

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет ПИ и КТ

Лабораторная работа №4

по дисциплине: «Вычислительная математика»

«Аппроксимация функции»

Вариант 1

Выполнил:

**Болорболд Аригуун**,

группа P3211

Преподаватель:

**Малышева Татьяна Алексеевна**

Санкт-Петербург

2024



1. **Цель работы:**

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

1. **Вычислительная реализация задачи:**

Линейная аппроксимация:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* |
|  | *–2* | *–1,8* | *–1,6* | *–1,4* | *–1,2* | *–1,0* | *–0,8* | *–0,6* | *–0,4* | *–0,2* | *0* |
|  | *–1,412* | *–1,879* | *–2,542* | *–3,47* | *–4,685* | *–6* | *–6,81* | *–6,374* | *–4,68* | *–2,396* | *0* |

Вычисляем суммы:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *–2* | *–1,8* | *–1,6* | *–1,4* | *–1,2* | *–1,0* | *–0,8* | *–0,6* | *–0,4* | *–0,2* | *0* |
|  | *–1,412* | *–1,879* | *–2,542* | *–3,47* | *–4,685* | *–6* | *–6,81* | *–6,374* | *–4,68* | *–2,396* | *0* |
|  | *7,743* | *6,926* | *6,109* | *5,293* | *4,476* | *3,659* | *2,842* | *2,025* | *1,209* | *0,392* | *–0,425* |
|  | *83,814* | *77,528* | *74,84* | *76,79* | *83,924* | *93,296* | *93,161* | *70,543* | *34,68* | *7,773* | *0,181* |

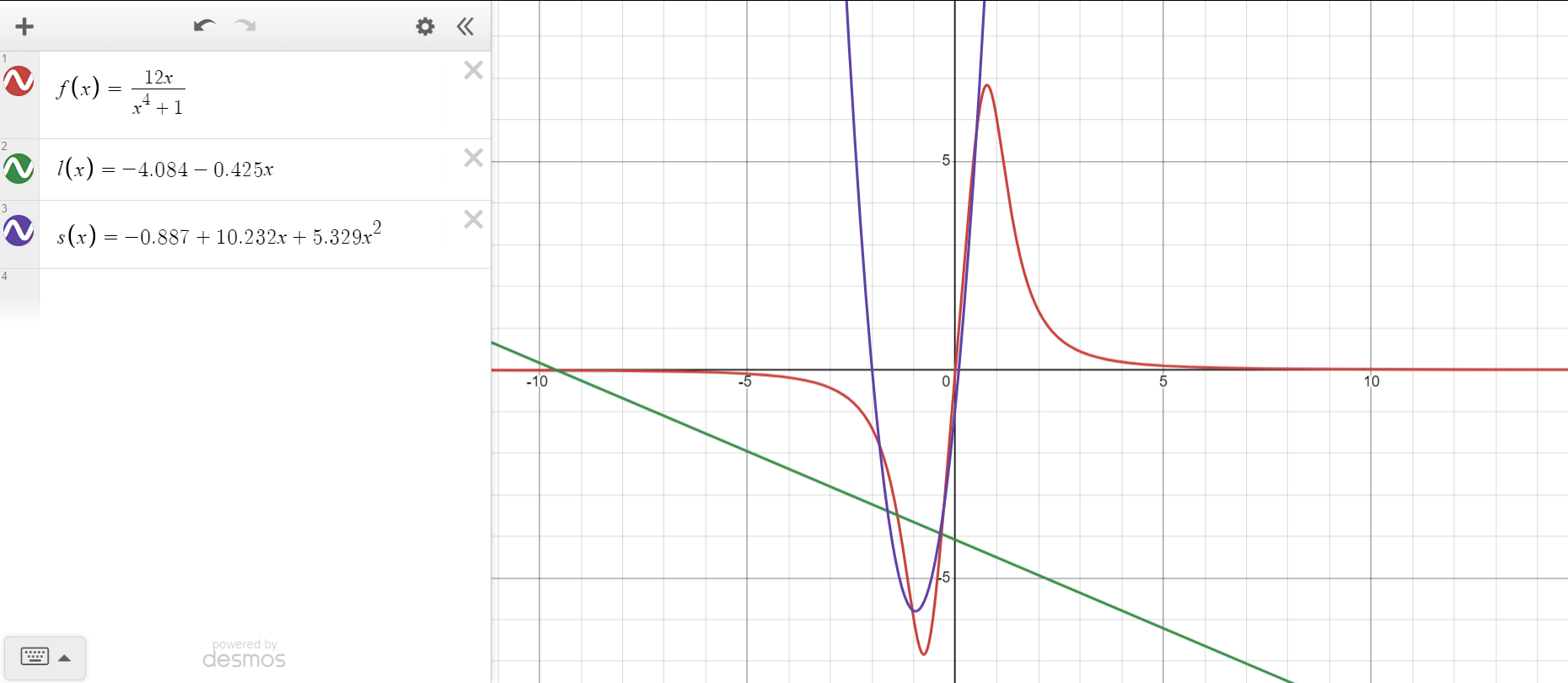
Квадратичная аппроксимация:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* |
|  | *–2* | *–1,8* | *–1,6* | *–1,4* | *–1,2* | *–1,0* | *–0,8* | *–0,6* | *–0,4* | *–0,2* | *0* |
|  | *–1,412* | *–1,879* | *–2,542* | *–3,47* | *–4,685* | *–6* | *–6,81* | *–6,374* | *–4,68* | *–2,396* | *0* |

Вычисляем суммы:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *–2* | *–1,8* | *–1,6* | *–1,4* | *–1,2* | *–1,0* | *–0,8* | *–0,6* | *–0,4* | *–0,2* | *0* |
|  | *–1,412* | *–1,879* | *–2,542* | *–3,47* | *–4,685* | *–6* | *–6,81* | *–6,374* | *–4,68* | *–2,396* | *0* |
|  | *–0,035* | *–2,039* | *–3,616* | *–4,767* | *–5,492* | *–5,79* | *–5,662* | *–5,108* | *–4,127* | *–2,72* | *–0,887* |
|  | *1,896* | *0,026* | *1,153* | *1,682* | *0,651* | *0,044* | *1,318* | *1,603* | *0,306* | *0,105* | *0,787* |

У квадратичной аппроксимации среднеквадратичное отклонение меньше, поэтому это приближение лучшее.

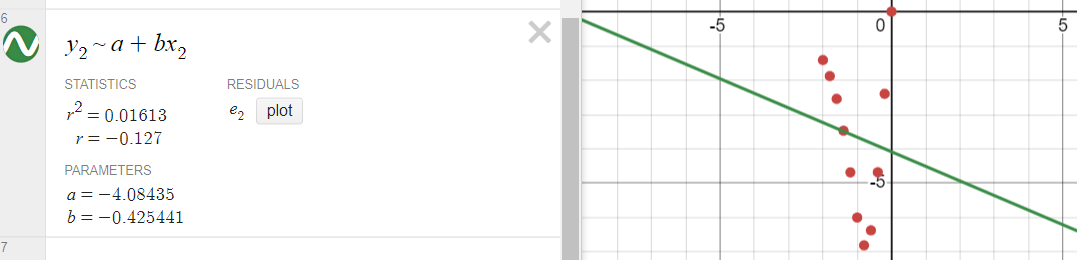


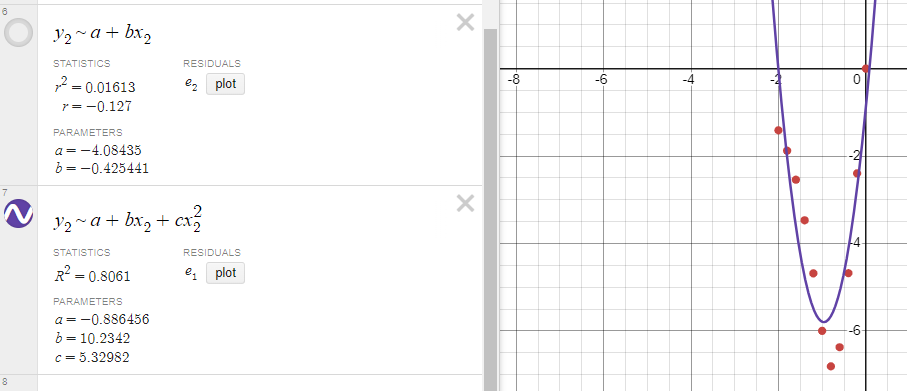
1. **Программная реализация задачи:**

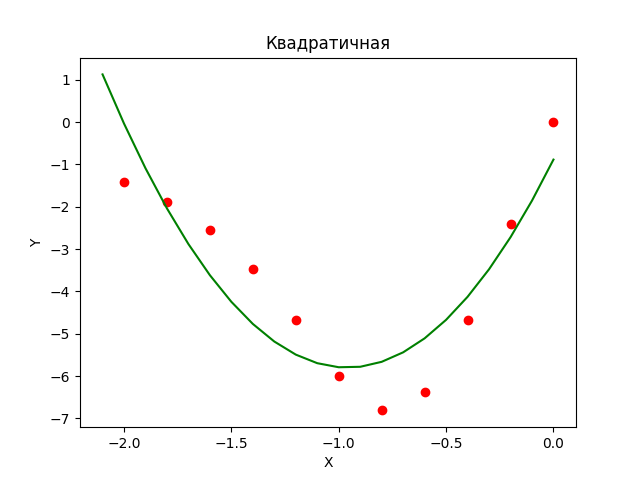
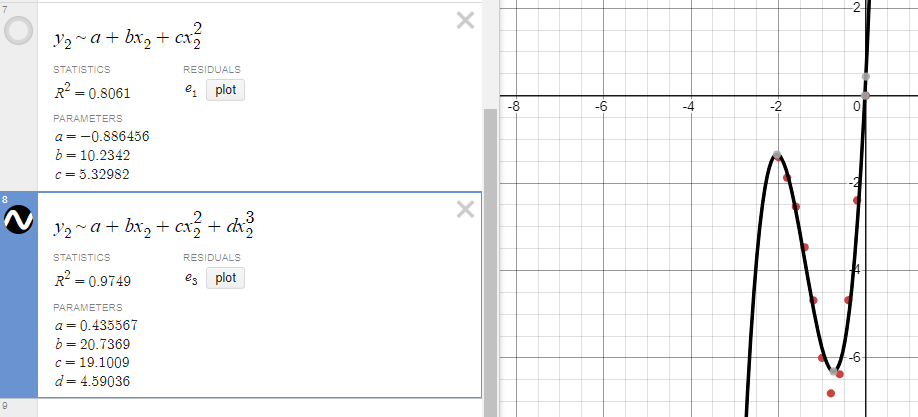
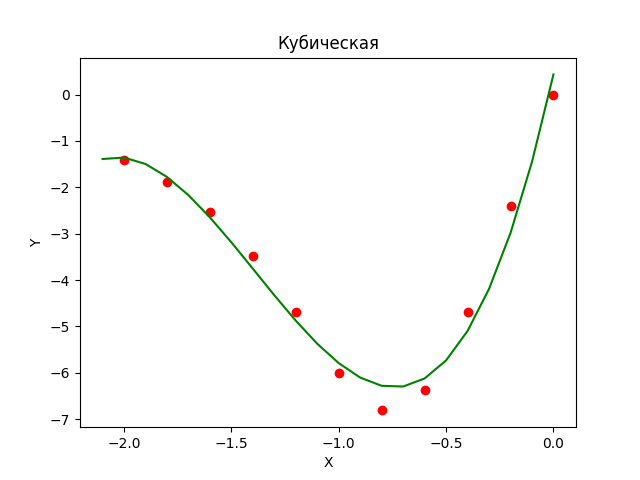
import inspect  
from math import sqrt, exp, log  
  
import matplotlib.pyplot as plt  
  
def add\_col(m, col):  
 for k, row in enumerate(m):  
 row.append(col[k])  
 return m  
  
def remove\_last\_col(m):  
 for k, row in enumerate(m):  
 row.pop()  
 return m  
  
def plus(src, ind, m):  
 for i in range(src + 1, len(m)):  
 \_plus(src, i, m, -m[i][ind] / m[src][ind])  
  
def \_plus(src, dest, m, mul: float = 1):  
 for i in range(len(m[0])):  
 m[dest][i] += m[src][i] \* mul  
  
def swap(src, dest, m):  
 m[src], m[dest] = m[dest], m[src]  
  
def rang(m):  
 return sum(any(row) for row in m)  
  
def determinant(m, k):  
 p = 1  
 for i in range(len(m)):  
 p \*= m[i][i]  
 return (-1) \*\* k \* p  
  
def solve(m):  
 k = 0  
 row = 0  
 col = 0  
 n = len(m)  
 while col < n:  
 for j in range(row, n):  
 if m[j][col]:  
 swap(j, row, m)  
 k += row != j  
 plus(row, col, m)  
 row += 1  
 break  
 col += 1  
 xs = []  
 for i in range(len(m)):  
 x = m[len(m) - i - 1][-1]  
 for j in range(1, i + 2):  
 if j == i + 1:  
 x /= m[len(m) - i - 1][-j - 1]  
 else:  
 x -= m[len(m) - i - 1][-j - 1] \* xs[j - 1]  
 xs.append(x)  
 return xs[::-1]  
  
def summator\_of\_exponents(arr, exp: int):  
 res = []  
 for i in range(1, exp + 1):  
 res.append(sum(a \*\* i for a in arr))  
 return res  
  
def summator\_of\_products(arr1, arr2, exp: int):  
 res = []  
 for i in range(exp + 1):  
 res.append(sum(x \*\* i \* y for x, y in zip(arr1, arr2)))  
 return res  
def linear\_approximation(xs, ys, n):  
 args = summator\_of\_exponents(xs, 2)  
 vals = summator\_of\_products(xs, ys, 1)  
 ext\_matrix = add\_col(  
 [  
 [n, args[0]],  
 [args[0], args[1]]  
 ],  
 [vals[0], vals[1]]  
 )  
  
 a, b = solve(ext\_matrix)  
 return lambda x: a + b \* x, a, b  
def quadratic\_approximation(xs, ys, n):  
 args = summator\_of\_exponents(xs, 4)  
 vals = summator\_of\_products(xs, ys, 2)  
 ext\_matrix = add\_col(  
 [  
 [n, args[0], args[1]],  
 [args[0], args[1], args[2]],  
 [args[1], args[2], args[3]]  
 ],  
 [vals[0], vals[1], vals[2]]  
 )  
  
 a, b, c = solve(ext\_matrix)  
 return lambda x: a + b \* x + c \* x \*\* 2, a, b, c  
  
def cubic\_approximation(xs, ys, n):  
 args = summator\_of\_exponents(xs, 6)  
 vals = summator\_of\_products(xs, ys, 3)  
 ext\_matrix = add\_col(  
 [  
 [n, args[0], args[1], args[2]],  
 [args[0], args[1], args[2], args[3]],  
 [args[1], args[2], args[3], args[4]],  
 [args[2], args[3], args[4], args[5]]  
 ],  
 [vals[0], vals[1], vals[2], vals[3]]  
 )  
  
 a, b, c, d = solve(ext\_matrix)  
 return lambda x: a + b \* x + c \* x \*\* 2 + d \* x \*\* 3, \  
 a, b, c, d  
  
def exponential\_approximation(xs, ys, n):  
 ys\_ = list(map(log, ys))  
 \_, a\_, b\_ = linear\_approximation(xs, ys\_, n)  
 a = exp(a\_)  
 b = b\_  
 return lambda x: a \* exp(b \* x), a, b  
  
def logarithmic\_approximation(xs, ys, n):  
 xs\_ = list(map(log, xs))  
 \_, a\_, b\_ = linear\_approximation(xs\_, ys, n)  
 a = a\_  
 b = b\_  
 return lambda x: a + b \* log(x), a, b  
  
def power\_approximation(xs, ys, n):  
 xs\_ = list(map(log, xs))  
 ys\_ = list(map(log, ys))  
 \_, a\_, b\_ = linear\_approximation(xs\_, ys\_, n)  
 a = exp(a\_)  
 b = b\_  
 return lambda x: a \* x \*\* b, a, b  
  
def calc\_deviation(xs, ys, fi):  
 return sum((eps \*\* 2 for eps in [fi(x) - y for x, y in zip(xs, ys)]))  
  
def calc\_standard\_deviation(xs, ys, fi, n):  
 return sqrt(sum(((fi(x) - y) \*\* 2 for x, y in zip(xs, ys))) / n)  
  
def calc\_pearson\_correlation\_coefficient(xs, ys, n):  
 av\_x, av\_y = sum(xs) / n, sum(ys) / n  
 return sum((x - av\_x) \* (y - av\_y) for x, y in zip(xs, ys)) / \  
 sqrt(sum((x - av\_x) \*\* 2 for x in xs) \*  
 sum((y - av\_y) \*\* 2 for y in ys))  
  
def calc\_coefficient\_of\_determination(xs, ys, fi, n):  
 return 1 - sum((y - fi(x)) \*\* 2 for x, y in zip(xs, ys)) / (sum(fi(x) \*\* 2 for x in xs) - sum(fi(x) for x in xs) \*\* 2 / n)  
  
def get\_str\_content\_of\_func(func):  
 str\_func = inspect.getsourcelines(func)[0][0]  
 return str\_func.split('lambda x: ')[-1].split(',')[0].strip()  
  
def draw\_plot(a, b, func, dx=0.1):  
 xs, ys = [], []  
 a -= dx  
 b += dx  
 x = a  
 while x <= b:  
 xs.append(x)  
 ys.append(func(x))  
 x += dx  
 plt.plot(xs, ys, 'g')  
  
def read\_number(s: str):  
 while True:  
 try:  
 return float(input(s))  
 except ValueError:  
 continue  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 read\_number("Введите количество точек: ")  
 xs = list(map(float, input("x: ").split()))  
 ys = list(map(float, input("y: ").split()))  
 if len(xs) != len(ys):  
 print("Некорректные данные")  
 n = len(xs)  
 names = {  
 linear\_approximation: "Линейная",  
 power\_approximation: "Степенная",  
 exponential\_approximation: "Экспоненциальная",  
 logarithmic\_approximation: "Логарифмическая",  
 quadratic\_approximation: "Квадратичная",  
 cubic\_approximation: "Кубическая"  
 }  
 if all(map(lambda x: x > 0, xs)) and all(map(lambda x: x > 0, ys)):  
 approximation\_funcs = [  
 linear\_approximation,  
 power\_approximation,  
 exponential\_approximation,  
 logarithmic\_approximation,  
 quadratic\_approximation,  
 cubic\_approximation  
 ]  
 else:  
 approximation\_funcs = [  
 linear\_approximation,  
 quadratic\_approximation,  
 cubic\_approximation  
 ]  
 best\_sigma = float('inf')  
 best\_apprxmt\_f = None  
 for apprxmt\_f in approximation\_funcs:  
 print(names[apprxmt\_f], ": ")  
 fi, \*coeffs = apprxmt\_f(xs, ys, n)  
 s = calc\_deviation(xs, ys, fi)  
 sigma = calc\_standard\_deviation(xs, ys, fi, n)  
 if sigma < best\_sigma:  
 best\_sigma = sigma  
 best\_apprxmt\_f = apprxmt\_f  
 r2 = calc\_coefficient\_of\_determination(xs, ys, fi, n)  
 print('fi(x) =', get\_str\_content\_of\_func(fi))  
 print(f'coeffs:', list(map(lambda cf: round(cf, 4), coeffs)))  
 print(f'S = {s:.5f}, σ = {sigma:.5f}, R^2 = {r2:.5f}')  
 if apprxmt\_f is linear\_approximation:  
 print('r =', calc\_pearson\_correlation\_coefficient(xs, ys, n))  
 plt.title(names[apprxmt\_f])  
 draw\_plot(xs[0], xs[-1], fi)  
 for i in range(n):  
 plt.scatter(xs[i], ys[i], c='r')  
 plt.xlabel("X")  
 plt.ylabel("Y")  
 plt.show()  
 print('-' \* 50)  
 print(f'Лучшая функция: {names[best\_apprxmt\_f]}')

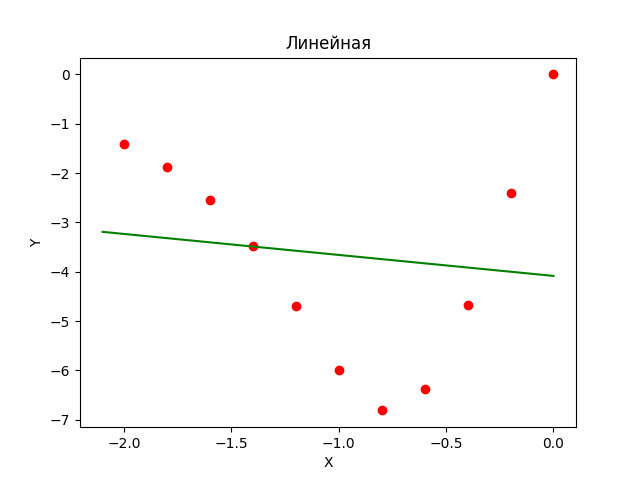
1. **Тестовые данные и сравнение с десмосом:**

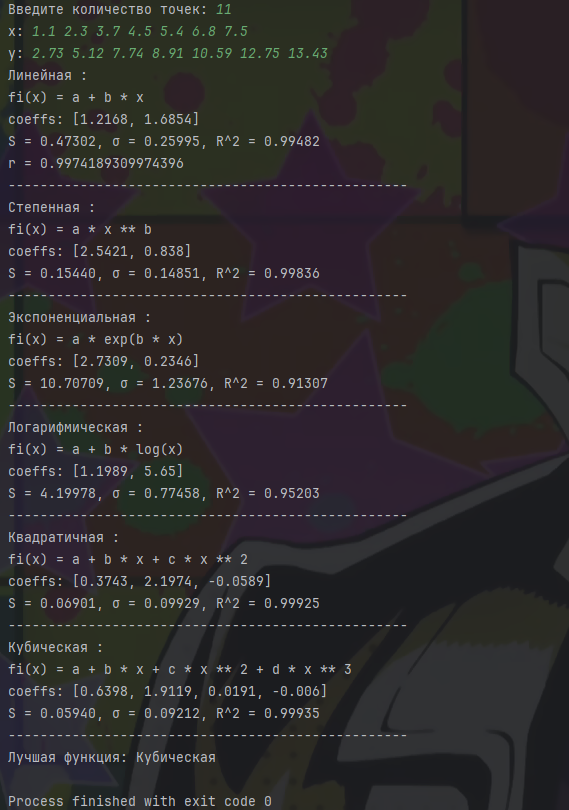


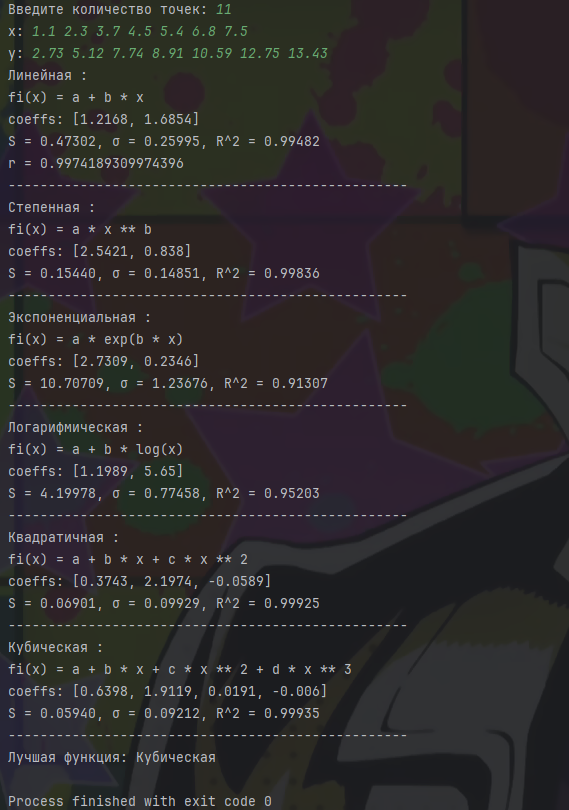


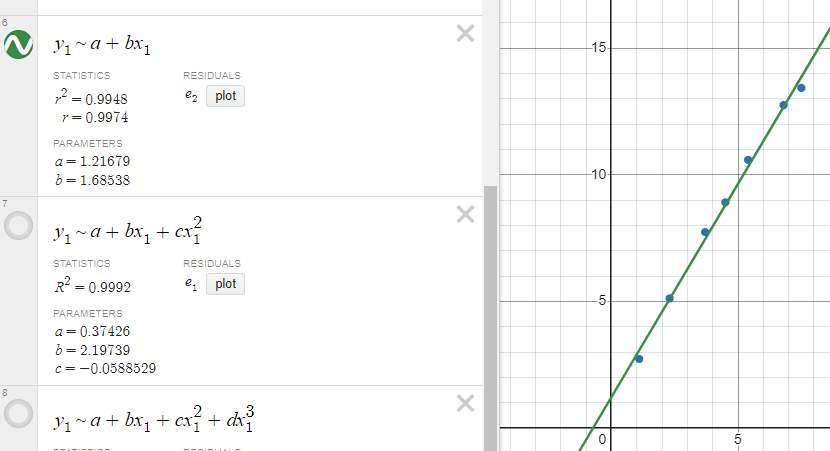


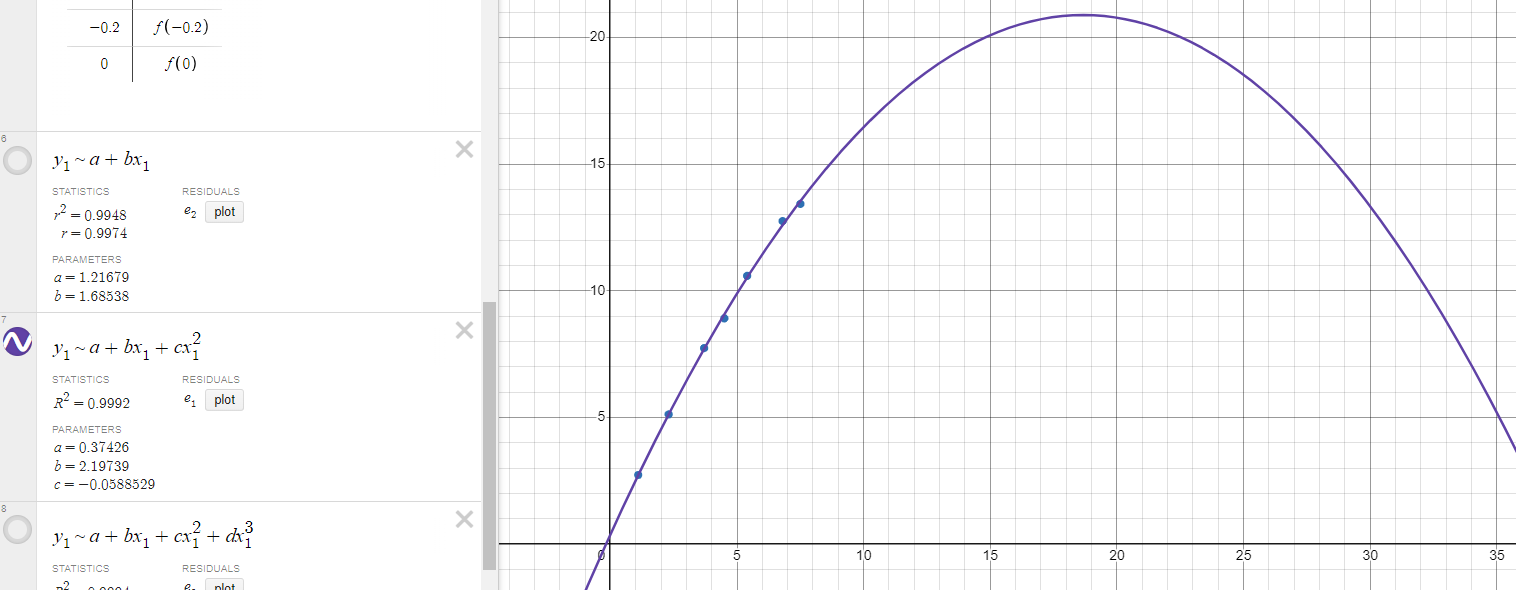


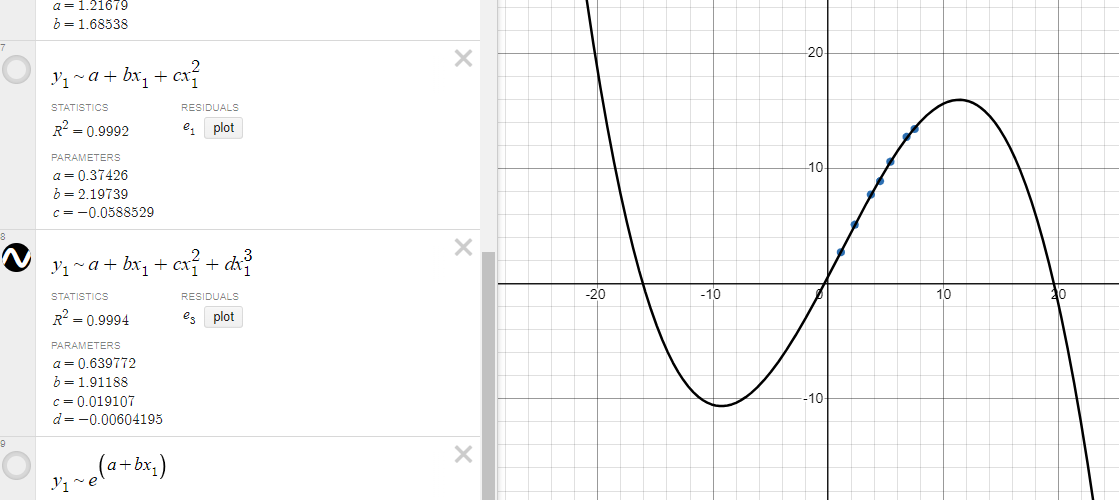


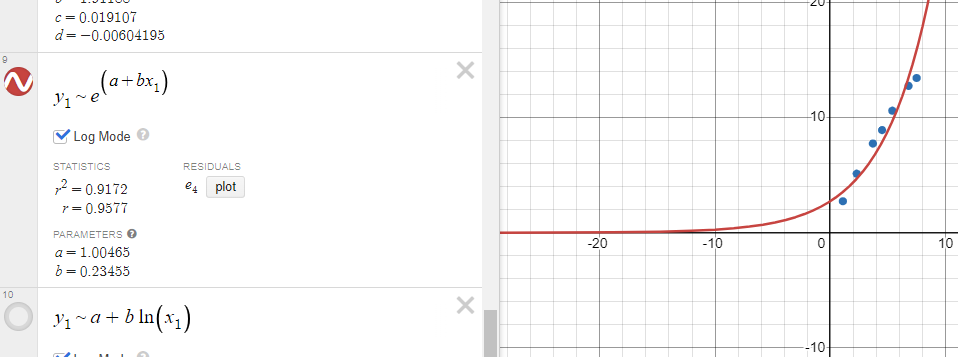


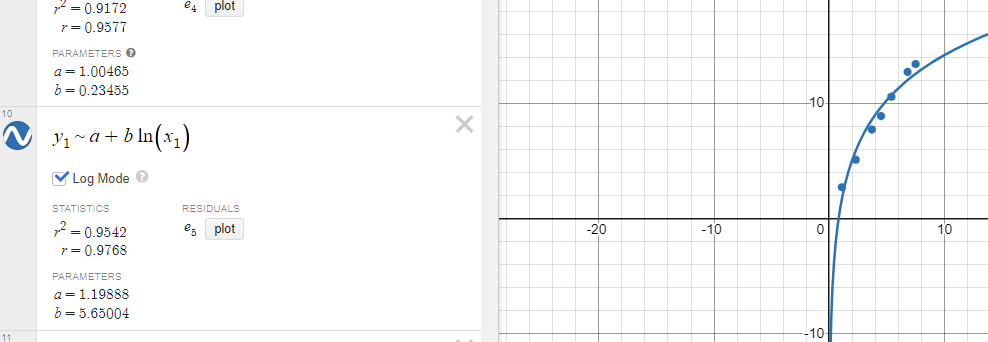


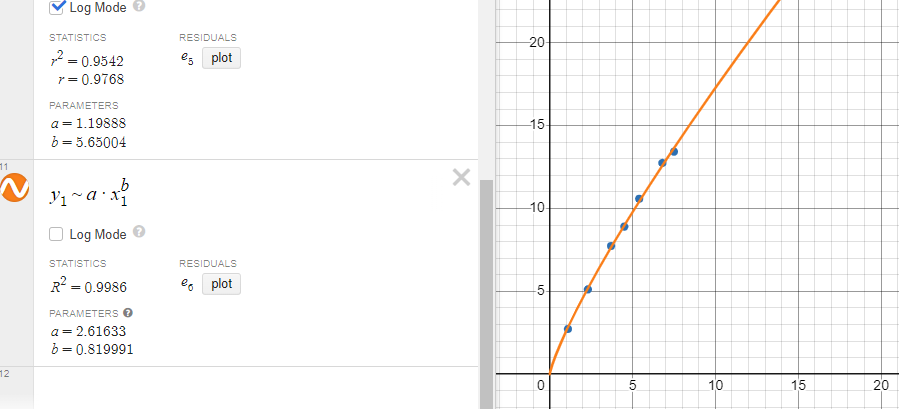


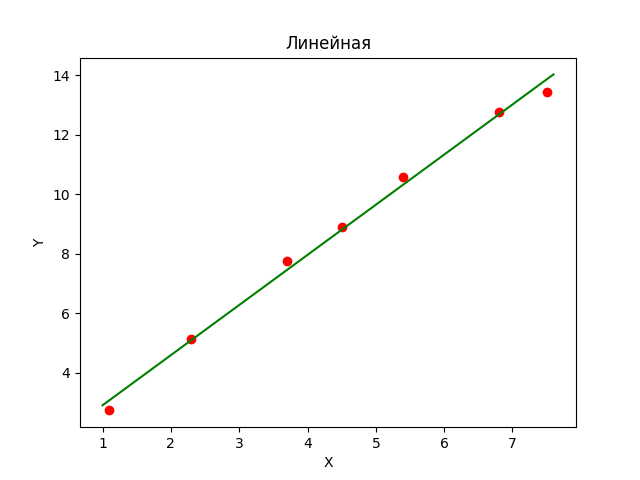
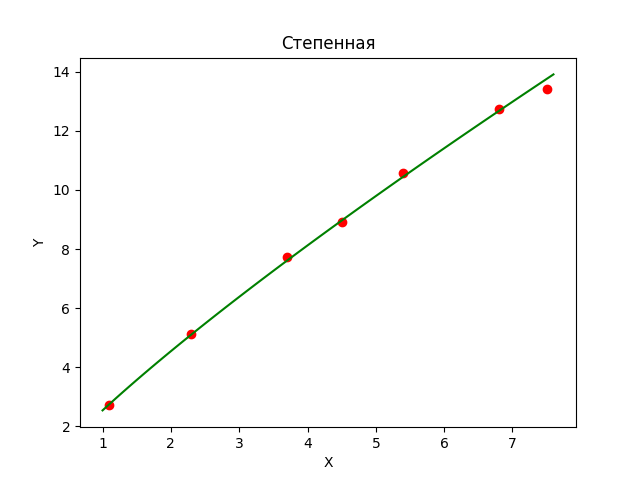
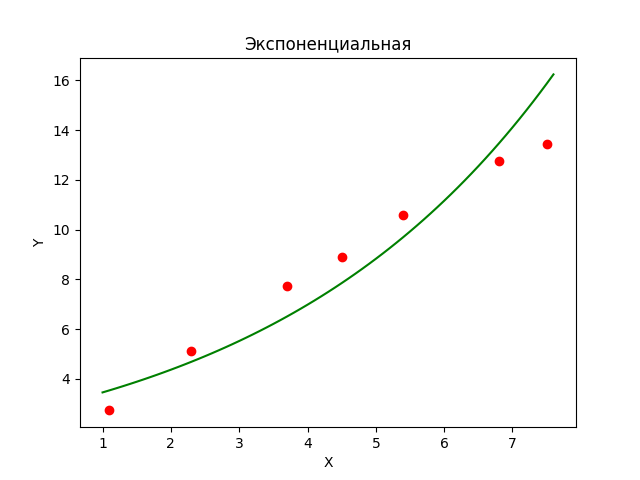
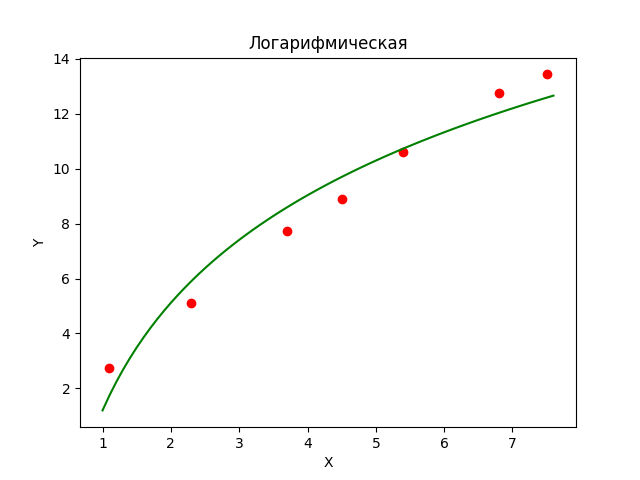
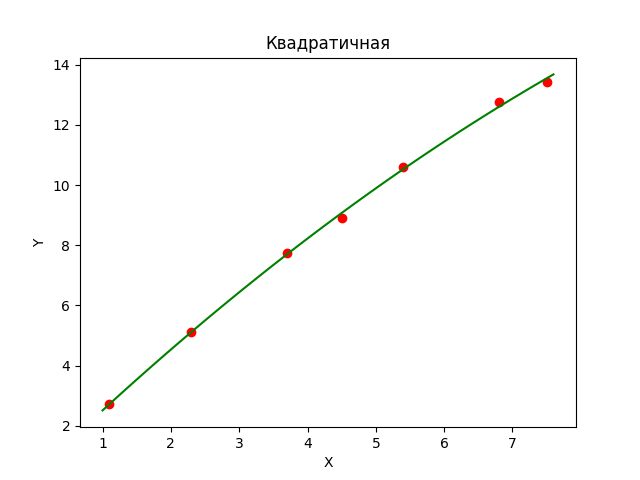
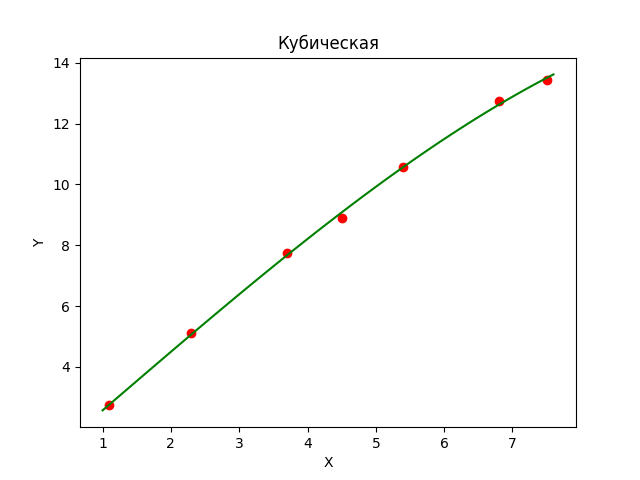












1. **Выводы:**

В ходе лабораторной работы я познакомился с аппроксимацией функции методом наименьших квадратов.

Сам МНК — хороший метод, где определяются параметры, при которых значения аппроксимирующей функции приблизительно совпадали бы со значениями исходной функции. В качестве аппроксимирующих функций обычно берут многочлены. Чем выше степень многочлена, тем точнее. Можно использовать экспоненциальные, логарифмические, степенные функции (сводя их преобразованиями к линейному аппроксимированию). Аппроксимирующая функций проходит в минимальной расстоянии от точек из заданного массива данных.