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 № 9

по курсу «Методы оптимизации»

«Метод Крэгга и Леви для различных n»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

1. Реализовать метод Крэгга и Леви

2. Установить зависимость количества итераций алгоритма от количества параметров.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```

using LinearAlgebra
using Plots
using PlotlyBase
plotly()

function approximate_gradient(f, x, h=1e-8)
    n = length(x)
    grad = zeros(n)

    for i in 1:n
        x_plus_h = copy(x)
        x_plus_h[i] += h

        x_minus_h = copy(x)
        x_minus_h[i] -= h

        grad[i] = (f(x_plus_h) - f(x_minus_h)) / (2*h)
    end

    return grad
end

#

function golden_section_search(f, a, b, tol=1e-6, max_iter=100)
    golden_ratio = (sqrt(5) + 1) / 2
    c = b - (b - a) / golden_ratio
    d = a + (b - a) / golden_ratio

    fc = f(c)
    fd = f(d)
```

```
    iter = 0
    while abs(b - a) > tol && iter < max_iter
        iter += 1

        if fc < fd
            b = d
            d = c
            fd = fc
            c = b - (b - a) / golden_ratio
            fc = f(c)
        else
            a = c
            c = d
            fc = fd
            d = a + (b - a) / golden_ratio
            fd = f(d)
        end
    end

    return (a + b) / 2
end

function nelder_mead_nd(f; start_point=nothing, n_dim=2,  =1.0,  =2.0,  =0.5,  =0.5, tolerance=0.001, max_iter=10, edge_length=1.0)
    if start_point === nothing
        start_point = zeros(n_dim)
    end
    simplex = [copy(start_point)]
    for i in 1:n_dim
        vertex = copy(start_point)
        vertex[i] += edge_length
        push!(simplex, vertex)
    end
    xs = deepcopy(simplex)
    simplex_history = [deepcopy(simplex)]

    for iteration in 1:max_iter
        f_values = [f(p) for p in simplex]
        perm = sortperm(f_values)
        simplex = simplex[perm]
        f_values = f_values[perm]

        if std(f_values) < tolerance
            return simplex[1], xs, simplex_history
        end
```

```

        centroid = zeros(n_dim)
        for i in 1:length(simplex)-1
            centroid .+= simplex[i]
        end
        centroid ./= (length(simplex)-1)

        worst = simplex[end]
        x_r = centroid .+    .* (centroid .- worst)
        f_r = f(x_r)
        if f_values[1] <= f_r && f_r < f_values[end-1]
            simplex[end] = x_r
            push!(xs, x_r)
            push!(simplex_history, deepcopy(simplex))
            continue
        end

        if f_r < f_values[1]
            x_e = centroid .+    .* (x_r .- centroid)
            f_e = f(x_e)
            if f_e < f_r
                simplex[end] = x_e
            else
                simplex[end] = x_r
            end
            push!(xs, simplex[end])
            push!(simplex_history, deepcopy(simplex))
            continue
        end

        x_c = centroid .+    .* (worst .- centroid)
        f_c = f(x_c)
        if f_c < f_values[end]
            simplex[end] = x_c
            push!(xs, x_c)
            push!(simplex_history, deepcopy(simplex))
            continue
        end

        best = simplex[1]
        for i in 2:length(simplex)
            simplex[i] = best .+    .* (simplex[i] .- best)
            push!(xs, simplex[i])
        end
        push!(simplex_history, deepcopy(simplex))
    end
```

```
    return simplex[1], xs, simplex_history
end



function miel_contrell_n(f, initial_x,m; max_iter=1000, =1e-6)
    x_prev = copy(initial_x)
    x_current = copy(initial_x)
    n_dim = length(initial_x)
    prev_deltas = [zeros(n_dim) for _ in 1:m]
    trajectory = [copy(initial_x)]

    for k in 0:max_iter
         f_current = approximate_gradient(f,x_current)
        if k % (m+1) == 0
            prev_deltas = [zeros(n_dim) for _ in 1:m]
        end
        function target( )
            new_x = x_current - [1] .* f_current
            for i in 1:m
                new_x += [i+1] * prev_deltas[i]
            end
            return f(new_x)
        end

        start_point = zeros(m+1)
        optimum, _, _ = nelder_mead_nd(target, start_point=start_point,
    n_dim=(m+1), tolerance=1e-8, max_iter=25, edge_length=0.1)
         = optimum

        x_next = x_current - [1] .* f_current
        for i in 1:m
            x_next += [i+1] * prev_deltas[i]
        end

        push!(trajectory, copy(x_next))

        if k > 0 && abs(f(x_current) - f(x_next)) <
            println("Stopping condition met at iteration $k")
            return x_next, trajectory
        end

        new_delta = x_current - x_prev
        pushfirst!(prev_deltas, new_delta)
        pop!(prev_deltas)

```

```
        x_prev = copy(x_current)
        x_current = copy(x_next)
    end

    println("Maximum iterations reached")
    return x_current, trajectory
end

#

function plot_optimization_paths(f, x_range, y_range, paths_with_names,
    title="                                                              ",
    global_min=nothing)
    z = [f([x, y]) for y in y_range, x in x_range]

    clamp_level = maximum(filter(isfinite, z)) / 2
    z_clamped = [min(val, clamp_level) for val in z]

    p = Plots.contour(x_range, y_range, z_clamped,
                fill=false,
                levels=20,
                color=:thermal,
                xlabel="x",
                ylabel="y",
                title=title,
                size=(800, 600))

    colors = [:red, :green, :blue, :purple, :orange,:red, :green, :blue
    ,:blue,:blue,:blue,:blue,]
    for (i, (name, path)) in enumerate(paths_with_names)
        x_coords = [point[1] for point in path]
        y_coords = [point[2] for point in path]

        plot!(p, x_coords, y_coords,
              label=name,
              line=(colors[i], 2),
              marker=(:circle, 2, 0.5))

        annotate!(p, x_coords[1], y_coords[1], text("          ", :left,
     8, :white))
        annotate!(p, x_coords[end], y_coords[end], text("          ", :
    right, 8, :white))
    end

    if global_min !== nothing
        scatter!(p, [global_min[1]], [global_min[2]],
```

```
                    label="                                        ",
                    color=:white,
                    markersize=5,
                    markerstrokewidth=1,
                    markerstrokecolor=:black)
    end

    return p
end


#
function run_optimization(f, x0, title, x_range, y_range, global_min=nothing)
    println("\n======= $(title) =======")

    result_mcn0, path_mcn0 = miel_contrell_n(f, x0,0)

    println("                                    (                    ) 0: ", result_mcn0)
    println("                                   : ", f(result_mcn0))
    println("                                          : ", length(path_mcn0)-1)

    result_mcn1, path_mcn1 = miel_contrell_n(f, x0,1)

    println("                                    (                    ) 1: ", result_mcn1)
    println("                                   : ", f(result_mcn1))
    println("                                          : ", length(path_mcn1)-1)

    result_mcn2, path_mcn2 = miel_contrell_n(f, x0,2)

    println("                                    (                    ) 2: ", result_mcn2)
    println("                                   : ", f(result_mcn2))
    println("                                          : ", length(path_mcn2)-1)

    result_mcn3, path_mcn3 = miel_contrell_n(f, x0,3)

    println("                                    (                    ) 3: ", result_mcn3)
    println("                                   : ", f(result_mcn3))
```

```
    println("                              : ", length(path_mcn3)-1)

    result_mcn4, path_mcn4 = miel_contrell_n(f, x0,4)

    println("                                   (                ) 4: ", result_mcn4)
    println("                         : ", f(result_mcn4))
    println("                              : ", length(path_mcn4)-1)

    result_mcn5, path_mcn5 = miel_contrell_n(f, x0,5)

    println("                                   (                ) 5: ", result_mcn5)
    println("                         : ", f(result_mcn5))
    println("                              : ", length(path_mcn5)-1)

    result_mcn6, path_mcn6 = miel_contrell_n(f, x0,6)

    println("                                   (                ) 6: ", result_mcn6)
    println("                         : ", f(result_mcn6))
    println("                              : ", length(path_mcn6))

    result_mcn9, path_mcn9= miel_contrell_n(f, x0,9)

    println("                                   (                ) 9: ", result_mcn9)
    println("                         : ", f(result_mcn9))
    println("                              : ", length(path_mcn9))

    result_mcn10, path_mcn10 = miel_contrell_n(f, x0,10)

    println("                                   (                ) 10: ", result_mcn10)
    println("                         : ", f(result_mcn10))
    println("                              : ", length(path_mcn10))

    result_mcn12, path_mcn12 = miel_contrell_n(f, x0,12)

```

```
println("                                    (
                    ) 10: ", result_mcn12)
println("                              : ", f(result_mcn12))
println("                                    : ", length(path_mcn12))

result_mcn20, path_mcn20 = miel_contrell_n(f, x0,20)

println("                                    (
                    ) 20: ", result_mcn20)
println("                              : ", f(result_mcn20))
println("                                    : ", length(path_mcn20))

#
paths = [
    ("                                          - 0", path_mcn0),
    ("                                          - 1", path_mcn1),
    ("                                          - 2", path_mcn2),
    ("                                          - 3", path_mcn3),
    ("                                          - 4", path_mcn4),
    ("                                          - 5", path_mcn5),
    ("                                          - 6", path_mcn6),
    ("                                          - 9", path_mcn9),
    ("                                          - 10", path_mcn10),
    ("                                          - 12", path_mcn12),
    ("                                          - 20", path_mcn20),
]

p = plot_optimization_paths(f, x_range, y_range, paths, title, global_min)

n_values = [0,1,2,3,4,5,6,9,10,12,20]
num_iterations = [length(path_mcn0)-1,length(path_mcn1)-1,length(path_mcn2)-1,length(path_mcn3)-1,length(path_mcn4)-1,length(path_mcn5)-1,length(path_mcn6),length(path_mcn9),length(path_mcn10),length(path_mcn12),length(path_mcn20)]

r = plot(n_values, num_iterations, marker=:circle, label="
                    ", legend=:topleft,
  xlabel="n (                              )", ylabel="
                    ",
  title="
              n")

#
```

```
    display(p)
    display(r)

    # ,

    return nothing
    end

#

# 1.
function rosenbrock(x)
    return (1.0 - x[1])^2 + 100.0*(x[2] - x[1]^2)^2
end

# 2.
function rastrigin(x)
    return 20 + x[1]^2 - 10*cos(2*pi*x[1]) + x[2]^2 - 10*cos(2*pi*x[2])
end

# 3.
function schwefel(x)
    return 418.9829*2 - (x[1]*sin(sqrt(abs(x[1]))) + x[2]*sin(sqrt(abs(x[2]))))
end

# 4.
a = 5.0
b = 6.0
function quadratic(x)
    return a * x[1]^2 + b * x[2]^2
end

#
x0_rosenbrock = [0.0, 0.0]
x_range_rosenbrock = -2.0:0.1:2.0
y_range_rosenbrock = -1.0:0.1:3.0
global_min_rosenbrock = [1.0, 1.0]
run_optimization(rosenbrock, x0_rosenbrock, "", x_range_rosenbrock, y_range_rosenbrock, global_min_rosenbrock)
```

3 Результаты

В качестве метода поиска оптимальных параметров был выбран алгоритм Нелдера-Мида.

Результаты запуска представлены на рисунках 1- 2.

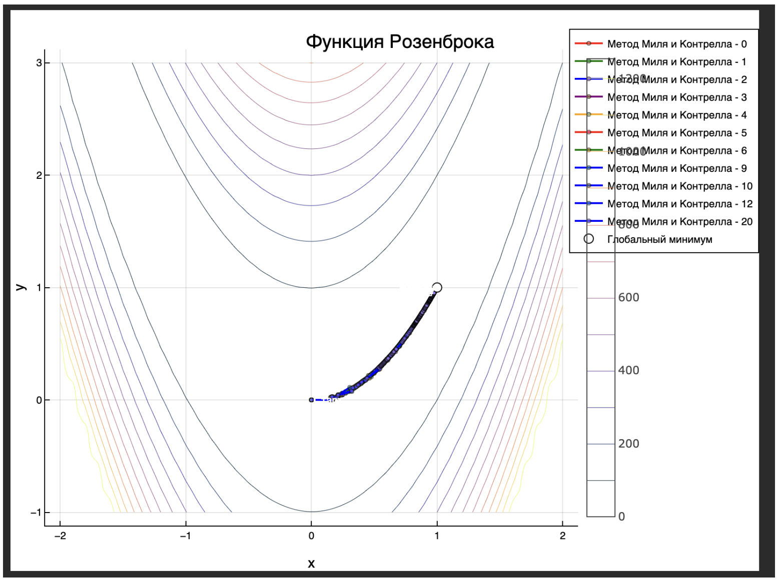

Рис. 1 — Функция Розенброка

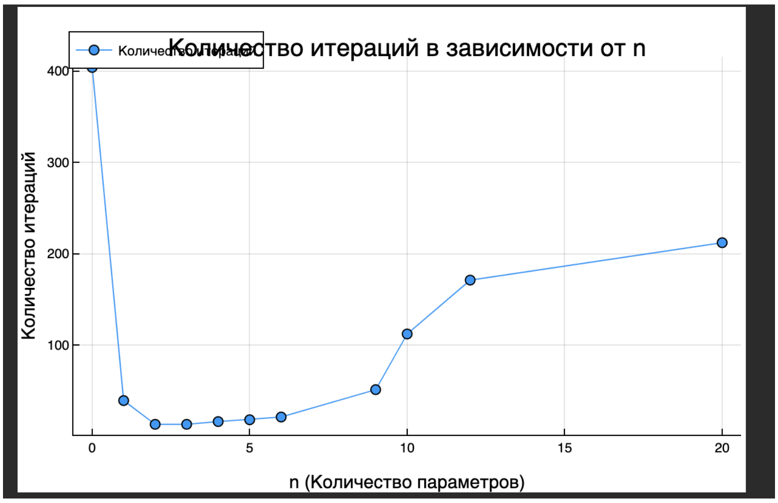

Рис. 2 — Функция Розенброка зависимость кол-во итераций от n

4 Выводы

В ходе выполнения лабораторной работы был применён метод Крэгга и Леви для исследования поиска на разных функциях. Была выявлена зависимость между числом итераций и количеством параметров на примере функции Розенброка. График показывает, что с добавлением первого параметра алгоритм ускоряется по сравнению с обычным градиентным спуском, достигая оптимума при 4 параметрах, после чего число итераций начинает медленно увеличиваться. Это объясняется накоплением шума из-за предыдущих шагов оптимизации, поскольку каждый из них является лишь приближённо оптимальным.