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 № 3

по курсу «Методы оптимизации»

«Симплексный алгоритм и алгоритм Нельдера-Мида»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

1. Реализовать симплексный метод поиска минимума.
2. Реализовать метод Нельдера-Мида для поиска минимума.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```

using Plots
using LinearAlgebra

plotly()

function simplex_method(x0)
    xs = []
    x1 = x0
    x2 = [(sqrt(3)+1)/(2*sqrt(2)), (sqrt(3)-1)/(2*sqrt(2))]
    x3 = [(sqrt(3)-1)/(2*sqrt(2)), (sqrt(3)+1)/(2*sqrt(2))]
    points = [x1,x2,x3]
    points[1] = x1
    points[2] = x2
    points[3] = x3
    push!(xs, x1)
    push!(xs, x2)
    push!(xs, x3)
    while true
        if(norm(points[1]- points[2])) < 0.001
            return points[1], xs
        end
        if(f(points[2]) >= f(points[1]) && f(points[2]) >= f(points[3]))
            temp = points[1]
            points[1] = points[2]
            points[2] = temp
        elseif (f(points[3]) >= f(points[1]) && f(points[3]) >= f(points[2]))
            temp = points[1]
            points[1] = points[3]
            points[3] = temp
        end
        x4 = points[2] + points[3] - points[1]
        if(f(x4) >= f(points[2]) && f(x4) >= f(points[3]))
```

```
                points[1] = x4
                points[2] = x4+(points[2] - x4)/2
                points[3] = x4+(points[3] - x4)/2
                push!(xs, x4)
                push!(xs, points[2])
                push!(xs, points[3])
            else
                points[1] = x4
                push!(xs, x4)
                push!(xs, points[2])
                push!(xs, points[3])
            end
        end
end

function nelder_meed(x0)
    x1 = x0
    x2 = [(sqrt(3)+1)/(2*sqrt(2)), (sqrt(3)-1)/(2*sqrt(2))]
    x3 = [(sqrt(3)-1)/(2*sqrt(2)), (sqrt(3)+1)/(2*sqrt(2))]
    points = [x1,x2,x3]
    points[1] = x1
    points[2] = x2
    points[3] = x3
    xs = []
    center = x0
    beta = 2.0
    while true
        points = sort(points, by=x -> f(x), rev=true)
        push!(xs, points[3])
        push!(xs, points[1])
        push!(xs, points[2])
        center = (points[2]+points[3])/2.0
        if (sqrt(((f(points[1]) - f(center))^2 +((f(points[2]) - f(center))^2) + ((f(points[2]) - f(center))^2))/(3.0)) < 0.001)
            return  points[3],xs
        end
        x4 = points[2]+points[3]-points[1]
        beta = 2.0
        y_min =  f(points[3])
        if(f(x4) < y_min)
            beta = 2.0
            x5 = beta*x4 + (1-beta)*center
            if(f(x5)< f(x4) && f(x5)< f(points[3]) && f(x5)< f(points[2]))
                points[1] = x5
            else
```

```
                if(f(x5) > f(x4))
                    points[1] = x4
                end
            end
        else
            if f(points[3]) < f(x4) < f(points[2])
                points[1] = x4
            else
                if f(points[2]) < f(x4) < f(points[1])
                    points[1] = x4
                end
                points = sort(points, by=x -> f(x), rev=true)
                beta = 0.5
                x5 = beta * points[1] + (1 - beta) * center
                if f(x5) < f(points[1])
                    points[1] = x5
                else
                    #c
                    points[1] = points[3] + 0.5 * (points[1] - points[3])
                    points[2] = points[3] + 0.5 * (points[2] - points[3])
                end
            end
        end
    end
end

f(x) = (1.0 - x[1])^2 + 100.0*(x[2] - x[1]^2)^2

x0 = [-1.0, -1.0]

x = -2.5:0.1:2.5
y = -2.5:0.1:2.5


plt = surface(x, y, (x, y) -> f([x, y]), color=:thermal, alpha=0.5, legend=true)

println("                                        ")
r, xs = simplex_method(x0)
println(xs)
println("                                              : ", r)

triangles_simplex = [xs[i:i+2] for i in 1:3:length(xs)]
x_coords_simplex = [x[1] for x in xs]
```

```
y_coords_simplex = [y[2] for y in xs]


res,xs = nelder_meed(x0)
println("          -      : ", res)

triangles_nm  = [xs[i:i+2] for i in 1:3:length(xs)]
x_coords_nm = [x[1] for x in xs]
y_coords_nm = [y[2] for y in xs]

println()

x = -2.5:0.1:2.5
y = -2.5:0.1:2.5

plt5 = surface(x, y, (x, y) -> f([x, y]), color=:thermal, alpha=0.5,
    legend=true)

Z = map((a,b) -> f([a,b]), x_coords_nm, y_coords_nm)

for (i, triangle) in enumerate(triangles_nm)
    x1 = [triangle[1][1], triangle[2][1], triangle[3][1], triangle
    [1][1]]  #                 x
    y1 = [triangle[1][2], triangle[2][2], triangle[3][2], triangle
    [1][2]]  #                 y
    z1 = [f([x1[1],y1[1]]),f([x1[2],y1[2]]), f([x1[3],y1[3]]),f([x1[4],
    y1[4]])]
    plot!(plt5,x1, y1,z1,linecolor=:blue)
end

scatter3d!(plt5, [x0[1]], [x0[2]], [f(x0)], color=:green, label="Start")
scatter3d!(plt5, [res[1]], [res[2]], [f(res)], color=:blue, label="End")

xlabel!(plt5, "x")
ylabel!(plt5, "y")
zlabel!(plt5, "f(x,y)")


plt6 = surface(x, y, (x, y) -> f([x, y]), color=:thermal, alpha=0.5,
    legend=true)

Z = map((a,b) -> f([a,b]), x_coords_simplex, y_coords_simplex)


scatter3d!(plt6, [x0[1]], [x0[2]], [f(x0)], color=:green, label="Start")
scatter3d!(plt6, [r[1]], [r[2]], [f(r)], color=:blue, label="End")
```

```

for (i, triangle) in enumerate(triangles_simplex)
    x1 = [triangle[1][1], triangle[2][1], triangle[3][1], triangle[1][1]]  #                 x
    y1 = [triangle[1][2], triangle[2][2], triangle[3][2], triangle[1][2]]  #                 y
    z1 = [f([x1[1],y1[1]]),f([x1[2],y1[2]]), f([x1[3],y1[3]]),f([x1[4],y1[4]])]
    plot!(plt6,x1, y1,z1,linecolor=:red)
end

xlabel!(plt6, "x")
ylabel!(plt6, "y")
zlabel!(plt6, "f(x,y)")

plt1 = contour(x, y, (x, y) -> f([x, y]),
    color=:thermal,
    levels=20,
    xlabel=" x  ",
    ylabel=" x  ",
    title="                                        (2D)",
    legend=:topleft,
    size=(700, 500)
)

colors = [:red, :green, :blue, :orange, :purple, :yellow]

for (i, triangle) in enumerate(triangles_nm)
    x1 = [triangle[1][1], triangle[2][1], triangle[3][1], triangle[1][1]]  #                 x
    y1 = [triangle[1][2], triangle[2][2], triangle[3][2], triangle[1][2]]  #                 y
    plot!(plt1,x1, y1, color=colors[i % length(colors) + 1], linewidth=2, label="Triangle $i")
end

scatter!(plt1,
    [x0[1], res[1]],
    [x0[2], res[2]],
    color=[:green :blue],
    markersize=8,
    label=["          " "          "]
)

plt2 = contour(x, y, (x, y) -> f([x, y]),
    color=:thermal,
```

```
    levels=20,
    xlabel=" x  ",
    ylabel=" x  ",
    title="                                (2D)",
    legend=:topleft,
    size=(700, 500)
)

colors = [:red, :green, :blue, :orange, :purple, :yellow]

scatter!(plt2,
    [x0[1], r[1]],
    [x0[2], r[2]],
    color=[:green :blue],
    markersize=8,
    label=["          " "          "]
)

for (i, triangle) in enumerate(triangles_simplex)
    x1 = [triangle[1][1], triangle[2][1], triangle[3][1], triangle[1][1]]  #                x
    y1 = [triangle[1][2], triangle[2][2], triangle[3][2], triangle[1][2]]  #                y
    plt2 = plot!(x1, y1, color=colors[i % length(colors) + 1], linewidth=2, label="Triangle $i")
end

display(plt6)
display(plt2)
display(plt5)
display(plt1)

f(x) = 20 + x[1]^2 - 10*cos(2*pi*x[1]) + x[2]^2 - 10*cos(2*pi*x[2])

x0 = [0.5, 0.5]

x = -2.5:0.1:2.5
y = -2.5:0.1:2.5


plt = surface(x, y, (x, y) -> f([x, y]), color=:thermal, alpha=0.5, legend=true)

println("                         ")
r,xs = simplex_method(x0)
println("                                 : ", r)
```

```
triangles_simplex  = [xs[i:i+2] for i in 1:3:length(xs)]
x_coords_simplex = [x[1] for x in xs]
y_coords_simplex = [y[2] for y in xs]

res,xs = nelder_meed(x0)
println("                -        : ", res)

triangles_nm  = [xs[i:i+2] for i in 1:3:length(xs)]
x_coords_nm = [x[1] for x in xs]
y_coords_nm = [y[2] for y in xs]

println()

f(x) = 418.9829*2 - (x[1]*sin(sqrt(abs(x[1]))) + x[2]*sin(sqrt(abs(x[2]))))

x0 = [0.0, 0.0]

x = -6:0.1:6
y = -6:0.1:6


plt = surface(x, y, (x, y) -> f([x, y]), color=:thermal, alpha=0.5, legend=true)
display(plt)


println("                                  ")
r,xs = simplex_method(x0)
println("                                                    : ", r)
triangles_simplex  = [xs[i:i+2] for i in 1:3:length(xs)]
x_coords_simplex = [x[1] for x in xs]
y_coords_simplex = [y[2] for y in xs]

res,xs = nelder_meed(x0)
println("                -        : ", res)

triangles_nm  = [xs[i:i+2] for i in 1:3:length(xs)]
x_coords_nm = [x[1] for x in xs]
y_coords_nm = [y[2] for y in xs]

println()
```

3 Результаты

Результаты запуска представлены на рисунках 1 - ??.

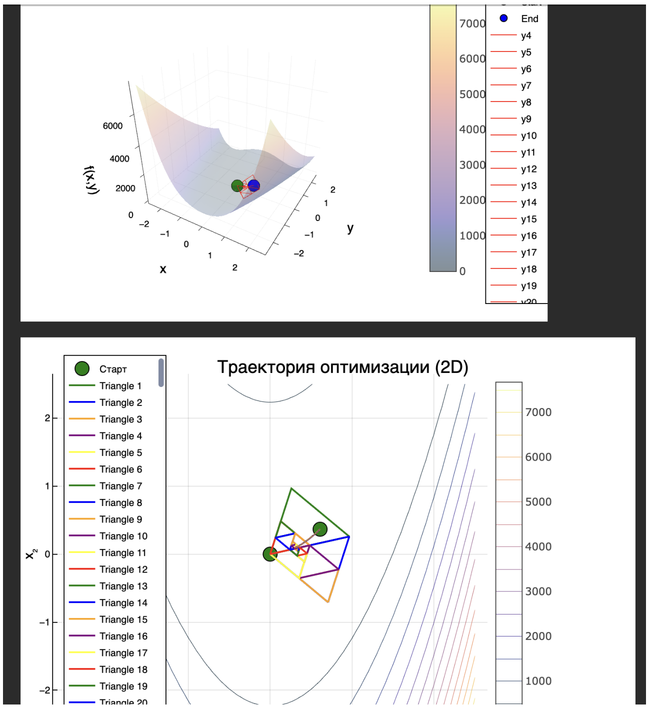

Рис. 1 — Симплексный метод (Розенброк)

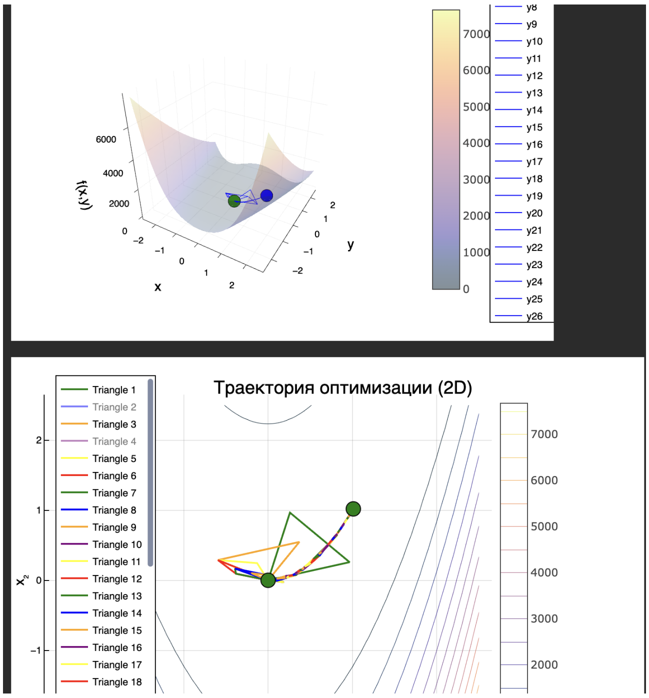

Рис. 2 — Метод Нельдера-Мида (Розенброк)

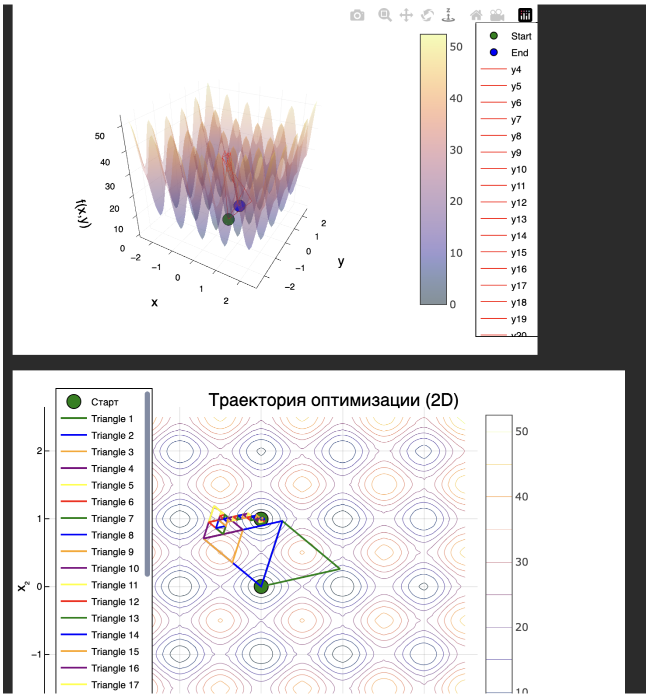

Рис. 3 — Симплексный метод (Растригин)

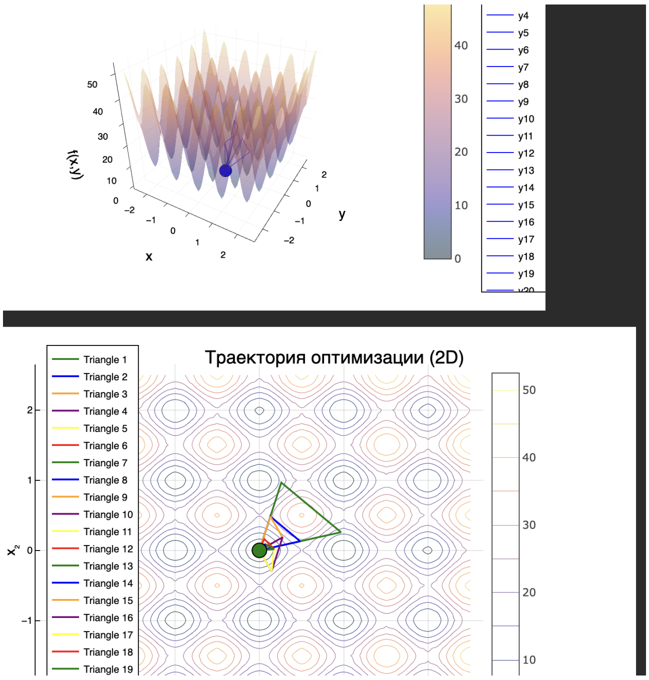

Рис. 4 — Метод Нельдера-Мида (Растригин)

4 Выводы

В ходе выполнения лабораторной работы были реализованы и сравнены методы оптимизации многомерных функций. Поиск экстремума был реализован с помощью следующих методов: метод Нельдера-Мида, симплексный алгоритм.

Наилучший результат показал метод Нельдера-Мида, который использует растяжение и редукцию симплексов.