|  |  |
| --- | --- |
|  | **Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**  **«Московский государственный технический университет имени Н.Э. Баумана**  **(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

# Лабораторная работа №5

**Тема:** Численное интегрирование

**Студент:** Артемьев И.О.

**Группа:** ИУ7-43Б

**Оценка (баллы): \_\_\_\_\_\_\_\_\_**

**Преподаватель:** Градов В.М.

Москва.

2021 г*.*

## **Задание**

Построить алгоритм и программу для вычисления двукратного интеграла при фикcированном значении параметра 𝜏



Применить метод последовательного интегрирования. По одному направлению использовать формулу Гаусса, а по другому - формулу Симпсона.

## **Входные данные**

Количество узлов сетки 𝑁, 𝑀 ; значение параметра 𝜏 , методы для направлений при последовательном интегрировании.

## **Выходные данные**

Значение интеграла при заданном параметре, график зависимости 𝜖(𝜏 ) в диапазоне 𝜏 = 0.05 − 10.

## **Описание алгоритма**

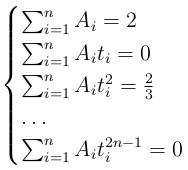
Имеем



Положим



Тогда, имеем систему



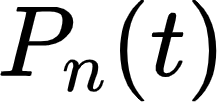
Система нелинейная, найти решение сложно. Для нахождения можно воспользоваться полиномом Лежандра. Формула полинома:

и



Узлами формулы Гаусса являются нули полинома Лежандра можно найти из вышеуказанной системы уравнений.

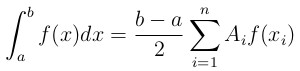
, а



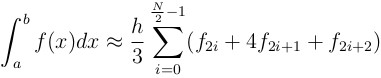
При вычислении интеграла на произвольном интервале [𝑎, 𝑏], для применения квадратурной формулы Гаусса необходимо выполнить преобразование переменной:



В таком случае, получаем конечную формулу для произвольного интервала [𝑎, 𝑏]:



Также, существует квадратнурная формула Симпсона:



Однако, эти методы можно применять и для приближенной оценки двукратных (и не только) интегралов. Рассмотрим интеграл по прямоугольной области:



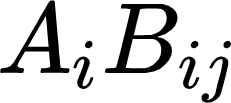
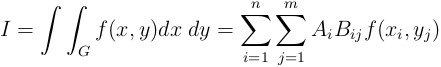
По каждой координате введем сетку узлов. Каждый однократный интеграл вычисляют по квадратурным формулам. Для разных направлений можно использовать квадратурные формулы разных порядков точности, в т.ч. и Гаусса.

Конечная формула:

где

-

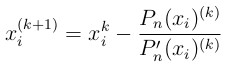
известные постоянные.



## **Результаты работы программы**

1. **Описать алгоритм вычисления n корней полинома Лежандра n-ой степени Pn (x) при реализации формулы Гаусса.**

Все корни полинома лежат на интервале [-1, 1]. При этом стоит заметить, что интервалы [-1, 0] и [0, 1] – симметричны, так что при поиске корней достаточно рассмотреть интервал [0, 1]. Корни полинома можно вычислить итеративно по методу Ньютона

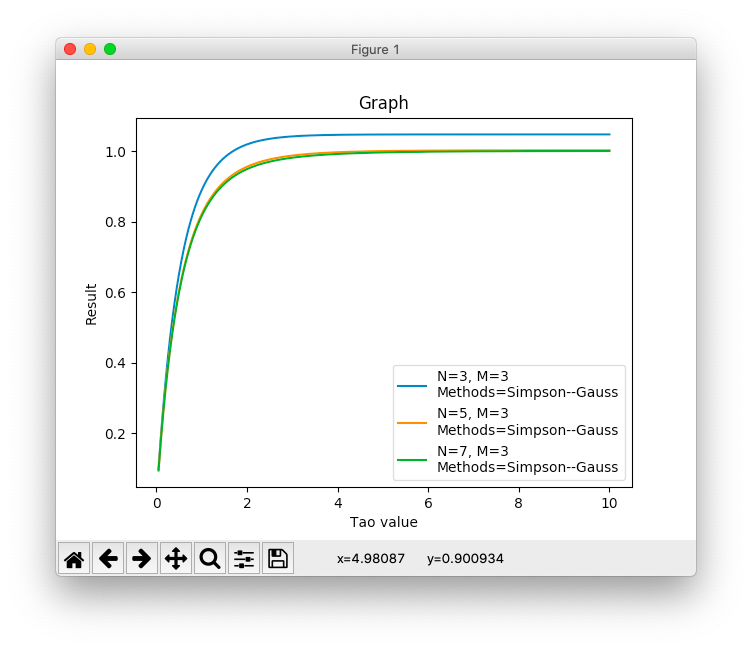


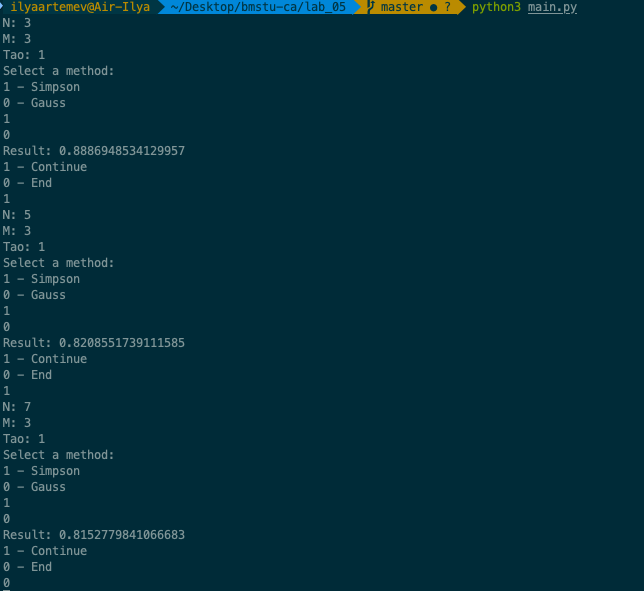
причем начальное приближение для 𝑖-го корня берется по формуле:



1. **Исследовать влияние количества выбираемых узлов сетки по каждому направлению на точность расчетов.**

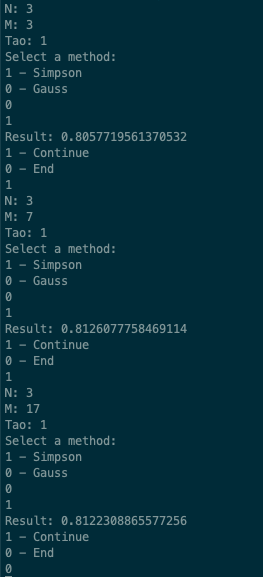
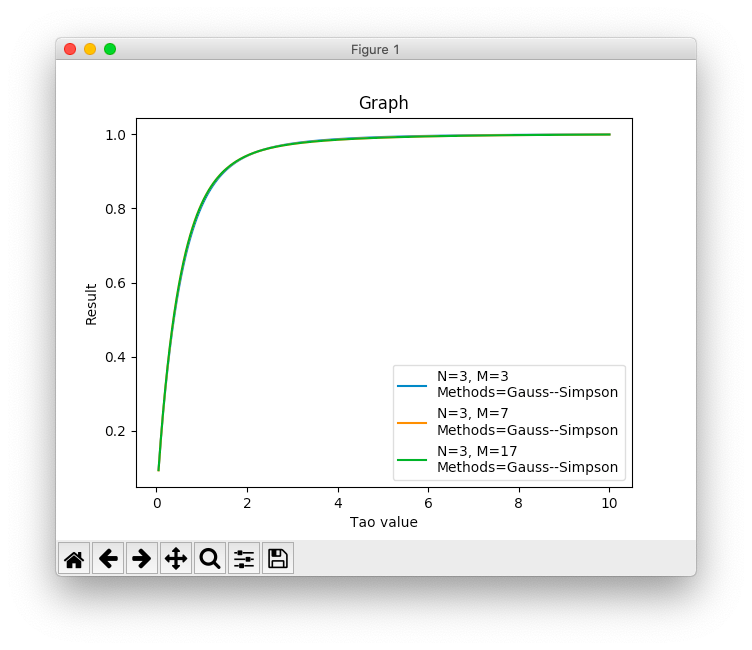
**Исследование для внешнего направления и метода Симпсона:**





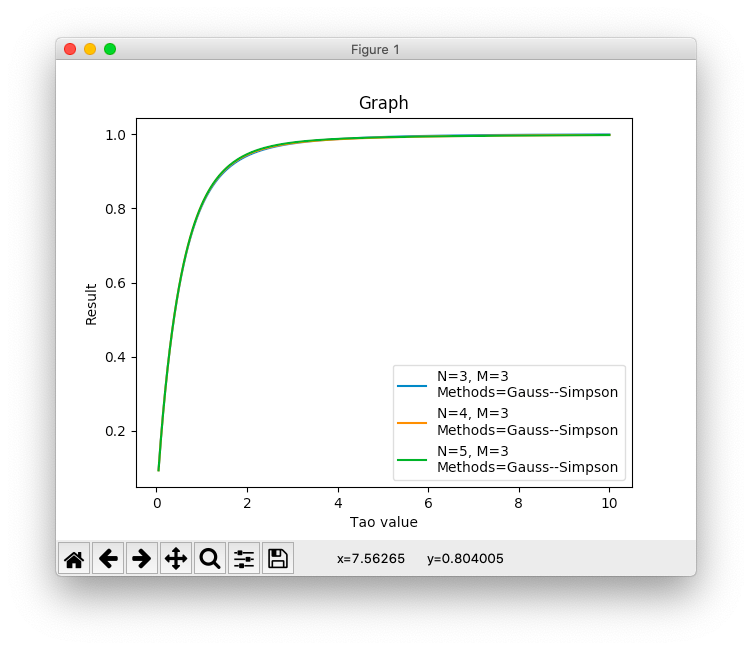
Видно, что при кол-ве узлов = 3 график отклоняется от графиков с кол-вом узлов 5 и 7. Сравнивая с программой расчета интегралов, можно убедиться, что при большом количестве узлов результат точнее.

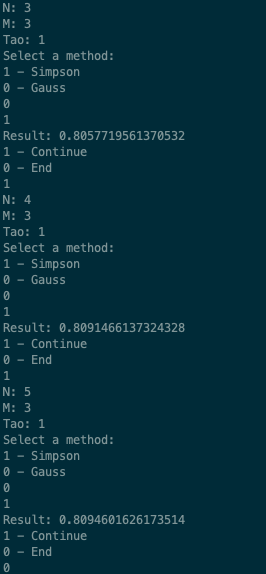
**Исследование для внутреннего направления и метода Симпсона:**



Результаты практически совпадают. Это говорит нам о том, что большее влияние оказывает точность внешнего интегрирования.

**Исследования для внешнего направления и метода Гаусса:**





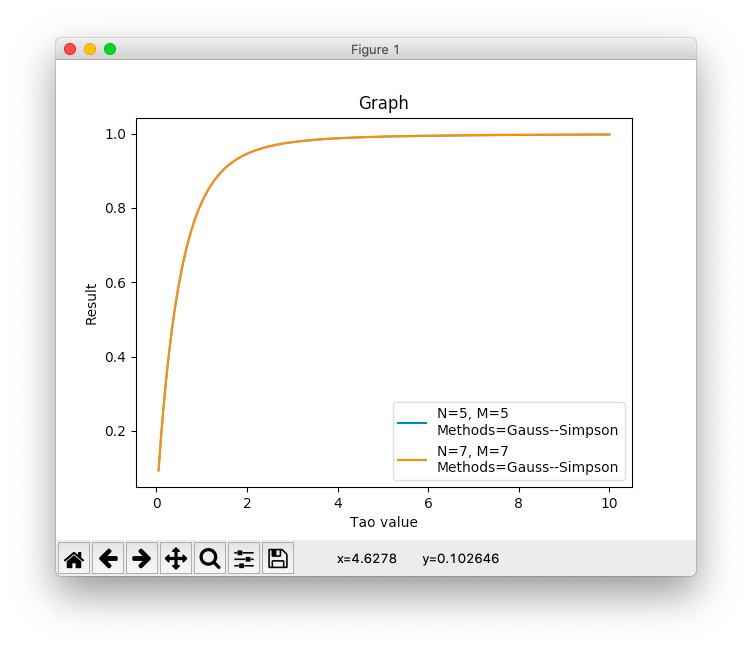
Результаты близки к действительному даже при малом кол-ве узлов.

Видно, что Гаусс дает более точные результаты, чем Симпсон на внешнем направлении.

Очевидно, что на внутреннем направлении Гаусса большую часть в погрешность будет вносить Симпсон на внешнем направлении.

1. **Построить график зависимости ε(τ) в диапазоне изменения τ =0.05-10. Указать при каком количестве узлов получены результаты.**

Такие графики зависимости есть выше, но здесь можно привести еще один для кол-ва узлов по направлениям (5; 5) и (7; 7).



## Код программы

"""

main.py

"""

from integral\_class import Integral

*def* main():

exe = Integral()

exe.fillData()

exe.drawData()

if \_\_name\_\_ == "\_\_main\_\_":

main()

"""

integral\_class.py

"""

from numpy import arange

from numpy.polynomial.legendre import leggauss

from math import pi, cos, sin, exp

import matplotlib.pyplot as plt

""" Calculation of the two-fold integral using the quadrature formulas of Gauss and Simpson """

*class* Integral:

*def* \_\_init\_\_(*self*):

*self*.choice = 1

""" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_METHODS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ """

@staticmethod

*def* gauss(*func*, *a*, *b*, *numb*):

args, coeffs = leggauss(numb)

res = 0

for i in range(numb):

res += (b - a) / 2 \* coeffs[i] \* func((b + a) / 2 + (b - a) \* args[i] / 2)

return res

@staticmethod

*def* simpson(*func*, *a*, *b*, *numb*):

h = (b - a) / (numb - 1)

x = a

res = 0

for \_ in range((numb - 1) // 2):

res += func(x) + 4 \* func(x + h) + func(x + 2 \* h)

x += 2 \* h

return res \* (h / 3)

""" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ """

@staticmethod

*def* \_\_getMainFuncLink(*param*):

subfunc = *lambda* *x*, *y*: 2 \* cos(x) / (1 - (sin(x) \*\* 2) \* (cos(y) \*\* 2))

func = *lambda* *x*, *y*: (4 / pi) \* (1 - exp(-param \* subfunc(x, y))) \* cos(x) \* sin(x)

return func

@staticmethod

*def* \_\_wrapperFunc2(*func2*, *value*):

return *lambda* *x*: func2(value, x)

*def* \_\_integrate(*self*, *func*, *limits*, *num\_of\_nodes*, *integrators*):

inner = *lambda* *x*: integrators[1](*self*.\_\_wrapperFunc2(func, x), limits[1][0], limits[1][1], num\_of\_nodes[1])

return integrators[0](inner, limits[0][0], limits[0][1], num\_of\_nodes[0])

@staticmethod

*def* \_\_taoConstructor(*integrate\_func*, *ar\_params*, *label*):

x, y = [], []

for i in arange(ar\_params[0], ar\_params[1] + ar\_params[2], ar\_params[2]):

x.append(i)

y.append(integrate\_func(i))

plt.plot(x, y, *label*=label)

*def* \_\_getLabel(*self*, *n*, *m*, *func1*, *func2*):

label = "N=" + *str*(n) + ", M=" + *str*(m) + "\nMethods="

if func1 == *self*.simpson: label += "Simpson"

else: label += "Gauss"

if func2 == *self*.simpson: label += "--Simpson"

else: label += "--Gauss"

return label

@staticmethod

*def* drawData():

plt.title("Graph")

plt.legend()

plt.ylabel("Result")

plt.xlabel("Tao value")

plt.show()

*def* fillData(*self*):

*self*.choice = 1

while *self*.choice:

N = *int*(input("N: "))

M = *int*(input("M: "))

param = *float*(input("Tao: "))

mode = *int*(input("Select a method:\n1 - Simpson\n0 - Gauss\n"))

if mode: func1 = *self*.simpson

else: func1 = *self*.gauss

mode = *int*(input())

if mode: func2 = *self*.simpson

else: func2 = *self*.gauss

param\_integrate = *lambda* *tao*: *self*.\_\_integrate(*self*.\_\_getMainFuncLink(tao), [[0, pi / 2], [0, pi / 2]], [N, M], [func1, func2])

print("Result:", param\_integrate(param))

*self*.\_\_taoConstructor(param\_integrate, [0.05, 10, 0.05], *self*.\_\_getLabel(N, M, func1, func2))

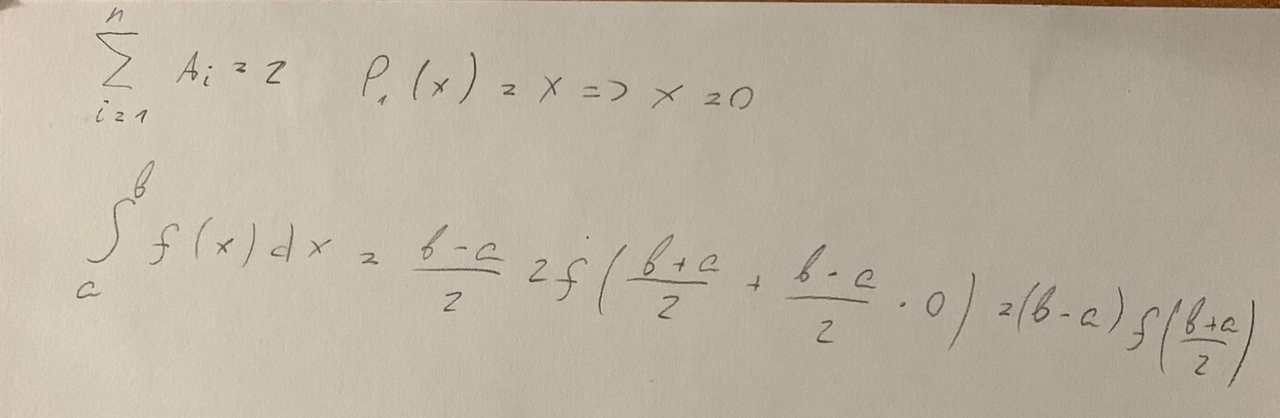
*self*.choice = *int*(input("1 - Continue\n0 - End\n"))

## Контрольные вопросы

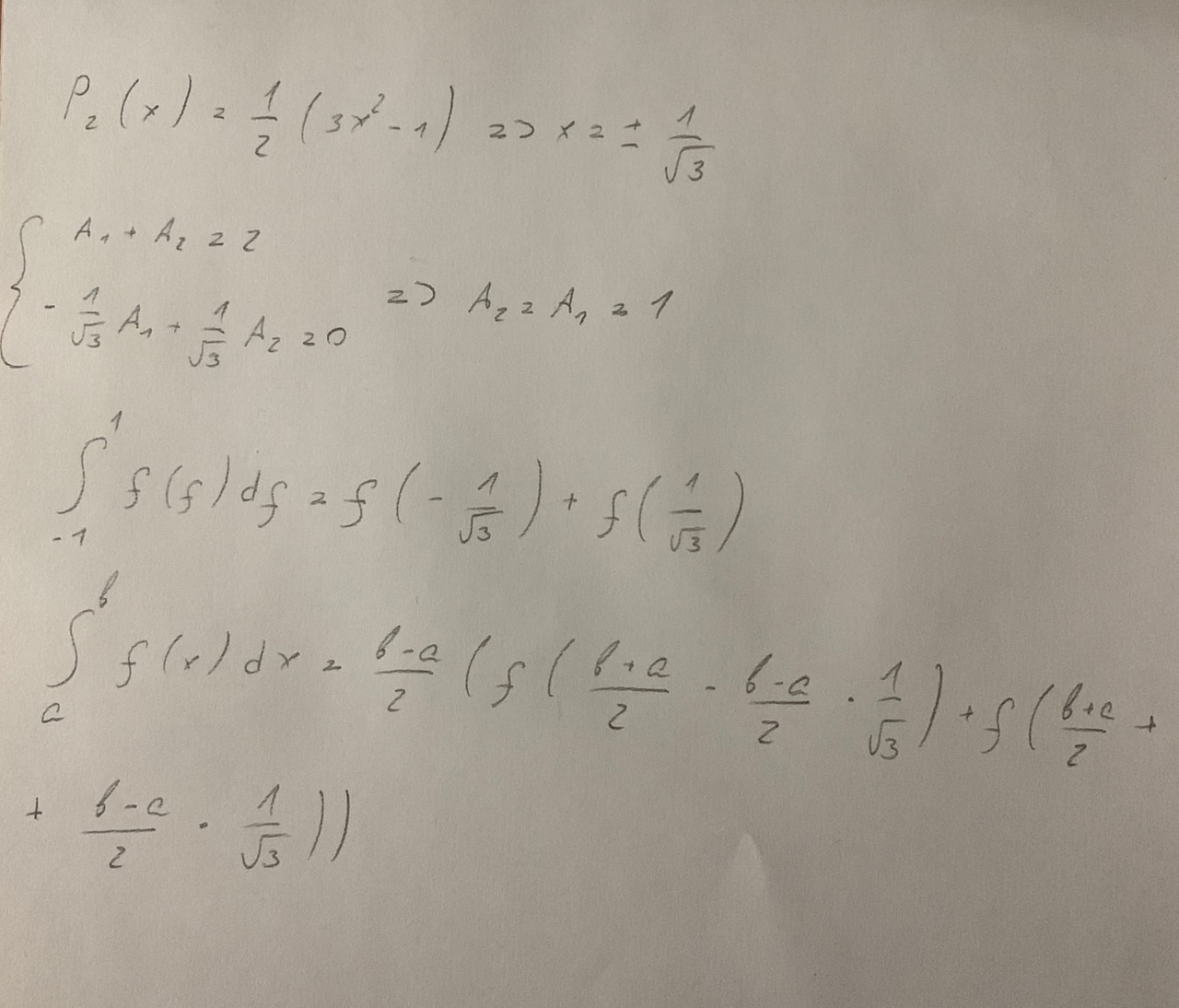
1. В каких ситуациях теоретический порядок квадратурных формул численного интегрирования не достигается?

Порядок квадратурных формул численного интегрирования не достигается в ситуациях, когда подынтегральная функция не имеет соответствующих производных. Порядок точности равен номеру последней существующей производной.

1. Построить формулу Гаусса численного интегрирования при одном узле.



1. Построить формулу Гаусса численного интегрирования при двух узлах.



1. Получить обобщенную кубатурную формулу, на основе методе трапеций, с тремя узлами на каждом направлении.

