Федеральное государственное автономное

образовательное учреждение высшего образования

Национальный Исследовательский Университет ИТМО

Лабораторная работа 4

«Аппроксимация функции методом наименьших квадратов»

Дисциплина: Вычислительная математика

Вариант 13

Выполнил: Терехин Никита Денисович

Факультет: Программной инженерии и компьютерной техники

Группа: P3208

Преподаватель: Машина Екатерина Алексеевна

г. Санкт-Петербург, 2024 год

Оглавление

Цель работы

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

Текст задания

Вычислительная реализация задачи
Вычислительная часть лабораторной работы должна быть представлена только в отчете.
Задание:

1. Сформировать таблицу табулирования заданной функции на указанном интервале (см. табл. 1)
2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала
3. Найти среднеквадратические отклонения для каждой аппроксимирующей функции. Ответы дать с тремя знаками после запятой
4. Выбрать наилучшее приближение
5. Построить графики заданной функции, а также полученные линейное и квадратичное приближения
6. Привести в отчете подробные вычисления

Программная реализация задачи
Для исследования использовать:

- линейную функцию
- полиномиальную функцию 2-й степени
- полиномиальную функцию 3-й степени
- экспоненциальную функцию
- логарифмическую функцию
- степенную функцию

Методика проведения исследования:

1. Вычислить меру отклонения: $S = \sum_{i=1}^{n} [\varphi(x_i) - y_i]^2$ для всех исследуемых функций
2. Уточнить значения коэффициентов эмпирических функций, минимизируя функцию S

3. Сформировать массивы предполагаемых эмпирических зависимостей $(\varphi(x_i),\ \varepsilon_i)$
4. Определить среднеквадратичное отклонение для каждой аппроксимирующей функции. Выбрать наименьшее значение и, следовательно, наилучшее приближение
5. Построить графики полученных эмпирических функций.

Задание:

1. Предусмотреть ввод исходных данных из файла/консоли (таблица $y = f(x)$ должна содержать от 8 до 12 точек);
2. Реализовать метод наименьших квадратов, исследуя все указанные функции
3. Предусмотреть вывод результатов в файл/консоль: коэффициенты аппроксимирующих функций, среднеквадратичное отклонение, массивы значений $x_i, y_i, \varphi(x_i), \varepsilon_i$
4. Для линейной зависимости вычислить коэффициент корреляции Пирсона
5. Вычислить коэффициент детерминации, программа должна выводить соответствующее сообщение в зависимости от полученного значения R^2
6. Программа должна отображать наилучшую аппроксимирующую функцию
7. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом)
8. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных;

Рабочие формулы методов

Линейная

$$S = \sum_{i=1}^{n} (ax_i + b - y_i)^2 \to min$$

$$\begin{cases} a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \\ a\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i \end{cases}$$

$$r = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{i=1}^{n}(y_i-\bar{y})^2}} \qquad \delta = \sqrt{\frac{S}{n}}$$

Квадратичная

$$S = \sum_{i=1}^{n}(ax_i + b - y_i)^2 \to min$$

$$\begin{cases} a_0 n + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} x_i y_i \\ a_0 \sum_{i=1}^{n} x_i^2 + a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} x_i^2 y_i \end{cases}$$

Вычислительная реализация

Исходные данные

№ варианта	Функция	Исследуемый интервал
13	$y = \frac{31x}{x^4+13}$	$x \in [0,4] \quad h = 0,4$

Таблица табулирования

x	0	0,4	0,8	1,2	1,6	2	2,4	2,8	3,2	3,6	4
$f(x)$	0	0,952	1,849	2,468	2,537	2,138	1,611	1,166	0,842	0,617	0,461

Линейное приближение

Вычисляем суммы

$$SX = \sum_{i=1}^{n} x_i = 22$$

$$SXX = \sum_{i=1}^{n} x_i^2 = 61,6$$

$$SY = \sum_{i=1}^{n} y_i = 14,64$$

$$SXY = \sum_{i=1}^{n} x_i y_i = 27,044$$

Получаем систему линейных уравнений:

$$\begin{cases} 61,6a + 22b = 27,044 \\ 22a + 11b = 14,64 \end{cases}$$

$$\Delta = SXX \cdot n - SX \cdot SX = 61,6 \cdot 11 - 22^2 = 193,6$$

$$\Delta_a = SXY \cdot n - SX \cdot SY = 27,044 \cdot 11 - 22 \cdot 14,64 = -24,596$$

$$\Delta_b = SXX \cdot SY - SX \cdot SXY = 61,6 \cdot 14,64 - 22 \cdot 27,044 = 306,856$$

$$a = \frac{\Delta_a}{\Delta} = \frac{-24,596}{193,6} \approx -0,127 \qquad b = \frac{\Delta_b}{\Delta} = \frac{306,856}{193,6} = 1,585$$

Аппроксимирующая линейная функция имеет вид

$$P(x) = 1,585 - 0,127x$$

x	0	0,4	0,8	1,2	1,6	2	2,4	2,8	3,2	3,6	4
$f(x)$	0	0,952	1,849	2,468	2,537	2,138	1,611	1,166	0,842	0,617	0,461
$P(x)$	1,585	1,534	1,483	1,433	1,382	1,331	1,28	1,229	1,178	1,128	1,077
ε	1,585	0,582	-0,366	-1,035	-1,155	-0,807	-0,331	0,063	0,336	0,511	0,616

$$S = \sum_{i=1}^{n} \varepsilon_i^2 = 6,908$$

$$\delta = \sqrt{\frac{S}{n}} = \sqrt{\frac{6,908}{11}} \approx 0,792$$

Квадратичное приближение

Вычисляем суммы

$$SX = \sum_{i=1}^{n} x_i = 22$$

$$SX^2 = \sum_{i=1}^{n} x_i^2 = 61,6$$

$$SX^3 = \sum_{i=1}^{n} x_i^3 = 193,6$$

$$SX^4 = \sum_{i=1}^{n} x_i^4 = 648,525$$

$$SY = \sum_{i=1}^{n} y_i = 14,64$$

$$SXY = \sum_{i=1}^{n} x_i y_i = 27,044$$

$$SX^2Y = \sum_{i=1}^{n} x_i^2 y_i = 62,352$$

Получаем систему линейных уравнений:

$$\begin{cases} 11a_0 + 22a_1 + 61,6a_2 = 14,64 \\ 22a_0 + 61,6a_1 + 193,6a_2 = 27,044 \\ 61,6a_0 + 193,6a_1 + 648,525a_2 = 62,352 \end{cases}$$

$$\Delta = 11 \cdot 61,6 \cdot 193,6 + 22 \cdot 193,6 \cdot 61,6 + 22 \cdot 193,6 \cdot 61,6 - 61,6^3 -$$

$$- 22^2 \cdot 648,525 - 193,6^2 \cdot 11 = 4252,424$$

$$\Delta_{a_0} = 14,64 \cdot 61,6 \cdot 193,6 + 27,044 \cdot 193,6 \cdot 61,6 + 22 \cdot 193,6 \cdot 62,352 -$$

$$- 61,6^2 \cdot 62,352 - 22 \cdot 648,525 \cdot 27,044 - 193,6^2 \cdot 14,64 = 1774,02$$

$$\Delta_{a_1} = 7736,536 \qquad \Delta_{a_2} = -2069,197$$

$$a_0 = \frac{\Delta_{a_0}}{\Delta} = \frac{1774,02}{4252,424} \approx 0,417 \qquad a_1 = \frac{\Delta_{a_1}}{\Delta} = \frac{7736,536}{4252,424} \approx 1,819$$

$$a_2 = \frac{\Delta_{a_2}}{\Delta} = \frac{-2069,197}{4252,424} \approx -0,487$$

Аппроксимирующая квадратичная функция имеет вид

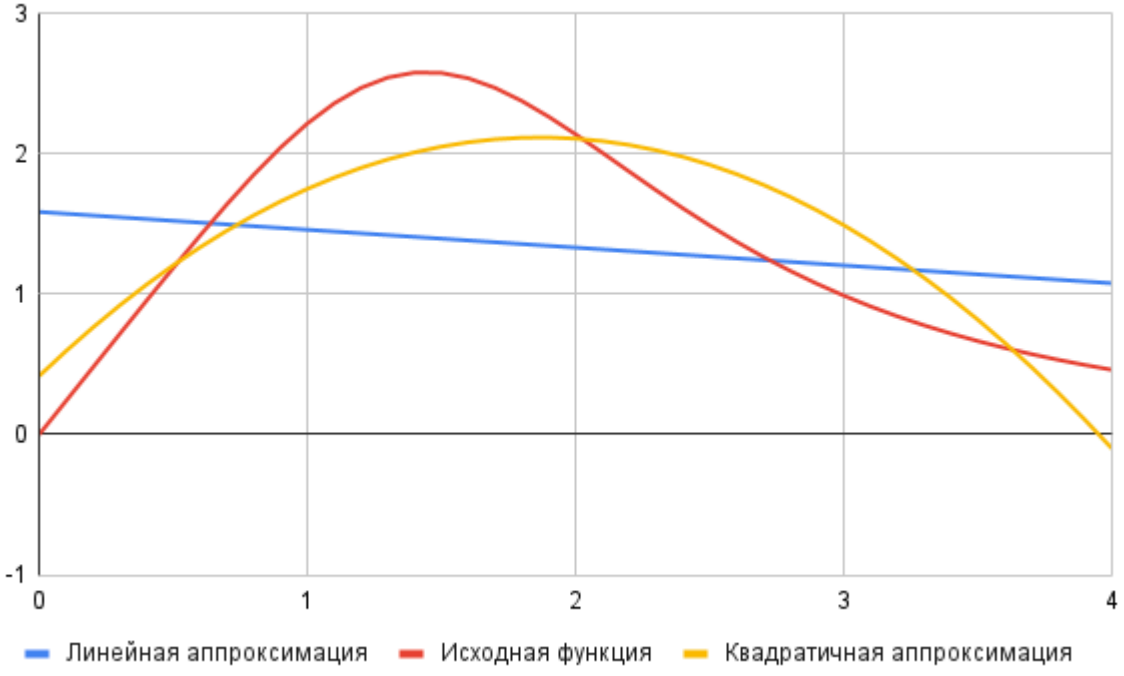
$$P(x) = -0,487x^2 + 1,819x + 0,417$$

x	0	0,4	0,8	1,2	1,6	2	2,4	2,8	3,2	3,6	4
$f(x)$	0	0,952	1,849	2,468	2,537	2,138	1,611	1,166	0,842	0,617	0,461
$P(x)$	0,417	1,067	1,561	1,899	2,081	2,107	1,977	1,692	1,251	0,654	-0,099
ε	0,417	0,115	-0,288	-0,569	-0,456	-0,031	0,366	0,526	0,409	0,037	-0,56

$$S = \sum_{i=1}^{n} \varepsilon_i^2 = 1,696$$

$$\delta = \sqrt{\frac{S}{n}} = \sqrt{\frac{1,696}{11}} \approx 0,393$$

Функции на графике



Программная реализация

Листинг программы

```
# approx.py

import math
from abc import abstractmethod
from typing import Final, Callable

from P3208.Terekhin_367558.lab2.functions import Describable


class Approximation(Describable):
    def __init__(self, description: str):
        super().__init__(description)
        self.func: Callable[[float], float] | None = None
        self.view: str = ''
```

```
    @abstractmethod
    def build_approximation(self, points: list[tuple[float,
float]]) -> None:
        pass

    def gauss_matrix_solve(self, matrix: list[list[float]]) ->
list[float]:
        n = len(matrix)
        for i in range(n):
            max_index = i
            max_value = abs(matrix[i][i])
            for k in range(i + 1, n):
                if abs(matrix[k][i]) > max_value:
                    max_index = k
                    max_value = abs(matrix[k][i])
            if max_index != i:
                matrix[i], matrix[max_index] = matrix[max_index],
matrix[i]

            pivot = matrix[i][i]
            if pivot == 0:
                raise ValueError("Matrix is singular")
            for j in range(i + 1, n):
                factor = matrix[j][i] / pivot
                for k in range(i, n + 1):
                    matrix[j][k] -= factor * matrix[i][k]

        solution: list[float] = [0] * n
        for i in range(n - 1, -1, -1):
            solution[i] = matrix[i][n] / matrix[i][i]
            for j in range(i - 1, -1, -1):
                matrix[j][n] -= matrix[j][i] * solution[i]
        return solution


class LinearApproximation(Approximation):
    def __init__(self):
        super().__init__("Linear Approximation")
        self.r = 0
        self.coefficients = []

    def build_approximation(self, points: list[tuple[float,
float]]) -> None:
        sum_x = sum(x for x, _ in points)
        sum_y = sum(y for _, y in points)
        sum_x_squared = sum(x ** 2 for x, _ in points)
        sum_xy = sum(x * y for x, y in points)
```

```
        n = len(points)

        average_x = sum_x / n
        average_y = sum_y / n

        a, b = map(lambda x: round(x, 3),
self.gauss_matrix_solve([[sum_x_squared, sum_x, sum_xy],
                                         [sum_x, n, sum_y]]))
        self.coefficients = [a, b]
        self.view = f'{a}x + {b}'

        self.r = round(
            sum((x - average_x) * (y - average_y) for x, y in
points) / (sum((x - average_x) ** 2 for x, y in points) * sum((y -
average_y) ** 2 for x, y in points)) ** 0.5,
            3)
        self.func = lambda x: a * x + b


class SquaredApproximation(Approximation):
    def __init__(self):
        super().__init__("Squared Approximation")

    def build_approximation(self, points: list[tuple[float,
float]]) -> None:
        sum_x = sum(x for x, _ in points)
        sum_y = sum(y for _, y in points)
        sum_x_squared = sum(x ** 2 for x, _ in points)
        sum_x_cubed = sum(x ** 3 for x, _ in points)
        sum_x_quad = sum(x ** 4 for x, _ in points)
        sum_xy = sum(x * y for x, y in points)
        sum_x_squared_y = sum(x ** 2 * y for x, y in points)

        n = len(points)
        c, b, a = map(lambda x: round(x, 3),
self.gauss_matrix_solve([[n, sum_x, sum_x_squared, sum_y],
                                             [sum_x, sum_x_squared,
sum_x_cubed, sum_xy],
                                             [sum_x_squared,
sum_x_cubed, sum_x_quad, sum_x_squared_y]]))
        self.view = f'{a}x^2 + {b}x + {c}'

        self.func = lambda x: a * x ** 2 + b * x + c


class CubedApproximation(Approximation):
    def __init__(self):
        super().__init__("Cubed Approximation")
```

```
    def build_approximation(self, points: list[tuple[float,
float]]) -> None:
        sum_x = sum(x for x, _ in points)
        sum_y = sum(y for _, y in points)
        sum_x_squared = sum(x ** 2 for x, _ in points)
        sum_x_cubed = sum(x ** 3 for x, _ in points)
        sum_x_quad = sum(x ** 4 for x, _ in points)
        sum_x_fifth = sum(x ** 5 for x, _ in points)
        sum_x_sixth = sum(x ** 6 for x, _ in points)
        sum_xy = sum(x * y for x, y in points)
        sum_x_squared_y = sum(y * x ** 2 for x, y in points)
        sum_x_cubed_y = sum(y * x ** 3 for x, y in points)

        n = len(points)

        d, c, b, a = map(lambda x: round(x, 3),
self.gauss_matrix_solve([[n, sum_x, sum_x_squared, sum_x_cubed,
sum_y],
                                                 [sum_x,
sum_x_squared, sum_x_cubed, sum_x_quad, sum_xy],
                                                 [sum_x_squared,
sum_x_cubed, sum_x_quad, sum_x_fifth, sum_x_squared_y],
                                                 [sum_x_cubed,
sum_x_quad, sum_x_fifth, sum_x_sixth, sum_x_cubed_y]]))

        self.view = f'{a}x^3 + {b}x^2 + {c}x + {d}'

        self.func = lambda x: a * x ** 3 + b * x ** 2 + c * x + d


class DegreeApproximation(Approximation):
    def __init__(self):
        super().__init__("Degree Approximation")
        self.linear = LinearApproximation()

    def build_approximation(self, points: list[tuple[float,
float]]) -> None:
        ln_points = list(map(lambda x: (math.log(x[0]),
math.log(x[1])), points))
        self.linear.build_approximation(ln_points)
        a, b = self.linear.coefficients
        self.view = f'{round(math.exp(b), 3)}x^{round(a, 3)}'
        self.func = lambda x: math.exp(b) * x ** a


class ExponentialApproximation(Approximation):
    def __init__(self):
```

```
        super().__init__("Exponential Approximation")
        self.linear = LinearApproximation()

    def build_approximation(self, points: list[tuple[float,
float]]) -> None:
        ln_points = list(map(lambda x: (x[0], math.log(x[1])),
points))
        self.linear.build_approximation(ln_points)
        a, b = self.linear.coefficients
        self.view = f'{round(math.exp(b), 3)}e^{a}x'
        self.func = lambda x: math.exp(b) * math.exp(a * x)


class LogarithmicApproximation(Approximation):
    def __init__(self):
        super().__init__("Logarithmic Approximation")
        self.linear = LinearApproximation()

    def build_approximation(self, points: list[tuple[float,
float]]) -> None:
        ln_points = list(map(lambda x: (math.log(x[0]), x[1]),
points))
        self.linear.build_approximation(ln_points)
        a, b = self.linear.coefficients
        self.view = f'{a} ln(x) + {b}'
        self.func = lambda x: a * (math.nan if x <= 0 else
math.log(x)) + b


APPROXIMATIONS: Final[list[Approximation]] = [
    LinearApproximation(),
    SquaredApproximation(),
    CubedApproximation(),
    DegreeApproximation(),
    ExponentialApproximation(),
    LogarithmicApproximation()
]
```

```
# main.py

import math

from matplotlib import pyplot as plt
from matplotlib.axes import Axes
from tabulate import tabulate

from P3208.Terekhin_367558.lab2.main import request_from_list
```

```
from P3208.Terekhin_367558.lab2.readers import AbstractReader,
READERS
from P3208.Terekhin_367558.lab4.approx import APPROXIMATIONS,
Approximation, LinearApproximation


if __name__ == '__main__':
    reader: AbstractReader = request_from_list(READERS)
    points: list[tuple[float, float]] = reader.read_points()
    x_values: list[float] = [x for x, y in points]
    y_values: list[float] = [y for x, y in points]
    x_range = [i / 100 for i in range(math.floor(min(x_values)),
100 * math.ceil(max(x_values)))]

    ax: Axes = plt.axes()
    ax.spines['left'].set_position('zero')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')

    best: float = math.inf
    ind: int = 0
    for i in range(len(APPROXIMATIONS)):
        method: Approximation = APPROXIMATIONS[i]
        method.build_approximation(points)
        print(tabulate([method.description]))
        results: list[list[float]] = []
        deviation: float = 0
        y_approx: list[float] = []
        if method.func is not None:
            y_range = [method.func(x) for x in x_range]
            plt.plot(x_range, y_range)
            for x, y in points:
                approx = round(method.func(x), 3)
                y_approx.append(approx)
                deviation += (y - approx) ** 2
                results.append([x, y, approx, y - approx])
        headers = ['Average deviation', 'R^2', 'Formula view']
        average_approx = sum(y_approx) / len(y_approx)
        average_deviation = round((deviation / len(points)) ** 0.5,
3)
        if average_deviation < best:
            best = average_deviation
            ind = i
        R = round(1 - sum([(y_values[i] - y_approx[i]) ** 2 for i
in range(len(y_values))]) / sum([(y - average_approx) ** 2 for y
in y_values]), 3)
        stat = [[average_deviation, R,  method.view]]
```

```
        if isinstance(method, LinearApproximation):
            headers.append('Pearson coefficient')
            stat[0].append(method.r)

        print(tabulate(stat, headers, numalign="center"), '\n')

        plt.plot(x_values, y_values, 'ro')

        print(tabulate(results, headers=['X', 'Y', 'Approximation',
'Epsilon'], numalign="center"))

    print(tabulate(["Best approximation"]))
    print(APPROXIMATIONS[ind].description)
    plt.show()
```

Выводы

В ходе выполнения работы удалось найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов

В конечном итоге использование того или иного вида аппроксимации зависит от исходных данных. На основе нескольких гипотез можно получить наиболее точное приближение для конкретных данных