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
Домінський Валентин

Викладач:
доц. Рибачук Л.В.

Київ – 2021

Зміст

1 Постановка задачі

1. 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 за формулою (1.7). Оцінити похибку результату.

2. Реалізувати програму, яка обчислює інтеграл за допомогою квадратурної формули Гауса (для всіх варіантів). Оцінити похибку результату.

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

2 Розв’язок

Вивід програми:

```
Simpson Method:
Result = 0.5301734025593877
 Difference = 0.000071715111221
 N = 6

Gauss Method:
Result = 0.5301640424954107
 Difference = 0.000099019003464
 N = 3
```

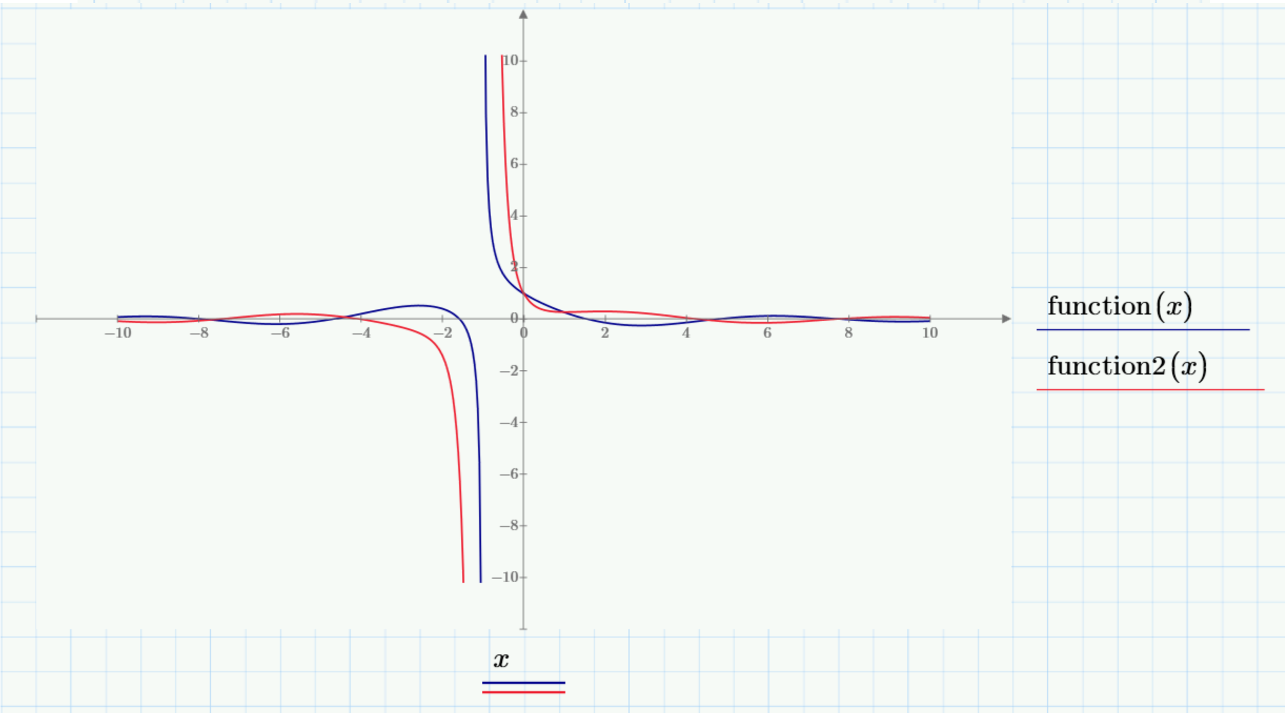
3 Розв’язок у Mathcad

Нижче наведено розв’язок у Mathcad

Початкові значення

$\mathrm{variant} := 9 \qquad \mathrm{function}(x) := \dfrac{\cos(x)}{x+1} \qquad a := 0.1 \qquad b := 1.1$

$$\mathrm{function2}(x) := \frac{d^2}{dx^2}\mathrm{function}(x)$$



Обчислюю похибки

$$integralValueMathcad := \int_a^b function(x)\, dx = 0.530172878347676$$

$$integralValueProgramSimpson := 0.5301734025593877$$

$$integralValueProgramGauss := 0.5301640424954107$$

$$MathcadAndSimpsonDiff := |integralValueMathcad - integralValueProgramSimpson| = 0.000000524211712$$

$$MathcadAndGaussDiff := |integralValueMathcad - integralValueProgramGauss| = 0.000008835852265$$

$$integralAnalyzeProgSimpsonDiff := 0.000071715111221$$

$$integralAnalyzeProgGaussDiff := 0.000099019003464$$

```
IsRealSmallerSimpson := if integralAnalyzeProgSimpsonDiff > MathcadAndSimpsonDiff
                            return 1
                        else
                            return 0
```

$$IsRealSmallerSimpson = 1$$

```
IsRealSmallerGauss := if integralAnalyzeProgGaussDiff > MathcadAndGaussDiff
                          return 1
                      else
                          return 0
```

$$IsRealSmallerGauss = 1$$

У технічних розрахунках точність вимірювань характеризують відносною похибкою. Результат вважають гарним, якщо відносна похибка не перевищує 0,1 %. Отже Наш результат є гарним

4 Лістинг програми

Lab7.py

```
# region Starting Values
import numpy
import scipy
numpy.set_printoptions(suppress=True,
  formatter={'float_kind':'{:0.2f}'.format})
import scipy.optimize
from math import factorial, sin, cos

epsilonValue = 0.0001
leftBoard = 0.1
rightBoard = 1.1
defaultNforSimpson = 1
defaultNforGauss = 2
rounding = 5

coeffcients =  [
            [ 0.5, 2],
            [-0.577350, 0.577350, 1, 1],
            [-0.774597, 0, 0.774597, 0.555555, 0.888889, 0.555555],
            [-0.861136, -0.339981, 0.339981, 0.861136, 0.347855, 0.652145, 0.652145, 0.347855],
            [-0.906180, -0.538470, 0, 0.538470, 0.906180, 0.236927, 0.478629, 0.568889, 0.478629, 0.236927],
            [-0.932470, -0.661210, -0.238620, 0.238620, 0.661210, 0.932470, 0.171324, 0.360761, 0.467914,
0.467914, 0.360761, 0.171324],
            [-0.949108, -0.741531, -0.405845, 0, 0.405845, 0.741531, 0.949108, 0.129485, 0.279705, 0.381830,
0.417960, 0.381830, 0.279705, 0.129485],
            [ -0.960290, -0.796666, -0.525532, -0.183434, 0.183434, 0.525532, 0.796666, 0.960290, 0.101228,
0.222381, 0.313707, 0.362684, 0.362684, 0.313707, 0.222381, 0.101228]
            ]

# endregion Starting Values

#region Default Functions

def MyFunction(x):
   func = cos(x) / (x + 1)
   return func

def ReverseMyFunction(t):
   x = ((t * (rightBoard - leftBoard)) / 2) + ((leftBoard + rightBoard) / 2)
   func = MyFunction(x)
   return func

def MyPrimeFunction(x):
   func = (-sin(x)/(x+1))-(cos(x)/(x+1)**2)
   return func

def MyFourthPrimeFunction(x):
   func = (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)
   return func

def MySixthPrimeFunction(x):
   func=(-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)
   return func

def MyEigthPrimeFunction(x):
   func=(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)
   return func
```

```
def MyTenthPrimeFunction(x):
   func=(-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)
   return func

def MyTwelwethPrimeFunction(x):
   func=(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)
   return func

def MyFourteenthPrimeFunction(x):
   func=(-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)
   return func

def MySixteenthPrimeFunction(x):
   func=(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)
   return func

SpecificFunctionsForTheGauss = [MyFourthPrimeFunction, MySixthPrimeFunction, MyEigthPrimeFunction,
                        MyTenthPrimeFunction, MyTwelwethPrimeFunction,
                        MyFourteenthPrimeFunction, MySixteenthPrimeFunction]

#endregion Default Functions

def Simpson(firstInterval, secondInterval):

   tempValues = [0, 0]
   totalAmount = MyFunction(secondInterval) + MyFunction(firstInterval)
   analyzeDifference, Nvalue = SimpsonDifference(firstInterval, secondInterval, defaultNforSimpson)
   intervalLength = (secondInterval - firstInterval) / (Nvalue * 2)
   for z in range(1, Nvalue + 1):
      tempValues[1] = tempValues[1] + 4 * MyFunction(intervalLength * (z * 2 - 1) + firstInterval)
   totalAmount = totalAmount + tempValues[1]
   for z in range(1, Nvalue):
      tempValues[0] = tempValues[0] + 2 * MyFunction(z * 2 * intervalLength + firstInterval)
   totalAmount = totalAmount + tempValues[0]
   finalResult = totalAmount * intervalLength / 3

   return finalResult, analyzeDifference, Nvalue

def SimpsonDifference(firstInterval, secondInterval, Nvalue):
```

```
    valueForAccuracy = scipy.optimize.fmin_l_bfgs_b(lambda x: -MyFourthPrimeFunction(x),
                                    1.0, bounds=[(firstInterval, secondInterval)], approx_grad=True)
    difference = (((secondInterval - firstInterval) ** 5) * abs(valueForAccuracy[1][0])) / ((Nvalue ** 4) * 180)
    while difference > epsilonValue:
        difference = (((secondInterval - firstInterval) ** 5) * abs(valueForAccuracy[1][0])) / ((Nvalue ** 4) * 180)
        Nvalue = Nvalue + 1

    return difference, Nvalue

def Gauss(firstInterval, secondInterval):

    analyzeDifference, Nvalue = GaussDifference(firstInterval, secondInterval, defaultNforGauss)
    tempResult = 0
    for z in range(Nvalue):
        tempIndex = int(((len(coeffcients[Nvalue])/2)+z)-1)
        tempResult = tempResult + coeffcients[Nvalue-1][tempIndex] * ReverseMyFunction(coeffcients[Nvalue-
1][z])
    finalResult = ((secondInterval - firstInterval) / 2) * tempResult

    return finalResult, analyzeDifference, Nvalue

def GaussDifference(firstInterval, secondInterval, Nvalue):

    for z in range(len(SpecificFunctionsForTheGauss)):
        valueForAccuracy = scipy.optimize.fmin_l_bfgs_b(lambda x: -SpecificFunctionsForTheGauss[z](x),
                                    1.0, bounds=[(firstInterval, secondInterval)], approx_grad=True)
        difference = (((secondInterval-
firstInterval)**((Nvalue+1)*2))*((factorial(Nvalue))**4))*abs(valueForAccuracy[1][0])/(((factorial(2*Nvalue)
)**3)*(2*Nvalue+1))
        if difference < epsilonValue:
            break
        Nvalue = Nvalue + 1

    return difference, Nvalue

print("Simpson Method:")
simpsonResult, simpsonDiff, simpsonN = Simpson(leftBoard, rightBoard)
print("Result =", simpsonResult, "\n", "Difference =", "%-.15f"%(simpsonDiff), "\n", "N =", simpsonN, "\n")

print("Gauss Method:")
gaussResult, gaussDiff, gaussN = Gauss(leftBoard, rightBoard)
print("Result =", gaussResult, "\n", "Difference =", "%-.15f"%(gaussDiff), "\n", "N =", gaussN, "\n")
```

Висновок:

Я навчився використовувати різні методи чисельного інтегрування функцій (методи трапецій, Сімпсона та Гауса. Також можна дійти до висновку, метод Сімпсона – найскладніший серед формул Ньютона – Котеса, але в той же час має меншу к-сть ітерацій та одну з найменших похибок. Також є метод Гауса – найскладніший у лабораторній, оскільки треба мати додаткові дані, але найточніший