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 №4

“Исследование методов локальной оптимизации”

ПО КУРСУ:

«Методы оптимизация»

Студент ИУ9-82Б Потребина В. В.

Преподаватель Посевин Д. П.

Москва, 2024 г.

ОГЛАВЛЕНИЕ

1. Постановка задачи

Цели:

Цель данной лабораторной работы — изучение и реализация методов локальной оптимизации, предназначенных для поиска минимума многомерных функций без использования градиента. В рамках эксперимента проводится сравнение двух методов:

Простой симплексный метод (метод Нелдера–Мида без дополнительных расширений).

Метод Нелдера–Мида (более сложный алгоритм, включающий операции сжатия и расширения симплекса).

Формальная постановка задачи

- **Дано**: функция $f(x)$
- **Требуется**: найти минимум функции с заданной точностью ϵ с использованием **метода симплексного поиска** и **метода Нелдера-Мида**.
- **Входные параметры**:
    - Начальная точка $x0$ — исходное приближение к минимуму.
    - Параметр сходимости ϵ— допустимая ошибка.
    - Максимальное число итераций.
- **Выходные параметры**:
    - Найденная точка минимума $x*$.
    - Количество итераций до сходимости.
    - Визуализация процесса оптимизации.

2. Практическая реализация

```julia
using Plots
using PlotlyJS
using SpecialFunctions

# Доп функции
function norm(p)
    return sqrt(sum(p .* p))
end


# ===================== Целевая функция =====================
function target_ravine(p::Vector{Float64})
    return sum(p .* p)
end

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


# ===================== Простой симплексный метод =====================
function simplex_method(f; tol=1e-3, maxiter=10_000)
    x1 = [0.0, 0.0]
    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]

    # Вектор истории
    xs = [x1, x2, x3]

    iter = 0
    while iter < maxiter
```

```
        iter += 1

        # --- [добавили] Вывод логов на первых двух итерациях ---
        if iter <= 2
            println("=== Итерация $iter (до сортировки и отражения) ===")
            println("points[1] = $(points[1])   f(points[1]) = ", f(points[1]))
            println("points[2] = $(points[2])   f(points[2]) = ", f(points[2]))
            println("points[3] = $(points[3])   f(points[3]) = ", f(points[3]))
        end

        # Проверка сходимости по расстоянию между вершинами
        if norm(points[1] - points[2]) < tol &&
           norm(points[2] - points[3]) < tol
            return (points[1], xs, iter)
        end

        # Сортируем так, чтобы points[1] была точкой с наибольшим значением f
        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

        # Отражённая точка
        x4 = points[2] + points[3] - points[1]

        # Если нужно ещё и вывод после сортировки — тоже можно добавить:
        if iter <= 2
            println("После сортировки (худшая точка в points[1]):")
            println("points[1] = $(points[1])   f(points[1]) = ", f(points[1]))
            println("points[2] = $(points[2])   f(points[2]) = ", f(points[2]))
            println("points[3] = $(points[3])   f(points[3]) = ", f(points[3]))
            println("Отражённая точка x4 = $x4   f(x4) = ", f(x4))
        end

        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

        if iter <= 2
            println("После шага обновления:")
            println("points[1] = $(points[1])   f(points[1]) = ", f(points[1]))
            println("points[2] = $(points[2])   f(points[2]) = ", f(points[2]))
            println("points[3] = $(points[3])   f(points[3]) = ", f(points[3]))
            println("-----------------------------------------------------")
        end
    end

    # Если зашли в предел maxiter, возвращаем то, что есть
    return (points[1], xs, iter)
end


# ===================== Метод Нелдера–Мида =====================
function nelder_meed(f; tol=1e-3, maxiter=10_000)
    x1 = [0.0, 0.0]
    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]

    # Вектор истории
    xs = [x1, x2, x3]

    iter = 0
    while iter < maxiter
        iter += 1
        # Сортируем вершины по убыванию f
        points = sort(points, by = x -> f(x), rev=true)

        # Запоминаем вершины (чтобы видеть «прыжки» в истории)
        push!(xs, points[1])
        push!(xs, points[2])
        push!(xs, points[3])

        # Центр (средняя «лучших» вершин)
        center = (points[2] + points[3]) ./ 2

        # Критерий остановки (например, дисперсия значений f)
        fx1, fx2, fx3 = f(points[1]), f(points[2]), f(points[3])
        fcenter = f(center)
        variance = ((fx1 - fcenter)^2 + (fx2 - fcenter)^2 + (fx3 - fcenter)^2) / 3
        if sqrt(variance) < tol
            return (points[3], xs, iter)
        end
```

```
        # Отражённая точка
        x4 = points[2] + points[3] - points[1]

        # Параметр расширения
        beta = 2.0
        if f(x4) < f(points[3])  # x4 лучше «лучшей» вершины
            # Пробуем расширить
            x5 = beta*x4 + (1 - beta)*center
            if f(x5) < f(x4)
                points[1] = x5
            else
                points[1] = x4
            end
        else
            # Если x4 не улучшил, то пробуем другие операции сжатия
            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
                # Дополнительное сжатие
                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
                    points[1] = points[3] .+ 0.5 .* (points[1] .- points[3])
                    points[2] = points[3] .+ 0.5 .* (points[2] .- points[3])
                end
            end
        end
    end

    return (points[3], xs, iter)
end

# ===================== Функции построения графиков =====================
function plot_method(f, points, func_min, iterations, title)
    n = 1
    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=(800, 600), title=title)
    # Рисуем контур уровней
    Plots.plot!(plt, x, y, (xx,yy)->f([xx,yy]),
```

```
                st = :contour, levels=:40, fill=false, cbar=false)
    # Точки + ломаная
    Plots.plot!(plt, px, py, seriestype=:scatter, color="blue", label="Траектория")
    Plots.plot!(plt, px, py, color="blue", label="")
    # Отмечаем найденный минимум красной точкой
    Plots.plot!(plt, [func_min[1]], [func_min[2]], seriestype=:scatter,
color="red", label="min")
    # Отображаем кол-во итераций
    Plots.plot!(plt, [], [], labels="iterations = $iterations")
    return plt
end

function plot_method_dim(f, points, func_min, func_title)
    n = 7
    x_vals = range(-n, n, length=50)
    y_vals = range(-n, n, 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=0.7,
                            name="Surface"),
            PlotlyJS.scatter3d(
                x=first.(points),
                y=last.(points),
                z=[f(p) for p in points],
                mode="markers+lines",
                marker=attr(size=4, color="red"),
                name="Path"
            ),
            PlotlyJS.scatter3d(
                x=[func_min[1]],
                y=[func_min[2]],
                z=[f(func_min)],
                mode="markers",
                marker=attr(size=6, color="blue"),
                name="Minimum"
            )
        ],
        # Layout=PlotlyJS.Layout(
        #     title=func_title,
        #     scene_camera=attr(eye=attr(x=1.8, y=1.8, z=1.0))
        # )
    )
    return display(surface_plot)
end

# ===================== Общая функция optimize_func =====================
```

```
"""
    optimize_func(func, x0, h0, title="")

Запускает два метода (простой симплексный и Нелдера–Мида) на функции `func`.
Аргументы `x0` и `h0` здесь не используются напрямую (они нужны
только для совместимости с вашим шаблоном), но при желании
можно модифицировать методы, чтобы инициализировать начальные точки.
"""
function optimize_func(func, title="")
    if !isempty(title)
        println("====== $title ======")
    end

    # Запуск методов
    (final_simp, xs_simp, iter_simp) = simplex_method(func)
    (final_neld, xs_neld, iter_neid) = nelder_meed(func)

    # Печать результатов
    println("Simplex method:    k = $(iter_simp)  x_min = $(final_simp)  f(x_min) =
", func(final_simp))
    println("Nelder-Mead:       k = $(iter_neid)  x_min = $(final_neld)  f(x_min) =
", func(final_neld))
    println("==================================================\n")

    n = 3
    x_vals = range(-n, n, length=50)
    y_vals = range(-n, n, 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=0.5),
            PlotlyJS.scatter3d(x=first.(xs_simp), y=last.(xs_simp), z=[func(p) for
p in xs_simp],
                             mode="markers+lines", marker=attr(size=5,
color="red"),
                             name="Метод симплекс"),
            PlotlyJS.scatter3d(x=first.(xs_neld), y=last.(xs_neld), z=[func(p) for
p in xs_neld],
                             mode="markers+lines", marker=attr(size=5,
color="blue"),
                             name="Метод Н-М"),
        ]
    )
    p1 = plot_method(func, xs_simp, final_simp, iter_simp, "Simplex")
    p2 = plot_method(func, xs_neld, final_neld, iter_neid, "Nelder-Mead")
    display(Plots.plot(p1, p2))
    return display(p)
```

```
end

# ===================== Пример вызова =====================
optimize_func(target_ravine, "target_ravine")
# optimize_func(target_rastrygin, "target_rastrygin")
# optimize_func(target_rosenbrock, "target_rosenbrock")
# optimize_func(target_schefill, "target_schefill")

readline()
```

3. Результаты

Тестирование функционала было проведено на функции параболоида.

====== target_ravine ======

=== Итерация 1 (до сортировки и отражения) ===

points[1] = [0.0, 0.0]   f(points[1]) = 0.0

points[2] = [0.9659258262890682, 0.2588190451025207]   f(points[2]) = 0.9999999999999998

points[3] = [0.2588190451025207, 0.9659258262890682]   f(points[3]) = 0.9999999999999998

После сортировки (худшая точка в points[1]):

points[1] = [0.9659258262890682, 0.2588190451025207]   f(points[1]) = 0.9999999999999998

points[2] = [0.0, 0.0]   f(points[2]) = 0.0

points[3] = [0.2588190451025207, 0.9659258262890682]   f(points[3]) = 0.9999999999999998

Отражённая точка x4 = [-0.7071067811865475, 0.7071067811865475]   f(x4) = 0.9999999999999998

После шага обновления:

points[1] = [-0.7071067811865475, 0.7071067811865475]   f(points[1]) = 0.9999999999999998

points[2] = [-0.35355339059327373, 0.35355339059327373]   f(points[2]) = 0.24999999999999994

points[3] = [-0.22414386804201336, 0.8365163037378078]   f(points[3]) = 0.7499999999999999

=== Итерация 2 (до сортировки и отражения) ===

points[1] = [-0.7071067811865475, 0.7071067811865475]   f(points[1]) = 0.9999999999999998

points[2] = [-0.35355339059327373, 0.35355339059327373]   f(points[2]) = 0.24999999999999994

points[3] = [-0.22414386804201336, 0.8365163037378078]   f(points[3]) = 0.7499999999999999

После сортировки (худшая точка в points[1]):

points[1] = [-0.7071067811865475, 0.7071067811865475]   f(points[1]) = 0.9999999999999998

points[2] = [-0.35355339059327373, 0.35355339059327373]   f(points[2]) = 0.24999999999999994

points[3] = [-0.22414386804201336, 0.8365163037378078]   f(points[3]) = 0.7499999999999999

Отражённая точка x4 = [0.12940952255126037, 0.4829629131445341]   f(x4) = 0.24999999999999997

После шага обновления:

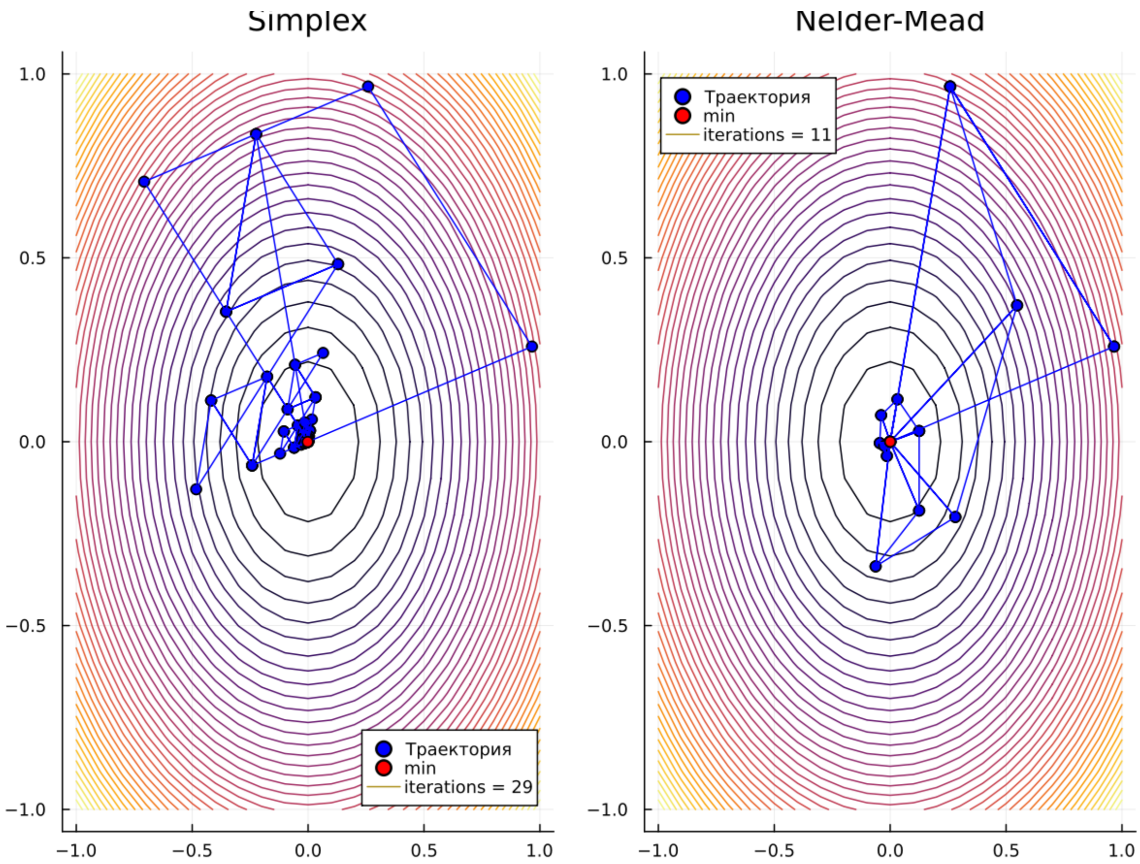
points[1] = [0.12940952255126037, 0.4829629131445341]   f(points[1]) = 0.24999999999999997

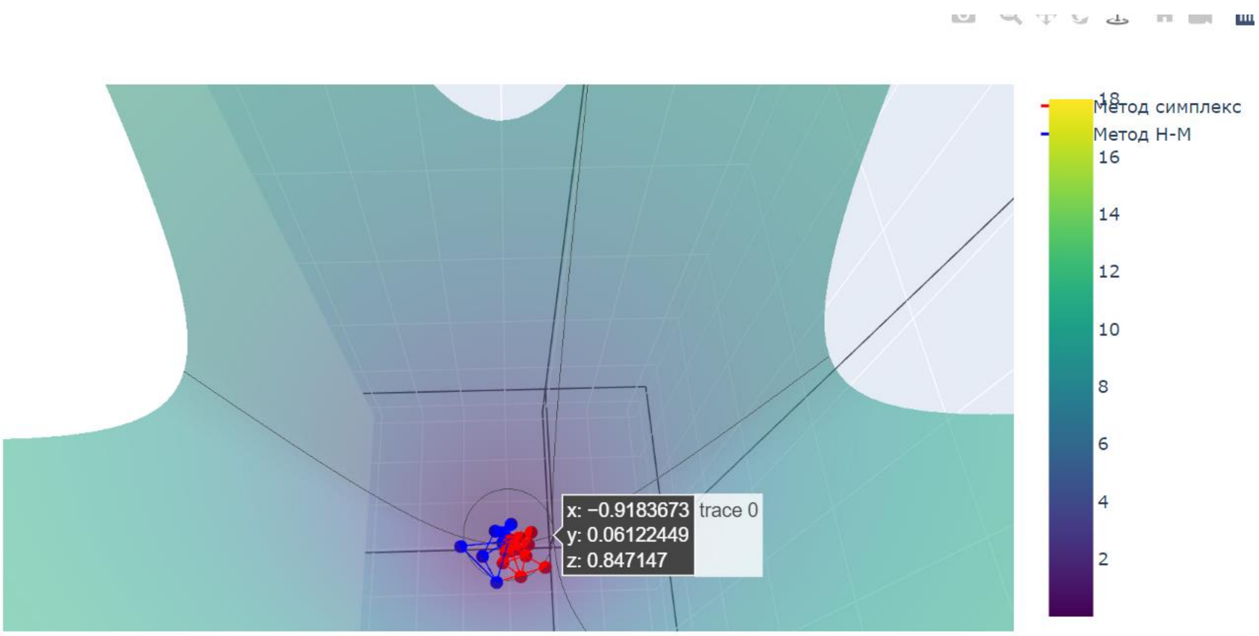
points[2] = [-0.35355339059327373, 0.35355339059327373]   f(points[2]) = 0.24999999999999994 points[3] = [-0.22414386804201336, 0.8365163037378078]   f(points[3]) = 0.7499999999999999

Simplex method: k = 29 x_min = [-0.0018865738794708363, -0.0005055059474658608] f(x_min) = 3.8146972656249996e-6

Nelder-Mead: k = 11 x_min = [0.0, 0.0] f(x_min) = 0.0

==





4. Выводы

В ходе лабораторной работы успешно реализованы два метода локальной оптимизации. Эксперименты показали, что метод Нелдера-Мида более эффективен, чем классический симплексный метод, и быстрее достигает минимума. Однако оба метода являются рабочими инструментами для решения задач оптимизации в многомерных пространствах.