***Санкт-Петербургский национальный исследовательский университет ИТМО***

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Вычислительная математика»

**Отчет**

По лабораторной работе №3

Вариант 9

Выполнила:

Миличенко А.Е.

P3210

Преподаватель:

Малышева Т.А.

Санкт-Петербург

2024 г.

**Цель работы**

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

**Листинг программы**

Файл main.py:

import numpy as np

import matplotlib.pyplot as plt

#from prettytable import PrettyTable

def read\_data\_from\_file(file\_name):

with open(file\_name, 'r') as file:

data = [line.split() for line in file.readlines()]

x = [float(item[0]) for item in data]

y = [float(item[1]) for item in data]

return x, y

def input\_data\_from\_console():

x = []

y = []

n = int(input("Введите количество точек (от 8 до 12): "))

for i in range(n):

xi, yi = map(float, input(f"Введите координаты точки {i + 1}: ").split())

x.append(xi)

y.append(yi)

return x, y

def linear\_approximation(x, y):

SX: float = sum([p for p in x])

SXX: float = sum([p \*\* 2 for p in x])

SY: float = sum([p for p in y])

SXY: float = sum([x[i] \* y[i] for i in range(len(x))])

delta = SXX \* len(x) - SX \* SX

delta1 = SXY \* len(x) - SX \* SY

delta2 = SXX \* SY - SX \* SXY

a = delta1 / delta

b = delta2 / delta

func = lambda x: a \* x + b

return func, a, b

def polynomial\_approximation\_2(x, y):

SX = sum([p for p in x])

SX2 = sum([p \*\* 2 for p in x])

SX3 = sum([p \*\* 3 for p in x])

SX4 = sum([p \*\* 4 for p in x])

SY = sum([p for p in y])

SXY = sum([x[i] \* y[i] for i in range(len(x))])

SX2Y = sum([x[i] \* x[i] \* y[i] for i in range(len(x))])

x1 = np.array([[len(x), SX, SX2], [SX, SX2, SX3], [SX2, SX3, SX4]])

y1 = np.array([SY, SXY, SX2Y])

a = np.linalg.solve(x1, y1)

func = lambda x: a[2] \* x \*\* 2 + a[1] \* x + a[0]

return func, a[2], a[1], a[0]

def polynomial\_approximation\_3(x, y):

SX = sum([p for p in x])

SX2 = sum([p \*\* 2 for p in x])

SX3 = sum([p \*\* 3 for p in x])

SX4 = sum([p \*\* 4 for p in x])

SX5 = sum([p \*\* 5 for p in x])

SX6 = sum([p \*\* 6 for p in x])

SY = sum([p for p in y])

SXY = sum([x[i] \* y[i] for i in range(len(x))])

SX2Y = sum([x[i] \* x[i] \* y[i] for i in range(len(x))])

SX3Y = sum([x[i] \* x[i] \* x[i] \* y[i] for i in range(len(x))])

x = np.array([[len(x), SX, SX2, SX3], [SX, SX2, SX3, SX4], [SX2, SX3, SX4, SX5], [SX3, SX4, SX5, SX6]])

y = np.array([SY, SXY, SX2Y, SX3Y])

a = np.linalg.solve(x, y)

func = lambda x: a[3] \* x \*\* 3 + a[2] \* x \*\* 2 + a[1] \* x + a[0]

return func, a[3], a[2], a[1], a[0]

def polynomial\_approximation\_4(x, y):

SX = sum([p for p in x])

SX2 = sum([p \*\* 2 for p in x])

SX3 = sum([p \*\* 3 for p in x])

SX4 = sum([p \*\* 4 for p in x])

SX5 = sum([p \*\* 5 for p in x])

SX6 = sum([p \*\* 6 for p in x])

SX7 = sum([p \*\* 7 for p in x])

SX8 = sum([p \*\* 8 for p in x])

SY = sum([p for p in y])

SXY = sum([x[i] \* y[i] for i in range(len(x))])

SX2Y = sum([x[i] \* x[i] \* y[i] for i in range(len(x))])

SX3Y = sum([x[i] \* x[i] \* x[i] \* y[i] for i in range(len(x))])

SX4Y = sum([x[i] \* x[i] \* x[i] \* x[i] \* y[i] for i in range(len(x))])

x = np.array([

[len(x), SX, SX2, SX3, SX4],

[SX, SX2, SX3, SX4, SX5],

[SX2, SX3, SX4, SX5, SX6],

[SX3, SX4, SX5, SX6, SX7],

[SX4, SX5, SX6, SX7, SX8]

])

y = np.array([

SY,

SXY,

SX2Y,

SX3Y,

SX4Y

])

a = np.linalg.solve(x, y)

func = lambda x: a[4] \* x \*\* 4 + a[3] \* x \*\* 3 + a[2] \* x \*\* 2 + a[1] \* x + a[0]

return func, a[4], a[3], a[2], a[1], a[0]

def exponential\_approximation(x, y):

y\_ln = [np.log(p) for p in y]

\_, b1, a1 = linear\_approximation(x, y\_ln)

a = np.exp(a1)

b = b1

func = lambda x: a \* np.exp(b \* x)

return func, a, b

def logarithmic\_approximation(x, y):

if not all([p > 0 for p in x]):

return None, None, None

x\_ln = [np.log(p) for p in x]

\_, a1, b1 = linear\_approximation(x\_ln, y)

func = lambda x: a1 \* np.log(x) + b1

return func, a1, b1

def power\_approximation(x, y):

if not (all([p > 0 for p in x]) and all([p > 0 for p in y])):

return None, None, None

x\_ln = [np.log(p) for p in x]

y\_ln = [np.log(p) for p in y]

\_, b1, a1 = linear\_approximation(x\_ln, y\_ln)

a = np.exp(a1)

b = b1

func = lambda x: a \* x \*\* b

return func, a, b

def get\_S(f, x, y):

return sum([(f(x[i]) - y[i]) \*\* 2 for i in range(len(x))])

def mean\_squared\_error(x, y, func):

return np.sqrt(get\_S(func, x, y) / len(x))

def pearson\_correlation\_coefficient(x, y):

x\_ = sum([p for p in x]) / len(x)

y\_ = sum([p for p in y]) / len(y)

return (sum([(x[i] - x\_) \* (y[i] - y\_) for i in range(len(x))]) /

np.sqrt(sum([(p - x\_) \*\* 2 for p in x]) \* sum([(p - y\_) \*\* 2 for p in y])))

def coefficient\_of\_determination(x, y, func):

phi = sum([p for p in x]) / len(x)

return 1 - (sum([(y[i] - func(x[i])) \*\* 2 for i in range(len(x))]) / sum([(p - phi) \*\* 2 for p in y]))

def write\_results\_to\_file(file\_name, approximation, coefficients, mse, r\_squared):

with open(file\_name, 'w') as file:

file.write(f'Наилучшая аппроксимирующая функция: {approximation}\n')

file.write(f'Коэффициенты аппроксимирующей функции: {coefficients}\n')

file.write(f'Среднеквадратичное отклонение: {mse}\n')

file.write(f'Коэффициент детерминации: {r\_squared}\n')

def main():

choice = input("Выберите способ ввода данных f/c: ")

if choice == "f":

file\_name = input("Введите имя файла: ")

x, y = read\_data\_from\_file(file\_name)

else:

x, y = input\_data\_from\_console()

f1, a1, b1 = linear\_approximation(x, y)

f2, a2, b2, c2 = polynomial\_approximation\_2(x, y)

f3, a3, b3, c3, d3 = polynomial\_approximation\_3(x, y)

f4, a4, b4 = exponential\_approximation(x, y)

f5, a5, b5 = logarithmic\_approximation(x, y)

f6, a6, b6 = power\_approximation(x, y)

f7, a7, b7, c7, d7, e7 = polynomial\_approximation\_4(x, y)

f = [f1, f2, f3, f4, f5, f6]

v = ["ax + b", "ax^2 + bx + c", "ax^3 + bx^2 + cx + d", "ae^(bx)", "alnx + b", "ax^b"]

a = [a1, a2, a3, a4, a5, a6]

b = [b1, b2, b3, b4, b5, b6]

c = ["-", c2, c3, "-", "-", "-"]

d = ["-", "-", d3, "-", "-", "-"]

titles = ["Линейная аппроксимация", "Квадратичная аппроксимация", "Кубическая аппроксимация",

"Экспоненциальная аппроксимация", "Логарифмическая аппроксимация", "Степенная аппроксимация"]

for i in range(6):

if f[i] is None:

continue

print(titles[i])

print("{:<3} {:<10} {:<10} {:<10}".format("№", "X", "Y", "P", "eps"))

for j in range(len(x)):

print("{:<3} {:<10.3f} {:<10.3f} {:<10.3f} {:<10.3f}".format(j + 1, x[j], y[j], f[i](x[j]), f[i](x[j]) - y[j]))

r2 = coefficient\_of\_determination(x, y, f[i])

print(f"Коэффициент детерминации: {r2:.5f}")

if r2 < 0.5:

print("Точность аппроксимации недостаточна")

elif r2 < 0.75:

print("Слабая аппроксимация")

elif r2 < 0.95:

print("Удовлетворительная аппроксимация")

else:

print("Высокая точность аппроксимации")

if i == 0:

pr = pearson\_correlation\_coefficient(x, y)

print(f"Коэффициент Пирсона: {pr:.5f}")

if pr == 0:

print("Связь между переменными отсутствует")

elif pr == 1 or pr == -1:

print("Строгая линейная зависимость")

elif pr < 0.3:

print("Связь слабая")

elif pr < 0.5:

print("Связь умеренная")

elif pr < 0.7:

print("Связь заметная")

elif pr < 0.9:

print("Связь высокая")

else:

print("Связь весьма высокая")

linear\_mse = mean\_squared\_error(x, y, f1)

polynomial\_2\_mse = mean\_squared\_error(x, y, f2)

polynomial\_3\_mse = mean\_squared\_error(x, y, f3)

exponential\_mse = mean\_squared\_error(x, y, f4)

if f5 is not None:

logarithmic\_mse = mean\_squared\_error(x, y, f5)

else:

logarithmic\_mse = np.inf

if f6 is not None:

power\_mse = mean\_squared\_error(x, y, f6)

else:

power\_mse = np.inf

approximations = ["Линейная", "Квадратичная", "Кубическая", "Экспоненциальная", "Логарифмическая", "Степенная"]

mses = [linear\_mse, polynomial\_2\_mse, polynomial\_3\_mse, exponential\_mse, logarithmic\_mse, power\_mse]

best\_approximation\_index = np.argmin(mses)

print(f"Наилучшая аппроксимирующая функция: {approximations[best\_approximation\_index]}")

plt.figure(figsize=(10, 6))

plt.scatter(x, y, label='Исходные данные')

x\_ = np.linspace(min(x) - 1, max(x) + 1, 1000)

for i in range(6):

if f[i] is None:

continue

yi = np.array([f[i](a) for a in x\_])

plt.plot(x\_, yi, label=titles[i])

plt.legend()

plt.xlabel('X')

plt.ylabel('Y')

plt.grid()

plt.show()

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

main()

Файл methods.py:

def rectangle\_left(func, a, b, n):

h = (b - a) / n

result = 0

for i in range(n):

result += func(a + i \* h)

result \*= h

return result

def rectangle\_middle(func, a, b, n):

h = (b - a) / n

result = 0

for i in range(n):

result += func(a + (i + 0.5) \* h)

result \*= h

return result

def rectangle\_right(func, a, b, n):

h = (b - a) / n

result = 0

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

result += func(a + i \* h)

result \*= h

return result

def trapezoid\_rule(func, a, b, n):

h = (b - a) / n

result = (func(a) + func(b)) / 2

for i in range(1, n):

result += func(a + i \* h)

result \*= h

return result

def simpson(func, a, b, n):

h = (b - a) / n

result = func(a) + func(b)

for i in range(1, n):

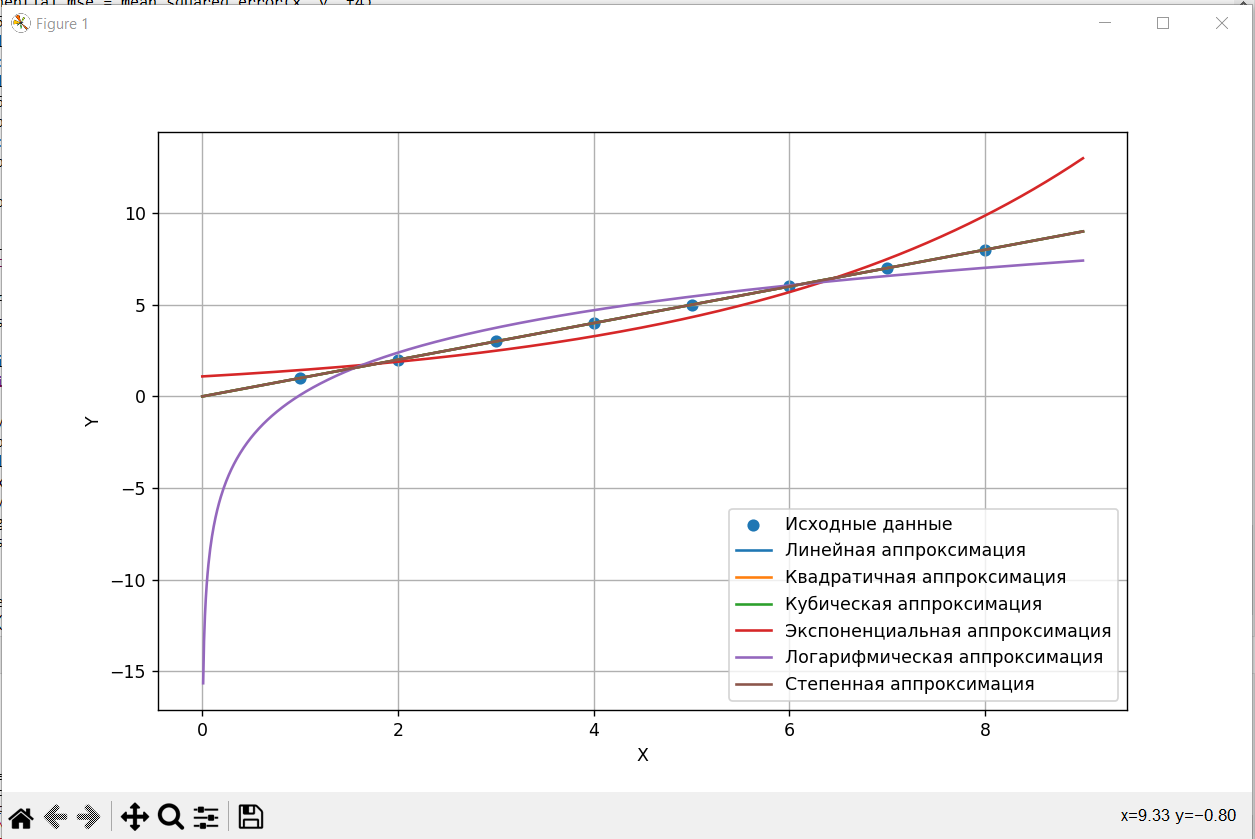
coef = 3 + (-1)\*\*(i + 1)

result += coef \* func(a + i \* h)

result \*= h / 3

return result

**Результат работы программы**



Выберите способ ввода данных f/c: c

Введите количество точек (от 8 до 12): 8

Введите координаты точки 1: 1 1

Введите координаты точки 2: 2 2

Введите координаты точки 3: 3 3

Введите координаты точки 4: 4 4

Введите координаты точки 5: 5 5

Введите координаты точки 6: 6 6

Введите координаты точки 7: 7 7

Введите координаты точки 8: 8 8

Линейная аппроксимация

№ X Y P

1 1.000 1.000 1.000 0.000

2 2.000 2.000 2.000 0.000

3 3.000 3.000 3.000 0.000

4 4.000 4.000 4.000 0.000

5 5.000 5.000 5.000 0.000

6 6.000 6.000 6.000 0.000

7 7.000 7.000 7.000 0.000

8 8.000 8.000 8.000 0.000

Коэффициент детерминации: 1.00000

Высокая точность аппроксимации

Коэффициент Пирсона: 1.00000

Строгая линейная зависимость

Квадратичная аппроксимация

№ X Y P

1 1.000 1.000 1.000 0.000

2 2.000 2.000 2.000 0.000

3 3.000 3.000 3.000 0.000

4 4.000 4.000 4.000 0.000

5 5.000 5.000 5.000 0.000

6 6.000 6.000 6.000 0.000

7 7.000 7.000 7.000 0.000

8 8.000 8.000 8.000 0.000

Коэффициент детерминации: 1.00000

Высокая точность аппроксимации

Кубическая аппроксимация

№ X Y P

1 1.000 1.000 1.000 -0.000

2 2.000 2.000 2.000 0.000

3 3.000 3.000 3.000 0.000

4 4.000 4.000 4.000 0.000

5 5.000 5.000 5.000 0.000

6 6.000 6.000 6.000 -0.000

7 7.000 7.000 7.000 -0.000

8 8.000 8.000 8.000 0.000

Коэффициент детерминации: 1.00000

Высокая точность аппроксимации

Экспоненциальная аппроксимация

№ X Y P

1 1.000 1.000 1.436 0.436

2 2.000 2.000 1.892 -0.108

3 3.000 3.000 2.491 -0.509

4 4.000 4.000 3.280 -0.720

5 5.000 5.000 4.320 -0.680

6 6.000 6.000 5.689 -0.311

7 7.000 7.000 7.491 0.491

8 8.000 8.000 9.865 1.865

Коэффициент детерминации: 0.87478

Удовлетворительная аппроксимация

Логарифмическая аппроксимация

№ X Y P

1 1.000 1.000 0.075 -0.925

2 2.000 2.000 2.389 0.389

3 3.000 3.000 3.742 0.742

4 4.000 4.000 4.703 0.703

5 5.000 5.000 5.448 0.448

6 6.000 6.000 6.056 0.056

7 7.000 7.000 6.571 -0.429

8 8.000 8.000 7.017 -0.983

Коэффициент детерминации: 0.91889

Удовлетворительная аппроксимация

Степенная аппроксимация

№ X Y P

1 1.000 1.000 1.000 0.000

2 2.000 2.000 2.000 0.000

3 3.000 3.000 3.000 0.000

4 4.000 4.000 4.000 0.000

5 5.000 5.000 5.000 0.000

6 6.000 6.000 6.000 0.000

7 7.000 7.000 7.000 0.000

8 8.000 8.000 8.000 0.000

Коэффициент детерминации: 1.00000

Высокая точность аппроксимации

Наилучшая аппроксимирующая функция: Линейная

**Вывод**

В процессе работы над задачей нахождения приближенного значения определенного интеграла с заданной точностью мы изучили и применили несколько численных методов, включая метод прямоугольников, метод трапеций, метод Симпсона и метод Гаусса.