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 №1 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 безусловной поисковой оптимизации с использованием производных, и применить один из них.

**Ход работы:** для реализации был выбран алгоритм градиентного спуска с постоянным шагом.

**Шаги алгоритма**

*Шаг 1*. Задать *х ,* 0 < ε < 1, ε 1 > 0, ε 2 > 0, *М* –предельное число итераций. Найти градиент функции в произвольной точке .

*Шаг 2.* Положить *k* = 0.

*Шаг 3.* Вычислить *f(xk).*

*Шаг 4.* Проверить выполнение критерия окончания *|f(x\*)| <* ε1*:*

а) если критерий выполнен, то расчёт закончен и *х\* = xk*;

б) если критерий не выполнен, то перейти к шагу 5.

*Шаг 5.* Проверить выполнение неравенства *k ≥ M:*

а) если неравенство выполнено, то расчет окончен: *х\* = xk*;

б) если нет, то перейти к шагу 6.

*Шаг 6*. Задать величину шага *tk.*

*Шаг 7*. Вычислить *xk+1 = xk - tkf(xk)*.

*Шаг 8*. Проверить выполнение условия

*f*(*xk+1*) *- f*(*xk*) *< 0* (или *|f*(*xk+1*) *- f*(*xk*) *|<*  ε *||f(xk)||2*);

а) если условие выполнено, то перейти к шагу 9;

б) если условие не выполнено, положить  иперейти к шагу 7.  
*Шаг 9*. Проверить выполнение условий

*||xk+1 - xk|| <* ε2*, ||f(xk+1) - f(xk)|| <* ε2:

а) если оба условия выполнены при текущем значении *k* и *k = k -*1, то расчет окончен и *x\* = xk+1*;

б) если хотя бы одно из условий не выполнено, положить *k = k* +1 и перейти к шагу 3.

**Программа**

Для создания программы используется язык программирования Python. Для графической визуализации используется графический фреймворк tkinter & matplotlib

В созданной программе одно главное активное окно.

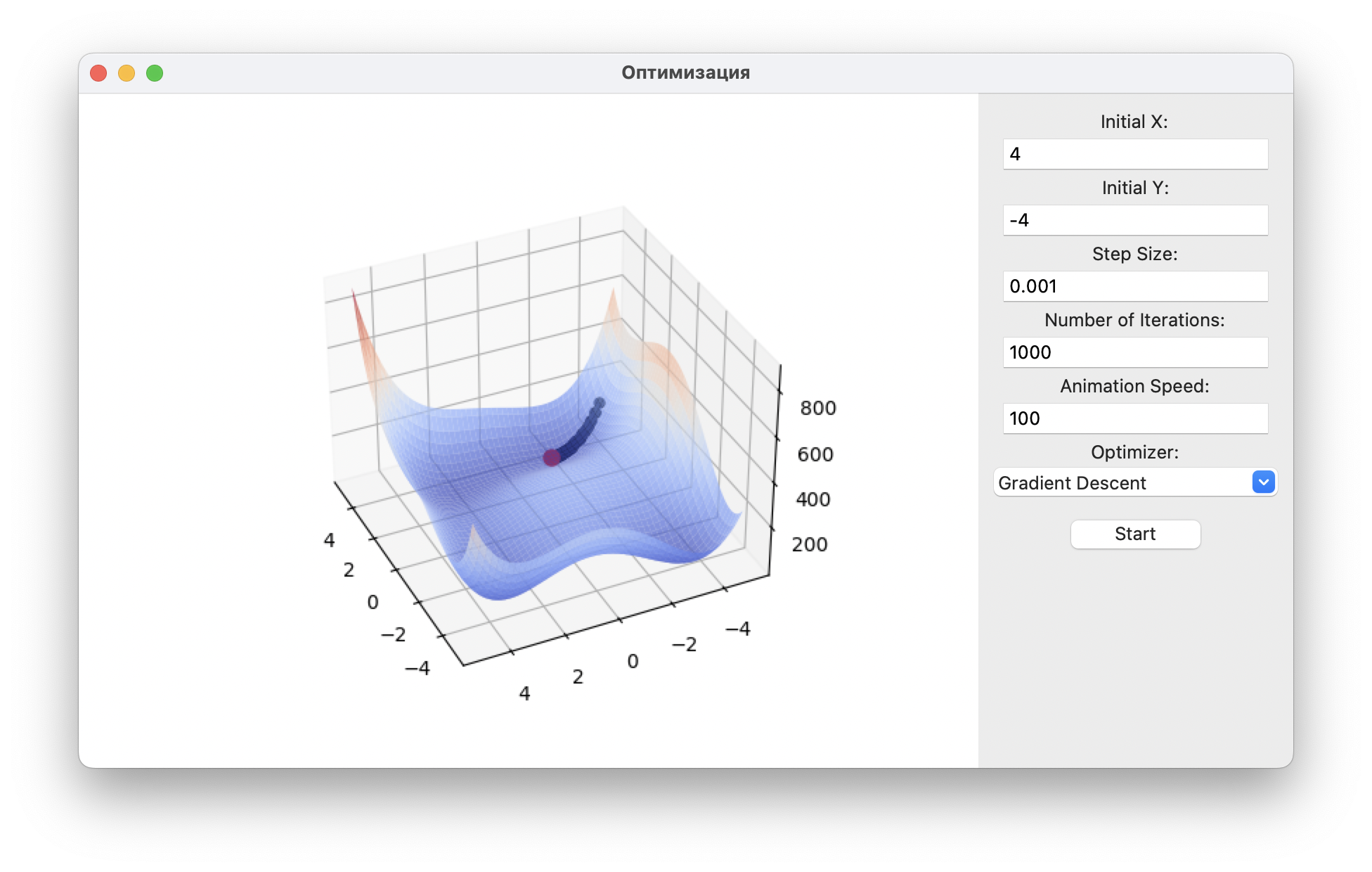


Рисунок 1 – Главное окно программы.

В данном окне можно, задать начальный шаг алгоритма, начальное приближение, число итераций (шагов) алгоритма, необходимую точность и задержку между этими итерациями в секундах (используется для наглядного представления работы алгоритма).

Также, в интерфейсе программы есть панель управления, в которой можно выбрать необходимый алгоритм с функцией и задать параметры:

1. Шаг
2. Максимальное количество итераций
3. Точность
4. Скорость анимации
5. Начальные (x;y)

На рисунке 2 показана «Панель управления»

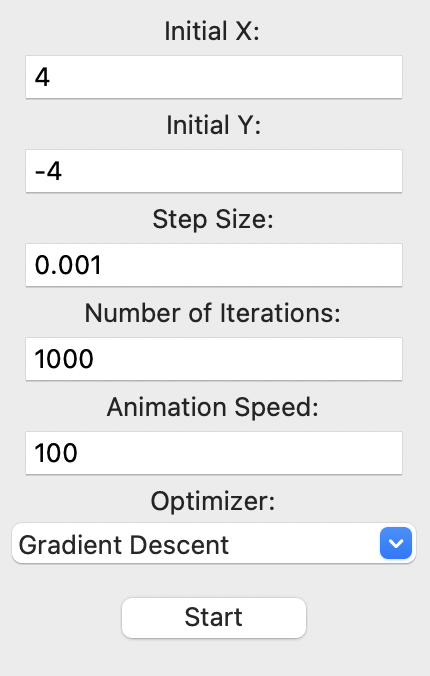


Рисунок 2 – Панель управления.

Для выбора задания используется выпадающий список с номером лабораторной работы. Более высокие значения функции показаны светлым тоном, а самые низкие значения – более темным, как показано на рисунке 3

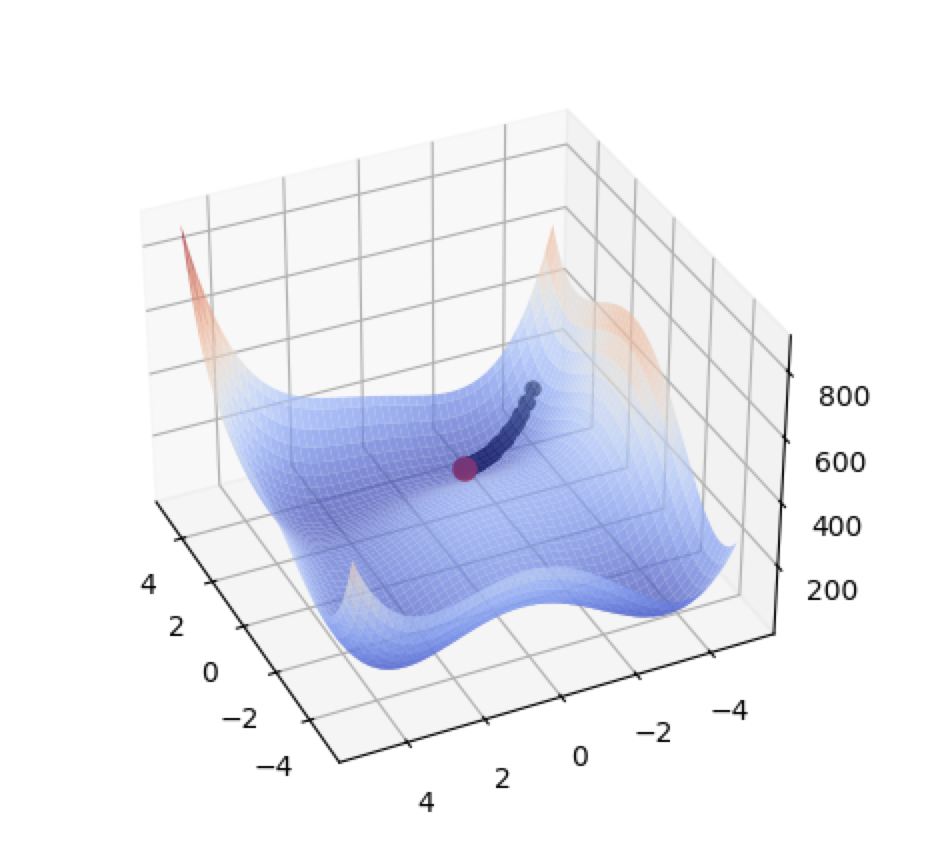


Рисунок 3 – Функция Химмельблау.

Для запуска алгоритма необходимо нажать кнопку «Start», при этом на отображаемой функции можно наглядно увидеть функционирование алгоритма – искомые точки, соединенные линией, отображенные в консоли, как показано на рисунке 4.

Изображение выглядит как текст, Шрифт, снимок экрана, черно-белый

Автоматически созданное описание

Рисунок 4 – Результат работы программы

**Вывод:** в ходе работы были изучены различные методы безусловной поисковой оптимизации с использованием производных, реализован метод градиентного спуска с постоянным шагом.

**Листинг программы:**

from model.himmelblau import himmelblau  
from model.gradient\_descent import GradientDescent, gradient  
from model.quadratic\_task import quadratic\_function, quadratic\_gradient  
from model.quadratic\_optimizer import QuadraticOptimizer  
  
class OptimizerController:  
 def \_\_init\_\_(self):  
 self.optimizers = {  
 "Gradient Descent": lambda step, it: GradientDescent(himmelblau, gradient, step, it),  
 "Quadratic Task": lambda step, it: QuadraticOptimizer(quadratic\_function, quadratic\_gradient, step, it)  
 }  
  
 def get\_optimizer(self, name, step\_size, iterations):  
 return self.optimizers[name](step\_size, iterations)  
  
 def get\_available\_optimizers(self):  
 return list(self.optimizers.keys())

import numpy as np  
from model.base\_optimizer import BaseOptimizer  
  
class GradientDescent(BaseOptimizer):  
 def optimize(self, x0, y0):  
 path = [(x0, y0, self.func(x0, y0))]  
 for \_ in range(self.iterations):  
 grad = self.grad\_func(x0, y0)  
 x0 -= self.step\_size \* grad[0]  
 y0 -= self.step\_size \* grad[1]  
 z = self.func(x0, y0)  
 print(f"{\_}: ({x0:.4f}, {y0:.4f}, {z:.4f})")  
 path.append((x0, y0, z))  
  
 path.append((x0, y0, self.func(x0, y0)))  
 return np.array(path)  
  
def gradient(x, y):  
 df\_dx = 4 \* x \* (x\*\*2 + y - 11) + 2 \* (x + y\*\*2 - 7)  
 df\_dy = 2 \* (x\*\*2 + y - 11) + 4 \* y \* (x + y\*\*2 - 7)  
 return np.array([df\_dx, df\_dy])

def himmelblau(x, y):  
 return (x\*\*2 + y - 11) \*\* 2 + (x + y\*\*2 - 7) \*\* 2

import tkinter as tk  
from tkinter import ttk  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
from matplotlib import animation  
from mpl\_toolkits.mplot3d import Axes3D  
  
from controller.optimizer\_controller import OptimizerController  
from model.himmelblau import himmelblau  
from model.quadratic\_task import quadratic\_function  
  
class OptimizationApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Оптимизация")  
  
 self.controller = OptimizerController()  
  
 main\_frame = tk.Frame(root)  
 main\_frame.pack(fill=tk.BOTH, expand=True)  
  
 control\_panel = tk.Frame(main\_frame)  
 control\_panel.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=10)  
  
 self.entries = {}  
 for label, default in [  
 ("Initial X:", "0"),  
 ("Initial Y:", "0"),  
 ("Step Size:", "0.01"),  
 ("Number of 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  
  
 ttk.Label(control\_panel, text="Optimizer:").pack()  
 self.optimizer\_choice = ttk.Combobox(  
 control\_panel, values=self.controller.get\_available\_optimizers()  
 )  
 self.optimizer\_choice.set("Gradient Descent")  
 self.optimizer\_choice.pack()  
  
 self.start\_button = ttk.Button(  
 control\_panel, text="Start", command=self.start\_optimization  
 )  
 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)  
  
 self.all\_points = None  
 self.timer\_speed = 100  
  
 def start\_optimization(self):  
 x0 = float(self.entries["Initial X:"].get())  
 y0 = float(self.entries["Initial Y:"].get())  
 step = float(self.entries["Step Size:"].get())  
 iterations = int(self.entries["Number of Iterations:"].get())  
 self.timer\_speed = int(self.entries["Animation Speed:"].get())  
  
 optimizer\_name = self.optimizer\_choice.get()  
 optimizer = self.controller.get\_optimizer(optimizer\_name, step, iterations)  
  
 self.all\_points = optimizer.optimize(x0, y0)  
 self.animate\_path()  
  
 def animate\_path(self):  
 self.ax.clear()  
 if self.optimizer\_choice.get() == "Gradient Descent":  
 x\_vals = np.linspace(-5, 5, 100)  
 y\_vals = np.linspace(-5, 5, 100)  
 Z\_func = himmelblau  
 else:  
 x\_vals = np.linspace(0, 1, 100)  
 y\_vals = np.linspace(0, 1, 100)  
 Z\_func = quadratic\_function  
  
 X, Y = np.meshgrid(x\_vals, y\_vals)  
 Z = Z\_func(X, Y)  
  
 self.ax.plot\_surface(X, Y, Z, cmap="coolwarm", alpha=0.7)  
 self.ax.set\_title("")  
  
 path = self.all\_points  
 (scat,) = self.ax.plot([], [], [], "ko", markersize=5)  
 (red\_dot,) = self.ax.plot([], [], [], "ro", markersize=8)  
  
 def update(frame):  
 if path.shape[1] == 3:  
 *# Path already contains x, y, z* scat.set\_data(path[: frame + 1, 0], path[: frame + 1, 1])  
 scat.set\_3d\_properties(path[: frame + 1, 2])  
 red\_dot.set\_data([path[frame, 0]], [path[frame, 1]])  
 red\_dot.set\_3d\_properties([path[frame, 2]])  
 else:  
 *# Path contains only x, y, need to compute z* x\_vals = path[: frame + 1, 0]  
 y\_vals = path[: frame + 1, 1]  
 z\_vals = np.array([Z\_func(x, y) for x, y in zip(x\_vals, y\_vals)])  
  
 scat.set\_data(x\_vals, y\_vals)  
 scat.set\_3d\_properties(z\_vals)  
 red\_dot.set\_data([x\_vals[-1]], [y\_vals[-1]])  
 red\_dot.set\_3d\_properties([z\_vals[-1]])  
 return scat, red\_dot  
  
 ani = animation.FuncAnimation(  
 self.fig, update, frames=len(path), interval=self.timer\_speed, blit=False  
 )  
 self.canvas.draw()

from tkinter import Tk  
from view.gui import OptimizationApp  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = Tk()  
 app = OptimizationApp(root)  
 root.mainloop()