数值分析第 5 章上机

By 211870125 陈睿硕

§1 问题

Q1 数学上已经证明 $\int_0^1 f(x)dx = \pi$，其中 $f(x) = \frac{4}{1+x^2}$。分别使用复合梯形，复合 Simpson 求积公式计算 π 的近似值。选择不同的 h , 对每种求积公式，试将误差刻画成对每种求积公式，试将误差刻画成 h 的函数，并比较两方法的精度。是否存在某个 h 值，当低于这个值之后再继续减小 h 的值，计算不再有所改进？为什么？

Q2 实现 Romberg 求积方法，重复**Q1**的过程。

Q3 实现自适应积分方法，重复**Q1**的过程。

Q4 用所掌握的所有数值积分方法计算积分 $\int_0^\infty \frac{x^3}{e^x-1}dx$，并比较不同方法的计算效率和精度。

§2 算法思路

I 对Q1的解答

我们实现了复合梯形，复合 Simpson 求积方法，代码见代码 1。得到 h 和误差（绝对值）的关系如图 1。

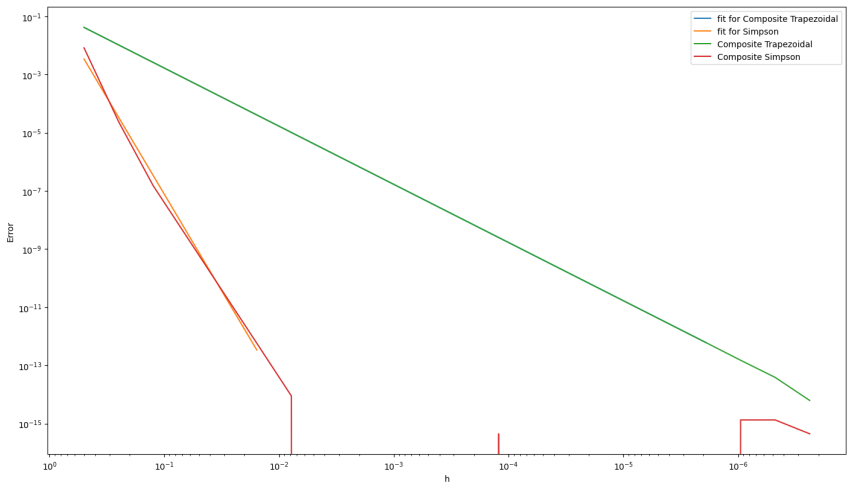

图 1: 复合梯形，复合 Simpson 求积法的误差随 h 的变化

II 对Q2的解答

使用 Python 实现，代码见代码 2。得到 h 和误差（绝对值）的关系如图 2。

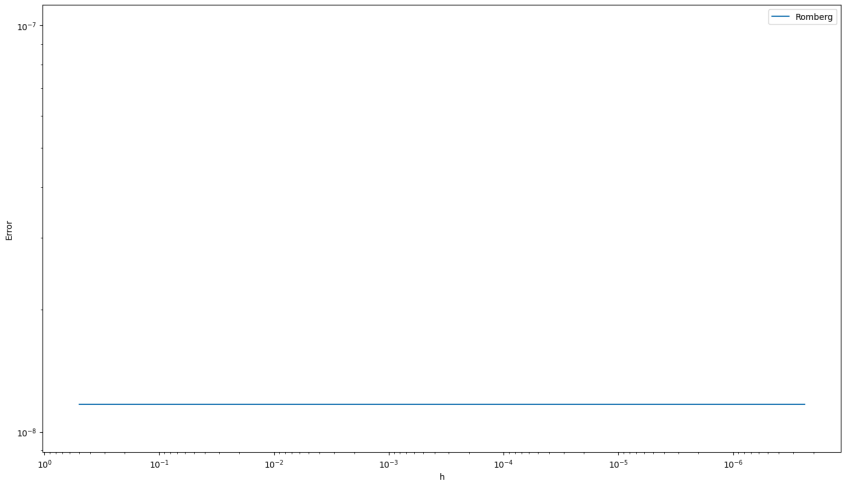

图 2: Romberg 求积法的误差随 h 的变化

III 对Q3的解答

使用 Python 实现，代码见代码 3。得到 h 和误差（绝对值）的关系如图 3。

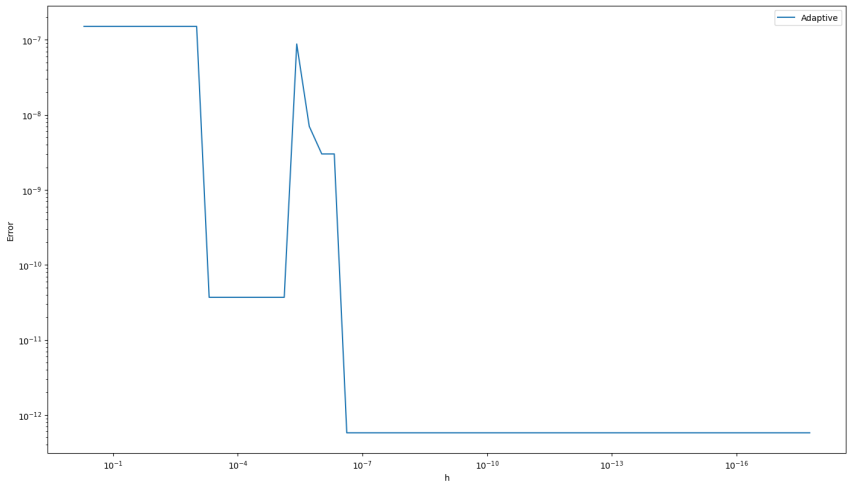

图 3: 自适应 Simpson 积分方法的误差随 h 的变化

IV 对Q4的解答

首先，我们使用 $t = \frac{1}{1+x}$ 的坐标变换将积分区间从 $[0, +\infty]$ 变换到 $[0, 1]$, 并补充定义 0 和 1 处的函数值为 0。然后我们使用了复合梯形公式、复合 Simpson 公式、Romberg 积分法、自适应积分法计算变换后的积分，代码见代码 4。得到计算出的误差（用数学方法可以算出此积分的精确值为 $\frac{\pi^4}{15}$）和 h 的关系如图 4。

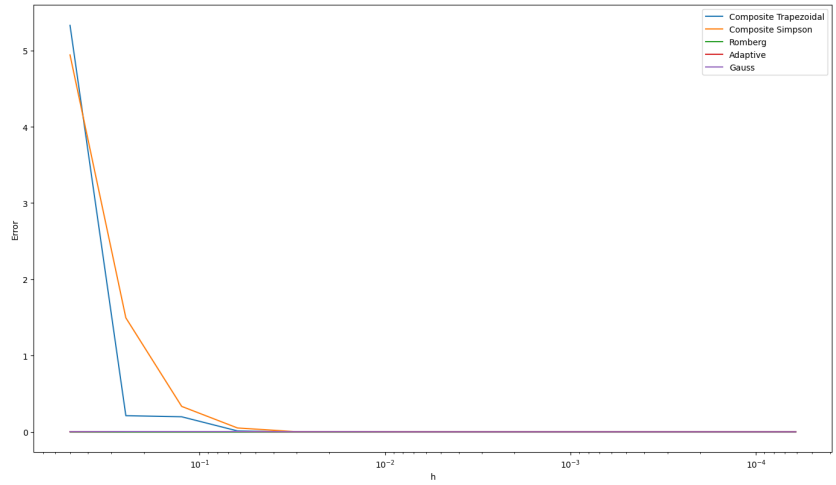

图 4: 多种方法计算积分的误差

得到各种方法在 h 取不同值的计算时间如图 5。

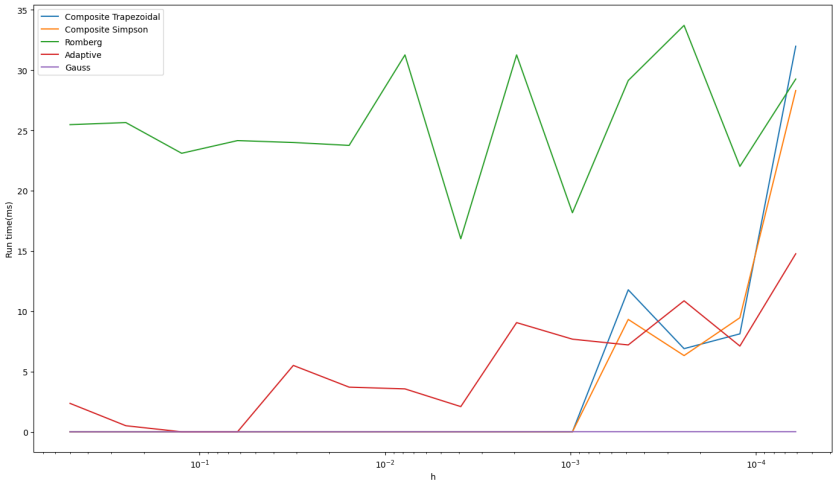

图 5: 多种方法计算积分的运行时间

§3 结果分析

I 对Q1的分析

可以看到，h 和误差的关系在 h 不特别小时几乎是对数线性关系，对于复合积分法 $Error \approx 10^{-0.77821}h^{2.0000}$，而对于复合 Simpson 法 $Error \approx 10^{-0.46800}h^{6.6460}$。

我们知道，复合梯形公式的误差（绝对值）$Error = \frac{1}{12}h^2 f''(\xi)$，其中 $\xi \in (0,1)$，这与我们的估计式相吻合（$f''(x)$ 在 $[0,1]$ 上变化不大）。

而复合 Simpson 公式的误差（绝对值）$Error = \frac{1}{180}h^4 f''(\xi)$，其中 $\xi \in (0,1)$, 这就与我们的估计式有一定的差距，可能是因为函数拟合的位置 h 值还比较大，也可能是这个被积函数所特有的“过收敛现象”。

h 小于 10^{-2} 次时，复合 Simpson 求积法的误差不再发生明显缩小，因为此时舍入误差对误差的贡献开始提升。

II 对Q2的分析

我们可以看到，随着 h 的减小，误差几乎没有变化，这是因为在 Romberg 积分法迭代时，每迭代一次，插值点间的距离便会缩小一半，所以算法的效果并不很依赖于初始的 h 值。

III 对Q3的分析

我们可以看到，误差变动很大，但在 h 小于 10^{-7} 后稳定在了一个值，这应该是因为每一个小区间中积分的误差都已经小于规定的阈值（10^{-6}）。

IV 对Q4的分析

各种方法的最小、最大误差和对于某个 h 计算使用的最长时间如下：

Method	Min error	Max error	Run time(ms)
复合梯形	8.881784197001252e-16	5.3299859885281755	31.97479248046875
复合 Simpson	8.881784197001252e-16	4.942001517281957	28.298377990722
Romberg	8.881784197001252e-15	8.881784197001252e-15	33.70404243469238
自适应 Simpson	1.8865497430908817e-07	0.0036696045968218627	14.763355255126953
复合 Gauss 两点	1.8865497430908817e-07	0.0036696045968218627	8.5732936859

注：Romberg 求积法中不同的迭代终止条件会极大地影响算法的结果与运行时间，于是我们设置了最大迭代次数为 14 作为唯一的终止条件，使其运行时间与其他方法较为接近，以便比较。

结合上表、图 4和图 5，复合梯形公式与复合 Simpson 公式在 h 足够小时误差能够达到浮点数下界，但运行时间较长，并且需要随着 h 的减小收敛；Romberg 求积法得益于多次的迭代，对初始的 h 值非常不敏感，运行时间长但能够始终维持误差几乎不变。而自适应和复合 Gauss 两点求积法能达到的精度较低，但运行时间有较大优势，同时误差始终保持在一个较低水准，稳定性很强。

不过事实上，我们可以发现，h 在 10^{-3} 次处复合梯形公式和复合 Simpson 公式的运行时间几乎为 0，但是此处计算的误差已经达到 10^{-15} 的量级，所以从单个任务的应用来说，我们可以选取合适的 h，使用复合梯形或者复合 Simpson 求积法达到时间和精度的完美平衡。

§4 附录：程序代码

代码 1: Fifth Chapter 1A.py

```python
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    return 4 / (1 + x**2)


def composite_trapezoidal(a, b, h):
    n = int((b - a) / (2 * h)) * 2
```

```
    x = np.linspace(a, b, n+1)
    y = f(x)
    s = (y[0] + y[n] + 2 * np.sum(y[1:n])) * h / 2
    return s


def composite_simpson(a, b, h):
    n = int((b - a) / h)
    x = np.linspace(a, b, n+1)
    y = f(x)
    s = (y[0] + y[n] + 4 * np.sum(y[1:n:2]) + 2 * np.sum(y[2:n-1:2])) * h
        / 3
    return s


a, b = 0, 1

trap_errors = np.array([])
simp_errors = np.array([])

h_values = np.array([0.5**i for i in range(1, 23)])

for i in h_values:
    trap_error = np.abs(composite_trapezoidal(a, b, i) - np.pi)
    simp_error = np.abs(composite_simpson(a, b, i) - np.pi)
    trap_errors = np.append(trap_errors, trap_error)
    simp_errors = np.append(simp_errors, simp_error)

mask1 = np.where(h_values > 1e-6)
coefficients1 = np.polyfit(
    np.log10(h_values[mask1]), np.log10(trap_errors[mask1]), 1)
# 输出拟合的函数式
print(
    f"对于复合梯形公式拟合的函数式为 log10(y) = {coefficients1[0]}log10(x
       ) + {coefficients1[1]}")

mask2 = np.where(h_values >= 10**(-2.1))
coefficients2 = np.polyfit(
    np.log10(h_values[mask2]), np.log10(simp_errors[mask2]), 1)
# 输出拟合的函数式
print(
```

```
    f"对于复合梯形公式拟合的函数式为 log10(y) = {coefficients2[0]}log10(x
       ) + {coefficients2[1]}")

poly_fit1 = np.poly1d(coefficients1)
poly_fit2 = np.poly1d(coefficients2)
plt.plot(h_values[mask1], np.power(10, poly_fit1(
    np.log10(h_values[mask1]))), label='fit for Composite Trapezoidal')
plt.plot(h_values[mask2], np.power(10, poly_fit2(
    np.log10(h_values[mask2]))), label='fit for Simpson')
plt.plot(h_values, trap_errors, label="Composite Trapezoidal")
plt.plot(h_values, simp_errors, label="Composite Simpson")
plt.xscale('log')
plt.yscale('log')
plt.xlabel('h')
plt.ylabel('Error')
plt.gca().invert_xaxis()
plt.legend()
plt.show()
```

代码 2: Fifth Chapter 1B.py

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


def f(x):
    return 4 / (1 + x**2)

def romberg(f, a, b, h, max_iterations=20, epsilon=1e-7):
    R = [[0] * (max_iterations + 1) for _ in range(max_iterations + 1)]
    R[0][0] = 0.5 * (b - a) * (f(a) + f(b))
    for i in range(1, max_iterations + 1):
        h = (b - a) / (2 ** i)
        sum = 0.0
        for k in range(1, 2 ** (i - 1) + 1):
            x = a + (k - 0.5) * 2 * h
            sum += f(x)
        R[i][0] = 0.5 * R[i - 1][0] + h * sum
        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 j==i and np.abs(R[i][j]-R[i][j-1])<epsilon:
```

```
                return R[i][j]
    return R[max_iterations][max_iterations]

a, b = 0, 1

romberg_errors = np.array([])

h_values = np.array([0.4**i for i in range(1, 15)])

for i in h_values:
    romberg_error = np.abs(romberg(f, a, b, i) - np.pi)
    romberg_errors = np.append(romberg_errors, romberg_error)

plt.plot(h_values, romberg_errors, label="Romberg")
plt.xscale('log')
plt.yscale('log')
plt.xlabel('h')
plt.ylabel('Error')
plt.gca().invert_xaxis()
plt.legend()
plt.show()
```

代码 3: Fifth Chapter 1C.py

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


def f(x):
    return 4 / (1 + x**2)

def composite_simpson(f, a, b, h):
    n = int((b - a) / (2 * h)) * 2
    x = np.linspace(a, b, n+1)
    y = f(x)
    s = (y[0] + y[n] + 4 * np.sum(y[1:n:2]) + 2 * np.sum(y[2:n-1:2])) * h
        / 3
    return s

def adaptive_integration(f, a, b, h, epsilon=1e-6, max_depth=50):
    x0 = a
    x2 = b
```

```
    x1 = (a + b) / 2.0
    S0 = composite_simpson(f, a, b, (b-a)/2)
    S1 = composite_simpson(f, x0, x1, (x1-x0)/2)
    S2 = composite_simpson(f, x1, x2, (x2-x1)/2)
    error = abs(S0 - S1 - S2)
    if error < epsilon or max_depth == 0:
        return S1 + S2
    else:
        left_integral = adaptive_integration(
            f, x0, x1, epsilon / 2, h / 2, max_depth - 1)
        right_integral = adaptive_integration(
            f, x1, x2, epsilon / 2, h / 2, max_depth - 1)
        return left_integral + right_integral


a, b = 0, 1

adaptive_errors = np.array([])

h_values = np.array([0.5**i for i in range(1, 60)])

for i in h_values:
    adaptive_error = np.abs(adaptive_integration(f, a, b, i) - np.pi)
    adaptive_errors = np.append(adaptive_errors, adaptive_error)

plt.plot(h_values, adaptive_errors, label="Adaptive")
plt.xscale('log')
plt.yscale('log')
plt.xlabel('h')
plt.ylabel('Error')
plt.gca().invert_xaxis()
plt.legend()
plt.show()
```

代码 4: Fifth Chapter 2.py

```
import matplotlib.pyplot as plt
import time

def f(t):
        return t**3 / (1 - t)**5 * 1/(np.exp(t / (1 - t))-1) if t!=0 and
           t!=1 else 0
```

```
def composite_trapezoidal(f, a, b, h):
    n = int((b - a) / (2 * h)) * 2
    x = np.linspace(a, b, n+1)
    y = list(map(f, x))
    s = (y[0] + y[n] + 2 * np.sum(y[1:n])) * h / 2
    return s


def composite_simp_errorson(f, a, b, h):
    n = int((b - a) / h)
    x = np.linspace(a, b, n+1)
    y = list(map(f,x))
    s = (y[0] + y[n] + 4 * np.sum(y[1:n:2]) + 2 * np.sum(y[2:n-1:2])) * h
        / 3
    return s


def romberg_integration(f, a, b, h, max_iterations=14, epsilon=1e-13):
    R = [[0] * (max_iterations + 1) for _ in range(max_iterations + 1)]
    R[0][0] = 0.5 * (b - a) * (f(a) + f(b))
    for i in range(1, max_iterations + 1):
        h = (b - a) / (2 ** i)
        sum = 0.0
        for k in range(1, 2 ** (i - 1) + 1):
            x = a + (k - 0.5) * 2 * h
            sum += f(x)
        R[i][0] = 0.5 * R[i - 1][0] + h * sum
        for j in range(1, i + 1):
            R[i][j] = (4 ** j * R[i][j - 1] - R[i - 1][j - 1]) / (4 ** j
               - 1)
    return R[max_iterations][max_iterations]


def adaptive_integration(f, a, b, h, epsilon=1e-6, max_depth=50):
    x0 = a
    x2 = b
    x1 = (a + b) / 2.0
    S0 = composite_simp_errorson(f, a, b, (b-a)/2)
    S1 = composite_simp_errorson(f, x0, x1, (x1-x0)/2)
```

```
    S2 = composite_simp_errorson(f, x1, x2, (x2-x1)/2)
    error = abs(S0 - S1 - S2)
    if error < epsilon or max_depth == 0:
        return S1 + S2
    else:
        left_integral = adaptive_integration(
            f, x0, x1, epsilon / 2, h / 2, max_depth - 1)
        right_integral = adaptive_integration(
            f, x1, x2, epsilon / 2, h / 2, max_depth - 1)
        return left_integral + right_integral


def composite_gauss_twopoint(f, a, b, h):
    x = np.array([-np.sqrt(1/3), np.sqrt(1/3)])
    w = np.array([1, 1])
    n = (b - a) / h
    intervals = np.linspace(a, b, n+1)
    integral = 0
    for i in range(n):
        x_new = (x + 1) * intervals[i] / 2 + (1 - x) * intervals[i+1] / 2
        integral += h/2 * np.sum(w * f(x_new))
    return integral

a, b = 0, 1
h_values = np.array([0.5**i for i in range(1, 15)])

trap_errors = np.array([])
trap_times=np.array([])
for i in h_values:
    _st = time.time()
    trap = np.abs(composite_trapezoidal(f, a, b, i))
    _ed=time.time()
    trap_errors = np.append(trap_errors, np.abs(trap-np.power(np.pi,4)
       /15))
    trap_times=np.append(trap_times,_ed-_st)
print(f"复合梯形公式最小误差: {np.min(trap_errors)}, 最大误差: {np.max(
   trap_errors)}, 最多用时（ms）: {1000*max(trap_times)}")

simp_errors = np.array([])
simp_times=np.array([])
for i in h_values:
```

```
    _st = time.time()
    simp = np.abs(composite_simp_errorson(f, a, b, i))
    _ed = time.time()
    simp_errors = np.append(simp_errors, np.abs(simp-np.power(np.pi,4)
       /15))
    simp_times=np.append(simp_times,_ed-_st)
print(f"复合Simpson公式最小误差: {np.min(simp_errors)}, 最大误差: {np.max
   (simp_errors)}, 最多用时（ms）: {1000*max(simp_times)}")

romberg_errors = np.array([])
romberg_times=np.array([])
for i in h_values:
    _st = time.time()
    romberg = np.abs(romberg_integration(f, a, b, i))
    _ed = time.time()
    romberg_errors = np.append(romberg_errors, np.abs(romberg-np.power(np
       .pi,4)/15))
    romberg_times=np.append(romberg_times,_ed-_st)
print(f"Romberg最小误差: {np.min(romberg_errors)}, 最大误差: {np.max(
   romberg_errors)}, 最多用时（ms）: {1000*max(romberg_times)}")

adaptive_errors = np.array([])
adaptive_times=np.array([])
for i in h_values:
    _st = time.time()
    adaptive = np.abs(adaptive_integration(f, a, b, i))
    _ed = time.time()
    adaptive_errors = np.append(adaptive_errors, np.abs(adaptive-np.power
       (np.pi,4)/15))
    adaptive_times=np.append(adaptive_times,_ed-_st)
print(f"自适应最小误差: {np.min(adaptive_errors)}, 最大误差: {np.max(
   adaptive_errors)}, 最多用时（ms）: {1000*max(adaptive_times)}")

gauss_errors=np.array([])
gauss_times=np.array([])
for i in h_values:
    _st = time.time()
    gauss = np.abs(adaptive_integration(f, a, b, i))
    _ed = time.time()
    gauss_errors = np.append(gauss_errors, np.abs(gauss-np.power(np.pi,4)
       /15))
```

```
    gauss_times=np.append(gauss_times,_ed-_st)
print(f"复合Gauss两点最小误差: {np.min(gauss_errors)}, 最大误差: {np.max(
    gauss_errors)}, 最多用时（ms）: {1000*max(gauss_times)}")

plt.plot(h_values, trap_errors, label="Composite Trapezoidal")
plt.plot(h_values, simp_errors, label="Composite Simpson")
plt.plot(h_values, romberg_errors, label="Romberg")
plt.plot(h_values, adaptive_errors, label="Adaptive")
plt.plot(h_values, gauss_errors, label="Gauss")
plt.xscale('log')
plt.xlabel('h')
plt.ylabel('Error')
plt.gca().invert_xaxis()
plt.legend()
plt.show()

plt.plot(h_values, trap_times*1000, label="Composite Trapezoidal")
plt.plot(h_values, simp_times*1000, label="Composite Simpson")
plt.plot(h_values, romberg_times*1000, label="Romberg")
plt.plot(h_values, adaptive_times*1000, label="Adaptive")
plt.plot(h_values, gauss_times, label="Gauss")
plt.xscale('log')
plt.xlabel('h')
plt.ylabel('Run time(ms)')
plt.gca().invert_xaxis()
plt.legend()
plt.show()
```