

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет ПИ и КТ

Лабораторная работа №3

по дисциплине: «Вычислительная математика»

«Численное интегрирование»

Вариант 1

Выполнил:

**Болорболд Аригуун**,

группа P3211

Преподаватель:

**Малышева Татьяна Алексеевна**

Санкт-Петербург

2024



1. **Цель работы:**

Найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

1. **Задание:**
   1. Вычислительная реализация задачи:
      1. Вычислить интеграл, приведенный в таблице 3.1, точно;
      2. Вычислить интеграл по формуле Ньютона – Котеса при ;
      3. Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона при ;
      4. Сравнить результаты с точным значением интеграла;
      5. Определить относительную погрешность вычислений для каждого метода;
      6. В отчете отразить последовательность вычислений.
   2. Программная реализация задачи:
      1. Пользователь выбирает функцию, интеграл которой требуется вычислить (3–5 функций), из тех, которые предлагает программа.
      2. Ввод исходных данных осуществляется с клавиатуры: пределы интегрирования, точность вычисления, начальное значение числа разбиения интервала интегрирования;
      3. Реализовать в программе методы по выбору пользователя:

* Метод прямоугольников (3 модификации: левые, правые, средние),
* Метод трапеций,
* Метод Симпсона.
  + 1. Методы должны быть реализованы в виде класса/метода/функции;
    2. Вычисление значений функции оформить в виде метода/функции;
    3. Для оценки погрешности и завершения вычислительного процесса использовать правило Рунге;
    4. Предусмотреть вывод результатов: значение интеграла, число разбиения интервала интегрирования для достижения требуемой точности;
    5. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.

1. **Вычислительная реализация:**

Точное вычисление:

По формуле Ньютона-Котеса при :

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| x | 0 | 0,25 | 0,5 | 0,75 | 1 | 1,25 | 1,5 | 1,75 | 2 |
| y | 1 | 0,421875 | –0,375 | –1,484375 | –3 | −5,015625 | −7,625 | −10,921875 | –15 |
|  | 0,07 | 0,4154 | –0,065 | 0,74 | –0,3203 | 0,74 | –0,065 | 0,4154 | 0,07 |

По формуле средних прямоугольников при :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| xi | 0 | 0,2 | 0,4 | 0,6 | 0,8 | 1 | 1,2 | 1,4 | 1,6 | 1,8 | 2 |
| (xi+ xi-1)/2 |  | 0,1 | 0,3 | 0,5 | 0,7 | 0,9 | 1,1 | 1,3 | 1,5 | 1,7 | 1,9 |
| f((xi+ xi-1)/2) |  | 0,789 | 0,283 | –0,375 | –1,233 | –2,339 | –3,741 | –5,487 | –7,625 | –10,203 | –13,269 |

По формуле трапеции при :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| xi | 0 | 0,2 | 0,4 | 0,6 | 0,8 | 1 | 1,2 | 1,4 | 1,6 | 1,8 | 2 |
| yi | 1 | 0,552 | –0,024 | –0,776 | –1,752 | –3 | –4,568 | –6,504 | –8,856 | –11,672 | –15 |

По формуле Симпсона при :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| xi | 0 | 0,2 | 0,4 | 0,6 | 0,8 | 1 | 1,2 | 1,4 | 1,6 | 1,8 | 2 |
| yi | 1 | 0,552 | –0,024 | –0,776 | –1,752 | –3 | –4,568 | –6,504 | –8,856 | –11,672 | –15 |

1. **Исходный код (*Python*):**

Методы:

def l\_rects\_method(function, a, b, n):  
 h = (b - a) / n  
 x\_list = [a + i \* h for i in range(n + 1)]  
 return h \* sum([function(x\_list[i]) for i in range(n)])  
  
def r\_rects\_method(function, a, b, n):  
 h = (b - a) / n  
 x\_list = [a + i \* h for i in range(n + 1)]  
 return h \* sum([function(x\_list[i]) for i in range(1, n + 1)])  
  
def mid\_rects\_method(function, a, b, n):  
 h = (b - a) / n  
 x\_list = [a + i \* h for i in range(n + 1)]  
 return h \* sum([function((x\_list[i - 1] + x\_list[i]) / 2) for i in range(1, n + 1)])  
  
def trapezoid\_method(function, a, b, n):  
 h = (b - a) / n  
 x\_list = [a + i \* h for i in range(n + 1)]  
 y\_list = [function(x) for x in x\_list]  
 return h \* ((y\_list[0] + y\_list[n]) / 2 +  
 sum([y\_list[i] for i in range(1, n)]))  
  
def simpson\_method(function, a, b, n):  
 h = (b - a) / n  
 x\_list = [a + i \* h for i in range(n + 1)]  
 y\_list = [function(x) for x in x\_list]  
 return h / 3 \* (y\_list[0] +  
 4 \* sum([y\_list[i] for i in range(1, n, 2)]) +  
 2 \* sum([y\_list[i] for i in range(2, n - 1, 2)]) +  
 y\_list[n])

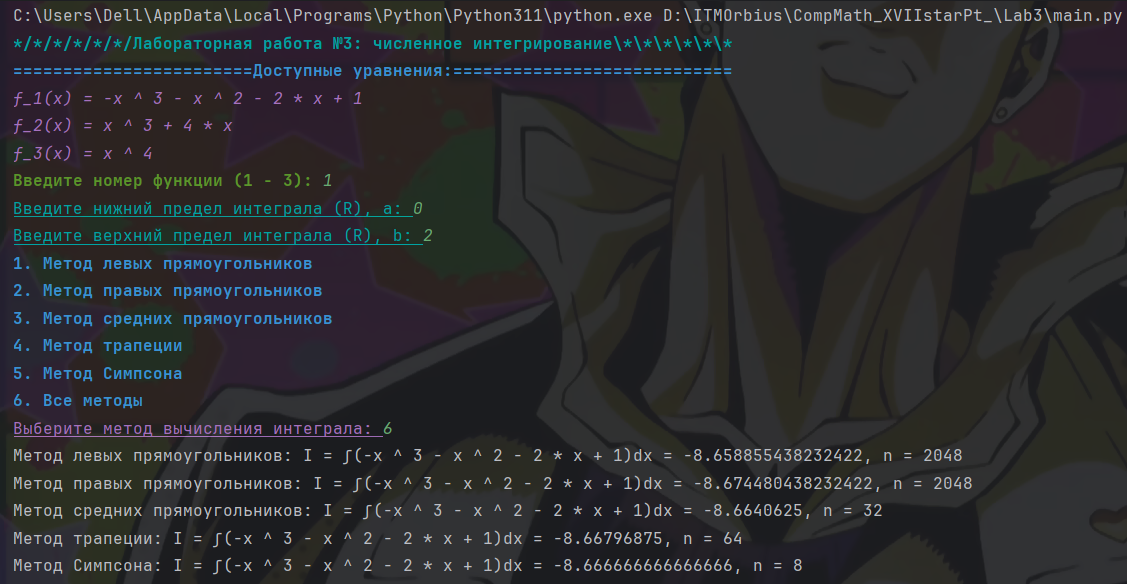
Обязательное задание:

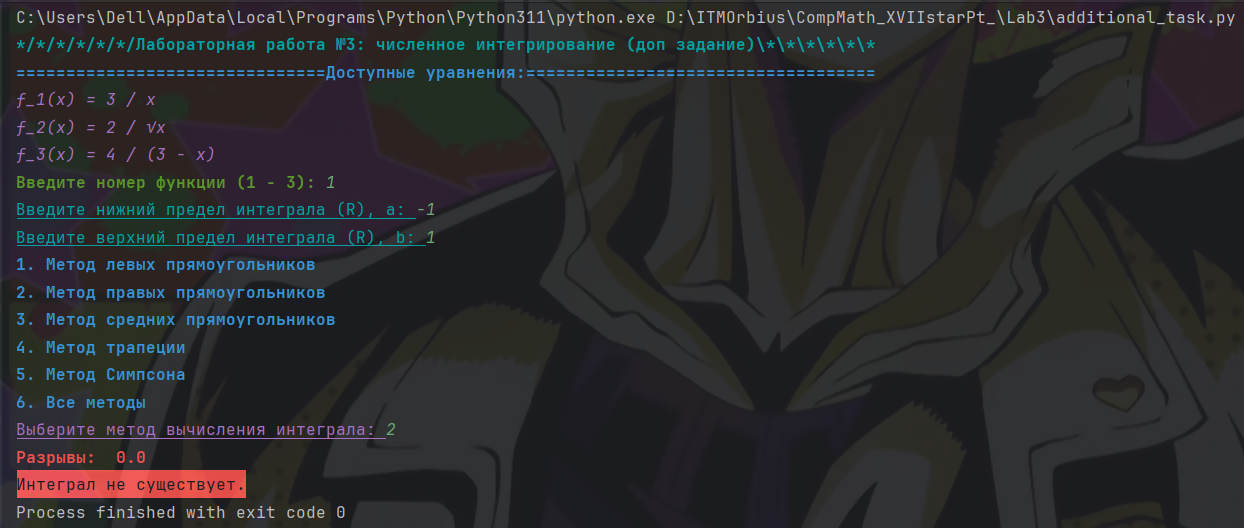
from methods import \*  
from scipy.integrate import quad  
def solve(function, a, b, eps, method):  
 k = 1  
 n = 8  
 i0 = method(function, a, b, int(n / 2))  
 i1 = method(function, a, b, n)  
 if method == l\_rects\_method or method == r\_rects\_method:  
 k = 1  
 elif method == mid\_rects\_method or method == trapezoid\_method:  
 k = 2  
 elif method == simpson\_method:  
 k = 4  
 while abs(i1 - i0) / (2\*\*k - 1) > eps:  
 i0 = i1  
 n \*= 2  
 i1 = method(function, a, b, n)  
 return i1, n  
  
methods = {  
 "Метод левых прямоугольников": l\_rects\_method,  
 "Метод правых прямоугольников": r\_rects\_method,  
 "Метод средних прямоугольников": mid\_rects\_method,  
 "Метод трапеции": trapezoid\_method,  
 "Метод Симпсона": simpson\_method  
}  
k = 0  
eps = 0.0001  
if eps <= 0.00001:  
 places = int(str(eps)[-1:]) + 4  
else:  
 places = str(eps)[::-1].find(".") + 4  
print("\033[1;36m\*/\*/\*/\*/\*/\*/" + "Лабораторная работа №3: численное интегрирование" + "\\*\\*\\*\\*\\*\\*\033[0m")  
print("\033[1;34m========================" + "Доступные уравнения:" + "============================\033[0m")  
functions = ["-x ^ 3 - x ^ 2 - 2 \* x + 1", "x ^ 3 + 4 \* x", "x ^ 4"]  
for func in functions:  
 k += 1  
 print("\033[3;35m" + f"f\_{k}(x) = {func}" + "\033[0m")  
f\_order = input("\033[1;32mВведите номер функции (1 - 3): \033[0m")  
while f\_order not in {"1", "2", "3"}:  
 print("\033[1;31mЭто не то.\033[0m")  
 f\_order = input("\033[1;32mВведите номер функции (1 - 3): \033[0m")  
  
while 1:  
 try:  
 a = float(input("\033[4;36mВведите нижний предел интеграла (R), a: \033[0m"))  
 except Exception:  
 continue  
 break  
while 1:  
 try:  
 b = float(input("\033[4;36mВведите верхний предел интеграла (R), b: \033[0m"))  
 except Exception:  
 continue  
 break  
  
f\_order = int(f\_order)  
if f\_order == 1:  
 f = lambda x: -x \*\* 3 - x \*\* 2 - 2 \* x + 1  
elif f\_order == 2:  
 f = lambda x: x \*\* 3 + 4 \* x  
else:  
 f = lambda x: x \*\* 4  
k = 0  
for method in methods:  
 k += 1  
 print("\033[1;34m" + str(k) + ".", method, "\033[0m")  
print("\033[1;34m6. Все методы\033[0m")  
while True:  
 try:  
 method\_order = int(input("\033[4;35mВыберите метод вычисления интеграла: \033[0m"))  
 except Exception:  
 continue  
 break  
res = quad(f, a, b)  
print("\033[1;35m" + f"Точное значение интеграла: I = ∫({functions[f\_order - 1]})dx = {round(res[0], places)}" + "\033[0m")  
match method\_order:  
 case 1:  
 res, n = solve(f, a, b, eps, l\_rects\_method)  
 print(f"Метод левых прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}, n = {n}")  
 case 2:  
 res, n = solve(f, a, b, eps, r\_rects\_method)  
 print(f"Метод правых прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}, n = {n}")  
 case 3:  
 res, n = solve(f, a, b, eps, mid\_rects\_method)  
 print(f"Метод средних прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}, n = {n}")  
 case 4:  
 res, n = solve(f, a, b, eps, trapezoid\_method)  
 print(f"Метод трапеции: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}, n = {n}")  
 case 5:  
 res, n = solve(f, a, b, eps, simpson\_method)  
 print(f"Метод Симпсона: I = ∫({functions[f\_order - 1]})dx = {round(res. places)}, n = {n}")  
 case \_:  
 for method in methods:  
 res, n = solve(f, a, b, eps, methods[method])  
 print(f"{method}: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}, n = {n}")

Дополнительное задание:

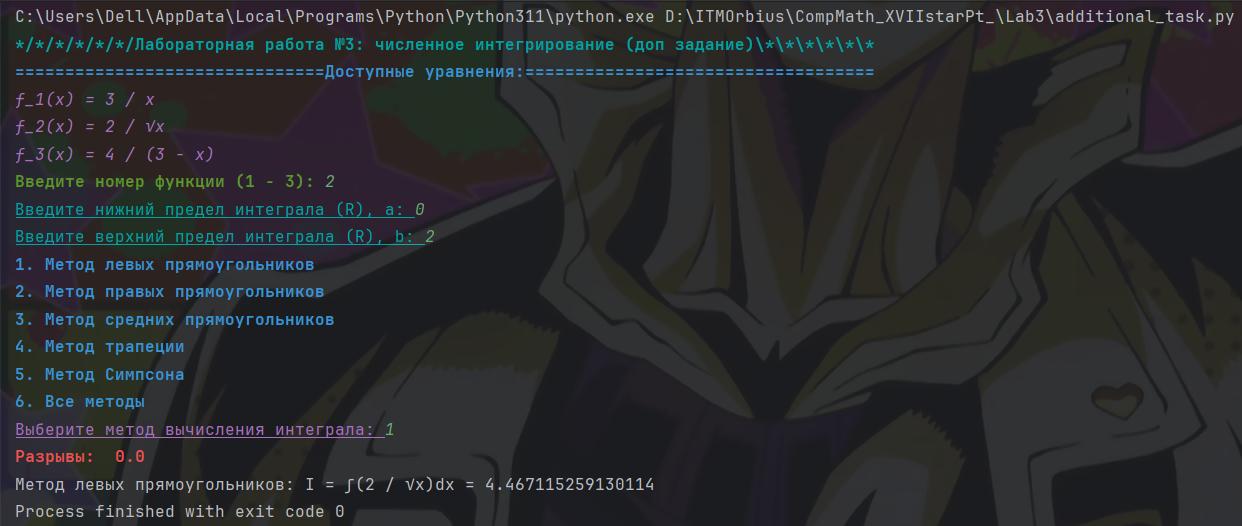
from math import sqrt  
from methods import \*  
from scipy.integrate import quad  
import sympy  
  
def find\_discontinuities(function, a, b, eps):  
 discontinuities = []  
 x = a  
 while x <= b:  
 try:  
 function(x)  
 except ArithmeticError:  
 discontinuities.append(x)  
 x = round(x + eps, 3)  
 return discontinuities  
def try\_to\_solve(function, x):  
 try:  
 return function(x)  
 except Exception:  
 return None  
  
def solve(function, a, b, eps, method):  
 n, k = 4, 0  
 i0 = method(function, a, b, n)  
 i1 = method(function, a, b, n \* 2)  
 if method == l\_rects\_method or method == r\_rects\_method:  
 k = 1  
 elif method == mid\_rects\_method or method == trapezoid\_method:  
 k = 2  
 elif method == simpson\_method:  
 k = 4  
 while abs(i1 - i0) / (2 \*\* k - 1) > eps:  
 n \*= 2  
 i0 = i1  
 i1 = method(function, a, b, n \* 2)  
 return i1  
  
methods = {  
 "Метод левых прямоугольников": l\_rects\_method,  
 "Метод правых прямоугольников": r\_rects\_method,  
 "Метод средних прямоугольников": mid\_rects\_method,  
 "Метод трапеции": trapezoid\_method,  
 "Метод Симпсона": simpson\_method  
}  
print("\033[1;36m\*/\*/\*/\*/\*/\*/" + "Лабораторная работа №3: численное интегрирование (доп задание)" + "\\*\\*\\*\\*\\*\\*\033[0m")  
print("\033[1;34m===============================" + "Доступные уравнения:" + "===================================\033[0m")  
eps = 0.0001  
functions = ["3 / x", "2 / √x", "4 / (3 - x)", "1 / (4 \* x - x \*\* 3)"]  
k = 0  
for func in functions:  
 k += 1  
 print("\033[3;35m" + f"f\_{k}(x) = {func}" + "\033[0m")  
if eps <= 0.00001:  
 places = int(str(eps)[-1:]) + 4  
else:  
 places = str(eps)[::-1].find(".") + 4  
f\_order = input("\033[1;32mВведите номер функции (1 - 4): \033[0m")  
while f\_order not in {"1", "2", "3", "4"}:  
 print("\033[1;31mЭто не то.\033[0m")  
 f\_order = input("\033[1;32mВведите номер функции (1 - 4): \033[0m")  
  
while 1:  
 try:  
 a = float(input("\033[4;36mВведите нижний предел интеграла (R), a: \033[0m"))  
 except Exception:  
 continue  
 break  
while 1:  
 try:  
 b = float(input("\033[4;36mВведите верхний предел интеграла (R), b: \033[0m"))  
 except Exception:  
 continue  
 break  
  
f\_order = int(f\_order)  
if f\_order == 1:  
 f = lambda x: 3 / x  
elif f\_order == 2:  
 f = lambda x: 2 / sqrt(x)  
elif f\_order == 3:  
 f = lambda x: 4 / (3 - x)  
else:  
 f = lambda x: 1 / (4 \* x - x \*\* 3)  
cnt = 0  
for method in methods:  
 cnt += 1  
 print("\033[1;34m" + str(cnt) + ".", method, "\033[0m")  
print("\033[1;34m6. Все методы\033[0m")  
while True:  
 try:  
 method\_order = int(input("\033[4;35mВыберите метод вычисления интеграла: \033[0m"))  
 except Exception:  
 continue  
 break  
try:  
 discontinuities = find\_discontinuities(f, a, b, 0.001)  
except ValueError:  
 print("\033[7;31mДанный интеграл не определен на заданном промежутке.\033[0m", end = "")  
 exit(0)  
if not bool(discontinuities):  
 print("\033[1;31mРазрыва нет.\033[0m")  
else:  
 print("\033[1;31mРазрывы: ", \*discontinuities, "\033[0m")  
for d in discontinuities:  
 y1 = try\_to\_solve(f, d - eps)  
 y2 = try\_to\_solve(f, d + eps)  
 if y1 is not None and y2 is not None and abs(y1 - y2) > eps:  
 print("\033[7;31mДанный интеграл не сходится.\033[0m", end="")  
 exit(0)  
stop = False  
if not stop:  
 if a in discontinuities:  
 a += eps  
 elif b in discontinuities:  
 b -= eps  
 elif discontinuities:  
 try:  
 res = quad(f, a, b)  
 res += quad(f, d + eps, b)  
 print("\033[1;35m" + f"Точное значение интеграла: I = ∫({functions[f\_order - 1]})dx = {round(res[0], places)}" + "\033[0m")  
 except ValueError:  
 print("Данный интеграл не определен на заданном промежутке.")  
 match method\_order:  
 case 1:  
 for d in discontinuities:  
 res = solve(f, a, b, d - eps, l\_rects\_method)  
 res += solve(f, d + eps, b - eps, eps, l\_rects\_method)  
 print(f"\033[1;35mМетод левых прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 2:  
 for d in discontinuities:  
 res = solve(f, a, b, d - eps, r\_rects\_method)  
 res += solve(f, d + eps, b - eps, eps, r\_rects\_method)  
 print(f"\033[1;35mМетод правых прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 3:  
 for d in discontinuities:  
 res = solve(f, a, b, d - eps, mid\_rects\_method)  
 res += solve(f, d + eps, b - eps, eps, mid\_rects\_method)  
 print(f"\033[1;35mМетод средних прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 4:  
 for d in discontinuities:  
 res = solve(f, a, b, d - eps, trapezoid\_method)  
 res += solve(f, d + eps, b - eps, eps, trapezoid\_method)  
 print(f"\033[1;35mМетод трапеции: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 5:  
 for d in discontinuities:  
 res = solve(f, a, b, d - eps, simpson\_method)  
 res += solve(f, d + eps, eps - d, eps, simpson\_method)  
 print(f"\033[1;35mМетод Симпсона: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case \_:  
 for method in methods:  
 res = solve(f, a, b, d - eps, methods[method])  
 res += solve(f, d + eps, b - eps, eps, methods[method])  
 print(f"\033[1;35m{method}: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
  
 if not discontinuities or a - eps in discontinuities or b + eps in discontinuities:  
 res = quad(f, a, b)  
 print("\033[1;35m" + f"Точное значение интеграла: I = ∫({functions[f\_order - 1]})dx = {round(res[0], places)}" + "\033[0m")  
 match method\_order:  
 case 1:  
 res = solve(f, a, b, eps, l\_rects\_method)  
 print(f"\033[1;35mМетод левых прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 2:  
 res = solve(f, a, b, eps, r\_rects\_method)  
 print(f"\033[1;35mМетод правых прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 3:  
 res = solve(f, a, b, eps, mid\_rects\_method)  
 print(f"\033[1;35mМетод средних прямоугольников: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 4:  
 res = solve(f, a, b, eps, trapezoid\_method)  
 print(f"\033[1;35mМетод трапеции: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case 5:  
 res = solve(f, a, b, eps, simpson\_method)  
 print(f"\033[1;35mМетод Симпсона: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")  
 case \_:  
 for method in methods:  
 res = solve(f, a, b, eps, methods[method])  
 print(f"\033[1;35m{method}: I = ∫({functions[f\_order - 1]})dx = {round(res, places)}\033[0m")

1. **Примеры работы программ:**









1. **Вывод:**

В ходе этой работы я познакомился с численными методами решения интегралов. Непосредственное аналитическое интегрирование самой функции — довольно сложное дело, поэтому численные решения интегралов являются практичной заменой в условиях компьютерных вычислений.

Главные 5 методы (прямоугольников, трапеции и Симпсона) — частные случаи метода Ньютона-Котеса, которые довольно просты в освоении и применении, и имеют свои нюансы в контексте точности решения. Сам метод Ньютона-Котеса является общей случаем вышеупомянутых методов. Использует более общие и сложные формулы, и сложно применить в практике (пришлось даже использовать табличные значения для нахождения коэффициентов!).

Квадратурная формула Гаусса позволяет повысить порядок точности методов за счёт специального выбора узлов интегрирования. Это позволяет выбору оптимальных границ разбиений, в следствие которого уменьшается общая погрешность. Сложна в понимании и реализации.