Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет информационных технологий и программирования

**Лабораторная работа**

на тему:

Численное дифференцирование и интегрирование

Авторы:

Кичмарев Александр Вадимович

Гусаченко Дмитрий Сергеевич

Образовательная программа ИС

Группа №M32081

Преподаватель:

*Свинцов Михаил Викторович*

Санкт-Петербург  
2023

**Задание №1. Реализуйте перечисленные выше методы нахождения производной при фиксированном значении шага.**

**Решение:**

Для вычисления производной функции в каждой точке x возможно применение различных методов. Рассмотрим несколько таких методов:

* Правая разностная производная

Она используется, например, для анализа возрастания функции или для определения направления касательной к графику функции в данной точке.

Ниже приведена реализация метода на языке Python:

def right\_diff(f, x, h):  
 return (f(x + h) - f(x))/h

* Левая разностная производная

Она используется, например, для анализа убывания функции или для определения направления касательной к графику функции в данной точке.

Ниже приведена реализация метода на языке Python:

def left\_diff(f, x, h):  
 return (f(x) - f(x - h))/h

* Центральной разностной производной

Она используется, например, для анализа перегибов функции или для определения скорости изменения функции в окрестности данной точки.

Ниже приведена реализация метода на языке Python:

def central\_diff(f, x, h):  
 return (f(x + h) - f(x - h))/(2\*h)

**Задание №2. Возьмите 2 произвольные функции. Вычислите аналитически производные**

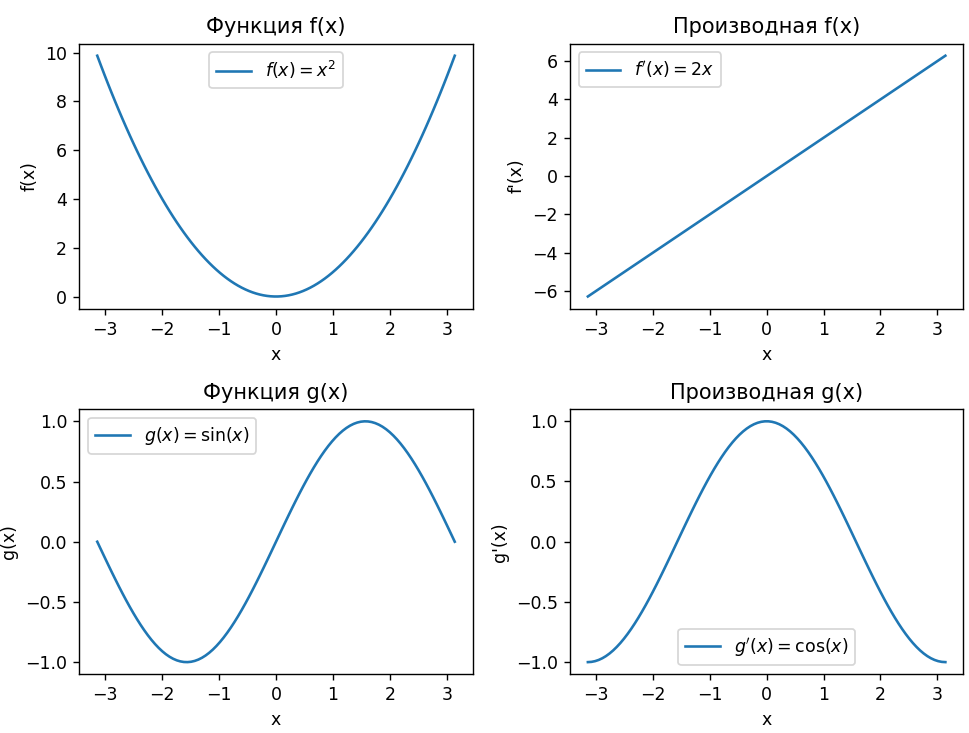
**этих функций. Постройте их графики, а также вычисленные значения численной производной в узлах сетки.**

**Решение:**

Возьмем функцию и , где , h = 0.1

Ниже приведен код и графики:

import numpy as np  
import matplotlib.pyplot as plt  
  
# Определим функции  
def f(x):  
 return x\*\*2  
  
def g(x):  
 return np.sin(x)  
  
# Вычислим аналитические производные  
def df(x):  
 return 2\*x  
  
def dg(x):  
 return np.cos(x)  
  
# Построим графики функций и производных  
x = np.linspace(-np.pi, np.pi, 100)  
fig, ax = plt.subplots(2, 2, figsize=(8, 6))  
  
ax[0, 0].plot(x, f(x), label=r'$f(x) = x^2$')  
ax[0, 0].set\_title('Функция f(x)')  
ax[0, 0].set\_xlabel('x')  
ax[0, 0].set\_ylabel('f(x)')  
ax[0, 0].legend()  
  
ax[0, 1].plot(x, df(x), label=r"$f'(x) = 2x$")  
ax[0, 1].set\_title('Производная f(x)')  
ax[0, 1].set\_xlabel('x')  
ax[0, 1].set\_ylabel("f'(x)")  
ax[0, 1].legend()  
  
ax[1, 0].plot(x, g(x), label=r"$g(x) = \sin(x)$")  
ax[1, 0].set\_title('Функция g(x)')  
ax[1, 0].set\_xlabel('x')  
ax[1, 0].set\_ylabel('g(x)')  
ax[1, 0].legend()  
  
ax[1, 1].plot(x, dg(x), label=r"$g'(x) = \cos(x)$")  
ax[1, 1].set\_title('Производная g(x)')  
ax[1, 1].set\_xlabel('x')  
ax[1, 1].set\_ylabel("g'(x)")  
ax[1, 1].legend()  
  
plt.tight\_layout()  
plt.show()  
  
# Определим функции  
def f(x):  
 return x\*\*2  
  
def g(x):  
 return np.sin(x)  
  
# Зададим интервал и шаг  
a, b = -np.pi, np.pi  
h = 0.1  
  
# Вычислим производные  
x\_grid = np.arange(a, b + h, h)  
  
# Левая разностная производная  
df\_left = (f(x\_grid) - f(x\_grid - h)) / h  
dg\_left = (g(x\_grid) - g(x\_grid - h)) / h  
  
# Правая разностная производная  
df\_right = (f(x\_grid + h) - f(x\_grid)) / h  
dg\_right = (g(x\_grid + h) - g(x\_grid)) / h  
  
# Центральная разностная производная  
df\_central = (f(x\_grid + h) - f(x\_grid - h)) / (2 \* h)  
dg\_central = (g(x\_grid + h) - g(x\_grid - h)) / (2 \* h)  
  
# Выведем результаты  
print("Левые разностные производные функции f(x) в узлах сетки:\n", df\_left)  
print("Правые разностные производные функции f(x) в узлах сетки:\n", df\_right)  
print("Центральные разностные производные функции f(x) в узлах сетки:\n", df\_central)  
  
print("Левые разностные производные функции g(x) в узлах сетки:\n", dg\_left)  
print("Правые разностные производные функции g(x) в узлах сетки:\n", dg\_right)  
print("Центральные разностные производные функции g(x) в узлах сетки:\n", dg\_central)

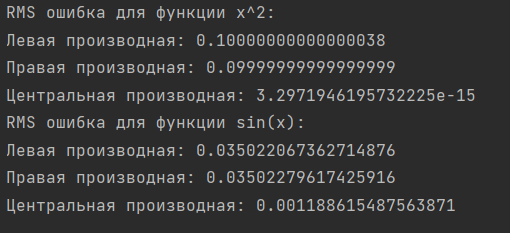


**Задание №3. Найдите среднеквадратичные отклонения численных от истинных значений производной.**

Результирующий код представлен ниже:

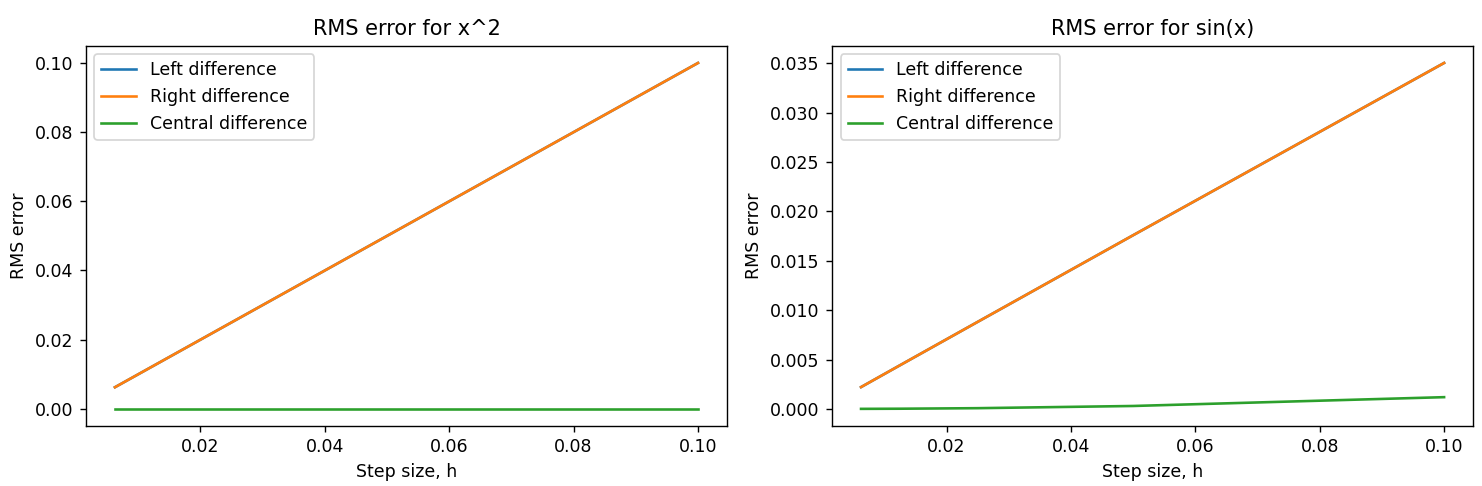
import numpy as np  
import matplotlib.pyplot as plt  
  
# определяем функции  
def f(x):  
 return x\*\*2  
  
def g(x):  
 return np.sin(x)  
  
# определяем аналитические производные функций  
def df(x):  
 return 2\*x  
  
def dg(x):  
 return np.cos(x)  
  
# задаем шаг сетки и диапазон значений x  
h = 0.1  
x\_values = np.arange(-np.pi, np.pi+h, h)  
  
# вычисляем значения функций в узлах сетки  
f\_values = f(x\_values)  
g\_values = g(x\_values)  
  
# вычисляем значения аналитических производных в узлах сетки  
df\_values = df(x\_values)  
dg\_values = dg(x\_values)  
  
# определяем функции для вычисления численных производных  
def left\_difference(f, x, h):  
 return (f(x) - f(x-h))/h  
  
def right\_difference(f, x, h):  
 return (f(x+h) - f(x))/h  
  
def central\_difference(f, x, h):  
 return (f(x+h) - f(x-h))/(2\*h)  
  
# вычисляем значения численных производных в узлах сетки  
df\_left\_values = left\_difference(f, x\_values, h)  
dg\_left\_values = left\_difference(g, x\_values, h)  
  
df\_right\_values = right\_difference(f, x\_values, h)  
dg\_right\_values = right\_difference(g, x\_values, h)  
  
df\_central\_values = central\_difference(f, x\_values, h)  
dg\_central\_values = central\_difference(g, x\_values, h)  
  
# вычисляем значения rms ошибок  
df\_rms\_left = np.sqrt(np.mean((df\_left\_values - df\_values)\*\*2))  
dg\_rms\_left = np.sqrt(np.mean((dg\_left\_values - dg\_values)\*\*2))  
  
df\_rms\_right = np.sqrt(np.mean((df\_right\_values - df\_values)\*\*2))  
dg\_rms\_right = np.sqrt(np.mean((dg\_right\_values - dg\_values)\*\*2))  
  
df\_rms\_central = np.sqrt(np.mean((df\_central\_values - df\_values)\*\*2))  
dg\_rms\_central = np.sqrt(np.mean((dg\_central\_values - dg\_values)\*\*2))  
  
# выводим результаты  
print("RMS ошибка для функции x^2:")  
print("Левая производная:", df\_rms\_left)  
print("Правая производная:", df\_rms\_right)  
print("Центральная производная:", df\_rms\_central)  
  
print("RMS ошибка для функции sin(x):")  
print("Левая производная:", dg\_rms\_left)  
print("Правая производная:", dg\_rms\_right)  
print("Центральная производная:", dg\_rms\_central)

Вывод:

****

**Задание №4. Выполните предыдущий пункт при уменьшении шага (увеличения количества узлов) в 2, 4, 8 и 16. Как изменяется среднеквадратичное отклонение при изменении шага? Постройте график зависимости среднеквадратичного отклонения от величины шага.**

import numpy as np  
import matplotlib.pyplot as plt  
  
# определяем функции  
def f(x):  
 return x\*\*2  
  
def g(x):  
 return np.sin(x)  
  
# определяем аналитические производные функций  
def df(x):  
 return 2\*x  
  
def dg(x):  
 return np.cos(x)  
  
# задаем шаги сетки  
h\_values = [0.1, 0.05, 0.025, 0.0125, 0.00625]  
df\_rms\_left\_values = []  
df\_rms\_right\_values = []  
df\_rms\_central\_values = []  
dg\_rms\_left\_values = []  
dg\_rms\_right\_values = []  
dg\_rms\_central\_values = []  
for h in h\_values:  
 # задаем диапазон значений x  
 x\_values = np.arange(-np.pi, np.pi+h, h)  
  
 # вычисляем значения функций в узлах сетки  
 f\_values = f(x\_values)  
 g\_values = g(x\_values)  
  
 # вычисляем значения аналитических производных в узлах сетки  
 df\_values = df(x\_values)  
 dg\_values = dg(x\_values)  
  
 # определяем функции для вычисления численных производных  
 def left\_difference(f, x, h):  
 return (f(x) - f(x-h))/h  
  
 def right\_difference(f, x, h):  
 return (f(x+h) - f(x))/h  
  
 def central\_difference(f, x, h):  
 return (f(x+h) - f(x-h))/(2\*h)  
  
 # вычисляем значения численных производных в узлах сетки  
 df\_left\_values = left\_difference(f, x\_values, h)  
 dg\_left\_values = left\_difference(g, x\_values, h)  
  
 df\_right\_values = right\_difference(f, x\_values, h)  
 dg\_right\_values = right\_difference(g, x\_values, h)  
  
 df\_central\_values = central\_difference(f, x\_values, h)  
 dg\_central\_values = central\_difference(g, x\_values, h)  
  
 # вычисляем значения rms ошибок  
 df\_rms\_left = np.sqrt(np.mean((df\_left\_values - df\_values)\*\*2))  
 dg\_rms\_left = np.sqrt(np.mean((dg\_left\_values - dg\_values)\*\*2))  
  
 df\_rms\_right = np.sqrt(np.mean((df\_right\_values - df\_values)\*\*2))  
 dg\_rms\_right = np.sqrt(np.mean((dg\_right\_values - dg\_values)\*\*2))  
  
 df\_rms\_central = np.sqrt(np.mean((df\_central\_values - df\_values)\*\*2))  
 dg\_rms\_central = np.sqrt(np.mean((dg\_central\_values - dg\_values)\*\*2))  
  
 # выводим результаты  
 print("Шаг сетки:", h)  
 print("RMS ошибка для функции x^2:")  
 print("Левая производная:", df\_rms\_left)  
 print("Правая производная:", df\_rms\_right)  
 print("Центральная производная:\n", df\_rms\_central)  
  
 print("RMS ошибка для функции sin(x):")  
 print("Левая производная:", dg\_rms\_left)  
 print("Правая производная:", dg\_rms\_right)  
 print("Центральная производная:", dg\_rms\_central)  
 df\_rms\_left\_values.append(df\_rms\_left)  
 df\_rms\_right\_values.append(df\_rms\_right)  
 df\_rms\_central\_values.append(df\_rms\_central)  
 dg\_rms\_left\_values.append(dg\_rms\_left)  
 dg\_rms\_right\_values.append(dg\_rms\_right)  
 dg\_rms\_central\_values.append(dg\_rms\_central)  
# create the figure and subplots  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))  
  
# plot the first graph  
ax1.plot(h\_values, df\_rms\_left\_values, label='Left difference')  
ax1.plot(h\_values, df\_rms\_right\_values, label='Right difference')  
ax1.plot(h\_values, df\_rms\_central\_values, label='Central difference')  
ax1.set\_xlabel('Step size, h')  
ax1.set\_ylabel('RMS error')  
ax1.set\_title('RMS error for x^2')  
ax1.legend()  
  
# plot the second graph  
ax2.plot(h\_values, dg\_rms\_left\_values, label='Left difference')  
ax2.plot(h\_values, dg\_rms\_right\_values, label='Right difference')  
ax2.plot(h\_values, dg\_rms\_central\_values, label='Central difference')  
ax2.set\_xlabel('Step size, h')  
ax2.set\_ylabel('RMS error')  
ax2.set\_title('RMS error for sin(x)')  
ax2.legend()  
  
# adjust the layout and show the plot  
plt.tight\_layout()  
plt.show()



Мы замечаем, что при уменьшении размера сетки среднеквадратичное отклонение значений левых и правых производных уменьшается. Однако центральная производная всегда правильно вычисляет производную функции на заданном интервале.

**5. Реализуйте методы численного интегрирования.**

**Решение:**

Методы численного интегрирования прямоугольника включают в себя следующие подходы

* левых прямоугольников

Ниже представлен код метода на python:

def left\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода левых прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a # начальное значение x  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral

* правых прямоугольников

Ниже представлен код метода на python:

def right\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода правых прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx # начальное значение x  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral

* средних прямоугольников

Ниже представлен код метода на python:

def midpoint\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода средних прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx / 2 # начальное значение x (смещаем на половину интервала)  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral

* трапеций

Ниже представлен код метода на python:

def trapezoidal\_rule(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием формулы трапеций  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx # начальное значение x  
 integral = (f(a) + f(b)) / 2 # значение интеграла (суммируем значения на краях отрезка)  
 for i in range(1, n):  
 integral += f(x) # добавляем значение функции в текущей точке  
 x += dx # переходим к следующей точке  
 integral \*= dx # умножаем на длину интервала  
 return integral

* Симпсона

Ниже представлен код метода на python:

def simpson\_method(f, a, b, n):  
 *"""  
 Вычисляет значение определенного интеграла на интервале [a, b] с использованием метода Симпсона.  
  
 f - функция, интеграл которой нужно вычислить  
 a, b - границы интервала интегрирования  
 n - количество отрезков, на которые нужно разбить интервал [a, b]  
 """* h = (b - a) / n  
 s = f(a) + f(b)  
  
 for i in range(1, n):  
 x = a + i \* h  
 if i % 2 == 0:  
 s += 2 \* f(x)  
 else:  
 s += 4 \* f(x)  
  
 return h / 3 \* s

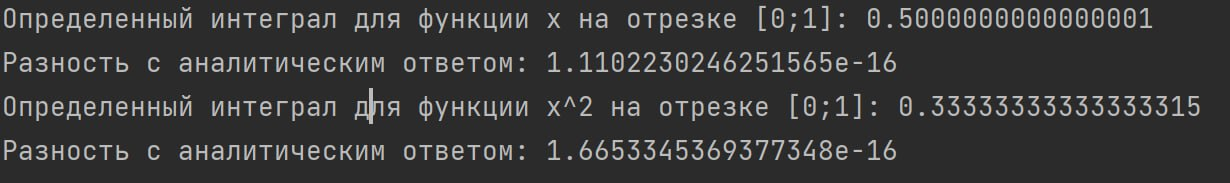
**6. Выберите 2 функции и вычислите для них определенный интеграл на отрезке. Сравните полученное значение с ответом, полученным аналитически.**

**Решение:**

Ниже представлен код для метода Симпсона:

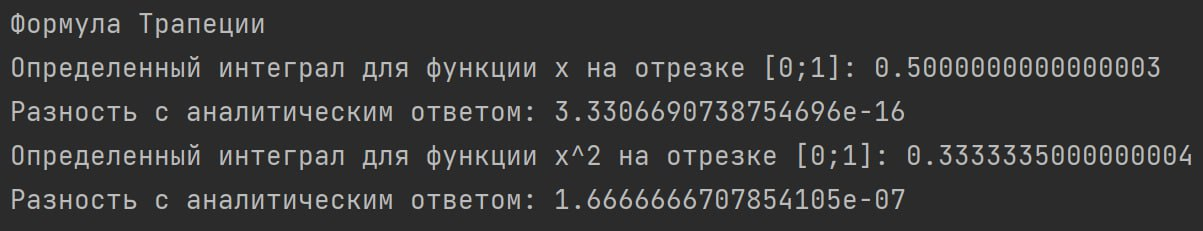
def simpson\_method(f, a, b, n):  
 *"""  
 Вычисляет значение определенного интеграла на интервале [a, b] с использованием метода Симпсона.  
  
 f - функция, интеграл которой нужно вычислить  
 a, b - границы интервала интегрирования  
 n - количество отрезков, на которые нужно разбить интервал [a, b]  
 """* h = (b - a) / n  
 s = f(a) + f(b)  
  
 for i in range(1, n):  
 x = a + i \* h  
 if i % 2 == 0:  
 s += 2 \* f(x)  
 else:  
 s += 4 \* f(x)  
  
 return h / 3 \* s  
  
def f\_x(x):  
 return x  
  
def f\_x2(x):  
 return x\*\*2  
  
a, b = 0, 1  
n = 1000  
  
# Вычисляем определенный интеграл функции x  
integral\_x = simpson\_method(f\_x, a, b, n)  
print("Определенный интеграл для функции x на отрезке [0;1]:", integral\_x)  
# Сравниваем с аналитическим ответом  
analytic\_x = 1/2  
difference\_x = abs(integral\_x - analytic\_x)  
print("Разность с аналитическим ответом:", difference\_x)  
  
# Вычисляем определенный интеграл функции x^2  
integral\_x2 = simpson\_method(f\_x2, a, b, n)  
print("Определенный интеграл для функции x^2 на отрезке [0;1]:", integral\_x2)  
# Сравниваем с аналитическим ответом  
analytic\_x2 = 1/3  
difference\_x2 = abs(integral\_x2 - analytic\_x2)  
print("Разность с аналитическим ответом:", difference\_x2)

Ниже представлен вывод программы:



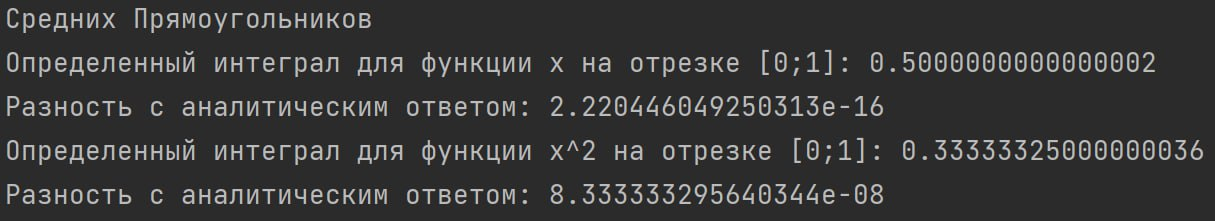
Ниже представлен код для метода трапеций:

import numpy as np  
from scipy.integrate import quad  
import sympy  
import matplotlib.pyplot as plt  
  
def trapezoidal\_rule(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием формулы трапеций  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx # начальное значение x  
 integral = (f(a) + f(b)) / 2 # значение интеграла (суммируем значения на краях отрезка)  
 for i in range(1, n):  
 integral += f(x) # добавляем значение функции в текущей точке  
 x += dx # переходим к следующей точке  
 integral \*= dx # умножаем на длину интервала  
 return integral  
  
def f\_x(x):  
 return x  
  
def f\_x2(x):  
 return x\*\*2  
  
a, b = 0, 1  
n = 1000  
  
# Вычисляем определенный интеграл функции x  
integral\_x = trapezoidal\_rule(f\_x, a, b, n)  
print("Определенный интеграл для функции x на отрезке [0;1]:", integral\_x)  
# Сравниваем с аналитическим ответом  
analytic\_x = 1/2  
difference\_x = abs(integral\_x - analytic\_x)  
print("Разность с аналитическим ответом:", difference\_x)  
  
# Вычисляем определенный интеграл функции x^2  
integral\_x2 = trapezoidal\_rule(f\_x2, a, b, n)  
print("Определенный интеграл для функции x^2 на отрезке [0;1]:", integral\_x2)  
# Сравниваем с аналитическим ответом  
analytic\_x2 = 1/3  
difference\_x2 = abs(integral\_x2 - analytic\_x2)  
print("Разность с аналитическим ответом:", difference\_x2)



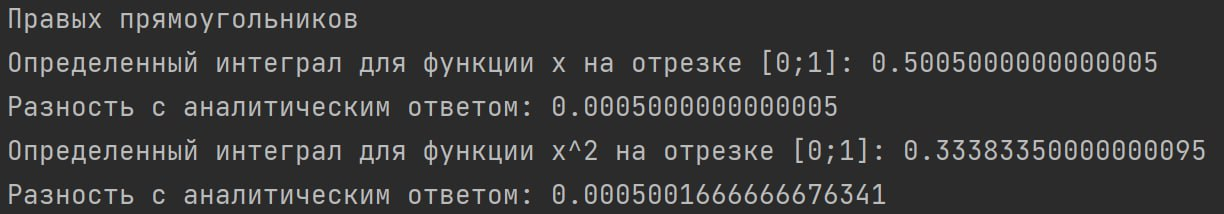
Ниже представлен код для метода средних прямоугольников:

import numpy as np  
from scipy.integrate import quad  
import sympy  
import matplotlib.pyplot as plt  
def midpoint\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода средних прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx / 2 # начальное значение x (смещаем на половину интервала)  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral  
  
def f\_x(x):  
 return x  
  
def f\_x2(x):  
 return x\*\*2  
  
a, b = 0, 1  
n = 1000  
  
# Вычисляем определенный интеграл функции x  
integral\_x = midpoint\_rectangle\_method(f\_x, a, b, n)  
print("Определенный интеграл для функции x на отрезке [0;1]:", integral\_x)  
# Сравниваем с аналитическим ответом  
analytic\_x = 1/2  
difference\_x = abs(integral\_x - analytic\_x)  
print("Разность с аналитическим ответом:", difference\_x)  
  
# Вычисляем определенный интеграл функции x^2  
integral\_x2 = midpoint\_rectangle\_method(f\_x2, a, b, n)  
print("Определенный интеграл для функции x^2 на отрезке [0;1]:", integral\_x2)  
# Сравниваем с аналитическим ответом  
analytic\_x2 = 1/3  
difference\_x2 = abs(integral\_x2 - analytic\_x2)  
print("Разность с аналитическим ответом:", difference\_x2)



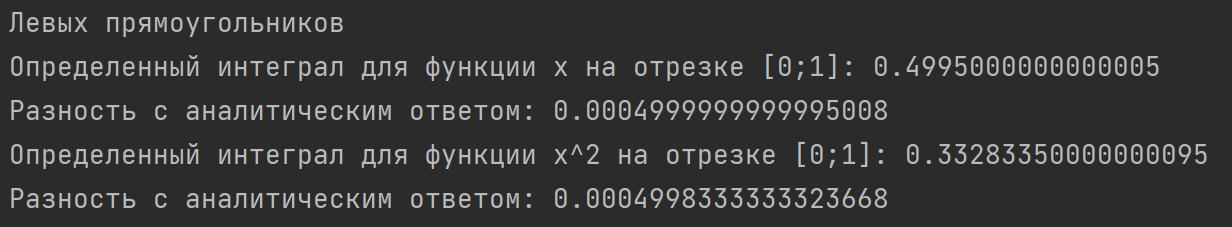
Ниже представлен код для метода правых прямоугольников:

import numpy as np  
from scipy.integrate import quad  
import sympy  
import matplotlib.pyplot as plt  
def right\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода правых прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx # начальное значение x  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral  
  
  
def f\_x(x):  
 return x  
  
def f\_x2(x):  
 return x\*\*2  
  
a, b = 0, 1  
n = 1000  
  
# Вычисляем определенный интеграл функции x  
integral\_x = right\_rectangle\_method(f\_x, a, b, n)  
print("Определенный интеграл для функции x на отрезке [0;1]:", integral\_x)  
# Сравниваем с аналитическим ответом  
analytic\_x = 1/2  
difference\_x = abs(integral\_x - analytic\_x)  
print("Разность с аналитическим ответом:", difference\_x)  
  
# Вычисляем определенный интеграл функции x^2  
integral\_x2 = right\_rectangle\_method(f\_x2, a, b, n)  
print("Определенный интеграл для функции x^2 на отрезке [0;1]:", integral\_x2)  
# Сравниваем с аналитическим ответом  
analytic\_x2 = 1/3  
difference\_x2 = abs(integral\_x2 - analytic\_x2)  
print("Разность с аналитическим ответом:", difference\_x2)



Ниже представлен код для метода левых прямоугольников:

import numpy as np  
from scipy.integrate import quad  
import sympy  
import matplotlib.pyplot as plt  
def left\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода левых прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a # начальное значение x  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral  
  
  
  
def f\_x(x):  
 return x  
  
def f\_x2(x):  
 return x\*\*2  
  
a, b = 0, 1  
n = 1000  
  
# Вычисляем определенный интеграл функции x  
integral\_x = left\_rectangle\_method(f\_x, a, b, n)  
print("Определенный интеграл для функции x на отрезке [0;1]:", integral\_x)  
# Сравниваем с аналитическим ответом  
analytic\_x = 1/2  
difference\_x = abs(integral\_x - analytic\_x)  
print("Разность с аналитическим ответом:", difference\_x)  
  
# Вычисляем определенный интеграл функции x^2  
integral\_x2 = left\_rectangle\_method(f\_x2, a, b, n)  
print("Определенный интеграл для функции x^2 на отрезке [0;1]:", integral\_x2)  
# Сравниваем с аналитическим ответом  
analytic\_x2 = 1/3  
difference\_x2 = abs(integral\_x2 - analytic\_x2)  
print("Разность с аналитическим ответом:", difference\_x2)



**7. Проанализируйте зависимость отклонения численного ответа от аналитического в зависимости от шага при уменьшении его в 2, 4, 8 и 16 раз. Постройте график зависимости отклонения от величины шага.**

**Решение:**

import numpy as np  
from scipy.integrate import quad  
import sympy  
import matplotlib.pyplot as plt  
def simpson\_method(f, a, b, n):  
 *"""  
 Вычисляет значение определенного интеграла на интервале [a, b] с использованием метода Симпсона.  
  
 f - функция, интеграл которой нужно вычислить  
 a, b - границы интервала интегрирования  
 n - количество отрезков, на которые нужно разбить интервал [a, b]  
 """* h = (b - a) / n  
 s = f(a) + f(b)  
  
 for i in range(1, n):  
 x = a + i \* h  
 if i % 2 == 0:  
 s += 2 \* f(x)  
 else:  
 s += 4 \* f(x)  
  
 return h / 3 \* s  
  
  
def f\_x(x):  
 return x  
  
  
def f\_x2(x):  
 return x \*\* 2  
  
  
def trapezoidal\_rule(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием формулы трапеций  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx # начальное значение x  
 integral = (f(a) + f(b)) / 2 # значение интеграла (суммируем значения на краях отрезка)  
 for i in range(1, n):  
 integral += f(x) # добавляем значение функции в текущей точке  
 x += dx # переходим к следующей точке  
 integral \*= dx # умножаем на длину интервала  
 return integral  
  
def midpoint\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода средних прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx / 2 # начальное значение x (смещаем на половину интервала)  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral  
  
def right\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода правых прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a + dx # начальное значение x  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral  
  
def left\_rectangle\_method(f, a, b, n):  
 *"""  
 Вычисление интеграла функции f(x) на отрезке [a,b] с использованием метода левых прямоугольников  
 и разбиением отрезка на n равных частей  
 """* dx = (b - a) / n # длина интервала  
 x = a # начальное значение x  
 integral = 0 # значение интеграла  
 for i in range(n):  
 integral += f(x) \* dx # добавляем площадь прямоугольника  
 x += dx # переходим к следующей точке  
 return integral  
  
functions = [(f\_x, 1/2), (f\_x2, 1/3)]  
  
a, b = 0, 1  
n = 1000  
ns = []  
differences = []  
for i in range(5):  
 for f, analytic\_solution in functions:  
 integral\_simpson = simpson\_method(f, a, b, n)  
 difference\_simpson = abs(integral\_simpson - analytic\_solution)  
 integral\_trapezoidal = trapezoidal\_rule(f, a, b, n)  
 difference\_trapezoidal = abs(integral\_trapezoidal - analytic\_solution)  
 integral\_midpoint\_rectangle\_method = midpoint\_rectangle\_method(f, a, b, n)  
 difference\_midpoint\_rectangle\_method = abs(integral\_midpoint\_rectangle\_method - analytic\_solution)  
 integral\_right\_rectangle\_method = right\_rectangle\_method(f, a, b, n)  
 difference\_right\_rectangle\_method = abs(integral\_right\_rectangle\_method - analytic\_solution)  
 integral\_left\_rectangle\_method = left\_rectangle\_method(f, a, b, n)  
 difference\_left\_rectangle\_method = abs(integral\_left\_rectangle\_method - analytic\_solution)  
 print(f"Функция: {f.\_\_name\_\_}")  
 print(f"Аналитическое решение: {analytic\_solution}")  
 print(f"Определенный интеграл методом Симпсона: {integral\_simpson}, разность: {difference\_simpson}")  
 print(f"Определенный интеграл формулой трапеций: {integral\_trapezoidal}, разность: {difference\_trapezoidal}")  
 print(f"Определенный интеграл формулой средних прямоугольников: {integral\_midpoint\_rectangle\_method}, разность: {difference\_midpoint\_rectangle\_method}")  
 print(f"Определенный интеграл формулой правых прямоугольников: {integral\_right\_rectangle\_method}, разность: {difference\_right\_rectangle\_method}")  
 print(f"Определенный интеграл формулой левых прямоугольников: {integral\_left\_rectangle\_method}, разность: {difference\_left\_rectangle\_method}")  
 print()  
 ns.append(n)  
 # differences.append(difference\_simpson)  
 # differences.append(difference\_trapezoidal)  
 differences.append(difference\_midpoint\_rectangle\_method)  
 # differences.append(difference\_right\_rectangle\_method)  
 # differences.append(difference\_left\_rectangle\_method)  
 n \*= 2  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))  
  
# plot the first graph  
ax1.plot(ns, differences, label='Left difference')  
ax1.plot(ns, differences, label='Right difference')  
ax1.set\_xlabel('Step size, h')  
ax1.set\_ylabel('RMS error')  
ax1.set\_title('RMS error for x^2')  
ax1.legend()  
  
# plot the second graph  
ax2.plot(ns, differences, label='Left difference')  
ax2.plot(ns, differences, label='Right difference')  
ax2.set\_xlabel('Step size, h')  
ax2.set\_ylabel('RMS error')  
ax2.set\_title('RMS error for sin(x)')  
ax2.legend()  
  
# adjust the layout and show the plot  
plt.tight\_layout()  
plt.show()