МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по лабораторной работе №2 по курсу**

**«МЕТОДЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ»**

Работу выполнил

Студенты 39 группы

Козлов Э.Д. Кличенко Д.А

Преподаватель:

Нигодин Е.А.

Краснодар

2025

**Цель работы:** изучить методы оптимизации с использованием квадратичной целевой функции и линейными ограничениями, и применить один из них.

**Ход работы:** для реализации был выбран симплекс-метод с условием Куна-Таккера

**Шаги алгоритма**

*Шаг 1*. Преобразовать ограничения исходной задачи к виду g(x) <=0

*Шаг 2.* Составить функцию Лагранжа для получения линейного уравнения

*Шаг 3.* Находим частные производные и составляем условия Куна-Таккера для получения линейных ограничений:

а) Стационарность (равенство градиента нулю)

б) Допустимость

в) Дополняющая нежёсткость

*Шаг 4.* Преобразуем задачу в задачу линейного программирования

*Шаг 5.* Решаем задачу линейного программирования симплекс-методом с учетом условий дополняющей нежесткости

а) Создаем симплекс-таблицу

б) Определяем разрешающий столбец

в) Определяем разрешающую строку

г) Пересчёт таблицы

д) Проверка на оптимальность

**Программа**

Для создания программы используется язык программирования Python. Для графической визуализации используется графический фреймворк tkinter & matplotlib. В приложение первой лабораторной был добавлен симплекс-метод.

В созданной программе одно главное активное окно.

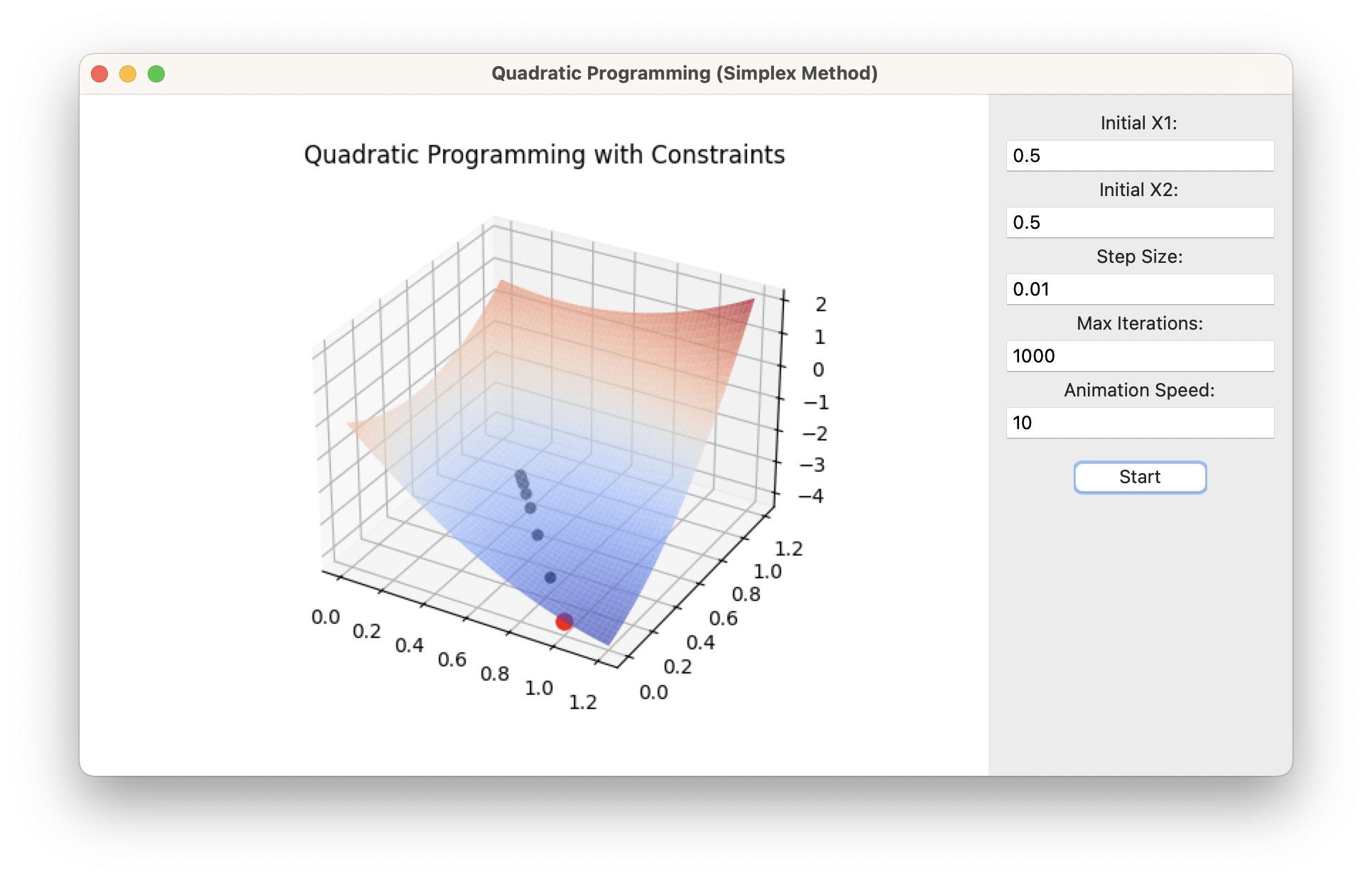


Рисунок 1 – Главное окно программы.

В данном окне можно задать параметры для алгоритма и указать скорость анимации для большей наглядности и представления работы алгоритма.

На рисунке 2 показана «Панель управления»

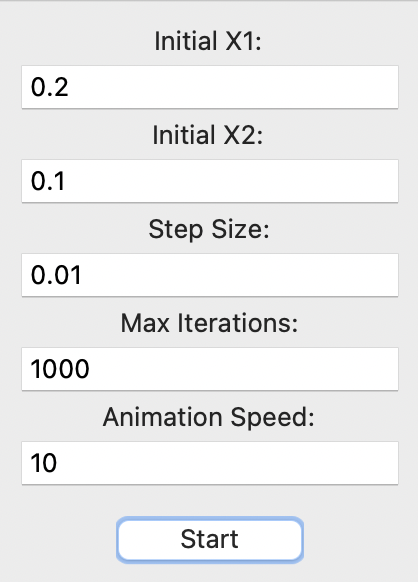


Рисунок 2 – Панель управления.

Для запуска алгоритма необходимо нажать кнопку «Start», при этом на отображаемой функции можно наглядно увидеть функционирование алгоритма – искомые точки, соединенные линией, как показано на рисунке 3

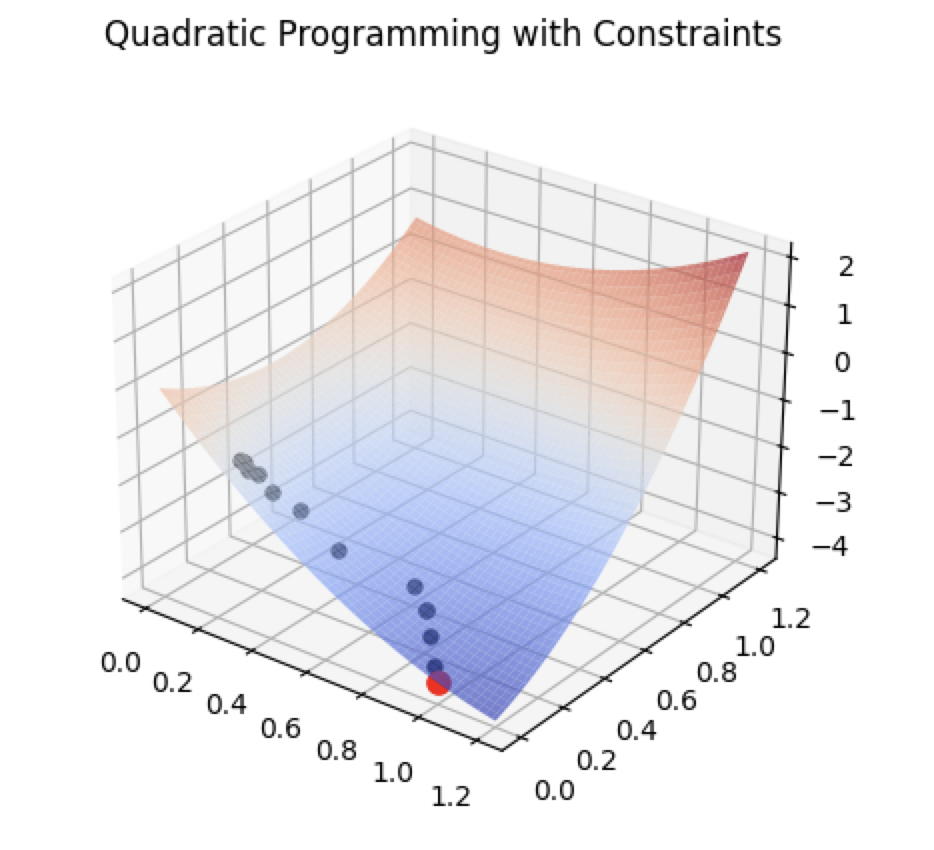


Рисунок 3 – Результат работы программы

**Вывод:** в ходе работы были изучены различные методы с использованием квадратичной целевой функции и линейными ограничениями, реализован симплекс-метод с условием Куна-Таккера.

**Листинг программы:**

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.animation as animation

import tkinter as tk

from tkinter import ttk

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

# Целевая функция: f(x1, x2) = 2x1^2 + 3x2^2 + 4x1x2 - 6x1 - 3x2

def quadratic\_func(x, y):

return 2 \* x\*\*2 + 3 \* y\*\*2 + 4 \* x \* y - 6 \* x - 3 \* y

# Ограничения

def satisfies\_constraints(x, y):

return x + y <= 1 and 2 \* x + 3 \* y <= 4 and x >= 0 and y >= 0

def project(x, y):

x, y = max(0, x), max(0, y)

if x + y > 1:

total = x + y

x, y = x / total, y / total

if 2 \* x + 3 \* y > 4:

scale = 4 / (2 \* x + 3 \* y)

x, y = x \* scale, y \* scale

return x, y

# Симплекс-метод (Nelder-Mead) с учётом ограничений

def constrained\_nelder\_mead(func, x0, y0, step\_size=0.1, max\_iter=100, tol=1e-6):

simplex = np.array([

[x0, y0],

[x0 + step\_size, y0],

[x0, y0 + step\_size]

])

simplex = np.array([project(x, y) for x, y in simplex])

values = [func(x, y) for x, y in simplex]

history = []

for \_ in range(max\_iter):

indices = np.argsort(values)

simplex = simplex[indices]

values = [values[i] for i in indices]

best = simplex[0]

worst = simplex[-1]

second = simplex[-2]

history.append((best[0], best[1], func(best[0], best[1])))

centroid = np.mean(simplex[:-1], axis=0)

reflected = centroid + (centroid - worst)

reflected = project(\*reflected)

f\_reflected = func(\*reflected)

if f\_reflected < values[0]:

expanded = centroid + 2 \* (centroid - worst)

expanded = project(\*expanded)

f\_expanded = func(\*expanded)

if f\_expanded < f\_reflected:

simplex[-1] = expanded

values[-1] = f\_expanded

else:

simplex[-1] = reflected

values[-1] = f\_reflected

elif f\_reflected < values[-2]:

simplex[-1] = reflected

values[-1] = f\_reflected

else:

contracted = centroid + 0.5 \* (worst - centroid)

contracted = project(\*contracted)

f\_contracted = func(\*contracted)

if f\_contracted < values[-1]:

simplex[-1] = contracted

values[-1] = f\_contracted

else:

simplex[1:] = best + 0.5 \* (simplex[1:] - best)

simplex[1:] = [project(x, y) for x, y in simplex[1:]]

values = [func(x, y) for x, y in simplex]

if np.std(values) < tol:

break

return np.array(history)

# GUI часть

class QPSimplexApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Quadratic Programming (Simplex Method)")

main\_frame = tk.Frame(root)

main\_frame.pack(fill=tk.BOTH, expand=True)

control\_panel = tk.Frame(main\_frame, width=200)

control\_panel.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=10)

self.entries = {}

for label, default in [

("Initial X1:", "0.5"),

("Initial X2:", "0.5"),

("Step Size:", "0.1"),

("Max Iterations:", "100"),

("Animation Speed:", "100"),

]:

ttk.Label(control\_panel, text=label).pack()

entry = ttk.Entry(control\_panel)

entry.pack()

entry.insert(0, default)

self.entries[label] = entry

self.start\_button = ttk.Button(control\_panel, text="Start", command=self.start)

self.start\_button.pack(pady=10)

self.fig = plt.Figure()

self.ax = self.fig.add\_subplot(111, projection="3d")

self.canvas = FigureCanvasTkAgg(self.fig, master=main\_frame)

self.canvas.get\_tk\_widget().pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

def start(self):

x0 = float(self.entries["Initial X1:"].get())

y0 = float(self.entries["Initial X2:"].get())

step = float(self.entries["Step Size:"].get())

iters = int(self.entries["Max Iterations:"].get())

speed = int(self.entries["Animation Speed:"].get())

path = constrained\_nelder\_mead(quadratic\_func, x0, y0, step, iters)

for i, (x, y, z) in enumerate(path):

print(f"{i}: (x: {x:.4f}, y: {y:.4f}, Значение = {z:.4f})")

self.ax.clear()

x\_vals = np.linspace(0, 1.2, 100)

y\_vals = np.linspace(0, 1.2, 100)

X, Y = np.meshgrid(x\_vals, y\_vals)

Z = quadratic\_func(X, Y)

self.ax.plot\_surface(X, Y, Z, cmap="coolwarm", alpha=0.7)

self.ax.set\_title("Quadratic Programming with Constraints")

(scat,) = self.ax.plot([], [], [], "ko", markersize=5)

(red\_dot,) = self.ax.plot([], [], [], "ro", markersize=8)

def update(frame):

scatter\_data = path[:frame+1]

scat.set\_data(scatter\_data[:, 0], scatter\_data[:, 1])

scat.set\_3d\_properties(scatter\_data[:, 2])

red\_dot.set\_data([path[frame, 0]], [path[frame, 1]])

red\_dot.set\_3d\_properties([path[frame, 2]])

return scat, red\_dot

ani = animation.FuncAnimation(self.fig, update, frames=len(path), interval=speed, blit=False)

self.canvas.draw()

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = QPSimplexApp(root)

root.mainloop()