**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»

Лабораторная работа № 7

по курсу « Методы оптимизации »

**МНОГОКРИТЕРИАЛЬНЫЙ МЕТОД**

Студент: Яровикова А. С.

Группа: ИУ9-81Б

Преподаватель: Посевин Д. П.

Москва, 2024

СОДЕРЖАНИЕ

[ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ 3](#_Toc158799284)

[ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ 3](#_Toc158799285)

[РЕЗУЛЬТАТЫ 7](#_Toc158799286)

[ВЫВОДЫ 8](#_Toc158799287)

# **ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ**

**Цель работы:**

Реализовать поиск экстремума многокритериальным методом.

**Постановка задачи:**

1. Исследовать алгоритм многокритериальн методом.
2. Реализовать численные методы оптимизации на языке программирования Julia, включая функции для вычисления градиента и гессиана, а также методы поиска оптимального шага.
3. Применить алгоритмы оптимизации к заданной функции многих переменных с целью минимизации значения функционала, используя различные начальные условия и параметры методов по вариантам

Вариант 1. .

# **ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ**

Вспомогательные функции:

**import** Pkg

Pkg.add("Plots")

**using** Plots

Pkg.add("LinearAlgebra")

**using** LinearAlgebra

**function df**(f, x::Vector{**Float64**}, i, h=**1e-5**)

x\_plus = copy(x)

x\_minus = copy(x)

x\_plus[i] += h

x\_minus[i] -= h

df\_dx = (f(x\_plus) - f(x\_minus)) / (**2**h)

**return** df\_dx

**end**

**function gradient**(f, x::Vector{**Float64**})

grad = []

**for** i **in** **1**:length(x)

push!(grad, df(f, x, i))

**end**

**return** grad

**end**

xs = []

Метод Свенна для локализации минимума:

**using** LinearAlgebra

**function swann\_method**(f, x0, h=**0.1**)

first = x0

second = x0 + h

# если функция растет, меняем направление движения

**if** f(second) > f(first)

h = -h

first, second = second, second + h

**end**

last = second + h

# увеличиваем шаг движения, если функция уменьшается

**while** f(last) < f(second)

h \*= **2**

first, second, last = second, last, last + h

**end**

# перепрыгнули далеко

**if** second > last

first, second, last = last, second, first

**end**

**return** first, last

**end**

Метод Золотого сечения для поиска минимума:

**using** LinearAlgebra

**function golden\_section\_search**(f, a, b, eps=**1e-5**)

phi = (sqrt(**5**) - **1**) / **2**

x1 = b - phi \* (b - a)

x2 = a + phi \* (b - a)

**while** abs(b - a) > eps

**if** f(x1) <= f(x2)

b = x2

**else**

a = x1

**end**

x1 = b - phi \* (b - a)

x2 = a + phi \* (b - a)

**end**

**return** (a + b) / **2**

**end**

Многокритериальный метод:

**function conjugate\_gradient**(f, x0)

eps1, eps2 = **1e-6**, **1e-10**

x = x0

prev\_x = copy(x)

grad = gradient(f, x)

d = -grad

trajectory = [x]

**while** true

prev\_grad=copy(grad)

l, r = swann\_method(alpha -> f(x + alpha \* d), **1e-7**)

alpha = golden\_section\_search(alpha -> f(x + alpha \* d), l, r)

x += alpha \* d

grad = gradient(f, x)

**if** norm(x - prev\_x) < eps1 || norm(f(x) - f(prev\_x)) < eps2

**break**

**end**

beta = dot(grad, grad) / dot(prev\_grad, prev\_grad)

d = -grad + beta \* d

prev\_x = copy(x)

push!(trajectory, x)

**end**

**return** x, trajectory

**end**

x\_min, trajectory = conjugate\_gradient(f, x0)

print("Точка минимума: ", x\_min, "; Количество итераций: ", length(trajectory)-**1**)

**using** ForwardDiff

**function multicriterial\_search**(f, x0)

eps1, eps2 = **1e-6**, **1e-10**

x = x0

prev\_x = copy(x)

trajectory = [x]

n = **2**

i = **0**

**while** true

i += **1**

grad = gradient(f, x)

delta = x - prev\_x

**if** mod(i, n) == **0**

delta = Vector{**Float64**}([**0**, **0**])

**end**

g = a -> f(x - a[**1**] \* grad + a[**2**] \* delta)

a\_k, tr = conjugate\_gradient(g, prev\_x)

x, prev\_x = x - a\_k[**1**] \* grad + a\_k[**2**] \* delta, x

**if** norm(f(x) - f(prev\_x)) < eps1

**break**

**end**

prev\_x = copy(x)

push!(trajectory, x)

**end**

**return** x, trajectory

**end**

x\_min, trajectory = multicriterial\_search(f, x0)

print("Точка минимума: ", x\_min, "; Количество итераций: ", length(trajectory)-**1**)

func(x, y) = (x-**4**\*y)^**2** +(y+**5**)^**2**

f(x) = func(x[**1**],x[**2**])

x0 = [**10.0**, -**5.0**]

x = -**20**:**0.1**:**10**

y = -**20**:**0.1**:**20**

x\_min, trajectory = multicriterial\_search(f, x0)

x\_coords = [point[**1**] **for** point **in** trajectory]

y\_coords = [point[**2**] **for** point **in** trajectory]

contour(x, y, (x, y) -> f([x, y]), levels = **10**, xlabel="x", ylabel="y", colorbar=false, size=(**500**, **500**))

scatter!(x\_coords, y\_coords, markersize=**2**, markershape=:circle, markercolor=:red, label = "")

p = plot!(x\_coords, y\_coords, label="Движение", line=:red)

println("Многокритериальный поиск:**\n** точка минимума: $x\_min**\n** кол-во итераций = $(length(trajectory)-1)")

p

# **РЕЗУЛЬТАТЫ**

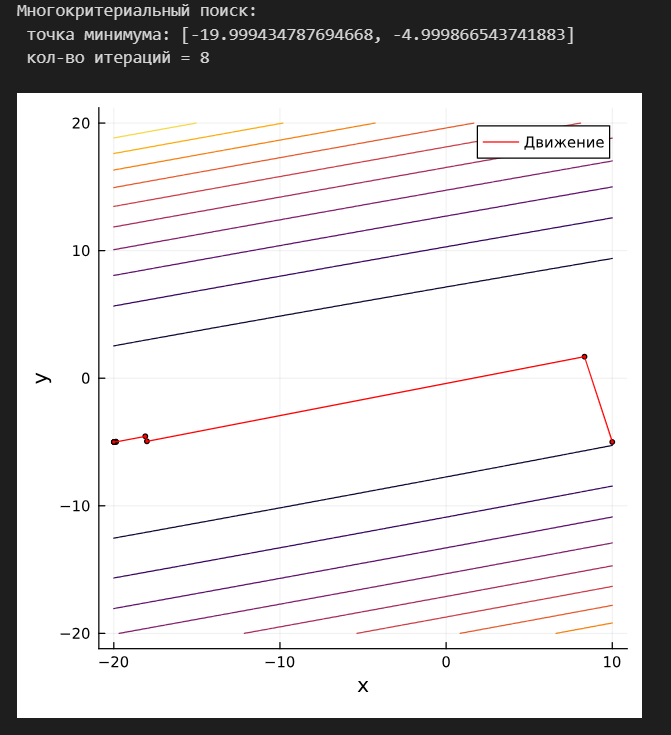


Рисунок 1 – Многокритериальный метод на функции по варианту

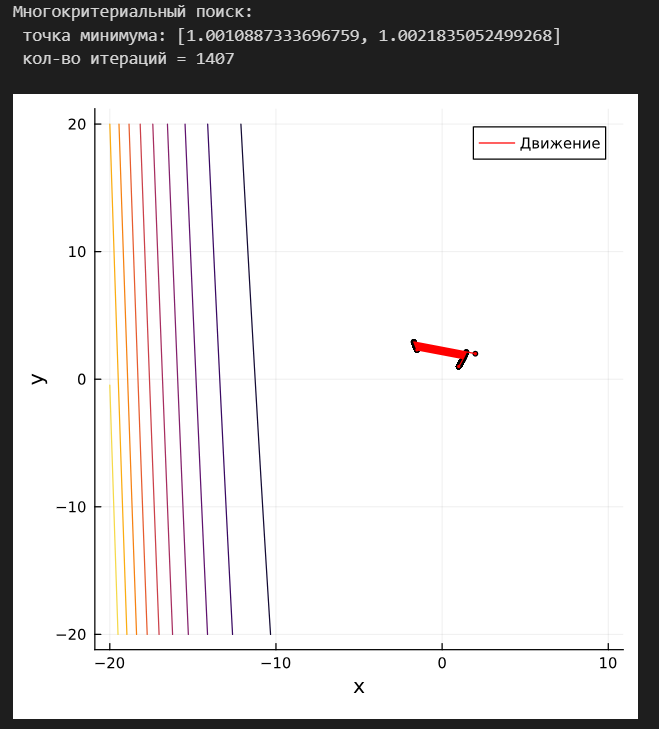


Рисунок 2 - Многокритериальный метод на функции Розенброка

# **ВЫВОДЫ**

В ходе выполнения лабораторной работы был реализован многокритериальный метод.