数值分析第 4 次作业

211240021 田铭扬

§1 问题

1. 使用数值积分 $\int_0^1 \frac{4}{1+x^2}dx$ 来求 **π** 的近似值。

1）分别用复合梯形，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 **h** 的函数，并比较两种方法的精度。是否存在某个 **h** 值，当低于这个值之后再继续减小 **h** 的值，计算结果不再有所改进？为什么？

2）实现 Romberg 求积方法，并重复上面的计算。

3）实现自适应积分方法，并重复上面的计算。

2. 用所掌握的所有数值积分方法计算 $\int_0^\infty \frac{x^3}{e^x-1}dx$ ，并比较计算精度和效率。

§2 问题分析与实现

由于各数值积分方法均在课上进行过详细的推导，可以按部就班地编程实现，故此处不再过多分析。仅有几处难点需要简要说明：

首先，Dev C++ IDE 自带的程序运行时间统计只能精确到 1ms，对于效率较高的几种算法，难以看出差距。故我使用了 chrono 头文件中的 steady_clock 方法，以在 1μs 的精度上统计程序运行时间。

其次，第 2 题所求积分是无穷积分，可以进行如下的变换：

$$\int_1^\infty \frac{x^3}{e^x-1}dx=\int_1^0 \frac{y^{-3}}{e^{y^{-1}}-1}d\frac{1}{y}=\int_0^1 \frac{y^{-5}}{e^{y^{-1}}-1}dy$$

将原积分转化为有限区间[0,1]上的积分：

$$\int_0^\infty \frac{x^3}{e^x-1}dx=\int_0^1 \frac{x^3}{e^x-1}+\frac{x^{-5}}{e^{x^{-1}}-1}dx \quad (1)$$

特别注意到，**x**→0 时被积函数极限为 0，但直接代入 **x**=0 没有意义。因此在计算函数值的子程序中要进行判断，若输入值小于 1e-7，则要返回 0。

§3 运行结果与分析

（一）第 1 题

1. 1）复合梯形/Simpson 算法 结果如下表。

“用时”单位为毫秒(ms)

梯形	步长	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
	用时	0.000	0.000	0.000	0.990	10.916	109.751
	误差	1.67×10^{-7}	1.67×10^{-9}	1.66×10^{-11}	4.44×10^{-16}	-1.94×10^{-13}	-4.32×10^{-13}
Simpson	步长	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
	用时	0.000	0.000	0.000	1.010	11.001	108.679
	误差	6.67×10^{-7}	6.67×10^{-9}	6.67×10^{-11}	7.44×10^{-13}	1.36×10^{-13}	-4.35×10^{-14}
Simpson	步长	2×10^{-3}	2×10^{-4}	2×10^{-5}	2×10^{-6}	2×10^{-7}	2×10^{-8}
	用时	0.000	0.000	0.000	2.010	20.009	204.503
	误差	1.67×10^{-7}	1.67×10^{-9}	1.66×10^{-11}	2.19×10^{-13}	1.47×10^{-13}	8.84×10^{-13}

表 1 第 1 题 复合梯形公式与复合 Simpson 公式性能对比

注意到，当步长小于 10^{-6} 时，步长的缩短对于数值积分精度提升已没有明显的帮助，此时的误差“阈值”在 10^{-13} 数量级。甚至在使用复合梯形公式时出现了不稳定（误差增大）的现象。猜测此现象产生的原因与算法本身无关，而是由于机器精度的限制，积分区间变多时，舍入误差产生了显著的影响。

上表中的数据是使用 double 精度得到的，因而可以进一步实验：将程序中的所有 double 型变量均改为 float 型，得到了如下表的数据。我们发现，上述“阈值”现象变得极为明显，这就证实了前述猜测。

梯形 double 精度	步长	/	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
	用时	/	0.000	0.000	0.000	0.990	10.916
	误差	/	1.67×10^{-7}	1.67×10^{-9}	1.66×10^{-11}	4.44×10^{-16}	-1.94×10^{-13}
梯形 float 精度	步长	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
	用时	0.000	0.000	0.000	0.000	0.999	10.005
	误差	1.69×10^{-5}	7.15×10^{-7}	-9.53×10^{-9}	6.06×10^{-5}	2.44×10^{-4}	0.0422

表 2 第 1 题 复合梯形公式 float 精度与 double 精度对比

此外，根据表 1 的数据，复合 Simpson 公式相对于复合梯形公式并无性能上的优势，这与理论上不太相符，暂时还不清楚原因。

1. 2）Romberg 算法 结果如下表。

TOL	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}	10^{-18}	10^{-21}
用时(ms)	4.114	4.742	4.686	3.340	3.725	4.840	5.236
误差	5.27×10^{-3}	8.15×10^{-5}	5.09×10^{-6}	7.95×10^{-7}	4.97×10^{-9}	1.24×10^{-9}	1.24×10^{-9}
最终步长	2^{-2}	2^{-5}	2^{-7}	2^{-10}	2^{-12}	2^{-13}	2^{-13}

表 3 第 1 题 Romberg 求积方法

从表中可以看到，Romberg 求积法的表现并不好。它出现了类似于复合梯形/Simpson 公式的“阈值”现象，而且阈值时误差大得多（4 个数量级）。

此外，此算法有约 4ms（且不太稳定）的与精度无关的耗时，这可能是算法本身固有的复杂度导致的，具体原因还不清楚。

1. 3）自适应梯形算法 结果如下表。

误差限	10^{-3}	10^{-5}	10^{-7}	10^{-9}	10^{-11}	10^{-13}	10^{-15}
用时(ms)	0.000	0.000	0.000	0.000	2.012	22.099	212.940
误差	4.22×10^{-5}	2.23×10^{-7}	3.15×10^{-9}	2.89×10^{-11}	2.50×10^{-13}	3.11×10^{-15}	1.22×10^{-16}
最大深度	6	9	13	16	19	23	26

表 4 第 1 题 自适应梯形求积方法

可以看到，自适应梯形方法对于本题积分的表现很好。它是第 1 题中唯一能达到 10^{-16} 数量级的精度——注意到 $10^{-16}\approx2^{-52}$，即 double 型的机器精度——的算法，而且用时可以接受（200ms）。因而可以推测，对于本题中的积分，如果能够使用精度更高的数据类型，采用此方法将会得到更好的精度（不过耗时也会相应增加）。

综上，第 1 题中表现最好的是自适应梯形方法。

（二）第 2 题

容易得到所求积分的精确值为 $\pi^4/15$，用于计算各算法的误差。

另外，由于部分算法较为相似，或有明显的“上位替代”关系，因此我将只比较下述算法：两点 Gauss-Laguerre 算法、区间分次减半梯形/Simpson 算法、Romberg 算法、自适应梯形算法、两点复合 Gauss-Legendre 算法。

2. 1）两点 Gauss-Laguerre 算法

因为此方法只用到两个基点，所以耗时可以忽略。使用 Mathmatica 进行计算并于精确值比较，误差约为 8.02×10^{-2}，在精度需求不大的场景可以接受。

```
In[ ]:= f[x_] = x^3 / (Exp[x] - 1);
        g[x_] = f[x] * Exp[x];
        a = Integrate[f[x], {x, 0, Infinity}];
        b = ((2 + Sqrt[2]) * g[2 - Sqrt[2]] + (2 - Sqrt[2]) * g[2 + Sqrt[2]]) / 4;
        N[b, 15]
        N[Abs[b - a], 15]
Out[ ]= 6.41372746951758
Out[ ]= 0.0802119327492464
```

图 1 第 2 题 两点 Gauss-Laguerre 算法

2. 2）区间分次减半梯形/Simpson 算法 结果如下表。

	区间分次减半梯形算法			区间分次减半 Simpson 算法		
TOL	用时(ms)	误差	最终步长	用时(ms)	误差	最终步长
10^{-6}	1.993	1.48×10^{-6}	2^{-8}	0.997	2.02×10^{-8}	2^{-7}
10^{-9}	1.994	1.45×10^{-9}	2^{-13}	1.995	3.25×10^{-12}	2^{-9}
10^{-12}	27.420	1.39×10^{-12}	2^{-18}	1.992	2.03×10^{-13}	2^{-10}
10^{-15}	超时	/	/	3.987	-1.24×10^{-14}	2^{-15}

表 5 第 2 题 区间分次减半梯形/Simpson 算法性能比较

这里的结果与理论相符，即 Simpson 算法的表现比梯形算法好，而并没有出现 1.1)的情况。暂时不清楚这是由于“逐次减半”方法规避了事先指定步长的某种缺点，还是第 1 题的被积函数有某种特殊性。

在采集数据过程中，发现此算法的用时并不稳定，尤其是在用时较短的时候。此外，在 TOL 较大时，还出现用时并不随 TOL 减小而显著增长的情况。这有可能是因为算法固有的复杂度（与第 1 题 2)的情况类似）；也可能是由于某些原因，导致此方法受到系统的“后台占用”情况的影响较大。

2. 3）Romberg 算法 结果如下表。

TOL	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}	10^{-18}
用时(ms)	1.504	1.905	2.019	0.995	2.941	2.782
误差	7.33×10^{-4}	1.03×10^{-3}	3.19×10^{-6}	4.63×10^{-8}	2.89×10^{-9}	2.89×10^{-9}
最终步长	2^{-4}	2^{-5}	2^{-7}	2^{-10}	2^{-12}	2^{-12}

表 6 第 2 题 Romberg 求积方法

出现的现象与 1.2)相仿，故不再详述。

2. 4）自适应梯形算法 结果如下表。

误差限	10^{-3}	10^{-5}	10^{-7}	10^{-9}	10^{-11}	10^{-13}	10^{-15}
用时(ms)	0.000	0.000	4.061	41.945	443.723	3948.781	超时
误差	-1.16×10^{-4}	-1.18×10^{-6}	-1.23×10^{-8}	-1.15×10^{-10}	-1.19×10^{-12}	-1.24×10^{-14}	/
最大深度	11	14	17	21	24	27	/

表 7 第 2 题 自适应梯形求积方法

在本题中，自适应梯形方法的性能表现不如区间逐次分半 Simpson 方法：两者的绝对误差均能达到 10^{-14} 数量级，但是前者的用时是后者的 10^3 倍。事实上，前者最小步长为 2^{-27}，而后者步长为 2^{-15}，相差 $2^{12}=4096\approx4\times10^3$ 倍。我们观察 (1) 式被积函数的图像（下图），发现它在 [0,1] 区间的多数地方都比较“陡峭”（即更有可能需要分半），这样的结果也就不难理解了。

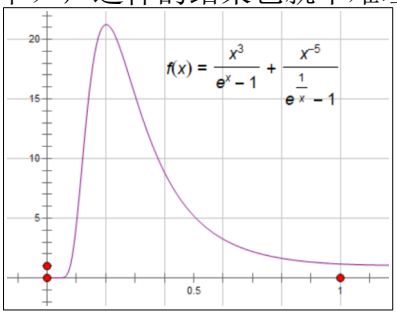

图 2 第 2 题 变换后的被积函数图像（由“几何画板”绘制）

2. 5）两点复合 Gauss-Legendre 算法 结果如下表。

步长	10^{-1}	10^{-2}	10^{-3}	10^{-5}	10^{-7}
用时(ms)	0.000	0.000	0.000	17.950	1864.918
误差	2.03×10^{-2}	-9.84×10^{-11}	-1.60×10^{-14}	1.33×10^{-14}	4.65×10^{-13}

表 8 第 2 题 两点复合 Gauss-Legendre 算法

从表中的数据可以看到，作为两点复合 Gauss-Legendre 算法展现出了极快的收敛性：步长取 10^{-3}，用时小于 5×10^{-4}ms，绝对误差就能达到 10^{-14} 数量级。

综上，两点复合 Gauss-Legendre 算法是在第 2 题中性能表现最好的算法。

附录 完整代码

由于报告长度和排版的限制，完整代码放在了 github 上，还请助教老师见谅。已设置超链接跳转：