МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
"ЛЭТИ"им. В.И.Ульянова(Ленина)
Кафедра САПР

ОТЧЁТ
к лабораторной работе №3
по дисциплине "Информационные технологии"
Тема: "Аппроксимация и преобразования функций"

Студент гр. 4353	_______Сизых П.В.
Преподаватель	_______Копец Е.Е.

Санкт-Петербург
2025

Цель работы: научиться высчитывать среднеквадратичную ошибку для наборов точек и аппроксимирующих функций; изменять коэффициенты функции для изменения MSE

Действия проделанной работы

Задание 1. Среднеквадратичная ошибка для набора точек и аппроксимирующих функций.

Для выполнения работы будет использована программа Jupiter Notebook. С помощью методических материалов были написаны программы и построены графики (рис. 1-6) функций (1-3) и отмечены наборы точек из условия задания, чтобы посмотреть, как проходят через точки графики аппроксимирующих функций. Затем была высчитана среднеквадратичная ошибка для каждого набора точек и аппроксимирующих функций.

$(1) f(x) = 3x + 5, (2; 12,258), (4; 17,24), (8; 30,151)$

$(2) f(x) = 0,25x^2 + 0,75x + 0,25, (2; 3,688), (4; 10,791), (8; 20,705)$

$(3) f(x) = 0,5x^3 + 0,25x^2 + 0,75x + 0,25, (2; 4,872), (4; 29,707), (8; 246,971), (10; 485,727), (12; 840,658)$

```python
[1]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# 1. Определение функции
def f(x):
  """Функция f(x) = 3x + 5"""
  return 3*x + 5

# 2. Заданные точки
points = [(2, 12.258), (4, 17.24), (8, 30.151)]

# 3. Вычисление значений функции для заданных x-координат
y_predicted = [f(x) for x, _ in points]

# 4. Извлечение фактических значений y (y_true) из точек
y_true = [y for _, y in points]

# 5. Расчет среднеквадратичной ошибки (MSE)
mse = mean_squared_error(y_true, y_predicted)
print(f"Среднеквадратичная ошибка (MSE): {mse:.4f}")

# 6. Создание графика
x_values = np.linspace(0, 10, 100)  # Создаем массив x-значений для графика
y_values = f(x_values)

plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, label="f(x) = 3x + 5")  # Строим график функции

# Отмечаем точки на графике
x_points = [x for x, _ in points]
plt.scatter(x_points, y_true, color='red', label="Заданные точки")

plt.xlabel("x")
plt.ylabel("y")
plt.title("График функции f(x) = 3x + 5 и заданные точки")
plt.legend()
plt.grid(True)
plt.show()
```

Рис. 1 – Программа 1

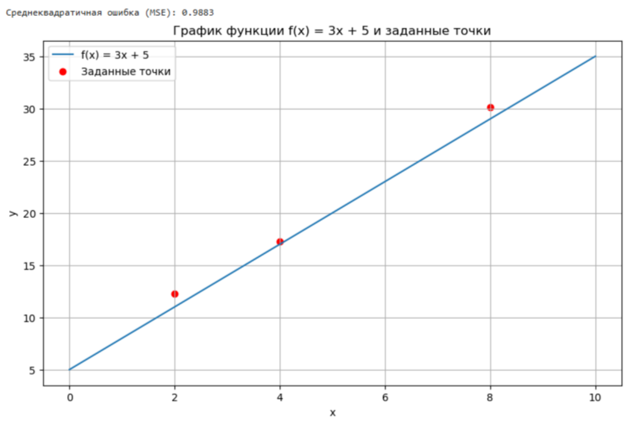


Рис. 2 – График 1

```python
[2]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# 1. Определение функции
def f(x):
  """Функция f(x) = 0.25x^2 + 0.75x + 0.25"""
  return 0.25*x**2 + 0.75*x + 0.25

# 2. Заданные точки
points = [(2, 3.688), (4, 10.791), (8, 20.705)]

# 3. Вычисление значений функции для заданных x-координат
y_predicted = [f(x) for x, _ in points]

# 4. Извлечение фактических значений y (y_true) из точек
y_true = [y for _, y in points]

# 5. Расчет среднеквадратичной ошибки (MSE)
mse = mean_squared_error(y_true, y_predicted)
print(f"Среднеквадратичная ошибка (MSE): {mse:.4f}")

# 6. Создание графика
x_values = np.linspace(0, 10, 100)  # Создаем массив x-значений для графика
y_values = f(x_values)

plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, label="f(x) = 0.25x^2 + 0.75x + 0.25")  # Строим график функции

# Отмечаем точки на графике
x_points = [x for x, _ in points]
plt.scatter(x_points, y_true, color='red', label="Заданные точки")

plt.xlabel("x")
plt.ylabel("y")
plt.title("График функции f(x) = 0.25x^2 + 0.75x + 0.25 и заданные точки")
plt.legend()
plt.grid(True)
plt.show()
```

Рис. 3 – Программа 2

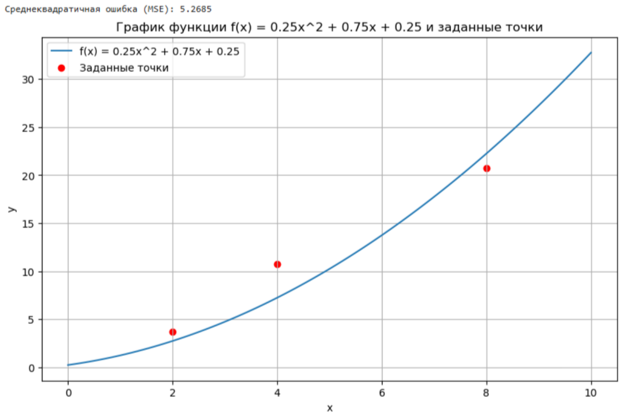


Рис. 4 – График 2

```python
[3]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# 1. Определение функции
def f(x):
  """Функция f(x) = 0.5x^3 + 0.25x^2 + 0.75x + 0.25"""
  return 0.5*x**3 + 0.25*x**2 + 0.75*x + 0.25

# 2. Заданные точки
points = [(2, 4.872), (4, 29.707), (8, 246.971), (10, 485.727), (12, 840.658)]

# 3. Вычисление значений функции для заданных x-координат
y_predicted = [f(x) for x, _ in points]

# 4. Извлечение фактических значений y (y_true) из точек
y_true = [y for _, y in points]

# 5. Расчет среднеквадратичной ошибки (MSE)
mse = mean_squared_error(y_true, y_predicted)
print(f"Среднеквадратичная ошибка (MSE): {mse:.4f}")

# 6. Создание графика
x_values = np.linspace(0, 12, 100)  # Создаем массив x-значений для графика
y_values = f(x_values)

plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, label="f(x) = 0.5x^3 + 0.25x^2 + 0.75x + 0.25")  # Строим график функции

# Отмечаем точки на графике
x_points = [x for x, _ in points]
plt.scatter(x_points, y_true, color='red', label="Заданные точки")

plt.xlabel("x")
plt.ylabel("y")
plt.title("График функции f(x) = 0.5x^3 + 0.25x^2 + 0.75x + 0.25 и заданные точки")
plt.legend()
plt.grid(True)
plt.show()
```

Рис. 5 – Программа 3

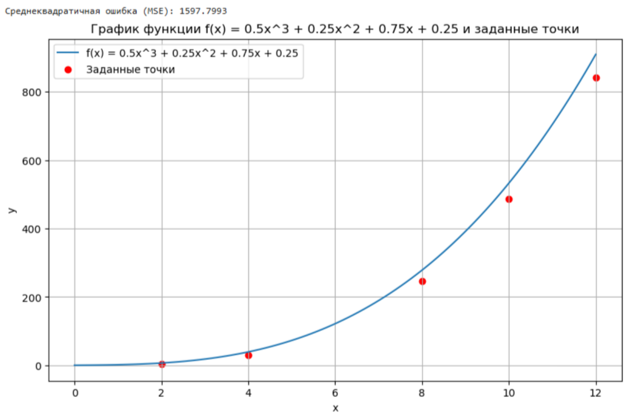


Рис. 6 – График 3

Значения среднеквадратичных ошибок:

1)MSE1 = 0.9883

2)MSE2 = 5.2685

3)MSE3 = 1597.7993

Задание 2. Получение значения MSE, заданного в задании.

2.1. Получить значение MSE меньшее 110.

С помощью методических материалов была написана программа для функции (1) и высчитана среднеквадратичная ошибка(MSE) для неё (рис. 7-8).

$(1) f(x) = 5 - 14x$

```
[1]: from sympy import *
     from sympy.plotting import plot
     init_printing(use_unicode=False, wrap_line=False, no_global=True)

[2]: import matplotlib.pyplot as plt
     import numpy as np

[3]: x = Symbol('x')

[4]: import matplotlib.pyplot as plt
     import numpy as np

     def print_points_ands_function1(sympy_function):
         def function(x_): return float(sympy_function.subs(x, x_))

         points_X = np.array([-3, -2, -1, 1, 2, 3])
         points_Y = np.array([60, 30, 19, -16, -37, -23])
         plt.xlim(-15, 15)
         plt.ylim(-40, 80)

         plt.scatter(points_X, points_Y, c='r')
         x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
         function_Y = [function(x_) for x_ in x_range]
         plt.plot(x_range, function_Y, 'b')
         plt.show()

         MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
         print(f'MSE = {MSE}')

[5]: x = Symbol('x')
     f1 = - 14 * x +5
     f1

[5]: 5 - 14x

[6]: print_points_ands_function1(f1)
```

Рис. 7 – Программа 4

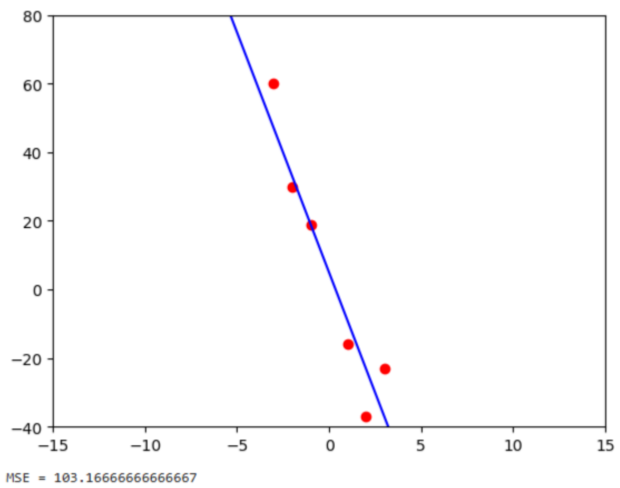
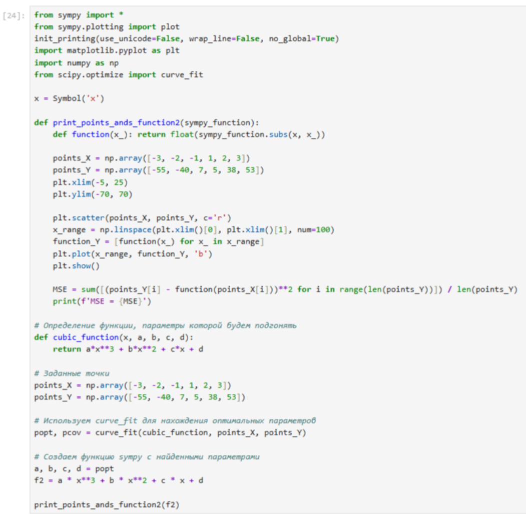


Рис. 8 – График 4

2.2. Получить значение MSE меньшее 150.

С помощью методических материалов была написана программа для функции (2) и высчитана среднеквадратичная ошибка(MSE) для неё (рис. 9-10).

$(2) f(x) = 2x^3 - 121x^2 + 2443x - 16455$



```
[24]: from sympy import *
      from sympy.plotting import plot
      init_printing(use_unicode=False, wrap_line=False, no_global=True)
      import matplotlib.pyplot as plt
      import numpy as np
      from scipy.optimize import curve_fit

      x = Symbol('x')

      def print_points_ands_function2(sympy_function):
          def function(x_): return float(sympy_function.subs(x, x_))

          points_X = np.array([-3, -2, -1, 1, 2, 3])
          points_Y = np.array([-55, -40, 7, 5, 38, 53])
          plt.xlim(-5, 25)
          plt.ylim(-70, 70)

          plt.scatter(points_X, points_Y, c='r')
          x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
          function_Y = [function(x_) for x_ in x_range]
          plt.plot(x_range, function_Y, 'b')
          plt.show()

          MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
          print(f'MSE = {MSE}')

      # Определение функции, параметры которой будем подгонять
      def cubic_function(x, a, b, c, d):
          return a*x**3 + b*x**2 + c*x + d

      # Заданные точки
      points_X = np.array([-3, -2, -1, 1, 2, 3])
      points_Y = np.array([-55, -40, 7, 5, 38, 53])

      # Используем curve_fit для нахождения оптимальных параметров
      popt, pcov = curve_fit(cubic_function, points_X, points_Y)

      # Создаем функцию sympy с найденными параметрами
      a, b, c, d = popt
      f2 = a * x**3 + b * x**2 + c * x + d

      print_points_ands_function2(f2)
```

Рис. 9 – Программа 5

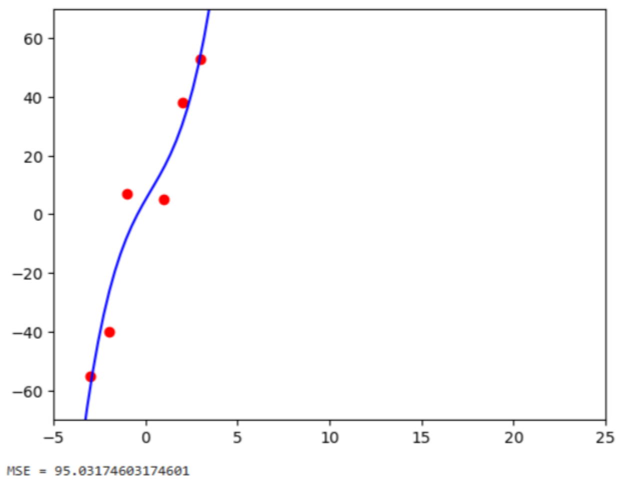


Рис. 10 – График 5

Задание 3. Получение значения MSE, заданного в задании.

3.1. Получить значение MSE меньшее 50.

Результатом кода, данного в задании, является MSE=5270.741845703125 (рис. 11-12).

```
[10]: from sympy import *
      from sympy.plotting import plot
      init_printing(use_unicode=False, wrap_line=False, no_global=True)

[11]: import matplotlib.pyplot as plt
      import numpy as np

[12]: x = Symbol('x')

[13]: def print_points_ands_function1(sympy_function):
          def function(x_): return float(sympy_function.subs(x, x_))

          points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
          points_Y = np.array([-15, -1, 4, -9, -2, -5, -8, 4, 13, 21])
          plt.xlim(-3, 6)
          plt.ylim(-20, 20)

          plt.scatter(points_X, points_Y, c='r')
          x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
          function_Y = [function(x_) for x_ in x_range]
          plt.plot(x_range, function_Y, 'b')
          plt.show()

          MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
          print(f'MSE = {MSE}')

[14]: f1 = 3.375 * x ** 3 - 9 * x**2
      f1

[14]: 3.375x^3 - 9x^2

[15]: print_points_ands_function1(f1)
```

Рис. 11 – Программа 6

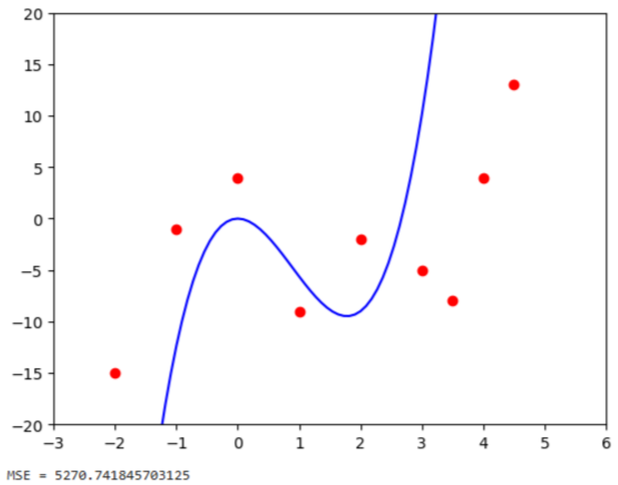


Рис. 12 – График 6

Чтобы получить MSE меньше 50 для заданных точек, нужно модифицировать функцию $f1$. В задании она задана как $f1 = 3.375x^3 - 9x^2$. Нужно изменить коэффициенты, чтобы функция лучше соответствовала заданным точкам.

Для этого будем использовать функцию scipy.optimize.curve_fit из заранее установленных пакетов scipy и scikit – learn. Итоговый код программы и её результат представлены (рис.13-14). Результат кода MSE=11.027213664391295.

```
[16]: from sympy import *
      from sympy.plotting import plot
      init_printing(use_unicode=False, wrap_line=False, no_global=True)
      import matplotlib.pyplot as plt
      import numpy as np
      from scipy.optimize import curve_fit

      x = Symbol('x')

      def print_points_ands_function1(sympy_function):
          def function(x_): return float(sympy_function.subs(x, x_))

          points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
          points_Y = np.array([-15, -1, 4, -9, -2, -5, -8, 4, 13, 21])
          plt.xlim(-3, 6)
          plt.ylim(-20, 20)

          plt.scatter(points_X, points_Y, c='r')
          x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
          function_Y = [function(x_) for x_ in x_range]
          plt.plot(x_range, function_Y, 'b')
          plt.show()

          MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
          print(f'MSE = {MSE}')

      # Определение функции, параметры которой будем подгонять
      def cubic_function(x, a, b, c, d):
          return a*x**3 + b*x**2 + c*x + d

      # Заданные точки
      points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
      points_Y = np.array([-15, -1, 4, -9, -2, -5, -8, 4, 13, 21])

      # Используем curve_fit для нахождения оптимальных параметров
      popt, pcov = curve_fit(cubic_function, points_X, points_Y)

      # Создаем функцию sympy с найденными параметрами
      a, b, c, d = popt
      f1 = a * x**3 + b * x**2 + c * x + d

      print_points_ands_function1(f1)
```

Рис. 13 – Программа 7

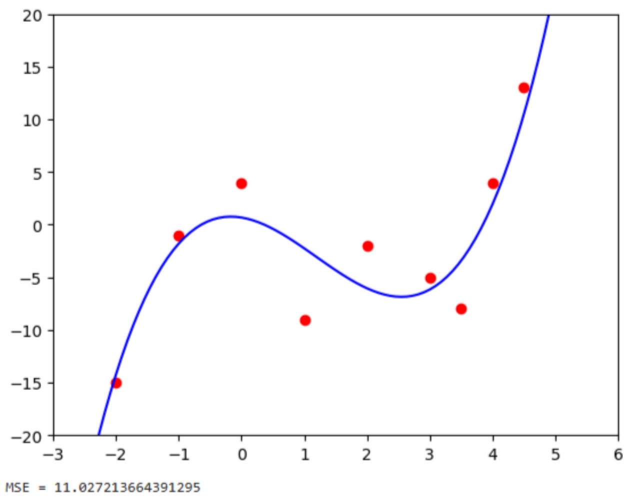


Рис. 14 – График 7

3.2. Получить значение MSE меньшее 150.

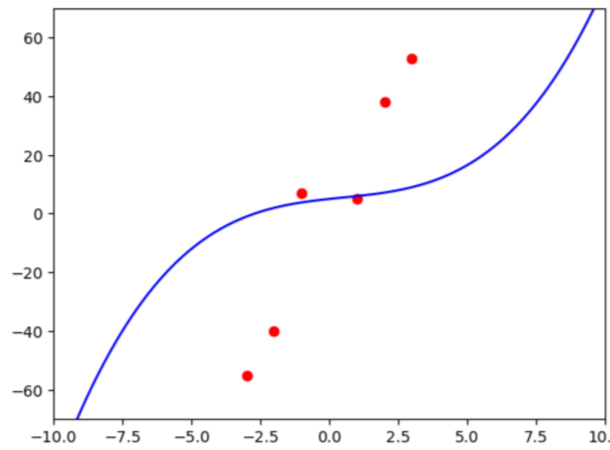
Результатом кода, данного в задании, является MSE=1262.7619146666668 (рис. 15-16).

```
[17]: from sympy import *
      from sympy.plotting import plot
      init_printing(use_unicode=False, wrap_line=False, no_global=True)

[18]: import matplotlib.pyplot as plt
      import numpy as np

[19]: x = Symbol('x')

[20]: def print_points_ands_function2(sympy_function):
          def function(x_): return float(sympy_function.subs(x, x_))

          points_X = np.array([-3, -2, -1, 1, 2, 3])
          points_Y = np.array([-55, -40, 7, 5, 38, 53])
          plt.xlim(-10, 10)
          plt.ylim(-70, 70)

          plt.scatter(points_X, points_Y, c='r')
          x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
          function_Y = [function(x_) for x_ in x_range]
          plt.plot(x_range, function_Y, 'b')
          plt.show()

          MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
          print(f'MSE = {MSE}')

[21]: f2 = 0.074 * x**3 - 0.11 * x**2 + x + 5
      f2
```

[21]: $0.074x^3 - 0.11x^2 + x + 5$

```
[22]: print_points_ands_function2(f2)
```

Рис. 15 – Программа 8




```
MSE = 1262.7619146666668
```

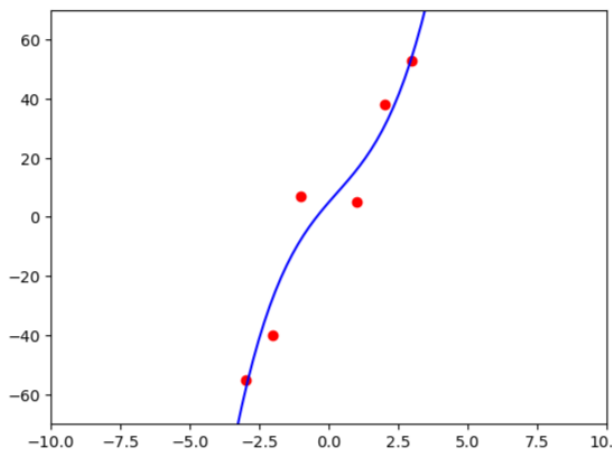
Рис. 16 – График 8

Чтобы получить MSE меньше 150 для заданных точек, нужно модифицировать функцию $f2$. В задании она задана как $f2 = 0.074x^3 - 0.11x^2 + x + 5$. Нужно изменить коэффициенты, чтобы функция лучше соответствовала заданным точкам.

Для этого будем использовать функцию scipy.optimize.curve_fit из заранее установленных пакетов scipy и scikit – learn. Итоговый код программы и её результат представлены (рис.17-18). Результат кода MSE=95.03174603174601.

```
[23]: from sympy import *
      from sympy.plotting import plot
      init_printing(use_unicode=False, wrap_line=False, no_global=True)
      import matplotlib.pyplot as plt
      import numpy as np
      from scipy.optimize import curve_fit

      x = Symbol('x')

      def print_points_ands_function2(sympy_function):
          def function(x_): return float(sympy_function.subs(x, x_))

          points_X = np.array([-3, -2, -1, 1, 2, 3])
          points_Y = np.array([-55, -40, 7, 5, 38, 53])
          plt.xlim(-10, 10)
          plt.ylim(-70, 70)

          plt.scatter(points_X, points_Y, c='r')
          x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
          function_Y = [function(x_) for x_ in x_range]
          plt.plot(x_range, function_Y, 'b')
          plt.show()

          MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
          print(f'MSE = {MSE}')

      # Определение функции, параметры которой будем подгонять
      def cubic_function(x, a, b, c, d):
          return a*x**3 + b*x**2 + c*x + d

      # Заданные точки
      points_X = np.array([-3, -2, -1, 1, 2, 3])
      points_Y = np.array([-55, -40, 7, 5, 38, 53])

      # Используем curve_fit для нахождения оптимальных параметров
      popt, pcov = curve_fit(cubic_function, points_X, points_Y)

      # Создаем функцию sympy с найденными параметрами
      a, b, c, d = popt
      f2 = a * x**3 + b * x**2 + c * x + d

      print_points_ands_function2(f2)
```

Рис. 17 – Программа 9




```
MSE = 95.03174603174601
```

Рис. 18 – График 9

Задание 4. Повторение работы с полиномами. Получение значения MSE, заданного в задании.

Блок 1

С помощью symPy были найдены полиномы, описывающие данные наборы точекЁ(1-2). Затем проведено полное исследование каждого полученного полинома: проверены чётность/нечётность, найдены нули, промежутки знакопостоянства. Построены их графики. Изменяя интервал для x, было достигнуто то, чтобы все нули отображались на графиках функций(рис. 19-22).

$(1)(-4; -4268), (-3; -1227), (-1; -17), (1; 17), (3; 1227), (4; 4268)$

$(2)(-4; -16729), (-3; -3999), (-1; 5), (1; 1), (3; 4005), (4; 16735)$

```
[27]: from sympy import *
import matplotlib.pyplot as plt
import numpy as np

# 1. [illegible]
points_X = np.array([-4, -3, -1, 1, 3, 4])
points_Y = np.array([-4268, -1227, -17, 17, 1227, 4268])

# 2. [illegible]
x = symbols('x')
a0, a1, a2, a3, a4, a5 = symbols('a0, a1, a2, a3, a4, a5')
polynomial = a0 + a1*x + a2*x**2 + a3*x**3 + a4*x**4 + a5*x**5  #[illegible]
equations = []
for i in range(len(points_X)):
    equations.append(Eq(polynomial.subs(x, points_X[i]), points_Y[i]))

# 3. [illegible]
solution = solve(equations, (a0, a1, a2, a3, a4, a5))

# 4. [illegible]
if solution:
    polynomial = polynomial.subs(solution)
    print("Найденный полином:")
    print(polynomial)
else:
    print("[illegible]")
    exit()

# 5. [illegible]

# [illegible]
is_even = all([polynomial.subs(x, i) == polynomial.subs(x, -i) for i in range(-5,6)])
is_odd = all([polynomial.subs(x, i) == -polynomial.subs(x, -i) for i in range(-5,6)])

if is_even:
    print("Функция четная")
elif is_odd:
    print("Функция нечетная")
else:
    print("[illegible]")

# [illegible]
zeros = solve(polynomial, x)
print("Нули функции:")
for zero in zeros:
    print(zero.evalf()) # evalf() [illegible]

# [illegible]
# [illegible]
critical_points = sorted([float(zero.evalf()) for zero in solve(polynomial, x) if isinstance(zero.evalf(), Number)]) #[illegible]
print("критические точки: ")
print(critical_points)

#[illegible]
intervals = []
if len(critical_points) > 0:
    intervals.append((float('-inf'), critical_points[0]))
    for i in range(len(critical_points) - 1):
        intervals.append((critical_points[i], critical_points[i+1]))
    intervals.append((critical_points[-1], float('inf')))
else:
    intervals = [(float('-inf'), float('inf'))] #[illegible]

print("Интервалы знакопостоянства:")
for interval in intervals:
  test_point = (interval[0] + interval[1]) / 2 #[illegible]
  if test_point == float('-inf') or test_point == float('inf'):
    test_point = 0  #[illegible]

  sign = polynomial.subs(x, test_point).evalf()
  if sign > 0:
    print(f"Интервал {interval}: функция положительна")
  elif sign < 0:
    print(f"Интервал {interval}: функция отрицательна")
  else:
    print(f"Интервал {interval}: функция равна нулю")

# 6. [illegible]
x_values = np.linspace(-5, 5, 400) #[illegible]

#Convert sympy expression to a function that numpy can understand.
f = lambdify(x, polynomial, modules=['numpy'])

y_values = f(x_values)

plt.figure(figsize=(12, 6))
plt.plot(x_values, y_values, label=f"y = {polynomial}")
plt.scatter(points_X, points_Y, color='red', label="Заданные точки")
plt.xlabel("x")
plt.ylabel("y")
plt.title("График полинома и заданные точки")
plt.grid(True)
plt.legend()
plt.ylim(-5000,5000) #[illegible]
plt.show()
```

Рис. 19 – Программа 10

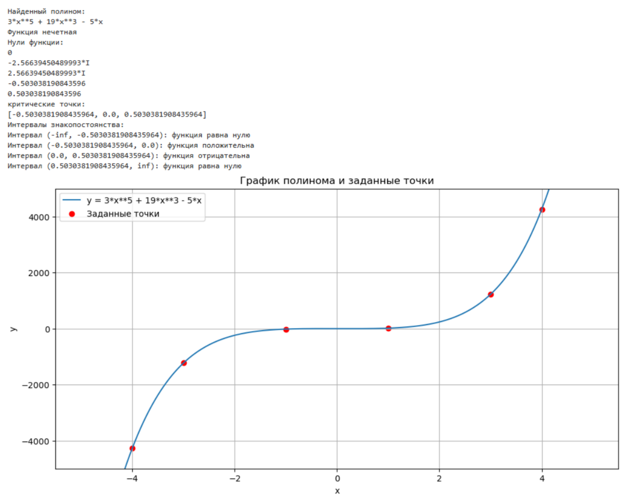


Рис. 20 – График 10

Рис. 21 – Программа 11

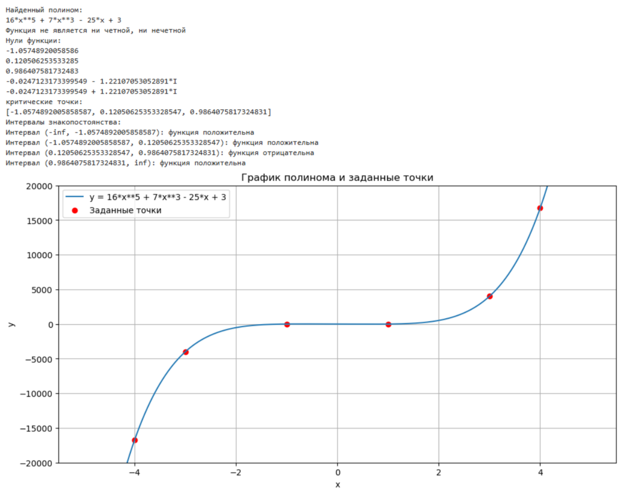


Рис. 22 – График 11

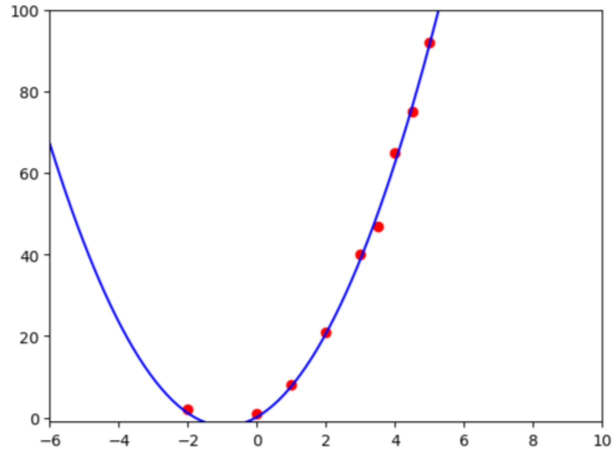
Блок 2

Задание 2.1. Получить значение MSE меньшее 5.

Результатом кода, данного в задании, является MSE=93691.3. После модификации исходной функции при помощи функций scipy.optimize.curve_fit из заранее установленных пакетов scipy и scikit – learn был получен итоговый код программы и её результат(рис. 23-24). Результат MSE=2.419459167648215.

```
[31]: from sympy import *
from sympy.plotting import plot
init_printing(use_unicode=False, wrap_line=False, no_global=True)
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

x = Symbol('x')

def print_points_ands_function1(sympy_function):
    def function(x_): return float(sympy_function.subs(x, x_))

    points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
    points_Y = np.array([2, -4, 1, 8, 21, 40, 47, 65, 75, 92])
    plt.xlim(-6, 10)
    plt.ylim(-1, 100)

    plt.scatter(points_X, points_Y, c='r')
    x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
    function_Y = [function(x_) for x_ in x_range]
    plt.plot(x_range, function_Y, 'b')
    plt.show()

    MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
    print(f'MSE = {MSE}')

# Определение функции, параметры которой будем подгонять (квадратичная)
def quadratic_function(x, a, b, c):
    return a*x**2 + b*x + c

# Заданные точки
points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
points_Y = np.array([2, -4, 1, 8, 21, 40, 47, 65, 75, 92])

# Используем curve_fit для нахождения оптимальных параметров
popt, pcov = curve_fit(quadratic_function, points_X, points_Y)

# Создаем функцию sympy с найденными параметрами
a, b, c = popt
f1 = a * x**2 + b * x + c

print_points_ands_function1(f1)
```

Рис. 23 – Программа 12




```
MSE = 2.419459167648215
```

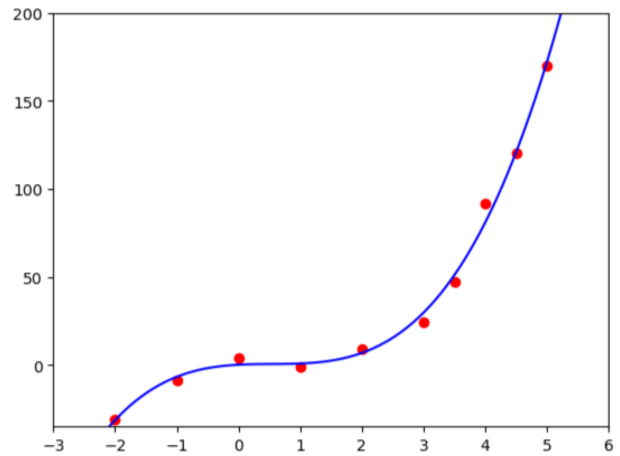
Рис. 24 – График 12

Задание 2.2. Получить значение MSE меньшее 35.

Результатом кода, данного в задании, является MSE=607200.2. После модификации исходной функции при помощи функций scipy.optimize.curve_fit из заранее установленных пакетов scipy и scikit – learn был получен итоговый код программы и её результат(рис. 25-26). Результат MSE=20.421439318416382.

```
[32]: from sympy import *
from sympy.plotting import plot
init_printing(use_unicode=False, wrap_line=False, no_global=True)
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

x = Symbol('x')

def print_points_ands_function2(sympy_function):
    def function(x_): return float(sympy_function.subs(x, x_))

    points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
    points_Y = np.array([-31, -9, 4, -1, 9, 24, 47, 92, 120, 170])
    plt.xlim(-3, 6)
    plt.ylim(-35, 200)

    plt.scatter(points_X, points_Y, c='r')
    x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
    function_Y = [function(x_) for x_ in x_range]
    plt.plot(x_range, function_Y, 'b')
    plt.show()

    MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
    print(f'MSE = {MSE}')

# Определение функции, параметры которой будем подгонять (кубическая)
def cubic_function(x, a, b, c, d):
    return a*x**3 + b*x**2 + c*x + d

# Заданные точки
points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
points_Y = np.array([-31, -9, 4, -1, 9, 24, 47, 92, 120, 170])

# Используем curve_fit для нахождения оптимальных параметров
popt, pcov = curve_fit(cubic_function, points_X, points_Y)

# Создаем функцию sympy с найденными параметрами
a, b, c, d = popt
f2 = a * x**3 + b * x**2 + c * x + d

print_points_ands_function2(f2)
```

Рис. 25 – Программа 13




```
MSE = 20.421439318416382
```

Рис. 26 – График 13

Задание 2.3. Получить значение MSE меньшее 3300.

Результатом кода, данного в задании, является MSE=1346344545, 35. После модификации исходной функции при помощи функций scipy.optimize.curve_fit из заранее установленных пакетов scipy и scikit — learn был получен итоговый код программы и её результат(рис. 27-28). Результат MSE=1836.4166902495733.

```
[33]: from sympy import *
from sympy.plotting import plot
init_printing(use_unicode=False, wrap_line=False, no_global=True)
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

x = Symbol('x')

def print_points_ands_function3(sympy_function):
    def function(x_): return float(sympy_function.subs(x, x_))

    points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
    points_Y = np.array([60, 25, 4, -0, -57, -195, -295, -540, -700, -760])
    plt.xlim(-10, 6)
    plt.ylim(-850, 100)

    plt.scatter(points_X, points_Y, c='r')
    x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
    function_Y = [function(x_) for x_ in x_range]
    plt.plot(x_range, function_Y, 'b')
    plt.show()

    MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
    print(f'MSE = {MSE}')

# Определение функции, параметры которой будем подгонять (кубическая)
def cubic_function(x, a, b, c, d):
    return a*x**3 + b*x**2 + c*x + d

# Заданные точки
points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
points_Y = np.array([60, 25, 4, -0, -57, -195, -295, -540, -700, -760])

# Используем curve_fit для нахождения оптимальных параметров
popt, pcov = curve_fit(cubic_function, points_X, points_Y)

# Создаем функцию sympy с найденными параметрами
a, b, c, d = popt
f3 = a * x**3 + b * x**2 + c * x + d

print_points_ands_function3(f3)
```

Рис. 27 – Программа 14

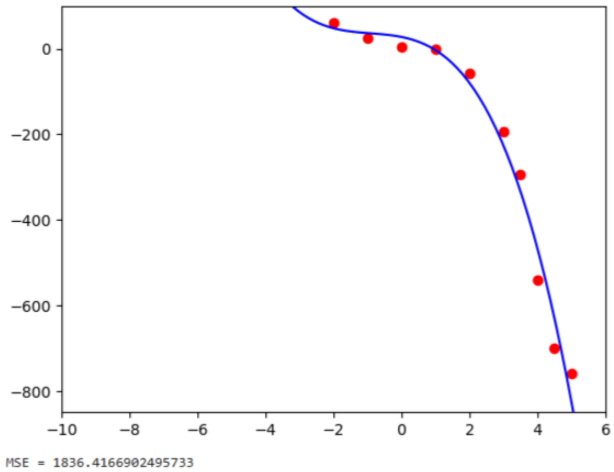


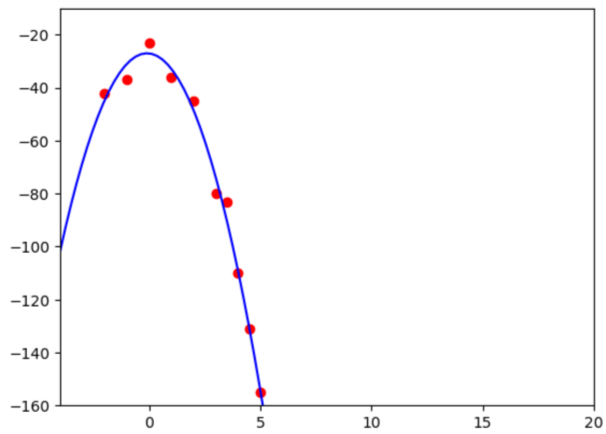
Рис. 28 – График 14

Задание 2.4. Получить значение MSE меньшее 25.

Результатом кода, данного в задании, является MSE=9143, 51953125. После модификации исходной функции при помощи функций scipy.optimize.curve_fit из заранее установленных пакетов scipy и scikit — learn был получен итоговый код программы и её результат(рис. 29-30). Результат MSE=17.402842559874365.

```python
[34]: from sympy import *
from sympy.plotting import plot
init_printing(use_unicode=False, wrap_line=False, no_global=True)
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

x = Symbol('x')

def print_points_ands_function4(sympy_function):
    def function(x_): return float(sympy_function.subs(x, x_))

    points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
    points_Y = np.array([-42, -37, -23, -36, -45, -80, -83, -110, -131, -155])
    plt.xlim(-4, 20)
    plt.ylim(-160, -10)

    plt.scatter(points_X, points_Y, c='r')
    x_range = np.linspace(plt.xlim()[0], plt.xlim()[1], num=100)
    function_Y = [function(x_) for x_ in x_range]
    plt.plot(x_range, function_Y, 'b')
    plt.show()

    MSE = sum([(points_Y[i] - function(points_X[i]))**2 for i in range(len(points_Y))]) / len(points_Y)
    print(f'MSE = {MSE}')

# Определение функции, параметры которой будем подгонять (квадратичная)
def quadratic_function(x, a, b, c):
    return a*x**2 + b*x + c

# Заданные точки
points_X = np.array([-2, -1, 0, 1, 2, 3, 3.5, 4, 4.5, 5])
points_Y = np.array([-42, -37, -23, -36, -45, -80, -83, -110, -131, -155])

# Используем curve_fit для нахождения оптимальных параметров
popt, pcov = curve_fit(quadratic_function, points_X, points_Y)

# Создаем функцию sympy с найденными параметрами
a, b, c = popt
f4 = a * x**2 + b * x + c

print_points_ands_function4(f4)
```

Рис. 29 – Программа 15




```
MSE = 17.402842559874365
```

Рис. 30 – График 15

Вывод: было изучено высчитывание среднеквадратичной ошибки для наборов точек и аппроксимирующих функций; было усвоено изменение коэффициентов функции для изменения MSE.