数值分析第二次上机试验报告
龙贝格积分法

吴天阳

2204210460

59

西安交通大学

2022 年 4 月 17 日

目录

1 问题描述

使用龙贝格积分法计算如下四个积分，使计算结果尽可能准确。

$$\int_0^1 \frac{1}{1+x}\,dx; \quad (2)\int_0^1 \frac{\ln(1+x)}{1+x^2}\,dx; \quad (3)\int_0^1 \frac{\ln(1+x)}{x}\,dx; \quad (4)\int_0^{\pi/2} \frac{\sin x}{x}\,dx.$$

2 算法实现

直接利用 Romberg 积分公式，有如下递推式

$$\begin{aligned}
T_1 &= \frac{b-a}{2}[f(a)+f(b)], \\
T_{2^i} &= \frac{1}{2}T_{2^{i-1}} + \frac{b-a}{2^i}\sum_{k=0}^{2^i-1} f(a+(2k+1)\frac{b-1}{2^i}) \qquad (i=1,2,\cdots) \\
S_{2^i} &= T_{2^{i+1}} + \frac{1}{4-1}(T_{2^{i+1}} - T_{2^k}) \\
C_{2^i} &= S_{2^{i+1}} + \frac{1}{4^2-1}(S_{2^{i+1}} - S_{2^k}) \qquad (i=0,1,2,\cdots) \\
R_{2^i} &= C_{2^{i+1}} + \frac{1}{4^3-1}(C_{2^{i+1}} - C_{2^k})
\end{aligned}$$

使用 Python 的列表实现上述递推过程，构造如下列表:

T_{2^k}	S_{2^k}	C_{2^k}	R_{2^k}
T_1			
T_2	S_1		
T_{2^2}	S_2	C_1	
T_{2^3}	S_{2^2}	C_2	R_1
T_{2^4}	S_{2^3}	C_{2^2}	R_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$

主要算法部分代码:

```
upper = 20 # 计算次数的上界
table = [[(b-a)/2 * (fun(a) + fun(b))]] # T1初始化
for i in range(1, upper):
    tmp = 0
    for k in range(np.power(2, i-1)):
        tmp += fun(a + (2*k+1) * (b-a) / np.power(2, i))
    tmp *= (b - a) / np.power(2, i)
    table.append([1/2 * table[i-1][0] + tmp]) # 计算T_2^i
    for j in range(1, min(i, 3) + 1): # 递归求值
        table[i].append(table[i][j-1] + (table[i][j-1]-table[
            i-1][j-1]) / (np.power(4, j)-1))
    if i >= 4 and np.abs(table[i][3] - table[i - 1][3]) <=
        eps: # 达到精度要求
        break
return table[i][3]
```

3 试验结果

由于 Python 的浮点型最多只有 16 位有效数字，所以计算结果误差最小只能达到 10^{-15}。

3.1 积分一

截断误差为 10^{-15}，$\int_0^1 \frac{1}{1+x}dx \approx 0.6\,931\,471\,805\,599\,452$，下表列出了前 7 次的计算结果:

$T_{2^k}(k=0,1,\cdots,6)$	S_{2^k}	C_{2^k}	R_{2^k}
0.750 000 000 000			
0.708 333 333 333	0.694 444 444 444		
0.697 023 809 524	0.693 253 968 254	0.693 174 603 175	
0.694 121 850 372	0.693 154 530 655	0.693 147 901 481	0.693 147 477 645
0.693 391 202 208	0.693 147 652 819	0.693 147 194 297	0.693 147 183 072
0.693 208 208 269	0.693 147 210 290	0.693 147 180 788	0.693 147 180 573
0.693 162 438 883	0.693 147 182 421	0.693 147 180 564	0.693 147 180 560

3.2 积分二

截断误差为 10^{-15}，$\int_0^1 \frac{\ln(1+x)}{1+x^2}dx \approx 0.27\,219\,826\,128\,795\,027$，下表列出了前 7 次的计算结果：

$T_{2^k}(k=0,1,\cdots,6)$	S_{2^k}	C_{2^k}	R_{2^k}
0.173 286 795 140			
0.248 829 440 813	0.274 010 322 704		
0.266 457 611 491	0.272 333 668 384	0.272 221 891 429	
0.270 768 638 296	0.272 205 647 230	0.272 197 112 487	0.272 196 719 170
0.271 841 192 282	0.272 198 710 278	0.272 198 247 814	0.272 198 265 835
0.272 109 014 944	0.272 198 289 164	0.272 198 261 090	0.272 198 261 301
0.272 175 951 006	0.272 198 263 027	0.272 198 261 285	0.272 198 261 288

3.3 积分三

由于收敛速度较慢，截断误差只能为 10^{-6}，$\int_0^1 \frac{\ln(1+x)}{x}dx \approx 0.822\,466$，下表列出了前 20 次的计算结果：

$T_{2^k}(k=0,1,\cdots,19)$	S_{2^k}	C_{2^k}	R_{2^k}
0.346 573 590 280			
0.578 751 903 248	0.656 144 674 238		
0.699 058 098 917	0.739 160 164 140	0.744 694 530 133	
0.760 366 129 038	0.780 802 139 078	0.783 578 270 740	0.784 195 472 972
0.791 316 892 026	0.801 633 813 022	0.803 022 591 285	0.803 331 231 293
0.806 867 003 395	0.812 050 373 851	0.812 744 811 240	0.812 899 132 191
0.814 660 776 250	0.817 258 700 535	0.817 605 922 314	0.817 683 082 807
0.818 562 344 151	0.819 862 866 785	0.820 036 477 869	0.820 075 058 115
0.820 514 298 607	0.821 164 950 093	0.821 251 755 646	0.821 271 045 770
0.821 490 568 470	0.821 815 991 758	0.821 859 394 535	0.821 869 039 597
0.821 978 776 561	0.822 141 512 591	0.822 163 213 980	0.822 168 036 511
0.822 222 898 896	0.822 304 273 007	0.822 315 123 702	0.822 317 534 967
0.822 344 964 636	0.822 385 653 216	0.822 391 078 563	0.822 392 284 196
0.822 405 998 649	0.822 426 343 320	0.822 429 055 994	0.822 429 658 810
0.822 436 515 941	0.822 446 688 372	0.822 448 044 709	0.822 448 346 117
0.822 451 774 659	0.822 456 860 898	0.822 457 539 066	0.822 457 689 771
0.822 459 404 036	0.822 461 947 161	0.822 462 286 245	0.822 462 361 597
0.822 463 218 728	0.822 464 490 293	0.822 464 659 835	0.822 464 697 511
0.822 465 126 076	0.822 465 761 858	0.822 465 846 629	0.822 465 865 467
0.822 466 079 750	0.822 466 397 641	0.822 466 440 027	0.822 466 449 446

3.4 积分四

截断误差为 10^{-15}，$\int_0^{\pi/2} \frac{\sin x}{x}\, dx \approx 1.3\,707\,621\,681\,544\,881$，下表列出了前 7 次的计算结果：

$T_{2^k}(k=0,1,\cdots,6)$	S_{2^k}	C_{2^k}	R_{2^k}
1.285 398 163 397			
1.349 805 862 885	1.371 275 096 048		
1.365 546 207 978	1.370 792 989 676	1.370 760 849 251	
1.369 459 609 054	1.370 764 076 079	1.370 762 148 506	1.370 762 169 129
1.370 436 617 600	1.370 762 287 115	1.370 762 167 851	1.370 762 168 158
1.370 680 786 089	1.370 762 175 585	1.370 762 168 150	1.370 762 168 155
1.370 741 822 986	1.370 762 168 619	1.370 762 168 154	1.370 762 168 154

4 总结

本次上机实验更换编程语言为 Python，Python 的好处主要在于数据处理功能强大，以上的表格是使用 tabulate 包直接输出的 LaTex 代码，十分方便，而且 Python 代码更加简洁易懂。缺点在于 Python 不能做带有自变量的函数运算，所以遇到需要求函数确切表达式的问题还是要用 MATLAB。

A 代码

Python 中所用到的包为：numpy 和 tabulate。

可供复制代码链接[1]，这里也附上完整代码：

```
# coding=utf-8
import numpy as np
from tabulate import tabulate

def Romberg(fun, a, b, eps):
    """
    Romberg积分法
    :param fun: 一维被积函数
```

[1]https://paste.ubuntu.com/p/vc97fBVmZs/

```python
    :param a: 积分区间左端点
    :param b: 积分区间右端点
    :param eps: 截断误差
    :return: 积分结果
    """
    upper = 20 # 计算次数的上界
    table = [[(b-a)/2 * (fun(a) + fun(b))]] # T1初始化
    for i in range(1, upper):
        tmp = 0
        for k in range(np.power(2, i-1)):
            tmp += fun(a + (2*k+1) * (b-a) / np.power(2, i))
        tmp *= (b - a) / np.power(2, i)
        table.append([1/2 * table[i-1][0] + tmp]) # 计算T_2^i
        for j in range(1, min(i, 3) + 1): # 递归求值
            table[i].append(table[i][j-1] + (table[i][j-1]-table[
                i-1][j-1]) / (np.power(4, j)-1))
        if i >= 4 and np.abs(table[i][3] - table[i - 1][3]) <=
            eps: # 达到精度要求
            break
    print(tabulate(table, tablefmt='latex', floatfmt=".12f")) #
        以LaTex格式输出表格
    return table[i][3]


if __name__ == '__main__':
    print(Romberg(lambda x: 1/(1+x), 0, 1, 1e-15))
    # print(Romberg(lambda x: np.log(1+x)/(1+np.power(x,2)), 0,
        1, 1e-15))
    # print(Romberg(lambda x: np.log(1+x)/x, 1e-30, 1, 1e-10))
    # print(Romberg(lambda x: np.sin(x)/x, 1e-30, np.pi/2, 1e
        -15))
```