计算方法实验（实验 2）

2022 年 4 月 23 日

姓名：	梁鑫嵘
学号：	200110619
院系：	计算机系
专业：	计算机专业
班级：	6 班

数据显示结果已保留 4 位小数。

1　实验题目 2：龙贝格积分法

1.1　问题分析

准确描述并总结出实验题目（摘要），并准确分析原题的目的和意义。

1.1.1　方法概要

利用复化梯形求积公式、复化辛普生求积公式、复化柯特斯求积公式的误差估计式计算积分 $\int_a^b f(x)dx$。

1.1.2　实验目的

用龙贝格积分法求函数 $f(x)$ 从 a 到 b 的积分，即 $\int_a^b f(x)$。

输入： a, b, ε, f

输出： 龙贝格 T 数表

1.2 数学原理

数学原理表达清晰且书写准确。

利用复化梯形求积公式、复化辛普生求积公式、复化柯特斯求积公式的误差估计式计算积分 $\int_a^b f(x)dx$，记 $h=\frac{b-a}{n}, x_k=a+k\times h, k=0,1,\cdots,n$，其计算公式：

$$
\begin{aligned}
T_n &= \frac{1}{2}h\sum_{k=1}^{n}[f(x_{k-1}+f(x_k))] \\
T_{2n} &= \frac{1}{2}T_n + \frac{1}{2}h\sum_{k=1}^{n}f(x_k-\frac{1}{2}h) \\
S_n &= \frac{1}{3}(4T_{2n}-T_n) \\
C_n &= \frac{1}{15}(16S_{2n}-S_n) \\
R_n &= \frac{1}{63}(64C_{2n}-C_n)
\end{aligned}
$$

或者：

$$
\begin{aligned}
T_0(h) &= T(h) \\
T_m(h) &= \frac{T_{m-1}(\frac{h}{2})-(\frac{1}{2})^{2m}T_{m-1}(h)}{1-(\frac{1}{2})^{2m}} \\
&= \frac{4^m T_{m-1}(\frac{h}{2})-T_{m-1}(h)}{4^m-1}
\end{aligned}
$$

1.3 程序设计流程

编译通过，根据输入能得到正确输出。

[2]:
```python
# 添加需要的库


import numpy as np
from pandas import DataFrame
from typing import *
```

[3]:
```python
def romberg(
        f: Callable[[float], float],
        a: float, b: float, epsilon: float,
        *args,
        get_steps: bool = False, max_len: int = 32, **kwargs):
    max_len: int = 32
```

```python
    h = b - a
    i = 1
    T = np.array([[0.0 for _ in range(max_len)] for _ in range(max_len)])
    T[0][0] = (f(a) + f(b)) * h / 2
    # print(T[0][0])

    def get_slice():
        return np.array(T[0:(i+1), 0:(i+1)])
    while True:
        ii = 2**(i-1)
        # print(f"i = {i}, ii = {ii}")
        T[0][i] = T[0][i-1] / 2 + h * \
            sum([f(a + (0.0 + k - 1 / 2) * h) for k in range(1, ii + 1)]) / 2
        # print(f"T[0][i] = {T[0][i]}")
        for m in range(1, i + 1):
            k = i - m
            T[m][k] = (4**m * T[m-1][k+1] - T[m-1][k]) / (4**m - 1)
        # print(f"T[i][0] - T[i-1][0] = {T[i][0]} - {T[i-1][0]} = {T[i][0] - T[i-1][0]}")
        # print(f"T[i][0] - T[i-1][0] = {T[i][0] - T[i-1][0]}")
        if abs(T[i][0] - T[i-1][0]) < epsilon:
            if get_steps:
                return True, i
            else:
                return True, get_slice()
        h = h / 2
        i = i + 1
    if get_steps:
        return False, i
    else:
        return False, get_slice()
```

[4]:

```python
# 使用 Romberg 计算积分


global_args = [
    [lambda x: x**2 * np.exp(x), 0, 1, 1e-6],
```

```python
    [lambda x: np.sin(x) * np.exp(x), 1, 3, 1e-6],
    [lambda x: 4 / (1 + x**2), 0, 1, 1e-6],
    [lambda x: 1 / (1 + x), 0, 1, 1e-6]
]



def run_once(*args, show_result: bool = True, show_T: bool = True, **kwargs):
    res, T = romberg(*args, **kwargs)
    # print(T)
    if res:
        if not isinstance(T, int):
            if show_T:
                print(DataFrame(T))
            if show_result:
                print(f"result = {T[-1][0]}")
            return T[-1][0]
        else:
            return T
    else:
        print("Error")
        return None



def run(index: int, data_source=global_args, **kwargs):
    return run_once(*data_source[index], **kwargs)
```

[5]:
```python
# 第 (1) 问
run(0)
```

```
          0         1         2         3         4
0  1.359141  0.885661  0.760596  0.728890  0.720936
1  0.727834  0.718908  0.718321  0.718284  0.000000
2  0.718313  0.718282  0.718282  0.000000  0.000000
3  0.718282  0.718282  0.000000  0.000000  0.000000
4  0.718282  0.000000  0.000000  0.000000  0.000000
result = 0.7182818284623739
```

[5]: 0.7182818284623739

[6]:
```
# 第 (2) 问
run(1)
```

	0	1	2	3	4	5
0	5.121826	9.279763	10.520554	10.842043	10.923094	10.943398
1	10.665742	10.934151	10.949207	10.950111	10.950167	0.000000
2	10.952045	10.950210	10.950171	10.950170	0.000000	0.000000
3	10.950181	10.950170	10.950170	0.000000	0.000000	0.000000
4	10.950170	10.950170	0.000000	0.000000	0.000000	0.000000
5	10.950170	0.000000	0.000000	0.000000	0.000000	0.000000

result = 10.950170314683838

[6]: 10.950170314683838

[7]:
```
# 第 (3) 问
run(2)
```

	0	1	2	3	4	5
0	3.000000	3.100000	3.131176	3.138988	3.140942	3.14143
1	3.133333	3.141569	3.141593	3.141593	3.141593	0.00000
2	3.142118	3.141594	3.141593	3.141593	0.000000	0.00000
3	3.141586	3.141593	3.141593	0.000000	0.000000	0.00000
4	3.141593	3.141593	0.000000	0.000000	0.000000	0.00000
5	3.141593	0.000000	0.000000	0.000000	0.000000	0.00000

result = 3.141592653638244

[7]: 3.141592653638244

[8]:
```
# 第 (4) 问
run(3)
```

	0	1	2	3	4
0	0.750000	0.708333	0.697024	0.694122	0.693391
1	0.694444	0.693254	0.693155	0.693148	0.000000
2	0.693175	0.693148	0.693147	0.000000	0.000000
3	0.693147	0.693147	0.000000	0.000000	0.000000

```
4  0.693147  0.000000  0.000000  0.000000  0.000000
result = 0.6931471819167452
```

[8]:
```
0.6931471819167452
```

1.4 实验结果

准确规范地给出各个实验题目的结果，并对相应的思考题给出正确合理的回答与说明。

[9]:
```python
DataFrame([run(i, show_T=False, show_result=False)
           for i in range(3)], ["(1)", "(2)", "(3)"])
```

[9]:

	0
(1)	0.718282
(2)	10.950170
(3)	3.141593

实验题目 1 中各个小问的结果如上表格所示。

思考题：*在实验 1 中二分次数和精度的关系如何？*

我们使用更高的精度要求进行进一步测试：

[10]:
```python
def test_epsilon():

    def get_data(e: float):  # -> List[List[float]]:
        return [[*item[:-1], e] for item in global_args]

    def get_once(epsilon: float):
        return [run(i, data_source=get_data(epsilon), get_steps=True) for i in␣
 ↪range(3)]

    epsilon_list = [1e-5, 1e-6, 1e-9, 1e-12, 1e-14, 1e-16]
    print(DataFrame([get_once(e) for e in epsilon_list], epsilon_list))


test_epsilon()
```

```
              0   1   2
1.000000e-05  4   5   4
```

```
1.000000e-06  4   5   5
1.000000e-09  5   6   6
1.000000e-12  6   7   7
1.000000e-14  6   7   8
1.000000e-16  6  11  13
```

由数据可知，随着要求精度的提高，二分次数也在随之升高。