Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №4

Вариант 3

Группа: P3269

Выполнили:

Грибкова В.Е

Долганова О.А

Проверил:

Машина Е. А.

Г. Санкт-Петербург

1. Цель лабораторной работы: найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.
2. Рабочие формулы используемых методов.
3. Вычислительная часть.
   Функция:

$$y = \frac{4x}{x^4 + 3}$$

   Исследуемый интервал:

$$x \in [-2, 0] \quad h = 0{,}2$$

   1. Сформируем таблицу табулирования заданной функции на указанном интервале

x	y
-2.00	-0.421
-1.80	-0.533
-1.60	-0.670
-1.40	-0.819
-1.20	-0.946
-1.00	-1.000
-0.80	-0.939
-0.60	-0.767
-0.40	-0.529
-0.20	-0.267
0	0.000

   2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала.

      Используем метод наименьших квадратов для нахождения коэффициентов линейной и квадратичной функций, которые будут наилучшим образом приближать заданные точки.

Для построения **линейного** приближения воспользуемся формулой уравнения прямой y = ax + b, где m - коэффициент наклона прямой, b - коэффициент сдвига по оси y.
Эти коэффициенты можно найти, используя формулы:

a = (n∑xy - ∑x∑y) / (n∑x^2 - (∑x)^2)
b = (∑y - m∑x) / n

где n - количество точек, x и y - координаты точек.

a = (11*7.632 - (-11)*(-6,891)) / (11*15,4 - (-11)^2)=0.168
b = ((-6,891) - 0.168*(-11)) / 11=-0.458

Уравнения прямой y1 = 0.168*x -0.458

Для построения **квадратичного** приближения воспользуемся формулой уравнения параболы y = ax^2 + bx + c, где a, b, c - коэффициенты уравнения параболы. Эти коэффициенты можно найти, используя формулы:

a = (n∑xy - ∑x∑y) / (n∑x^2 - (∑x)^2)
b = (∑y∑x^2 - ∑x∑xy) / (n∑x^2 - (∑x)^2)
c = (∑y - a∑x^2 - b∑x) / n

a = (11*7.632 - (-11)*(-6,891)) / (11*15,4 -(-11)^2)=0.168
b = ((-6,891)*15,4 -(-11)*7.632) / (11*15,4 - (-11)^2)=-0,458
c = ((-6,891) - 0.168*15,4 - (-0,458)*(-11)) / 11=-1,320

Уравнение параболы y2= 0.168*x^2 -0,458*x -1,320

3. Найти среднеквадратические отклонения для каждой аппроксимирующей функции. Выбрать наилучшее приближение

	x	y	y1 (линейная)	y2 (квадр)	(y1-y)^2	(y2-y)^2
	-2.00	-0.421	-0,794	0,268	0,139	0,475
	-1.80	-0.533	-0,760	0,049	0,052	0,338
	-1.60	-0.670	-0,727	-0,157	0,003	0,263
	-1.40	-0.819	-0,693	-0,350	0,016	0,220
	-1.20	-0.946	-0,660	-0,528	0,082	0,174
	-1.00	-1.000	-0,626	-0,694	0,140	0,094

	-0.80	-0.939	-0,592	-0,846	0,120	0,009
	-0.60	-0.767	-0,559	-0,985	0,043	0,047
	-0.40	-0.529	-0,525	-1,110	0,000	0,337
	-0.20	-0.267	-0,492	-1,222	0,050	0,911
	0	0.000	-0,458	-1,320	0,210	1,742
Σ					0,855	4,612

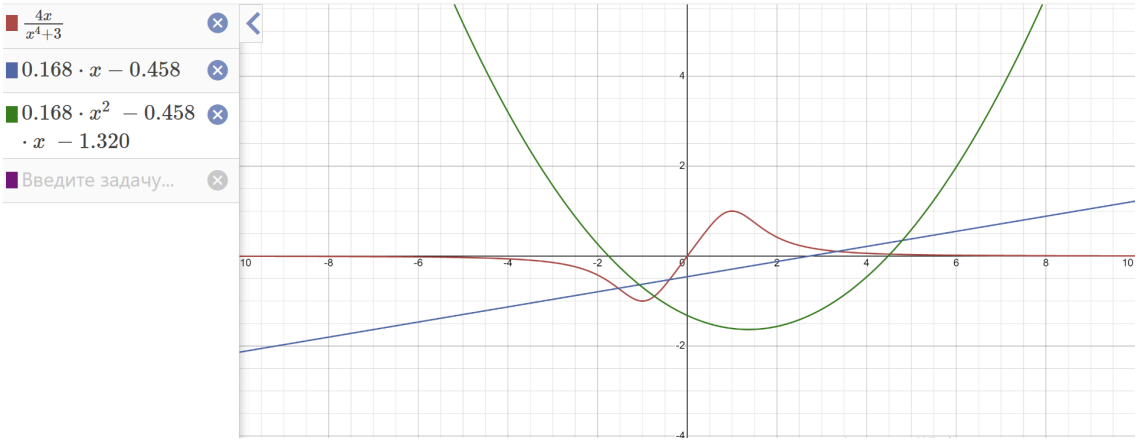
СКО:

для линейной функции: $\sqrt{\frac{\Sigma(y1-y)^2}{11}} = \sqrt{\frac{0,855*0,855}{11}} = 0,258$

для квадратичной функции: $\sqrt{\frac{\Sigma(y2-y)^2}{11}} = \sqrt{\frac{4,612*4,612}{11}} = 1,391$

Из расчетов следует, что наилучшее приближение у линейной аппроксимирующей функции

4. Построим графики заданной функции, а также полученные линейное и квадратичное приближения



4. Листинг программы.

```
import inspect
from math import sqrt, exp, log
import matplotlib.pyplot as plt

# Функция для вычисления определителя 2x2 матрицы
def calculate_determinant_2x2(matrix):
    return matrix[0][0] * matrix[1][1] - matrix[0][1] *
matrix[1][0]

# Функция для решения системы линейных уравнений 2x2
```

```
def solve_linear_system_2x2(matrix, constants):
    n = 2
    det = calculate_determinant_2x2(matrix)
    det1 = calculate_determinant_2x2([[constants[r], matrix[r][1]]
for r in range(n)])
    det2 = calculate_determinant_2x2([[matrix[r][0], constants[r]]
for r in range(n)])
    x1 = det1 / det
    x2 = det2 / det
    return x1, x2

# Функция для вычисления определителя 3x3 матрицы
def calculate_determinant_3x3(matrix):
    positive_sum = matrix[0][0] * matrix[1][1] * matrix[2][2] + \
                   matrix[0][1] * matrix[1][2] * matrix[2][0] + \
                   matrix[0][2] * matrix[1][0] * matrix[2][1]
    negative_sum = matrix[0][2] * matrix[1][1] * matrix[2][0] + \
                   matrix[0][1] * matrix[1][0] * matrix[2][2] + \
                   matrix[0][0] * matrix[1][2] * matrix[2][1]
    return positive_sum - negative_sum

# Функция для решения системы линейных уравнений 3x3
def solve_linear_system_3x3(matrix, constants):
    n = 3
    det = calculate_determinant_3x3(matrix)
    det1 = calculate_determinant_3x3([[constants[r], matrix[r][1],
matrix[r][2]] for r in range(n)])
    det2 = calculate_determinant_3x3([[matrix[r][0], constants[r],
matrix[r][2]] for r in range(n)])
    det3 = calculate_determinant_3x3([[matrix[r][0], matrix[r][1],
constants[r]] for r in range(n)])
    x1 = det1 / det
    x2 = det2 / det
    x3 = det3 / det
    return x1, x2, x3

# Функция для вычисления определителя 4x4 матрицы
def calculate_determinant_4x4(matrix):
    n = 4
    sign = 1
    row = 0
    result = 0
    for column in range(n):
        submatrix = [[matrix[r][c] for c in range(n) if c !=
column] for r in range(n) if r != row]
        result += sign * matrix[row][column] *
calculate_determinant_3x3(submatrix)
        sign *= -1
```

```
    return result

# Функция для решения системы линейных уравнений 4x4
def solve_linear_system_4x4(matrix, constants):
    n = 4
    det = calculate_determinant_4x4(matrix)
    det1 = calculate_determinant_4x4([[constants[r], matrix[r][1],
matrix[r][2], matrix[r][3]] for r in range(n)])
    det2 = calculate_determinant_4x4([[matrix[r][0], constants[r],
matrix[r][2], matrix[r][3]] for r in range(n)])
    det3 = calculate_determinant_4x4([[matrix[r][0], matrix[r][1],
constants[r], matrix[r][3]] for r in range(n)])
    det4 = calculate_determinant_4x4([[matrix[r][0], matrix[r][1],
matrix[r][2], constants[r]] for r in range(n)])
    x1 = det1 / det
    x2 = det2 / det
    x3 = det3 / det
    x4 = det4 / det
    return x1, x2, x3, x4

# Линейная аппроксимация данных
def linear_approximation(xs, ys, n):
    sum_x = sum(xs)
    sum_x_squared = sum(x ** 2 for x in xs)
    sum_y = sum(ys)
    sum_xy = sum(x * y for x, y in zip(xs, ys))
    a, b = solve_linear_system_2x2(
        [
            [n, sum_x],
            [sum_x, sum_x_squared]
        ],
        [sum_y, sum_xy])
    return lambda x: a + b * x, a, b

# Квадратичная аппроксимация данных
def quadratic_approximation(xs, ys, n):
    sum_x = sum(xs)
    sum_x_squared = sum(x ** 2 for x in xs)
    sum_x_cubed = sum(x ** 3 for x in xs)
    sum_x_quad = sum(x ** 4 for x in xs)
    sum_y = sum(ys)
    sum_xy = sum(x * y for x, y in zip(xs, ys))
    sum_xx_y = sum(x * x * y for x, y in zip(xs, ys))
    a, b, c = solve_linear_system_3x3(
        [
            [n, sum_x, sum_x_squared],
            [sum_x, sum_x_squared, sum_x_cubed],
            [sum_x_squared, sum_x_cubed, sum_x_quad]
```

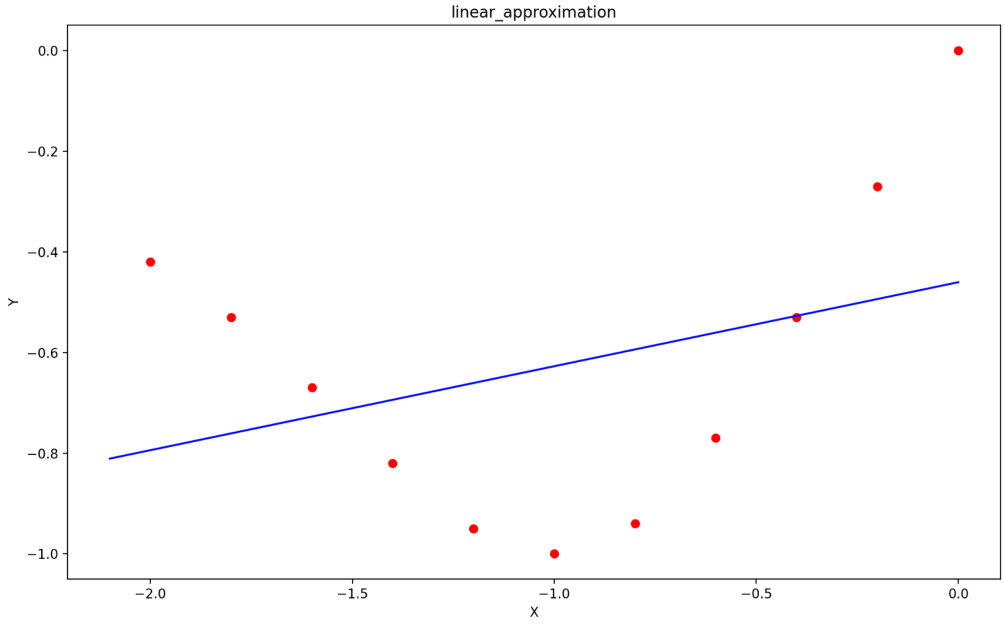
```
        ],
        [sum_y, sum_xy, sum_xx_y]
    )
    return lambda x: a + b * x + c * x ** 2, a, b, c

# Кубическая аппроксимация данных
def cubic_approximation(xs, ys, n):
    sum_x = sum(xs)
    sum_x_squared = sum(x ** 2 for x in xs)
    sum_x_cubed = sum(x ** 3 for x in xs)
    sum_x_quad = sum(x ** 4 for x in xs)
    sum_x_quint = sum(x ** 5 for x in xs)
    sum_x_sext = sum(x ** 6 for x in xs)
    sum_y = sum(ys)
    sum_xy = sum(x * y for x, y in zip(xs, ys))
    sum_xx_y = sum(x * x * y for x, y in zip(xs, ys))
    sum_xxx_y = sum(x * x * x * y for x, y in zip(xs, ys))
    a, b, c, d = solve_linear_system_4x4(
        [
            [n, sum_x, sum_x_squared, sum_x_cubed],
            [sum_x, sum_x_squared, sum_x_cubed, sum_x_quad],
            [sum_x_squared, sum_x_cubed, sum_x_quad, sum_x_quint],
            [sum_x_cubed, sum_x_quad, sum_x_quint, sum_x_sext]
        ],
        [sum_y, sum_xy, sum_xx_y, sum_xxx_y]
    )
    return lambda x: a + b * x + c * x ** 2 + d * x ** 3, a, b, c,
d

# Экспоненциальная аппроксимация данных
def exponential_approximation(xs, ys, n):
    log_ys = list(map(log, ys))
    _, a, b = linear_approximation(xs, log_ys, n)
    a = exp(a)
    return lambda x: a * exp(b * x), a, b

# Логарифмическая аппроксимация данных
def logarithmic_approximation(xs, ys, n):
    log_xs = list(map(log, xs))
    _, a, b = linear_approximation(log_xs, ys, n)
    return lambda x: a + b * log(x), a, b

# Степенная аппроксимация данных
def power_approximation(xs, ys, n):
    log_xs = list(map(log, xs))
    log_ys = list(map(log, ys))
    _, a, b = linear_approximation(log_xs, log_ys, n)
    a = exp(a)
```

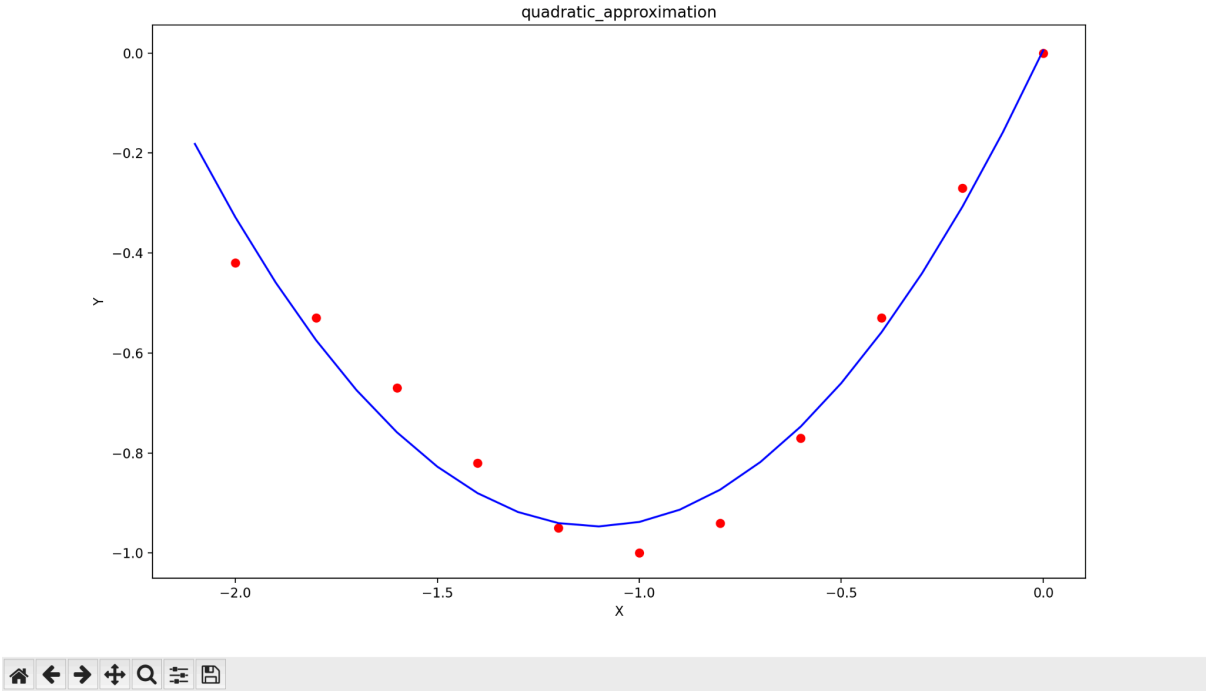
```
    return lambda x: a * x ** b, a, b

# Вычисление меры отклонения
def calculate_deviation(xs, ys, approximation_func, n):
    errors = [approximation_func(x) - y for x, y in zip(xs, ys)]
    return sum((error ** 2 for error in errors))

# Вычисление стандартного отклонения
def calculate_standard_deviation(xs, ys, approximation_func, n):
    return sqrt(sum(((approximation_func(x) - y) ** 2 for x, y in
zip(xs, ys))) / n)

# Вычисление коэффициента корреляции Пирсона
def calculate_pearson_correlation(xs, ys, n):
    avg_x = sum(xs) / n
    avg_y = sum(ys) / n
    return sum((x - avg_x) * (y - avg_y) for x, y in zip(xs, ys)) /
\
           sqrt(sum((x - avg_x) ** 2 for x in xs) * sum((y - avg_y)
** 2 for y in ys))

# Вычисление коэффициента детерминации
def calculate_coefficient_of_determination(xs, ys,
approximation_func, n):
    avg_fi = sum(approximation_func(x) for x in xs) / n
    return 1 - sum((y - approximation_func(x)) ** 2 for x, y in
zip(xs, ys)) / sum((y - avg_fi) ** 2 for y in ys)

# Получение строкового представления функции
def get_function_string(func):
    func_str = inspect.getsourcelines(func)[0][0]
    return func_str.split('lambda x: ')[-1].split(',')[0].strip()

# Рисование графика
def plot_graph(start, end, func, step=0.1):
    xs, ys = [], []
    start -= step
    end += step
    x = start
    while x <= end:
        xs.append(x)
        ys.append(func(x))
        x += step
    plt.plot(xs, ys, 'b')

# Основная функция для выбора наилучшей аппроксимации
def main(xs, ys, n):
    if all(x > 0 for x in xs) and all(y > 0 for y in ys):
```

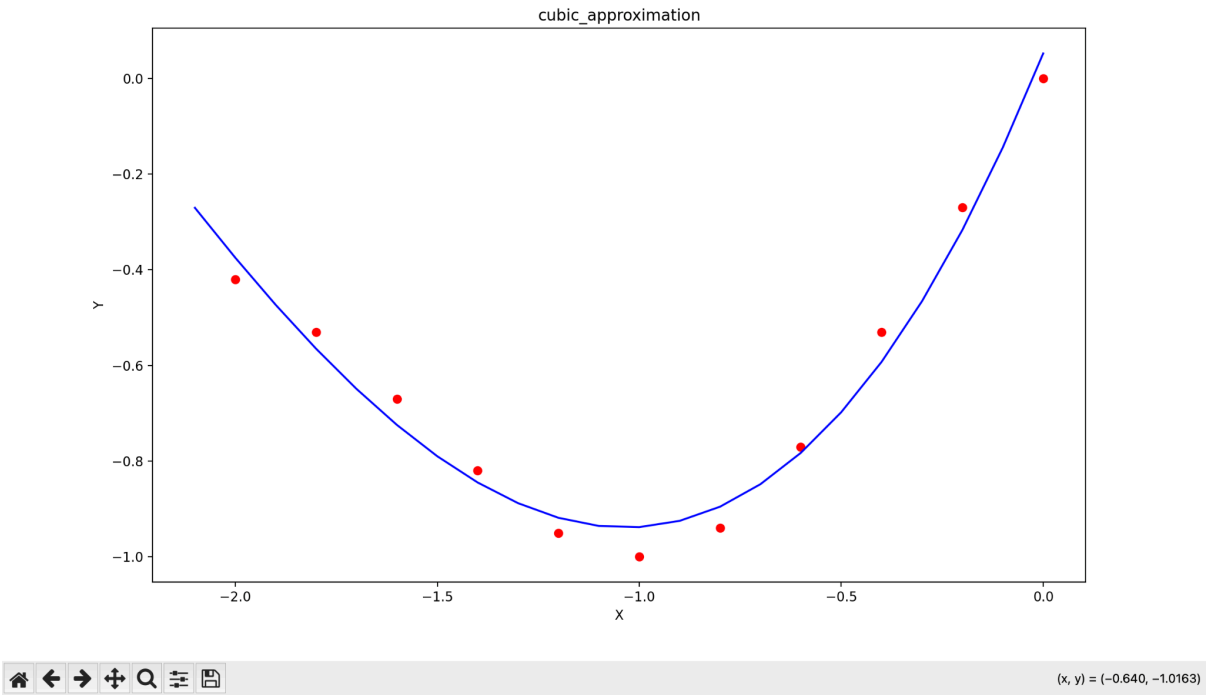
```
        approximation_funcs = [
            linear_approximation,
            power_approximation,
            exponential_approximation,
            logarithmic_approximation,
            quadratic_approximation,
            cubic_approximation
        ]
    else:
        approximation_funcs = [
            linear_approximation,
            quadratic_approximation,
            cubic_approximation
        ]

    best_sigma = float('inf')
    best_approximation_func = None

    for approximation_func in approximation_funcs:
        print(approximation_func.__name__, ": ")
        fi, *coeffs = approximation_func(xs, ys, n)
        deviation = calculate_deviation(xs, ys, fi, n)
        sigma = calculate_standard_deviation(xs, ys, fi, n)
        if sigma < best_sigma:
            best_sigma = sigma
            best_approximation_func = approximation_func
        r2 = calculate_coefficient_of_determination(xs, ys, fi, n)
        print('fi(x) =', get_function_string(fi))
        coeff_format = '(a, b, c)' if len(coeffs) == 3 else '(a,
b)'
        print(f'coeffs {coeff_format}:', list(map(lambda cf:
round(cf, 4), coeffs)))
        print(f'S = {deviation:.5f}, σ = {sigma:.5f}, R2 =
{r2:.5f}')
        if approximation_func is linear_approximation:
            print('r =', calculate_pearson_correlation(xs, ys, n))
        plt.title(approximation_func.__name__)

        plot_graph(xs[0], xs[-1], fi)
        for i in range(n):
            plt.scatter(xs[i], ys[i], c='r')
        plt.xlabel("X")
        plt.ylabel("Y")
        plt.show()
        print('-' * 50)
    print(f'best_func: {best_approximation_func.__name__}')

# Точка входа в программу
```

```
if __name__ == '__main__':
    case = 4
    if case == 0:
        n = int(input('Введите n: '))
        xs = list(map(float, input('Введите xs: ').split()))
        ys = list(map(float, input('Введите ys: ').split()))
    elif case == 1:
        xs = [1.5, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
        ys = [8.0, 10.0, 11.5, 13.0, 15.0, 17.0, 18.0, 20.0]
    elif case == 2:
        xs = [2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0]
        ys = [7.0, 9.0, 11.0, 13.0, 15.0, 17.0, 19.0, 21.0]
    elif case == 3:
        xs = [0.6, 1.4, 1.8, 2.6, 2.8, 3.5, 3.9, 4.5]
        ys = [5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0]
    else:
        h = 0.2
        x0 = -2
        n = 11
        xs = [round(x0 + i * h, 2) for i in range(n)]
        f = lambda x: 4 * x / (x ** 4 + 3)
        ys = [round(f(x), 2) for x in xs]
    n = len(xs)
    main(xs, ys, n)
```

5. Результаты выполнения программы







6. Выводы

В ходе лабораторной работы мы изучили аппроксимацию функции методом наименьших квадратов.