实验报告

专业 ____学号____姓名 ____________

一、实验目的

通过编程，进一步理解数值积分中的复合梯形公式、复合辛普森公式、龙贝格公式；数值微分中的插值型求导公式。

二、实验题目

数值微积分实验

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

1. 1 复合梯形公式

将积分区间[a, b]分为 n 等分，分点为$x_k = a + kh$ (k=1, …, n)，其中步长$h = \frac{b-a}{n}$。在每个小区间[x_k，x_{k+1}]上用梯形公式，则

$$
\begin{aligned}
I=\int_a^b f(x)dx &= \sum_{k=0}^{n-1}\int_{x_k}^{x_{k+1}} f(x)dx \\
&\approx \sum_{k=0}^{n-1}\left\{\frac{x_{k+1}-x_k}{2}[f(x_k)+f(x_{k+1})]\right\} \\
&\approx \frac{h}{2}\sum_{k=0}^{n-1}[f(x_k)+f(x_{k+1})]
\end{aligned}
$$

记

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

1.2 复合辛普森公式

将积分区间[a, b]分为 n 等分，分点为 $x_k=a+kh$ (k=1, …, n)， 其中步长 $h=\frac{b-a}{n}$。记每个小区间[x_k， x_{k+1}]的中点为 $x_{k+\frac{1}{2}}$， 在每个小区间[x_k， x_{k+1}]上用辛普森公式，则

$$
\begin{aligned}
I=\int_a^b f(x)dx &= \sum_{k=0}^{n-1}\int_{x_k}^{x_{k+1}} f(x)dx \\
&\approx \sum_{k=0}^{n-1}\left\{\frac{x_{k+1}-x_k}{6}[f(x_k)+f(x_{k+1})+4f(x_{k+\frac{1}{2}})]\right\} \\
&\approx \frac{h}{6}[f(a)+f(b)+2\sum_{k=1}^{n-1}f(x_k)+4\sum_{k=0}^{n-1}f(x_{k+\frac{1}{2}})]
\end{aligned}
$$

记

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

1.3 龙贝格积分公式

梯形求积公式递推：

原区间[a, b]n 等分共 n+1 个分点 $\xrightarrow{\text{二分一次}}$ 分点增至 2n+1 个

单个小区间上 $S=\frac{h}{2}(f(x_k)+f(x_{k+1}))\xrightarrow{\text{二分一次}}S=\frac{h}{4}(f(x_k)+f(x_{k+1})+2f(x_{k+\frac{1}{2}}))$

整个区间上 $T_n=\frac{h}{2}\sum_{k=0}^{n-1}[f(x_k)+f(x_{k+1})]\xrightarrow{\text{二分一次}}T_{2n}=\frac{1}{2}T_n+\frac{h}{2}\sum_{k=0}^{n-1}f(x_{k+\frac{1}{2}})$

优点：通过二分可以提升积分精度。

缺点：收敛过慢。

理查森外推加速：

记I为原始积分，则有$I-T_n=-\frac{b-a}{12}f^{"}(\eta),\eta\in[a,b],h=\frac{b-a}{n}$。

定义$T_n\triangleq T(h)\Rightarrow T_{2n}=T(\frac{h}{2})$

利用泰勒展开式可得$T(h)=I+\alpha_1h^2+\alpha_2h^4\ldots+\alpha_lh^{2l}\ldots$ ①

此时$T(h/2)=I+\frac{\alpha_1}{4}h^2+\frac{\alpha_2}{16}h^4\ldots+\frac{\alpha_l}{2^{2l}}h^{2l}\ldots$②

可得$S(h)=\frac{4T(h/2)-T(h)}{3}\triangleq I+\beta_1h^4+\beta_2h^6\ldots$误差阶变为$o(h^4)$，相较于$T(h)$的$o(h^2)$，精度提高。

按此方法依次递推，可见精度会越来越高。该方法即理查森外推加速算法。

龙贝格算法：

定义$T_m^{(k)}$：其中 k 为二分次数，m 为加速次数。

借助理查森外推公式：$T_m^{(k)}=\frac{4^m}{4^m-1}T_{m-1}^{(k+1)}-\frac{1}{4^m-1}T_{m-1}^{(k)}$，及复合梯形递推公式$T_{2n}=\frac{1}{2}T_n+\frac{h}{2}\sum_{k=0}^{n-1}f(x_{k+\frac{1}{2}})$，可求得更为精确的积分值。

算法：

1) 取$k=0,h=b-a$，求$T_0^{(0)}=\frac{h}{2}(f(a)+f(b))$，令$1\to k$ (k 记区间[a, b]的二分次数)

2) 求梯形值$T_0(\frac{b-a}{2^k})$，即按递推公式计算$T_0^{(k)}$。

3) 求加速值$T_j^{(k-j)}(j=1,\ldots,k)$

4) 若$\left|T_k^{(0)}-T_{k-1}^{(0)}\right|<\varepsilon$ (预先给定的精度)，则终止计算，并取$T_k^{(0)}\approx I$；

否则令 $k+1 \to k$ 转 2)继续计算。

2 一阶导数和二阶导数的数值方法

向前插商：$f(a)=\dfrac{f(a+h)-f(a)}{h}$

向后插商：$f(a)=\dfrac{f(a)-f(a\text{-h})}{h}$

中心插商：$f(a)=\dfrac{f(a+h)\text{-}f(a\text{-h})}{2h}$

二阶导数：$f(a)=\dfrac{f(a+h)\text{-}2f(a)+f(a\text{-h})}{h^2}$

四、实验内容

1.1 复合梯形公式

变量定义：

T：积分数组，T[i]即为 i 等分积分区间所得积分值

a：积分区间左端点

b：积分区间右端点

eps：绝对误差限

实施步骤：

1)初始化变量：a，b，T，eps

2)i=2

3)计算 T[i],T[i+1]

4)如果|T[i+1]-T[i]|< eps，输出积分结果 T[i+1]以及等分数 i

5)否则 i=i+1 并跳转至步骤 2)

1.2 复合辛普森公式

变量定义：

S：积分数组，S[i]即为 i 等分积分区间所得积分值

a：积分区间左端点

b：积分区间右端点

eps：绝对误差限

实施步骤：

1)初始化变量：a，b，S，eps

2)i=2

3)计算 S[i],S[i+1]

4)如果|S[i+1]-S[i]|< eps，输出积分结果 S[i+1]以及等分数 i

5)否则 i=i+1 并跳转至步骤 2)

1.3 龙贝格积分公式

变量定义：

T：二维数组，用于存放 T 表中的数据。T[i][i]表示第 i 行 i 列的数据。

w：二维数组 T 的维度

a：积分区间左端点

b：积分区间右端点

eps：绝对误差限

实施步骤：

1)初始化变量：a，b，T，w，eps

2)使用梯形公式计算 T[0][0]

3)for i = 1:w-1

运用梯形递推公式计算 T[i][0]

for j = 1:i

运用理查森外推公式根据 p[i][j-1]、p[i-1][j-1]计算 T[i][j]

if |T[i][i]-T[i-1][i-1]|< eps,跳出循环，输出 T[i][i]以及加速次数 i

2 一阶导数和二阶导数的数值方法

变量定义：

a：积分区间左端点

b：积分区间右端点

n：区间的等分数

fun：原函数表达式

实施步骤：

1)初始化变量：a，b，n，fun

2)计算原函数一阶导数、二阶导数的显式表达式 dfun、ddfun

3)将区间[a,b]n 等分，使用不同方式求解一阶导、二阶导的数值解 dy1、dy2、dy3、ddy1；分别计算绝对误差 err1、err2、err3、errd2

4)绘制 fun,dfun,ddfun 在 n 等分区间[a,b]上的散点图并分析。

硬件：Personal Computer with Intel CPU

软件：Microsoft Windows, python3.6, Microsoft Office

五、实验结果

1.1） $e^4 = \int_1^2 \frac{2}{3} x^3 e^{x^2} dx$

```
#复合梯形公式
a = 1
b = 2
cankao = np.exp(4)#参考值

initial = tixing(a, b, 1, fun1)
ebs = initial - 0
i = 1
while (ebs >= 0.000000005):
    i = i+1
    ebs = np.abs(tixing(a, b, i, fun1) - tixing(a, b, i+1, fun1))
print("复合梯形公式等分数：", i+1)
print("复合梯形公式所得积分值：",tixing(1, 2, i+1, fun1))
print("复合梯形公式迭代的绝对误差：",np.abs(tixing(a, b, i, fun1) -
tixing(a, b, i+1, fun1)))
print("复合梯形公式与参考值的绝对误差：",np.abs(cankao-tixing(a, b, i+1,
fun1)))
```

输出结果：

等分数： 3759
所得复合梯形积分值： 54.59815942495601
迭代的绝对误差： 4.998938152311894e-09
与参考值的绝对误差： 9.391811772729852e-06

```
#复合辛普森公式
a = 1
b = 2
cankao = np.exp(4)#参考值

initial = xinpusen(a, b, 2, fun1)
ebs = initial - 0
i = 1
while (ebs >= 0.000000005):
    i = i+1
    ebs = np.abs(xinpusen(a, b, i, fun1) - xinpusen(a, b, i+1, fun1))
print("复合辛普森公式等分数：",i+1)
print("复合辛普森公式所得积分值：",xinpusen(1, 2, i+1, fun1))
print("复合辛普森公式迭代的绝对误差：",np.abs(xinpusen(a, b, i, fun1) -
xinpusen(a, b, i+1, fun1)))
print("复合辛普森公式与参考值的绝对误差：",np.abs(cankao - xinpusen(a, b,
i+1, fun1)))
```

输出结果：

复合辛普森公式等分数： 110
复合辛普森公式所得积分值： 54.5981501622893

复合辛普森公式迭代的绝对误差： 4.804654452073009 5e-09
复合辛普森公式与参考值的绝对误差： 1.291450644202996e-07

```
#龙贝格积分
w = 10#T 表维度
a = 1
b = 2
cankao = np.exp(4)#参考值
eps = 0.000000005
x = np.array([a,b]).reshape([1,2])#初始化分点
h = b - a

T = T_initial(w, a, b, fun1)#初始化 T 表
T, x, h = one_N2O(T, x, 1, h, fun1)#先加一次速
n = 1#累计加速次数
while (np.abs(T[n][n] - T[n - 1][n - 1]) >= eps):
    n = n + 1
    T, x, h = one_N2O(T, x, n, h, fun1)#加一次速
print("加速次数：",n)
print("龙贝格算法所得积分值：",T[n][n])
print("T 表：")
print(T)
print("龙贝格算法迭代的绝对误差：",np.abs(T[n][n] - T[n - 1][n - 1]))
print("龙贝格算法与参考值的绝对误差：",np.abs(T[n][n] - cankao))
```

输出结果：

```
加速次数： 7
龙贝格算法所得积分值： 54.59815003314422
T 表：
[[146.5011607    0.           0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [ 83.92428316  63.06532399   0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [ 62.61319631  55.50950069  55.00577913   0.           0.
    0.           0.           0.           0.           0.        ]
 [ 56.65349937  54.66693373  54.61076259  54.60449249   0.
    0.           0.           0.           0.           0.        ]
 [ 55.11538708  54.60268299  54.5983996   54.59820337  54.5981787
    0.           0.           0.           0.           0.        ]
 [ 54.72767471  54.59843725  54.5981542   54.59815031  54.5981501
   54.59815007   0.           0.           0.           0.        ]
 [ 54.63054471  54.59816805  54.5981501   54.59815003  54.59815003
   54.59815003  54.59815003   0.           0.           0.        ]
 [ 54.60624955  54.59815116  54.59815003  54.59815003  54.59815003
   54.59815003  54.59815003  54.59815003   0.           0.        ]
```

```
 [  0.           0.           0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.           0.           0.           0.        ]]
```

龙贝格算法迭代的绝对误差： 1.269029326067539e-11

龙贝格算法与参考值的绝对误差： 1.4210854715202004e-14

1.2) $\ln 6=\int_2^3 \frac{2x}{x^2-3}dx$

```
#复合梯形公式
a = 2
b = 3
cankao = np.log(6)

initial = tixing(a, b, 1, fun2)
ebs = initial - 0
i = 1
while (ebs >= 0.000000005):
    i = i+1
    ebs = np.abs(tixing(a, b, i, fun2) - tixing(a, b, i+1, fun2))
print("复合梯形公式等分数：", i+1)
print("复合梯形公式所得积分值：",tixing(a, b, i+1, fun2))
print("复合梯形公式迭代的绝对误差：",np.abs(tixing(a, b, i, fun2) - tixing(a, b, i+1, fun2)))
print("复合梯形公式与参考值的绝对误差：",np.abs(cankao-tixing(a, b, i+1, fun2)))
```

输出结果：

复合梯形公式等分数： 764

复合梯形公式所得积分值： 1.7917613728017425

复合梯形公式迭代的绝对误差： 4.992964930394805e-09

复合梯形公式与参考值的绝对误差： 1.9035736875672171e-06

```
#复合辛普森公式
a = 2
b = 3
cankao = np.log(6)

initial = xinpusen(a, b, 2, fun2)
ebs = initial - 0
i = 1
while (ebs >= 0.000000005):
    i = i+1
    ebs = np.abs(xinpusen(a, b, i, fun2) - xinpusen(a, b, i+1, fun2))
print("复合梯形公式等分数：", i+1)
print("复合梯形公式所得积分值：",xinpusen(a, b, i+1, fun2))
```

```
print("复合梯形公式迭代的绝对误差：",np.abs(xinpusen(a, b, i, fun2) - xinpusen(a, b, i+1, fun2)))
print("复合梯形公式与参考值的绝对误差：",np.abs(cankao-xinpusen(a, b, i+1, fun2)))
```

输出结果：

复合梯形公式等分数： 51
复合梯形公式所得积分值： 1.7917595286610115
复合梯形公式迭代的绝对误差： 4.8909591932044805e-09
复合梯形公式与参考值的绝对误差： 5.943295655619352e-08

```
#龙贝格积分
w = 10#T表维度
a = 2
b = 3
cankao = np.log(6)#参考值
eps = 0.000000005
x = np.array([a,b]).reshape([1,2])#初始化分点
h = b - a

T = T_initial(w, a, b, fun2)#初始化T表
T, x, h = one_N2O(T, x, 1, h, fun2)#先加一次速
n = 1#累计加速次数
while (np.abs(T[n][n] - T[n - 1][n - 1]) >= eps):
    n = n + 1
    T, x, h = one_N2O(T, x, n, h, fun2)#加一次速
print("加速次数：",n)
print("龙贝格算法所得积分值：",T[n][n])
print("T表：")
print(T)
print("龙贝格算法迭代的绝对误差：",np.abs(T[n][n] - T[n - 1][n - 1]))
print("龙贝格算法与参考值的绝对误差：",np.abs(T[n][n] - cankao))
```

输出结果：

加速次数： 7
龙贝格算法所得积分值： 1.7917594692290668
T表：

```
[[2.5        0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [2.01923077 1.85897436 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [1.85643979 1.80217613 1.79838959 0.         0.         0.
  0.         0.         0.         0.        ]
 [1.80876065 1.7928676  1.79224703 1.79214953 0.         0.
  0.         0.         0.         0.        ]
```

```
 [1.79607573 1.79184743 1.79177942 1.791772   1.79177052 0.
  0.         0.         0.         0.        ]
 [1.79284301 1.79176544 1.79175997 1.79175966 1.79175961 1.7917596
  0.         0.         0.         0.        ]
 [1.79203064 1.79175985 1.79175948 1.79175947 1.79175947 1.7917594
7
  1.79175947 0.         0.         0.        ]
 [1.79182728 1.79175949 1.79175947 1.79175947 1.79175947 1.7917594
7
  1.79175947 1.79175947 0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]]
龙贝格算法迭代的绝对误差： 6.066045443731127e-10
龙贝格算法与参考值的绝对误差： 1.0118572646433677e-12
```

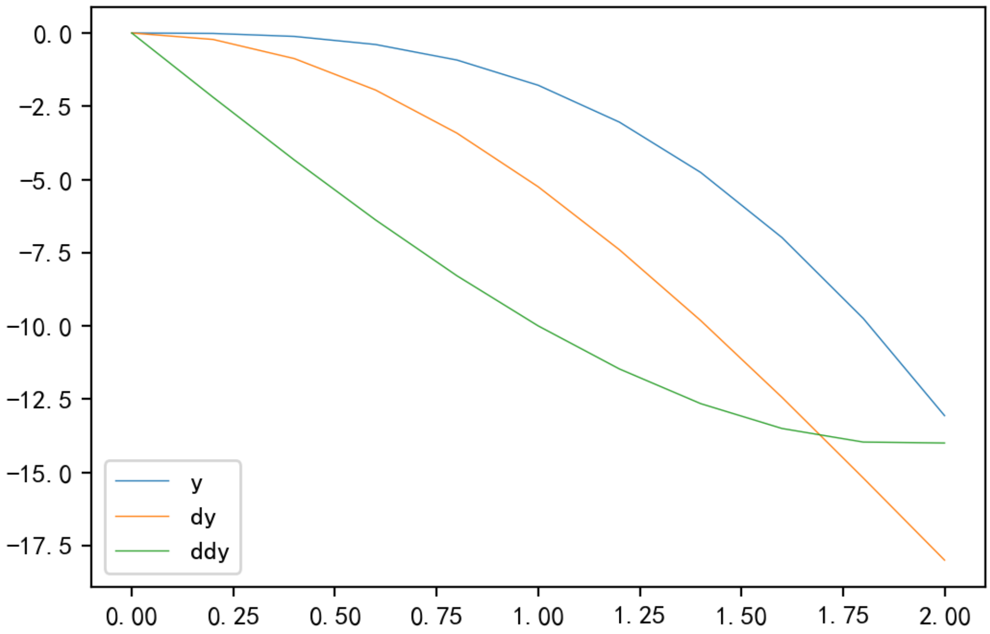
2. 1） $y=\frac{1}{20}x^5-\frac{11}{6}x^3, x\in[0,2]$

```
#函数 1
a = 0
b = 2
n = 10

interval, h= slice_interval(a, b, n)

#计算函数的实际值
fun_list = []
for i in range(len(interval)):
    fun_list.append(fun1(interval[i]))

#计算一阶导数的实际值
dfun_list = []
for i in range(len(interval)):
    dfun_list.append(dfun1(interval[i]))

#计算二阶导数的实际值
ddfun_list = []
for i in range(len(interval)):
    ddfun_list.append(ddfun1(interval[i]))

dy1_list = dy1(interval, fun1, h)#向前差商法计算导数
dy2_list = dy2(interval, fun1, h)#向后差商法计算导数
dy3_list = dy3(interval, fun1, h)#中心差商法计算导数
ddy_list = ddy(interval, fun1, h)#计算二阶导数
```

```
print(n,"等分区间后，向前差商法所得导数与实际一阶导数的误差值（忽略右端点）：")
print(np.abs(np.array(dy1_list) - np.array(dfun_list)))
print(n,"等分区间后，向后差商法所得导数与实际一阶导数的误差值（忽略左端点）：")
print(np.abs(np.array(dy2_list) - np.array(dfun_list)))
print(n,"等分区间后，中心差商法所得导数与实际一阶导数的误差值(忽略左、右端点)：")
print(np.abs(np.array(dy3_list) - np.array(dfun_list)))
print(n,"等分区间后，所得二阶导数与实际二阶导数的误差值（忽略左、右端点）：")
print(np.abs(np.array(ddy_list) - np.array(ddfun_list)))

#做原函数、一阶导数、二阶导数图像
print("原函数、一阶导数、二阶导数图像为：")
plt.rcParams['axes.unicode_minus']=False  # 用来正常显示负号
plt.figure(dpi = 200)
y, = plt.plot(interval,fun_list,lw=0.5,label = 'y')
dy, = plt.plot(interval,dfun_list,lw=0.5, label = 'dy')
ddy, = plt.plot(interval,ddfun_list,lw=0.5, label = 'ddy')
plt.legend(handles=[y, dy, ddy], labels=['y', 'dy','ddy'],
    loc='lower left')
plt.show()
```

输出结果：

```
10 等分区间后，向前差商法所得导数与实际一阶导数的误差值（忽略右端点）：
[ 0.07325333  0.29125333  0.50285333  0.70325333  0.88765333  1.05125333
  1.18925333  1.29685333  1.36925333  1.40165333 18.        ]
10 等分区间后，向后差商法所得导数与实际一阶导数的误差值（忽略左端点）：
[0.         0.14634667 0.36274667 0.57114667 0.76674667 0.94474667
 1.10034667 1.22874667 1.32514667 1.38474667 1.40274667]
10 等分区间后，中心差商法所得导数与实际一阶导数的误差值（忽略左、右端点）：
[0.00000000e+00 7.24533333e-02 7.00533333e-02 6.60533333e-02
 6.04533333e-02 5.32533333e-02 4.44533333e-02 3.40533333e-02
 2.20533333e-02 8.45333333e-03 1.80000000e+01]
10 等分区间后，所得二阶导数与实际二阶导数的误差值（忽略左、右端点）：
[0.0e+00 4.0e-03 8.0e-03 1.2e-02 1.6e-02 2.0e-02 2.4e-02 2.8e-02 3.2e-02
 3.6e-02 1.4e+01]
原函数、一阶导数、二阶导数图像为：
```



2.2) $y = e^{-\frac{1}{x}}, x \in [-2.5, -0.5]$

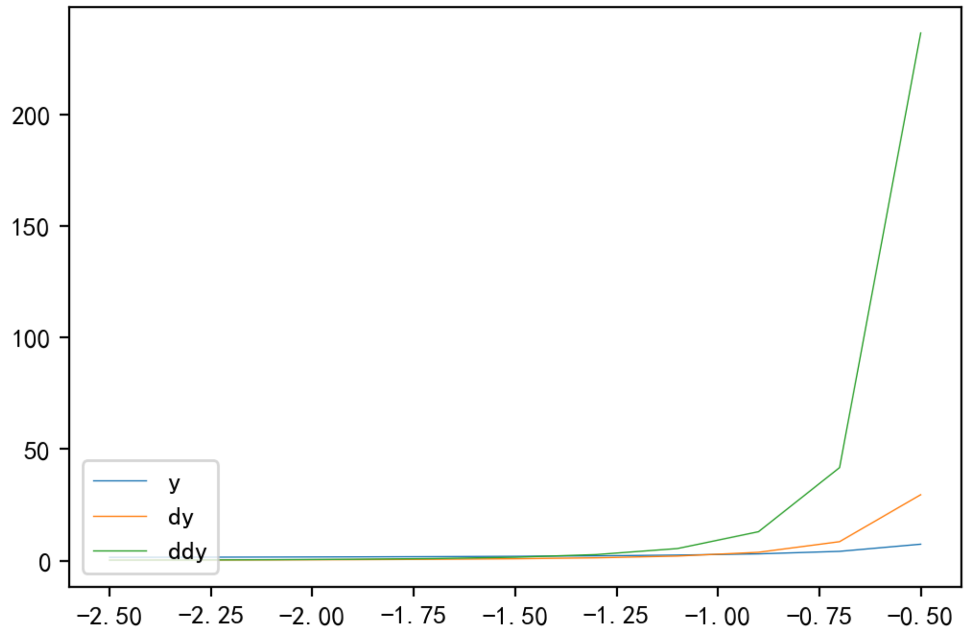
```
#函数 2
a = -2.5
b = -0.5
n = 10

interval, h= slice_interval(a, b, n)

#计算函数的实际值
fun_list = []
for i in range(len(interval)):
    fun_list.append(fun2(interval[i]))

#计算一阶导数的实际值
dfun_list = []
for i in range(len(interval)):
    dfun_list.append(dfun2(interval[i]))

#计算二阶导数的实际值
ddfun_list = []
for i in range(len(interval)):
    ddfun_list.append(ddfun2(interval[i]))

dy1_list = dy1(interval, fun2, h)#向前差商法计算导数
dy2_list = dy2(interval, fun2, h)#向后差商法计算导数
dy3_list = dy3(interval, fun2, h)#中心差商法计算导数
ddy_list = ddy(interval, fun2, h)#计算二阶导数
```

```
print(n,"等分区间后，向前差商法所得导数与实际一阶导数的误差值（忽略右端点）：")
print(np.abs(np.array(dy1_list) - np.array(dfun_list)))
print(n,"等分区间后，向后差商法所得导数与实际一阶导数的误差值（忽略左端点）：")
print(np.abs(np.array(dy2_list) - np.array(dfun_list)))
print(n,"等分区间后，中心差商法所得导数与实际一阶导数的误差值(忽略左、右端点):")
print(np.abs(np.array(dy3_list) - np.array(dfun_list)))
print(n,"等分区间后，所得二阶导数与实际二阶导数的误差值（忽略左、右端点）：")
print(np.abs(np.array(ddy_list) - np.array(ddfun_list)))

#做原函数、一阶导数、二阶导数图像
print("原函数、一阶导数、二阶导数图像为：")
plt.rcParams['axes.unicode_minus']=False  # 用来正常显示负号
plt.figure(dpi = 200)
y, = plt.plot(interval,fun_list,lw=0.5,label = 'y')
dy, = plt.plot(interval,dfun_list,lw=0.5, label = 'dy')
ddy, = plt.plot(interval,ddfun_list,lw=0.5, label = 'ddy')
plt.legend(handles=[y, dy, ddy], labels=['y', 'dy','ddy'],
    loc='lower left')
plt.show()
```

输出结果：

```
10 等分区间后，向前差商法所得导数与实际一阶导数的误差值（忽略右端点）：
[2.53207298e-02 3.45220052e-02 4.87113911e-02 7.17278662e-02
 1.11514781e-01 1.86197891e-01 3.42812230e-01 7.27040006e-01
 1.92472439e+00 7.56582764e+00 2.95562244e+01]
10 等分区间后，向后差商法所得导数与实际一阶导数的误差值（忽略左端点）：
[ 0.23869195  0.02797734  0.03855138  0.0551129   0.08250129  0.131
02794
  0.22512806  0.4314957   0.97195268  2.84077291 13.47461332]
10 等分区间后，中心差商法所得导数与实际一阶导数的误差值（忽略左、右端点）：
[2.38691952e-01 3.27233063e-03 5.08000558e-03 8.30748342e-03
 1.45067458e-02 2.75849767e-02 5.88420859e-02 1.47772151e-01
 4.76385852e-01 2.36252737e+00 2.95562244e+01]
10 等分区间后，所得二阶导数与实际二阶导数的误差值（忽略左、右端点）：
[2.29144274e-01 3.39577351e-03 5.85359799e-03 1.07520374e-02
 2.13903238e-02 4.71787865e-02 1.19495597e-01 3.67770236e-01
 1.51943322e+00 1.03230430e+01 2.36449795e+02]
```

原函数、一阶导数、二阶导数图像为:



六、实验结果分析

1.当需要达到相同的误差范围时，利用复合辛普森公式比利用复合梯形公式需要等分区间的次数 n 更小，故复合辛普森公式的计算效率更高。

2.对于同一公式，所取 n 的大小与给出的精度 eps 之间的关系：当 n 越大，与精度 eps 之间的误差越小；反之，当 n 越小，与精度 eps 之间的误差越大。

3.对于龙贝格求积公式，其巧妙地利用了泰勒展开的原理，借助理查森加速算法，通过适当的线性组合，把复合梯形公式的近似值组合成更高精度的积分近似值，相比复合梯形公式公式与复合辛普森公式，其运算效率是最高的。

4.对于等距节点计算导数值，其步长 h=(b-a)/n 不宜过大，也不宜过小。过大导致函数值之差过大，精度较低；过小导致有效数字严重损失。

附录：

程序清单：

#积分问题相关代码

```
import numpy as np
import pandas as pd

def fun1(x):#被积函数 1，传入的 x 支持数组
    return 2.0 / 3.0 * (x ** 3) * np.exp(x ** 2)
def fun2(x):#被积函数 2，传入的 x 支持数组
    return 2.0 * x / (x ** 2 - 3)

#梯形求积公式,a,b 为积分上下限，n 为等分数,fun 为被积函数
def tixing(a, b, n, fun):
    h = (b - a)/n
    x = np.zeros([1,n+1])#记录各分点，x[0][0]为 a,x[0][n]为 b
    for i in range(n+1):
        x[0][i] = a + i * h
    y = fun(x)#计算各分点的函数值
    if n == 1:
        Tn = h / 2 * (y[0][0] + y[0][n])
    else:
        Tn = h / 2 * (y[0][0] + y[0][n] + 2 * np.sum(y[:,1:n]))#ndarray
取和行列范围为[a,b)左开右闭形式
    return Tn

#辛普森求积公式,a,b 为积分上下限，n 为等分数,fun 为被积函数
def xinpusen(a, b, n, fun):
    h = (b - a)/n
    x = np.zeros([1,n+1])#记录各分点，x[0][0]为 a,x[0][n]为 b
    x1_2 = np.zeros([1,n])#每个小区间的二分点
    for i in range(n+1):
        x[0][i] = a + i * h
    for i in range(n):
        x1_2[0][i] = (x[0][i] + x[0][i+1]) / 2.0
    y = fun(x)#计算各分点的函数值
    y1_2 = fun(x1_2)#计算每个小区间二分点的函数值
    if n == 1:
        Sn = h / 6 * (y[0][0] + y[0][n] + 2 * (y[0][0] + y[0][n]))
    else:
        Sn = h / 6 * (y[0][0] + y[0][n] + 2 * np.sum(y[:,1:n]) + 4 *
np.sum(y1_2))#ndarray 取和行列范围为[a,b)左开右闭形式
    return Sn
```

```
#龙贝格积分
#T 表初始化，填入 T[0][0]
def T_initial(w, a, b, fun):#w 为维度,a、b 为积分上下限，fun 为函数解析式
    T = np.zeros([w,w])
    T[0][0] = (b-a) / 2 * (fun(a) + fun(b))
    return T

#一次加速
def one_N2O(T, x, i, h, fun):#T 为 T 表，x 为分点数组，i 为累计加速次数,h
为间隔长度
    len_x = x.shape[1]#未加速前的分点个数
    x1_2 = np.zeros([1,len_x - 1])#每个小区间的二分点
    for m in range(len_x - 1):
        x1_2[0][m] = (x[0][m] + x[0][m+1]) / 2.0
    T[i][0] = 1 / 2 * T[i-1][0] + h / 2 * np.sum(fun(x1_2))#T 表开新
的一行，并计算首列值
    for j in range (1 , i + 1):
        T[i][j] =  (4 ** j) / (4 ** j - 1) * T[i][j-1] - T[i-1][j-1]
/ (4 ** j - 1)
    total_point = x.shape[1] + x1_2.shape[1]#现在总分点的个数
    temp = []
    for i in range(total_point):
        if i % 2 == 0:#为偶数时
            temp.append(x[0][int(i / 2)])
        else:
            temp.append(x1_2[0][int((i - 1) / 2)])
    x = np.array(temp).reshape([1,total_point])
    return T, x,  h / 2.0
```

#微分问题相关代码

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def fun1(x):#函数 1，传入的 x 支持数组
    return 1 / 20.0 * x ** 5 - 11 / 6.0 * x ** 3
def dfun1(x):#函数 1 的一阶导数
    return 1 / 4.0 * x ** 4 - 11 / 2.0 * x ** 2
def ddfun1(x):#函数 1 的二阶导数
    return x ** 3 - 11 * x
def fun2(x):#函数 2，传入的 x 支持数组
    return np.exp(-1 / x)
def dfun2(x):#函数 2 的一阶导数
```

```
    return 1 / (x ** 2) * np.exp(-1 / x)
def ddfun2(x):#函数 2 的二阶导数
    return (x ** (-4)) * np.exp(-1 / x) - 2 * (x ** (-3)) * np.exp(-1
/ x)

def slice_interval(a, b, n):#将求导区间 n 等分，得到 n+1 个端点
    temp = []#端点
    h = (b - a)/n
    for i in range(n+1):
        temp.append(a + i * h)
    return temp, h

def dy1(slice_list, fun, h):#向前差商法计算导数
    dy_result = []
    for i in range(len(slice_list)):
        if i+1 < len(slice_list):
            dy_result.append((fun(slice_list[i+1]) -
fun(slice_list[i])) / h)
    dy_result.append(0)#向前差商法无法计算后端点的导数，故添加零
    return dy_result

def dy2(slice_list, fun, h):#向后差商法计算导数
    dy_result = []
    dy_result.append(0)#向后差商法无法计算前端点的导数，故添加零
    for i in range(len(slice_list)):
        if i-1 >= 0:
            dy_result.append((fun(slice_list[i]) -
fun(slice_list[i-1])) / h)
    return dy_result

def dy3(slice_list, fun, h):#中心差商法计算导数
    dy_result = []
    dy_result.append(0)#中心差商法无法计算前端点的导数，故添加零
    for i in range(len(slice_list)):
        if i-1 >= 0 and i+1 < len(slice_list):
            dy_result.append((fun(slice_list[i+1]) -
fun(slice_list[i-1])) / h / 2)
    dy_result.append(0)#中心差商法无法计算后端点的导数，故添加零
    return dy_result

def ddy(slice_list, fun, h):#二阶导数
    ddy_result = []
    ddy_result.append(0)#二阶导数无法计算前端点的导数，故添加零
    for i in range(len(slice_list)):
```

```
        if i-1 >= 0 and i+1 < len(slice_list):
            ddy_result.append((fun(slice_list[i+1]) +
fun(slice_list[i-1]) - 2 * fun(slice_list[i])) / h / h)
    ddy_result.append(0)#二阶导数无法计算后端点的导数，故添加零
    return ddy_result
```