|  |  |
| --- | --- |
|  | **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 Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Лабораторная работа №3**

**“Методы локальной оптимизации”**

**ПО КУРСУ:**

***«Методы оптимизация»***

Студент ИУ9-82Б Потребина В. В.

Преподаватель Посевин Д. П.

*Москва, 2024 г.*

**ОГЛАВЛЕНИЕ**

[1. Постановка задачи 3](#_heading=h.30j0zll)

[2. Практическая реализация 7](#_heading=h.1fob9te)

3[. Результаты](#_heading=h.1fob9te) 10

# 1. Постановка задачи

**Цели:**

Изучение и реализация различных методов **локальной оптимизации**, включая:

* **Метод Хука-Дживса** (Hooke-Jeeves)
* **Метод покоординатного спуска** (Coordinate Descent)
* **Метод Гаусса-Зейделя** (Gauss-Seidel)

Методы применяются для поиска минимума многомерных функций **без использования градиента**.

**Постановка задач:**

В данной лабораторной работе требуется реализовать и исследовать методы **локальной оптимизации** для поиска минимума многомерных функций **без использования градиента**. Основная цель — нахождение оптимального решения с заданной точностью при минимальном количестве итераций.

# 2. Практическая реализация

#import Pkg; Pkg.add("Gaston"); Pkg.add("SpecialFunctions"); Pkg.add("Plots"); Pkg.add("PlotlyJS");

using Plots, PlotlyJS, SpecialFunctions

target(x::Float64) = (x^3-15\*x^2+7\*x+1)/10

target\_ravine(p) = sum(p .\* p)

function target\_rastrygin(p)

    A = 10

    result = A\*length(p)

    for idx in 1:length(p)

        result += p[idx]^2 - A\*cos(2\*pi\*p[idx])

    end

    return result

end

function target\_schefill(p)

    A = 418.9829

    result = A\*length(p)

    for idx in 1:length(p)

        result -= p[idx]\*sin(sqrt(abs(p[idx])))

    end

    return result

end

target\_rosenbrock(p) = (1 - p[1])^2 + 100\*(p[2] - p[1]^2)^2

global Lf\_max=0

function df(f, points)

    result = []

    for idx in 2:length(points)-1

        push!(result, (f(points[idx+1]) - f(points[idx-1]))/(points[idx+1]-points[idx-1]))

        global Lf\_max = max(Lf\_max, max(abs(result[end])))

    end

    return result

end

function ddf(f, points)

    result = []

    for idx in 2:length(points)-1

        push!(result, (f(points[idx+1]) - 2 \* f(points[idx]) + f(points[idx-1]))/(points[idx+1]-points[idx])^2)x\_candidate

    end

    return result

end

function ddf\_with\_points(f, points)

    result = []

    for idx in 2:length(points)-1

        push!(result, (f(points[idx+1]) - 2 \* f(points[idx]) + f(points[idx-1]))/(points[idx+1]-points[idx])^2)

    end

    return result, points[2:length(points)-1]

end

function get\_fib(n)

    sqrt5 = sqrt(5)

    trunc(Int, (1 / sqrt5) \* (((1 + sqrt5) / 2)^n - ((1  - sqrt5) / 2)^n))

end

function is\_unimodal\_df(f, a, b, N=1000)

    deriv = df(f, range(a, b, N))

    last\_deriv = deriv[1]

    for current\_deriv in deriv[2:length(deriv)-1]

        if current\_deriv <= last\_deriv

            return false

        end

    end

    return true

end

function is\_unimodal\_ddf(f, a, b, N=1000)

    for current\_deriv in ddf(f, range(a, b, N))

        if current\_deriv < 0

            return false

        end

    end

    return true

end

function search\_unimodal\_segment(f, a, b, N=1000)

    deriv, points = ddf\_with\_points(f, range(a, b, N))

    interval\_start, interval\_end = nothing, nothing

    result\_start, result\_end = nothing, nothing

    for (second\_deriv, point) in zip(deriv, points)

        if second\_deriv < 0

            if interval\_start != nothing

                if (interval\_end - interval\_start) > (result\_end - result\_start)

                    result\_start, result\_end = interval\_start, interval\_end

                end

                interval\_start = nothing

                interval\_end = nothing

            end

        elseif second\_deriv > 0

            if interval\_start == nothing

                interval\_start = point

            end

            interval\_end = point

        end

    end

    if result\_end == nothing || result\_start == nothing

        return interval\_start, interval\_end

    end

    if (interval\_end - interval\_start) > (result\_end - result\_start)

        result\_start, result\_end = interval\_start, interval\_end

    end

    return result\_start, result\_end

end

function norm(p)

    return sqrt(sum(p .\* p))

end

function function\_min\_iter(f, start, stop, n)

    dots = []

    step = (stop - start) / n

    min\_value = f(start)

    min\_point = start

    for i in 1:n

        x = start + i \* step

        append!(dots, x)

        if f(x) < min\_value

            min\_value = f(x)

            min\_point = x

        end

    end

    error = (stop - start)/n

    return min\_point, dots, error, n

end

function function\_min\_dyhotomy(f, a, b, eps=1.e-6)

    dots = []

    iter = 0

    while abs(b - a) > eps

        iter += 1

        mid = (a + b) / 2

        f1 = f(mid - eps)

        f2 = f(mid + eps)

        append!(dots, mid - eps)

        append!(dots, mid + eps)

        if f1 < f2

            b = mid

        else

            a = mid

        end

    end

    error = Lf\_max \* eps / 2

    return (a + b) / 2, dots, error, iter

end

function function\_min\_dyhotomy\_impl(f, a, b, eps=1.e-6)

    while abs(b - a) > eps

        mid = (a + b) / 2

        f1 = f(mid - eps)

        f2 = f(mid + eps)

        if f1 < f2

            b = mid

        else

            a = mid

        end

    end

    return (a + b) / 2

end

function function\_min\_golden(f, start, stop, eps=1.e-6)

    phi = (sqrt(5.0) - 1) / 2

    dots = []

    iter = 0

    while abs(stop - start) > eps

        iter += 1

        c = stop - (stop - start) \* phi

        d = start + (stop - start) \* phi

        append!(dots, c)

        append!(dots, d)

        if f(c) < f(d)

            stop = d

        else

            start = c

        end

    end

    error = Lf\_max \* phi \* eps

    return (start + stop) / 2, dots, error, iter

end

function function\_min\_fib(f, a, b, n, eps=1.e-6)

    x1 = a + (get\_fib(n) / get\_fib(n + 2)) \* (b - a)

    x2 = a + b - x1

    y1 = f(x1)

    y2 = f(x2)

    dots = []

    iter = 0

    while abs(b - a) > eps

        iter += 1

        if y1 <= y2

            b = x2

            x2 = x1

            x1 = a + b - x1

            y2 = y1

            y1 = f(x1)

        else

            a = x1

            x1 = x2

            x2 = a + b - x2

            y1 = y2

            y2 = f(x2)

        end

        append!(dots, (a + b) / 2)

    end

    error = Lf\_max \* eps

    return (a + b) / 2, dots, error, iter

end

function sven\_localization(f, x\_0, h = 0.01)

    x\_i = []

    x\_0 = copy(x\_0)

    push!(x\_i, x\_0)

    direction = 1

    x = x\_0 + direction \* h

    if f(x) > f(x\_0)

        direction = -1

        x = x\_0 + direction \* h

    end

    push!(x\_i, x)

    while f(x + direction \* h) < f(x)

        h \*= 2

        buf = x\_0

        x\_0 = x

        x = buf + direction \* h

        push!(x\_i, x)

    end

    if size(x\_i)[1] > 2

        return minmax(x\_i[end], x\_i[lastindex(x\_i) - 1])

    else

        return minmax(x\_i[end], x\_i[lastindex(x\_i) - 1])

    end

end

function one\_dim\_optimize(f, x\_0=0.0)

    a, b = sven\_localization(f, x\_0)

    x\_opt = function\_min\_dyhotomy\_impl(f, a, b)

    return x\_opt

end

function coordinate\_descent(f, x\_0, h\_0, eps = 1e-6, upper\_bound=1e3)

    alpha = h\_0

    x = [x\_0]

    k = 0

    while k < upper\_bound

        x\_min = copy(x[end])

        k += 1

        for i in 1:length(x\_0)

            e\_k = x[end][i] / norm(x[end])

            x\_i\_new\_1 = x[end][i] + alpha \* e\_k

            x\_i\_new\_2 = x[end][i] - alpha \* e\_k

            x\_candidate = copy(x[end])

            x\_candidate[i] = x\_i\_new\_1

            if f(x\_min) > f(x\_candidate)

                x\_min = x\_candidate

            end

            x\_candidate[i] = x\_i\_new\_2

            if f(x\_min) > f(x\_candidate)

                x\_min = x\_candidate

            end

        end

        if norm(x\_min - x[end]) < eps

            break

        end

        if f(x\_min) < f(x[end])

            push!(x, x\_min)

        else

            alpha = alpha \* 0.5

        end

        if length(x) > 1 && norm(x[end] - x[end-1]) < eps

            break

        end

    end

    return x[end], k, x

end

function hooke\_jeeves\_method(f, x\_0, h\_0, eps = 1e-6, upper\_bound=1e3)

    alpha = h\_0

    x = [x\_0]

    k = 0

    points\_added = 1

    while k < upper\_bound

        x\_min = copy(x[end])

        k += 1

        if points\_added > 0 && points\_added == 3

            points\_added = 1

            x0 = copy(x[lastindex(x) - 2])

            x1 = copy(x[lastindex(x) - 1])

            x2 = copy(x[end])

            x\_min = 2\*x2-x0

            push!(x, x\_min)

        end

        for i in 1:length(x\_0)

            x\_i\_new\_1 = x[end][i] + alpha

            x\_i\_new\_2 = x[end][i] - alpha

            x\_candidate = copy(x[end])

            x\_candidate[i] = x\_i\_new\_1

            if f(x\_min) > f(x\_candidate)

                x\_min = copy(x\_candidate)

            end

            x\_candidate[i] = x\_i\_new\_2

            if f(x\_min) > f(x\_candidate)

                x\_min = copy(x\_candidate)

            end

        end

        if norm(x\_min - x[end]) < eps

            break

        end

        if f(x\_min) < f(x[end])

            push!(x, x\_min)

            points\_added+=1

        else

            alpha \*= 0.5

        end

        if length(x) > 1 && norm(x[end] - x[end-1]) < eps

            break

        end

    end

    return x[end], k, x

end

function Gauss\_Zeidel(f, x\_0, h\_0, eps = 1e-6, upper\_bound=1e3)

    alpha = h\_0

    x = [x\_0]

    k = 0

    while k < upper\_bound

        k += 1

        x\_min = copy(x[end])

        for i in 1:length(x\_0)

            x\_candidate = copy(x[end])

            e\_k = zeros(length(x\_0))

            e\_k[i] = x\_candidate[i] / norm(x\_candidate)

            g(a) = f(x\_candidate + a\*e\_k)

            alpha\_new = one\_dim\_optimize(g, alpha)

            x\_new\_i\_1 = x\_candidate + alpha\_new \* e\_k

            x\_new\_i\_2 = x\_candidate - alpha\_new \* e\_k

            if f(x\_min) > f(x\_new\_i\_1)

                x\_min = x\_new\_i\_1

                alpha = alpha\_new

            end

            if f(x\_min) > f(x\_new\_i\_2)

                x\_min = x\_new\_i\_2

                alpha = alpha\_new

            end

        end

        push!(x, x\_min)

        if length(x) > 1 && norm(x[end] - x[end-1]) < eps

            break

        end

    end

    return x[end], k, x

end

function plot\_method(f, points, func\_min, iterations, title)

    n = 5

    x = y = -n:0.1:n

    px = [points[i][1] for i in 1:length(points)]

    py = [points[i][2] for i in 1:length(points)]

    plt = Plots.plot(size=(1000, 1000), title=title)

    Plots.plot!(plt, x, y, (x,y)->f([x,y]), st = :contour, levels=:40)

    Plots.plot!(plt, px, py, seriestype=:scatter, color = "blue")

    Plots.plot!(plt, px, py, color = "blue")

    Plots.plot!(plt, [func\_min[1]], [func\_min[2]], seriestype=:scatter, color = "red")

    Plots.plot!(plt, [], [], labels="iterations=$(iterations)")

    return plt

end

function plot\_method\_dim(f, points, func\_min, func\_title)

    x\_vals = range(-10, 10, length=50)

    y\_vals = range(-10, 10, length=50)

    Z = [f([x, y]) for x in x\_vals, y in y\_vals]

    surface\_plot = PlotlyJS.plot(

        [

            PlotlyJS.surface(z=Z, x=x\_vals, y=y\_vals, colorscale="Viridis", opacity=0.6),

            PlotlyJS.scatter3d(x=first.(points), y=last.(points), z=[f(x) for x in points],

                               mode="markers", marker=attr(size=5, color="red"))

        ]

    )

    display(surface\_plot)

end

function optimize\_func(func, x0, h0, title="")

    hooke\_jeeves\_min, hooke\_jeeves\_iterations, hooke\_jeeves\_points = hooke\_jeeves\_method(func, x0, h0)

    coordinate\_descent\_min, coordinate\_descent\_iterations, coordinate\_descent\_points = coordinate\_descent(func, x0, h0)

    gauss\_zeidel\_min, gauss\_zeidel\_iterations, gauss\_zeidel\_points = Gauss\_Zeidel(func, x0, h0)

    if title != ""

        print("$(repeat('=', 20))$(title)$(repeat('=', 20))\n")

    end

    print("HookeJeeves: k = ", hooke\_jeeves\_iterations, "    x\_min = (", hooke\_jeeves\_min[1], ", ", hooke\_jeeves\_min[2], ")\n")

    print("CoordDescent: k = ", coordinate\_descent\_iterations, "    x\_min = (", coordinate\_descent\_min[1], ", ", coordinate\_descent\_min[2], ")\n")

    print("GaussZeidel: k = ", gauss\_zeidel\_iterations, "    x\_min = (", gauss\_zeidel\_min[1], ", ", gauss\_zeidel\_min[2], ")\n")

    print("$(repeat('=', 50))\n")

    x\_vals = range(-10, 10, length=50)

    y\_vals = range(-10, 10, length=50)

    Z = [func([x, y]) for x in x\_vals, y in y\_vals]

    p = PlotlyJS.plot(

        [

            PlotlyJS.surface(z=Z, x=x\_vals, y=y\_vals, colorscale="Viridis", opacity=0.6),

            PlotlyJS.scatter3d(x=first.(hooke\_jeeves\_points), y=last.(hooke\_jeeves\_points), z=[func(p) for p in hooke\_jeeves\_points],

                             mode="markers+lines", marker=attr(size=5, color="red"),

                             name="Метод Хука-Дживса"),

            PlotlyJS.scatter3d(x=first.(coordinate\_descent\_points), y=last.(coordinate\_descent\_points), z=[func(p) for p in coordinate\_descent\_points],

                             mode="markers+lines", marker=attr(size=5, color="blue"),

                             name="Метод покоординатного спуска"),

            PlotlyJS.scatter3d(x=first.(gauss\_zeidel\_points), y=last.(gauss\_zeidel\_points), z=[func(p) for p in gauss\_zeidel\_points],

                             mode="markers+lines", marker=attr(size=5, color="green"),

                             name="Метод Гаусса-Зейделя")

        ]

    )

    p1 = plot\_method(func, hooke\_jeeves\_points, hooke\_jeeves\_min, hooke\_jeeves\_iterations, "HookeJeeves")

    p2 = plot\_method(func, coordinate\_descent\_points, coordinate\_descent\_min, coordinate\_descent\_iterations, "CoordDescent")

    p3 = plot\_method(func, gauss\_zeidel\_points, gauss\_zeidel\_min, gauss\_zeidel\_iterations, "GaussZeidel")

    display(Plots.plot(p1, p2, p3))

    return display(p)

end

optimize\_func(target\_ravine, [4.0,4.0], 1, "target")

readline()

# Результаты

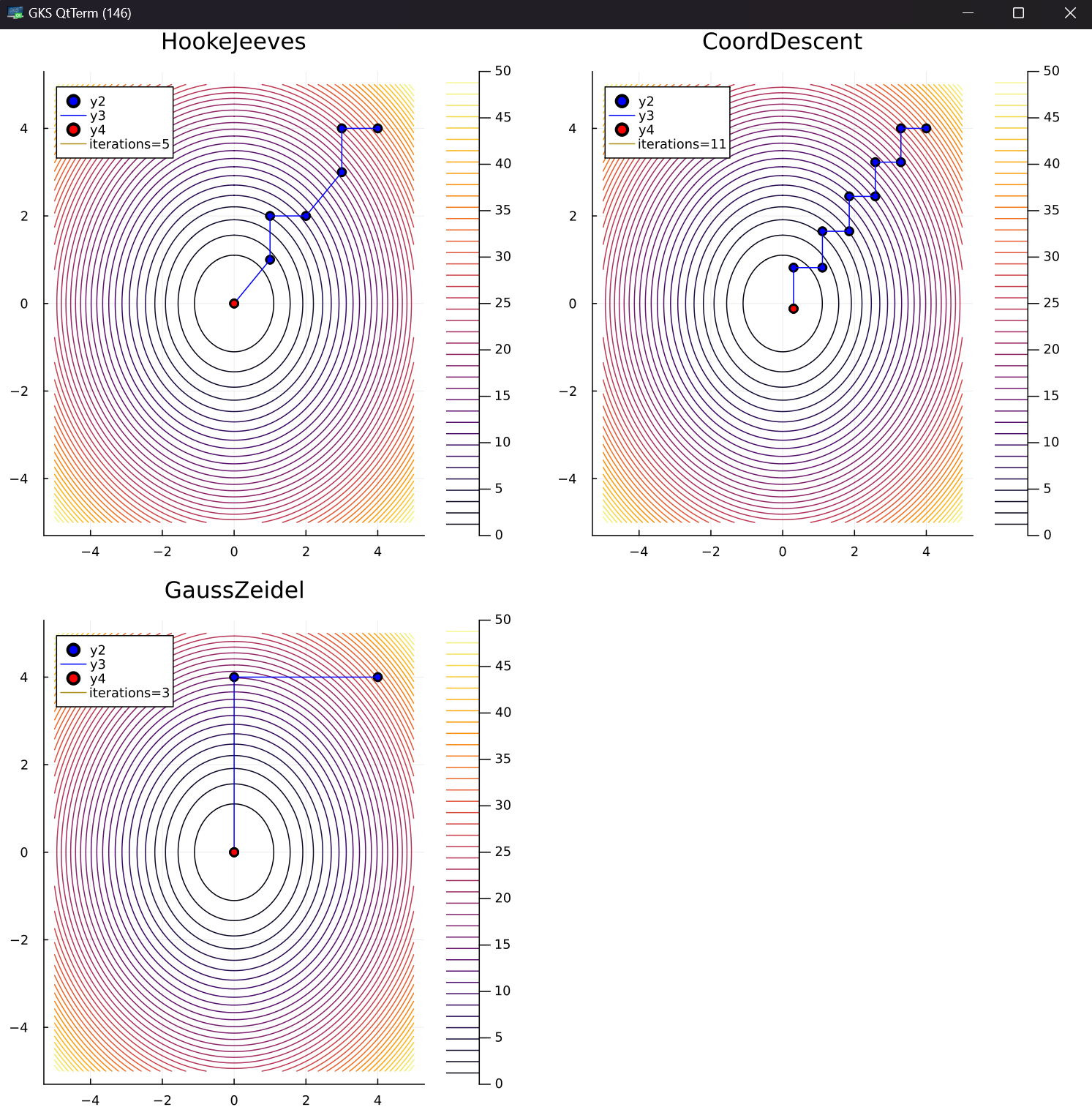
Тестирование функционала было проведено на функции параболоида.

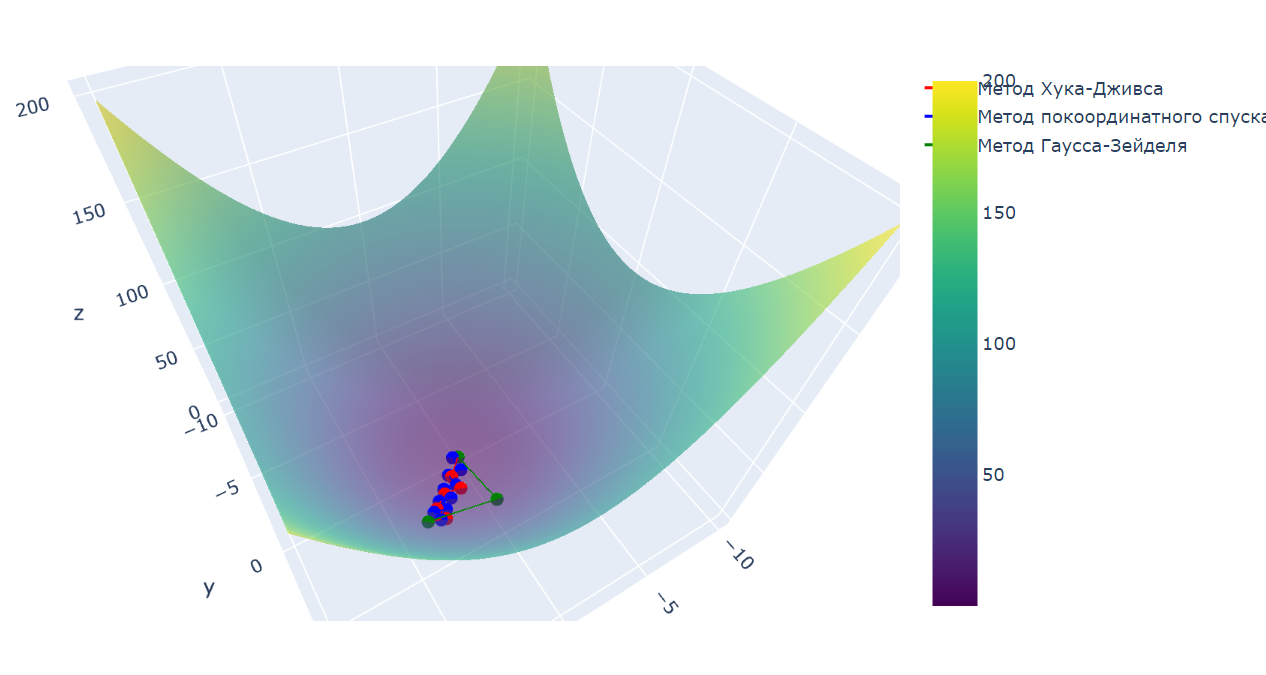
====================target====================

HookeJeeves: k = 5 x\_min = (0.0, 0.0)

CoordDescent: k = 11 x\_min = (0.30262736095475495, -0.11897800378573609)

GaussZeidel: k = 3 x\_min = (2.040967794769699e-8, -4.410743592586641e-8)





# Выводы

**Все три метода успешно находят минимум целевой функции**.

**Метод Гаусса-Зейделя** сходится быстрее, чем покоординатный спуск, так как использует обновлённые значения на каждой итерации.

**Метод Хука-Дживса** эффективно комбинирует поиск в различных направлениях и базовое перемещение, что делает его полезным в задачах без гладкости.

**Графическая визуализация подтверждает эффективность алгоритмов** и помогает анализировать их поведение.