|  |  |
| --- | --- |
|  | **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 3](#_Toc192150468)

[2. Практическая реализация 4](#_Toc192150469)

[3. Результаты 10](#_Toc192150470)

[4. Выводы 13](#_Toc192150471)

# 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

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

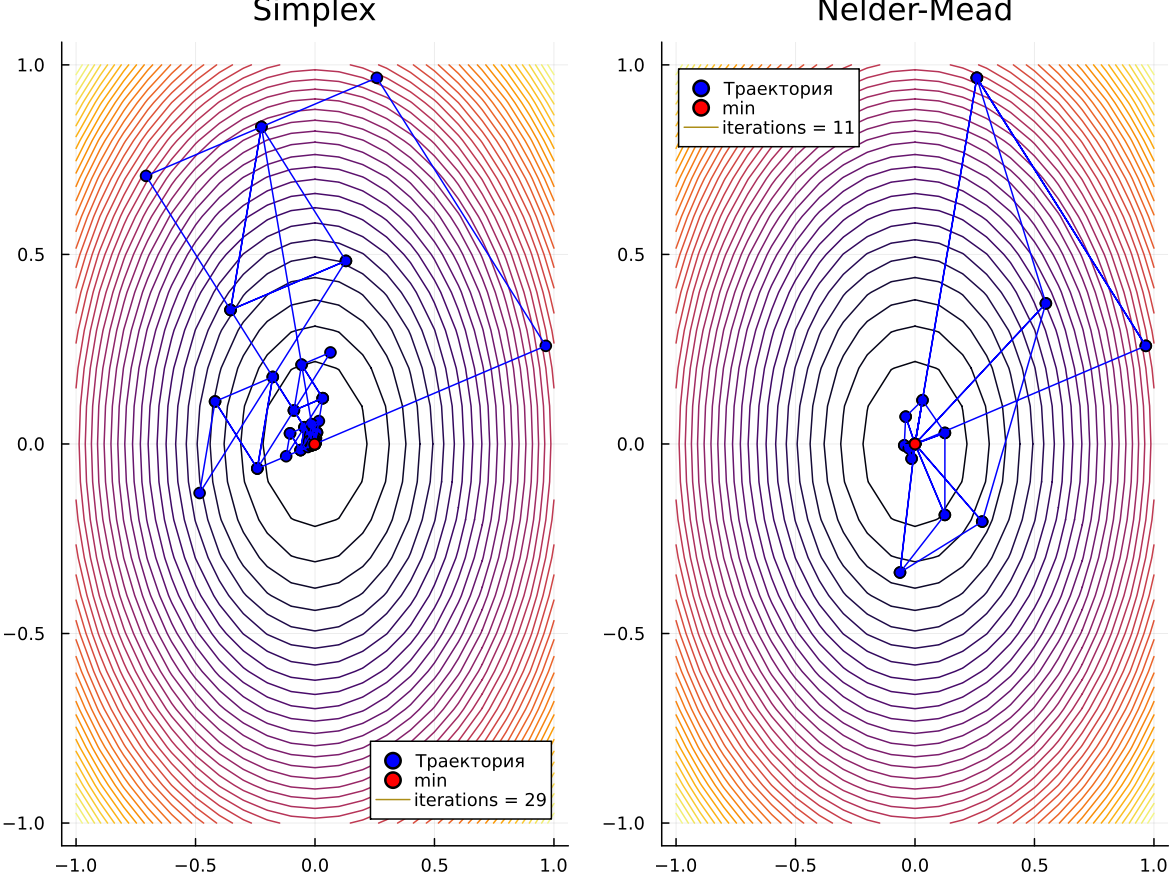
points[3] = [-0.22414386804201336, 0.8365163037378078] f(points[3]) = 0.7499999999999999

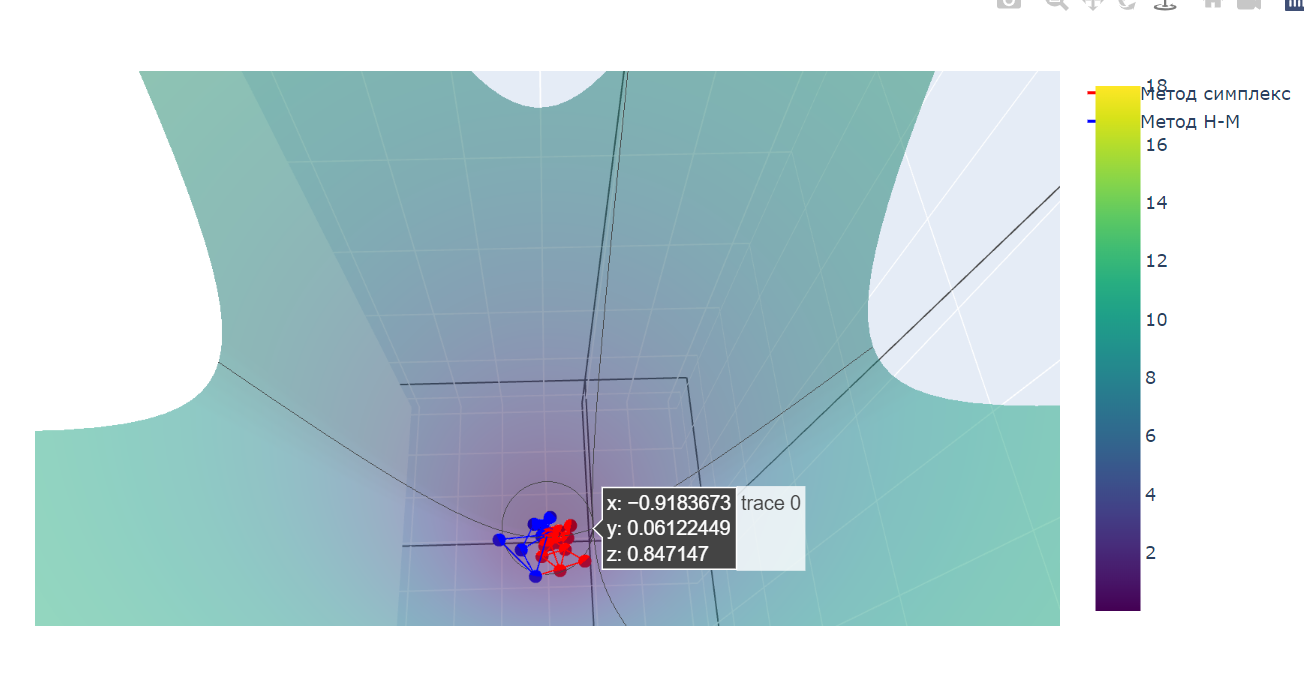
-----------------------------------------------------

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.