**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»

Летучка № 4

по курсу « Методы оптимизации »

**СРАВНЕНИЕ МЕТОДОВ ОПТИМИЗАЦИИ МНОГОМЕРНЫХ ФУНКЦИЙ – МЕТОДЫ ПИРСОНА, НЬЮТОНА-РАФСОНА, ГРИНШТАДТА, ГОЛЬДФАРБА И ФЛЕТЧЕРА**

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 11](#_Toc158799286)

[ВЫВОДЫ 15](#_Toc158799287)

# **ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ**

**Цель работы:**

Реализовать поиск экстремума функции следующими методами: метод Пирсона и его модификации, Ньютона-Рафсона, методы Гринштадта и Гольдфарба, метод Флетчера.

**Постановка задачи:**

1. Исследовать алгоритмы оптимизации, включая метод Пирсона и его модификации, Ньютона-Рафсона, методы Гринштадта и Гольдфарба, метод Флетчера, с целью нахождения локального минимума функции многих переменных.
2. Реализовать численные методы оптимизации на языке программирования Julia.
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

**using** Plots

**using** LinearAlgebra

**using** SymPy

x, y = symbols("x y")

# функция по варианту

my\_func = (x-**4**\*y)^**2** + (y+**5**)^**2**

**function find\_grad**(point)

df\_dx = diff(my\_func, x)

df\_dy = diff(my\_func, y)

**return** [df\_dx.subs([(x,point[**1**]), (y, point[**2**])]), df\_dy.subs([(x,point[**1**]), (y, point[**2**])])]

**end**

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

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

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

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**

Метод Пирсона 1:

**using** Plots

**using** LinearAlgebra

**function pirson\_1**(x\_start, eps)

x = x\_start

trajectory = [x]

gradient = find\_grad(x)

H = [**1** **0**; **0** **1**]

prev\_x = x

prev\_grad = gradient

alpha = **0.01**

**while** norm(gradient) > eps

s = H \* gradient

x\_new = [x[**1**] - alpha \* s[**1**], x[**2**] - alpha \* s[**2**]]

prev\_x = x

x = x\_new

push!(trajectory, x)

prev\_grad = gradient

gradient = find\_grad(x)

d\_x = [x[**1**] - prev\_x[**1**], x[**2**] - prev\_x[**2**]]

delta\_g = [gradient[**1**] - prev\_grad[**1**], gradient[**2**] - prev\_grad[**2**]]

N = (d\_x - H \* delta\_g) \* transpose(d\_x) / dot(transpose(d\_x), delta\_g)

H = H + N

**end**

**return** trajectory

**end**

trajectory = pirson\_1(x0, **0.000001**)

x\_min = trajectory[**end**]

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

contour\_range = -**20**:**0.1**:**20**

contour\_plot = contour(contour\_range, contour\_range, (x, y) -> (x-**4**\*y)^**2** +(y+**5**)^**2**, levels=**20**, c=:blues)

plot!([point[**1**] **for** point **in** trajectory], [point[**2**] **for** point **in** trajectory], label="Движение", c=:reds)

plot!(xlabel="x1", ylabel="x2")

Метод Пирсона 2:

**using** Plots

**using** LinearAlgebra

**function pirson\_2**(x\_start, eps)

x = x\_start

trajectory = [x]

gradient = find\_grad(x)

H = [**1** **0**; **0** **1**]

prev\_x = x

prev\_grad = gradient

**while** norm(gradient) > eps

s = H \* gradient

l, r = swann\_method(alpha -> f(x - alpha \* s), **1e-2**)

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

x\_new = [x[**1**] - alpha \* s[**1**], x[**2**] - alpha \* s[**2**]]

prev\_x = x

x = x\_new

push!(trajectory, x)

prev\_grad = gradient

gradient = find\_grad(x)

d\_x = [x[**1**] - prev\_x[**1**], x[**2**] - prev\_x[**2**]]

delta\_g = [gradient[**1**] - prev\_grad[**1**], gradient[**2**] - prev\_grad[**2**]]

N = (d\_x - H \* delta\_g) \* transpose(d\_x) / dot(transpose(d\_x), delta\_g)

H = H + N

**end**

**return** trajectory

**end**

trajectory = pirson\_2(x0, **0.001**)

x\_min = trajectory[**end**]

println("Метод Пирсона 2:**\n** точка минимума: $x\_min**\n** кол-во итераций = $(length(trajectory)-1)")

contour\_range = -**20**:**0.1**:**20**

contour\_plot = contour(contour\_range, contour\_range, (x, y) -> (x-**4**\*y)^**2** +(y+**5**)^**2**, levels=**20**, c=:blues)

plot!([point[**1**] **for** point **in** trajectory], [point[**2**] **for** point **in** trajectory], label="Движение", c=:reds)

plot!(xlabel="x1", ylabel="x2")

Метод Пирсона 3:

**using** Plots

**using** LinearAlgebra

**function pirson\_3**(x\_start, eps)

x = x\_start

trajectory = [x]

gradient = find\_grad(x)

H = [**1** **0**; **0** **1**]

prev\_x = x

prev\_grad = gradient

**while** norm(gradient) > eps

s = H \* gradient

l, r = swann\_method(alpha -> f(x - alpha \* s), **1e-2**)

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

x\_new = [x[**1**] - alpha \* s[**1**], x[**2**] - alpha \* s[**2**]]

prev\_x = x

x = x\_new

push!(trajectory, x)

prev\_grad = gradient

gradient = find\_grad(x)

d\_x = [x[**1**] - prev\_x[**1**], x[**2**] - prev\_x[**2**]]

delta\_g = [gradient[**1**] - prev\_grad[**1**], gradient[**2**] - prev\_grad[**2**]]

N = (d\_x - H \* delta\_g) \* transpose((H \* delta\_g)) / dot(transpose(delta\_g), H \* delta\_g)

H = H + N

**end**

**return** trajectory

**end**

trajectory = pirson\_3(x0, **1e-6**)

x\_min = trajectory[**end**]

println("Метод Пирсона 3:**\n** точка минимума: $x\_min**\n** кол-во итераций = $(length(trajectory)-1)")

contour\_range = -**20**:**0.1**:**20**

contour\_plot = contour(contour\_range, contour\_range, (x, y) -> (x-**4**\*y)^**2** +(y+**5**)^**2**, levels=**20**, c=:blues)

plot!([point[**1**] **for** point **in** trajectory], [point[**2**] **for** point **in** trajectory], label="Движение", c=:reds)

plot!(xlabel="x1", ylabel="x2")

Метод Ньютона-Рафсона:

**using** Plots

**using** LinearAlgebra

**function newtone\_rafson**(x\_start, eps)

x = x\_start

trajectory = [x]

gradient = find\_grad(x)

H = [**1** **0**; **0** **1**]

prev\_x = x

prev\_grad = gradient

**while** norm(gradient) > eps

s = H \* gradient

l, r = swann\_method(alpha -> f(x - alpha \* s), **1e-2**)

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

x\_new = [x[**1**] - alpha \* s[**1**], x[**2**] - alpha \* s[**2**]]

prev\_x = x

x = x\_new

push!(trajectory, x)

prev\_grad = gradient

gradient = find\_grad(x)

d\_x = [x[**1**] - prev\_x[**1**], x[**2**] - prev\_x[**2**]]

delta\_g = [gradient[**1**] - prev\_grad[**1**], gradient[**2**] - prev\_grad[**2**]]

N = (H \* delta\_g) \* transpose((H \* delta\_g)) / dot(transpose(delta\_g), H \* delta\_g)

H = H - N

**end**

**return** trajectory

**end**

trajectory = newtone\_rafson(x0, **1e-6**)

x\_min = trajectory[**end**-**1**]

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

contour\_range = -**20**:**0.1**:**20**

contour\_plot = contour(contour\_range, contour\_range, (x, y) -> (x-**4**\*y)^**2** +(y+**5**)^**2**, levels=**20**, c=:blues)

plot!([point[**1**] **for** point **in** trajectory], [point[**2**] **for** point **in** trajectory], label="Движение", c=:reds)

plot!(xlabel="x1", ylabel="x2")

Метод Гринштадта:

**using** Plots

**using** LinearAlgebra

**function greenshtadt**(x\_start, eps)

x = x\_start

trajectory = [x]

gradient = find\_grad(x)

H = [**1** **0**; **0** **1**]

prev\_x = x

prev\_grad = gradient

**while** norm(gradient) > eps

s = H \* gradient

l, r = swann\_method(alpha -> f(x - alpha \* s), **1e-2**)

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

x\_new = [x[**1**] - alpha \* s[**1**], x[**2**] - alpha \* s[**2**]]

prev\_x = x

x = x\_new

push!(trajectory, x)

prev\_grad = gradient

gradient = find\_grad(x)

d\_x = [x[**1**] - prev\_x[**1**], x[**2**] - prev\_x[**2**]]

delta\_g = [gradient[**1**] - prev\_grad[**1**], gradient[**2**] - prev\_grad[**2**]]

multiplier = **1** / dot(transpose(delta\_g), H \* delta\_g)

first\_add = d\_x\*transpose(delta\_g)\*H+H\*delta\_g\*transpose(d\_x)

second\_chisl\_multyp = transpose(delta\_g)\*d\_x - transpose(delta\_g)\*H\*delta\_g

second\_chisl\_multyp\_2 = H\*delta\_g\*transpose(delta\_g)\*H

second\_chisl\_multyp = second\_chisl\_multyp\*second\_chisl\_multyp\_2

znam\_second = dot(transpose(delta\_g),H\*delta\_g)

second\_add = second\_chisl\_multyp/znam\_second

first\_add = first\_add-second\_add

N = first\_add \* multiplier

H = H - N

**end**

**return** trajectory

**end**

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

trajectory = greenshtadt(x0, **1e-3**)

x\_min = trajectory[**end**]

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

contour\_range = -**20**:**0.1**:**20**

contour\_plot = contour(contour\_range, contour\_range, (x, y) -> (x-**4**\*y)^**2** +(y+**5**)^**2**, levels=**20**, c=:blues)

plot!([point[**1**] **for** point **in** trajectory], [point[**2**] **for** point **in** trajectory], label="Движение", c=:reds)

plot!(xlabel="x1", ylabel="x2")

Метод Гольдфарба:

**function goldfarb**(x\_start, eps)

x = x\_start

trajectory = [x]

gradient = find\_grad(x)

H = [**1** **0**; **0** **1**]

prev\_x = x

prev\_grad = gradient

**while** norm(gradient) > eps

s = H \* gradient

l, r = swann\_method(alpha -> f(x - alpha \* s), **1e-2**)

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

x\_new = [x[**1**] - alpha \* s[**1**], x[**2**] - alpha \* s[**2**]]

prev\_x = x

x = x\_new

push!(trajectory, x)

prev\_grad = gradient

gradient = find\_grad(x)

d\_x = [x[**1**] - prev\_x[**1**], x[**2**] - prev\_x[**2**]]

delta\_g = [gradient[**1**] - prev\_grad[**1**], gradient[**2**] - prev\_grad[**2**]]

multiplier = **1** / dot(transpose(delta\_g), H \* delta\_g)

first = d\_x\*transpose(delta\_g)\*H + H\*delta\_g\*transpose(d\_x)

ss = transpose(delta\_g)\*d\_x

sss = H \* delta\_g \* transpose(delta\_g) \* H

second = **1** + ss/dot(transpose(delta\_g), H\*delta\_g)

second = second \* sss

N = multiplier \* (first - second)

H = H + N

**end**

**return** trajectory

**end**

trajectory = goldfarb(x0, **0.001**)

x\_min = trajectory[**end**]

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

contour\_range = -**20**:**0.1**:**20**

contour\_plot = contour(contour\_range, contour\_range, (x, y) -> (x-**4**\*y)^**2** +(y+**5**)^**2**, levels=**20**, c=:blues)

plot!([point[**1**] **for** point **in** trajectory], [point[**2**] **for** point **in** trajectory], label="Движение", c=:reds)

plot!(xlabel="x1", ylabel="x2")

Метод Флетчера:

**using** Plots

**using** LinearAlgebra

**function fletcher**(x\_start, eps)

x = x\_start

trajectory = [x]

gradient = find\_grad(x)

H = [**1** **0**; **0** **1**]

prev\_x = x

prev\_grad = gradient

**while** norm(gradient) > eps

s = H \* gradient

l, r = swann\_method(alpha -> f(x - alpha \* s), **1e-2**)

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

x\_new = [x[**1**] - alpha \* s[**1**], x[**2**] - alpha \* s[**2**]]

prev\_x = x

x = x\_new

push!(trajectory, x)

prev\_grad = gradient

gradient = find\_grad(x)

de\_x = [x[**1**] - prev\_x[**1**], x[**2**] - prev\_x[**2**]]

delta\_g = [gradient[**1**] - prev\_grad[**1**], gradient[**2**] - prev\_grad[**2**]]

first\_mnog = **1** / dot(transpose(delta\_g), H\*delta\_g)

first\_add = d\_x\*transpose(delta\_g)\*H + H\*delta\_g\*transpose(d\_x)

second\_multy\_chisl = transpose(delta\_g) \* delta\_x

second\_multy\_znam = dot(transpose(delta\_g), H\*delta\_g)

second\_multy\_chisl = second\_multy\_chisl/second\_multy\_znam

sec\_2 = **1** + second\_multy\_chisl

sec\_3 = H \* delta\_g \* transpose(delta\_g)\*H

second\_add = sec\_2 \* sec\_3

chisl=first\_add-second\_add

E = [**1** **0**; **0** **1**]

first\_chisl = d\_x \* transpose(delta\_g)

first\_znam = dot(transpose(d\_x), delta\_g)

first\_mnog = first\_chisl / first\_znam

first\_mnog = E - first\_mnog

second\_chisl = delta\_g \* transpose(d\_x)

second\_znam = dot(transpose(d\_x), delta\_g)

second\_mnog = second\_chisl / second\_znam

second\_mnog = E - second\_mnog

N = first\_mnog \* H \* second\_mnog

chisl = d\_x \* transpose(d\_x)

znam = dot(transpose(d\_x), delta\_g)

H = N + chisl / znam

**end**

**return** trajectory

**end**

trajectory = fletcher(x0, **1e-6**)

x\_min = trajectory[**end**]

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

contour\_range = -**20**:**0.1**:**20**

contour\_plot = contour(contour\_range, contour\_range, (x, y) -> (x-**4**\*y)^**2** +(y+**5**)^**2**, levels=**20**, c=:blues)

plot!([point[**1**] **for** point **in** trajectory], [point[**2**] **for** point **in** trajectory], label="Движение", c=:reds)

plot!(xlabel="x1", ylabel="x2")

# **РЕЗУЛЬТАТЫ**

|  |  |  |
| --- | --- | --- |
| **Метод** | **Точка экстремума** | **Количество итераций** |
| Метод Пирсона 1 | (-19.999999802805906, -4.999999982020858) | 1880 |
| Метод Пирсона 2 | (-19.999999648901706, -4.999999916237441) | 5 |
| Метод Пирсона 3 | (-19.9999999994923, -4.9999999998837845) | 5 |
| Метод Ньютона-Рафсона | (-20.000131717104438, -4.9994805077455515) | 4 |
| Метод Гольдфарба | (-20.000000001173138, -4.999999992205972) | 3 |
| Метод Гринштадта | (-20.000000006821168, -4.999999991047872) | 3 |
| Метод Флетчера | (-19.9999999995769, -5.0000000013060335) | 3 |

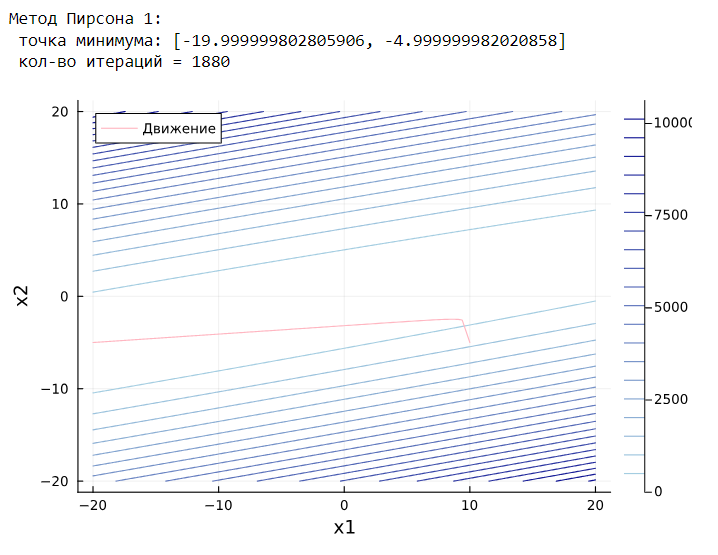


Рисунок 1 - Метод Пирсона 1

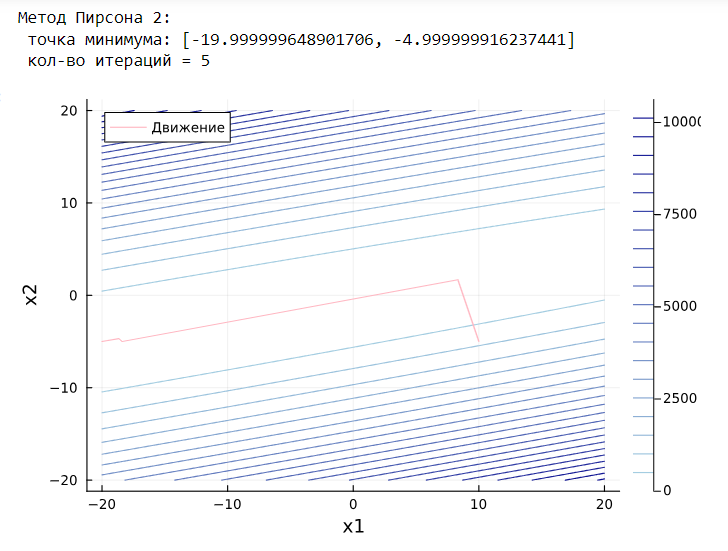


Рисунок 2 - Метод Пирсона 2

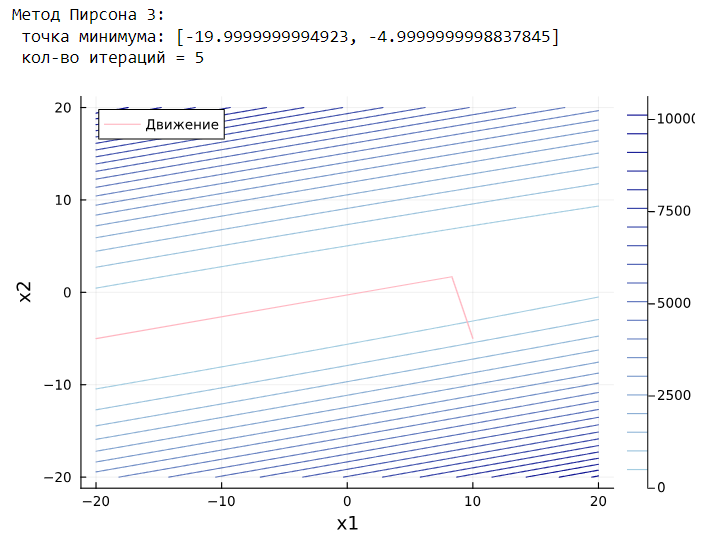


Рисунок 3 - Метод Пирсона 3

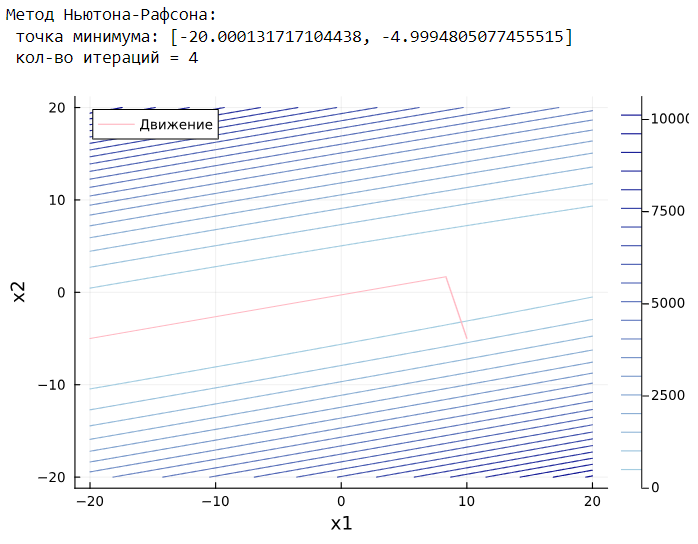


Рисунок 4 - метод Ньютона-Рафсона

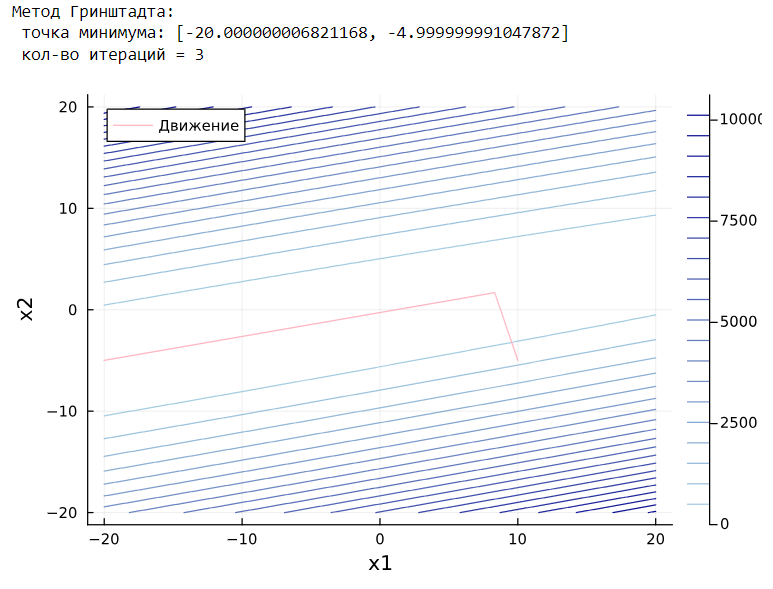


Рисунок 5 - метод Гринштадта

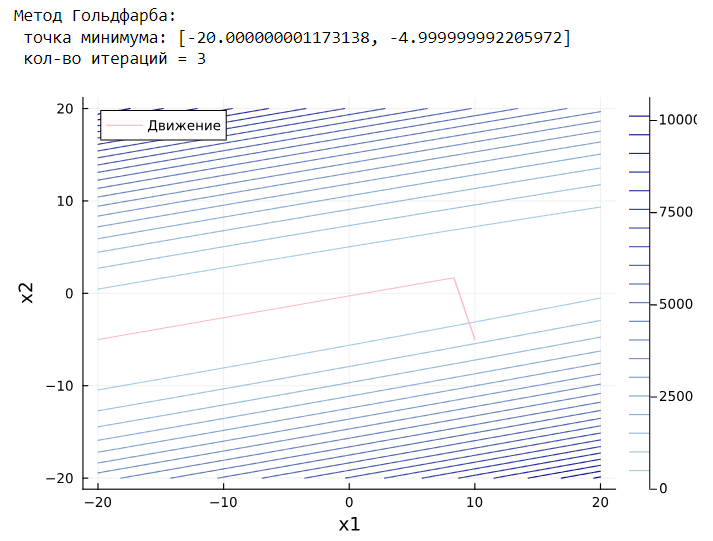


Рисунок 6 - метод Гольдфарба

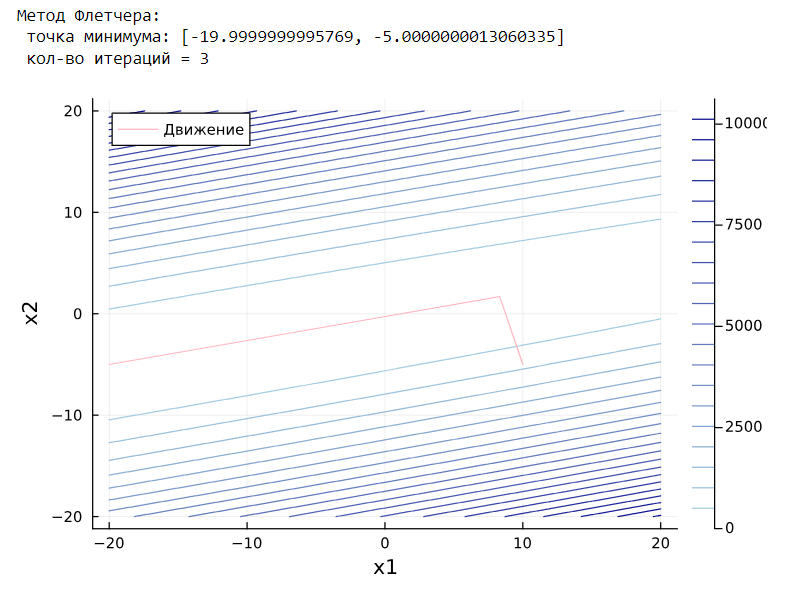


Рисунок 7 - метод Флетчера

# **ВЫВОДЫ**

В ходе выполнения лабораторной работы были реализованы и сравнены методы оптимизации многомерных функций. Поиск экстремума был реализован с помощью следующих методов: метод Пирсона и его модификации, Ньютона-Рафсона, методы Гринштадта и Гольдфарба, метод Флетчера. Наилучший результат показали методы Гринштадта и Гольдфарба и метод Флетчера.