中国科学技术大学

实验报告

计算方法 B

Project 2

学生姓名：　朱云沁

学生学号：　PB20061372

完成时间：　二〇二二年四月五日

目录

一、 实验题目

编写程序, 用复化 Simpson 自动控制误差方式计算积分 $\int_a^b f(x)\,dx$.

输入: 积分区间 $[a,b]$, 精度控制值 ϵ, 定义函数 $f(x)$.

输出: 积分值 S.

利用 $\int_1^2 \ln x\,dx$, $\epsilon = 1\text{e-}4$ 验证结果.

二、 实验结果

1. 数值结果

n	$S_n(\ln x)$	$\frac{1}{15}\lvert S_n(\ln x) - S_{\frac{n}{2}}(\ln x)\rvert$
1	0.23104906018664842	
2	0.3858346021654338	0.010319036131919026
4	0.386259562814567	2.833070994221476e-05
8	0.386292043466313	2.1653767830661272e-06
16	0.38629421367579253	1.446806319675235e-07

表 1: 复化 Simpson 公式计算 $\int_1^2 \ln x\,dx$

2. 图示结果

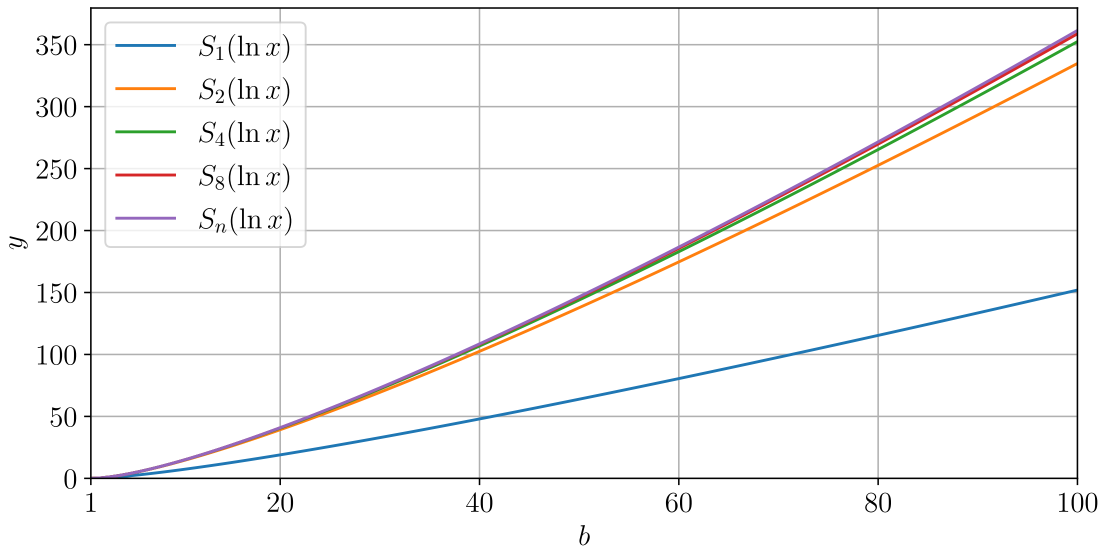

图 1: 复化 Simpson 公式计算 $\int_1^b \ln x\,dx$

三、 算法分析

1. Simpson 积分

记

$$I(f) = \int_a^b f(x)\,dx \tag{1}$$

将积分区间 $[a,b]\,n$ 等分, 记步长为 $h = \frac{b-a}{n}$, 取等分点为 $\{x_i = a + ih, i = 0, 1, \cdots, n\}$ 为数值积分节点, 构造 Lagrange 插值多项式 $L_n(x)$, 则

$$I(f) \approx I_n(f) = \int_a^b L_n(x)\,dx = (b-a)\sum_{i=0}^{n} C_i^{(n)} f(x_i) \tag{2}$$

$$C_i^{(n)} = \frac{(-1)^{n-i}}{i!(n-i)!n}\int_0^n t(t-1)\cdots(t-i+1)(t-i-1)\cdots(t-n)\,dt \tag{3}$$

称 $I_n(x)$ 为 Newton-Cotes 积分, $C_i^{(n)}$ 为 Newton-Cotes 系数.

在 Newton-Cotes 积分中, 取 $n = 2$, 有

$$S(f) = I_2(f) = \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] \tag{4}$$

称 $S(f)$ 为 Simpson 积分. 设 $f \in C^4[a,b]$, 则误差为

$$E_2(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi),\ a \le \xi \le b \tag{5}$$

2. 复化 Simpson 积分

由于插值的 Runge 现象, 高阶 Newton-Cotes 积分不能保证收敛性. 此外, 当 $n > 7$ 时, Newton-Cotes 积分的计算公式不稳定. 因此实际计算中常用低阶复化积分.

将积分区间 $[a,b]\,n$ 等分, 其中 n 为偶数. 记 $h = \frac{b-a}{n}$, $x_i = a + ih, i = 0, 1, \cdots, n$. 对每个子区间 $[x_{2i}, x_{2i+1}]$ 应用 Simpson 积分公式, 得复化 Simpson 积分 $S_n(f)$ 的计算公式

$$S_n(f) = \frac{h}{3}\left[f(a) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + 2\sum_{i=1}^{n/2-1} f(x_{2i}) + f(b)\right] \tag{6}$$

设 $f \in C^4[a,b]$, 则截断误差为

$$I(f) - S_n(f) = -\frac{b-a}{180} h^4 f^{(4)}(\xi),\ a \le \xi \le b \tag{7}$$

记 $[x_i, x_{i+1}]$ 的中点为 $x_{i+1/2}$, 进而将积分区间 $2n$ 等分, 得复化 Simpson 积分 $S_{2n}(f)$, 其截断误差

$$I(f) - S_{2n}(f) = -\frac{b-a}{180}\left(\frac{h}{2}\right)^4 f^{(4)}(\eta),\ a \le \eta \le b \tag{8}$$

作近似 $f^{(4)}(\xi) \approx f^{(4)}(\eta)$, 由 (7) 式和 (8) 式得后验误差估计方法

$$I(f) - S_{2n}(f) \approx \frac{1}{15}(S_{2n}(f) - S_n(f)) \tag{9}$$

3. 自适应复化 Simpson 积分算法

由上述讨论可知, 当分点无限增多, 复化 Simpson 积分收敛到 $I(f)$. 对任给的误差限 ϵ, 令区间段数 n 不断倍增, 依次计算复化 Simpson 数值积分, 直到

$$|S_{2n}(f) - S_n(f)| < 15\epsilon \tag{10}$$

可以认为, 此时已满足精度要求. 据此, 给出自动控制误差的复化积分算法如下

算法 1 自适应复化 Simpson 积分算法

输入: 积分区间 $[a, b]$; 精度控制值 ϵ; 定义函数 $f(x)$.

输出: 积分值 $I(f) = \texttt{S1}$.

$n \leftarrow 1,\ h \leftarrow b - a$;
$\texttt{S1} \leftarrow S_1(x)$; // 按 (4) 式计算.
$\texttt{S2} \leftarrow +\infty$;
while $|\texttt{S2} - \texttt{S1}| < 15\epsilon$ **do**
    $n \leftarrow 2n,\ h \leftarrow h/2$;
    $\texttt{S2} \leftarrow \texttt{S1}$;
    $\texttt{S1} \leftarrow S_n(x)$; // 按 (6) 式计算.
end

四、 结果分析

- 根据表 1, 当 $n = 1$ 时, 后验误差 $\frac{1}{15}|S_2(\ln x) - S_1(\ln x| > 1\text{e-}4$, 不满足精度要求; 当 $n = 2$ 时, 后验误差 $\frac{1}{15}|S_4(\ln x) - S_2(\ln x)| < 1\text{e-}4$, 满足精度要求. 故自适应复化 Simpson 算法得到的数值积分结果为

$$S = S_4(\ln x) = 0.386259562814567$$

- 由图 1 可知, 在积分上限 $b \in (1, 100)$ 范围内, 复化 Simpson 积分 $S_n(\ln x)$ 随 n 增大逐渐趋于自适应算法的结果, 映证了复化 Simpson 公式的收敛性.
- 接上一条, 当积分上限 b 逐渐增大, 固定分点数的复化 Simpson 公式的截断误差逐渐累积而增大, 映证了增加分点数的必要性.
- 比较表 2 (见附录) 和表 1, 图 2 和图 1, 可知在计算 $\int_1^b \ln x\, dx$ 时, Simpson 公式较梯形公式在自适应算法下收敛更快, 映证了 Newton-Cotes 型数值积分的精度理论. 然而在同样的分点数下, Simpson 积分的误差不一定比梯形积分小, 说明高阶 Newton-Cotes 型数值积分不能保证收敛性.

附录 A　其他数值积分方法的结果

1.　复化梯形积分

n	$T_n(\ln x)$	$\frac{1}{3}\lvert T_n(\ln x) - T_{\frac{n}{2}}(\ln x)\rvert$
1	0.34657359027997264	
2	0.37601934919406854	0.009815252971365299
4	0.38369950940944236	0.002560053405124607
8	0.3856439099520953	0.0006481335142176451
16	0.3861316377448683	0.00016257593092433575
32	0.3862536733329669	4.0678529366196724e-05

表 2: 复化梯形公式计算 $\int_1^2 \ln x\, dx$

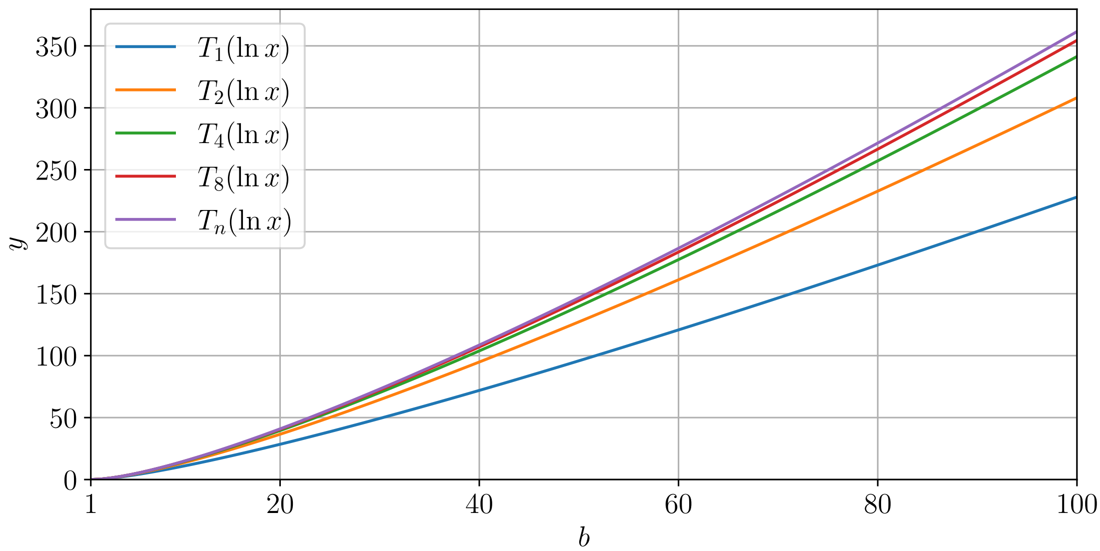

图 2: 复化梯形公式计算 $\int_1^b \ln x\, dx$

2.　Romberg 积分

k	$R_{k,1}$	$R_{k,2}$	$R_{k,3}$	$R_{k,4}$	$\lvert R_{k,k} - R_{k-1,k-1}\rvert$
1	0.3465736				
2	0.3760193	0.3858346			0.0392610
3	0.3836995	0.3862596	0.3862879		0.0004533
4	0.3856439	0.3862920	0.3862942	0.3862943	6.4155617e-06

表 3: Romberg 算法计算 $\int_1^2 \ln x\, dx$

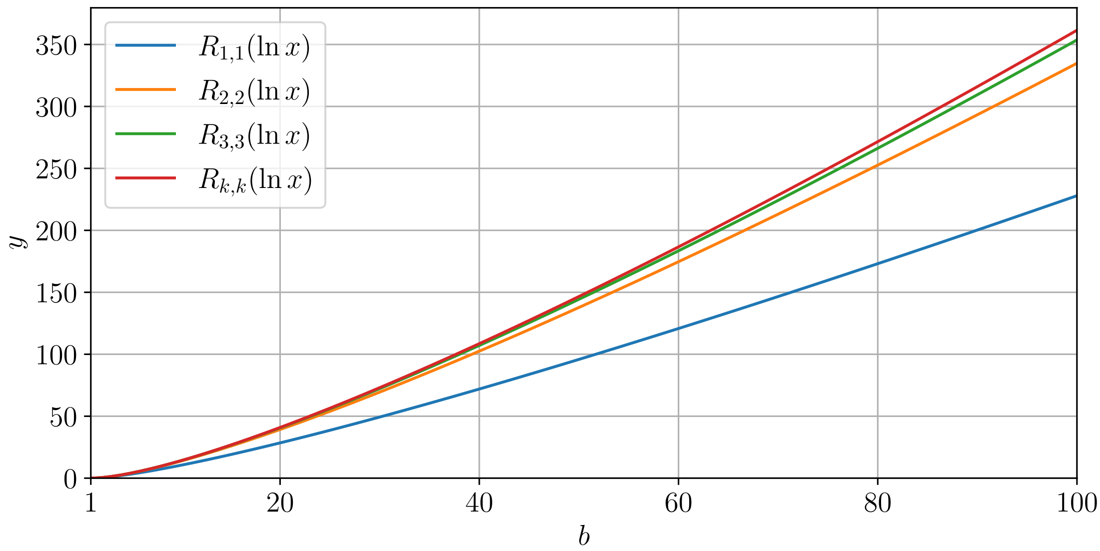

图 3: Romberg 算法计算 $\int_1^b \ln x \, dx$

3. Gauss-Legendre 积分

n	$G_n(x^5+x)$	$G_n(\ln x)$
1	-8.0	0.4054651081081644
2	-96.88888888888887	0.3865949441167409
3	-125.33333333333331	0.38630042158401123
4	-125.33333333333321	0.3862944969387142

表 4: Gauss-Legendre 公式计算 $\int_{-3}^{1}(x^5+x)\,dx$ 和 $\int_1^2 \ln x \, dx$

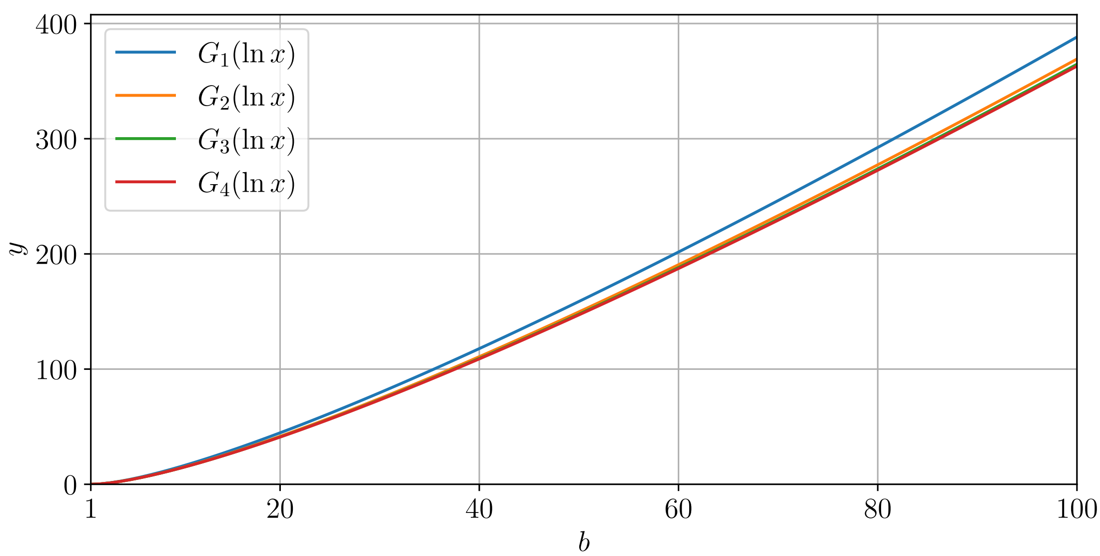

图 4: Gauss-Legendre 公式计算 $\int_1^b \ln x \, dx$

附录 B　Python 程序代码

1. 复化 Simpson 积分

```python
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from math import log

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)


def simpson(f, a, b, n):
    h = (b - a) / n
    s = f(a) + f(b)
    for i in range(1, n):
        if i % 2 == 0:
            s += 2 * f(a + i * h)
        else:
            s += 4 * f(a + i * h)
    return h * s / 3


def adaptive_simpson(f, a, b, M=100, eps=0, print_flag=False):
    k = 0
    S1 = simpson(f, a, b, 1)
    S2 = float('inf')
    if print_flag: print(f'1,{S1},')
    while abs(S1 - S2) >= 15 * eps and k < M - 1:
        k += 1
        S2 = S1
        S1 = simpson(f, a, b, 2**k)
        if print_flag: print(f'{2**k},{S1},{abs(S1-S2)/15}')
    return S1


print(adaptive_simpson(log, 1, 2, M=5, print_flag=True))

x_range = np.linspace(1, 100, 100)
fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)
```

```
ax.plot(x_range, [simpson(log, 1, x, 1) for x in x_range],
        label=r'$S_1(\ln{x})$')
ax.plot(x_range, [simpson(log, 1, x, 2) for x in x_range],
        label=r'$S_2(\ln{x})$')
ax.plot(x_range, [simpson(log, 1, x, 4) for x in x_range],
        label=r'$S_4(\ln{x})$')
ax.plot(x_range, [simpson(log, 1, x, 8) for x in x_range],
        label=r'$S_8(\ln{x})$')
ax.plot(x_range, [adaptive_simpson(log, 1, x, eps=1e-4) for x in
↪  x_range],
        label=r'$S_n(\ln{x})$')
ax.set_xticks(np.append(ax.get_xticks(), 1))
ax.set_xlim(1, 100)
ax.set_ylim(0)
ax.set_xlabel(r'$b$')
ax.set_ylabel(r'$y$')
ax.grid()
ax.legend()
fig.savefig('../assets/output/simpson.pdf')
```

2. 复化梯形积分

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from math import log

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)


def trapezoid(f, a, b, n):
    h = (b - a) / n
    s = f(a) + f(b)
    for i in range(1, n):
        s += 2 * f(a + i * h)
    return h * s / 2


```

```python
def adaptive_trapezoid(f, a, b, M=100, eps=0.0, print_flag=False):
    k = 0
    T1 = trapezoid(f, a, b, 1)
    T2 = float('inf')
    if print_flag: print(f'1,{T1},')
    while abs(T1 - T2) >= 3 * eps and k < M - 1:
        k += 1
        T2 = T1
        T1 = trapezoid(f, a, b, 2**k)
        if print_flag: print(f'{2**k},{T1},{abs(T1-T2)/3}')
    return T1


print(adaptive_trapezoid(log, 1, 2, eps=1e-4, print_flag=True))

x_range = np.linspace(1, 100, 100)
fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)
ax.plot(x_range, [trapezoid(log, 1, x, 1) for x in x_range],
        label=r'$T_1(\ln{x})$')
ax.plot(x_range, [trapezoid(log, 1, x, 2) for x in x_range],
        label=r'$T_2(\ln{x})$')
ax.plot(x_range, [trapezoid(log, 1, x, 4) for x in x_range],
        label=r'$T_4(\ln{x})$')
ax.plot(x_range, [trapezoid(log, 1, x, 8) for x in x_range],
        label=r'$T_8(\ln{x})$')
ax.plot(x_range, [adaptive_trapezoid(log, 1, x, eps=1e-4) for x in x_range],
        label=r'$T_n(\ln{x})$')
ax.set_xticks(np.append(ax.get_xticks(), 1))
ax.set_xlim(1, 100)
ax.set_ylim(0)
ax.set_xlabel(r'$b$')
ax.set_ylabel(r'$y$')
ax.grid()
ax.legend()
fig.savefig('../assets/output/trapezoid.pdf')
```

3. Romberg 积分

```python
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from math import log

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)


def romberg(f, a, b, M=100, eps=0.0, print_flag=False):
    k = 0
    R = [[float('inf')], [trapezoid(f, a, b, 1)]]
    if print_flag: print(f'1,{R[1]}')
    while abs(R[0][k - 1] - R[1][k]) >= eps and k < M - 1:
        k += 1
        R[0] = R[1]
        R[1] = [trapezoid(f, a, b, 2**k)]
        for j in range(1, k + 1):
            R[1].append((4**j * R[1][j - 1] - R[0][j - 1]) / (4**j - 1))
        if print_flag: print(f'{k+1},{R[1]},{abs(R[0][k - 1] - R[1][k])}')
    return R[1][k]


romberg(log, 1, 2, eps=1e-4, print_flag=True)

x_range = np.linspace(1, 100, 100)
fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)
ax.plot(x_range, [romberg(log, 1, x, M=1) for x in x_range],
        label=r'$R_{1,1}(\ln{x})$')
ax.plot(x_range, [romberg(log, 1, x, M=2) for x in x_range],
        label=r'$R_{2,2}(\ln{x})$')
ax.plot(x_range, [romberg(log, 1, x, M=3) for x in x_range],
        label=r'$R_{3,3}(\ln{x})$')
ax.plot(x_range, [romberg(log, 1, x, eps=1e-4) for x in x_range],
        label=r'$R_{k,k}(\ln{x})$')
ax.set_xticks(np.append(ax.get_xticks(), 1))
ax.set_xlim(1, 100)
```

```python
ax.set_ylim(0)
ax.set_xlabel(r'$b$')
ax.set_ylabel(r'$y$')
ax.grid()
ax.legend()
fig.savefig('../assets/output/romberg.pdf')
```

4. Gauss-Legendre 积分

```python
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from math import log
from numpy.polynomial.legendre import leggauss

mpl.rc('font', family='serif', size=15)
mpl.rc('text', usetex=True)


def gauss_legendre(f, a, b, n):
    x, w = leggauss(n)
    return (b - a) / 2 * sum(
        [w[i] * f((a + b + (b - a) * x[i]) / 2) for i in range(n)])


print(gauss_legendre(lambda x: x**5 + x, -3, 1, 1))
print(gauss_legendre(lambda x: x**5 + x, -3, 1, 2))
print(gauss_legendre(lambda x: x**5 + x, -3, 1, 3))
print(gauss_legendre(log, 1, 2, 1))
print(gauss_legendre(log, 1, 2, 2))
print(gauss_legendre(log, 1, 2, 3))

x_range = np.linspace(1, 100, 100)
fig, ax = plt.subplots(figsize=(8, 4), constrained_layout=True)
ax.plot(x_range, [gauss_legendre(log, 1, x, 1) for x in x_range],
        label=r'$G_1(\ln{x})$')
ax.plot(x_range, [gauss_legendre(log, 1, x, 2) for x in x_range],
        label=r'$G_2(\ln{x})$')
ax.plot(x_range, [gauss_legendre(log, 1, x, 3) for x in x_range],
```

```python
        label=r'$G_3(\ln{x})$')
ax.plot(x_range, [gauss_legendre(log, 1, x, 4) for x in x_range],
        label=r'$G_4(\ln{x})$')
ax.set_xticks(np.append(ax.get_xticks(), 1))
ax.set_xlim(1, 100)
ax.set_ylim(0)
ax.set_xlabel(r'$b$')
ax.set_ylabel(r'$y$')
ax.grid()
ax.legend()
fig.savefig('../assets/output/gauss-legendre.pdf')
```