МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

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 № 13

***Выполнил:***

Студент группы P3218

Рамеев Тимур

Ильгизович

***Преподаватель:***

Бострикова Дарья

Константиновна

Оглавление

[Цель работы 3](#_Toc167723958)

[Используемые методы 3](#_Toc167723959)

[Метод наименьших квадратов 3](#_Toc167723960)

[Аппроксимация многочленом 3](#_Toc167723961)

[Степенная аппроксимация 4](#_Toc167723962)

[Экспоненциальная аппроксимация 4](#_Toc167723963)

[Логарифмическая аппроксимация 4](#_Toc167723964)

[Выбор аппроксимирующей функции 4](#_Toc167723965)

[Вычислительная часть 5](#_Toc167723966)

[Реализация и пример работы 5](#_Toc167723967)

[Вывод 6](#_Toc167723968)

# Цель работы

Нахождение функции, являющейся наилучшим приближением заданной табличной функции по методу наименьших квадратов

# Используемые методы

## Метод наименьших квадратов

В общем случае методы аппроксимации основаны на подборе общего вида аппроксимирующей функции и определении значений ее параметров.

В методе МНК значения этих параметров определяются с помощью критерия минимизации (S):

То есть значения параметров находятся посредством минимизации функции .

## Аппроксимация многочленом

Наиболее простым видом аппроксимирующей функции является многочлен степени m:

Коэффициенты находятся из условия стационарности точки (именно в стационарной точке у функции S – минимум):

## Степенная аппроксимация

Рассмотрим аппроксимирующую функцию вида:

Задачу нахождения значений параметров a и b можно свести к задаче нахождению коэффициентов многочлена, логарифмирую обе стороны равенства:

При этом вместо необходимо использовать , а вместо -. Коэффициент восстанавливаем потенцированием.

## Экспоненциальная аппроксимация

Аналогично можно найти параметры экспоненциальной функции:

## Логарифмическая аппроксимация

При логарифмической аппроксимации и логарифмировать обе части не требуется:

Нужно лишь не забыть вместо использовать .

## Выбор аппроксимирующей функции

В качестве показателей того, насколько хорошо найденное приближение аппроксимирует функцию можно использовать: коэффициент корреляции (для линейной связи), достоверность аппроксимации или, если нужно сравнить несколько аппроксимаций – среднеквадратичное отклонение.

Среднеквадратичное отклонение:

# Вычислительная часть

Аппроксимируем следующую функцию:

Сформируем таблицу табулирования заданной функции на интервале [0; 4] с шагом h = 0.4:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | 0 | 0.4 | 0.8 | 1.2 | 1.6 | 2 | 2.4 | 2.8 | 3.2 | 3.6 | 4 |
|  | 0 | 0.952 | 1.849 | 2.468 | 2.537 | 2.138 | 1.611 | 1.166 | 0.842 | 0.617 | 0.461 |

Построим линейную аппроксимацию:

Для этого решим следующее матричное уравнение:

Решая методом Гаусса, получаем:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | 0 | 0.4 | 0.8 | 1.2 | 1.6 | 2 | 2.4 | 2.8 | 3.2 | 3.6 | 4.0 |
|  | 1.585 | 1.534 | 1.483 | 1.433 | 1.382 | 1.331 | 1.280 | 1.229 | 1.179 | 1.128 | 1.077 |
|  | 0 | 0.952 | 1.849 | 2.468 | 2.537 | 2.138 | 1.611 | 1.166 | 0.842 | 0.617 | 0.461 |
|  | 1.585 | 0.582 | -0.366 | -1.035 | -1.155 | -0.807 | -0.331 | 0.064 | 0.337 | 0.511 | 0.616 |
|  | 2.512 | 0.339 | 0.134 | 1.072 | 1.334 | 0.651 | 0.110 | 0.004 | 0.114 | 0.261 | 0.379 |

Найдем среднеквадратичное отклонение:

Аналогичным образом найдем квадратичное приближение:

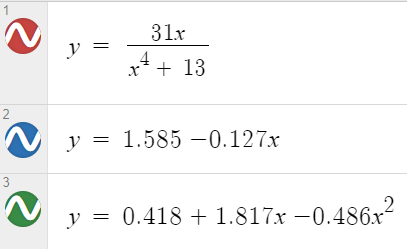
Для этого решим следующее матричное уравнение:

Решая методом Гаусса, получаем:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | 0 | 0.4 | 0.8 | 1.2 | 1.6 | 2 | 2.4 | 2.8 | 3.2 | 3.6 | 4.0 |
|  | 0.418 | 1.067 | 1.561 | 1.899 | 2.081 | 2.108 | 1.979 | 1.695 | 1.256 | 0.661 | -0.090 |
|  | 0 | 0.952 | 1.849 | 2.468 | 2.537 | 2.138 | 1.611 | 1.166 | 0.842 | 0.617 | 0.461 |
|  | 0.418 | 0.115 | -0.289 | -0.569 | -0.456 | -0.030 | 0.368 | 0.530 | 0.414 | 0.044 | -0.551 |
|  | 0.174 | 0.013 | 0.083 | 0.324 | 0.208 | 0.001 | 0.136 | 0.281 | 0.171 | 0.002 | 0.304 |

Найдем среднеквадратичное отклонение:

У квадратичной аппроксимации среднеквадратичное отклонение меньше, чем у линейной, а значит оно лучше.

Изображение выглядит как линия, График, диаграмма, Параллельный

Автоматически созданное описание

# Реализация и пример работы

from math import sqrt, exp, log

import numpy as np

*def* find\_approximations(*x\_values*, *y\_values*):

    local\_x\_values = []

    local\_y\_values = []

    for i in range(len(x\_values)):

        if x\_values[i] != "" and y\_values[i] != "":

            local\_x\_values.append(*float*(x\_values[i].replace(',', '.')))

            local\_y\_values.append(*float*(y\_values[i].replace(',', '.')))

    return { "status" : 1, "x\_values" : [round(x, 3) for x in local\_x\_values], "y\_values" : [round(y, 3) for y in local\_y\_values],

            "linear\_result" : find\_linear\_function(local\_x\_values, local\_y\_values),

            "quadratic\_result" : find\_quadratic\_function(local\_x\_values, local\_y\_values),

            "cubic\_result" : find\_cubic\_function(local\_x\_values, local\_y\_values),

            "degree\_result" : find\_degree\_function(local\_x\_values, local\_y\_values),

            "exp\_result" : find\_exp\_function(local\_x\_values, local\_y\_values),

            "log\_result" : find\_log\_function(local\_x\_values, local\_y\_values)

            }

*def* find\_polinomial\_function(*x\_values*, *y\_values*, *n*):

    current\_degree = n

    left\_matrix = []

    right\_matrix = []

    for i in range(n + 1):

        current\_degree -= n

        current\_column = []

        for j in range(n + 1):

            current\_sum = 0

            for x in x\_values:

                current\_sum += x\*\*current\_degree

            current\_degree += 1

            current\_column.append(current\_sum)

        left\_matrix.append(current\_column)

    current\_degree = 0

    for i in range(n + 1):

        current\_sum = 0

        for i in range(len(y\_values)):

            current\_sum += x\_values[i]\*\*current\_degree \* y\_values[i]

        current\_degree += 1

        right\_matrix.append(current\_sum)

    main\_det = np.linalg.det(np.array(left\_matrix))

    ratio\_arr = []

    for i in range(n + 1):

        curr\_det = left\_matrix.copy()

        curr\_det[i] = right\_matrix

        curr\_high\_det = np.linalg.det(np.array(curr\_det))

        ratio\_arr.append(round(curr\_high\_det / main\_det, 3))

    ret\_arr = []

    for x in x\_values:

        current\_sum = 0

        current\_degree = -1

        for i in range(n + 1):

            current\_degree += 1

            current\_sum += x\*\*current\_degree \* ratio\_arr[i]

        ret\_arr.append(round(current\_sum, 3))

    return [ret\_arr, ratio\_arr]

*def* find\_linear\_function(*x\_values*, *y\_values*):

    result\_arr = find\_polinomial\_function(x\_values, y\_values, 1)

    linear\_values = result\_arr[0]

    reliability = round(find\_reliability(y\_values, linear\_values), 3)

    return {"a" : result\_arr[1][1], "b" : result\_arr[1][0], "values" : linear\_values, "differences" : find\_differences(y\_values, linear\_values),

            "deviation" : find\_diviation(y\_values, linear\_values), "pirson" : round(find\_pirson(x\_values, y\_values), 3),

            "reliability" : reliability, "status" : define\_status(reliability) }

*def* find\_quadratic\_function(*x\_values*, *y\_values*):

    result\_arr = find\_polinomial\_function(x\_values, y\_values, 2)

    quadratic\_values = result\_arr[0]

    reliability = round(find\_reliability(y\_values, quadratic\_values), 3)

    return {"a" : result\_arr[1][2], "b" : result\_arr[1][1], "c" : result\_arr[1][0], "values" : quadratic\_values, "differences" : find\_differences(y\_values, quadratic\_values),

            "deviation" : find\_diviation(y\_values, quadratic\_values),

            "reliability" : reliability, "status" : define\_status(reliability) }

*def* find\_cubic\_function(*x\_values*, *y\_values*):

    result\_arr = find\_polinomial\_function(x\_values, y\_values, 3)

    cubic\_values = result\_arr[0]

    reliability = round(find\_reliability(y\_values, cubic\_values), 3)

    return {"a" : result\_arr[1][3], "b" : result\_arr[1][2], "c" : result\_arr[1][1], "d" : result\_arr[1][0], "values" : cubic\_values, "differences" : find\_differences(y\_values, cubic\_values),

            "deviation" : find\_diviation(y\_values, cubic\_values),

            "reliability" : reliability, "status" : define\_status(reliability) }

*def* find\_degree\_function(*x\_values*, *y\_values*):

    try:

        ln\_x\_values = [log(x) for x in x\_values]

        ln\_y\_values = [log(y) for y in y\_values]

        result\_arr = find\_polinomial\_function(ln\_x\_values, ln\_y\_values, 1)

        a = exp(result\_arr[1][0])

        b = result\_arr[1][1]

        degree\_values = []

        for x in x\_values:

            degree\_values.append(round(a \* x\*\*b, 3))

        reliability = round(find\_reliability(y\_values, degree\_values), 3)

        return {"a" : round(a, 3), "b" : round(b, 3), "values" : degree\_values, "differences" : find\_differences(y\_values, degree\_values),

            "deviation" : find\_diviation(y\_values, degree\_values),

            "reliability" : reliability, "status" : define\_status(reliability) }

    except *Exception*:

        return {"error" : "Для аппроксимации спомощью степенной функции используется логарифмы по x и y, поэтому координаты точек должны быть положительными" }

*def* find\_exp\_function(*x\_values*, *y\_values*):

    try:

        ln\_y\_values = [log(y) for y in y\_values]

        result\_arr = find\_polinomial\_function(x\_values, ln\_y\_values, 1)

        a = exp(result\_arr[1][0])

        b = result\_arr[1][1]

        degree\_values = []

        for x in x\_values:

            degree\_values.append(round(a \* x\*\*b, 3))

        reliability = round(find\_reliability(y\_values, degree\_values), 3)

        return {"a" : round(a, 3), "b" : round(b, 3), "values" : degree\_values, "differences" : find\_differences(y\_values, degree\_values),

            "deviation" : find\_diviation(y\_values, degree\_values),

            "reliability" : reliability, "status" : define\_status(reliability) }

    except *Exception*:

        return {"error" : "Для аппроксимации спомощью экспоненциальной функции используется логарифм по y, поэтому координатa точек y должна быть положительной" }

*def* find\_log\_function(*x\_values*, *y\_values*):

    try:

        ln\_x\_values = [log(x) for x in x\_values]

        result\_arr = find\_polinomial\_function(ln\_x\_values, y\_values, 1)

        a = result\_arr[1][1]

        b = result\_arr[1][0]

        log\_values = []

        for x in x\_values:

            log\_values.append(round(a \* log(x) + b, 3))

        reliability = round(find\_reliability(y\_values, log\_values), 3)

        return {"a" : round(a, 3), "b" : round(b, 3), "values" : log\_values, "differences" : find\_differences(y\_values, log\_values),

            "deviation" : find\_diviation(y\_values, log\_values),

            "reliability" : reliability, "status" : define\_status(reliability) }

    except *Exception*:

        return {"error" : "Для аппроксимации спомощью логарифмической функции используется логарифм по x, поэтому координатa точек x должна быть положительной" }

*def* find\_differences(*y\_values*, *phi\_values*):

    differences = []

    for i in range(len(y\_values)):

        differences.append(round(phi\_values[i] - y\_values[i], 3))

    return differences

*def* find\_diviation(*y\_values*, *phi\_values*):

    deviation = 0

    for i in range(len(y\_values)):

        deviation += (phi\_values[i] - y\_values[i])\*\*2

    deviation /= len(y\_values)

    deviation = round(sqrt(deviation), 3)

    return deviation

*def* find\_pirson(*x\_values*, *y\_values*):

    n = len(x\_values)

    x\_average = 0

    y\_average = 0

    for i in range(n):

        x\_average += x\_values[i]

        y\_average += y\_values[i]

    y\_average /= n

    x\_average /= n

    pirson\_top = 0

    bottom\_1 = 0

    bottom\_2 = 0

    for i in range(n):

        pirson\_top += ((x\_values[i] - x\_average) \* (y\_values[i] - y\_average))

        bottom\_1 += (x\_values[i] - x\_average)\*\*2

        bottom\_2 += (y\_values[i] - y\_average)\*\*2

    pirson\_bottom = sqrt(bottom\_1 \* bottom\_2)

    try:

        return pirson\_top / pirson\_bottom

    except *Exception*:

        return 1

*def* find\_reliability(*y\_values*, *phi\_values*):

    n = len(y\_values)

    reliability\_top = 0

    reliability\_bottom = 0

    average\_phi = 0

    for i in range(n):

        average\_phi += phi\_values[i]

    average\_phi /= n

    for i in range(n):

        reliability\_top += (y\_values[i] - phi\_values[i])\*\*2

        reliability\_bottom += (y\_values[i] - average\_phi)\*\*2

    if round(reliability\_top, 3) == 0:

        return 1

    return 1 - reliability\_top / reliability\_bottom

*def* define\_status(*reliability*):

    if (reliability >= 0.95):

        return "Высокая точность аппрокимации"

    elif (reliability >= 0.75):

        return "Средняя точность аппроксимации"

    elif (reliability >= 0.5):

        return "Низкая точность аппроксимации"

    else:

        return "Неудовлетворительная точность аппроксимации"

# Вывод

Мы изучили один из методов аппроксимации функций – метод наименьших квадратов, с его помощью аппроксимировали заданную функцию по нескольким точкам, а также привели программную реализацию приближения многочленами, степенной, экспоненциальной и логарифмической функцией.