## Gerb-BMSTU_01Министерство науки и высшего образования 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»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

## Лабораторная работа № 5

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

**Студент** Климов И.С.

**Группа** ИУ7-42Б

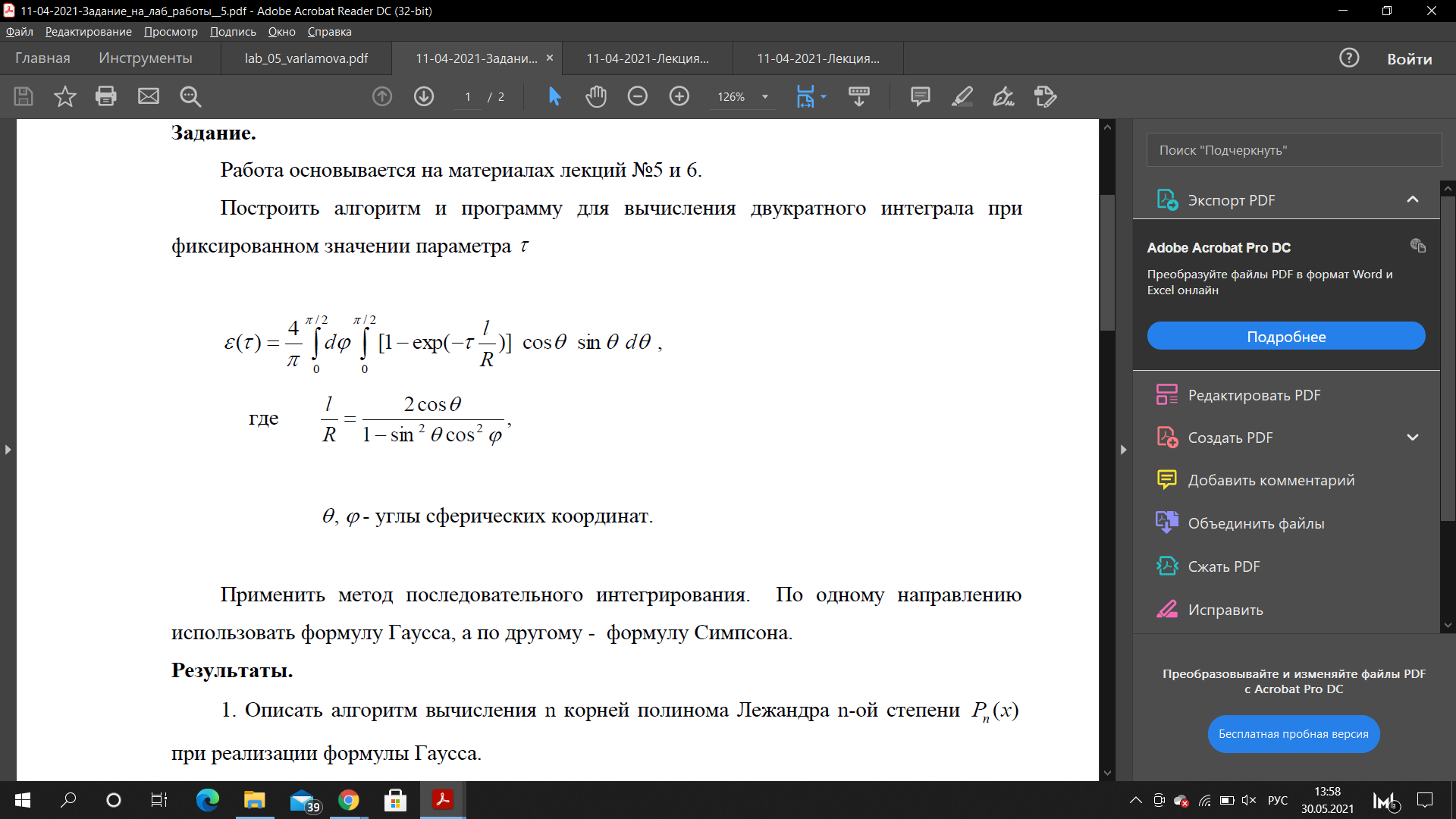
## Оценка (баллы)

**Преподаватель** Градов В.М.

Москва. 2021 г

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

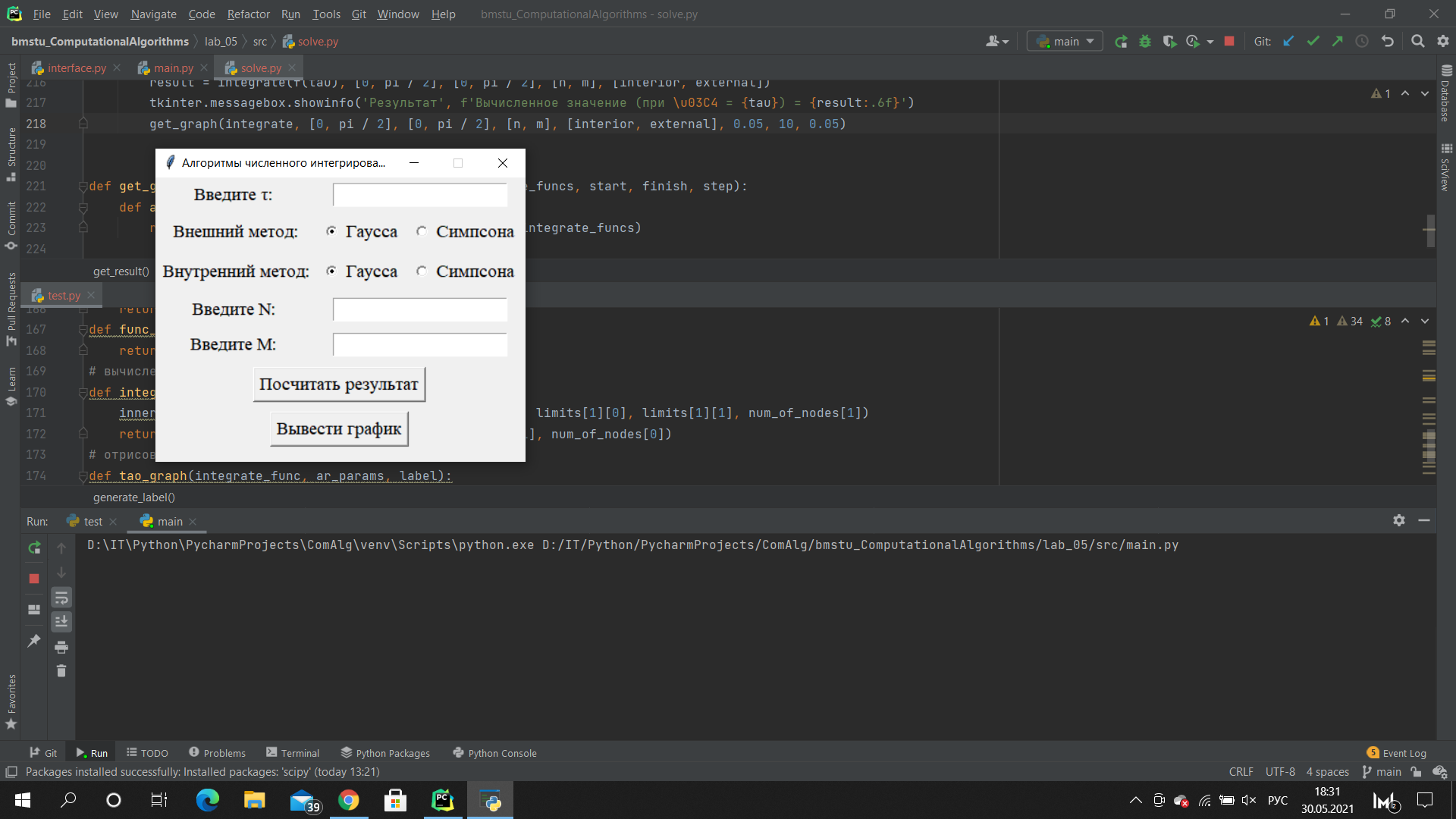
1. **Исходные данные**
2. Двухкратный интеграл:



(Применяется метод последовательного интегрирования)

1. **Код программы**

Интерфейс программы:



*Код для написания интерфейса ниже приведен не будет.*

**Листинг 1. solve.py**

**import** **tkinter.messagebox**

**from** **math** **import** pi, sin, cos, exp

**import** **matplotlib**

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**def** f(tau):

**def** auxiliary(x, y):

**return** 2 \* cos(x) / (1 - (sin(x) \*\* 2) \* (cos(y) \*\* 2))

**def** main\_func(x, y):

**return** (4 / pi) \* (1 - exp(-tau \* auxiliary(x, y))) \* cos(x) \* sin(x)

**return** main\_func

**def** multi\_polynomials(polynomial\_1, polynomial\_2):

len\_1, len\_2 = len(polynomial\_1), len(polynomial\_2)

result = [0 **for** \_ **in** range(len\_1 + len\_2 - 1)]

**for** i **in** range(len\_1):

**for** j **in** range(len\_2):

result[i + j] += polynomial\_1[i] \* polynomial\_2[j]

**return** result

**def** derivative(polynomial):

result = []

**for** i **in** range(1, len(polynomial)):

result.append(polynomial[i] \* i)

**return** result

**def** find\_legendre(n):

result = [1]

on\_multi = [-1, 0, 1]

two\_degree = 1

factorial = 1

**for** i **in** range(1, n + 1):

two\_degree \*= 2

factorial \*= i

result = multi\_polynomials(result, on\_multi)

**for** i **in** range(n):

result = derivative(result)

**for** i **in** range(len(result)):

result[i] \*= 1 / (two\_degree \* factorial)

**return** result

**def** half\_method(polynomial, left, right, flag):

m = (left + right) / 2

**if** abs(left - right) < 1e-6:

**return** m

tmp = get\_value(polynomial, m)

**if** flag:

**if** tmp > 0:

**return** half\_method(polynomial, left, m, flag)

**elif** tmp < 0:

**return** half\_method(polynomial, m, right, flag)

**else**:

**if** tmp < 0:

**return** half\_method(polynomial, left, m, flag)

**elif** tmp > 0:

**return** half\_method(polynomial, m, right, flag)

**return** m

**def** get\_value(polynomial, value):

result = 0

**for** i **in** range(len(polynomial)):

result += polynomial[i] \* value \*\* i

**return** result

**def** find\_roots(polynomial):

n = len(polynomial) - 1

k = 1

**if** get\_value(polynomial, - 1) \* get\_value(polynomial, 0) > 0:

k = 0

segments = [[-1, 0]]

t = n / 2

**while** k < t:

tmp = []

k = 0

**for** i **in** range(len(segments)):

m = (segments[i][0] + segments[i][1]) / 2

tmp.append([segments[i][0], m])

tmp.append([m, segments[i][1]])

**if** get\_value(polynomial, m) == 0:

k += 1

**else**:

**if** get\_value(polynomial, segments[i][0]) \* get\_value(polynomial, m) <= 0:

k += 1

**if** get\_value(polynomial, segments[i][1]) \* get\_value(polynomial, m) <= 0:

k += 1

segments = tmp[:]

roots = []

**for** segment **in** segments:

left = get\_value(polynomial, segment[0])

right = get\_value(polynomial, segment[1])

**if** left == 0:

roots.append(segment[0])

**if** right == 0:

**continue**

**if** get\_value(polynomial, segment[0]) < 0 **and** get\_value(polynomial, segment[1]) > 0:

roots.append(half\_method(polynomial, segment[0], segment[1], True))

**elif** get\_value(polynomial, segment[0]) > 0 **and** get\_value(polynomial, segment[1]) < 0:

roots.append(half\_method(polynomial, segment[0], segment[1], False))

**if** get\_value(polynomial, segments[-1][1]) == 0:

roots.append(segments[-1][1])

**for** i **in** range(n // 2):

roots.append(-roots[i])

**return** roots

**def** solve\_gauss(matrix, n):

**for** k **in** range(n):

**for** i **in** range(k + 1, n):

coefficient = -(matrix[i][k] / matrix[k][k])

**for** j **in** range(k, n + 1):

matrix[i][j] += coefficient \* matrix[k][j]

result = [0 **for** \_ **in** range(n)]

**for** i **in** range(n - 1, -1, -1):

**for** j **in** range(n - 1, i, -1):

matrix[i][n] -= result[j] \* matrix[i][j]

result[i] = matrix[i][n] / matrix[i][i]

**return** result

**def** find\_coefficients(nodes):

matrix = []

**for** i **in** range(len(nodes)):

array = []

**for** j **in** range(len(nodes)):

array.append(nodes[j] \*\* i)

**if** i % 2 == 0:

array.append(2 / (i + 1))

**else**:

array.append(0)

matrix.append(array)

result = solve\_gauss(matrix, len(nodes))

**return** result

**def** gauss(function, a, b, n):

legendre = find\_legendre(n)

args = find\_roots(legendre)

coefficients = find\_coefficients(args)

result = 0

**for** i **in** range(n):

result += (b - a) / 2 \* coefficients[i] \* function((a + b) / 2 + (b - a) \* args[i] / 2)

**return** result

**def** simpson(function, a, b, n):

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

x = a

result = 0

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

result += function(x) + 4 \* function(x + h) + function(x + 2 \* h)

x += 2 \* h

**return** result \* h / 3

**def** integrate(function, limit\_1, limit\_2, nodes, integrate\_funcs):

**def** interior(x):

**return** integrate\_funcs[1](**lambda** y: function(x, y), limit\_2[0], limit\_2[1], nodes[1])

**return** integrate\_funcs[0](interior, limit\_1[0], limit\_1[1], nodes[0])

**def** get\_from\_window(window):

**try**:

tau, interior, external, n, m = float(window.tau), window.interior, window.external, int(window.n), int(window.m)

**if** (interior == 'simpson' **and** (n < 3 **or** n % 2 == 0)) **or** (external == 'simpson' **and** (m < 3 **or** m % 2 == 0)):

**raise** **ArithmeticError**

**return** tau, interior, external, n, m

**except** **ValueError**:

tkinter.messagebox.showerror('Ошибка', 'Убедитесь, что введены действительные числв')

**return** None

**except** **ArithmeticError**:

tkinter.messagebox.showerror('Ошибка', 'Проверьте количество узлов для метода Симпсона')

**return** None

**def** get\_result(window):

**if** get\_from\_window(window):

tau, interior, external, n, m = get\_from\_window(window)

interior = eval(interior)

external = eval(external)

result = integrate(f(tau), [0, pi / 2], [0, pi / 2], [n, m], [interior, external])

tkinter.messagebox.showinfo('Результат', f'Вычисленное значение (при **\u03C4** = {tau}) = {result:.6f}')

get\_graph(integrate, [0, pi / 2], [0, pi / 2], [n, m], [interior, external], 0.05, 10, 0.05)

**def** get\_graph(function, limit\_1, limit\_2, nodes, integrate\_funcs, start, finish, step):

**def** auxiliary(arg):

**return** function(f(arg), limit\_1, limit\_2, nodes, integrate\_funcs)

x, y = [], []

**for** tau **in** np.arange(start, finish + step / 2, step):

x.append(tau)

y.append(auxiliary(tau))

label\_1 = f'n = {nodes[0]}, метод = {integrate\_funcs[0].\_\_name\_\_}'

label\_2 = f'm = {nodes[1]}, метод = {integrate\_funcs[1].\_\_name\_\_}'

matplotlib.use('TkAgg')

plt.plot(x, y, label=f'{label\_1}**\n**{label\_2}')

**def** show\_graph():

plt.legend(loc='lower right')

plt.xlabel('Значение **\u03C4**')

plt.ylabel('Результат')

plt.show()

**Листинг 2. main.py**

**from** **interface** **import** MainWindow

**def** main():

window = MainWindow()

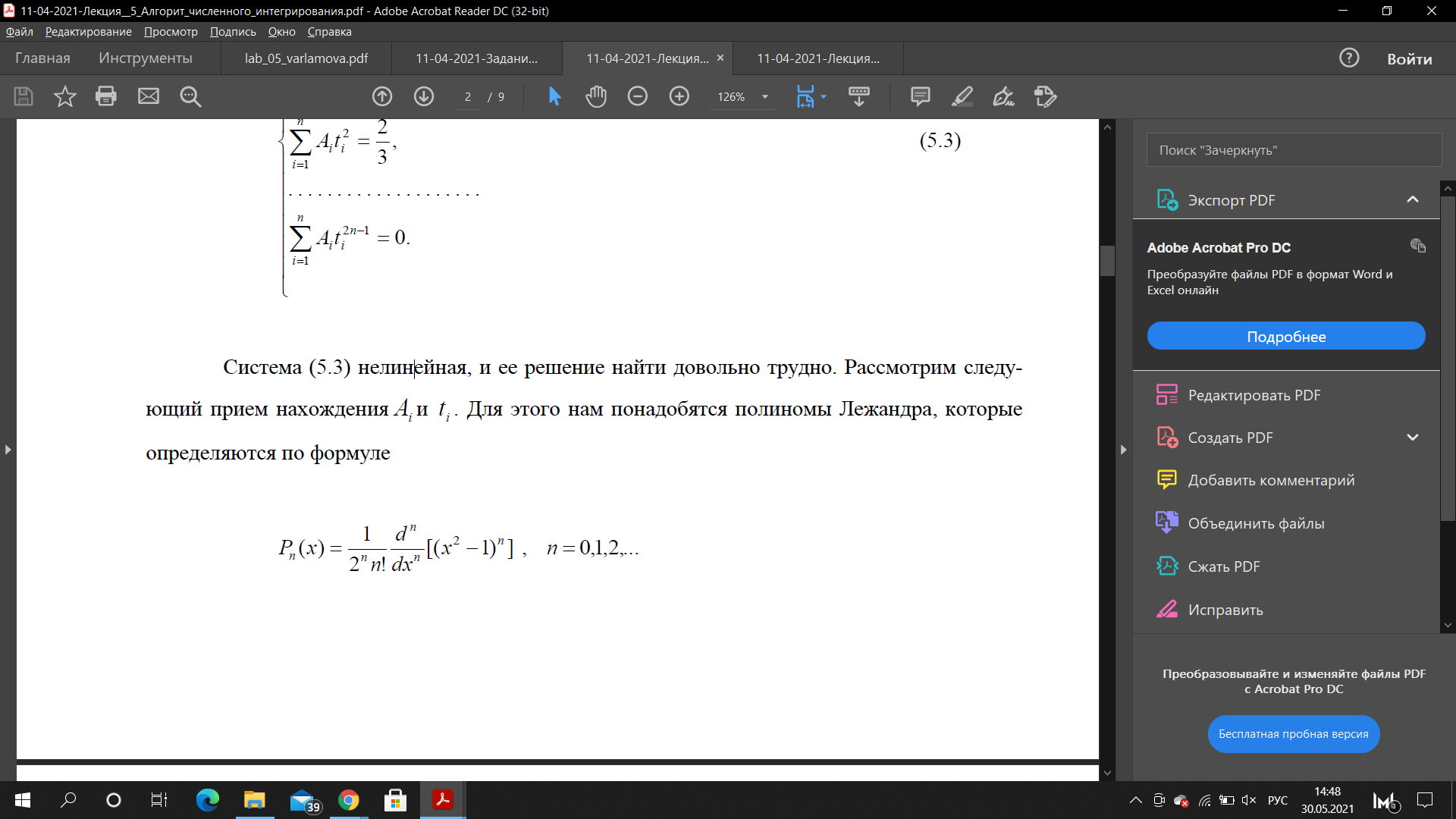
window.mainloop()

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

main()

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

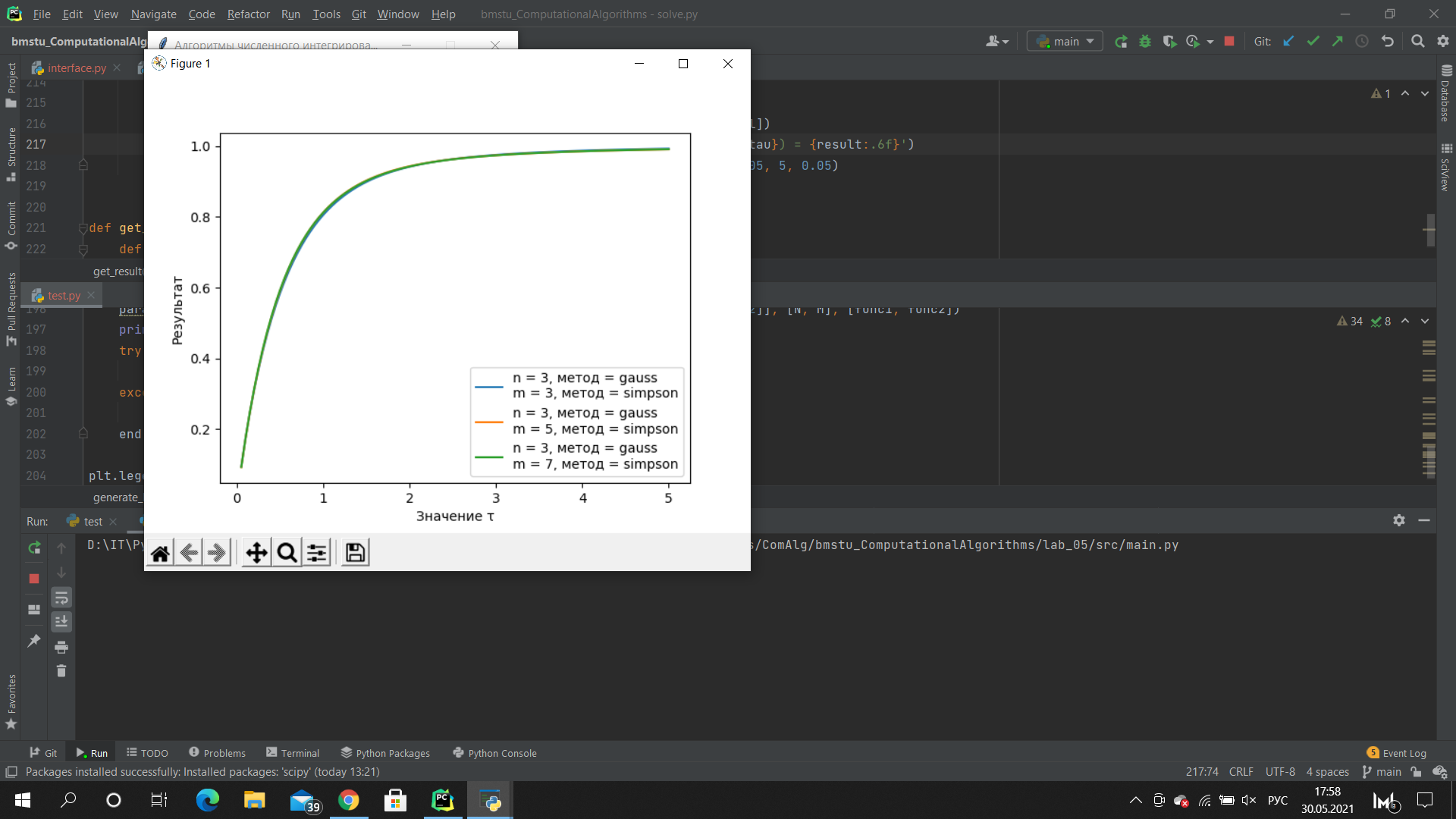
Полиномы Лежандра определяются по формуле:



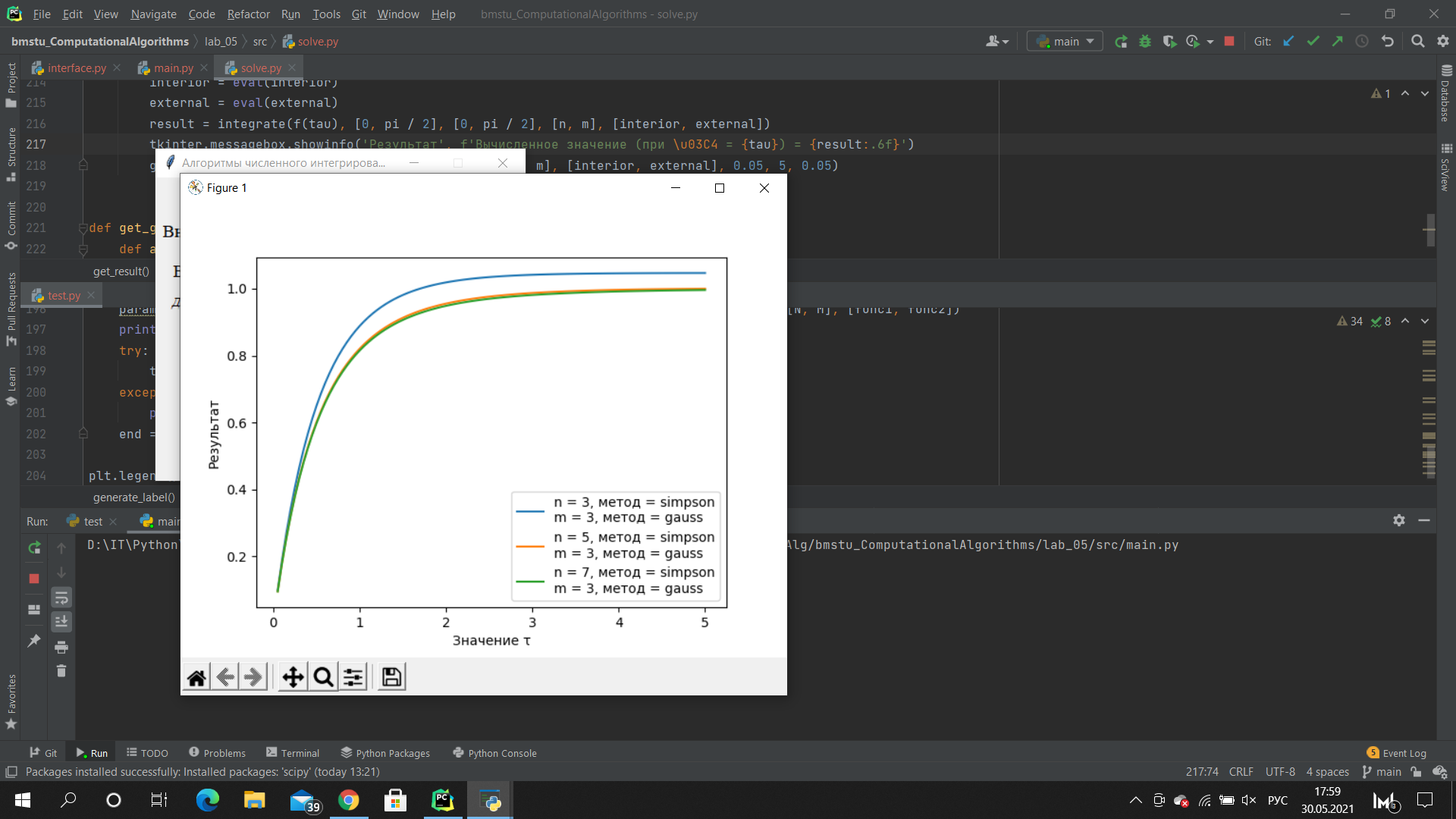
Из свойств полиномов Лежандра мы знаем, что полином имеет n действительных и различных корней, лежащих на промежутке [-1, 1]. Можно увидеть, что (x2 – 1)n – четная, тогда достаточно найти корни на промежутке [-1, 0]. Затем, пока не будет найдено n / 2 корней, все текущие отрезки разбиваются пополам. В итоге будут найдены отрезки, на каждом из которых по одному корню, и ко всем отрезкам применяется метод половинного деления. Его идея основывается на том, что если на концах отрезка функция имеет разный знак, то корень находится внутри этого отрезка. Деля отрезок пополам, определяем в какой половине лежит корень, тем самым уточняя его значение.

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

Исследование метода Симпсона (изменяем число узлов для этого метода):



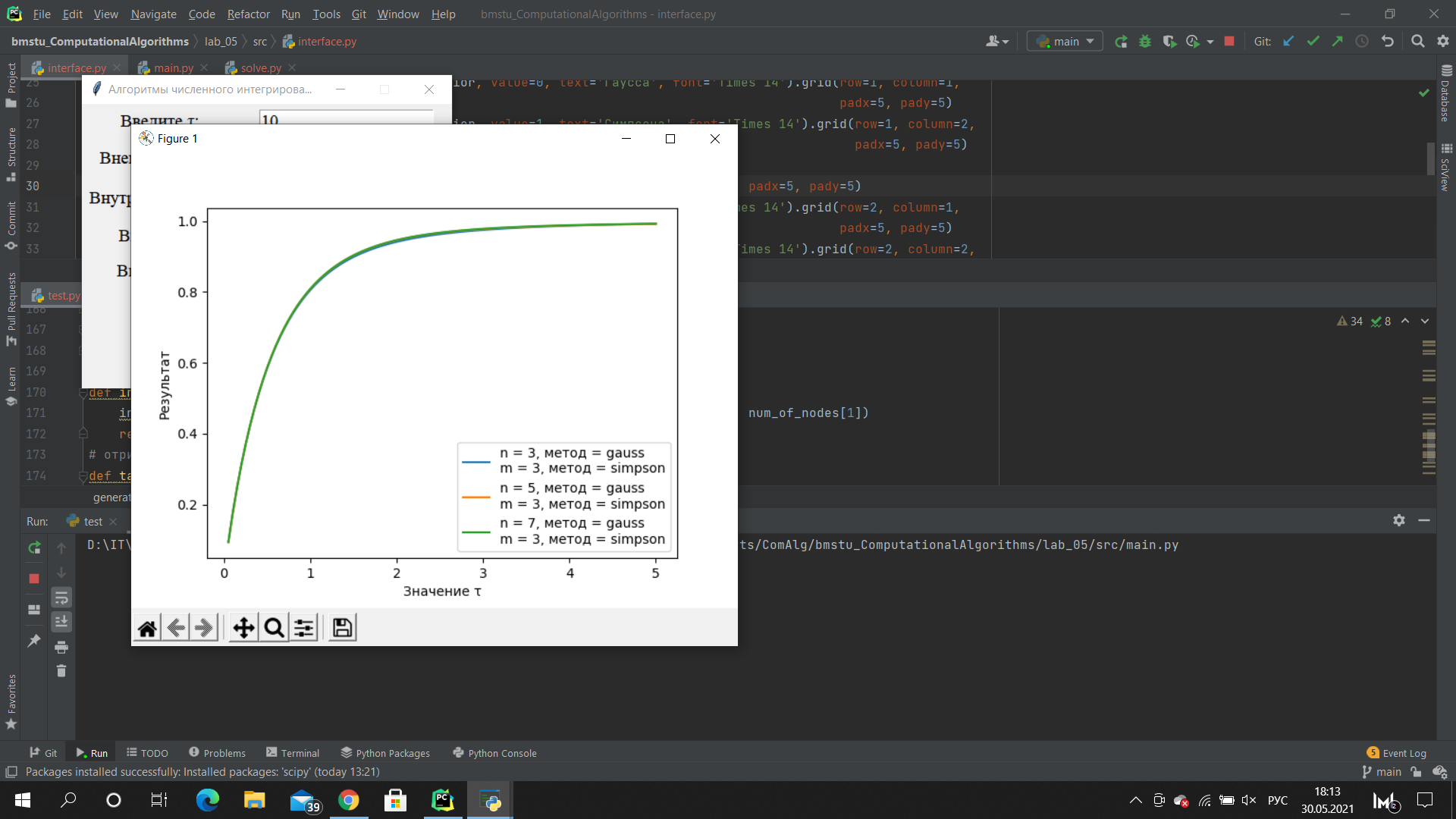
*Внешний – метод Гаусса, внутренний – метод Симпсона*



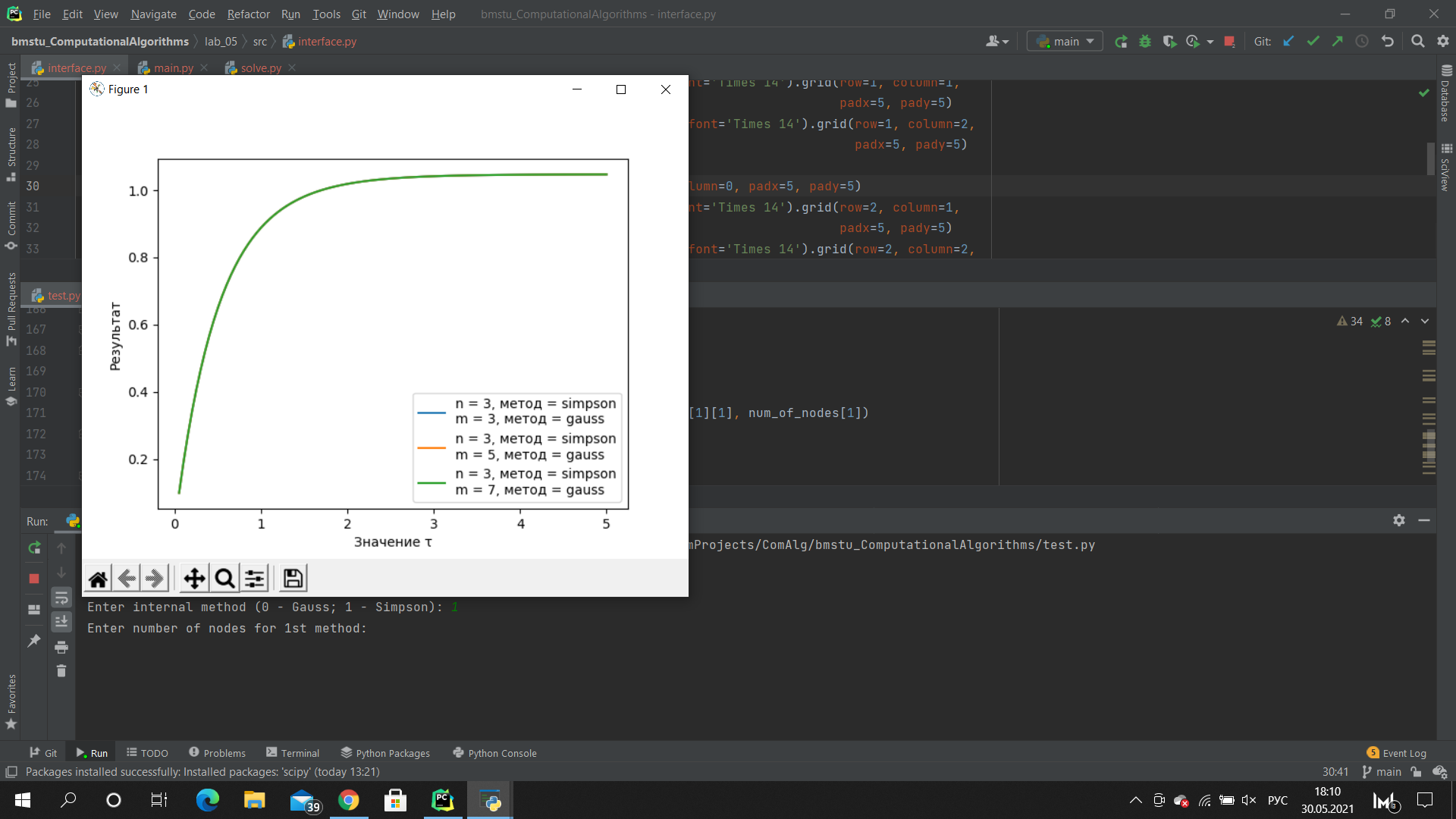
*Внешний – метод Симпсона, внутренний – метод Гаусса*

Заметим, что при применении метода Симпсона на внешнем направлении результат при меньшем количестве узлов менее точный.

Исследование метода Гаусса (изменяем число узлов для этого метода):



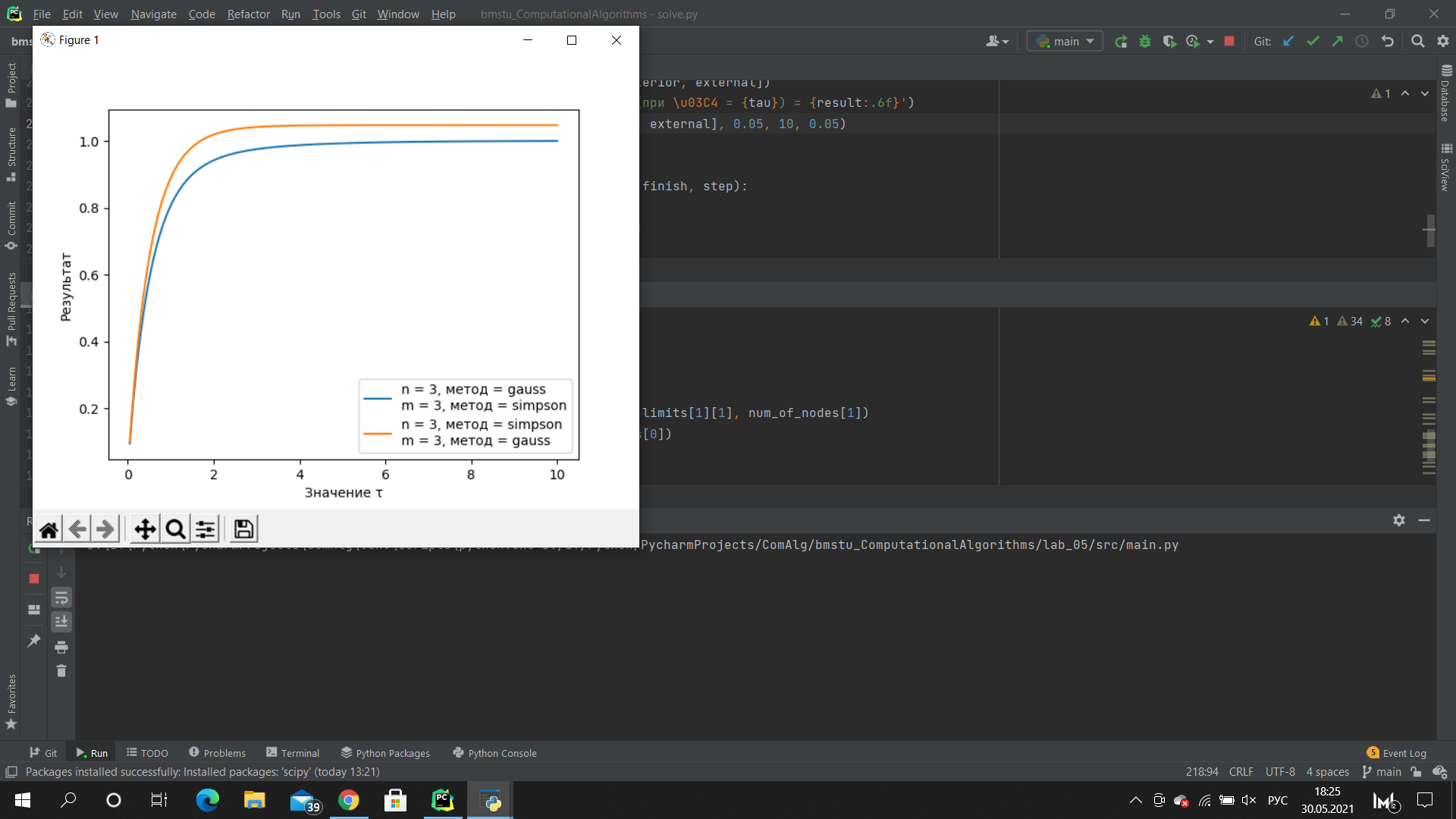
*Внешний – метод Гаусса, внутренний – метод Симпсона*

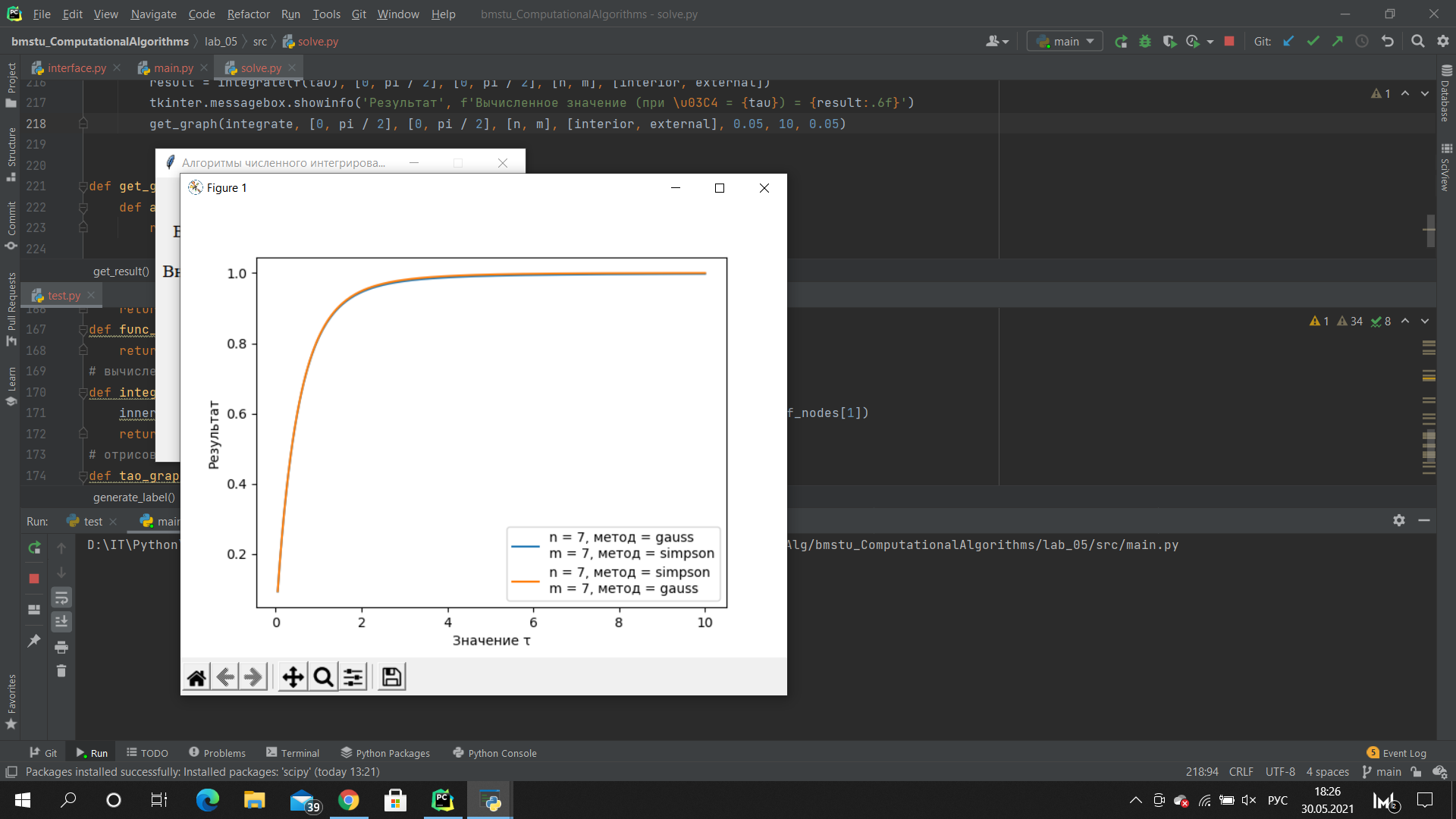


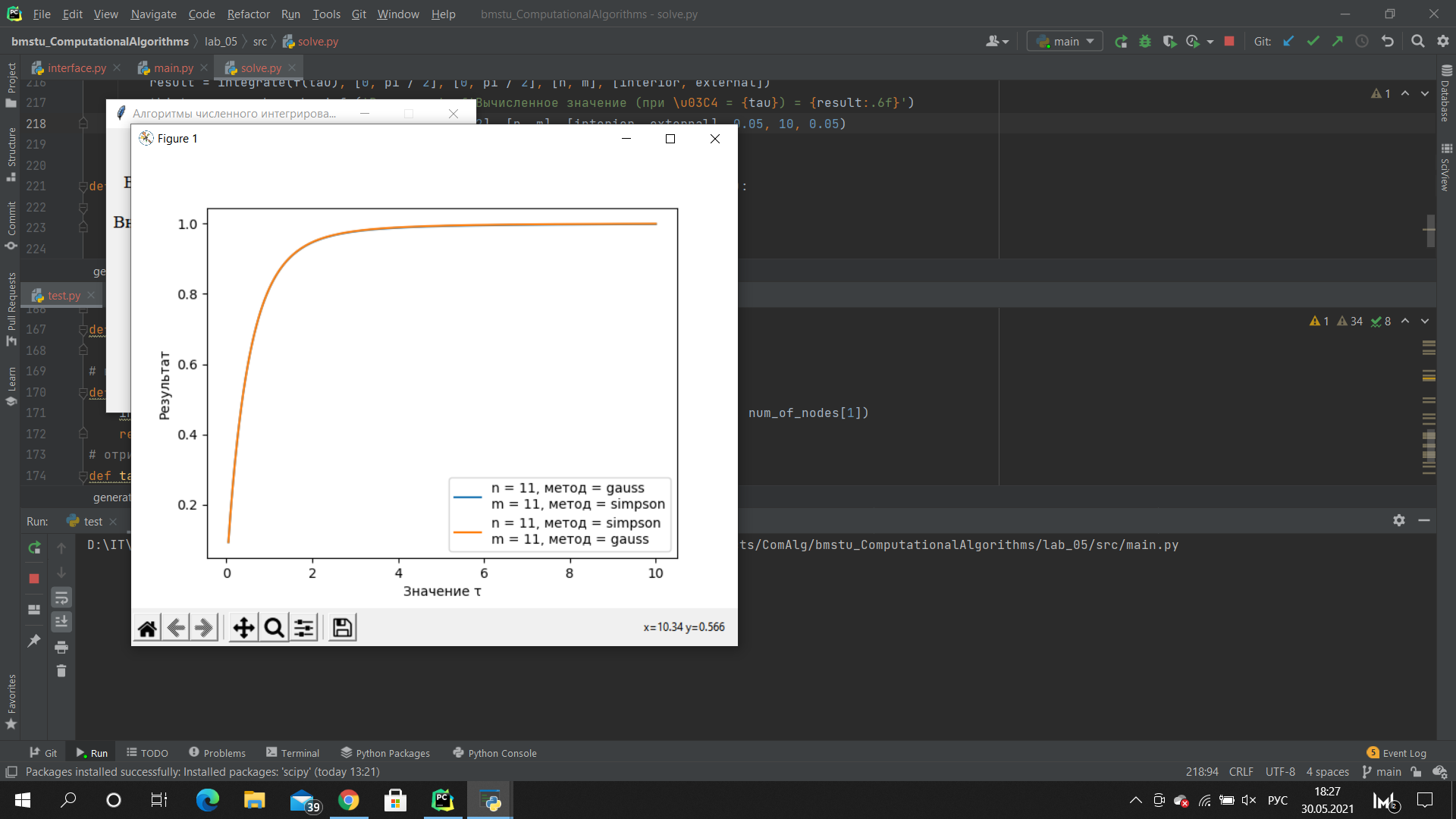
*Внешний – метод Симпсона, внутренний – метод Гаусса*

Заметим, что в обоих направлениях метод Гаусса является достаточно точным.

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







Можно сделать вывод, что метод Гаусса является более эффективным в любом направлении и вне зависимости от количества узлов. Тогда как метод Симпсона оказывается менее точным при внешнем интегрировании и малом количестве узлов.

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

Теоретический порядок точности не достигается, если у подынтегральной функции нет производных соответствующего порядка. Так, например, для формулы Симпсона, если на отрезке интегрирования не существует третьей и четвертой производных, то порядок точности будет только второй.

1. **Построить формулу Гаусса численного интегрирования при одном узле.**
2. **Построить формулу Гаусса численного интегрирования при двух узлах.**
3. **Получить обобщенную кубатурную формулу для вычисления двойного интеграла методом последовательного интегрирования на основе формулы трапеций с тремя узлами по каждому направлению.**