Министерство образования и науки Российской Федерации федеральное государственное автономное образовательное учреждение высшего образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет «Программной инженерии и компьютерной техники.»

Вычислительная математика

Лабораторная работа №4
“Аппроксимация функции методом наименьших квадратов”
Вариант №3

Выполнил
Григорьев Давид Владимирович
Группа: P3215
Проверила
Малышева Татьяна Алексеевна

Санкт-Петербург 2025

Содержание

1 Вычислительная часть

1. Таблица табулирования функции

x	$y = \dfrac{4x}{x^4+3}$
-2.0	$\dfrac{-8}{16+3} = -0.421$
-1.8	$\dfrac{-7.2}{10.4976+3} \approx -0.596$
-1.6	$\dfrac{-6.4}{6.5536+3} \approx -0.685$
-1.4	$\dfrac{-5.6}{3.8416+3} \approx -0.770$
-1.2	$\dfrac{-4.8}{2.0736+3} \approx -0.837$
-1.0	$\dfrac{-4}{1+3} = -1.000$
-0.8	$\dfrac{-3.2}{0.4096+3} \approx -0.906$
-0.6	$\dfrac{-2.4}{0.1296+3} \approx -0.753$
-0.4	$\dfrac{-1.6}{0.0256+3} \approx -0.521$
-0.2	$\dfrac{-0.8}{0.0016+3} \approx -0.265$
0.0	$\dfrac{0}{0+3} = 0$

Таблица 1: Табличные значения функции $f(x) = \frac{4x}{x^4+3}$ на интервале $x \in [-2, 0]$ с шагом $h = 0.2$.

2. Линейная аппроксимация $y = ax + b$

Система уравнений метода наименьших квадратов (МНК):

$$\begin{cases} \Sigma y = a\Sigma x + bn \\ \Sigma xy = a\Sigma x^2 + b\Sigma x \end{cases}$$

Расчет сумм:

$$\begin{aligned} \Sigma x &= -11.0, \\ \Sigma y &\approx -7.852, \\ \Sigma xy &\approx 8.216, \\ \Sigma x^2 &= 14.6. \end{aligned}$$

Решение системы:

$$\begin{cases} -7.852 = a(-11.0) + 11b \\ 8.216 = a(14.6) - 11.0b \end{cases} \Rightarrow a \approx 0.189, \quad b \approx -0.637.$$

Линейная модель: $y = 0.189x - 0.637$.

3. Квадратичная аппроксимация $y = ax^2 + bx + c$

Система уравнений МНК:

$$\begin{cases} \Sigma y = a\Sigma x^2 + b\Sigma x + cn \\ \Sigma xy = a\Sigma x^3 + b\Sigma x^2 + c\Sigma x \\ \Sigma x^2 y = a\Sigma x^4 + b\Sigma x^3 + c\Sigma x^2 \end{cases}$$

Расчет сумм:

$$\Sigma x^3 \approx -21.044,$$
$$\Sigma x^4 \approx 31.180,$$
$$\Sigma x^2 y \approx -5.394.$$

Решение системы:

$$\begin{cases} -7.852 = 14.6a - 11.0b + 11c \\ 8.216 = -21.044a + 14.6b - 11.0c \\ -5.394 = 31.180a - 21.044b + 14.6c \end{cases} \Rightarrow a \approx 0.123, \quad b \approx 0.341, \quad c \approx -0.768.$$

Квадратичная модель: $y = 0.123x^2 + 0.341x - 0.768$.

4. Среднеквадратические отклонения

$$S = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_{\text{approx},i})^2.$$

- Линейная модель: $S_{\text{lin}} \approx 0.087$.
- Квадратичная модель: $S_{\text{quad}} \approx 0.012$.

5. Наилучшее приближение

Квадратичная модель обеспечивает меньшее среднеквадратическое отклонение ($S_{\text{quad}} < S_{\text{lin}}$), поэтому она является наилучшим приближением.

6. Графики

- **Линейная аппроксимация:** Прямая с наклоном 0.189.
- **Квадратичная аппроксимация:** Парабола, повторяющая поведение исходной функции.
- **Исходная функция:** $f(x) = \frac{4x}{x^4+3}$ — нечётная, сингулярностей нет.

2 Вывод программы

2.1 Пример 1

```
1 2 3 4 5 6
0 0.6931471806 1.0986122887 1.3862943611 1.6094379124 1.7917594692

--- Квадратичная ---
Формула: y = -0.058632x² + 0.753151x + -0.650229
Мера отклонения: S = 0.011590
Среднеквадратичное отклонение: delta = 0.043950
Достоверность аппроксимации: R² = 0.994721
R2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Линейная ---
Формула: y = 0.342724x + -0.102993
Мера отклонения: S = 0.139933
Среднеквадратичное отклонение: delta = 0.152716
Достоверность аппроксимации: R² = 0.936263
```

```
0.75 <= R2 < 0.95 -> Удовлетворительная аппроксимация, модель в целом адекватно описывае
Коэффицент корреляции: r = 0.967607
1 > |r| >= 0.9 -> Связь между переменными весьма высокая

--- Кубическая ---
Формула: y = 0.012908x³ + -0.194163x² + 1.162326x + -0.975504
Мера отклонения: S = 0.000793
Среднеквадратичное отклонение: delta = 0.011499
Достоверность аппроксимации: R² = 0.999639
R2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Степенная ---
Ошибка: x или y содержит неположительные значения. Логарифмирование невозможно.

--- Экспоненциальная ---
Ошибка: y содержит неположительные значения. Логарифмирование невозможно.

--- Логарифмическая ---
Формула: y = 1.000000 * lnx + 0.000000
Мера отклонения: S = 0.000000
Среднеквадратичное отклонение: delta = 0.000000
Достоверность аппроксимации: R² = 1.000000
R2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Пятая степень ---
Формула: y = 0.000726x^5 + -0.015751x^4 + 0.138637x³ + -0.647224x² + 1.878121x + -1.3545
Мера отклонения: S = 0.000000
Среднеквадратичное отклонение: delta = 0.000000
Достоверность аппроксимации: R² = 1.000000
R2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Подсчёт результатов ---
Минимальное СКО: 0.000000
Лучшие модели:
- Логарифмическая: (СКО: 0.000000)
- Пятая степень: (СКО: 0.000000)
```

2.2 Пример 2

```
-1 0 1 ^2 3 4 5 6
-1 0 1 8 ^27 64 1^25 ^216

--- Квадратичная ---
Формула: y = 7.500000x^2 + -9.500000x + -7.500000
Мера отклонения: S = 594.000000
Среднеквадратичное отклонение: delta = 8.616844
Достоверность аппроксимации: R^2 = 0.986177
R^2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Линейная ---
Формула: y = ^28.000000x + -15.000000
Мера отклонения: S = 10044.000000
Среднеквадратичное отклонение: delta = 35.433035
Достоверность аппроксимации: R^2 = 0.766^266
0.75 <= R^2 < 0.95 -> Удовлетворительная аппроксимация, модель в целом адекватно описыва
```

```
Коэффицент корреляции: r = 0.875366
0.9 > |r| >= 0.5 -> Связь между переменными высокая

--- Кубическая ---
Формула: y = 1.000000x³ + 0.000000x^2 + -0.000000x + 0.000000
Мера отклонения: S = 0.000000
Среднеквадратичное отклонение: delta = 0.000000
Достоверность аппроксимации: R^2 = 1.000000
R^2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Степенная ---
Ошибка: x или y содержит неположительные значения. Логарифмирование невозможно.

--- Экспоненциальная ---
Ошибка: y содержит неположительные значения. Логарифмирование невозможно.

--- Логарифмическая ---
Ошибка: x содержит неположительные значения. Логарифмирование невозможно.

--- Пятая степень ---
Формула: y = 0.000000x^5 + -0.000000x^4 + 1.000000x³ + -0.000000x^2 + -0.000000x + 0.000
Мера отклонения: S = 0.000000
Среднеквадратичное отклонение: delta = 0.000000
Достоверность аппроксимации: R^2 = 1.000000
R^2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Подсчёт результатов ---
Минимальное СКО: 0.000000
Лучшие модели:
- Кубическая: (СКО: 0.000000)
- Пятая степень: (СКО: 0.000000)
```

2.3 Пример 3

```
1 ^2 3 4 5 6
0 0.6931471806 1.09861^2^2887 1.386^2943611 1.60943791^24 1.791759469^2

--- Квадратичная ---
Формула: y = -0.05863^2x^2 + 0.753151x + -0.650^2^29
Мера отклонения: S = 0.011590
Среднеквадратичное отклонение: delta = 0.043950
Достоверность аппроксимации: R^2 = 0.9947^21
R^2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Линейная ---
Формула: y = 0.34^27^24x + -0.10^2993
Мера отклонения: S = 0.139933
Среднеквадратичное отклонение: delta = 0.15^2716
Достоверность аппроксимации: R^2 = 0.936^263
0.75 <= R^2 < 0.95 -> Удовлетворительная аппроксимация, модель в целом адекватно описыва
Коэффицент корреляции: r = 0.967607
1 > |r| >= 0.9 -> Связь между переменными весьма высокая

--- Кубическая ---
Формула: y = 0.01^2908x³ + -0.194163x^2 + 1.16^23^26x + -0.975504
```

```
Мера отклонения: S = 0.000793
Среднеквадратичное отклонение: delta = 0.011499
Достоверность аппроксимации: R^2 = 0.999639
R^2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Степенная ---
Ошибка: x или y содержит неположительные значения. Логарифмирование невозможно.

--- Экспоненциальная ---
Ошибка: y содержит неположительные значения. Логарифмирование невозможно.

--- Логарифмическая ---
Формула: y = 1.000000 * lnx + 0.000000
Мера отклонения: S = 0.000000
Среднеквадратичное отклонение: delta = 0.000000
Достоверность аппроксимации: R^2 = 1.000000
R^2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Пятая степень ---
Формула: y = 0.0007^26x^5 + -0.015751x^4 + 0.138637x³ + -0.647^2^24x^2 + 1.8781^21x + -1
Мера отклонения: S = 0.000000
Среднеквадратичное отклонение: delta = 0.000000
Достоверность аппроксимации: R^2 = 1.000000
R^2 >= 0.95 -> Высокая точность аппроксимации, модель хорошо описывает явление

--- Подсчёт результатов ---
Минимальное СКО: 0.000000
Лучшие модели:
- Логарифмическая: (СКО: 0.000000)
- Пятая степень: (СКО: 0.000000)
```

3 Листинг программы

3.1 main.py

```
from typing import Callable, Dict, Tuple
import numpy as np
import matplotlib.pyplot as plt
from typing import Dict, List
from methods.fifthApprox import fifthApprox
from methods.cubeApprox import cubeApprox
from methods.exponentialApprox import exponentialApprox
from methods.linearApprox import linealApprox
from methods.squareApprox import squareApprox
from methods.powerApprox import powerApprox
from methods.logApprox import logApprox


def readData():
    with open("input.txt", "r") as file:
        x = np.array(list(map(float, file.readline().split())))
        y = np.array(list(map(float, file.readline().split())))
    return x, y


def drawFunctions(
    x,
    y,
    results: List[Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]]],
):

    plt.figure(figsize=(12, 8))

    plt.scatter(x, y, c="red", s=70, label="Исходные данные", zorder=3)

    x_min = min(x)
    x_max = max(x)
    margin = 0.1 * (x_max - x_min)
    x_smooth = np.linspace(x_min - margin, x_max + margin, 500)

    colors = ["blue", "green", "purple", "orange", "brown", "pink", "black"]

    for i, res in enumerate(results):
        if not res:
            continue

        name, dictionary, model = res
        y_smooth = model(x_smooth)

        plt.plot(
            x_smooth,
            y_smooth,
            color=colors[i],
            linewidth=2,
            label=f"{name} (R²={dictionary['R2']:.4f})",
        )
```

```python
    plt.title("Сравнение методов аппроксимации", fontsize=16)
    plt.xlabel("x", fontsize=14)
    plt.ylabel("y", fontsize=14)
    plt.grid(True, linestyle="--", alpha=0.7)
    plt.legend(fontsize=10, loc="best")
    plt.tight_layout()

    plt.savefig("approximations_comparison.png", dpi=300)


def countResult(
    results: List[Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]]],
):
    print("\n--- Подсчёт результатов ---")

    minDelta = float("inf")

    for result in results:
        name, dictionary, func = result
        delta = dictionary.get("delta", float("inf"))
        minDelta = min(minDelta, delta)

    bestModels = []
    for result in results:
        name, dictionary, func = result
        delta = dictionary.get("delta", float("inf"))
        if abs(delta - minDelta) < 1e-6:
            bestModels.append(result)

    print(f"Минимальное СКО: {minDelta:.6f}")
    print("Лучшие модели:")
    if len(bestModels) != 0:
        for model in bestModels:
            name, dictionary, func = model

            print(f"- {name}: (СКО: {dictionary.get('delta', float("inf")):.6f})")
    else:
        print("Решения не найдено")


def main():
    x, y = readData()
    if len(x) != len(y):
        print("ошибка ввода точек")
        return

    results = [
        squareApprox(x, y),
        linealApprox(x, y),
        cubeApprox(x, y),
        powerApprox(x, y),
        exponentialApprox(x, y),
        logApprox(x, y),
        fifthApprox(x, y),
```

```
    ]

    valid_results = [res for res in results if res]

    countResult(valid_results)
    drawFunctions(x, y, valid_results)


if __name__ == "__main__":
    main()
```

3.2 approximation_characteristics.py

```
from typing import Callable, Tuple
import numpy as np

from helpers.interpretationR import interpretR
from helpers.interpretationCorrel import interCorrel
from least_squares_method import least_squares_method


def get_approximation_characteristics(
    x: np.ndarray,
    y: np.ndarray,
    func: Callable[[np.ndarray], np.ndarray],
) -> dict[str, float] | None:

    n = x.size

    f = func(x)
    S = np.sum(np.pow(f - y, 2))
    delta = np.sqrt(S / n)
    f_mean = np.mean(f)
    R2 = 1 - np.sum(np.pow(y - f, 2)) / np.sum(np.pow(y - f_mean, 2))

    print(f"Мера отклонения: S = {S:.6f}")
    print(f"Среднеквадратичное отклонение:  = {delta:.6f}")
    print(f"Достоверность аппроксимации: R² = {R2:.6f}")

    interpretR(R2)
    return {
        "S": S,
        "delta": delta,
        "R2": R2,
    }
```

3.3 least_squares_method.py

```
import numpy as np


import numpy as np


def least_squares_method(X: np.ndarray, Y: np.ndarray, degree: int) -> np.ndarray:
    """
```

```python
    Perform least squares regression to fit a polynomial of degree
    to the data points (X, Y).
    """
    assert X.ndim == 1 and Y.ndim == 1, "X and Y must be 1D arrays"
    assert X.size == Y.size, "X and Y must have the same number of elements"
    assert degree >= 0, "Degree must be positive"

    n = X.size

    def get_x_nth_degree_sum(_degree: int) -> float:
        assert _degree >= 0
        return np.sum(np.power(X, _degree))

    matrix = []
    for i in range(degree + 1):
        row = []
        for j in range(degree + 1):
            row.append(get_x_nth_degree_sum(i + j))
        matrix.append(row)

    matrix = np.array(matrix)

    def get_x_nth_degree_mul_Y_sum(_degree: int) -> float:
        assert _degree >= 0
        return np.sum(np.power(X, _degree) * Y)

    answers = []
    for i in range(degree + 1):
        answers.append(get_x_nth_degree_mul_Y_sum(i))
    answers = np.array(answers)

    return np.linalg.solve(matrix, answers)
```

3.4 tests.py

3.5 helpers/interpretationCorrel.py

3.6 helpers/interpretationR.py

3.7 methods/cubeApprox.py

```python
from typing import Callable, Tuple
import numpy as np

from approximation_characteristics import get_approximation_characteristics
from helpers.interpretationR import interpretR
from least_squares_method import least_squares_method


def cubeApprox(
    x: np.ndarray, y: np.ndarray
) -> Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]] | None:
    print("")
    print("--- Кубическая ---")

    try:
```

```
        solution = least_squares_method(x, y, 3)
    except np.linalg.LinAlgError:
        print("Система уравнений вырождена, решение не существует")
        return None
    if solution is None:
        print("ошибка в вычислении матрицы")
        return None

    def solution_to_string():
        a0, a1, a2, a3 = solution
        return f"y = {a3:.6f}x³ + {a2:.6f}x² + {a1:.6f}x + {a0:.6f}"

    def func(x: np.ndarray):
        a0, a1, a2, a3 = solution
        return a0 + a1 * x + a2 * x**2 + a3 * x**3

    print(f"Формула: {solution_to_string()}")
    res = get_approximation_characteristics(x, y, func)
    if res is not None:
        return ("Кубическая", res, func)
    else:
        return None
```

3.8 methods/exponentialApprox.py

```
from typing import Callable, Tuple
import numpy as np

from approximation_characteristics import get_approximation_characteristics
from helpers.interpretationR import interpretR
from least_squares_method import least_squares_method


def exponentialApprox(
    x: np.ndarray, y: np.ndarray
) -> Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]] | None:
    print("")
    print("--- Экспоненциальная ---")

    if np.any(y <= 0):
        print(
            "Ошибка: y содержит неположительные значения. Логарифмирование невозможно."
        )
        return None

    try:
        solution = least_squares_method(x, np.log(y), 1)
    except np.linalg.LinAlgError:
        print("Система уравнений вырождена, решение не существует")
        return None
    if solution is None:
        print("ошибка в вычислении матрицы")
        return None

    def solution_to_string():
```

```
        # y=ae^{bx}      \ ln
        # ln(y)=ln(a) + bx

        ln_a, b = solution
        a = np.exp(ln_a)

        return f"y = {a:.6f} * e^{b:.6f}x"

    def func(x: np.ndarray):
        a0, a1 = solution
        a0 = np.exp(a0)
        return a0 * np.exp(a1 * x)

    print(f"Формула: {solution_to_string()}")
    res = get_approximation_characteristics(x, y, func)

    if res is not None:
        return ("Экспоненциальная", res, func)
    else:
        return None
```

3.9 methods/fifthApprox.py

```
from typing import Callable, Tuple
import numpy as np

from approximation_characteristics import get_approximation_characteristics
from helpers.interpretationR import interpretR
from least_squares_method import least_squares_method


def fifthApprox(
    x: np.ndarray, y: np.ndarray
) -> Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]] | None:
    print("")
    print("--- Пятая степень ---")

    try:
        solution = least_squares_method(x, y, 5)
    except np.linalg.LinAlgError:
        print("Система уравнений вырождена, решение не существует")
        return None
    if solution is None:
        print("ошибка в вычислении матрицы")
        return None

    def solution_to_string():
        a0, a1, a2, a3, a4, a5 = solution
        return f"y = {a5:.6f}x^5 + {a4:.6f}x^4 + {a3:.6f}x³ + {a2:.6f}x² + {a1:.6f}x + {

    def func(x: np.ndarray):
        a0, a1, a2, a3, a4, a5 = solution
        return a0 + a1 * x + a2 * x**2 + a3 * x**3 + a4 * x**4 + a5 * x**5

    print(f"Формула: {solution_to_string()}")
```

```
    res = get_approximation_characteristics(x, y, func)
    if res is not None:
        return ("Пятая степень", res, func)
    else:
        return None
```

3.10 methods/linearApprox.py

```
from typing import Callable, Tuple
import numpy as np

from approximation_characteristics import get_approximation_characteristics
from helpers.interpretationR import interpretR
from helpers.interpretationCorrel import interCorrel
from least_squares_method import least_squares_method


def linealApprox(
    x: np.ndarray, y: np.ndarray
) -> Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]] | None:
    print("")
    print("--- Линейная ---")

    try:
        solution = least_squares_method(x, y, 1)
    except np.linalg.LinAlgError:
        print("Система уравнений вырождена, решение не существует")
        return None
    if solution is None:
        print("ошибка в вычислении матрицы")
        return None

    def solution_to_string():
        a0, a1 = solution
        return f"y = {a1:.6f}x + {a0:.6f}"

    def func(x: np.ndarray):
        a0, a1 = solution
        return a0 + a1 * x

    print(f"Формула: {solution_to_string()}")
    res = get_approximation_characteristics(x, y, func)

    r = get_correlation_coefficient(x, y)
    print(f"Коэффицент корреляции: r = {r:.6f}")
    interCorrel(r)

    if res is not None:
        return ("Линейная", res, func)
    else:
        return None


def get_correlation_coefficient(x: np.ndarray, y: np.ndarray):
    x_mean = np.average(x)
```

```
    y_mean = np.average(y)
    numerator = np.sum((x - x_mean) * (y - y_mean))
    denominator = np.sqrt(np.sum(np.pow(x - x_mean, 2)) * np.sum(np.pow(y - y_mean, 2)))

    return numerator / denominator
```

3.11 methods/logApprox.py

```
from typing import Callable, Tuple
import numpy as np

from approximation_characteristics import get_approximation_characteristics
from least_squares_method import least_squares_method


def logApprox(
    x: np.ndarray, y: np.ndarray
) -> Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]] | None:
    print("")
    print("--- Логарифмическая ---")

    if np.any(x <= 0):
        print(
            "Ошибка: x содержит неположительные значения. Логарифмирование невозможно."
        )
        return None

    try:
        solution = least_squares_method(np.log(x), y, 1)
    except np.linalg.LinAlgError:
        print("Система уравнений вырождена, решение не существует")
        return None
    if solution is None:
        print("ошибка в вычислении матрицы")
        return None

    def solution_to_string():
        # y=a*ln(x) + b

        b, a = solution

        return f"y = {a:.6f} * lnx + {b:.6f}"

    def func(x: np.ndarray):
        b, a = solution

        return a * np.log(x) + b

    print(f"Формула: {solution_to_string()}")
    res = get_approximation_characteristics(x, y, func)

    if res is not None:
        return ("Логарифмическая", res, func)
    else:
        return None
```

3.12 methods/powerApprox.py

```
from typing import Callable, Tuple
import numpy as np

from approximation_characteristics import get_approximation_characteristics
from helpers.interpretationR import interpretR
from least_squares_method import least_squares_method


def powerApprox(
    x: np.ndarray, y: np.ndarray
) -> Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]] | None:
    print("")
    print("--- Степенная ---")

    if np.any(x <= 0) or np.any(y <= 0):
        print(
            "Ошибка: x или y содержит неположительные значения. Логарифмирование невозмо
        )
        return None

    try:
        solution = least_squares_method(np.log(x), np.log(y), 1)
    except np.linalg.LinAlgError:
        print("Система уравнений вырождена, решение не существует")
        return None
    if solution is None:
        print("ошибка в вычислении матрицы")
        return None

    def solution_to_string():
        # y=ax^b     \ ln
        # ln(y) = ln(a) + b*ln(x)

        ln_a, b = solution

        a = np.exp(ln_a)

        return f"y = {a:.6f} * x^{b:.6f}"

    def func(x: np.ndarray):
        ln_a, b = solution

        a = np.exp(ln_a)

        return a * np.pow(x, b)

    print(f"Формула: {solution_to_string()}")
    res = get_approximation_characteristics(x, y, func)

    if res is not None:
        return ("Степенная", res, func)
    else:
        return None
```

3.13 methods/squareApprox.py

```
from typing import Callable, Tuple
import numpy as np

from approximation_characteristics import get_approximation_characteristics
from helpers.interpretationR import interpretR
from least_squares_method import least_squares_method


def squareApprox(
    x: np.ndarray, y: np.ndarray
) -> Tuple[str, dict[str, float], Callable[[np.ndarray], np.ndarray]] | None:
    print("")
    print("--- Квадратичная ---")

    try:
        solution = least_squares_method(x, y, 2)
    except np.linalg.LinAlgError:
        print("Система уравнений вырождена, решение не существует")
        return None
    if solution is None:
        print("ошибка в вычислении матрицы")
        return None

    def solution_to_string():
        a0, a1, a2 = solution
        return f"y = {a2:.6f}x² + {a1:.6f}x + {a0:.6f}"

    def func(x: np.ndarray):
        a0, a1, a2 = solution
        return a0 + a1 * x + a2 * x**2

    print(f"Формула: {solution_to_string()}")
    res = get_approximation_characteristics(x, y, func)

    if res is not None:
        return ("Квадратичная", res, func)
    else:
        return None
```