**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 № 2

по курсу « Методы оптимизации »

**СРАВНЕНИЕ МЕТОДОВ ФЛЭТЧЕРА-РИВСА, ПОЛАКА-РИБЬЕРА, ХЕСТЕНСА-СТИФЕЛЯ, ДАЯНА И ДИКСОНА**

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 10](#_Toc158799287)

# **ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ**

**Цель работы:**

Реализовать поиск экстремума функции следующими методами: сопряженных градиентов (Флэтчер-Ривс), Полака-Рибьера, Хейстонса-Стифеля, Даяна, Диксона. Сравнить полученные результаты

**Постановка задачи:**

1. Исследовать алгоритмы оптимизации, включая метод Флэтчера-Ривса, метод Полака-Рибьера, метод Хейстонса-Стифеля, метод Даяна и метод Диксона, с целью нахождения локального минимума функции многих переменных.
2. Реализовать численные методы оптимизации на языке программирования Julia.
3. Применить алгоритмы оптимизации к функции Розенброка и 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:

**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**

**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**

**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 сопряженных градиентов Флэтчера-Ривса:

**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**

Метод Полака-Рибьера:

**function polak\_rebier**(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-prev\_grad)) / dot(prev\_grad, prev\_grad)

d = -grad + beta \* d

prev\_x = copy(x)

push!(trajectory, x)

**end**

**return** x, trajectory

**end**

Метод Хестенса-Стифеля:

**function hestens\_stifel**(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)

prev\_d = copy(d)

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-prev\_grad)) / dot(prev\_d, (grad-prev\_grad))

d = -grad + beta \* prev\_d

prev\_x = copy(x)

push!(trajectory, x)

**end**

**return** x, trajectory

**end**

Метод Даяна:

**function dai\_yuan**(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-prev\_grad)) / dot(d, (grad-prev\_grad))

d = -grad + beta \* d

prev\_x = copy(x)

push!(trajectory, x)

**end**

**return** x, trajectory

**end**

Метод Диксона:

**function dickson**(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(d, prev\_grad)

d = -grad + beta \* d

prev\_x = copy(x)

push!(trajectory, x)

**end**

**return** x, trajectory

**end**

# **РЕЗУЛЬТАТЫ**

Таблица 1 - Сравнение результатов для функции Розенброка

|  |  |  |
| --- | --- | --- |
| **Метод** | **Точка экстремума** | **Количество итераций** |
| Метод Флэтчера-Ривса | (0.9999942529858412, 0.999988482684302) | 9 |
| Метод Полака-Рибьера | (0.9999999667944447, 0.9999999356877168) | 13 |
| Метод Даяна | (1.0000000150814559, 1.0000000303704493) | 9 |
| Метод Хестенса-Стифеля | (1.0000000150814559, 1.0000000303704493) | 9 |
| Метод Диксона | (0.998285921003996, 0.9965697243308758) | 39 |

Таблица 2 - Сравнение результатов для функции по варианту

|  |  |  |
| --- | --- | --- |
| **Метод** | **Точка экстремума** | **Количество итераций** |
| Метод Флэтчера-Ривса | (-20.00000002844816, -5.0000000067147665) | 5 |
| Метод Полака-Рибьера | (-19.999999998777607, -5.000000004984948) | 3 |
| Метод Даяна | (-19.99999999880998, -5.000000004804756) | 3 |
| Метод Хестенса-Стифеля | (-19.99999999880998, -5.000000004804756) | 3 |
| Метод Диксона | (-19.99978251887352, -4.999948893121517) | 365 |

Далее представлены графики траекторий движения к минимуму для функции Рознеброка.

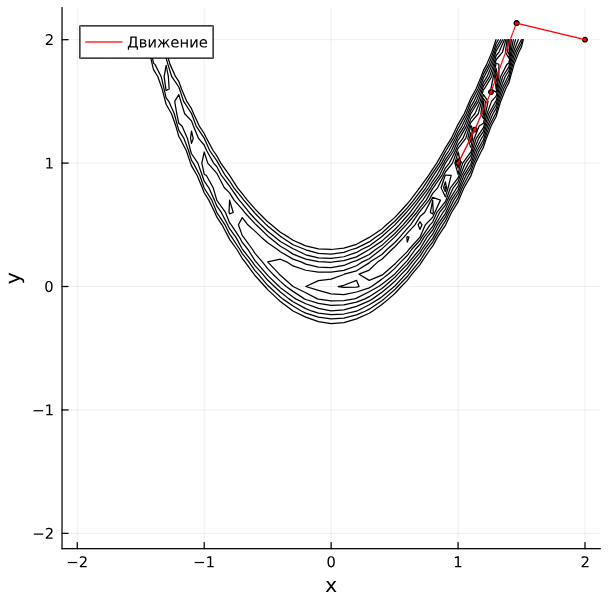


Рисунок 1 - Метод сопряженных градиентов Флэтчера-Ривса

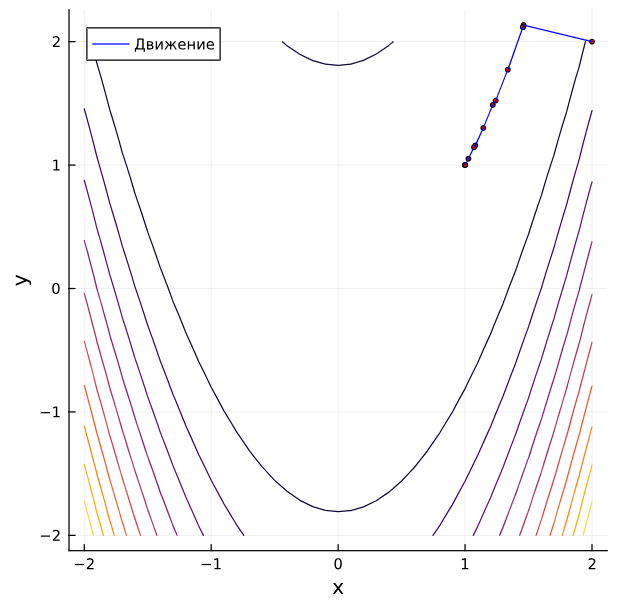


Рисунок 2 - Метод Полака-Рибьера

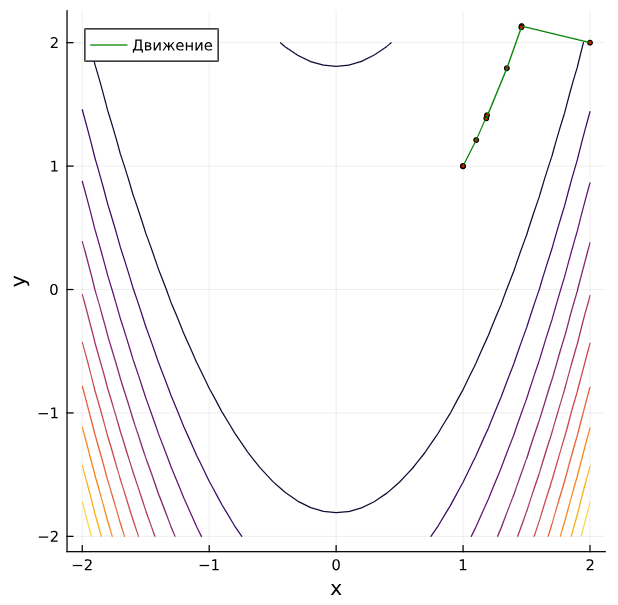


Рисунок 3 - Метод Хестенса-Стифеля

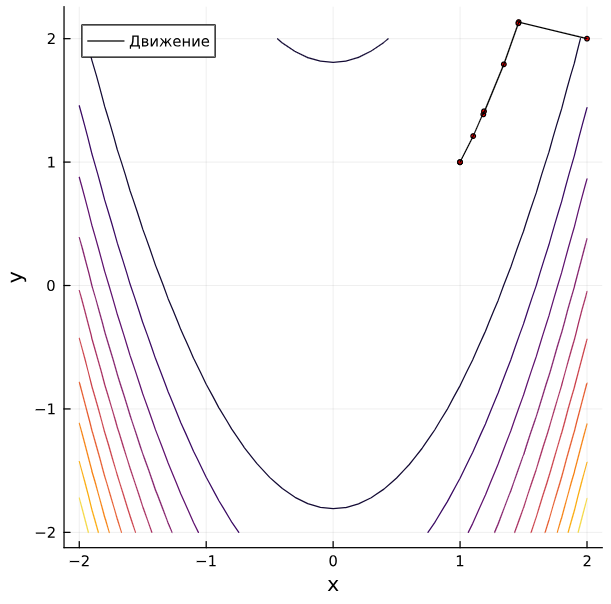


Рисунок 4 - метод Даяна

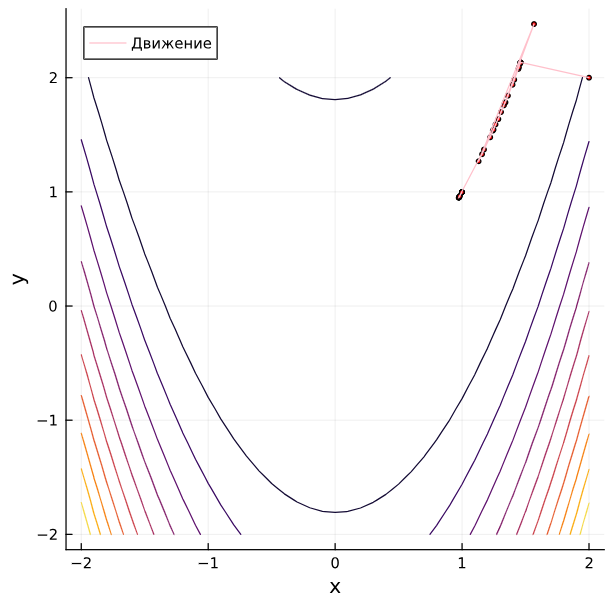


Рисунок 5 - метод Диксона

# **ВЫВОДЫ**

В ходе выполнения лабораторной работы были реализованы и сравнены методы оптимизации многомерных функций. Поиск экстремума был реализован с помощью следующих методов: метод Флэтчера-Ривса, метод Полака-Рибьера, метод Хейстонса-Стифеля, метод Даяна и метод Диксона. Наилучшие результаты (наименьшее число итераций и наилучшая точность значения точки экстремума) показали методы Хейстонса-Стифеля и Даяна.