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

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

```julia
#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

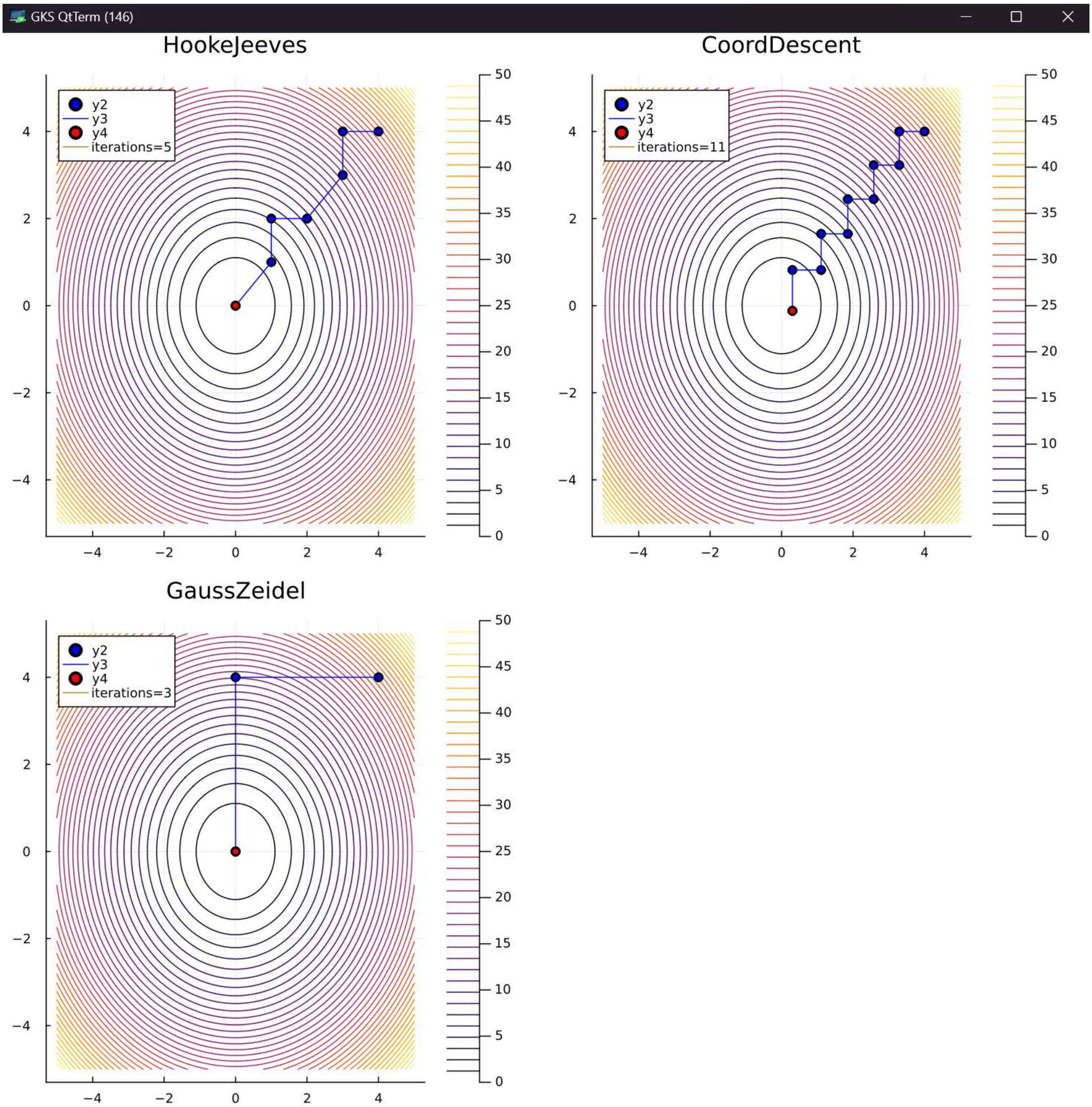
Тестирование функционала было проведено на функции параболоида.

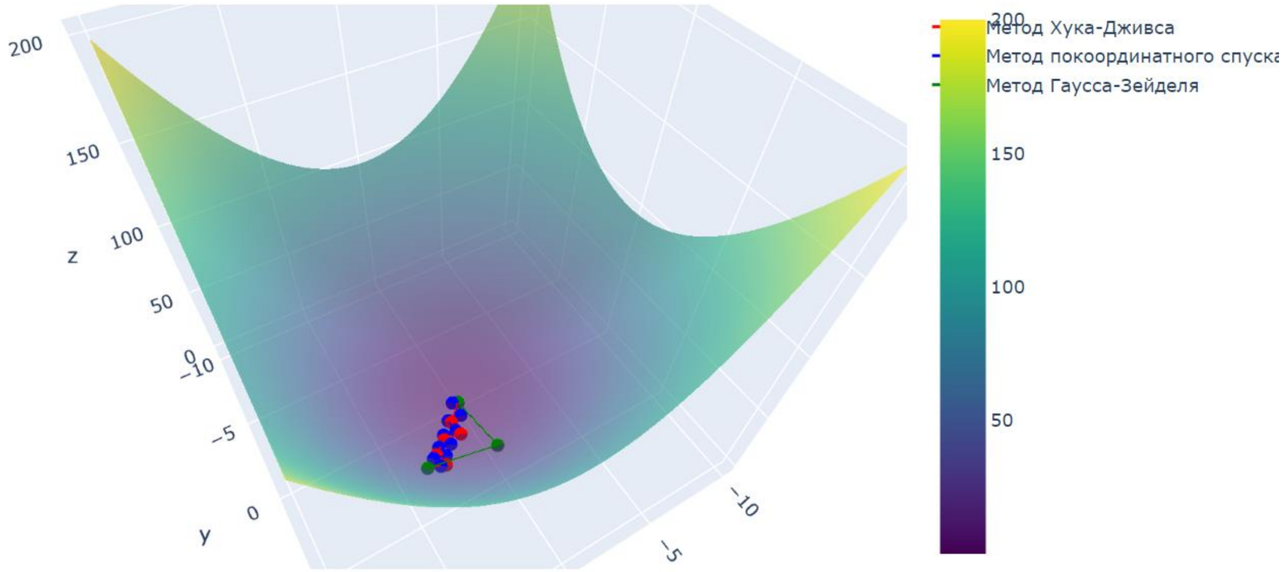
```
====================target====================

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)
```


GKS QtTerm (146)
HookeJeeves
y2
y3
y4
iterations=5
CoordDescent
y2
y3
y4
iterations=11
GaussZeidel
y2
y3
y4
iterations=3



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.