Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра обчислювальної техніки

Лабораторна робота № 7

з дисципліни «Чисельні методи»

на тему

**“Чисельне інтегрування функцій”**

Виконав:

студент гр. ІП-93

Завальнюк Максим

Викладач:

доц. Рибачук Л.В.

Київ – 2021

### Зміст

[Зміст 2](#_Toc70277394)

[1 Постановка задачі 3](#_Toc70277395)

[2 Розв’язок 4](#_Toc70277396)

[Допрограмовий етап 4](#_Toc70277397)

[Програмовий етап 7](#_Toc70277398)

[3 Розв’язок за допомогою NumPy 8](#_Toc70277399)

[4 Лістинг програми 9](#_Toc70277400)

### 1 Постановка задачі

Реалізувати програму, яка обчислює інтеграл за допомогою формули трапеції або Сімпсона, в залежності від варіанту. Точність обчислень має бути 0,0001. Мінімальну кількість кроків визначити по формулі. Оцінити похибку результату.

Реалізувати програму, яка обчислює інтеграл за допомогою квадратурної формули Гауса. Оцінити похибку результату.

Обчислити визначений інтеграл у NumPy та порівняти реальну похибку кожного метода (це різниця між розрахованим значенням інтегралу і значенням у NumPy) з аналітичною похибкою кожного методу. Реальна похибка має бути не більша ніж аналітична.

### 2 Розв’язок

########## Trapezium method ##########

N = 48

Analytical fault = 9.614646498891427e-06

Real fault = 7.733035472773375e-06

True

Result: 0.13497557143224748

########## Simpson method ##########

N = 5

Analytical fault = 7.596878224902542e-06

Real fault = 4.277850243483705e-08

True

Result: 0.13496788117527714

########## Gaussian method ##########

N = 3

Analytical fault = 1.525664313758989e-06

Real fault = 4.302506664555228e-07

True

Result: 0.13496740814610825

Можемо зробити висновки: оскільки порядок похибки формули трапецій , а формули Сімпсона - , то очевидно, що у Сімпсона краща точність, але менша, ніж у квадратурної формули Гауса

### 3 Розв’язок за допомогою NumPy

Нижче наведено розв’язок системи у NumPy:

np\_integrate = integrate.quad(main\_func, a, b)[0]

Результат:

########## Result from NumPy ##########

0.1349678383967747

### 4 Лістинг програми

from scipy import integrate  
import scipy.optimize as opt  
from math import cos, sin, factorial  
from string import Template  
  
a, b = 0.8, 1.7  
epsilon = 10 \*\* (-5)  
template = Template(**'#'** \* 10 + **' $string '** + **'#'** \* 10)  
  
coefficients = {  
 1: {**'x1'**: 0.5, **'c1'**: 2},  
 2: {**'x1'**: -0.577350, **'x2'**: 0.577350, **'c1'**: 1, **'c2'**: 1},  
 3: {**'x1'**: -0.774597, **'x2'**: 0, **'x3'**: 0.774597, **'c1'**: 0.555555, **'c2'**: 0.888889, **'c3'**: 0.555555},  
 4: {**'x1'**: -0.861136, **'x2'**: -0.339981, **'x3'**: 0.339981, **'x4'**: 0.861136, **'c1'**: 0.347855, **'c2'**: 0.652145,  
 **'c3'**: 0.652145, **'c4'**: 0.347855},  
 5: {**'x1'**: -0.906180, **'x2'**: -0.538470, **'x3'**: 0, **'x4'**: 0.538470, **'x5'**: 0.906180, **'c1'**: 0.236927, **'c2'**: 0.478629,  
 **'c3'**: 0.568889, **'c4'**: 0.478629, **'c5'**: 0.236927},  
 6: {**'x1'**: -0.932470, **'x2'**: -0.661210, **'x3'**: -0.238620, **'x4'**: 0.238620, **'x5'**: 0.661210, **'x6'**: 0.932470,  
 **'c1'**: 0.171324, **'c2'**: 0.360761,  
 **'c3'**: 0.467914, **'c4'**: 0.467914, **'c5'**: 0.360761, **'c6'**: 0.171324},  
 7: {**'x1'**: -0.949108, **'x2'**: -0.741531, **'x3'**: -0.405845, **'x4'**: 0, **'x5'**: 0.405845, **'x6'**: 0.741531, **'x7'**: 0.949108,  
 **'c1'**: 0.129485, **'c2'**: 0.279705,  
 **'c3'**: 0.381830, **'c4'**: 0.417960, **'c5'**: 0.381830, **'c6'**: 0.279705, **'c7'**: 0.129485},  
 8: {**'x1'**: -0.960290, **'x2'**: -0.796666, **'x3'**: -0.525532, **'x4'**: -0.183434, **'x5'**: 0.183434, **'x6'**: 0.525532,  
 **'x7'**: 0.796666, **'x8'**: 0.960290,  
 **'c1'**: 0.101228, **'c2'**: 0.222381,  
 **'c3'**: 0.313707, **'c4'**: 0.362684, **'c5'**: 0.362684, **'c6'**: 0.313707, **'c7'**: 0.222381, **'c8'**: 0.101228},  
}  
  
  
def main\_func(x):  
 return cos(x) / (x + 1)  
  
  
def main\_func\_reverse(t):  
 x = ((a + b) / 2) + ((t \* (b - a)) / 2)  
 return cos(x) / (x + 1)  
  
  
def main\_func\_second(x):  
 return (-sin(x) / (x + 1)) - (cos(x) / (x + 1) \*\* 2)  
  
  
def main\_func\_fourth(x):  
 return (cos(x) - (4 \* sin(x) / (x + 1)) - (12 \* cos(x) / (x + 1) \*\* 2) + (24 \* sin(x) / (x + 1) \*\* 3) + (  
 24 \* cos(x) / (x + 1) \*\* 4)) / (x + 1)  
  
  
def main\_func\_sixth(x):  
 return (-cos(x) + (6 \* sin(x) / (x + 1)) + (30 \* cos(x) / (x + 1) \*\* 2) - (120 \* sin(x) / (x + 1) \*\* 3) - (  
 360 \* cos(x) / (x + 1) \*\* 4) + (720 \* sin(x) / (x + 1) \*\* 5) + (720 \* cos(x) / (x + 1) \*\* 6)) / (x + 1)  
  
  
def main\_function\_eight(x):  
 return (cos(x) - (8 \* sin(x) / (x + 1)) - (56 \* cos(x) / (x + 1) \*\* 2) + (336 \* sin(x) / (x + 1) \*\* 3) + (  
 1680 \* cos(x) / (x + 1) \*\* 4) - (6720 \* sin(x) / (x + 1) \*\* 5) - (20160 \* cos(x) / (x + 1) \*\* 6) + (  
 40320 \* sin(x) / (x + 1) \*\* 7) + (40320 \* cos(x) / (x + 1) \*\* 8)) / (x + 1)  
  
  
def main\_func\_tenth(x):  
 return (-cos(x) + (10 \* sin(x) / (x + 1)) + (90 \* cos(x) / (x + 1) \*\* 2) - (720 \* sin(x) / (x + 1) \*\* 3) - (  
 5040 \* cos(x) / (x + 1) \*\* 4) + (30240 \* sin(x) / (x + 1) \*\* 5) + (151200 \* cos(x) / (x + 1) \*\* 6) - (  
 604800 \* sin(x) / (x + 1) \*\* 7) - (1814400 \* cos(x) / (x + 1) \*\* 8) + (  
 3628800 \* sin(x) / (x + 1) \*\* 9) + (3628800 \* cos(x) / (x + 1) \*\* 10)) / (x + 1)  
  
  
def main\_function\_twelves(x):  
 return (cos(x) - (12 \* sin(x) / (x + 1)) - (132 \* cos(x) / (x + 1) \*\* 2) + (1320 \* sin(x) / (x + 1) \*\* 3) + (  
 1180 \* cos(x) / (x + 1) \*\* 4) - (95040 \* sin(x) / (x + 1) \*\* 5) - (665280 \* cos(x) / (x + 1) \*\* 6) + (  
 3991680 \* sin(x) / (x + 1) \*\* 7) + (19958400 \* cos(x) / (x + 1) \*\* 8) - (  
 79833600 \* sin(x) / (x + 1) \*\* 9) - (239500800 \* cos(x) / (x + 1) \*\* 10) + (  
 479001600 \* sin(x) / (x + 1) \*\* 11) + (479001600 \* cos(x) / (x + 1) \*\* 12)) / (x + 1)  
  
  
def main\_func\_fourteenth(x):  
 return (-cos(x) + (14 \* sin(x) / (x + 1)) + (182 \* cos(x) / (x + 1) \*\* 2) - (2184 \* sin(x) / (x + 1) \*\* 3) - (  
 24024 \* cos(x) / (x + 1) \*\* 4) + (240240 \* sin(x) / (x + 1) \*\* 5) + (2162160 \* cos(x) / (x + 1) \*\* 6) - (  
 17297280 \* sin(x) / (x + 1) \*\* 7) - (121080960 \* cos(x) / (x + 1) \*\* 8) + (  
 726485760 \* sin(x) / (x + 1) \*\* 9) + (3632428800 \* cos(x) / (x + 1) \*\* 10) - (  
 14529715200 \* sin(x) / (x + 1) \*\* 11) - (43589145600 \* cos(x) / (x + 1) \*\* 12) + (  
 87178291200 \* sin(x) / (x + 1) \*\* 13) + (87178291200 \* cos(x) / (x + 1) \*\* 14)) / (x + 1)  
  
  
def main\_function\_sixteenth(x):  
 return (cos(x) - (16 \* sin(x) / (x + 1)) - (240 \* cos(x) / (x + 1) \*\* 2) + (3360 \* sin(x) / (x + 1) \*\* 3) + (  
 43680 \* cos(x) / (x + 1) \*\* 4) - (524160 \* sin(x) / (x + 1) \*\* 5) - (  
 5765760 \* cos(x) / (x + 1) \*\* 6) + (57657600 \* sin(x) / (x + 1) \*\* 7) + (  
 518918400 \* cos(x) / (x + 1) \*\* 8) - (4151347200 \* sin(x) / (x + 1) \*\* 9) - (  
 29059430400 \* cos(x) / (x + 1) \*\* 10) + (174356582400 \* sin(x) / (x + 1) \*\* 11) + (  
 871782912000 \* cos(x) / (x + 1) \*\* 12) - (3487131648000 \* cos(x) / (x + 1) \*\* 13) + (  
 10461394944000 \* cos(x) / (x + 1) \*\* 14) + (20922789888000 \* cos(x) / (x + 1) \*\* 15) + (  
 20922789888000 \* cos(x) / (x + 1) \*\* 16)) / (x + 1)  
  
  
np\_integrate = integrate.quad(main\_func, a, b)[0]  
  
  
def trapezium\_method(a\_limit, b\_limit):  
 parts, analytical\_fault = trapezium\_method\_fault(a\_limit, b\_limit)  
 result = (main\_func(a\_limit) + main\_func(b\_limit)) / 2  
 h = (b\_limit - a\_limit) / parts  
 print(**f'N =** {parts}**'**)  
 print(**f'Analytical fault =** {analytical\_fault}**'**)  
 index = a\_limit + h  
 while index < b\_limit:  
 result += main\_func(index)  
 index += h  
 real\_fault = get\_fault(result \* h, np\_integrate)  
 print(**f'Real fault =** {real\_fault}**'**)  
 print(real\_fault < analytical\_fault)  
 return result \* h  
  
  
def simpson\_method(a\_limit, b\_limit):  
 parts, analytical\_fault = simpson\_method\_fault(a\_limit, b\_limit)  
 result = main\_func(a\_limit) + main\_func(b\_limit)  
 width = (b\_limit - a\_limit) / (2 \* parts)  
 print(**f'N =** {parts}**'**)  
 print(**f'Analytical fault =** {analytical\_fault}**'**)  
 firstPart = 0  
 secondPart = 0  
 for i in range(1, parts):  
 firstPart += main\_func(2 \* width \* i + a\_limit) \* 2  
 result += firstPart  
 for i in range(1, parts + 1):  
 secondPart += main\_func(width \* (2 \* i - 1) + a\_limit) \* 4  
 result += secondPart  
 real\_fault = get\_fault(result \* width / 3, np\_integrate)  
 print(**f'Real fault =** {real\_fault}**'**)  
 print(real\_fault < analytical\_fault)  
 return result \* width / 3  
  
  
def gaussian\_method(a\_limit, b\_limit):  
 parts, analytical\_fault = gaussian\_method\_fault(a\_limit, b\_limit)  
 print(**f'N =** {parts}**'**)  
 print(**f'Analytical fault =** {analytical\_fault}**'**)  
 result = 0  
 for index in range(parts):  
 result += coefficients[parts][**f'c**{index + 1}**'**] \* main\_func\_reverse(coefficients[parts][**f'x**{index + 1}**'**])  
 real\_fault = get\_fault(result \* ((b\_limit - a\_limit) / 2), np\_integrate)  
 print(**f'Real fault =** {real\_fault}**'**)  
 print(real\_fault < analytical\_fault)  
 return result \* ((b\_limit - a\_limit) / 2)  
  
  
def trapezium\_method\_fault(a\_limit, b\_limit):  
 n = 1  
 M = opt.fmin\_l\_bfgs\_b(lambda x: -main\_func\_second(x), 1.0, bounds=[(a\_limit, b\_limit)], approx\_grad=True)  
 fault = abs(M[1][0]) \* ((b\_limit - a\_limit) \*\* 3) / (12 \* n \*\* 2)  
 while epsilon < fault:  
 fault = abs(M[1][0]) \* ((b\_limit - a\_limit) \*\* 3) / (12 \* n \*\* 2)  
 n += 1  
 return n, fault  
  
  
def simpson\_method\_fault(a\_limit, b\_limit):  
 n = 1  
 M = opt.fmin\_l\_bfgs\_b(lambda x: -main\_func\_fourth(x), 1.0, bounds=[(a\_limit, b\_limit)], approx\_grad=True)  
 fault = abs(M[1][0]) \* ((b\_limit - a\_limit) \*\* 5) / (180 \* n \*\* 4)  
 while epsilon < fault:  
 fault = abs(M[1][0]) \* ((b\_limit - a\_limit) \*\* 5) / (180 \* n \*\* 4)  
 n += 1  
 return n, fault  
  
  
def gaussian\_method\_fault(a\_limit, b\_limit):  
 n = 2  
 func\_list = [main\_func\_fourth, main\_func\_sixth, main\_function\_eight, main\_func\_tenth, main\_function\_twelves,  
 main\_func\_fourteenth, main\_function\_sixteenth]  
 fault = 1  
 for func in func\_list:  
 M = opt.fmin\_l\_bfgs\_b(lambda x: -func(x), 1.0, bounds=[(a\_limit, b\_limit)], approx\_grad=True)  
 fault = abs(M[1][0]) \* (((factorial(n)) \*\* 4) \* (b\_limit - a\_limit) \*\* (2 \* n + 1)) / (  
 (2 \* n + 1) \* (factorial(2 \* n)) \*\* 3)  
 if epsilon > fault:  
 break  
 n += 1  
 return n, fault  
  
  
def get\_fault(my\_result, real\_result):  
 return abs(real\_result - my\_result)  
  
  
print(template.substitute(string=**'Trapezium method'**))  
print(**f'Result:** {trapezium\_method(a, b)}**'**)  
print(template.substitute(string=**'Simpson method'**))  
print(**f'Result:** {simpson\_method(a, b)}**'**)  
print(template.substitute(string=**'Gaussian method'**))  
print(**f'Result:** {gaussian\_method(a, b)}**'**)  
print(template.substitute(string=**'Result from NumPy'**))  
print(np\_integrate)