#Exercise 4.1

#15.10.14

#author: chuanlu

import numpy as np

import scipy as sp

import math

from scipy import integrate

divmax = 100000

def composite\_trapezoidal\_formula(a, b, n, func):

x\_series = [a + (b - a)/n \* i for i in range(n)]

f\_series = [func(x) for x in x\_series]

return (b - a)/n \* (sum(f\_series) \* 2 - f\_series[0] - f\_series[-1]) / 2

def composite\_simpson\_formula(a, b, n, func):

x\_series1 = [a + (b - a)/n \* i for i in range(n)]

x\_series2 = [x + (b - a)/(2 \* n) for x in x\_series1[:-1]]

f\_series1 = [func(x) for x in x\_series1]

f\_series2 = [func(x) for x in x\_series2]

return (b - a)/n \*(sum(f\_series1) \* 2 + sum(f\_series2) \* 4 - f\_series1[0] - f\_series1[-1]) / 6

def \_difftrap(func, interval, numtrap):

''' 求二分numtrap次后的梯形值，其中Interval为积分区间，func为被积函数'''

if numtrap == 1:

return (func(interval[1]) + func(interval[0])) \* 0.5

else:

num\_to\_sum = int(numtrap/2)

h = float(interval[1] - interval[0])/num\_to\_sum

start = interval[0] + 0.5 \* h

points = [start + h \* i for i in range(num\_to\_sum)]

s = sum([func(x) for x in points], 0)

return s

def \_romberg\_diff(b, c, k):

temp = 4.0 \*\* k

return (temp \* c - b)/(temp - 1.0)

def romberg\_formula(a, b, delta, func):

n = 1

interval = [a, b]

int\_range = b - a

ord\_sum = \_difftrap(func, interval, n)

result = int\_range \* ord\_sum

result\_mat = [[result]]

err = float('-inf')

for i in range(1, divmax + 1):

n \*= 2

ord\_sum += \_difftrap(func, interval, n)

result\_mat.append([])

result\_mat[i].append(int\_range \* ord\_sum / n)

for k in range(i):

result\_mat[i].append(\_romberg\_diff(result\_mat[i-1][k], result\_mat[i][k], k+1))

result = result\_mat[i][i]

lastresult = result\_mat[i-1][i-1]

err = abs(result - lastresult)

if err < delta:

break

return result

def self\_adapted\_simpson\_formula(func, a, b, delta):

count = 1

return simpson\_iter(func, a, b, delta, simpson(func, a, b), count)

def simpson\_iter(func, a, b, delta, prev\_sum, count):

mid = (b + a) / 2

left\_sum = simpson(func, a, mid)

right\_sum = simpson(func, mid, b)

if abs(prev\_sum - left\_sum - right\_sum) < 15 \* delta \* (0.5 \*\* count):

return left\_sum + right\_sum

else:

count += 1

return simpson\_iter(func, a, mid, delta, left\_sum, count) + simpson\_iter(func, mid, b, delta, right\_sum, count)

def simpson(func, low, high):

mid = (low + high) / 2

return (func(low) + 4\*func(mid) + func(high))\*(high - low)/6

def main():

a = 1e-300

b = 1

function = lambda x: math.sqrt(x) \* math.log(x)

exact\_value = list(integrate.quad(function, 0, 1))

print("exact\_value:", exact\_value[0])

for n in range(1, 10):

print(n)

trap\_delta = abs(composite\_trapezoidal\_formula(a, b, n, function) - exact\_value[0])

simp\_delta = abs(composite\_simpson\_formula(a, b, n, function) - exact\_value[0])

romberg\_delta = abs(romberg\_formula(a, b, 1/(2 \*\* n), function) - exact\_value[0])

print(trap\_delta)

print(simp\_delta)

print(romberg\_delta)

# for i in range(1000, 10000000):

self\_adapted\_simp\_value = self\_adapted\_simpson\_formula(function, a, b, 1e-8)

# if abs(self\_adapted\_simp\_value - exact\_value[0]) < 1e-4:

# print("i:", i)

# break

print(self\_adapted\_simp\_value)

if \_\_name\_\_ == '\_\_main\_\_':

main()

这是修改后的程序，解决了第三小题原有的bug。

程序运行结果如下：

exact\_value: -0.44444444444444425

1

0.44444444444444425

0.44444444444444425

0.11769172995492849

2

0.32191217651087584

0.17255129494660637

0.04405846562954807

3

0.1778395985026217

0.08410687091515556

0.04405846562954807

4

0.1174828835013148

0.05089110737834268

0.017343785594182592

5

0.08549398154308585

0.0346437376950125

0.017343785594182592

6

0.06608197586077313

0.02539073909398748

0.006840607651880215

7

0.05322156643344139

0.01957289272640006

0.002677605290685392

8

0.04416110369961457

0.01565134028107007

0.001039147942263674

9

0.03748061585725054

0.012868045016491525

0.001039147942263674

-0.44444444274139394

[Finished in 0.3s]