МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

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

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №7

Выполнила:
Конаныхина Антонина
P3215
Преподаватель:
Малышева Татьяна
Алексеевна

Санкт-Петербург, 2022

Цель работы

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

Задание:

1. Методика проведения исследования:

a. Вычислить меру отклонения: $S = \sum_{i=1}^{n}[\varphi(x_i) - y_i]^2$ для всех исследуемых функций.

b. Уточнить значения коэффициентов эмпирических функций, минимизируя функцию S.

c. Сформировать массивы предполагаемых эмпирических зависимостей $(\varphi(x_i), \varepsilon_i)$.

d. Определить среднеквадратичное отклонение для каждой аппроксимирующей функции. Выбрать наименьшее значение и, следовательно, наилучшее приближение.

e. Построить графики полученных эмпирических функций

<u>Программная реализация задачи:</u>

a. Предусмотреть ввод исходных данных из файла/консоли (таблица *y=f(x)* должна содержать 10 - 12 точек).

b. Реализовать метод наименьших квадратов, исследуя все функции п.1.

c. Предусмотреть вывод результатов в файл/консоль.

d. Для линейной зависимости вычислить коэффициент корреляции Пирсона.

e. Программа должна отображать наилучшую аппроксимирующую функцию.

f. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).

<u>Вычислительная реализация задачи:</u>

a) Для заданной функции (см. таблицу 1) построить наилучшие линейное и квадратичное приближения по 11 точкам указанного интервала.

b) Найти среднеквадратические отклонения. Ответы дать с тремя знаками после запятой.

c) Построить графики линейного и квадратичного приближений и заданной функции.

d) ***Привести в отчете подробные вычисления***.

Рабочие формулы используемых методов

Параметры $a_0, a_1, a_2, \dots, a_m$ эмпирической формулы находятся из условия минимума функции $S = S(a_0, a_1, a_2, \dots, a_m)$. Так как здесь параметры выступают в роли независимых переменных функции S, то ее минимум найдем, приравнивая к нулю частные производные по этим переменным.

$$\frac{\partial S}{\partial a_0} = 2\sum_{i=1}^{n} a_0 + a_1 x_i + \dots + a_{m-1}x_i^{m-1} + a_m x_i^m - y_i = 0$$

$$\frac{\partial S}{\partial a_1} = 2\sum_{i=1}^{n}(a_0 + a_1 x_i + \dots + a_{m-1}x_i^{m-1} + a_m x_i^m - y_i)x_i = 0$$

$$\dots\dots\dots$$

$$\frac{\partial S}{\partial a_m} = 2\sum_{i=1}^{n}(a_0 + a_1 x_i + \dots + a_{m-1}x_i^{m-1} + a_m x_i^m - y_i)x_i^m = 0$$

Преобразуем полученную линейную систему уравнений: раскроем скобки и перенесем свободные слагаемые в правую часть выражения:

$$\begin{cases} a_0 n + a_1 \sum_{i=1}^{n} x_i + \dots + a_{m-1} \sum_{i=1}^{n} x_i^{m-1} + a_m \sum_{i=1}^{n} x_i^m = \sum_{i=1}^{n} y_i \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + \dots + a_{m-1} \sum_{i=1}^{n} x_i^m + a_m \sum_{i=1}^{n} x_i^{m+1} = \sum_{i=1}^{n} x_i y_i \\ \dots \dots \dots \\ a_0 \sum_{i=1}^{n} x_i^m + a_1 \sum_{i=1}^{n} x_i^{m+1} + \dots + a_{m-1} \sum_{i=1}^{n} x_i^{2m-1} + a_m \sum_{i=1}^{n} x_i^{2m} = \sum_{i=1}^{n} x_i^m y_i \end{cases}$$

в матричном виде:

$$\begin{vmatrix} n & \sum_{i=1}^{n} x_i & \dots & \sum_{i=1}^{n} x_i^m \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \dots & \sum_{i=1}^{n} x_i^{m+1} \\ & \dots \dots \dots & & \\ \sum_{i=1}^{n} x_i^m & \sum_{i=1}^{n} x_i^{m+1} & \dots & \sum_{i=1}^{n} x_i^{2m} \end{vmatrix} \cdot \begin{vmatrix} a_0 \\ a_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ a_m \end{vmatrix} = \begin{vmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \dots \\ \sum_{i=1}^{n} x_i^m y_i \end{vmatrix}$$

Вычислительная реализация задачи:

Функция:

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

Составим таблицу с точками и значениями функции в этих точках на промежутке $x \in [-2, 0]$ с шагом 0.2:

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
$f(x_i)$	-0.316	-0.4	-0.502	-0.614	-0.71	-0.75	-0.704	-0.575	-0.397	-0.2	0

$SX = -2 - 1.8 - 1.6 - 1.4 - 1.2 - 1 - 0.8 - 0.6 - 0.4 - 0.2 = -11$

$SXX = 4 + 3.24 + 2.56 + 1.96 + 1.44 + 1 + 0.64 + 0.36 + 0.16 + 0.04 = 15.4$

$SXXX = -8 - 5.832 - 4.096 - 2.744 - 1.728 - 1 - 0.512 - 0.216 - 0.064 - 0.008 = -24.2$

$SXXXX = 16 + 10.498 + 6.554 + 3.842 + 2.074 + 1 + 0.41 + 0.13 + 0.026 + 0.0016 = 40.533$

$SY = -0.316 - 0.4 - 0.502 - 0.614 - 0.71 - 0.75 - 0.704 - 0.575 - 0.397 - 0.2 = 5.167$

$SXY = 0.632 + 0.72 + 0.804 + 0.859 + 0.852 + 0.75 + 0.563 + 0.345 + 0.159 + 0.04 = 5.723$

$SXXY = -1.263 - 0.296 - 1.286 - 1.203 - 1.022 - 0.75 - 0.45 - 0.207 - 0.063 - 0.008 = -7.55$

Линейная аппроксимация:

$$\begin{cases} 15.4a - 11b = 5.723 \\ -11a + 11b = -5.197 \end{cases}$$

$$\Delta = 15.4 * 11 - 11 * 11 = 48.4$$

$$\Delta_1 = 5.723 * 11 - 11 * 5.197 = 6.117$$

$$\Delta_2 = -15.4 * 5.197 + 11 * 5.723 = -16.62$$

$$a = \frac{\Delta_1}{\Delta} = \frac{6.117}{48.4} = 0.126$$

$$b = \frac{\Delta_2}{\Delta} = -\frac{16.62}{48.4} = -0.343$$

$$\varphi(x) = 0.126x - 0.343$$

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
$f(x_i)$	-0.316	-0.4	-0.502	-0.614	-0.71	-0.75	-0.704	-0.575	-0.397	-0.2	0
$\varphi(x)$	-0.596	-0.57	-0.546	-0.52	-0.495	-0.47	-0.444	-0.419	-0.39	-0.37	-0.343
ε_i	0.28	0.17	0.043	-0.036	-0.21	-0.28	-0.259	-0.156	-0.003	0.17	0.343

$$S = 0.28^2 + 0.17^2 + 0.043^2 + 0.036^2 + 0.21^2 + 0.28^2 + 0.259^2 + 0.156^2 + 0.003^2 + 0.17^2 + 0.343^2 = 0.481$$

$$\delta = \sqrt{\frac{S}{n}} = 0.209$$

Квадратичная аппроксимация:

$$\begin{cases} 11c - 11b + 15.4a = -5.167 \\ -11c + 15.4b - 24.2a = 5.723 \\ 15.4c - 24.2b + 40.533a = -7.55 \end{cases}$$

$$\Delta = \begin{vmatrix} 11 & -11 & 15.4 \\ -11 & 15.4 & -24.2 \\ 15.4 & -24.2 & 40.533 \end{vmatrix} = 66.44$$

$$\Delta_1 = \begin{vmatrix} -5.167 & -11 & 15.4 \\ 5.723 & 15.4 & -24.2 \\ -7.55 & -24.2 & 40.533 \end{vmatrix} = 0.318$$

$$\Delta_2 = \begin{vmatrix} 11 & -5.167 & 15.4 \\ -11 & 5.723 & -24.2 \\ 15.4 & -7.55 & 40.533 \end{vmatrix} = 85.5$$

$$\Delta_3 = \begin{vmatrix} 11 & -11 & -5.167 \\ -11 & 15.4 & 5.723 \\ 15.4 & -24.2 & -7.55 \end{vmatrix} = 38.556$$

$$c = \frac{\Delta_1}{\Delta} = \frac{0.318}{66.44} = 0.005$$

$$b = \frac{\Delta_2}{\Delta} = \frac{85.5}{66.44} = 1.287$$

$$a = \frac{\Delta_3}{\Delta} = \frac{38.556}{66.44} = 0.58$$

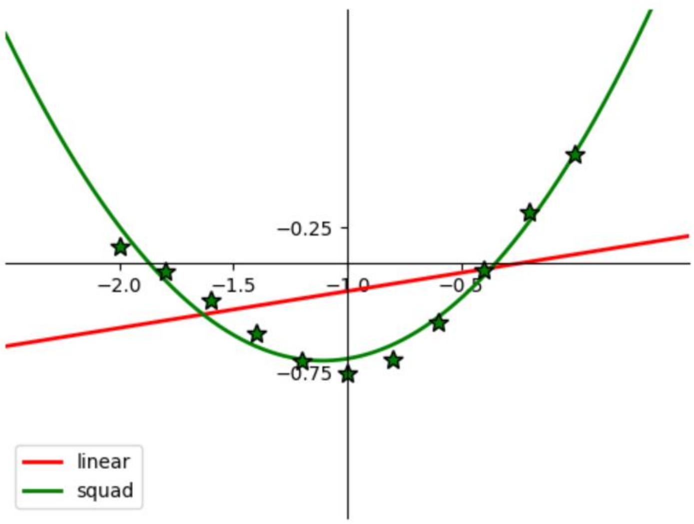
$$\varphi(x) = 0.58x^2 + 1.287x + 0.005$$

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
$f(x_i)$	-0.316	-0.4	-0.502	-0.614	-0.71	-0.75	-0.704	-0.575	-0.397	-0.2	0
$\varphi(x)$	-0.25	-0.43	-0.569	-0.66	-0.704	-0.702	-0.653	-0.558	-0.4	-0.2	0.005
ε_i	-0.068	0.032	0.066	0.046	-0.006	-0.048	-0.05	-0.017	0.02	0.03	-0.005

$$S = 0.068^2 + 0.032^2 + 0.066^2 + 0.046^2 + 0.006^2 + 0.048^2 + 0.05^2 + 0.017^2 + 0.02^2 + 0.03^2 + 0.005^2 = 0.019$$

$$\delta = \sqrt{\frac{S}{n}} = 0.042$$

Графики полученных функций:



Листинг программы

```
#Линейная аппроксимация
def linear_approximate(points):

    n = len(points)

    summ_x = 0
    for i in range(n):
        summ_x += points[i][0]

    summ_x_sqd = 0
    for i in range(n):
        summ_x_sqd += points[i][0]**2

    summ_y = 0
    for i in range(n):
        summ_y += points[i][1]

    summ_x_y = 0
    for i in range(n):
        summ_x_y += points[i][0] * points[i][1]

    #коэффициент корреляции Пирсона
    mid_x = summ_x / n
    mid_y = summ_y / n

    #числитель
    summ_1 = 0
    for i in range(n):
        summ_1 += (points[i][0] - mid_x) * (points[i][1] - mid_y)

    #знаменатель (суммы 2 и 3)
    summ_2 = 0
    for i in range(n):
        summ_2 += (points[i][0] - mid_x) ** 2
    summ_3 = 0
    for i in range(n):
        summ_3 += (points[i][1] - mid_y) ** 2

    try:
        r = (summ_1)/(math.sqrt(summ_2*summ_3))
        print(f"Коэффициент корреляции Пирсона равен: {round(r, 3)}")
    except Exception:
        print("Не получилось посчитать коэффициент корреляции Пирсона")
    ans = calc_system([[summ_x_sqd, summ_x, summ_x_y],[summ_x, n, summ_y]],
2)

    result_func = lambda x: ans[0]*x + ans[1]

    str_result_func = f"{round(ans[0], 3)}x + {round(ans[1], 3)}"

    #среднеквадратичное отклонение
    print([result_func(points[i][0]) for i in range(n)])
    print([(points[i][1] - result_func(points[i][0])) for i in range(n)])
    errors = [(points[i][1] - result_func(points[i][0]))**2 for i in
range(n)]
    mid_sqd_err = math.sqrt(sum(errors)/n)

    return result_func, str_result_func, errors, mid_sqd_err

#Квадратичная аппроксимация
```

```
def squad_approximate(points):
    n = len(points)
    summ_x = 0
    for i in range(n):
        summ_x += points[i][0]

    summ_x_sqd = 0
    for i in range(n):
        summ_x_sqd += points[i][0]**2

    summ_x_qub = 0
    for i in range(n):
        summ_x_qub += points[i][0]**3

    summ_x_forth = 0
    for i in range(n):
        summ_x_forth += points[i][0]**4

    summ_y = 0
    for i in range(n):
        summ_y += points[i][1]

    summ_x_y = 0
    for i in range(n):
        summ_x_y += points[i][0] * points[i][1]

    summ_x_sqd_y = 0
    for i in range(n):
        summ_x_sqd_y += (points[i][0]**2) * points[i][1]

    system = [
        [n, summ_x, summ_x_sqd, summ_y],
        [summ_x, summ_x_sqd, summ_x_qub, summ_x_y],
        [summ_x_sqd, summ_x_qub, summ_x_forth, summ_x_sqd_y]
    ]

    ans = calc_system(system, 3)

    result_func = lambda x: ans[2]*(x**2) + ans[1]*x + ans[0]

    str_result_func = f"{round(ans[2], 3)}x^2 + {round(ans[1], 3)}x +
{round(ans[0], 3)}"

    #СКО
    print([result_func(points[i][0]) for i in range(n)])
    print([(points[i][1] - result_func(points[i][0])) for i in range(n)])
    errors = [(points[i][1] - result_func(points[i][0]))**2 for i in
range(n)]
    mid_sqd_err = math.sqrt(sum(errors)/n)

    return result_func, str_result_func, errors, mid_sqd_err

#Кубическая аппроксимация
def qub_approximate(points):
    n = len(points)

    summ_x = 0
    for i in range(n):
        summ_x += points[i][0]

    summ_x_sqd = 0
    for i in range(n):
        summ_x_sqd += points[i][0]**2
```

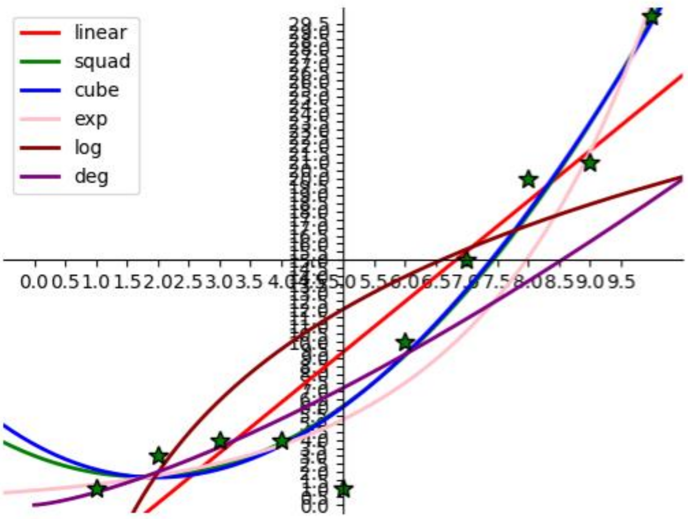
```
    summ_x_qub = 0
    for i in range(n):
        summ_x_qub += points[i][0]**3

    summ_x_forth = 0
    for i in range(n):
        summ_x_forth += points[i][0]**4

    summ_x_fifth = 0
    for i in range(n):
        summ_x_fifth += points[i][0]**5

    summ_x_six = 0
    for i in range(n):
        summ_x_six += points[i][0] ** 6

    summ_y = 0
    for i in range(n):
        summ_y += points[i][1]

    summ_x_y = 0
    for i in range(n):
        summ_x_y += points[i][0] * points[i][1]

    summ_x_sqd_y = 0
    for i in range(n):
        summ_x_sqd_y += (points[i][0]**2) * points[i][1]

    summ_x_cub_y = 0
    for i in range(n):
        summ_x_cub_y += (points[i][0] ** 3) * points[i][1]

    system = [
        [n, summ_x, summ_x_sqd, summ_x_qub, summ_y],
        [summ_x, summ_x_sqd, summ_x_qub, summ_x_forth, summ_x_y],
        [summ_x_sqd, summ_x_qub, summ_x_forth, summ_x_fifth, summ_x_sqd_y],
        [summ_x_qub, summ_x_forth, summ_x_fifth, summ_x_six,summ_x_cub_y]
    ]

    ans = calc_system(system, 4)

    result_func = lambda x: ans[3]*(x**3) + ans[2]*(x**2) + ans[1]*x + ans[0]

    str_result_func = f"{round(ans[3], 3)}x^3 + {round(ans[2], 3)}x^2 +
{round(ans[1], 3)}x + {round(ans[0], 3)}"

    #СКО
    errors = [(points[i][1] - result_func(points[i][0]))**2 for i in
range(n)]
    mid_sqd_err = math.sqrt(sum(errors)/n)

    return result_func, str_result_func, errors, mid_sqd_err

#Степенная аппроксимация
def degree_approximate(input_points):
    points = []

    #добавляем в массив только те точки, которые подходят по ОДЗ логарифма
    for i in input_points:
        if i[1] > 0 and i[0] > 0:
            points.append(i)

    #if len(points) < 2:, но это будет неидеальная аппроксимация
    if len(points) != len(input_points):
```

```
        return None, None, None, None

    n = len(points)
    summ_x = 0
    for i in range(n):
        summ_x += math.log(points[i][0])

    summ_x_sqd = 0
    for i in range(n):
        summ_x_sqd += math.log(points[i][0]) ** 2

    summ_y = 0
    for i in range(n):
        summ_y += math.log(points[i][1])

    summ_x_y = 0
    for i in range(n):
        summ_x_y += math.log(points[i][0]) * math.log(points[i][1])

    try:
        ans = calc_system([[summ_x_sqd, summ_x, summ_x_y], [summ_x, n,
summ_y]], 2)
    except Exception:
        return None, None, None, None
    result_func = lambda x: np.exp(ans[1])*(x ** ans[0])

    str_result_func = f"{round(math.exp(ans[1]), 3)}x^{round(ans[0], 3)}"

    #СКО
    errors = [(points[i][1] - result_func(points[i][0])) ** 2 for i in
range(n)]
    mid_sqd_err = math.sqrt(sum(errors) / n)

    return result_func, str_result_func, errors, mid_sqd_err

 #экспоненциальная аппроксимация
def exp_approximate(input_points):
    points = []

    for i in input_points:
        if i[1] > 0:
            points.append(i)

    #if len(points) < 2:, но это будет неидеальная аппроксимация
    if len(points) != len(input_points):
        return None, None, None, None

    n = len(points)
    summ_x = 0
    for i in range(n):
        summ_x += points[i][0]

    summ_x_sqd = 0
    for i in range(n):
        summ_x_sqd += points[i][0] ** 2

    summ_y = 0
    for i in range(n):
        summ_y += math.log(points[i][1])

    summ_x_y = 0
    for i in range(n):
        summ_x_y += points[i][0] * math.log(points[i][1])
    try:
```

```
        ans = calc_system([[summ_x_sqd, summ_x, summ_x_y], [summ_x, n,
summ_y]], 2)
    except Exception:
        return None, None, None, None
    result_func = lambda x: np.exp(ans[1])* np.exp(ans[0]*x)

    str_result_func = f"{round(math.exp(ans[1]), 3)}e^{round(ans[0], 3)}*x"
    #СКО
    errors = [(points[i][1] - result_func(points[i][0])) ** 2 for i in
range(n)]
    mid_sqd_err = math.sqrt(sum(errors) / n)

    return result_func, str_result_func, errors, mid_sqd_err

#Логарифмическая аппроксимация
def ln_approximate(input_points):
    points = []

    for i in input_points:
        if i[0] > 0:
            points.append(i)

    # if len(points) < 2:, но это будет неидеальная аппроксимация
    if len(points) != len(input_points):
        return None, None, None, None

    n = len(points)

    summ_x = 0
    for i in range(n):
        summ_x += math.log(points[i][0])

    summ_x_sqd = 0
    for i in range(n):
        summ_x_sqd += math.log(points[i][0]) ** 2

    summ_y = 0
    for i in range(n):
        summ_y += points[i][1]

    summ_x_y = 0
    for i in range(n):
        summ_x_y += math.log(points[i][0]) * points[i][1]

    try:
        ans = calc_system([[summ_x_sqd, summ_x, summ_x_y], [summ_x, n,
summ_y]], 2)
    except Exception:
        return None, None, None, None
    result_func = lambda x: ans[0]* np.log(x) + ans[1]

    str_result_func = f"{round(ans[0], 3)} ln(x) + {round(ans[1], 3)}"

    #СКО
    errors = [(points[i][1] - result_func(points[i][0])) ** 2 for i in
range(n)]
    mid_sqd_err = math.sqrt(sum(errors) / n)

    return result_func, str_result_func, errors, mid_sqd_err
```

Результаты выполнения программы:

```
Ведите источник точек. Для файла: 1, для консоли: 2, готовая функция: 3: 1
Полученные точки: [[1.0, 1.0], [2.0, 3.0], [4.0, 4.0], [5.0, 1.0], [6.0, 10.0], [7.0, 15.0], [8.0, 20.0], [9.0, 21.0], [10.0, 30

Коэффициент корреляции Пирсона равен: 0.923
Линейной аппроксимацией получена функция: 3.085x + -6.067, S = 135.806, sigma = 3.685

Квадратичной аппроксимацией получена функция: 0.413x^2 + -1.456x + 3.016, S = 45.798, sigma = 2.14

Кубической аппроксимацией получена функция: -0.008x^3 + 0.548x^2 + -2.077x + 3.715, S = 45.583, sigma = 2.135

Экспоненциальной аппроксимацией получена функция: 0.888e^0.355*x, S = 71.454, sigma = 2.673

Логарифмической аппроксимацией получена функция: 10.94 ln(x) + -5.624, S = 342.124, sigma = 5.849

Степенной апроксимацией получена функция: 0.777x^1.38, S = 246.602, sigma = 4.966

Минимальное среднеквадратичное отклонение: 2.135
Лучшая аппроксимация: кубическая

Process finished with exit code 0
```



Вывод:

В результате выполнения данной лабораторной работы я познакомилась с аппроксимациями функции методом наименьших квадратов и реализовала их на языке программирования Python. Достоинства метода: расчеты довольны просты – необходимо лишь найти коэффициенты; полученная функция также проста; разнообразие возможных аппроксимирующих функций. Основным недостатком МНК является чувствительность оценок к резким выбросам, которые встречаются в исходных данных.