Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Домашняя работа №3

по курсу «Теория искусственных нейронных сетей»

«Методы многомерного поиска»

Студент группы ИУ9-71Б Окутин Д. А.

Преподаватель Каганов Ю. Т.

Москва 2024

1 Цель

1. Изучение алгоритмов многомерного поиска 1-го и 2-го порядка.

2. Разработка программ реализации алгоритмов многомерного поиска 1-го и 2-го порядка.

3. Вычисление экстремумов функции.

2 Задание

Требуется найти минимум тестовой функции Розенброка:

1. Методами сопряженных градиентов (методом Флетчера-Ривза и методом Полака-Рибьера).

2. Квазиньютоновским методом (Девидона-Флетчера-Пауэлла).

3. Методом Левенберга-Марквардта.

В качестве методов одномерного поиска использовать любой из известных методов одномерного поиска.

Параметры функции Розенброка: a=220, b=3, $f_0 = 12, n = 2$;

3 Реализация

Исходный код представлен в листинге 1 - 7.

Листинг 1: Функция Розенброка

```

from matplotlib import pyplot as plt
import numpy as np
from IPython.display import HTML
plt.rc('animation', html='html5')

class Rosenbrock(object):
def __init__(self, f0=12, a=220, b=3):
    self.f0=f0
    self.a=a
    self.b=b
def func(self, x):
    return self.b*(1-x[0])**2 + self.a * (x[1]-x[0]**2)**2 + self.f0

def derivative(self, x):
```

```
        grad = np.zeros_like(x)
        grad[0] = -2*self.b *(1-x[0]) - self.a*4*x[0] * ( x[1]-x[0]**2)
        grad[1] = 2 * self.a * (x[1]-x[0]**2)
        return grad

    def hessian(self,x):
        hessian = np.zeros((2, 2))
        hessian[0, 0] = 2 - 4*self.a * x[1] + 12*self.a * x[0]**2
        hessian[0, 1] = -4*self.a * x[0]
        hessian[1, 0] = -4*self.a * x[0]
        hessian[1, 1] = 2*self.a
        return hessian

rosenbrock=Rosenbrock()

x = np.linspace(-2, 2, 400)
y = np.linspace(-1, 3, 400)
X, Y = np.meshgrid(x, y)

Z = rosenbrock.func([X, Y])

fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')

ax.set_title("                                                    ")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("f(x, y)")
ax.view_init(elev=10, azim=60)

plt.show()

print(f"
        (1;1): {rosenbrock.func([1,1])}")
```

Листинг 2: Функция для визуализации

```

from matplotlib.animation import FuncAnimation
def show_path(path):
    x1 = np.linspace(-2, 2, 400)  #
      x1
    x2 = np.linspace(-1, 3, 400)  #
      x2
    X1, X2 = np.meshgrid(x1, x2)  #
```

```
    #

    Z = rosenbrock.func(np.array([X1, X2]))

    #  3D
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')

    #
    ax.plot_surface(X1, X2, Z, cmap='viridis', alpha=0.8)  # alpha

    ax.set_title('  (3D)')
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('f(x1, x2)')

    #
    min_point = np.array([1, 1])
    ax.scatter(min_point[0], min_point[1], rosenbrock.func(min_point
), color='green', s=100, label='  (1, 1, 0)')
    ax.set_title('  (3D)')
    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('f(x1, x2)')
    ax.view_init(elev=30, azim=60)  #


    #

    path_x = path[:, 0]
    path_y = path[:, 1]
    path_z = rosenbrock.func(path.T)

    #

    line = ax.plot(path_x, path_y, path_z, 'ro-', markersize=5,
label='  ')

    #
    def init():
        """  . """
        line[0].set_data([], [])
        line[0].set_3d_properties([])
        return line

```

```python
    def update(frame):
        """
                 . """
        line[0].set_data(path[:frame + 1, 0], path[:frame + 1, 1])
        line[0].set_3d_properties(rosenbrock.func(path[:frame + 1].T).flatten())
        line[0].set_3d_properties(rosenbrock.func(path[:frame + 1].T).flatten())
        return line

    #
    num_frames = len(path)


    #
    anim = FuncAnimation(fig, update, frames=num_frames, init_func=init, blit=True, repeat=False)

    return HTML(anim.to_html5_video())
```

Листинг 3: Градиентный спуск

```python

def gradient_descent(x0, learning_rate=0.001, tol=1e-6, max_iter=10000):
x = x0
path = [x.copy()]

for k in range(max_iter):
    grad = rosenbrock.derivative(x)
    if np.linalg.norm(grad) > 100:
        grad /= 0.1 * np.linalg.norm(grad)
    x_new = x - learning_rate * grad

    if np.linalg.norm(x_new - x) < tol:
        break

    x = x_new
    path.append(x.copy())

return np.array(path), rosenbrock.func(x), k+1

#
x0 = np.array([3.0, 0.0])

#
solution, function_value, iterations = gradient_descent(x0)
```

```
print(solution[-1])
print(function_value)
print(iterations)

sols = []
step=len(solution)//50
for i in range (0,len(solution),step):
sols.append(solution[i])

show_path(np.array(sols))
```

Листинг 4: Метод Флетчера-Ривза

```

def fletcher_reeves(x0, tol=1e-6, max_iter=1000):
x = x0
path = [x.copy()]
g = rosenbrock.derivative(x)
d = -g
k = 0

while np.linalg.norm(g) > tol and k < max_iter:
    #
                                                                  (                    ,
                                                              )
    alpha = 1
    while rosenbrock.func(x + alpha * d) > rosenbrock.func(x) + 0.01
* alpha * np.dot(g, d):
        alpha *= 0.5  #                            ,


    x_new = x + alpha * d
    g_new = rosenbrock.derivative(x_new)


    beta = np.dot(g_new, g_new) / np.dot(g, g)
    d = -g_new + beta * d

    x = x_new
    g = g_new
    k += 1
    path.append(x.copy())

return np.array(path), rosenbrock.func(x), k

x0 = np.array([3.0, 0.0])
solution, function_value, iterations = fletcher_reeves(x0)
```

```
    print("                                                    :", solution[-1])
    print("                                                                        :",
function_value)
    print("                                   :", iterations)
    sols = []
    step=len(solution)//50
    for i in range (0,len(solution),step):
    sols.append(solution[i])

    show_path(np.array(sols))
```

Листинг 5: Метод Полока-Рибьера

```

    def polak_ribiere(x0, tol=1e-6, max_iter=1000):
    x = x0
    path = [x.copy()]
    g = rosenbrock.derivative(x)
    d = -g
    k = 0

    while np.linalg.norm(g) > tol and k < max_iter:
        #
                                                        (                    ,
                                                    )
        alpha = 1
        while rosenbrock.func(x + alpha * d) > rosenbrock.func(x) + 0.01
    * alpha * np.dot(g, d):
            alpha *= 0.5  #                          ,


        x_new = x + alpha * d
        g_new = rosenbrock.derivative(x_new)


        beta = np.dot(g_new, g_new-g) / np.dot(g, g)
        d = -g_new + beta * d

        x = x_new
        g = g_new
        k += 1
        path.append(x.copy())

    return np.array(path), rosenbrock.func(x), k

    x0 = np.array([3.0, 0.0])
```

```
    solution, function_value, iterations = polak_ribiere(x0)

    print("                                    :", solution[-1])
    print("                                                        :",
function_value)
    print("                          :", iterations)
    sols = []
    step=len(solution)//50
    for i in range (0,len(solution),step):
    sols.append(solution[i])

    show_path(np.array(sols))
```

Листинг 6: Метод Девидона-Флетчера-Пауэлла

```

    def dfp(x0, tol=1e-6, max_iter=1000):
    x = x0
    path = [x.copy()]
    g = rosenbrock.derivative(x)
    k = 0
    n=len(x)
    H = np.eye(n)

    while np.linalg.norm(g) > tol and k < max_iter:
        d = -H @ g
        #
                                                          (                  ,
                                                      )
        alpha = 1
        while rosenbrock.func(x + alpha * d) > rosenbrock.func(x) + 0.01
 * alpha * np.dot(g, d):
            alpha *= 0.5  #                         ,



        x_new = x + alpha * d
        g_new = rosenbrock.derivative(x_new)

        s = x_new - x
        y = g_new - g

        rho = 1.0 / (y @ s)
        H = (np.eye(n) - rho * np.outer(s, y)) @ H @ (np.eye(n) - rho *
np.outer(y, s)) + rho * np.outer(s, s)

        x = x_new
```

```
    g = g_new
    path.append(x.copy())
    k+=1

return np.array(path), rosenbrock.func(x), k

x0 = np.array([3.0, 0.0])
solution, function_value, iterations = dfp(x0)

print("                                        :", solution[-1])
print("                                                             :",
function_value)
print("                              :", iterations)

show_path(solution)
```

Листинг 7: Метод Левенберга-Марквардта

```

from numpy.linalg import inv
def levenberg_marquardt(x0, tol=1e-6, max_iter=10000, lambda_init=3)
:
    x = x0
    path = [x.copy()]
    k=0
    lambda_ = lambda_init

    for iteration in range(max_iter):
        grad = rosenbrock.derivative(x)

        if np.linalg.norm(grad) < tol:
            break

        H = rosenbrock.hessian(x)

        while True:
          H_lm = H + lambda_ * np.eye(len(x))

          delta = np.linalg.solve(H_lm, -grad)
          x_new = x + delta

          if rosenbrock.func(x_new) < rosenbrock.func(x):
              x = x_new
              lambda_ *= 0.5
              break
          else:
              lambda_ *= 2
```

```


        path.append(x.copy())
        k+=1

    return np.array(path), rosenbrock.func(x), k

x0 = np.array([3.0, 0.0])
solution, function_value, iterations = levenberg_marquardt(x0)

print(" :", solution[-1])
print(" :", function_value)
print(" :", iterations)

show_path(solution)
```

4 Результаты

Результат представлен на рисунках 1 - 6.

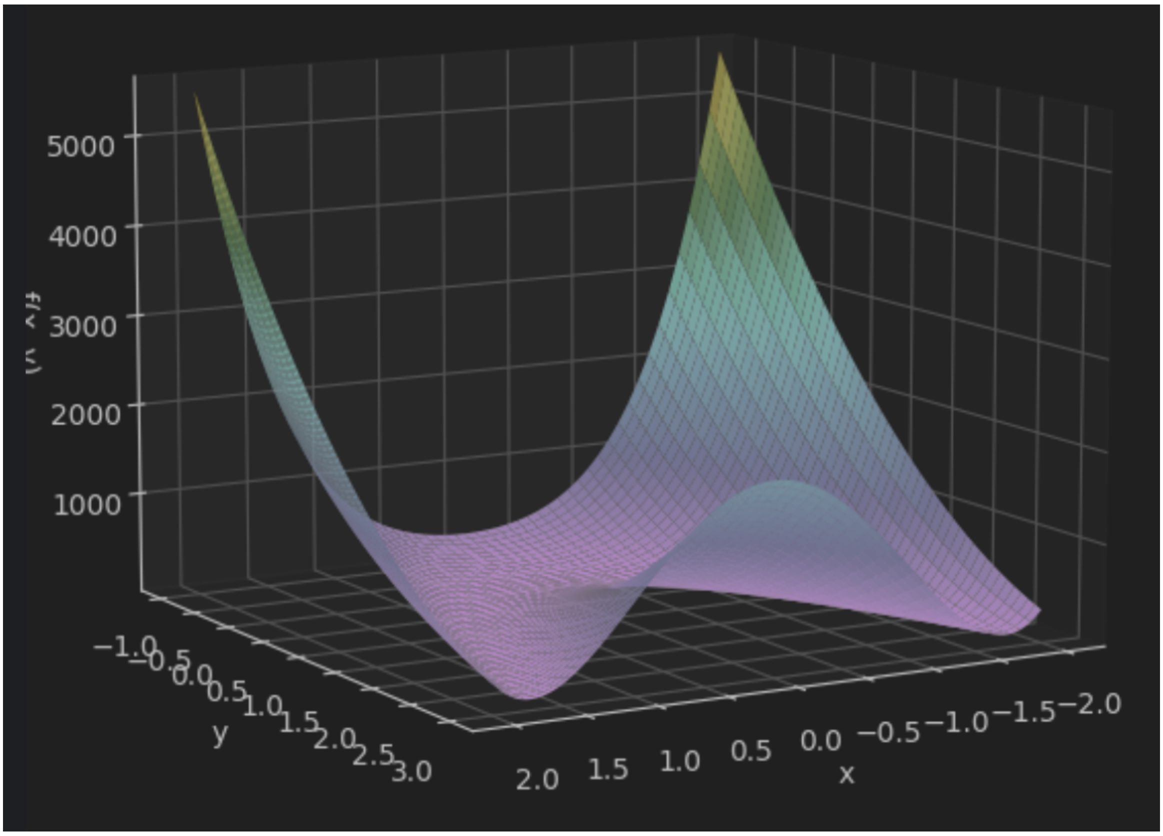

Рис. 1 — Функция Розеброка

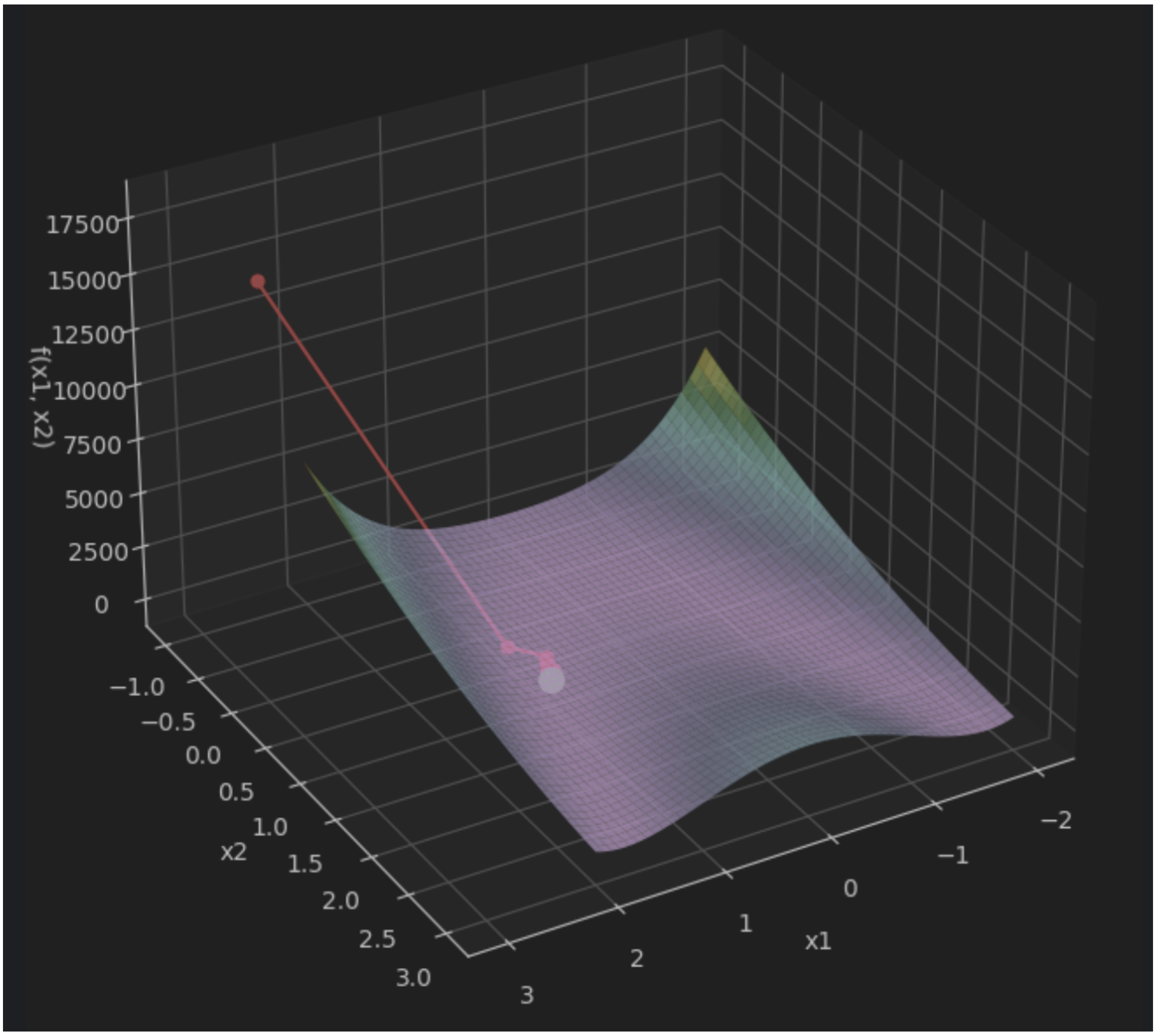

Рис. 2 — Метод градиентного спуска

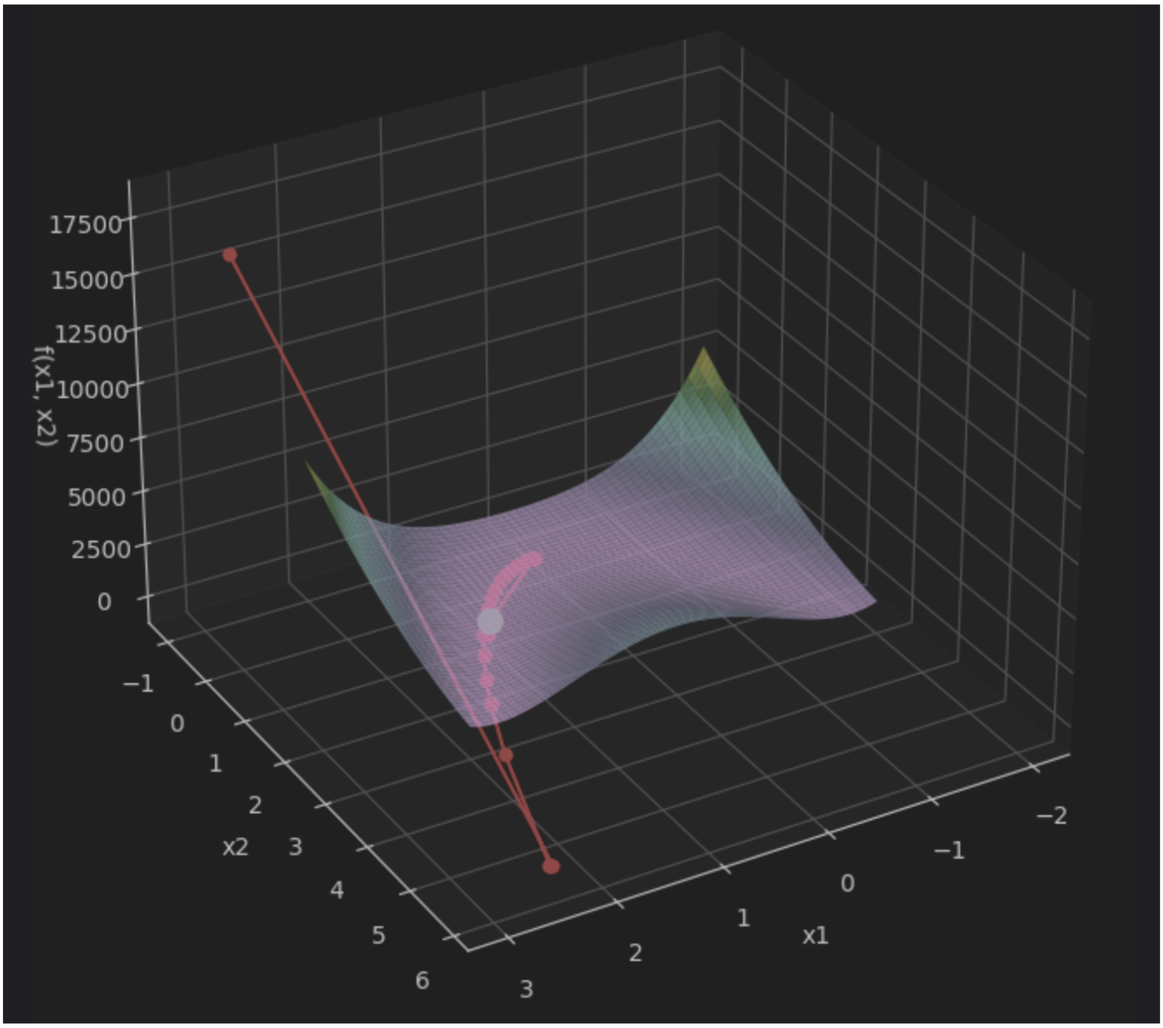

Рис. 3 — Метод Флетчера-Ривза

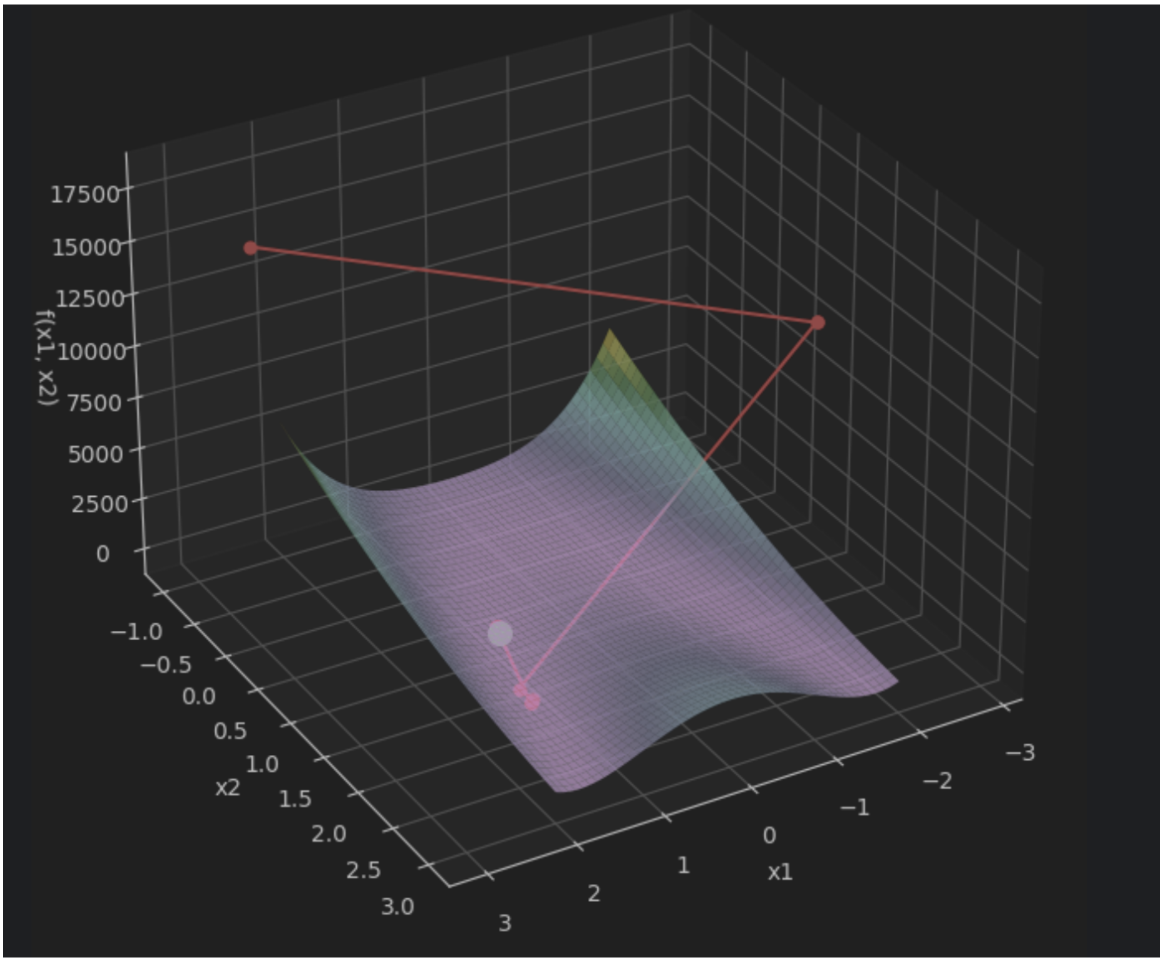

Рис. 4 — Метод Полока-Рибьера

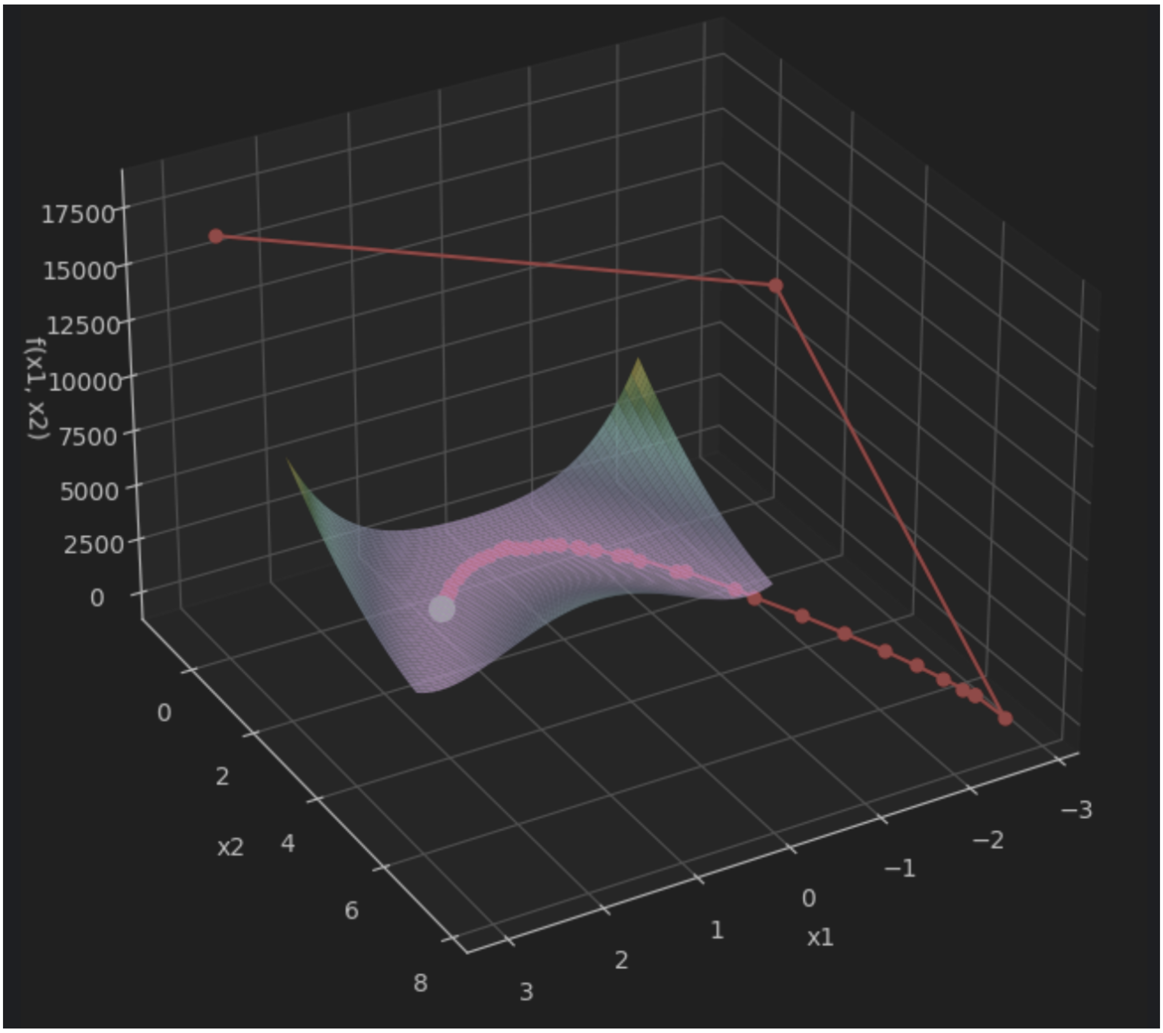

Рис. 5 — Метод Девидона-Флетчера-Пауэлла

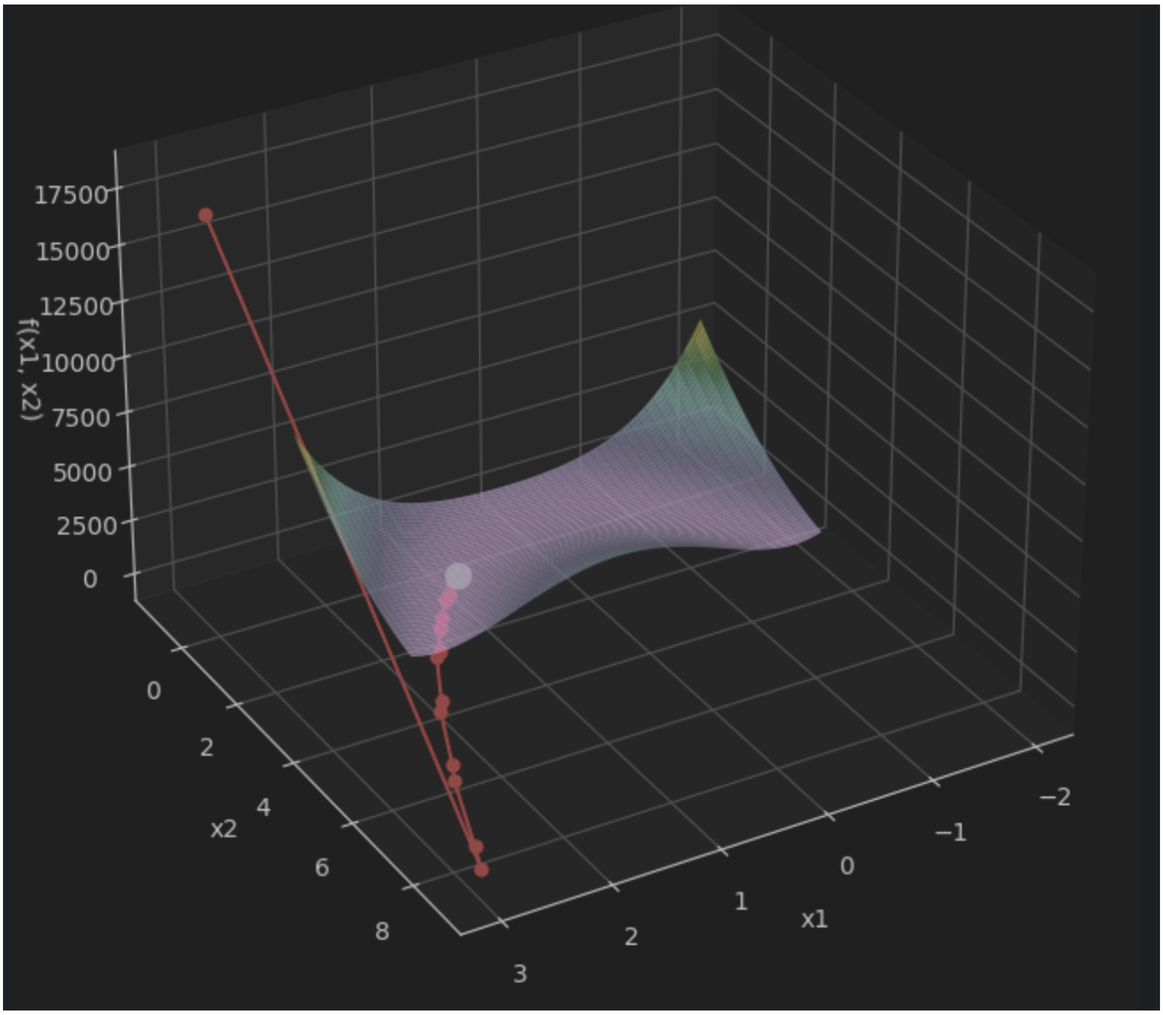

Рис. 6 — Метод Левенберга-Марквардта

5 Выводы

В результе выполнения данной лабораторной работы были реализованы алгоритмы, для поиска минимума функции. Данные алгоритмы были протестированы на функции Розенброка, полученные результаты были визуализированы в виде графиков.

Анализируя полученные результаты, видно, что оптимальнее всего точку минимума находит алгоритм Левенберга-Марквардта, это достаточно ожидаемый результат, учитывая, что методы рассматривались в порядке увеличения оптимизаций вычисления экстремума функции.