Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Вычислительная математика»

**Отчет**

По лабораторной работе №4

Вариант 6

Выполнил:

Манжиков Н.C

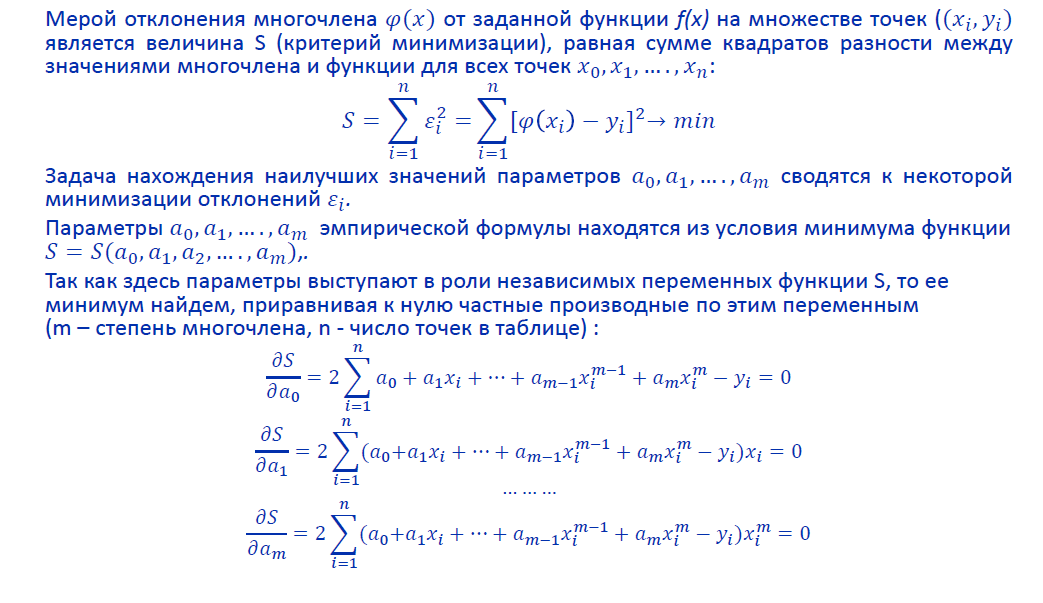
Преподаватель: Рыбаков Степан Дмитриевич

Санкт-Петербург, 2023 г.

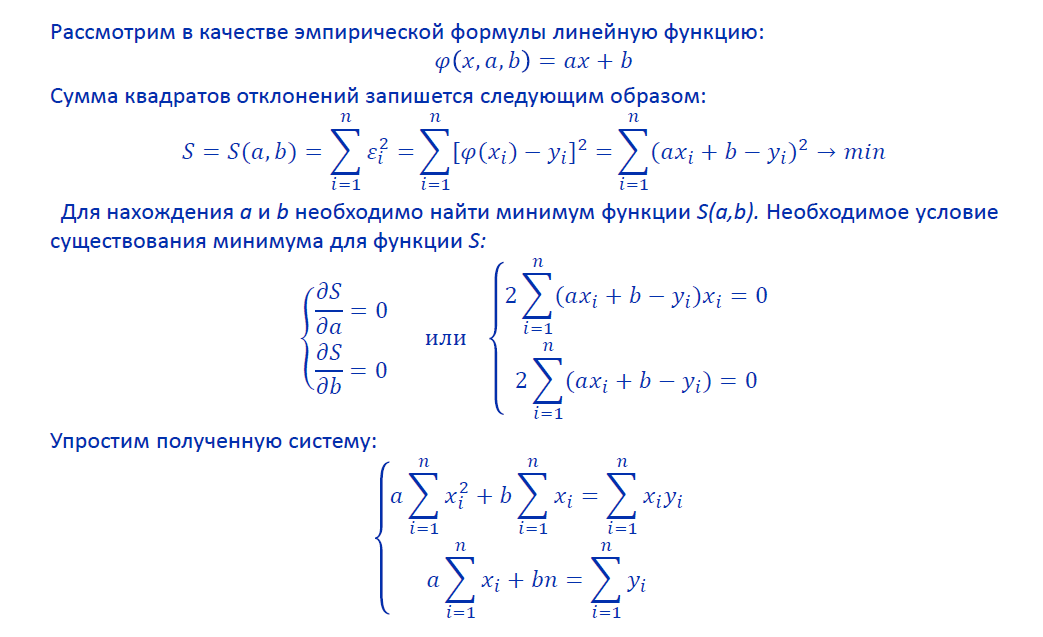
Цель работы

Нахождение аппроксимирующей функции к заданной в табличном виде функции методом средних квадратов.

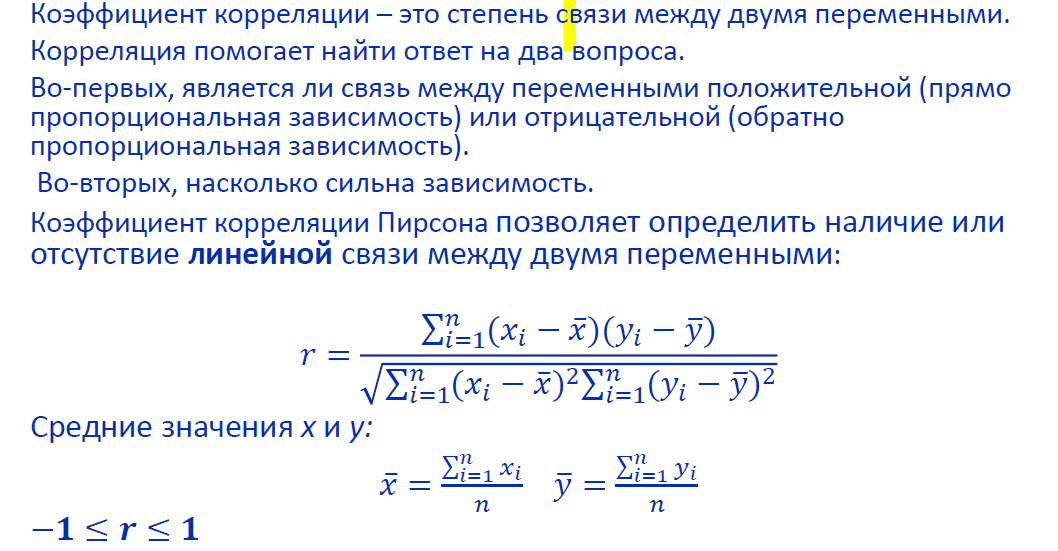
Метод Наименьших квадратов



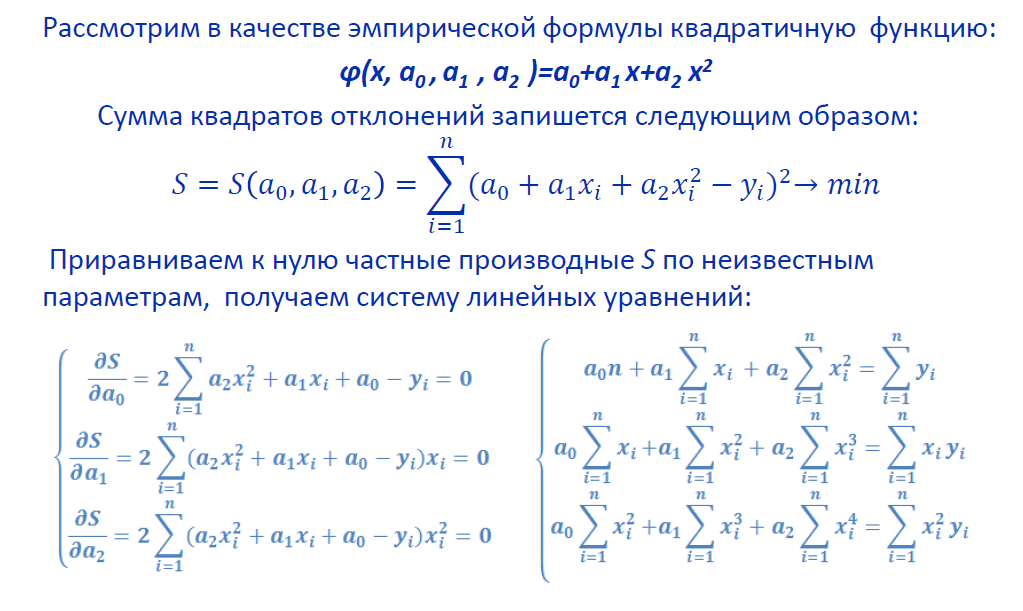
Линейная апроксимация



Коэффициент корреляции

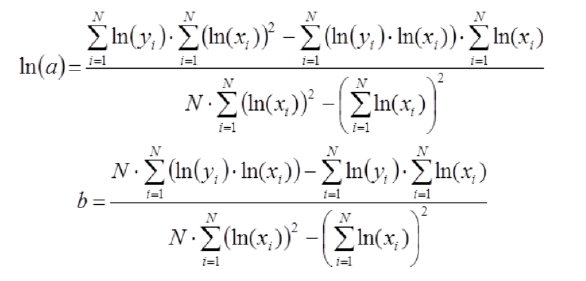


Квадратичная аппроксимация

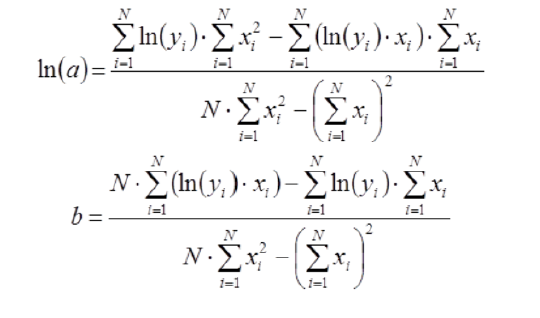


Вычисление коэффициентов для функций:

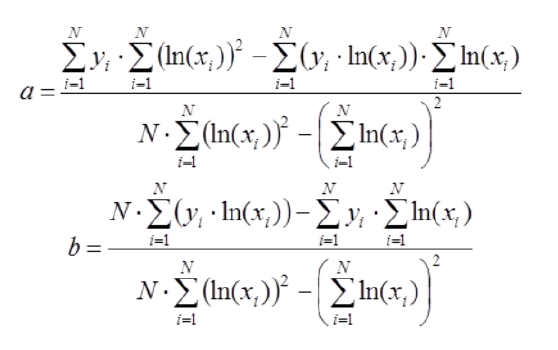
Для линейной функции:



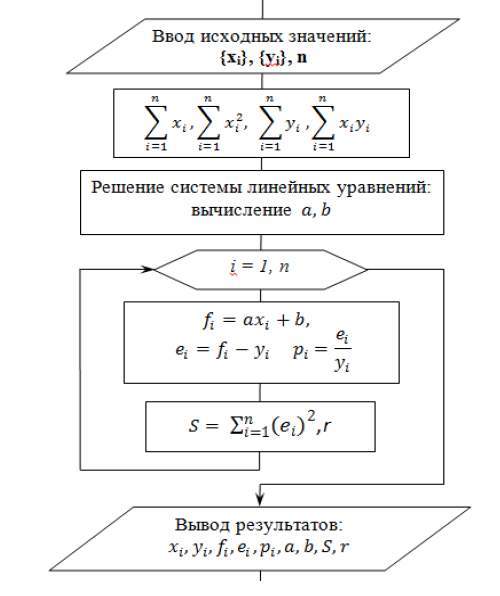
Для экспоненциальной функции:



Для логарифмической функции:



Блок схема метода наименьших квадратов:



***Вычислительная реализация***

|  |  |
| --- | --- |
| y=12x/(x^4+6) | x∈0, -2   h=0,2 |

|  |  |
| --- | --- |
| X | Y |
| -2 | -1.090 |
| -1.8 | -1.309 |
| -1.6 | -1.529 |
| -1.4 | -1.707 |
| -1.2 | -1.783 |
| -1.0 | -1.714 |
| -0.8 | -1.497 |
| -0.6 | -1.174 |
| -0.4 | -0.7966 |
| -0.2 | -0.3998 |
| 0 | 0 |

Система уравнений МНК:

an + b∑t + c∑t2 = ∑y

a∑t + b∑t2 + c∑t3 = ∑yt

a∑t2 + b∑t3 + c∑t4 = ∑yt2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| t | y | t2 | y2 | t y | t3 | t4 | t2 y |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| -0.2 | -0.4 | 0.04 | 0.16 | 0.08 | -0.008 | 0.0016 | -0.016 |
| -0.4 | -0.797 | 0.16 | 0.635 | 0.319 | -0.064 | 0.0256 | -0.127 |
| -0.6 | -1.174 | 0.36 | 1.378 | 0.704 | -0.216 | 0.13 | -0.423 |
| -0.8 | -1.497 | 0.64 | 2.241 | 1.198 | -0.512 | 0.41 | -0.958 |
| -1 | -1.714 | 1 | 2.938 | 1.714 | -1 | 1 | -1.714 |
| -1.2 | -1.783 | 1.44 | 3.179 | 2.14 | -1.728 | 2.074 | -2.568 |
| -1.4 | -1.707 | 1.96 | 2.914 | 2.39 | -2.744 | 3.842 | -3.346 |
| -1.6 | -1.529 | 2.56 | 2.338 | 2.446 | -4.096 | 6.554 | -3.914 |
| -1.8 | -1.309 | 3.24 | 1.713 | 2.356 | -5.832 | 10.498 | -4.241 |
| -2 | -1.09 | 4 | 1.188 | 2.18 | -8 | 16 | -4.36 |
| -11 | -12.999 | 15.4 | 18.684 | 15.527 | -24.2 | 40.533 | -21.667 |
| -1 | -1.182 | 1.4 | 1.699 | 1.412 |  |  |  |

Для наших данных система уравнений имеет вид

11a + -11b + 15.4c = -13

-11a + 15.4b + -24.2c = 15.53

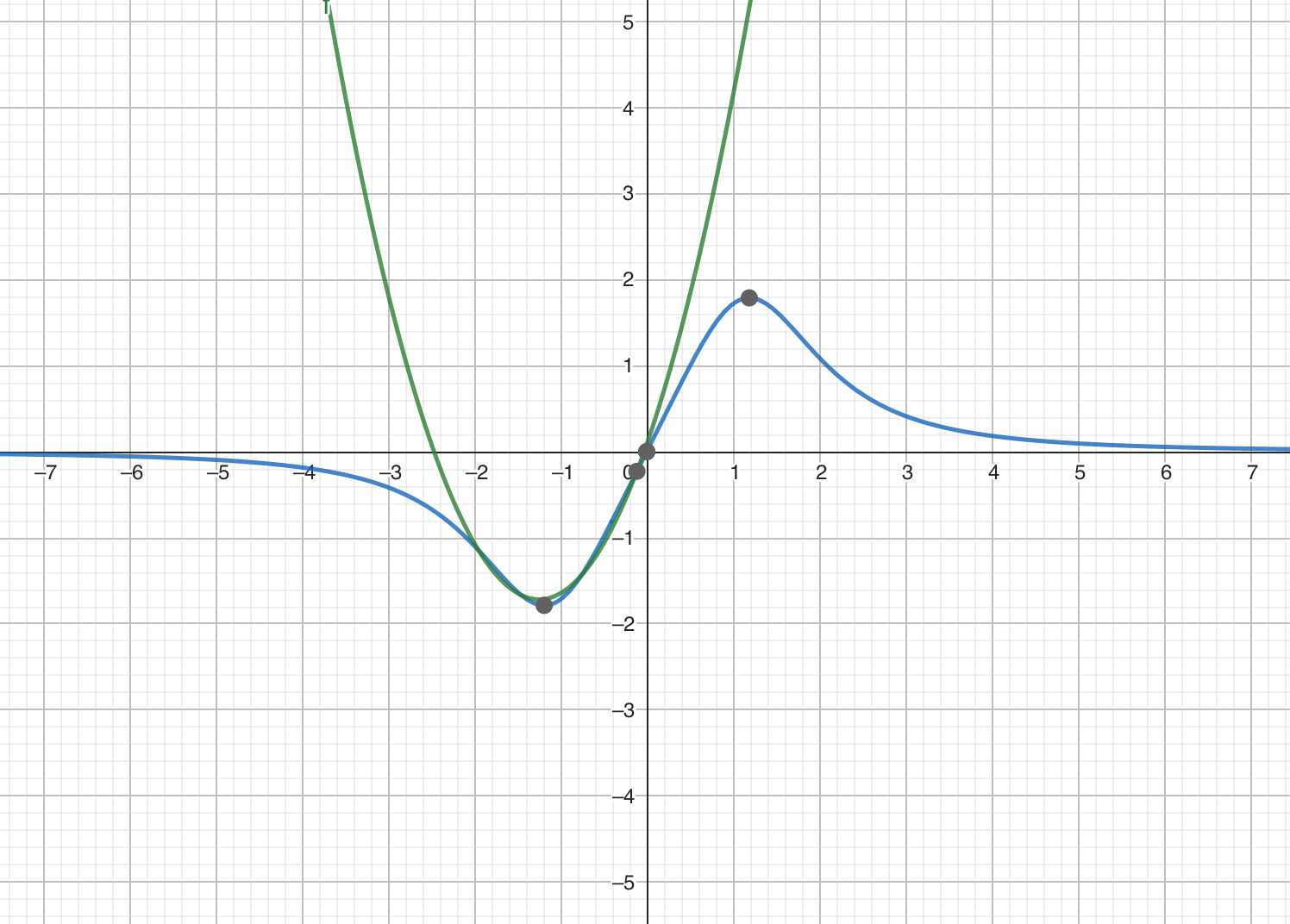
15.4a + -24.2b + 40.53c = -21.67

Получаем c = 1.156, b = 2.886, a = 0.0861

Уравнение тренда:

Оценим качество уравнения тренда с помощью средней относительной ошибки аппроксимации.

Ошибка аппроксимации в пределах 5%-7% свидетельствует о хорошем подборе уравнения тренда к исходным данным.



Система уравнений МНК:

an + b∑t = ∑y

a∑t + b∑t2 = ∑y∙t

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| t | y | t2 | y2 | t y |
| 0 | 0 | 0 | 0 | 0 |
| -0.2 | -0.4 | 0.04 | 0.16 | 0.08 |
| -0.4 | -0.797 | 0.16 | 0.635 | 0.319 |
| -0.6 | -1.174 | 0.36 | 1.378 | 0.704 |
| -0.8 | -1.497 | 0.64 | 2.241 | 1.198 |
| -1 | -1.714 | 1 | 2.938 | 1.714 |
| -1.2 | -1.783 | 1.44 | 3.179 | 2.14 |
| -1.4 | -1.707 | 1.96 | 2.914 | 2.39 |
| -1.6 | -1.529 | 2.56 | 2.338 | 2.446 |
| -1.8 | -1.309 | 3.24 | 1.713 | 2.356 |
| -2 | -1.09 | 4 | 1.188 | 2.18 |
| -11 | -12.999 | 15.4 | 18.684 | 15.527 |
| Ср.знач. | -1.182 | 1.4 | 1.699 | 1.412 |

Для наших данных система уравнений имеет вид:

11a + -11b = -13

-11a + 15.4b = 15.53

Из первого уравнения выражаем a и подставим во второе уравнение

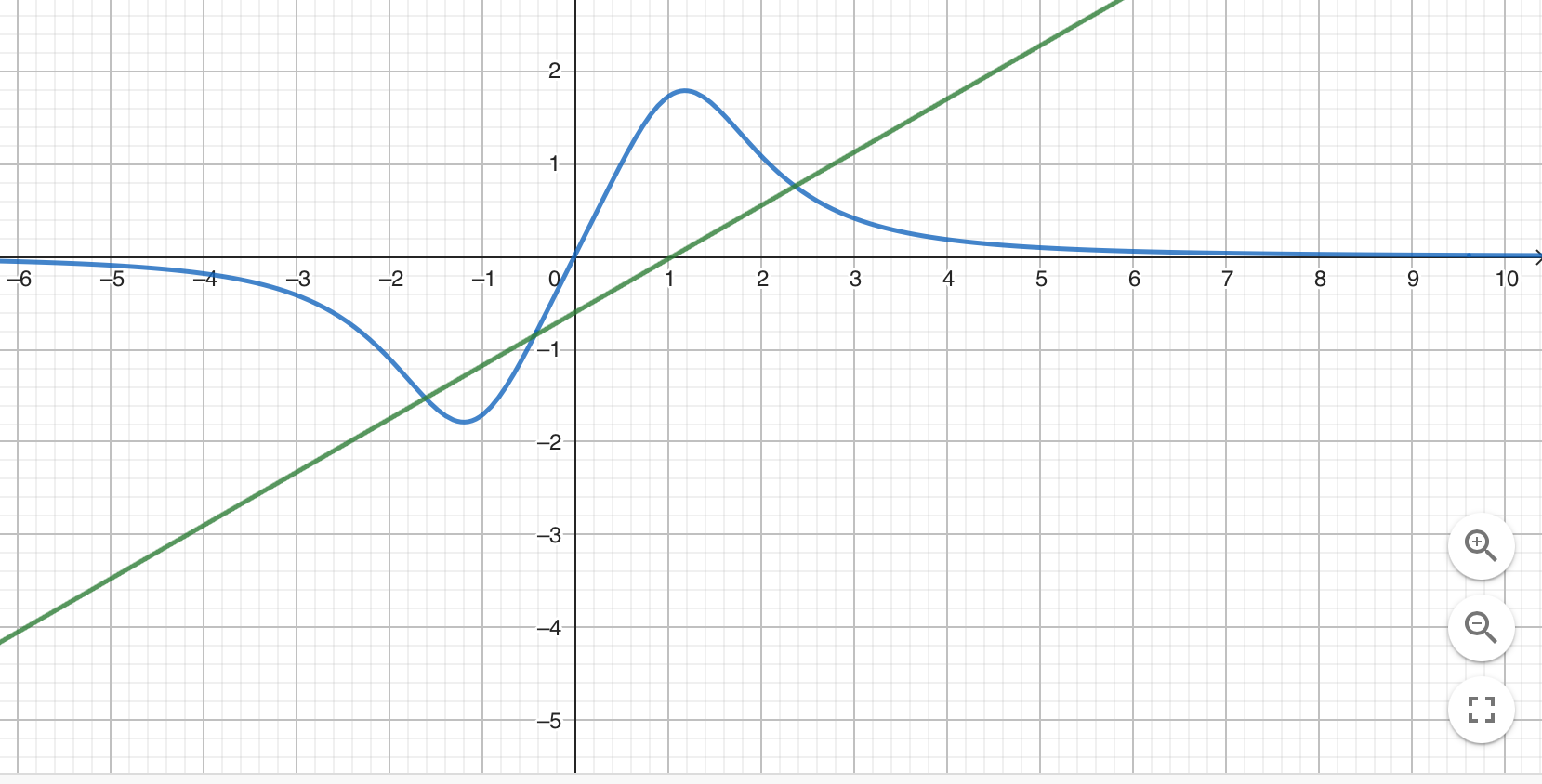
Получаем a = -0.607, b = 0.574

Уравнение тренда:

**Ошибка аппроксимации**.

Оценим качество уравнения тренда с помощью средней относительной ошибки аппроксимации.

Ошибка аппроксимации в пределах 5%-7% свидетельствует о хорошем подборе уравнения тренда к исходным данным.



Пример работы программы

Test

0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0  
0.4 0.7966 1.1746 1.4977 1.71428 1.7835 1.7070 1.529441 1.30928 1.0909

output.txt

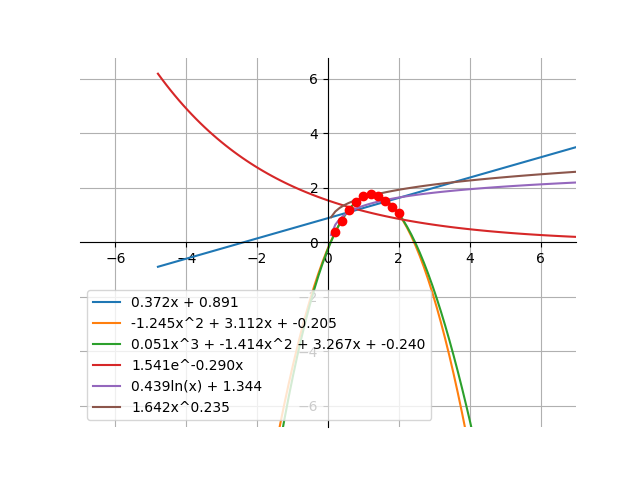
--- Линейная функция  
φ(x) = 0.372x + 0.891  
S = 1.329  
δ = 0.365  
R^2 = -1.909  
r = 0.506  
--- Квадратичная функция  
φ(x) = -1.245x^2 + 3.112x + -0.205  
S = 0.018  
δ = 0.043  
R^2 = 0.990  
--- Кубическая функция  
φ(x) = 0.051x^3 + -1.414x^2 + 3.267x + -0.240  
S = 0.018  
δ = 0.042  
R^2 = 0.990  
--- Экспоненциальная функция  
φ(x) = 1.541e^-0.290x  
S = 3.316  
δ = 0.576  
R^2 = -8.331  
--- Логарифмическая функция  
φ(x) = 0.439ln(x) + 1.344  
S = 0.853  
δ = 0.292  
R^2 = 0.085  
--- Степенная функция  
φ(x) = 1.642x^0.235  
S = 2.034  
δ = 0.451  
R^2 = -2.341  
Лучше всего аппроксимирует Кубическая функция: δ = 0.042

import numpy as np  
  
# Generate some sample data for a power function  
np.set\_printoptions(suppress=True)  
x = np.linspace(0.01, 10, 100) # Independent variable  
# print(x)  
y = 2 \* x\*\*4 # Dependent variable (power function)  
# print(y)  
# Transform the data using logarithms  
log\_x = np.log(x)  
log\_y = np.log(y)  
  
# Perform linear fit on the transformed data  
degree = 1 # Linear fit  
coeffs = np.polyfit(log\_x, log\_y, degree)  
  
# Retrieve coefficients for the power function  
a = np.exp(coeffs[1]) # Coefficient for x^0  
b = coeffs[0] # Coefficient for x^1  
  
# Print the coefficients  
print("a =", a)  
print("b =", b)

import approx  
from io\_helper import (  
 read\_data\_from\_console,  
 read\_data\_from\_file,  
 output,  
 show\_graph  
)  
  
approxes = [  
 ('Линейная функция', approx.approx\_lin),  
 ('Квадратичная функция', approx.approx\_quad),  
 ('Кубическая функция', approx.approx\_cube),  
 ('Экспоненциальная функция', approx.approx\_exp),  
 ('Логарифмическая функция', approx.approx\_log),  
 ('Степенная функция', approx.approx\_pow),  
]  
  
read\_mode = input('Ввод из файла (\_): ').strip()  
read = ((lambda: read\_data\_from\_file(read\_mode))  
 if read\_mode  
 else read\_data\_from\_console)  
xs, ys = read()  
  
results = [a[1](xs, ys) for a in approxes]  
index\_min = min(range(len(results)), key=results.\_\_getitem\_\_)  
  
write\_mode = input('Вывод в файл (\_): ').strip()  
file = open(write\_mode, 'w') if write\_mode else None  
  
out = lambda x: output(x, file)  
  
for i, result in enumerate(results):  
 name = approxes[i][0]  
 out(f'--- {name}')  
 out(f'φ(x) = {result.function\_str}')  
 out(f'S = {result.dispersion:.3f}')  
 out(f'δ = {result.deviation:.3f}')  
 out(f'R^2 = {result.confidence:.3f}')  
 if result.pearson:  
 out(f'r = {result.pearson:.3f}')  
  
out(f'Лучше всего аппроксимирует {approxes[index\_min][0]}: '  
 f'δ = {results[index\_min].deviation:.3f}')  
  
show\_graph(xs, ys, results)

import numpy as np  
import matplotlib.pyplot as plt  
  
  
def read\_data\_from\_lines(lines):  
 xs = list(map(float, lines[0].split()))  
 ys = list(map(float, lines[1].split()))  
 return xs, ys  
  
  
def read\_data\_from\_console():  
 lines = [input(), input()]  
 return read\_data\_from\_lines(lines)  
  
  
def read\_data\_from\_file(file\_name):  
 with open(file\_name, 'r') as file:  
 lines = list(map(lambda x: x.rstrip('\n'), file.readlines()))  
 return read\_data\_from\_lines(lines)  
  
  
def output(s, file=None):  
 if file:  
 file.write(str(s) + '\n')  
 else:  
 print(s)  
  
  
OFFSET = 5  
  
  
def show\_graph(xs, ys, results):  
 x1, x2, y1, y2 = min(xs), max(xs), min(ys), max(ys)  
 bx, by = max(abs(x1), abs(x2)) + OFFSET, max(abs(y1), abs(y2)) + OFFSET  
 x = np.linspace(min(xs) - OFFSET, max(xs) + OFFSET, 100)  
  
 fig = plt.figure()  
 ax = fig.add\_subplot(1, 1, 1)  
  
 plt.grid(True)  
 plt.xlim((-bx, bx))  
 plt.ylim((-by, by))  
  
 ax.spines['left'].set\_position('center')  
 ax.spines['bottom'].set\_position('center')  
 ax.spines['right'].set\_color('none')  
 ax.spines['top'].set\_color('none')  
  
 for result in results:  
 xt = x  
 y = np.vectorize(result.function)  
 try:  
 y(x)  
 except ValueError:  
 xt = x[x > 0]  
 finally:  
 ax.plot(xt, y(xt), label=result.function\_str)  
  
 ax.plot(xs, ys, 'ro')  
 plt.legend()  
  
 plt.show()

import math  
from typing import Optional  
  
import numpy as np  
from dataclasses import dataclass  
  
  
@dataclass  
class Result:  
 coefficients: iter  
 dispersion: float  
 deviation: float  
 confidence: float  
 function: callable  
 function\_str: str  
 pearson: Optional[float] = None  
  
 def \_\_lt\_\_(self, other):  
 return self.deviation < other.deviation  
  
 def \_\_le\_\_(self, other):  
 return self.deviation == other.deviation  
  
  
def msr(ps, ys, num):  
 return (np.sum((ps - ys) \*\* 2) / num) \*\* 0.5  
  
  
def confidence(ps, ys, num):  
 return 1 - np.sum((ys - ps) \*\* 2) / (np.sum(ps \*\* 2) - np.sum(ps) \*\* 2 / num)  
  
  
def get\_linspace(xs, ys):  
 xs = np.array(xs)  
 ys = np.array(ys)  
 return xs, ys, len(xs)  
  
  
def approx\_lin(xs, ys):  
 xs, ys, num = get\_linspace(xs, ys)  
 SX, SXX, SY, SXY = np.sum(xs), np.sum(xs \*\* 2), np.sum(ys), np.sum(xs \* ys)  
  
 a, b = np.linalg.solve(  
 np.array([[SXX, SX], [SX, num]]),  
 np.array([SXY, SY])  
 )  
  
 ps = a \* xs + b  
 S = np.sum((ps - ys) \*\* 2)  
  
 x0, y0 = np.mean(xs), np.mean(ys)  
 r = np.sum((xs - x0) \* (ys - y0)) / (np.sum((xs - x0) \*\* 2) \* np.sum((ys - y0) \*\* 2)) \*\* 0.5  
  
 return Result(  
 coefficients=(a, b),  
 dispersion=float(S),  
 deviation=msr(ps, ys, num),  
 pearson=r,  
 confidence=confidence(ps, ys, num),  
 function=lambda x: a \* x + b,  
 function\_str=f'{a:.3f}x + {b:.3f}'  
 )  
  
  
def approx\_quad(xs, ys):  
 xs, ys, num = get\_linspace(xs, ys)  
  
 pf = np.polyfit(xs, ys, 2)  
 ps = np.poly1d(pf)(xs)  
  
 S = np.sum((ps - ys) \*\* 2)  
  
 return Result(  
 coefficients=pf,  
 dispersion=float(S),  
 deviation=msr(ps, ys, num),  
 confidence=confidence(ps, ys, num),  
 function=lambda x: pf[0] \* x \*\* 2 + pf[1] \* x + pf[2],  
 function\_str=f'{pf[0]:.3f}x^2 + {pf[1]:.3f}x + {pf[2]:.3f}'  
 )  
  
  
def approx\_cube(xs, ys):  
 xs, ys, num = get\_linspace(xs, ys)  
  
 pf = np.polyfit(xs, ys, 3)  
 ps = np.poly1d(pf)(xs)  
  
 S = np.sum((ps - ys) \*\* 2)  
  
 return Result(  
 coefficients=pf,  
 dispersion=float(S),  
 deviation=msr(ps, ys, num),  
 confidence=confidence(ps, ys, num),  
 function=lambda x: pf[0] \* x \*\* 3 + pf[1] \* x \*\* 2 + pf[2] \* x + pf[3],  
 function\_str=f'{pf[0]:.3f}x^3 + {pf[1]:.3f}x^2 + {pf[2]:.3f}x + {pf[3]:.3f}'  
 )  
  
  
def approx\_pow(xs, ys):  
 log\_x = np.log(xs)  
 log\_y = np.log(ys)  
  
 # Perform linear fit on the transformed data  
 degree = 1 # Linear fit  
 coeffs = np.polyfit(log\_x, log\_y, degree)  
  
 # Retrieve coefficients for the power function  
 a = np.exp(coeffs[1]) # Coefficient for x^0  
 b = coeffs[0] # Coefficient for x^1  
 ps = a \* xs \*\* b  
 xs, ys, num = get\_linspace(xs, ys)  
  
  
 S = np.sum((ps - ys) \*\* 2)  
  
 return Result(  
 coefficients=(a, b),  
 dispersion=float(S),  
 deviation=msr(ps, ys, num),  
 confidence=confidence(ps, ys, num),  
 function=lambda x: a \* math.pow(x, b),  
 function\_str=f'{a:.3f}x^{b:.3f}'  
 )  
  
  
def approx\_exp(xs, ys):  
 xs, ys, num = get\_linspace(xs, ys)  
  
 YS = np.log(ys)  
 A, B = np.polyfit(xs, YS, 1)[:]  
 a, b = math.exp(A), B  
 ps = a \* np.exp(b \* xs)  
  
 S = np.sum((ps - ys) \*\* 2)  
  
 return Result(  
 coefficients=(a, b),  
 dispersion=float(S),  
 deviation=msr(ps, ys, num),  
 confidence=confidence(ps, ys, num),  
 function=lambda x: a \* math.exp(b \* x),  
 function\_str=f'{a:.3f}e^{b:.3f}x'  
 )  
  
  
class LnException(Exception):  
 pass  
  
  
def approx\_log(xs, ys):  
 xs, ys, num = get\_linspace(xs, ys)  
 if xs[xs < 0]:  
 raise LnException('x must be positive for ln log approximation')  
  
 XS = np.log(xs)  
 A, B = np.polyfit(XS, ys, 1)[:]  
 a, b = A, B  
 ps = a \* np.log(xs) + b  
  
 S = np.sum((ps - ys) \*\* 2)  
  
 return Result(  
 coefficients=(a, b),  
 dispersion=float(S),  
 deviation=msr(ps, ys, num),  
 confidence=confidence(ps, ys, num),  
 function=lambda x: a \* math.log(x) + b,  
 function\_str=f'{a:.3f}ln(x) + {b:.3f}'  
 )



Вывод

Во время выполнения лабораторной работы я изучил работу метода наименьших квадратов для построения аппроксимирующей функции, заданной в табличном виде. Программа реализует вычисление аппроксимации для нескольких функций по одному методу. Среднеквадратичное отклонение показывает лишь то, насколько данный тип функции подходит для заданной функции. В следствии этого, выбор аппроксимирующей функции зависит лишь от входных данных.