Министерство высшего образования и науки Российской Федерации

Национальный научно-исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по дисциплине

«Вычислительная математика»

Вариант 11

Работу выполнил:

Макеев Роман Ильич

Группа P3208

Преподаватель:

Машина Екатерина Алексеевна

Санкт-Петербург

2024

Цель работы:

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

Рабочие формулы используемых методов:

Вычислительная часть:

$$y = \frac{5x}{x^4 + 11}$$

$$x \in [-2; 0] \quad h = 0.2$$

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
y_i	-0.37	-0.419	-0.456	-0.472	-0.459	-0.417	-0.351	-0.27	-0.181	-0.091	0

Линейное приближение:

$$\phi_1(x) = ax + b$$

$$SX = \sum_{i=1}^{n} x_i = -11 \quad SXX = \sum_{i=1}^{n} x_i^2 = 15.4$$

$$SY = \sum_{i=1}^{n} y_i = -3.486 \quad SXY = \sum_{i=1}^{n} x_y y_i = 4.386$$

$$\begin{cases} a = \dfrac{SXY \cdot n - SX \cdot SY}{SXX \cdot n - SX \cdot SX} \\ b = \dfrac{SXX \cdot SY - SX \cdot SXY}{SXX \cdot n - SX \cdot SX} \end{cases} = \begin{cases} a = \dfrac{4.386 \cdot 11 - 11 \cdot 3.486}{15.4 \cdot 11 - 121} \\ b = \dfrac{15.4 \cdot (-3.486) + 11 \cdot 4.386}{15.4 \cdot 11 - 121} \end{cases} = \begin{cases} a \approx 0.205 \\ b \approx -0.112 \end{cases}$$

$$\phi_1(x) = 0.205x - 0.112$$

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
y_i	-0.37	-0.419	-0.456	-0.472	-0.459	-0.417	-0.351	-0.27	-0.181	-0.091	0
ϕ_{1_i}	-0.522	-0.481	-0.44	-0.399	-0.358	-0.317	-0.276	-0.235	-0.194	-0.153	-0.112
ε_i	-0.152	-0.062	0.016	0.073	0.101	0.1	0.075	0.035	-0.013	-0.062	-0.112

$$\sigma_1 = \sqrt{\frac{\sum_{i=1}^{n} \varepsilon_i^2}{n}} \approx 0.083$$

Квадратичное приближение:

$$\phi_2(x) = a + bx + cx^2$$

$$\sum_{i=1}^{n} x_i = -11 \quad \sum_{i=1}^{n} x_i^2 = 15.4 \quad \sum_{i=1}^{n} x_i^3 = -24.2$$

$$\sum_{i=1}^{n} x_i^4 = 40.533 \quad \sum_{i=1}^{n} y_i = -3.486 \quad \sum_{i=1}^{n} x_i y_i = 4.386$$

$$\sum_{i=1}^{n} x_i^2 y_i = -6.362$$

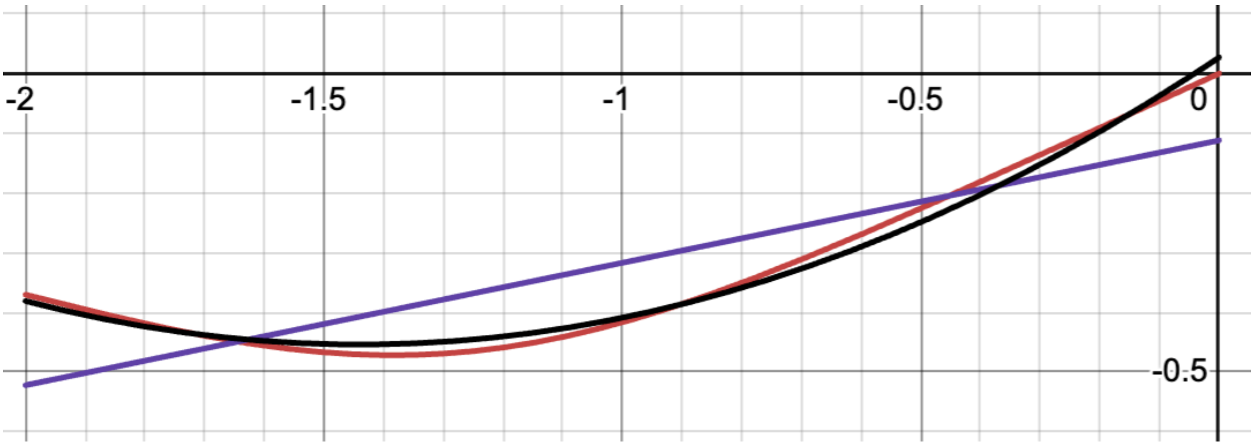
$$\begin{cases} n \cdot a + SX \cdot b + SXX \cdot c = SY \\ SX \cdot a + SXX \cdot b + S3X \cdot c = SXY \\ SXX \cdot a + S3X \cdot b + S4X \cdot c = SXXY \end{cases} = \begin{cases} 11 \cdot a - 11 \cdot b + 15.4 \cdot c = -3.486 \\ -11 \cdot a + 15.4 \cdot b - 24.2 \cdot c = 4.386 \\ 15.4 \cdot a - 24.2 \cdot b + 40.533 \cdot c = -6.362 \end{cases} = \begin{cases} a \approx 0.027 \\ b \approx 0.668 \\ c \approx 0.232 \end{cases}$$

$$\phi_2(x) = 0.232x^2 + 0.668x + 0.027$$

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
y_i	-0.37	-0.419	-0.456	-0.472	-0.459	-0.417	-0.351	-0.27	-0.181	-0.091	0
ϕ_{1_i}	-0.383	-0.425	-0.449	-0.454	-0.441	-0.409	-0.359	-0.291	-0.203	-0.098	-0.026
ε_i	-0.013	-0.006	0.007	0.018	0.018	0.008	-0.008	-0.021	-0.022	-0.007	-0.026

$$\sigma_2 = \sqrt{\frac{\sum_{i=1}^{n} \varepsilon_i^2}{n}} \approx 0.016$$

Таким образом наилучшее приближение - квадратичное, так как $\sigma_1 > \sigma_2$



Исходная функция
Линейное приближение
Квадратичное приближение

Листинг программы:

Вспомогательные методы

Расчет коэффициента Пирсона:

```
def calc_pirson_kf(def_data: DataTable) -> float:
    x_list: list[float] = def_data.x_list
    y_list: list[float] = def_data.y_list

    n: int = len(x_list)
    avg_x = avg(x_list)
    avg_y = avg(y_list)

    return (sum([(x_list[i] - avg_x) * (y_list[i] - avg_y) for i in range(n)]) /
                    math.sqrt(sum([(x_list[i] - avg_x) ** 2 for i in range(n)]) *
                              sum([(y_list[i] - avg_y) ** 2 for i in range(n)])
                              ))
```

Расчет значений $\phi(x_i)$ и ε_i:

```
def get_def_data(points: PointTable, func: callable) -> DataTable:
    x_list = points.get_all_x()
    y_list = points.get_all_y()
    phi_x = [func(x) for x in x_list]
    eps = [phi_x[i] - y_list[i] for i in range(points.n)]

    return DataTable(x_list, y_list, phi_x, eps)
```

Расчет СКО:

```
def calc_sko(eps: list[float]) -> float:
    n: int = len(eps)
    return math.sqrt(sum([eps[i] ** 2 for i in range(n)]) / n)
```

Расчет коэффициента детерминации:

```
def calc_det_kf(def_data: DataTable) -> float:
    phi_list: list[float] = def_data.phi_x
    y_list: list[float] = def_data.y_list

    n: int = len(phi_list)
    avg_phi = avg(phi_list)

    return (1 - sum([(y_list[i] - phi_list[i]) ** 2 for i in range(n)]) /
            sum([(y_list[i] - avg_phi) ** 2 for i in range(n)]))
```

Расчет параметров для линейной зависимости:

```python
def calc_linear_kfs(points: PointTable) -> tuple[float, float]:
    sx, sy, sxx, sxy, n = (points.sx(), points.sy(),
                           points.sxx(), points.sxy(), points.n)

    eq: Equation = Equation.create([
        [sxx, sx],
        [sx, n],
        [sxy, sy]
    ])
    eq.solve()
    a, b = eq.answers.elems
    return a, b
```

Методы аппроксимации

Линейная:

```python
def approx_linear(points: PointTable) -> ApproxRes:
    a, b = calc_linear_kfs(points)

    callback: callable = lambda x: a * x + b
    func_view: str = f'{a:.3g}x'
    if b > 0:
        func_view += f' + {b:.3g}'
    elif b < 0:
        func_view += f' - {-b:.3g}x'

    def_data: DataTable = get_def_data(points, callback)

    return ApproxRes(
        type='Линейная аппроксимация',
        data=LinearApproxData(
            func_view=func_view,
            callback=callback,
            sko=calc_sko(def_data.eps),
            x_list=def_data.x_list,
            y_list=def_data.y_list,
            phi_x=def_data.phi_x,
            eps=def_data.eps,
            pirson_kf=calc_pirson_kf(def_data),
            det_kf=calc_det_kf(def_data)
        ),
        error_message=None
    )
```

Квадратичная:

```python
def approx_quad(points: PointTable) -> ApproxRes:
    sx, sxx, s3x, s4x, sy, sxy, sxxy, n = \
        (points.sx(), points.sxx(), points.s3x(), points.s4x(),
        points.sy(), points.sxy(), points.sxxy(), points.n)

    eq: Equation = Equation.create([
        [s4x, s3x, sxx],
        [s3x, sxx, sx],
        [sxx, sx, n],
        [sxxy, sxy, sy]
    ])
    eq.solve()
    a, b, c = eq.answers.elems

    callback: callable = lambda x: a * x * x + b * x + c
    func_view: str = f'{a:.3g}x^2'
    if b > 0:
        func_view += f' + {b:.3g}x'
    elif b < 0:
        func_view += f' - {-b:.3g}x'
    if c > 0:
        func_view += f' + {c:.3g}'
    elif c < 0:
        func_view += f' - {-c:.3g}'

    def_data: DataTable = get_def_data(points, callback)
```

```
    return ApproxRes(
        type='Квадратичная аппроксимация',
        data=ApproxData(
            func_view=func_view,
            callback=callback,
            sko=calc_sko(def_data.eps),
            x_list=def_data.x_list,
            y_list=def_data.y_list,
            phi_x=def_data.phi_x,
            eps=def_data.eps,
            det_kf=calc_det_kf(def_data)
        ),
        error_message=None
    )
```

Кубическая:

```
def approx_cube(points: PointTable) -> ApproxRes:
    sx, sxx, s3x, s4x, s5x, s6x, sy, sxy, sxxy, s3xy, n = \
        (points.sx(), points.sxx(), points.s3x(), points.s4x(),
        points.s5x(), points.s6x(), points.sy(), points.sxy(),
        points.sxxy(), points.s3xy(), points.n)
    eq: Equation = Equation.create([
        [s6x, s5x, s4x, s3x],
        [s5x, s4x, s3x, sxx],
        [s4x, s3x, sxx, sx],
        [s3x, sxx, sx, n],
        [s3xy, sxxy, sxy, sy]
    ])
    eq.solve()
    a, b, c, d = eq.answers.elems

    callback: callable = lambda x: a * (x ** 3) + b * (x ** 2) + c * x + d
    func_view: str = f'{a:.3g}x^3'
    if b > 0:
        func_view += f' + {b:.3g}x^2'
    elif b < 0:
        func_view += f' - {-b:.3g}x^2'
    if c > 0:
        func_view += f' + {c:.3g}x'
    elif c < 0:
        func_view += f' - {-c:.3g}x'
    if d > 0:
        func_view += f' + {d:.3g}'
    elif d < 0:
        func_view += f' - {-d:.3g}'

    def_data: DataTable = get_def_data(points, callback)

    return ApproxRes(
        type='Кубическая аппроксимация',
        data=ApproxData(
            func_view=func_view,
            callback=callback,
            sko=calc_sko(def_data.eps),
            x_list=def_data.x_list,
            y_list=def_data.y_list,
            phi_x=def_data.phi_x,
            eps=def_data.eps,
            det_kf=calc_det_kf(def_data)
        ),
        error_message=None
    )
```

Экспоненциальная:

```
def approx_exp(points: PointTable) -> ApproxRes:
    if not points.log_y_is_safe():
        return ApproxRes(
            type='Экспоненциальная аппроксимация',
            data=None,
            error_message="Can't approximate with negative ordinates"
        )
    points_copy: PointTable = points.copy()
    for i in range(points.n):
        points_copy[i].y = math.log(points_copy[i].y)
```

```
    b, A = calc_linear_kfs(points_copy)
    a = math.exp(A)

    callback: callable = lambda x: a * math.exp(b * x)
    func_view: str = f'{a:.3g}e^({b:.3g}x)'
    def_data: DataTable = get_def_data(points, callback)

    return ApproxRes(
        type='Экспоненциальная аппроксимация',
        data=ApproxData(
            func_view=func_view,
            callback=callback,
            sko=calc_sko(def_data.eps),
            x_list=def_data.x_list,
            y_list=def_data.y_list,
            phi_x=def_data.phi_x,
            eps=def_data.eps,
            det_kf=calc_det_kf(def_data)
        ),
        error_message=None
    )
```

Логарифмическая:

```
def approx_log(points: PointTable) -> ApproxRes:
    if not points.log_x_is_safe():
        return ApproxRes(
            type='Логарифмическая аппроксимация',
            data=None,
            error_message="Can't approximate with negative abscisses"
        )

    points_copy: PointTable = points.copy()
    for i in range(points.n):
        points_copy[i].x = math.log(points_copy[i].x)

    a, b = calc_linear_kfs(points_copy)

    callback: callable = lambda x: a * math.log(x) + b
    func_view: str = f'{a:.3g}ln(x)'
    if b > 0:
        func_view += f' + {b:.3g}'
    elif b < 0:
        func_view += f' - {-b:.3g}'
    def_data: DataTable = get_def_data(points, callback)

    return ApproxRes(
        type='Логарифмическая аппроксимация',
        data=ApproxData(
            func_view=func_view,
            callback=callback,
            sko=calc_sko(def_data.eps),
            x_list=def_data.x_list,
            y_list=def_data.y_list,
            phi_x=def_data.phi_x,
            eps=def_data.eps,
            det_kf=calc_det_kf(def_data)
        ),
        error_message=None
    )
```

Степенная:

```
def approx_step(points: PointTable) -> ApproxRes:
    if not points.log_x_is_safe() or not points.log_y_is_safe():
        return ApproxRes(
            type='Степпеная аппроксимация',
            data=None,
            error_message="Can't approximate with negative abscisses or negative 
ordinates"
        )

    points_copy: PointTable = points.copy()
    for i in range(points.n):
        points_copy[i].x = math.log(points_copy[i].x)
        points_copy[i].y = math.log(points_copy[i].y)
```

```python
b, A = calc_linear_kfs(points_copy)
a = math.exp(A)

callback: callable = lambda x: a * (x ** b)
func_view: str = f'{a:.3g}x^({b:.3g})'
def_data: DataTable = get_def_data(points, callback)

return ApproxRes(
    type='Степеная аппроксимация',
    data=ApproxData(
        func_view=func_view,
        callback=callback,
        sko=calc_sko(def_data.eps),
        x_list=def_data.x_list,
        y_list=def_data.y_list,
        phi_x=def_data.phi_x,
        eps=def_data.eps,
        det_kf=calc_det_kf(def_data)
    ),
    error_message=None
)
```

Результаты работы программы:

1.

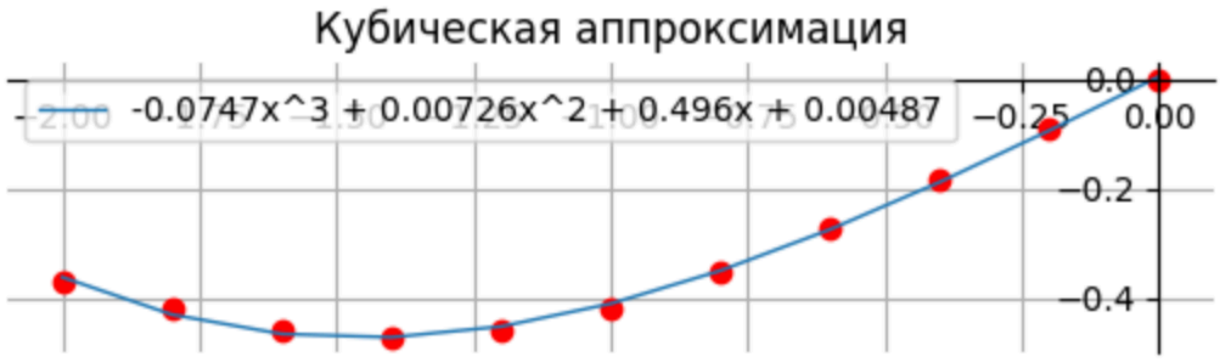
Ввод

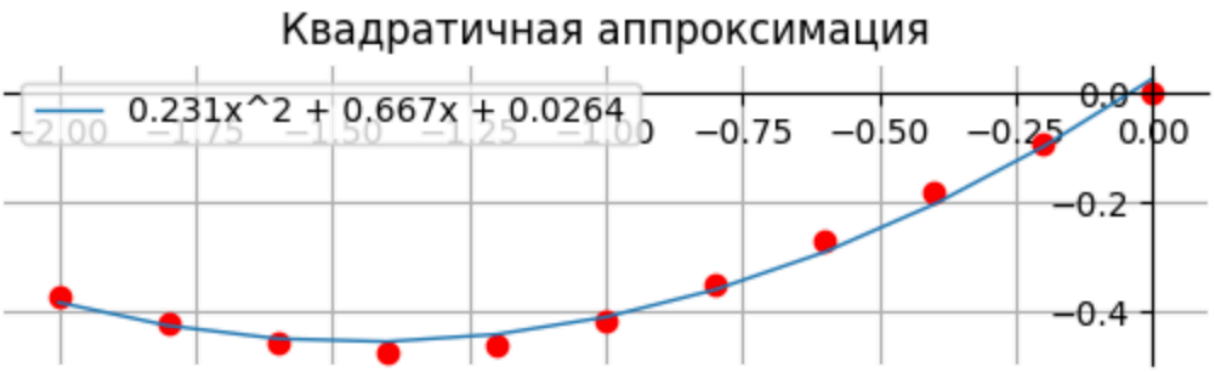
```
-1 -2 -1.8 -1.6 -1.4 -1.2 -0.8 -0.6 -0.4 -0.2 0
-0.417 -0.37 -0.419 -0.456 -0.472 -0.459 -0.351 -0.27 -0.181 -0.091 0
```

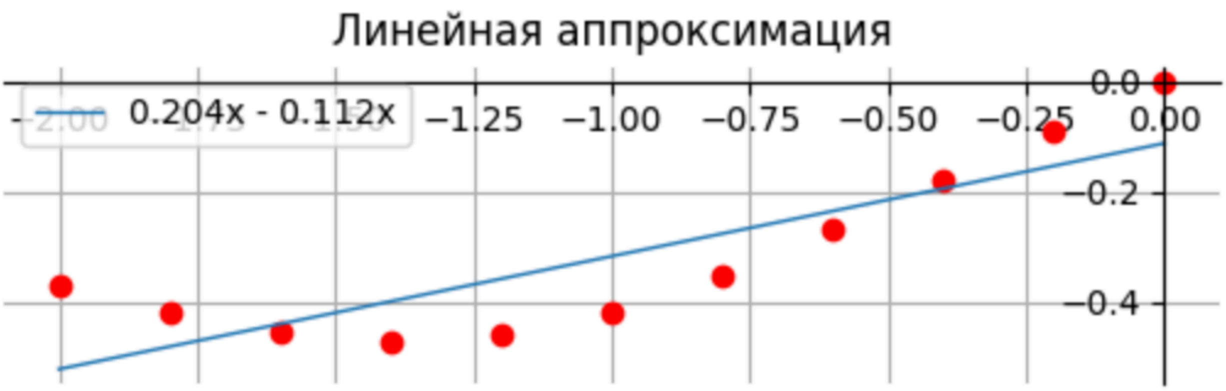
Вывод

```
Type input filename -> test
Best approximator is Кубическая аппроксимация with sko = 0.006574
```

```
-- Линейная аппроксимация
phi: 0.204x - 0.112x
sko: 0.0832
det_kf: 0.707
pirson: 0.841

-- Квадратичная аппроксимация
phi: 0.231x^2 + 0.667x + 0.0264
sko: 0.0156
det_kf: 0.99

-- Кубическая аппроксимация
phi: -0.0747x^3 + 0.00726x^2 + 0.496x + 0.00487
sko: 0.00657
det_kf: 0.998

-- Экспоненциальная аппроксимация
    ERROR: Can't approximate with negative ordinates

-- Логарифмическая аппроксимация
    ERROR: Can't approximate with negative abscisses

-- Степеная аппроксимация
    ERROR: Can't approximate with negative abscisses or negative ordinates
```







2.

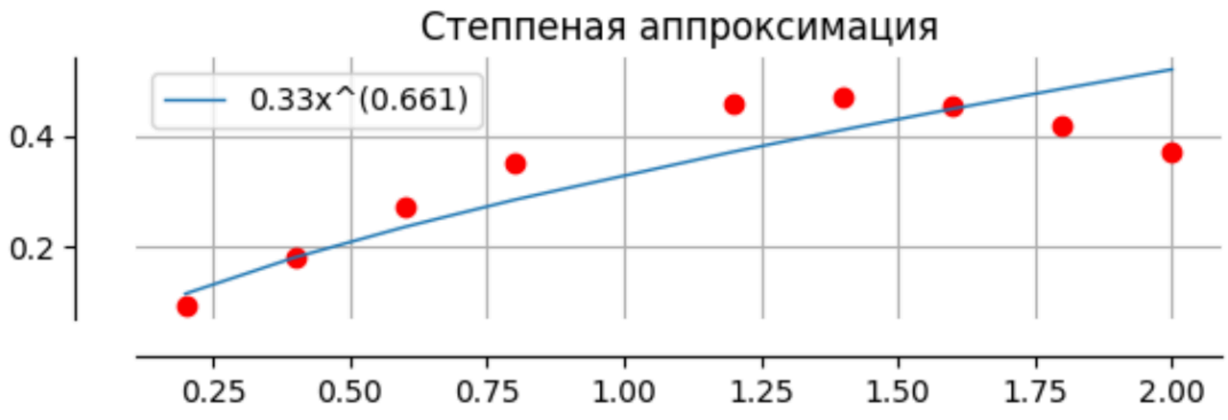
Ввод

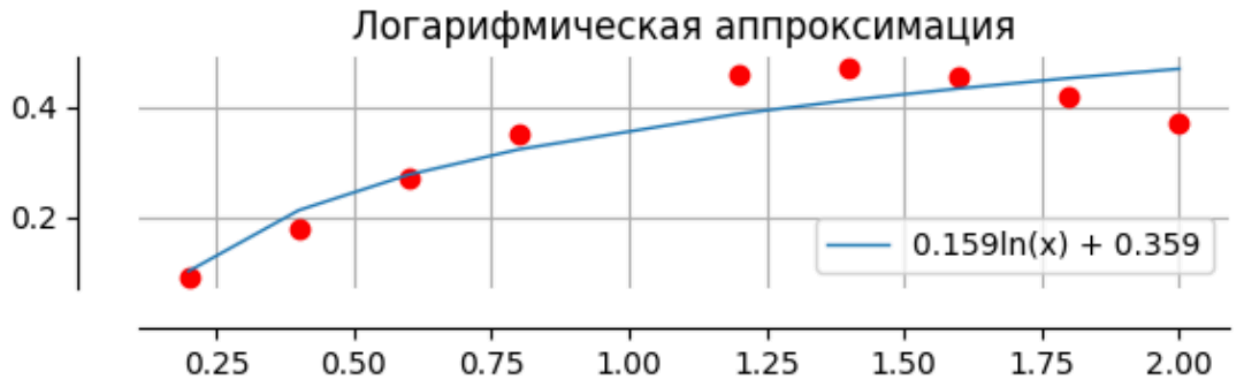
```
0.2 0.4 0.6 0.8 1.2 1.4 1.6 1.8 2
0.091 0.181 0.27 0.351 0.459 0.472 0.456 0.419 0.37
```

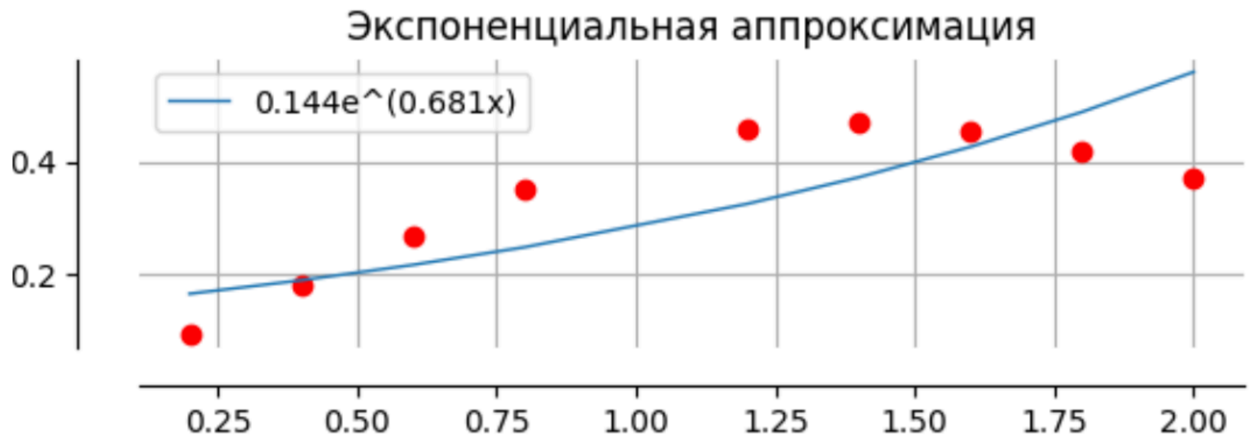
Вывод

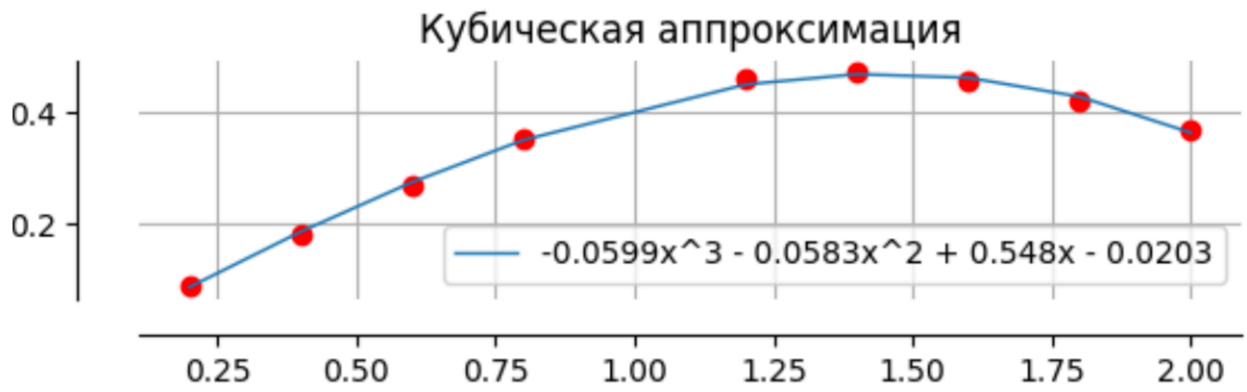
```
Type input filename -> test2
Best approximator is Кубическая аппроксимация with sko = 0.005969
```

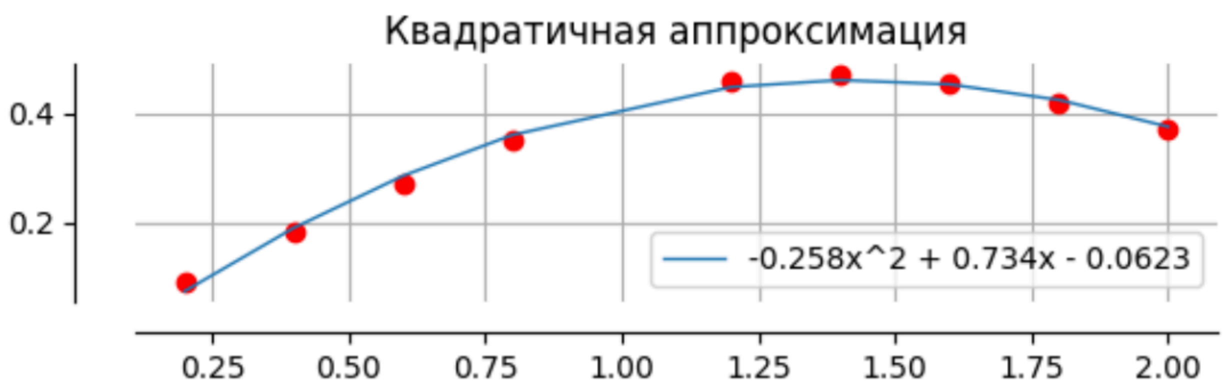
```
-- Линейная аппроксимация
phi: 0.17x + 0.152
sko: 0.0742
det_kf: 0.657
pirson: 0.811

-- Квадратичная аппроксимация
phi: -0.258x^2 + 0.734x - 0.0623
sko: 0.0106
det_kf: 0.993

-- Кубическая аппроксимация
phi: -0.0599x^3 - 0.0583x^2 + 0.548x - 0.0203
sko: 0.00597
det_kf: 0.998

-- Экспоненциальная аппроксимация
phi: 0.144e^(0.681x)
sko: 0.0995
det_kf: 0.385

-- Логарифмическая аппроксимация
phi: 0.159ln(x) + 0.359
sko: 0.0495
det_kf: 0.847

-- Степпеная аппроксимация
phi: 0.33x^(0.661)
sko: 0.0706
det_kf: 0.689
```

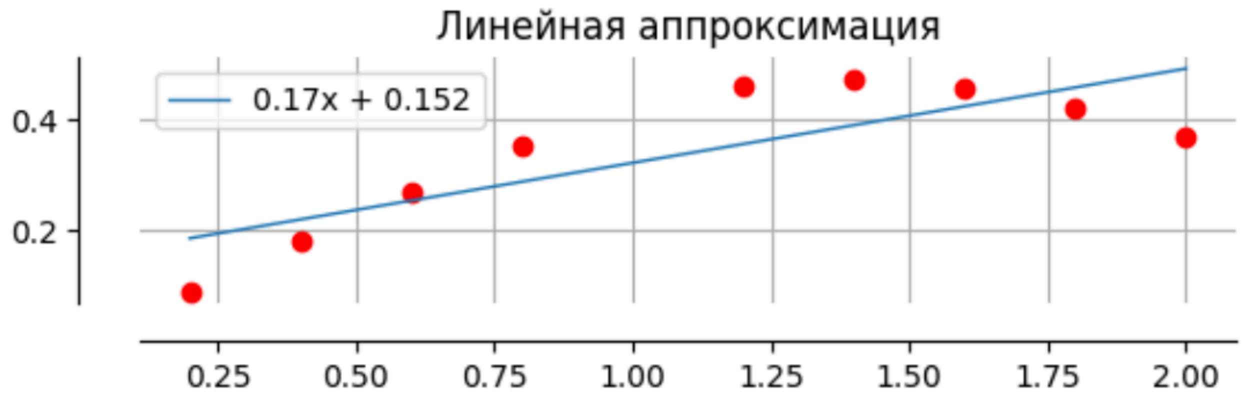



Логарифмическая аппроксимация
0.159ln(x) + 0.359
0.4
0.2
0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00


Экспоненциальная аппроксимация
0.144e^(0.681x)
0.4
0.2
0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00


Кубическая аппроксимация
-0.0599x^3 - 0.0583x^2 + 0.548x - 0.0203
0.4
0.2
0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00


Квадратичная аппроксимация
-0.258x^2 + 0.734x - 0.0623
0.4
0.2
0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00



Выводы:

Создана программа, аппроксимирующая 6 различных типов функций методом наименьших квадратов