

ФАКУЛЬТЕТ: Информатика и системы управления

КАФЕДРА: Программное обеспечение ЭВМ и информационные технологии

**Лабораторная работа №5**

**Тема** Построение и программная реализация алгоритмов численного интегрирования

**Студент** Зайцева А. А.

**Группа** ИУ7 – 42Б

**Оценка (баллы) \_\_\_\_\_\_\_**

**Преподаватель** Градов В. М.

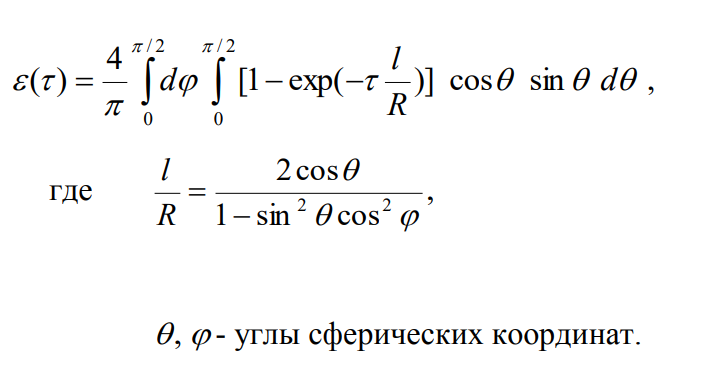
Москва.

2021 г

**Цель работы.** Получение навыков построения алгоритма вычисления двукратного интеграла с использованием квадратурных формул Гаусса и Симпсона.

1. Задание

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

****

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

1. **Физическое содержание задачи.**

У задачи есть физическое содержание, знание которого может пригодиться при отладке и тестировании программы. Речь здесь идет об определении степени черноты ε полупрозрачного однородного по объему цилиндра с большим отношением длины к радиусу, заполненного веществом с оптической плотностью τ = k R , где k , R - коэффициент оптического поглощения плазмы и внутренний радиус цилиндра. Степень черноты ε не может быть больше 1.

1. Код программы

|  |
| --- |
| import numpy as np from math import cos, sin, exp, pi from my\_gaus import solve\_gaus import matplotlib.pyplot as plt  EPS = 1e-6   # коэффициенты полинома Лежандра степени n def make\_legender\_polynom(n):    assert(n >= 0)     if n == 0:        coeffs = [1]    elif n == 1:        coeffs = [0, 1]    else:        coeffs = [0]        coeffs\_minus1 = make\_legender\_polynom(n - 1)        mult1 = (2 \* n - 1) / n        coeffs\_minus2 = make\_legender\_polynom(n - 2)        mult2 = -(n - 1) / n        for coeff in coeffs\_minus1:            coeffs.append(coeff \* mult1)        for i in range(n-1):            coeffs[i] += coeffs\_minus2[i] \* mult2    return coeffs   # вычисление полинома в точке x def count\_polynom(coeffs, x):    y = 0    for i in range(len(coeffs)):        y += coeffs[i] \* (x \*\* i)    return y   # нахождениие корня методом половинного деления def find\_root\_hdm(coeffs, a, b):    while abs(b - a) > EPS:        x = (b + a) / 2.0        if count\_polynom(coeffs, a) \* count\_polynom(coeffs, x) < 0:            b = x        else:            a = x    if abs(count\_polynom(coeffs, x)) < EPS\*10:        return x   # корни полинома Лежандра степени n def find\_roots\_legender\_polynom(n):    assert(n > 0)     coeffs = make\_legender\_polynom(n)    roots = []     if n % 2:        roots.append(0)     step = 1. / (n+1)    a = 0    b = step     while len(roots) < n:        root = find\_root\_hdm(coeffs, a, b)        if root and (root not in roots):            roots.append(root)            roots.append(-root)        a = b        b += step     return sorted(roots)   # нахождение весов квадратур Гаусса def find\_gaus\_weights(roots):    matrix = []    answers = []    for i in range(len(roots)):        line = np.array(roots) \*\* i        if i % 2:            answer = 0        else:            answer = 2. / (i + 1)         answers.append(answer)        matrix.append(line)    # solve\_gaus - решение СЛАУ методом Гаусса, реализация приводилась    # в предыдущих лабораторных    return solve\_gaus(matrix, answers)   # интегрирование методом Гаусса def integrate\_gaus(func, n, a, b):    roots = np.array(find\_roots\_legender\_polynom(n))    roots\_scaled = ((b + a) / 2) + roots \* ((b - a) / 2)    vectorized\_func = np.vectorize(func)    function\_values = vectorized\_func(roots\_scaled)    weights = np.array(find\_gaus\_weights(roots))    answer = sum(function\_values \* weights) \* ((b - a) / 2)    return answer   # интегрирование методом Симпсона def integrate\_simpson(func, n, a, b):    h = float(b - a) / (n - 1)    x = a    answer = 0    for i in range((n - 1) // 2):        answer += func(x) + 4 \* func(x + h) + func(x + 2 \* h)        x += 2 \* h    answer \*= (h / 3)    return answer   # перевод функции от двух переменных в функцию от одной переменной def reform\_21(func2, fixed):    return lambda x: func2(fixed, x)   # вычисление двукратного интеграла def tf\_integrate(func, limits, ns, methods):    Func = lambda x: methods[1](reform\_21(func, x), ns[1], limits[1][0], limits[1][1])    return methods[0](Func, ns[0], limits[0][0], limits[0][1])   # создание функции для для вычисления двукратного интеграла def make\_func\_to\_count(tau):    l\_r = lambda teta, phi: 2 \* cos(teta) / (1 - (sin(teta) \*\* 2) \* (cos(phi) \*\* 2))    func = lambda teta, phi: (4 / pi) \* (1 - exp(-tau \* l\_r(teta, phi))) \* cos(teta) \* sin(teta)    return func   # выполнение задания при фиксированном тау и количествах узлов def main\_count(tau, n1, n2):    func = make\_func\_to\_count(tau)    return tf\_integrate(func, [[0, pi / 2], [0, pi / 2]], [n1, n2], [integrate\_gaus, integrate\_simpson])   def comare\_ns():    tau = 1    ns\_g = [3, 5, 7, 9, 11]    ns\_s = [6, 10,  14]    plt.figure(figsize=(7, 7))    plt.rcParams['font.size'] = '12'    plt.title('Влияние количества выбираемых узлов на точность расчетов')    plt.xlabel('n\_s')    plt.ylabel('answer')    plt.grid()    m = []    for n\_g in ns\_g:        y = []        for n\_s in ns\_s:            answer = main\_count(tau, n\_g, n\_s)            y.append(answer)        m.append(y)     for i in range(len(m)):        plt.plot(ns\_s, m[i], label='n\_g=%d' % (ns\_g[i]))    plt.legend(loc='best')    plt.show()     plt.figure(figsize=(7, 7))    plt.rcParams['font.size'] = '12'    plt.title('Влияние количества выбираемых узлов на точность расчетов')    plt.xlabel('n\_g')    plt.ylabel('answer')    plt.grid()    m = []    for n\_s in ns\_s:        y = []        for n\_g in ns\_g:            answer = main\_count(tau, n\_g, n\_s)            y.append(answer)        m.append(y)     for i in range(len(m)):        plt.plot(ns\_g, m[i], label='n\_s=%d' % (ns\_s[i]))    plt.legend(loc='best')    plt.show()   def compare\_tau():    n\_s = n\_g = 5    plt.figure(figsize=(7, 7))    plt.rcParams['font.size'] = '12'    plt.title('epsilon(tau)')    plt.xlabel('tau')    plt.ylabel('epsilon')    plt.grid()    answers = []     taus = np.linspace(0.05, 10)    for tau in taus:        answer = main\_count(tau, n\_g, n\_s)        answers.append(answer)     plt.plot(taus, answers)    plt.show()   def main():    compare\_tau()    comare\_ns()   if \_\_name\_\_ == '\_\_main\_\_':    main() |

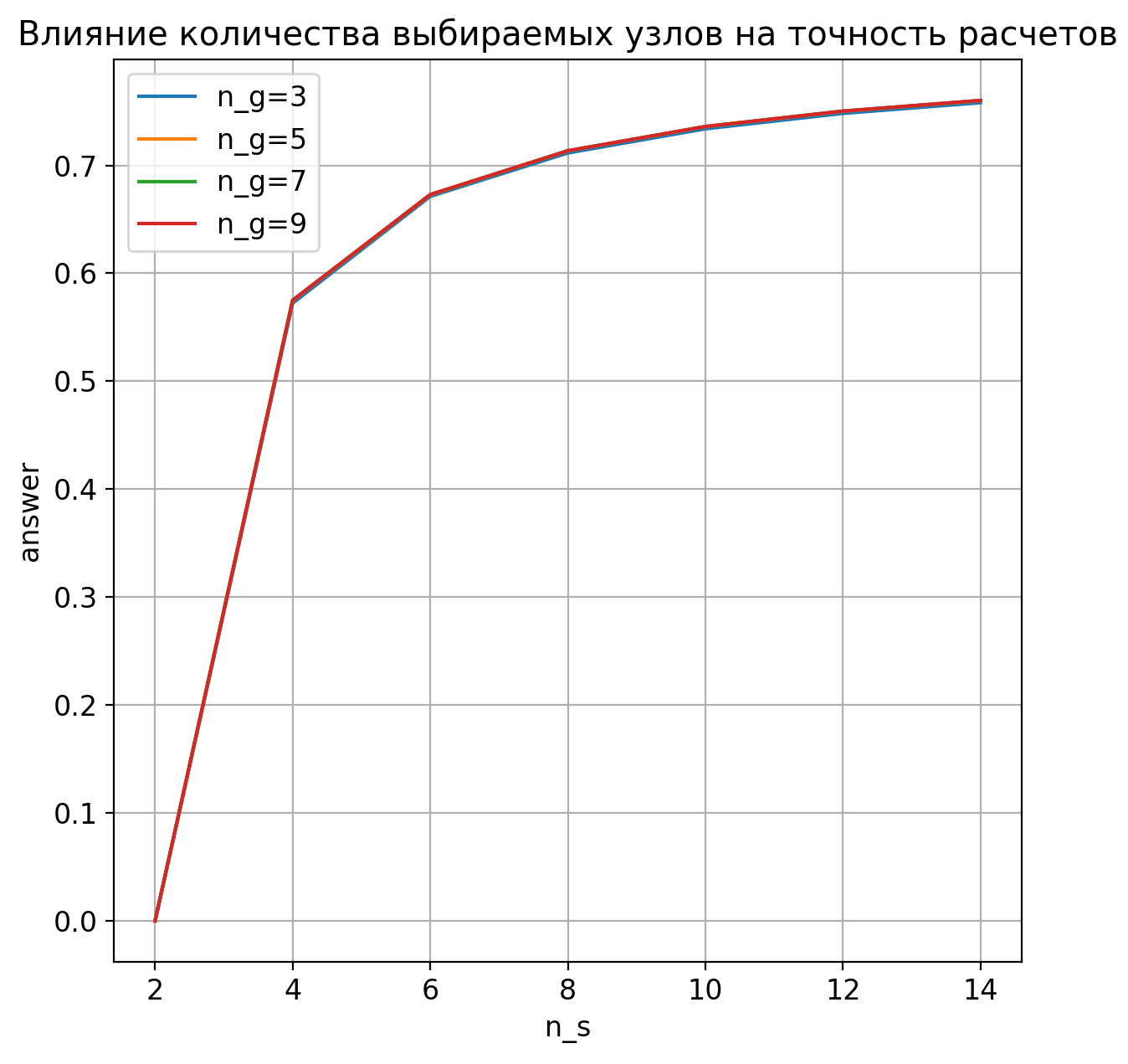
1. Результаты работы
   1. Описать алгоритм вычисления n корней полинома Лежандра n-ой степени Pn(x) при реализации формулы Гаусса.

Согласно свойству полиномов Лежандра, все корни располагаются на интервале [−1; 1], и они все действительны и различны, то есть кратных корней нет. Процедура отыскания корней выполняется численным методом, например, методом половинного деления (можно разбивать отрезок [-1, 1] на участки, искать на них корень данным методом и проверять, что найденный корень является новым). Процедура повторяется до тех пор, пока не будут найдены все n корней полинома. При отделении корней можно также пользоваться тем обстоятельством, что по теореме Ролля между корнями Pn всегда лежит корень производной этого полинома.

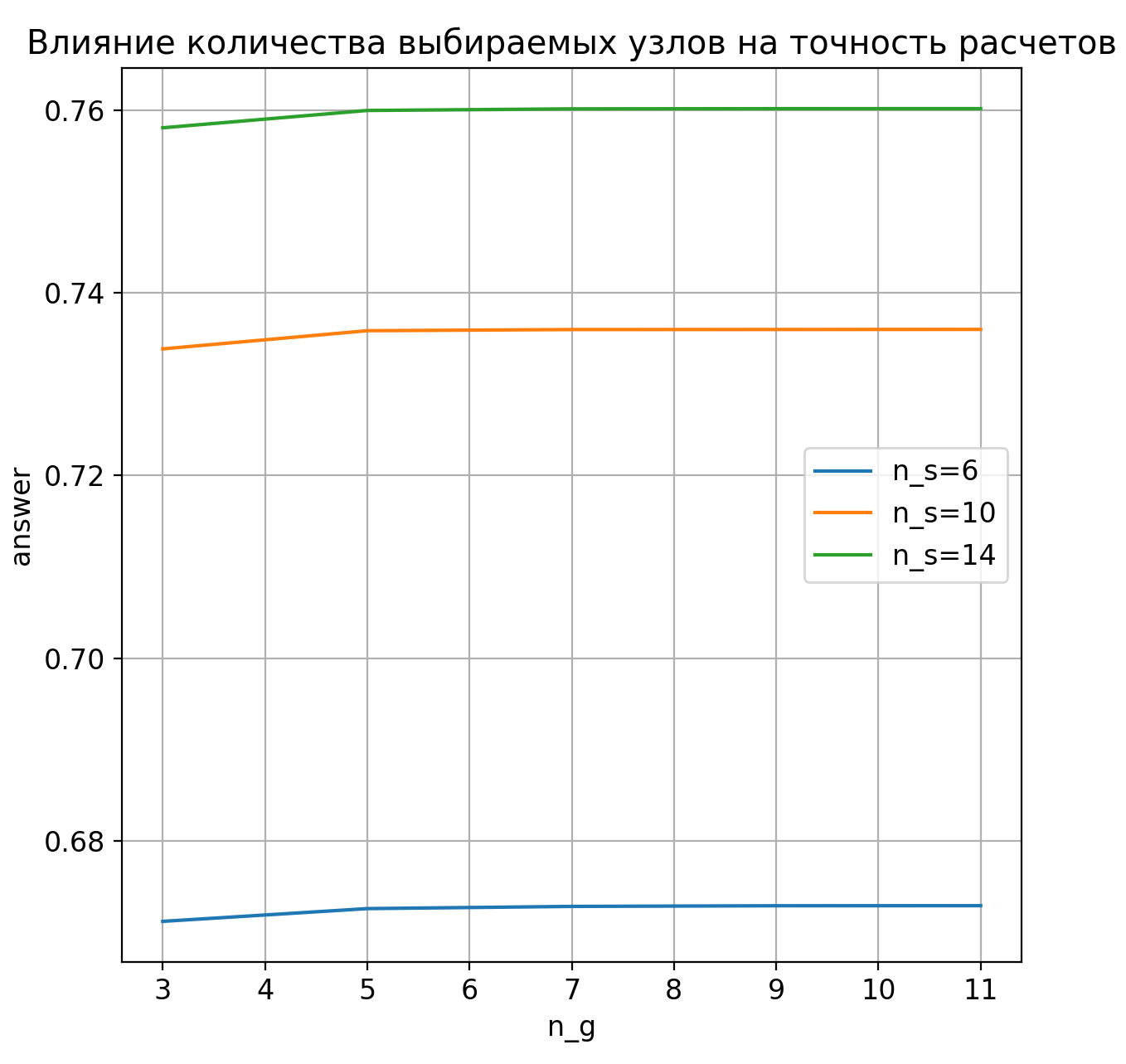
* 1. Исследовать влияние количества выбираемых узлов сетки по каждому направлению на точность расчетов.

Параметр τ=1=const, первым применяется метод Симпсона, вторым – метод Гаусса. n\_g – количество узлов для Гаусса, n\_s – для Симпсона.

При фиксированных n\_g:



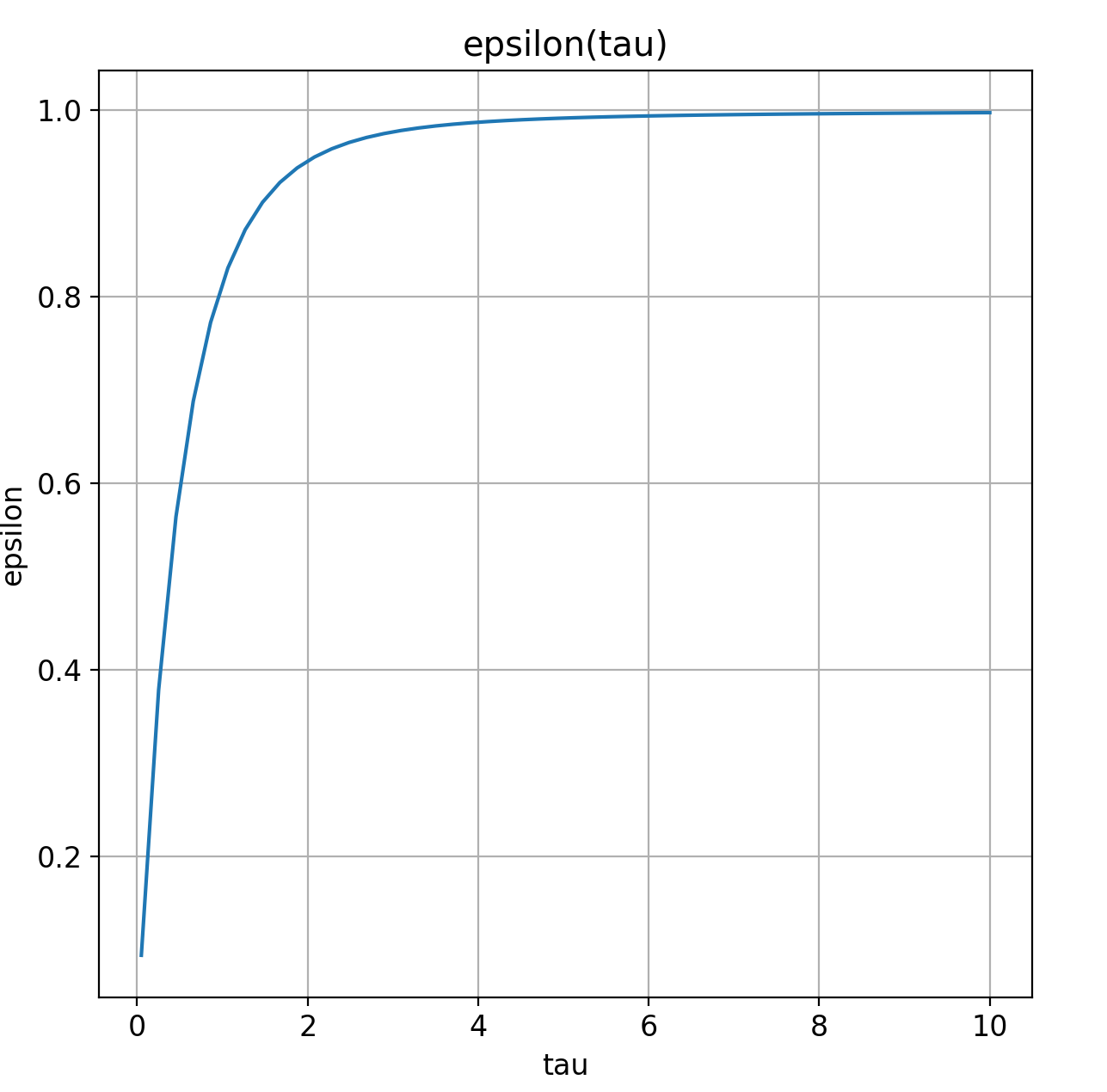
При фиксированных n\_s:



Увеличение n\_s приближает точность вычислений до выхода на асимптоту. При этом изменение n\_g не дает коренного вклада в точность результата (лишь незначительный).

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

При n\_s=n\_g=5:



С увеличением τ, увеличивается и ε, оставаясь меньше 1.

1. Вопросы при защите лабораторной работы
   1. В каких ситуациях теоретический порядок квадратурных формул численного интегрирования не достигается.

Если подынтегральная функция не имеет соответствующих производных. Например, если на отрезке интегрирования не существуют 3-я и 4-я производные, то порядок точности формулы Симпсона будет только 2-ой.

* 1. Построить формулу Гаусса численного интегрирования при одном узле.



* 1. Построить формулу Гаусса численного интегрирования при двух узлах.



* 1. Получить обобщенную кубатурную формулу, аналогичную (6.6) из лекции №6, для вычисления двойного интеграла методом последовательного интегрирования на основе формулы трапеций с тремя узлами по каждому направлению.

