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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа по вычислительной математике №4

Аппроксимация функции методом наименьших квадратов

Преподаватель: Малышева Татьяна Алексеевна

Выполнила: Голованова Дарья Владимировна

Группа: P3222

Санкт-Петербург,
2022г

Цель лабораторной работы:

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

Описание использованного метода:

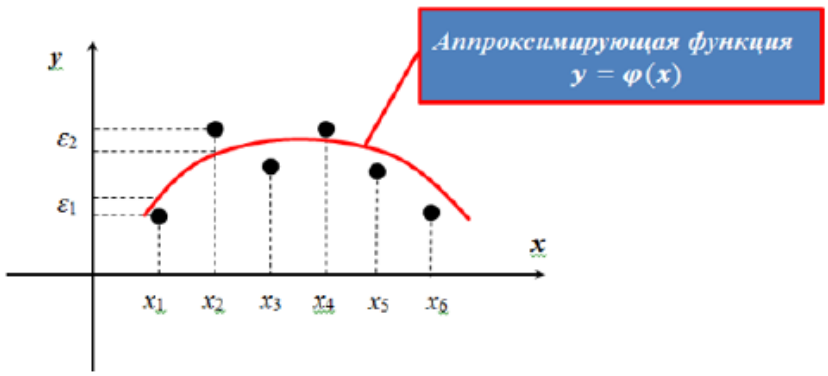
Представим, что вид аппроксимирующей функции уже известен:

$y=\varphi(x, a0, a1,a2,\ldots., am)$,

где φ – известная функция, $a0, a1,a2,\ldots., am$ – неизвестные параметры.

Требуется определить такие параметры, при которых значения аппроксимирующей функции приблизительно совпадали со значениями исследуемой функции в точках xi, т.е. $yi\approx\varphi(xi)$. Разность между этими значениями (отклонения) обозначим через εi.

Тогда $\varepsilon i=\varphi(x, a0, a1,a2,\ldots., am)-yi$, i=1,2, …n



Мерой отклонения многочлена $\varphi(x)$ от заданной функции *f(x)* на множестве точек ((xi, yi) является величина S (критерий минимизации), равная сумме квадратов разности между значениями многочлена и функции для всех точек $x0, x1,\ldots., xn$:

$$S=\sum_{i=1}^{n}\varepsilon_i^2=\sum_{i=1}^{n}[\varphi(x_i)-y_i]^2\rightarrow min$$

Задача нахождения наилучших значений параметров $a0, a1,\ldots., am$ сводятся к некоторой минимизации отклонений εi.

Параметры $a0, a1,\ldots., am$ эмпирической формулы находятся из условия минимума функции $S=S(a0, a1,a2,\ldots., am)$,.

Так как здесь параметры выступают в роли независимых переменных функции S, то её минимум найдем, приравнивая к нулю частные производные по этим переменным (m – степень многочлена, n - число точек):

$$\frac{\partial S}{\partial a_0} = 2\sum_{i=1}^{n} a_0 + |a_1 x_i + \cdots + a_{m-1} x_i^{m-1} + a_m x_i^m - y_i = 0$$

$$\frac{\partial S}{\partial a_1} = 2\sum_{i=1}^{n} (a_0 + a_1 x_i + \cdots + a_{m-1} x_i^{m-1} + a_m x_i^m - y_i) x_i = 0$$

$$\cdots \cdots \cdots$$

$$\frac{\partial S}{\partial a_m} = 2\sum_{i=1}^{n} (a_0 + a_1 x_i + \cdots + a_{m-1} x_i^{m-1} + a_m x_i^m - y_i) x_i^m = 0$$

Преобразуем полученную линейную систему уравнений: раскроем скобки и перенесем свободные слагаемые в правую часть выражения.

в матричном виде:

$$\begin{vmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^m \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{m+1} \\ & \cdots \cdots \cdots & & \\ \sum_{i=1}^{n} x_i^m & \sum_{i=1}^{n} x_i^{m+1} & \cdots & \sum_{i=1}^{n} x_i^{2m} \end{vmatrix} \cdot \begin{vmatrix} a_0 \\ a_1 \\ \cdots \\ \cdots \\ \cdots \\ a_m \end{vmatrix} = \begin{vmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \cdots \\ \sum_{i=1}^{n} x_i^m y_i \end{vmatrix}$$

Листинг численного метода:

1) Линейная аппроксимация:

```
class LinearApproximation(Approximation):
    function_type = "Линейная зависимость"

    def find_an_approximation(self, list_of_x: list, list_of_y: list) -
> Function:
        SX = sum(list_of_x)
        SXX = sum(x * x for x in list_of_x)
        SY = sum(list_of_y)
        SXY = sum(list_of_x[i] * list_of_y[i] for i in
range(len(list_of_x)))
        n = len(list_of_x)

        answer = self.solve_matrix([[SXX, SX], [SX, n]], [SXY, SY])
        a = answer[0]
        b = answer[1]
        if a is None or b is None or self.isNaN(a) or self.isNaN(b):
            return None
        fun = lambda x: a * x + b
        s = sum((fun(list_of_x[i]) - list_of_y[i]) ** 2 for i in
range(len(list_of_x)))
        root_mean_square_deviation = sqrt(s / n)
        f = Function(fun, f'ф = {round(a, 3)}*x {round(b, 3):+}', s,
root_mean_square_deviation)
        self.print_approximation_table(list_of_x, list_of_y, f,
self.function_type)

        average_x = SX / n
        average_y = SY / n
        r = (sum((list_of_x[i] - average_x) * (list_of_y[i] -
average_y) for i in range(len(list_of_x)))
             / sqrt(sum((x - average_x) ** 2 for x in list_of_x) *
                    sum((y - average_y) ** 2 for y in list_of_y)))
        logging.info(f'Коэффициент корреляции Пирсона равен {round(r,
3)}')
        return f
```

2) Квадратичная аппроксимация:

```
class SquareApproximation(Approximation):
    function_type = "Квадратичная зависимость"

    def find_an_approximation(self, list_of_x: list, list_of_y: list) -
> Function:
        SX = sum(list_of_x)
        SXX = sum(x * x for x in list_of_x)
        SXXX = sum(x * x * x for x in list_of_x)
        SXXXX = sum(x * x * x * x for x in list_of_x)
        SY = sum(list_of_y)
        SXY = sum(list_of_x[i] * list_of_y[i] for i in
range(len(list_of_x)))
        SXXY = sum(list_of_x[i] * list_of_x[i] * list_of_y[i] for i in
range(len(list_of_x)))
        n = len(list_of_x)

        answer = self.solve_matrix([[SXXXX, SXXX, SXX], [SXXX, SXX,
SX], [SXX, SX, n]], [SXXY, SXY, SY])
        a = answer[0]
        b = answer[1]
        c = answer[2]
```

```
        if a is None or b is None or c is None or self.isNaN(a) or
self.isNaN(b) or self.isNaN(c):
            return None
        fun = lambda x: a * x * x + b * x + c
        s = sum((fun(list_of_x[i]) - list_of_y[i]) ** 2 for i in
range(len(list_of_x)))
        root_mean_square_deviation = sqrt(s / n)
        f = Function(fun, f'ϕ = {round(a, 3):+}*x^2 {round(b, 3):+}*x
{round(c, 3):+}',
                     s, root_mean_square_deviation)
        self.print_approximation_table(list_of_x, list_of_y, f,
self.function_type)
        return f
```

3) Кубическая аппроксимация:

```
class ThirdApproximation(Approximation):
    function_type = "Кубическая зависимость"

    def find_an_approximation(self, list_of_x: list, list_of_y: list) -
> Function:
        SX = sum(list_of_x)
        SXX = sum(x * x for x in list_of_x)
        SXXX = sum(x * x * x for x in list_of_x)
        SXXXX = sum(x * x * x * x for x in list_of_x)
        SXXXXX = sum(x * x * x * x * x for x in list_of_x)
        SXXXXXX = sum(x * x * x * x * x * x for x in list_of_x)
        SY = sum(list_of_y)
        SXY = sum(list_of_x[i] * list_of_y[i] for i in
range(len(list_of_x)))
        SXXY = sum(list_of_x[i] * list_of_x[i] * list_of_y[i] for i in
range(len(list_of_x)))
        SXXXY = sum(list_of_x[i] * list_of_x[i] * list_of_x[i] *
list_of_y[i] for i in range(len(list_of_x)))
        n = len(list_of_x)

        answer = self.solve_matrix(
            [[SXXXXXX, SXXXXX, SXXXX, SXXX], [SXXXXX, SXXXX, SXXX,
SXX], [SXXXX, SXXX, SXX, SX], [SXXX, SXX, SX, n]],
            [SXXXY, SXXY, SXY, SY])
        a = answer[0]
        b = answer[1]
        c = answer[2]
        d = answer[3]
        if a is None or b is None or c is None or d is None or
self.isNaN(a) or self.isNaN(b) or self.isNaN(
                c) or self.isNaN(d):
            return None
        fun = lambda x: a * x * x * x + b * x * x + c * x + d
        s = sum((fun(list_of_x[i]) - list_of_y[i]) ** 2 for i in
range(len(list_of_x)))
        root_mean_square_deviation = sqrt(s / n)
        f = Function(fun, f'ϕ = {round(a, 3):+}*x^3 {round(b, 3):+}*x^2
{round(c, 3):+}*x {round(d, 3):+}',
                     s, root_mean_square_deviation)
        self.print_approximation_table(list_of_x, list_of_y, f,
self.function_type)
        return f
```

```
SXXX = sum(x * x * x for x in list_of_x)
SXXXX = sum(x * x * x * x for x in list_of_x)
SXXXXX = sum(x * x * x * x * x for x in list_of_x)
SXXXXXX = sum(x * x * x * x * x * x for x in list_of_x)
SY = sum(list_of_y)
SXY = sum(list_of_x[i] * list_of_y[i] for i in range(len(list_of_x)))
SXXY = sum(list_of_x[i] * list_of_x[i] * list_of_y[i] for i in range(len(list_of_x)))
SXXXY = sum(list_of_x[i] * list_of_x[i] * list_of_x[i] * list_of_y[i] for i in range(len(list_of_x)))
n = len(list_of_x)

answer = self.solve_matrix(
    [[SXXXXXX, SXXXXX, SXXXX, SXXX], [SXXXXX, SXXXX, SXXX, SXX], [SXXXX, SXXX, SXX, SX], [SXXX, SXX, SX,
    [SXXXY, SXXY, SXY, SY])
a = answer[0]
b = answer[1]
c = answer[2]
d = answer[3]
if a is None or b is None or c is None or d is None or self.isNaN(a) or self.isNaN(b) or self.isNaN(
        c) or self.isNaN(d):
    return None
fun = lambda x: a * x * x * x + b * x * x + c * x + d
s = sum((fun(list_of_x[i]) - list_of_y[i]) ** 2 for i in range(len(list_of_x)))
root_mean_square_deviation = sqrt(s / n)
f = Function(fun, f'ϕ = {round(a, 3):+}*x^3 {round(b, 3):+}*x^2 {round(c, 3):+}*x {round(d, 3):+}',
```

4) Степенная аппроксимация:

```
class PowerApproximation(Approximation):
    function_type = "Степенная зависимость"

    def find_an_approximation(self, list_of_x: list, list_of_y: list) -
> Function:
        try:
            SLNX = sum(log(x) for x in list_of_x)
            SLNXX = sum(log(x) * log(x) for x in list_of_x)
            SLNY = sum(log(y) for y in list_of_y)
            SLNXY = sum(log(list_of_x[i]) * log(list_of_y[i]) for i in
range(len(list_of_x)))
            n = len(list_of_x)
        except ValueError:
            return None

        try:
            answer = self.solve_matrix([[SLNXX, SLNX], [SLNX, n]],
[SLNXY, SLNY])
            b = answer[0]
            a = answer[1]
            if a is None or b is None or self.isNaN(a) or
self.isNaN(b):
                return None
            a = exp(a)
            fun = lambda x: a * (x ** b)
            s = sum((fun(list_of_x[i]) - list_of_y[i]) ** 2 for i in
range(len(list_of_x)))

            root_mean_square_deviation = sqrt(s / n)
            f = Function(fun, f'ϕ = {round(a, 3)}*x^({round(b, 3)})',
s, root_mean_square_deviation)
            self.print_approximation_table(list_of_x, list_of_y, f,
self.function_type)
            return f
        except TypeError:
            return None
```

5) Экспоненциальная аппроксимация:

```
class PowerApproximation(Approximation):
    function_type = "Степенная зависимость"

    def find_an_approximation(self, list_of_x: list, list_of_y: list) -
> Function:
        try:
            SLNX = sum(log(x) for x in list_of_x)
            SLNXX = sum(log(x) * log(x) for x in list_of_x)
            SLNY = sum(log(y) for y in list_of_y)
            SLNXY = sum(log(list_of_x[i]) * log(list_of_y[i]) for i in
range(len(list_of_x)))
            n = len(list_of_x)
        except ValueError:
            return None

        try:
            answer = self.solve_matrix([[SLNXX, SLNX], [SLNX, n]],
[SLNXY, SLNY])
            b = answer[0]
            a = answer[1]
            if a is None or b is None or self.isNaN(a) or
self.isNaN(b):
                return None
            a = exp(a)
            fun = lambda x: a * (x ** b)
            s = sum((fun(list_of_x[i]) - list_of_y[i]) ** 2 for i in
range(len(list_of_x)))

            root_mean_square_deviation = sqrt(s / n)
            f = Function(fun, f'ф = {round(a, 3)}*x^({round(b, 3)})',
s, root_mean_square_deviation)
            self.print_approximation_table(list_of_x, list_of_y, f,
self.function_type)
            return f
        except TypeError:
            return None
```

6) Логарифмическая аппроксимация:

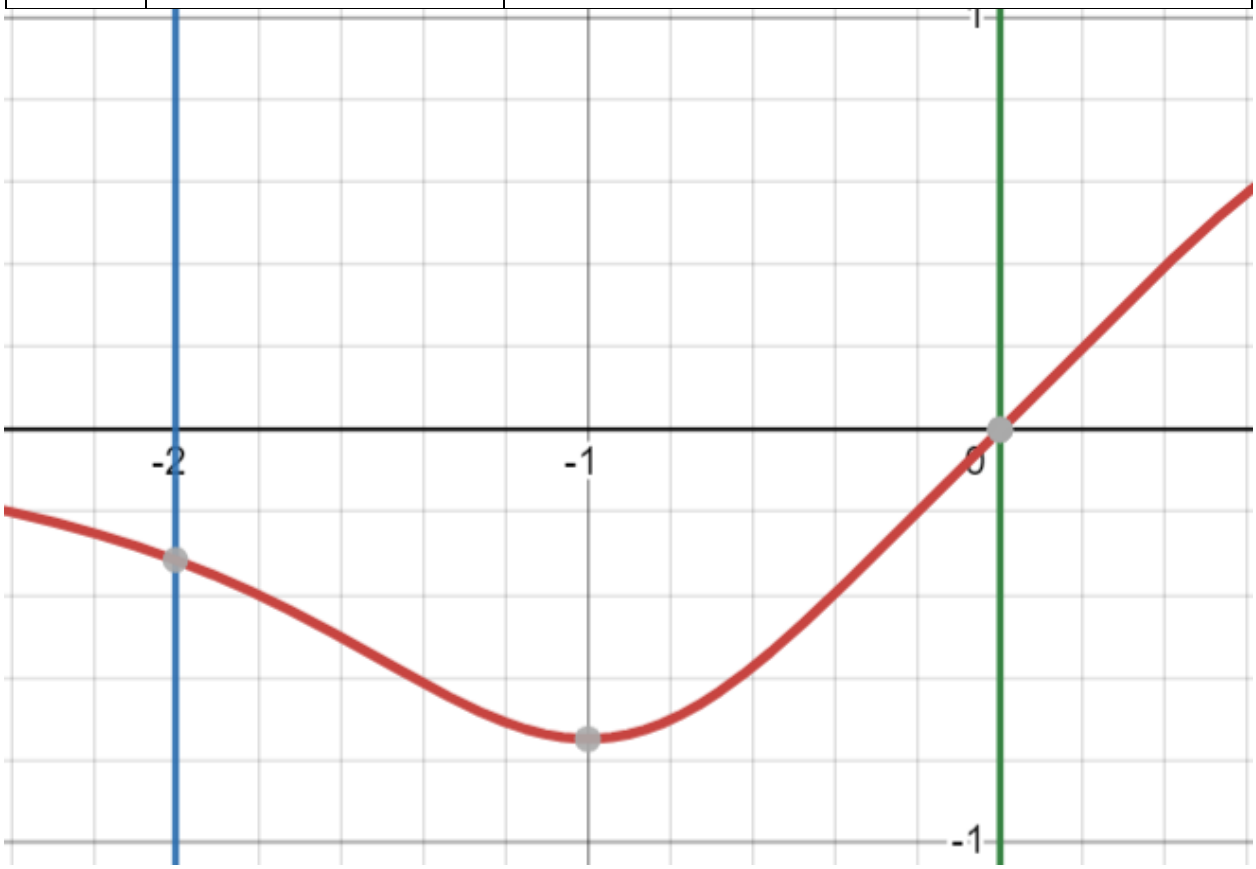
```
7)
class LogarithmicallyApproximation(Approximation):
    function_type = "Логарифмическая зависимость"

    def find_an_approximation(self, list_of_x: list, list_of_y: list) ->
Function:
        try:
            SLNX = sum(log(x) for x in list_of_x)
            SLNXX = sum(log(x) * log(x) for x in list_of_x)
            SY = sum(list_of_y)
            SYLNX = sum(log(list_of_x[i]) * list_of_y[i] for i in
range(len(list_of_x)))
            n = len(list_of_x)
        except ValueError:
            return None

        try:
            answer = self.solve_matrix([[SLNXX, SLNX], [SLNX, n]], [SYLNX,
SY])
            a = answer[0]
            b = answer[1]
            if a is None or b is None or self.isNaN(a) or self.isNaN(b):
                return None
            fun = lambda x: a * log(x) + b
```

```
            s = sum((fun(list_of_x[i]) - list_of_y[i]) ** 2 for i in
range(len(list_of_x)))
            root_mean_square_deviation = sqrt(s / n)
            f = Function(fun, f'ф = {round(a, 3)}*ln(x) {round(b, 3):+}',
s, root_mean_square_deviation)
            self.print_approximation_table(list_of_x, list_of_y, f,
self.function_type)
            return f
        except TypeError:
            return None
```

Вычислительная реализация:

Вариант 7:

7	$y = \frac{3x}{x^4 + 3}$	$x \in [-2, 0]$ $h = 0{,}2$



Использованные в лабораторной работе формулы:

$$S = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} [\varphi(x_i) - y_i]^2 \to min$$

$$\frac{\partial S}{\partial a_0} = 2\sum_{i=1}^{n} a_0 + a_1 x_i + \cdots + a_{m-1} x_i^{m-1} + a_m x_i^m - y_i = 0$$

$$\frac{\partial S}{\partial a_1} = 2\sum_{i=1}^{n} (a_0 + a_1 x_i + \cdots + a_{m-1} x_i^{m-1} + a_m x_i^m - y_i) x_i = 0$$

$$\cdots \cdots \cdots$$

$$\frac{\partial S}{\partial a_m} = 2\sum_{i=1}^{n} (a_0 + a_1 x_i + \cdots + a_{m-1} x_i^{m-1} + a_m x_i^m - y_i) x_i^m = 0$$

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

$$S = S(a_0, a_1, a_2) = \sum_{i=1}^{n} (a_0 + a_1 x_i + a_2 x_i^2 - y_i)^2 \to min$$

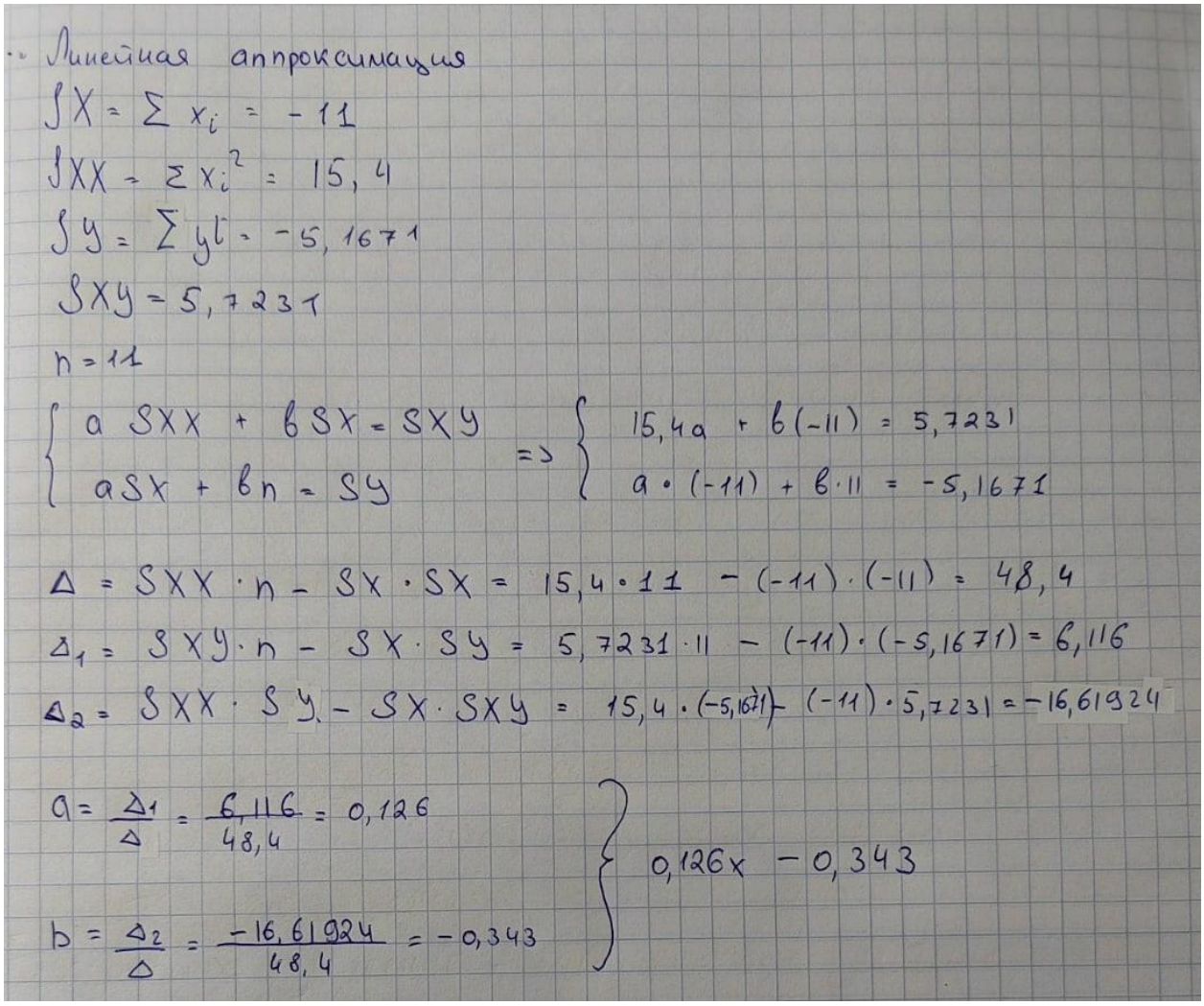
$$\begin{cases} \frac{\partial S}{\partial a_0} = 2\sum_{i=1}^{n} a_2 x_i^2 + a_1 x_i + a_0 - y_i = 0 \\ \frac{\partial S}{\partial a_1} = 2\sum_{i=1}^{n} (a_2 x_i^2 + a_1 x_i + a_0 - y_i) x_i = 0 \\ \frac{\partial S}{\partial a_2} = 2\sum_{i=1}^{n} (a_2 x_i^2 + a_1 x_i + a_0 - y_i) x_i^2 = 0 \end{cases} \quad \begin{cases} a_0 n + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} y_i \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 = \sum_{i=1}^{n} x_i y_i \\ a_0 \sum_{i=1}^{n} x_i^2 + a_1 \sum_{i=1}^{n} x_i^3 + a_2 \sum_{i=1}^{n} x_i^4 = \sum_{i=1}^{n} x_i^2 y_i \end{cases}$$

Таблица значений:

X	Y
0	0
-0.2	-0.1999
-0.4	-0.3966
-0.6	-0.5751
-0.8	-0.7039
-1.0	-0.75
-1.2	-0.7095

-1.4	-0.6139
-1.6	-0.5024
-1.8	-0.4
-2	-0.3158

Линейная аппроксимация:



Промежуточные результаты:

a = 0.126

b = -0.343

Полученная аппроксимация: P1(x)= 0.126x - 0.343

ε(0)= |p1(0)-y(0)|=|-0.343-0|=|-0.343|

X	Y	P1(x)=ax-b	ε	ε^2

0	0	-0.343	0.343	0.117649
-0.2	-0.1999	-0.3682	0.1683	0.02832489
-0.4	-0.3966	-0.3934	0.0032	0.00001024
-0.6	-0.5751	-0.4186	0.1565	0.02449225
-0.8	-0.7039	-0.4438	0.2601	0.06765201
-1.0	-0.75	-0.469	0.281	0.078961
-1.2	-0.7095	-0.4942	0.2153	0.04635409
-1.4	-0.6139	-0.5194	0.0945	0.00893025
-1.6	-0.5024	-0.5446	0.0422	0.00178084
-1.8	-0.4	-0.5698	0.1698	0.02883204
-2	-0.3158	-0.595	0.2792	0.07795264

Мера отклонения: $S = \sum \varepsilon_i^2$ = 0.48093925

Коэффициент корреляции: 0.857

Итоговые результаты:

a = 0.126

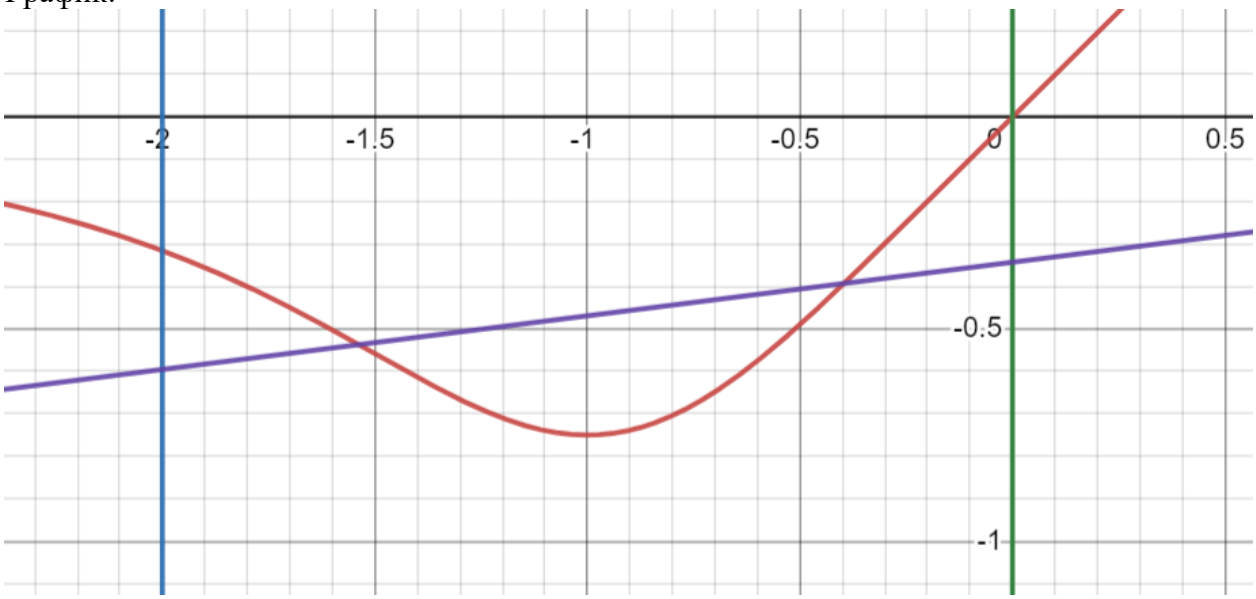
b = -0.343

Полученная аппроксимация: P1(x)= 0.126x - 0.343

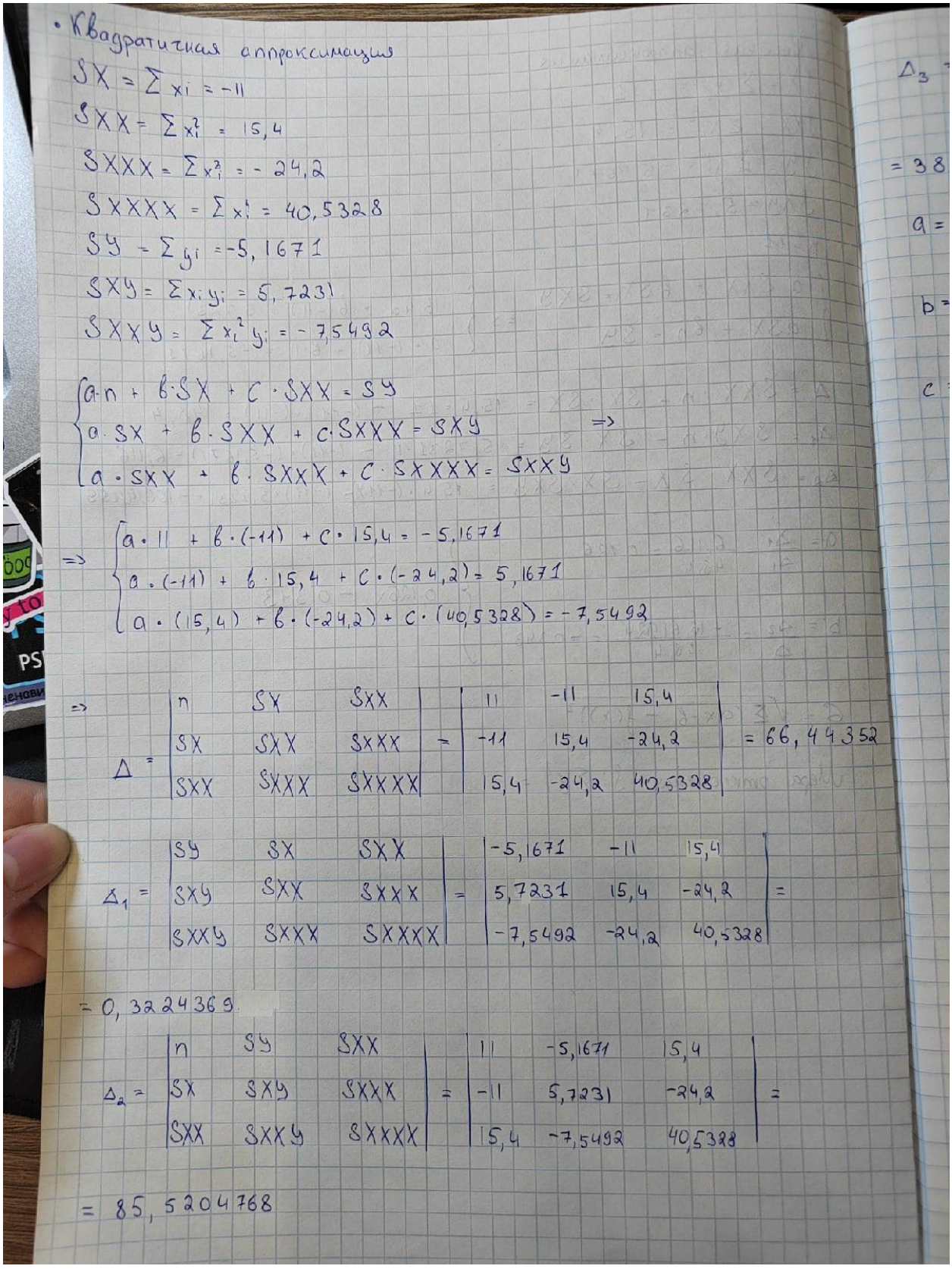
Мера отклонения: 0.48093925

Коэффициент корреляции: 0.857

График:



Квадратичная аппроксимация:



• Квадратичная аппроксимация

$SX = \sum x_i = -11$

$SXX = \sum x_i^2 = 15,4$

$SXXX = \sum x_i^3 = -24,2$

$SXXXX = \sum x_i^4 = 40,5328$

$SY = \sum y_i = -5,1671$

$SXY = \sum x_i y_i = 5,7231$

$SXXY = \sum x_i^2 y_i = -7,5492$

$$\begin{cases} a \cdot n + b \cdot SX + c \cdot SXX = SY \\ a \cdot SX + b \cdot SXX + c \cdot SXXX = SXY \\ a \cdot SXX + b \cdot SXXX + c \cdot SXXXX = SXXY \end{cases} \Rightarrow$$

$$\Rightarrow \begin{cases} a \cdot 11 + b \cdot (-11) + c \cdot 15,4 = -5,1671 \\ a \cdot (-11) + b \cdot 15,4 + c \cdot (-24,2) = 5,1671 \\ a \cdot (15,4) + b \cdot (-24,2) + c \cdot (40,5328) = -7,5492 \end{cases}$$

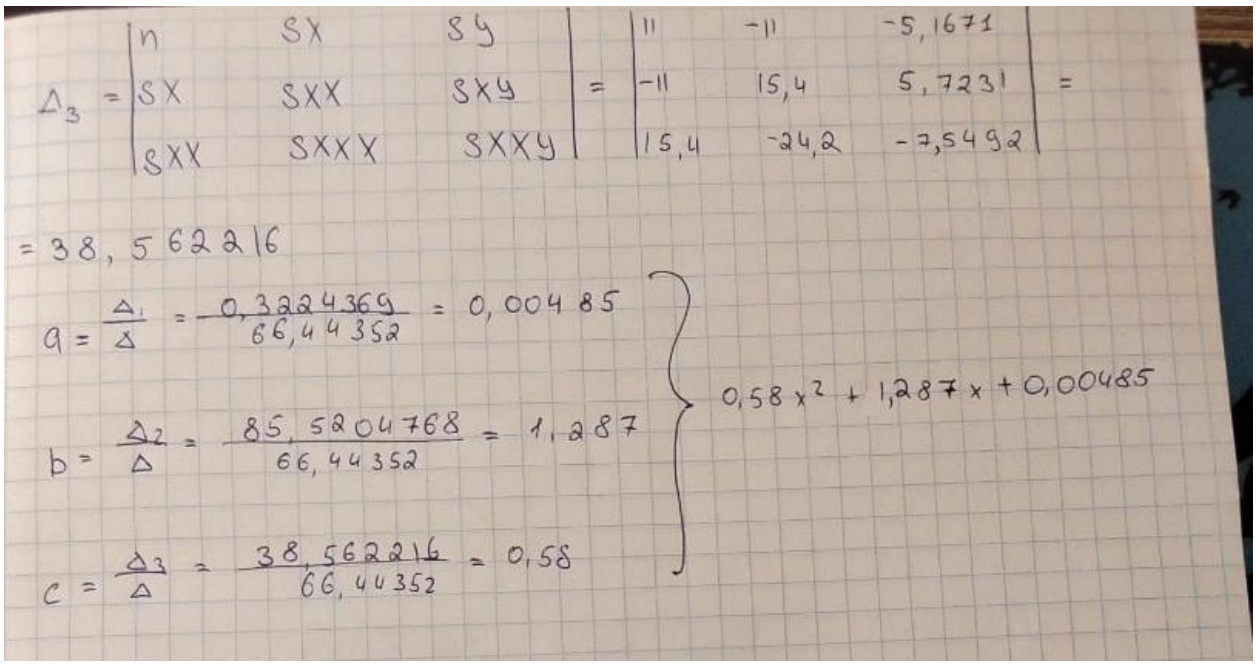
$$\Rightarrow \Delta = \begin{vmatrix} n & SX & SXX \\ SX & SXX & SXXX \\ SXX & SXXX & SXXXX \end{vmatrix} = \begin{vmatrix} 11 & -11 & 15,4 \\ -11 & 15,4 & -24,2 \\ 15,4 & -24,2 & 40,5328 \end{vmatrix} = 66,44352$$

$$\Delta_1 = \begin{vmatrix} SY & SX & SXX \\ SXY & SXX & SXXX \\ SXXY & SXXX & SXXXX \end{vmatrix} = \begin{vmatrix} -5,1671 & -11 & 15,4 \\ 5,7231 & 15,4 & -24,2 \\ -7,5492 & -24,2 & 40,5328 \end{vmatrix} =$$

$= 0,3224369$

$$\Delta_2 = \begin{vmatrix} n & SY & SXX \\ SX & SXY & SXXX \\ SXX & SXXY & SXXXX \end{vmatrix} = \begin{vmatrix} 11 & -5,1671 & 15,4 \\ -11 & 5,7231 & -24,2 \\ 15,4 & -7,5492 & 40,5328 \end{vmatrix} =$$

$= 85,5204768$



Промежуточные результаты:

a = 0.00485

b = 1.287

c = 0.58

Полученная аппроксимация: P2(x)=0.58x^2 + 1.287x + 0.00485

ε(0)= |p(0)-y(0)|=|0.00485-0|=|0.00485|

X	Y	P2(x)=a0+a1x+a2x	ε	ε ^2
0	0	0.00485	0.00485	0.00002352
-0.2	-0.1999	-0.22935	0.02945	0.0008673
-0.4	-0.3966	-0.41715	0.02055	0.0004223
-0.6	-0.5751	-0.55855	0.01655	0.0002739
-0.8	-0.7039	-0.65355	0.05035	0.00253512
-1.0	-0.75	-0.70215	0.04785	0.00228962

-1.2	-0.7095	-0.70435	0.00515	0.00002652
-1.4	-0.6139	-0.66016	0.04625	0.00213906
-1.6	-0.5024	-0.56955	0.06715	0.00450912
-1.8	-0.4	-0.43255	0.03255	0.0010595
-2	-0.3158	-0.24915	0.06665	0.00216946

Мера отклонения: $S = \sum \varepsilon_i^2$ = 0.0163154

Окончательные результаты:

a = 0.00485

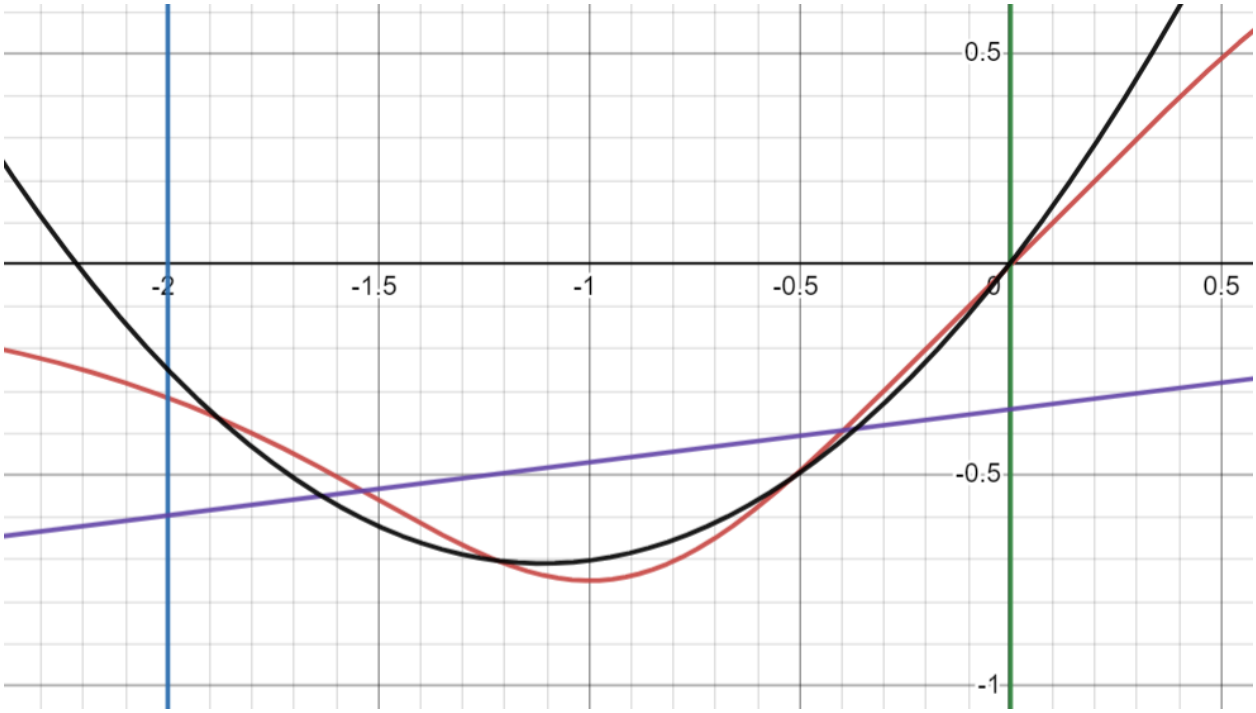
b = 1.287

c = 0.58

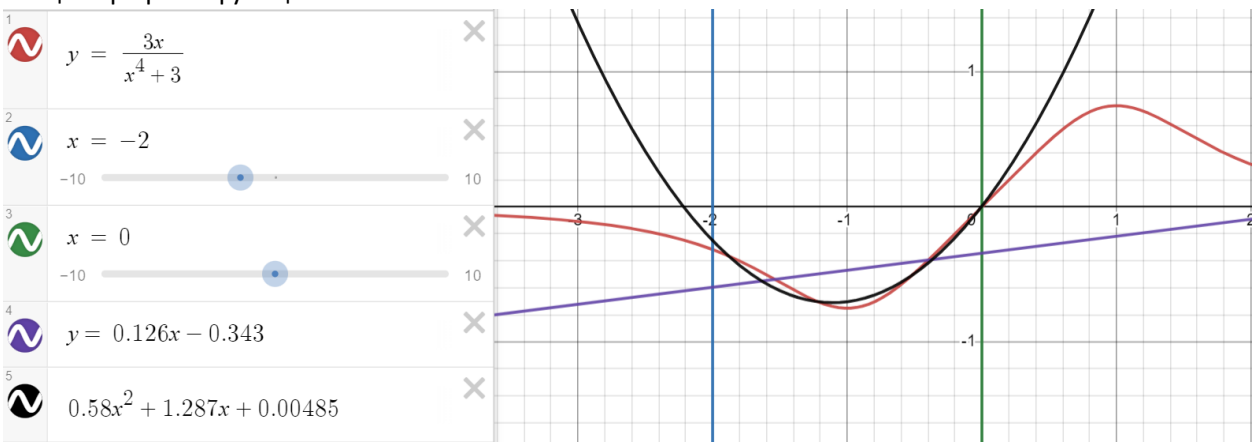
Полученная аппроксимация: P2(x)=0.58x^2 + 1.287x + 0.00485

Мера отклонения: 0.0163154

График:



Общий график с функциями:



Вывод:

Выполнив данную лабораторную работу, я сделала следующие выводы:

Во-первых, необходимо обратить внимание на разницу в постановке задач аппроксимации и интерполяции: интерполянт должен принадлежать к определенному классу и в точках $xi(i = 0,1,...,n)$ принимать те же значения, что и исходная функция, для аппроксиманта это требование обязательным не является, но должен выполняться критерий наилучшего приближения.

В большинстве практических случаев вычислений нам требуется установить определенный вид функциональной зависимости между характеристиками изучаемого явления. Этой цели и служит задача о приближении функции.

Т.е. задача о приближении (аппроксимации) функции состоит в том, чтобы данную функцию f(x) приближенно заменить (аппроксимировать) некоторой функцией ϕ(x), значения которой в заданной области мало отличались от опытных данных (f(x) ≈ ϕ(x)).

Построение эмпирической формулы состоит из 2 этапов:

1. Подбор общего вида формулы.

Иногда он известен из физических соображений.

Если характер зависимости неизвестен, то первоначально его выбирают геометрически: экспериментальные точки наносятся на график, и примерно угадывается общий вид зависимости путем сравнения полученной кривой с графиками известных функций (многочлена, логарифмической, показательной функций и т.п.).

Выбор вида эмпирической зависимости – наиболее сложная часть решения задачи, так как класс известных аналитических зависимостей необъятен. Практика, однако, показывает, что при выборе аналитической зависимости достаточно ограничиться довольно узким кругом функций: линейные, степенные и показательные.

2. Определение значений параметров аппроксимирующей функции.

(это и было реализовано в самой лабораторной работе, а алгоритм описан в теоретическом блоке выше)