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 №18

Выполнил:
Студент группы P3213
Хафизов Булат Ленарович
Преподаватель:
Малышева Татьяна
Алексеевна

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург, 2022

Цель работы

Изучить аппроксимации функции методом наименьших квадратов и реализовать их средствами программирования.

Вычислительная часть

$y = \frac{3x}{x^4+4}$; $x \in [0, 2]$ $h = 0,2$

Возьмем 11 точек:
$(0;\ 0), \left(0,2;\ \frac{375}{2501}\right), \left(0,4;\ \frac{375}{1258}\right), \left(0,6; \frac{1125}{2581}\right), \left(0,8; \frac{375}{689}\right), \left(1; \frac{3}{5}\right), \left(1,2; \frac{1125}{1898}\right), \left(1,4; \frac{2625}{4901}\right), \left(1,6; \frac{750}{1649}\right),$

$\left(1,8; \frac{3375}{9061}\right), (2; \frac{3}{10})$

Произведем линейную аппроксимацию:

$$SX = \sum_{i=1}^{n} x_i = 0 + 0.2 + 0.4 + 0.6 + 0.8 + 1 + 1.2 + 1.4 + 1.6 + 1.8 + 2 = 11$$

$$SXX = \sum_{i=1}^{n} x_i^2 = 0 + 0.2^2 + 0.4^2 + 0.6^2 + 0.8^2 + 1^2 + 1.2^2 + 1.4^2 + 1.6^2 + 1.8^2 + 2^2 = 15.4$$

$$SY = \sum_{i=1}^{n} y_i = 0 + \frac{375}{2501} + \frac{375}{1258} + \frac{1125}{2581} + \frac{375}{689} + \frac{3}{5} + \frac{1125}{1898} + \frac{2625}{4901} + \frac{750}{1649} + \frac{3375}{9061} + \frac{3}{10} = 4.284$$

$$SXY = \sum_{i=1}^{n} x_i y_i = 0.2 * \frac{375}{2501} + 0.4 * \frac{375}{1258} + 0.6 * \frac{1125}{2581} + 0.8 * \frac{375}{689} + \frac{3}{5} + 1.2 * \frac{1125}{1898} + 1.4 * \frac{2625}{4901} + 1.6 * \frac{750}{1649} + 1.8 * \frac{3375}{9061} + 2 * \frac{3}{10} = 4.905$$

$$\Delta = SXX * n - SX * SX = 15.4 * 10 - 11 * 11 = 154 - 121 = 33$$

$$\Delta_1 = SXY * n - SX * SY = 4.905 * 10 - 11 * 4.284 = 1.926$$

$$\Delta_2 = SXX * SY - SX * SXY = 15.4 * 4.284 - 11 * 4.905 = 12.019$$

$$a = \frac{\Delta_1}{\Delta} = \frac{1.926}{33} = 0.058$$

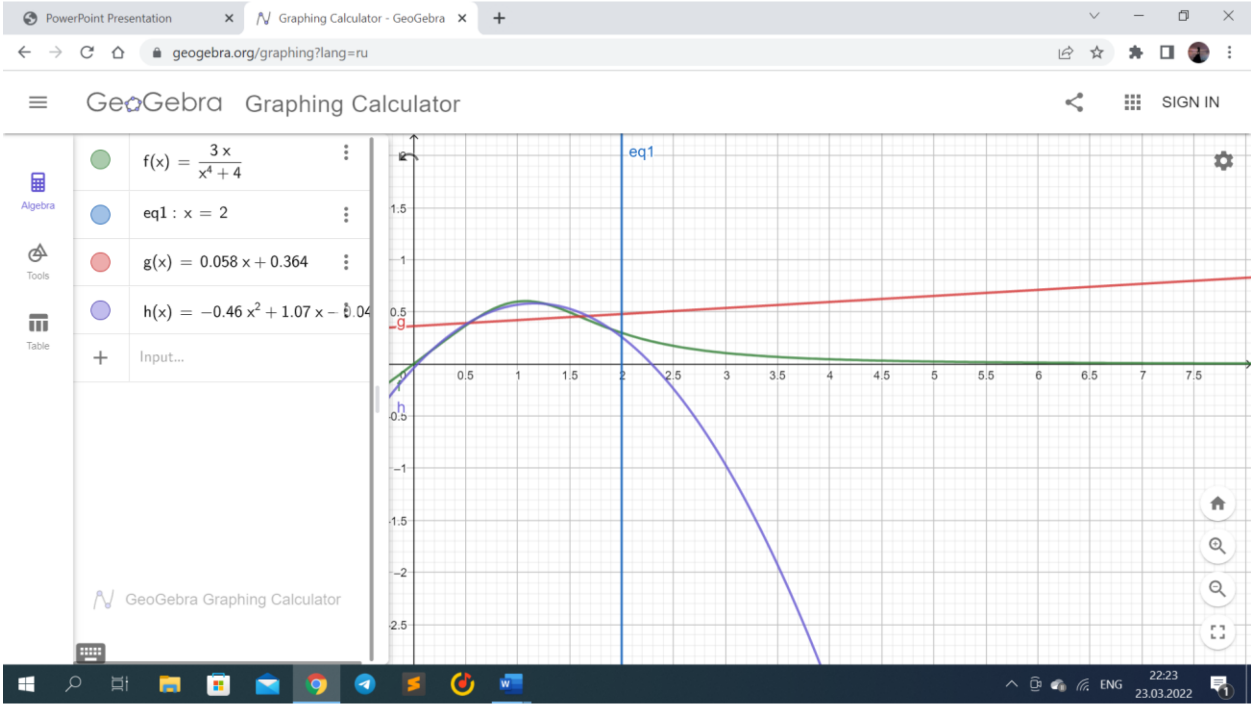
$$b = \frac{\Delta_2}{\Delta} = \frac{12.019}{33} = 0.364$$

Произведем квадратичную аппроксимацию:

$$\sum_{i=1}^{n} x_i = 11, \sum_{i=1}^{n} x_i^2 = 15.4, \sum_{i=1}^{n} x_i^3 = 24.2, \sum_{i=1}^{n} x_i^4 = 40.533, \sum_{i=1}^{n} y_i = 4.284, \sum_{i=1}^{n} x_i y_i = 4.905,$$

$$\sum_{i=1}^{n} x_i^2 y_i = 6.633$$

$$\begin{cases} 10a_0 + 11a_1 + 15.4a_2 = 4.284 \\ 11a_0 + 15.4a_1 + 24.2a_2 = 4.905 \\ 15.4a_0 + 24.2a_1 + 40.533a_2 = 6.633 \end{cases} => a_0 = -0.04, a_1 = 1.07, a_2 = -0.46$$



Вычислим среднеквадратичное отклонение:

$$\delta = \sqrt{\frac{\sum_{i=1}^{n}(\varphi(x_i) - y_i)^2}{n}}$$

Для линейной аппроксимации →

$$\delta = \sqrt{\frac{(0.364-0)^2 + \left(0.366 - \frac{375}{2501}\right)^2 + \left(0.387 - \frac{375}{1258}\right)^2 + \left(0.399 - \frac{1125}{2581}\right)^2 + \left(0.41 - \frac{375}{689}\right)^2 + \left(0.422 - \frac{3}{5}\right)^2 + \left(0.434 - \frac{1125}{1898}\right)^2 + \left(0.445 - \frac{2625}{4901}\right)^2 + \left(0.457 - \frac{750}{1649}\right)^2 + \left(0.468 - \frac{3375}{9061}\right)^2 + \left(0.48 - \frac{3}{10}\right)^2}{10}}$$

$$= 0.185$$

Для квадратической аппроксимации →

$$\delta = \sqrt{\frac{(-0.04-0)^2 + \left(0.156 - \frac{375}{2501}\right)^2 + \left(0.314 - \frac{375}{1258}\right)^2 + \left(0.436 - \frac{1125}{2581}\right)^2 + \left(0.522 - \frac{375}{689}\right)^2 + \left(0.57 - \frac{3}{5}\right)^2 + \left(0.582 - \frac{1125}{1898}\right)^2 + \left(0.556 - \frac{2625}{4901}\right)^2 + \left(0.494 - \frac{750}{1649}\right)^2 + \left(0.396 - \frac{3375}{9061}\right)^2 + \left(0.26 - \frac{3}{10}\right)^2}{10}}$$

$$= 0.066$$

Листинг программы

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


def solve_minor(matrix, i, j):
    n = len(matrix)
```

```
    return [[matrix[row][col] for col in range(n) if col != j] for row in
range(n) if row != i]


def solve_det(matrix):
    n = len(matrix)
    if n == 1:
        return matrix[0][0]
    det = 0
    sgn = 1
    for j in range(n):
        det += sgn * matrix[0][j] * solve_det(solve_minor(matrix, 0, j))
        sgn *= -1
    return det


def calc_s(dots, f):
    n = len(dots)
    x = [dot[0] for dot in dots]
    y = [dot[1] for dot in dots]
    return sum([(f(x[i]) - y[i]) ** 2 for i in range(n)])


def calc_stdev(dots, f):
    n = len(dots)
    return sqrt(calc_s(dots, f) / n)


def cor_pirson(dots):
    n = len(dots)
    x = [dot[0] for dot in dots]
    y = [dot[1] for dot in dots]
    xsum = sum(x)
    ysum = sum(y)
    r = sum([(x[i] - xsum) * (y[i] - ysum) for i in range(n)]) \
        / sqrt(sum([(x[i] - xsum) * (x[i] - xsum) for i in range(n)])
               * sum([(y[i] - ysum) * (y[i] - ysum) for i in range(n)]))
    return r


def linear_approximation(dots):
    data = {}
    n = len(dots)
    x = [dot[0] for dot in dots]
    y = [dot[1] for dot in dots]
    sx = sum(x)
    sy = sum(y)
    sxx = sum([xi ** 2 for xi in x])
    sxy = sum([x[i] * y[i] for i in range(n)])
    d = solve_det([[sxx, sx], [sx, n]])
    d1 = solve_det([[sxy, sx], [sy, n]])
    d2 = solve_det([[sxx, sxy], [sx, sy]])
    if d == 0:
        return None
    a = d1 / d
    b = d2 / d
    data['a'] = a
    data['b'] = b
    f = lambda z: a * z + b
    data['f'] = f
    data['str(f)'] = 'f = a * x + b'
    data['s'] = calc_s(dots, f)
    data['stdev'] = calc_stdev(dots, f)
    return data
```

```
def sqrt_approximation(dots):
    data = {}
    n = len(dots)
    x = [dot[0] for dot in dots]
    y = [dot[1] for dot in dots]
    sx = sum(x)
    sxx = sum([xi ** 2 for xi in x])
    sxxx = sum([xi ** 3 for xi in x])
    sxxxx = sum([xi ** 4 for xi in x])
    sy = sum(y)
    sxy = sum([x[i] * y[i] for i in range(n)])
    sxxy = sum([(x[i] ** 2) * y[i] for i in range(n)])
    d = solve_det([[n, sx, sxx],
                   [sx, sxx, sxxx],
                   [sxx, sxxx, sxxxx]])
    d1 = solve_det([[sy, sx, sxx],
                    [sxy, sxx, sxxx],
                    [sxxy, sxxx, sxxxx]])
    d2 = solve_det([[n, sy, sxx],
                    [sx, sxy, sxxx],
                    [sxx, sxxy, sxxxx]])
    d3 = solve_det([[n, sx, sy],
                    [sx, sxx, sxy],
                    [sxx, sxxx, sxxy]])
    if d == 0:
        return None
    c = d1 / d
    b = d2 / d
    a = d3 / d
    data['c'] = c
    data['b'] = b
    data['a'] = a
    f = lambda z: a * (z ** 2) + b * z + c
    data['f'] = f
    data['str(f)'] = "fi = a*x^2 + b*x + c"
    data['s'] = calc_s(dots, f)
    data['stdev'] = calc_stdev(dots, f)
    return data


def cube_approximation(dots):
    data = {}
    n = len(dots)
    x = [dot[0] for dot in dots]
    y = [dot[1] for dot in dots]
    sx = sum(x)
    sxx = sum([xi ** 2 for xi in x])
    sxxx = sum([xi ** 3 for xi in x])
    sxxxx = sum([xi ** 4 for xi in x])
    sxxxxx = sum([xi ** 5 for xi in x])
    sxxxxxx = sum([xi ** 6 for xi in x])
    sy = sum(y)
    sxy = sum([x[i] * y[i] for i in range(n)])
    sxxy = sum([(x[i] ** 2) * y[i] for i in range(n)])
    sxxxy = sum([(x[i] ** 3) * y[i] for i in range(n)])
    d = solve_det([[n, sx, sxx, sxxx],
                   [sx, sxx, sxxx, sxxxx],
                   [sxx, sxxx, sxxxx, sxxxxx],
                   [sxxx, sxxxx, sxxxxx, sxxxxxx]])
    d1 = solve_det([[sy, sx, sxx, sxxx],
                    [sxy, sxx, sxxx, sxxxx],
                    [sxxy, sxxx, sxxxx, sxxxxx],
```

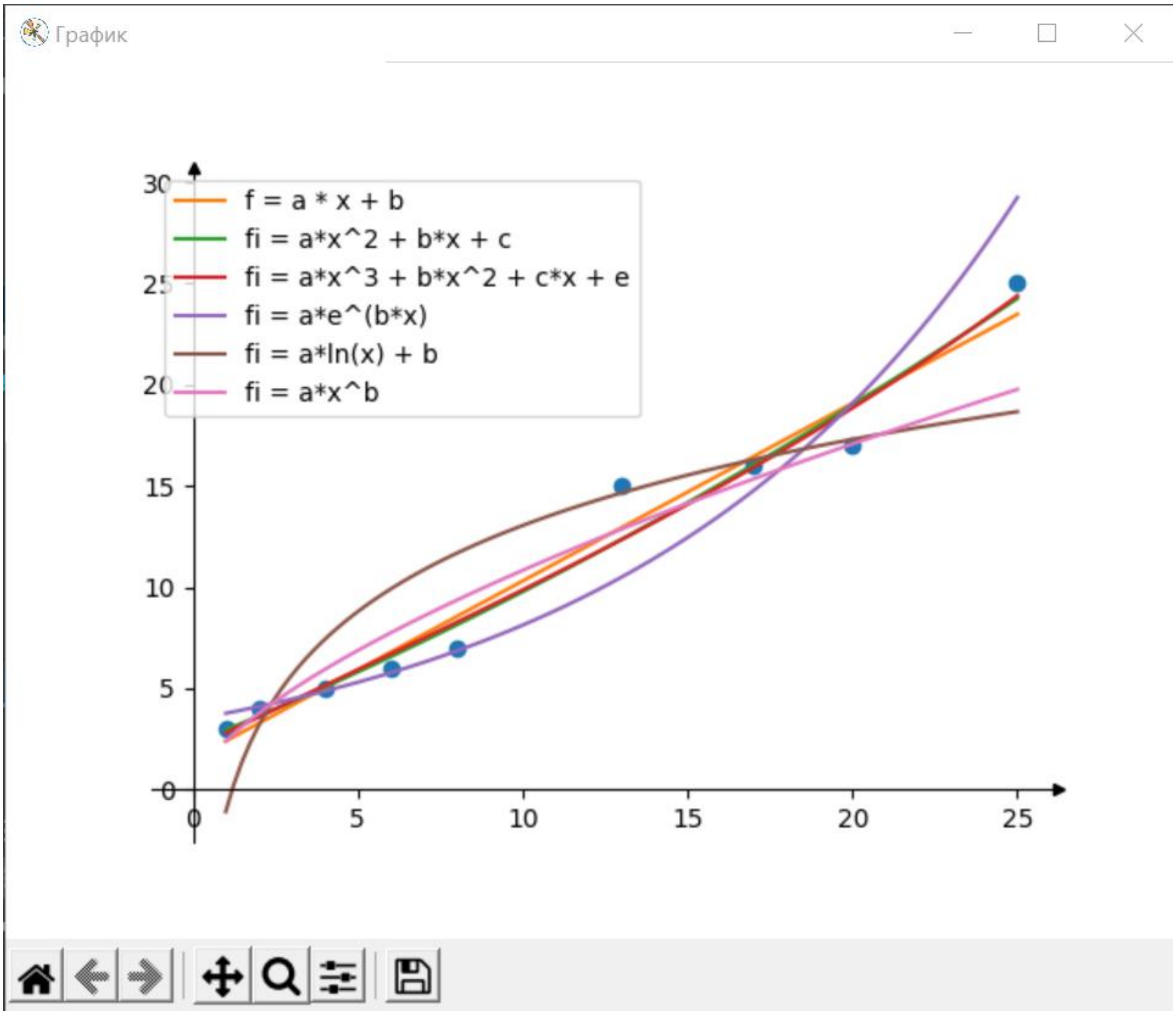
```
                    [sxxxy, sxxxx, sxxxxx, sxxxxxx]])
    d2 = solve_det([[n, sy, sxx, sxxx],
                    [sx, sxy, sxxx, sxxxx],
                    [sxx, sxxy, sxxxx, sxxxxx],
                    [sxxx, sxxxy, sxxxxx, sxxxxxx]])
    d3 = solve_det([[n, sx, sy, sxxx],
                    [sx, sxx, sxy, sxxxx],
                    [sxx, sxxx, sxxy, sxxxxx],
                    [sxxx, sxxxx, sxxxy, sxxxxxx]])
    d4 = solve_det([[n, sx, sxx, sy],
                   [sx, sxx, sxxx, sxy],
                   [sxx, sxxx, sxxxx, sxxy],
                   [sxxx, sxxxx, sxxxxx, sxxxy]])
    if d == 0:
        return None
    e = d1 / d
    c = d2 / d
    b = d3 / d
    a = d4 / d
    data['e'] = e
    data['c'] = c
    data['b'] = b
    data['a'] = a
    f = lambda z: a * (z ** 3) + b * (z ** 2) + c * z + e
    data['f'] = f
    data['str(f)'] = "fi = a*x^3 + b*x^2 + c*x + e"
    data['s'] = calc_s(dots, f)
    data['stdev'] = calc_stdev(dots, f)
    return data


def exp_approximation(dots):
    data = {}
    n = len(dots)
    x = [dot[0] for dot in dots]
    y = []
    for dot in dots:
        if dot[1] <= 0:
            return None
        y.append(dot[1])
    lin_y = [log(y[i]) for i in range(n)]
    lin_result = linear_approximation([(x[i], lin_y[i]) for i in range(n)])
    a = exp(lin_result['b'])
    b = lin_result['a']
    data['a'] = a
    data['b'] = b
    f = lambda z: a * exp(b * z)
    data['f'] = f
    data['str(f)'] = "fi = a*e^(b*x)"
    data['s'] = calc_s(dots, f)
    data['stdev'] = calc_stdev(dots, f)
    return data


def log_approximation(dots):
    data = {}
    n = len(dots)
    x = []
    for dot in dots:
        if dot[0] <= 0:
            return None
        x.append(dot[0])
    y = [dot[1] for dot in dots]
    lin_x = [log(x[i]) for i in range(n)]
```

```
    lin_result = linear_approximation([(lin_x[i], y[i]) for i in range(n)])
    a = lin_result['a']
    b = lin_result['b']
    data['a'] = a
    data['b'] = b
    f = lambda z: a * log(z) + b
    data['f'] = f
    data['str(f)'] = "fi = a*ln(x) + b"
    data['s'] = calc_s(dots, f)
    data['stdev'] = calc_stdev(dots, f)
    return data


def pow_approximation(dots):
    data = {}
    n = len(dots)
    x = []
    for dot in dots:
        if dot[0] <= 0:
            return None
        x.append(dot[0])
    y = []
    for dot in dots:
        if dot[1] <= 0:
            return None
        y.append(dot[1])
    lin_x = [log(x[i]) for i in range(n)]
    lin_y = [log(y[i]) for i in range(n)]
    lin_result = linear_approximation([(lin_x[i], lin_y[i]) for i in
range(n)])
    a = exp(lin_result['b'])
    b = lin_result['a']
    data['a'] = a
    data['b'] = b
    f = lambda z: a * (z ** b)
    data['f'] = f
    data['str(f)'] = "fi = a*x^b"
    data['s'] = calc_s(dots, f)
    data['stdev'] = calc_stdev(dots, f)
    return data


def plot(x, y, plot_x, plot_ys, labels):
    plt.gcf().canvas.manager.set_window_title("График")
    ax = plt.gca()
    ax.spines['left'].set_position('zero')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    ax.plot(1, 0, marker=">", ms=5, color='k',
            transform=ax.get_yaxis_transform(), clip_on=False)
    ax.plot(0, 1, marker="^", ms=5, color='k',
            transform=ax.get_xaxis_transform(), clip_on=False)
    plt.plot(x, y, 'o')
    for i in range(len(plot_ys)):
        plt.plot(plot_x, plot_ys[i], label=labels[i])
    plt.legend()
    plt.show()


def get_input():
    data = {'dots': []}
    print("Вводите координаты через пробел, каждая точка с новой строки")
    print("Чтобы закончить, введите 'END'")
```

```
    while True:
        read = input().strip()
        if read == 'END':
            if len(data['dots']) < 2:
                print("Минимальное количество точек - 2! Введите еще!")
            else:
                break
        cur_dot = tuple(map(float, read.split()))
        if len(cur_dot) != 2:
            print("Введите точку повторно - координаты некорректны!")
        else:
            data['dots'].append(cur_dot)
    return data


def get_file():
    data = {'dots': []}
    with open('input.txt', 'rt', encoding='UTF-8') as file:
        for line in file:
            cur_dot = tuple(map(float, line.split()))
            if len(cur_dot) != 2:
                return None
            data['dots'].append(cur_dot)
        if len(data['dots']) < 2:
            return None
    return data


def main():
    print("Лабораторная работа №4")
    print("Вариант №18")
    print("Аппроксимация функции методом наименьших квадратов")
    print("Ввести исходные данные с клавиатуры (+) или из файла (-)?")
    read_type = input("Режим ввода: ")
    while read_type != '+' and read_type != '-':
        print("Введите '+' или '-' для выбора способа ввода!")
        read_type = input("Режим ввода: ")
    data = None
    if read_type == '+':
        data = get_input()
    else:
        data = get_file()
        if data is None:
            print("Данные в файле некорректные!")
            exit(0)
    answers = []
    for answer in [linear_approximation(data['dots']),
sqrt_approximation(data['dots']),
                   cube_approximation(data['dots']),
exp_approximation(data['dots']),
                   log_approximation(data['dots']),
pow_approximation(data['dots'])]:
        if answer is not None:
            answers.append(answer)
    print("\n%30s%30s" % ("Вид функции", "Ср. отклонение"))
    print("-" * 60)
    for answer in answers:
        print("%30s%30.4f" % (answer['str(f)'], answer['stdev']))
    print("Коэффициент корреляции Пирсона для линейной апроксимации: ",
cor_pirson(data['dots']))
    x = np.array([dot[0] for dot in data['dots']])
    y = np.array([dot[1] for dot in data['dots']])
    plot_x = np.linspace(np.min(x), np.max(x), 100)
    plot_y = []
```

```
    labels = []
    for answer in answers:
        plot_y.append([answer['f'](x) for x in plot_x])
        labels.append(answer['str(f)'])
    plot(x, y, plot_x, plot_y, labels)
    best_answer = min(answers, key=lambda z: z['stdev'])
    print("Наилучшая аппроксимирующая функция:")
    print(f"{best_answer['str(f)']}, где")
    print(f"  a = {round(best_answer['a'], 4)}")
    print(f"  b = {round(best_answer['b'], 4)}")
    print(f"  c = {round(best_answer['c'], 4) if 'c' in best_answer else '-
'}")


main()
```

Результаты выполнения программы



```
Лабораторная работа №4
Вариант №18
Аппроксимация функции методом наименьших квадратов
Ввести исходные данные с клавиатуры (+) или из файла (-)?
Режим ввода: -

                   Вид функции                Ср. отклонение
------------------------------------------------------------
                 f = a * x + b                        1.2921
          fi = a*x^2 + b*x + c                        1.2142
  fi = a*x^3 + b*x^2 + c*x + e                        1.2085
                fi = a*e^(b*x)                        2.2344
              fi = a*ln(x) + b                        3.3465
                    fi = a*x^b                        2.1654
Коэффициент корреляции Пирсона для линейной апроксимации:  0.9998062938788294
Наилучшая аппроксимирующая функция:
fi = a*x^3 + b*x^2 + c*x + e, где
  a = 0.0003
  b = -0.0044
  c = 0.7965
```

Вывод

В результате выполнения данной лабораторной работой я познакомился с аппроксимациями функции методом наименьших квадратов и реализовал их на языке программирования Python, закрепив знания.