        if abs(R[n][n]) <= 0.5*10**(-8):
            break
        else:
            print(R[n][n])
            continue

    return R[n][n]+math.log(6),i
```

In[257]:

```
fhtx2()
```

In[258]:

```
fhsim2()
```

In[259]:

```
romberg2()
```

In[260]:

```
math.log(6)
```

![%E5%9B%BE%E7%89%87.png](attachment:%E5%9B%BE%E7%89%87.png)

第一问

In[233]:

```
import matplotlib.pyplot as plt
import math

y11 = lambda x: (x**5/20 - 11/6*x**3)
dy1 = lambda x: (1/4*x**4 - 11/2*(x**2))
ddy1 = lambda x: (x**3-11*x)
def y1(x):
    return (x**5)/20.00 - (11/6)*(x**3)

def y2(x):
    x = math.e**(-1/x)
import numpy as np
# n = 10
```

```
# h = 2/n
# for i in (1,n):

###差分公式

def diff1(y,x,h):
    return (y(x+h)-y(x))/h

def diff2(y,x,h):
    return (y(x)-y(x-h))/h

def diff3(y,x,h):
    return (y(x+h)-y(x-h))/(2*h)

def diff_c(y,x,h): #等距节点的二阶差分
    return (y(x-h)+y(x+h)-2*y(x+0))/(h**2)

v11 = []
d1 = []
x1 = []
# h = 0.01
for x100 in range(0,200,1):
    x = x100/100
    x1.append(x)
    v11.append((y1(x+0.01)-y1(x-0.01))/0.02)
    d1.append(dy1(x))
#plt.plot(x1,v1)
#plt.plot(v2)
plt.xlabel('X',fontsize = 12)
plt.ylabel('Y',fontsize = 12)
plt.tick_params(axis='both',labelsize = 12)
x = np.linspace(0,2,200)
plt.plot(x1,v11,':',label="Simulation") # 模拟值
plt.plot(x1,d1,label='True Value') #真实值
plt.legend(loc='best')


# In[235]:


ddy1 = lambda x: x**3-11*x


def y(x):
```

```
    return (x**5)/20.00 - (11/6)*(x**3)

v12 = []
d2 = []
x2 = []
x_ = np.linspace(0,2,200)
for x10 in range(1,201,1):
    x11 = x10/100
    x2.append(x11)
    v12.append((y(x11-0.01)+y(x11+0.01)-2*y(x11))/(0.01**2))
    d2.append(ddy1(x11))

plt.plot(x2,v12,':',label="Simulation")
plt.plot(x2,d2,label='True Value')
plt.legend(loc='best')


# ## 第二问

# In[238]:


v21 = []
d2 = []
x1 = []
def y2(x):
    return math.e**(-1/x)
dy2 = lambda x: math.e**(-1/x)*(1/x**2)
for x100 in range(-250,-50,1):
    x = x100/100
    x1.append(x)
    v21.append((y2(x+0.01)-y2(x-0.01))/0.02)
    d2.append(dy2(x))
#plt.plot(x1,v1)
#plt.plot(v2)
plt.xlabel('X',fontsize = 12)
plt.ylabel('Y',fontsize = 12)
plt.tick_params(axis='both',labelsize = 12)
x = np.linspace(-2.5,-0.5,200)
plt.plot(x1,v21,':',label="Simulation") # 模拟值
plt.plot(x1,d2,label='True Value') #真实值
plt.legend(loc='best')
```

```
# In[262]:


ddy2 = lambda x: math.e**(-1/x)*(-2/x**3) + math.e**(-
1/x)*(1/x**2)*(1/x**2) #二阶导

def y2(x):
    return math.e**(-1/x)

v22 = []
d22 = []
x2 = []
x_ = np.linspace(-2/5,-0.5,200)
h = 0.1
for x20 in range(-250,-50,1):
    x = x20/100
    x2.append(x)
    v22.append((y2(x-h)+y2(x+h)-2*y2(x))/(h**2))
    d22.append(ddy2(x))

plt.plot(x2,v22,':',label="Simulation")
plt.plot(x2,d22,label='True Value')
plt.legend(loc='best')
```