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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа №4

«АППРОКСИМАЦИЯ ФУНКЦИИ МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ»

по дисциплине «Вычислительная математика»

Вариант: **9**

Преподаватель:
Машина Екатерина Алексеевна

Выполнил:
Пронкин Алексей Дмитриевич
Группа: P3208

Санкт-Петербург, 2025 г.

Цель работы

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

$$f(x) = \frac{4x}{x^4 + 9}, \quad x \in [0, 2]\, h = 0.2$$

1. Табулирование функции

Точки: $x_i = 0,\ 0.2, \ldots, 2.0$ (всего 11).

x_i	$f(x_i)$
0.0	0.000000
0.2	0.088873
0.4	0.177274
0.6	0.262881
0.8	0.340078
1.0	0.400000
1.2	0.433463
1.4	0.436083
1.6	0.411480
1.8	0.369276
2.0	0.320000

Суммарные величины, необходимые для МНК (округлены до 10^{-6}):

$$n = 11, \qquad \sum x_i = 11.000000, \qquad \sum x_i^2 = 15.400000,$$
$$\sum x_i^3 = 24.200000, \qquad \sum x_i^4 = 40.532800,$$
$$\sum f_i = 3.239409, \qquad \sum x_i f_i = 4.012213, \qquad \sum x_i^2 f_i = 5.752960.$$

2. Линейное приближение

Модель $y_{\text{lin}} = a\,x + b$.

$$a = \frac{n \sum x_i f_i - \sum x_i \sum f_i}{n \sum x_i^2 - \left(\sum x_i\right)^2} = 0.175637, \quad b = \frac{\sum f_i - a \sum x_i}{n} = 0.118854.$$

$$\boxed{y_{\text{lin}}(x) = 0.175637\,x + 0.118854}$$

3. Квадратичное приближение

Модель $y_{\text{quad}} = a_2 x^2 + a_1 x + a_0$. Решая нормальную систему:

$$\begin{bmatrix} 11 & 11 & 15.4 \\ 11 & 15.4 & 24.2 \\ 15.4 & 24.2 & 40.5328 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 3.239409 \\ 4.012213 \\ 5.752960 \end{bmatrix},$$

получаем

$$a_0 = -0.024424, \qquad a_1 = 0.653231, \qquad a_2 = -0.238797.$$

$$\boxed{y_{\text{quad}}(x) = -0.238797\,x^2 + 0.653231\,x - 0.024424}$$

4. Погрешности аппроксимаций

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\bigl(f(x_i) - \hat{y}(x_i)\bigr)^2}.$$

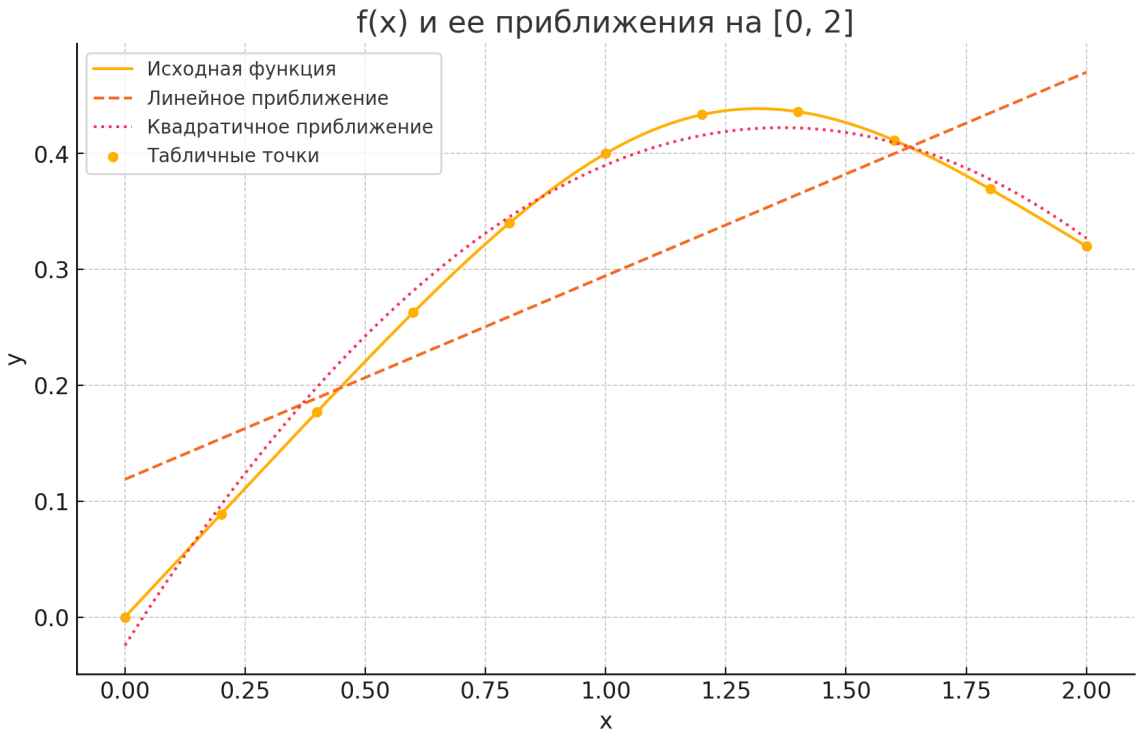
$$\sigma_{\text{lin}} = 0.086, \qquad \sigma_{\text{quad}} = 0.014.$$

Квадратичный полином аппроксимирует функцию заметно точнее.

5. Сводная таблица

x_i	$f(x_i)$	$y_{\text{lin}}(x_i)$	$y_{\text{quad}}(x_i)$
0.0	0.000000	0.118854	-0.024424
0.2	0.088873	0.153982	0.096671
0.4	0.177274	0.189109	0.198661
0.6	0.262881	0.224237	0.281548
0.8	0.340078	0.259364	0.345331
1.0	0.400000	0.294492	0.390011
1.2	0.433463	0.329619	0.415586
1.4	0.436083	0.364747	0.422058
1.6	0.411480	0.399874	0.409426
1.8	0.369276	0.435002	0.377690
2.0	0.320000	0.470129	0.326851

6. График



7. Листинг Python-скрипта

```python
#!/usr/bin/env python3
"""
Лабораторная: аппроксимация таблицы (x, y) методом наименьших квадратов.

Модели
------
1. Линейная            y = a + b·x
2. Квадратичная        y = a + b·x + c·x²
3. Кубическая          y = a + b·x + c·x² + d·x³
4. Экспоненциальная    y = a·e^{b·x}           (требует y>0)
5. Логарифмическая     y = a + b·ln x          (требует x>0)
6. Степенная           y = a·x^{b}             (требует x,y>0)

"""

from __future__ import annotations
import sys
import math
import argparse
from dataclasses import dataclass
from typing import Callable, Sequence

import numpy as np
import matplotlib.pyplot as plt


# ——————————————————————— вспомогательные структуры ——————————————————————— #

@dataclass
```

```python
class Fit:
    name: str
    f: Callable[[np.ndarray], np.ndarray]  # аппроксимирующая функция
    coeffs: Sequence[float]
    sse: float                             # сумма квадратов ошибок
    rms: float                             # среднеквадратичное отклонение
    r2: float                              # коэффициент детерминации


# ——————————————————————— построение полиномов ——————————————————————— #

def _poly_fit(x: np.ndarray, y: np.ndarray, deg: int) -> Fit:
    """Аппроксимация полиномом степени deg (0 ≤ deg ≤ 3) аналитически."""
    # матрица Вандермонда: [1, x, x², …]
    A = np.vander(x, N=deg + 1, increasing=True)  # shape (n, deg+1)
    ATA = A.T @ A
    ATy = A.T @ y
    coeffs = np.linalg.solve(ATA, ATy)  # a0, a1, …
    f = lambda t, c=coeffs: sum(c[i] * t ** i for i in range(len(c)))
    residuals = f(x) - y
    sse = float(np.sum(residuals ** 2))
    rms = float(np.sqrt(np.mean(residuals ** 2)))
    r2 = float(1 - sse / np.sum((y - y.mean()) ** 2))
    names = {1: "Линейная", 2: "Полиномиальная 2□й ст.", 3: "Полиномиальная 3□й ст."}
    return Fit(names[deg], f, coeffs, sse, rms, r2)


# ——————————————————————— нелинейные модели ——————————————————————— #

def _linear_ls(x: np.ndarray, y: np.ndarray) -> tuple[float, float]:
    """Простая линейная регрессия a + b·x через формулы МНК."""
    n = x.size
    sx, sy = x.sum(), y.sum()
    sxx = np.dot(x, x)
    sxy = np.dot(x, y)
    denom = n * sxx - sx * sx
    b = (n * sxy - sx * sy) / denom
    a = (sy - b * sx) / n
    return a, b


def _exp_fit(x: np.ndarray, y: np.ndarray) -> Fit:
    if np.any(y <= 0):
        raise ValueError("y ≤ 0 → экспоненциальная модель неприменима")
    y_ = np.log(y)
    a_, b_ = _linear_ls(x, y_)
    a = math.exp(a_)
    b = b_
    f = lambda t, A=a, B=b: A * np.exp(B * t)
    name = "Экспоненциальная"
    return _evaluate_model(name, f, (a, b), x, y)


def _log_fit(x: np.ndarray, y: np.ndarray) -> Fit:
    if np.any(x <= 0):
        raise ValueError("x ≤ 0 → логарифмическая модель неприменима")
    x_ = np.log(x)
    a, b = _linear_ls(x_, y)
    f = lambda t, A=a, B=b: A + B * np.log(t)
```

```
    return _evaluate_model("Логарифмическая", f, (a, b), x, y)


def _power_fit(x: np.ndarray, y: np.ndarray) -> Fit:
    if np.any((x <= 0) | (y <= 0)):
        raise ValueError("x ≤ 0 или y ≤ 0 → степенная модель неприменима")
    x_, y_ = np.log(x), np.log(y)
    a_, b_ = _linear_ls(x_, y_)
    a = math.exp(a_)
    b = b_
    f = lambda t, A=a, B=b: A * t ** B
    return _evaluate_model("Степенная", f, (a, b), x, y)


def _evaluate_model(name: str, f: Callable[[np.ndarray], np.ndarray],
                    coeffs: Sequence[float], x: np.ndarray, y: np.ndarray) -> Fit:
    residuals = f(x) - y
    sse = float(np.sum(residuals ** 2))
    rms = float(np.sqrt(np.mean(residuals ** 2)))
    r2 = float(1 - sse / np.sum((y - y.mean()) ** 2))
    return Fit(name, f, coeffs, sse, rms, r2)


# ──────────────────────────────── метрики ──────────────────────────────── #

def pearson_r(x: np.ndarray, y: np.ndarray) -> float:
    xm, ym = x.mean(), y.mean()
    num = np.sum((x - xm) * (y - ym))
    den = np.sqrt(np.sum((x - xm) ** 2) * np.sum((y - ym) ** 2))
    return float(num / den)


# ──────────────────────────────── ввод данных ──────────────────────────── #

def _read_points(path: str | None) -> tuple[np.ndarray, np.ndarray]:
    if path:
        src = open(path, encoding="utf-8")
    else:
        print("Введите пары x y по одной на строке (пустая строка — конец):")
        src = sys.stdin
    xs, ys = [], []
    for line in src:
        if not line.strip():
            break
        parts = line.replace(",", ".").split()
        if len(parts) != 2:
            print(f"! строка пропущена: {line.strip()}", file=sys.stderr)
            continue
        xs.append(float(parts[0]))
        ys.append(float(parts[1]))
    if path:
        src.close()
    if not (8 <= len(xs) <= 12):
        raise ValueError("Нужно от 8 до 12 точек")
    return np.array(xs, dtype=float), np.array(ys, dtype=float)


# ──────────────────────────── вывод табличных данных ──────────────────── #
```

```
148 def _table(x: np.ndarray, y: np.ndarray, fit: Fit):
149     print(f"\nТаблица для {fit.name}:")
150     print(f"{'i':>3} {'x':>10} {'y':>10} {'φ(x)':>10} {'ε':>10}")
151     for i, (xi, yi, fi) in enumerate(zip(x, y, fit.f(x)), 1):
152         print(f"{i:3d} {xi:10.4g} {yi:10.4g} {fi:10.4g} {fi - yi:10.4g}")
153
154
155 # ——————————————————————— визуализация ——————————————————————— #
156
157 def _plot_all(x: np.ndarray, y: np.ndarray, fits: list[Fit], best: Fit):
158     x_range = np.ptp(x)
159     x_dense = np.linspace(x.min() - 0.1 * x_range,
160                           x.max() + 0.1 * x_range, 500)
161     plt.figure(figsize=(10, 6))
162     plt.scatter(x, y, marker='o', label="Данные", zorder=5)
163     for fit in fits:
164         plt.plot(x_dense, fit.f(x_dense), label=fit.name)
165     plt.title(f"Аппроксимация (лучшая: {best.name})")
166     plt.xlabel("x")
167     plt.ylabel("y")
168     plt.legend()
169     plt.grid(True)
170     plt.tight_layout()
171     plt.show()
172
173
174 # ——————————————————————— основная программа ——————————————————————— #
175
176 def main():
177     p = argparse.ArgumentParser(description="Аппроксимация МНК без SciPy")
178     p.add_argument("-i", "--input", help="Файл с точками (x y)")
179     args = p.parse_args()
180
181     x, y = _read_points(args.input)
182
183     fits: list[Fit] = []
184     # полиномы
185     for deg in (1, 2, 3):
186         fits.append(_poly_fit(x, y, deg))
187     # нелинейные
188     for fn in (_exp_fit, _log_fit, _power_fit):
189         try:
190             fits.append(fn(x, y))
191         except ValueError as e:
192             print(f" {e}", file=sys.stderr)
193
194     # сортировка по RMS
195     fits.sort(key=lambda f: f.rms)
196     best = fits[0]
197
198     # вывод
199     print("===== РЕЗУЛЬТАТЫ =====")
200     r_lin = pearson_r(x, y)
201     print(f"Коэффициент Пирсона (линейная): r = {r_lin:.5f}")
202     for fit in fits:
203         coeffs = ", ".join(f"{c:.6g}" for c in fit.coeffs)
204         print(f"\n-- {fit.name} --")
205         print(f"Коэффициенты: {coeffs}")
206         print(f"SSE = {fit.sse:.6g}")
```

```
207             print(f"RMS = {fit.rms:.6g}")
208             print(f"R²  = {fit.r2:.6g}")
209             _table(x, y, fit)
210 
211     print(f"\nЛучшая модель: {best.name} (RMS = {best.rms:.6g})")
212 
213     _plot_all(x, y, fits, best)
214 
215 
216 if __name__ == "__main__":
217     try:
218         main()
219     except KeyboardInterrupt:
220         pass
```