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 №6

“Градиентный спуск”

ПО КУРСУ:

«Методы оптимизация»

Студент ИУ9-82Б Потребина В. В.

Преподаватель Посевин Д. П.

Москва, 2024 г.

ОГЛАВЛЕНИЕ

1. Цели

Цель данной лабораторной работы — **реализовать и исследовать методы градиентной оптимизации** для нахождения минимума функций. Рассматриваются три метода:

1. **Метод градиентного спуска (Gradient Descent, GD)**
   - Движение в направлении **отрицательного градиента** с фиксированным шагом.
   - Используется **адаптивное уменьшение шага**, если функция не уменьшается.
2. **Метод наискорейшего спуска (Steepest Descent, SD)**
   - Подбор **оптимального шага** α\alphaα на каждом шаге **методом золотого сечения**.
   - Позволяет **быстрее сходиться**, но требует дополнительных вычислений.
3. **Метод сопряжённых градиентов (Conjugate Gradient, CG)**
   - Улучшает стандартный градиентный спуск, **учитывая прошлые направления**.
   - Сходится **значительно быстрее** на квадратичных функциях.

Исследуется **сходимость** каждого метода на **различных тестовых функциях**.

2. Практическая реализация

```
using Plots

function norm(p)
    return sqrt(sum(p .* p))
end

# Функция для численного вычисления градиента методом конечных разностей
function numerical_gradient(f, x, h=1e-5)
    grad = zeros(length(x))
    for i in eachindex(x)
        x_step = copy(x)
        x_step[i] += h
        grad[i] = (f(x_step) - f(x)) / h
    end
```

```
    return grad
end

# Метод золотого сечения для одномерной оптимизации
function golden_section_search(f, a, b, tol=1e-5)
    ϕ = (sqrt(5) - 1) / 2  # Коэффициент золотого сечения
    c = b - (b - a) * ϕ
    d = a + (b - a) * ϕ
    while abs(b - a) > tol
        if f(c) < f(d)
            b = d
        else
            a = c
        end
        c = b - (b - a) * ϕ
        d = a + (b - a) * ϕ
    end
    return (a + b) / 2  # Оптимальное значение α
end

function gradient_descent(f, x0; α=0.1, tol_x=1e-5, tol_f=1e-5, maxiter=1000)
    x = x0  # Текущая точка
    history = [x]  # История точек
    α_init = α  # Сохраняем начальный шаг

    for k in 1:maxiter
        g = numerical_gradient(f, x)  # Вычисляем градиент вручную

        if norm(g) < tol_f  # Проверка ||∇f(x)|| ≤ ε3
            println("Метод сошелся по норме градиента на $k итерации")
            break
        end

        x_new = x - α * g  # Градиентный шаг

        # Откат шага, если f(x_new) >= f(x)
        while f(x_new) >= f(x)
            α /= 2  # Уменьшаем шаг
            x_new = x - α * g
        end

        # Условие остановки по изменениям
        if norm(x_new - x) < tol_x || abs(f(x_new) - f(x)) < tol_f
            println("Метод сошелся по критериям остановки на $k итерации")
            break
        end

        x = x_new
        α = α_init  # Сбрасываем α к начальному значению
```

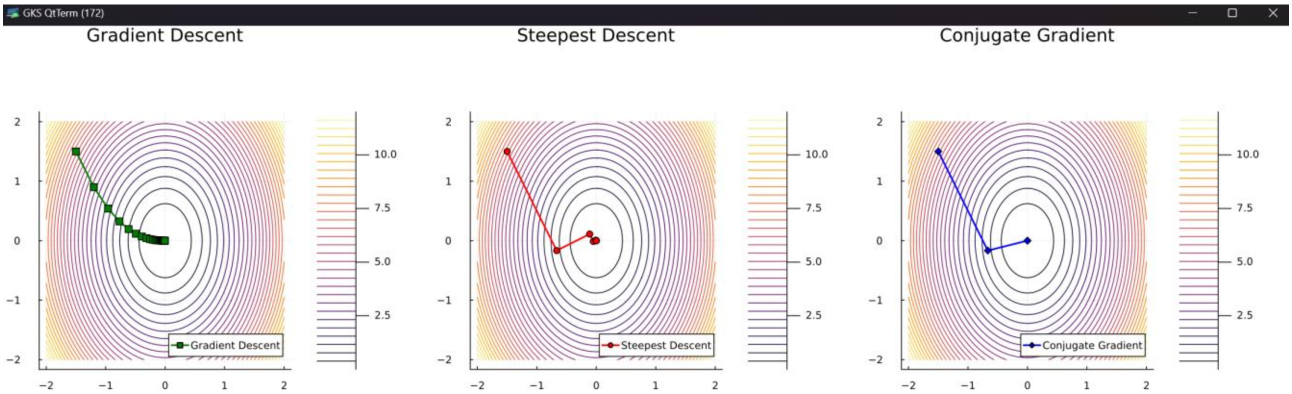
```
        push!(history, x)
    end

    return x, history
end

# Метод наискорейшего спуска
function steepest_descent(f, x0; tol=1e-5, maxiter=500)
    x = x0
    history = [x]

    for k in 1:maxiter
        g = numerical_gradient(f, x)

        if norm(g) < tol
            println("Метод сошелся по градиенту на $k итерации")
            break
        end

        # Ограничиваем диапазон поиска α
        α_func(α) = f(x - α * g)
        α_opt = golden_section_search(α_func, 1e-4, 1)

        x_new = x - α_opt * g

        if norm(x_new - x) < tol
            println("Метод сошелся по изменениям координат на $k итерации")
            break
        end

        x = x_new
        push!(history, x)
    end

    return x, history
end


# Метод сопряженных градиентов
function conjugate_gradient(f, x0; tol=1e-5, maxiter=1000)
    x = x0
    history = [x]

    g = numerical_gradient(f, x)
    p = -g

    for k in 1:maxiter
        if norm(g) < tol
            println("Метод сопряженных градиентов сошелся на $k итерации")
```

```
            break
        end

        # Оптимальный шаг α
        α_func(α) = f(x + α * p)
        α_opt = golden_section_search(α_func, 0, 1)

        x_new = x + α_opt * p
        g_new = numerical_gradient(f, x_new)

        if norm(x_new - x) < tol
            println("Метод сошелся по изменениям координат на $k итерации")
            break
        end

        # Коэффициент β
        β = (norm(g_new)^2) / (norm(g)^2)
        p = -g_new + β * p

        x, g = x_new, g_new
        push!(history, x)
    end

    return x, history
end

# Целевые функции для тестирования
function target_quadratic(x)
    return x[1]^2 + 2*x[2]^2
end

function target_rosenbrock(x)
    return (1 - x[1])^2 + 100 * (x[2] - x[1]^2)^2
end

# Функция для визуализации траектории спуска
function plot_descent(f, history; xmin=-2, xmax=2, ymin=-2, ymax=2)
    x = range(xmin, xmax, length=100)
    y = range(ymin, ymax, length=100)
    Z = [f([xi, yi]) for xi in x, yi in y]

    plt = contour(x, y, Z, levels=30, title="Траектория метода")
    traj_x = [p[1] for p in history]
    traj_y = [p[2] for p in history]

    plot!(plt, traj_x, traj_y, marker=:circle, color=:red, linewidth=2,
label="Траектория")
    display(plt)
end
```

```julia
# Универсальная функция для тестирования методов
function optimize_func(func, title="")
    if !isempty(title)
        println("====== $title ======")
    end

    x0 = [-1.5, 1.5]  # Начальная точка

    # Запуск методов
    (final_gd, history_gd) = gradient_descent(func, x0)  # Градиентный спуск
    (final_sd, history_sd) = steepest_descent(func, x0)  # Наискорейший спуск
    (final_cg, history_cg) = conjugate_gradient(func, x0)  # Сопряженные градиенты

    # Вывод результатов
    println("Gradient Descent:   k = $(length(history_gd))  x_min = 
$(final_gd)  f(x_min) = ", func(final_gd))
    println("Steepest Descent:   k = $(length(history_sd))  x_min = 
$(final_sd)  f(x_min) = ", func(final_sd))
    println("Conjugate Gradient: k = $(length(history_cg))  x_min = 
$(final_cg)  f(x_min) = ", func(final_cg))
    println("==================================================\n")

    # Определяем одинаковые границы для осей
    xmin, xmax = -2, 2
    ymin, ymax = -2, 2

    # --- Создание трех графиков ---
    plt1 = contour(range(xmin, xmax, length=100), range(ymin, ymax, length=100),
                   [func([xi, yi]) for xi in range(xmin, xmax, length=100), yi in 
range(ymin, ymax, length=100)],
                   levels=30, title="Gradient Descent", aspect_ratio=:equal)

    plt2 = contour(range(xmin, xmax, length=100), range(ymin, ymax, length=100),
                   [func([xi, yi]) for xi in range(xmin, xmax, length=100), yi in 
range(ymin, ymax, length=100)],
                   levels=30, title="Steepest Descent", aspect_ratio=:equal)

    plt3 = contour(range(xmin, xmax, length=100), range(ymin, ymax, length=100),
                   [func([xi, yi]) for xi in range(xmin, xmax, length=100), yi in 
range(ymin, ymax, length=100)],
                   levels=30, title="Conjugate Gradient", aspect_ratio=:equal)

    # Траектория метода градиентного спуска (Gradient Descent)
    if !isempty(history_gd)
        traj_x_gd = [p[1] for p in history_gd]
        traj_y_gd = [p[2] for p in history_gd]
        plot!(plt1, traj_x_gd, traj_y_gd, marker=:square, color=:green, 
linewidth=2, label="Gradient Descent")
```

```
    else
        println("Ошибка: пустая история для метода градиентного спуска")
    end

    # Траектория метода наискорейшего спуска (Steepest Descent)
    if !isempty(history_sd)
        traj_x_sd = [p[1] for p in history_sd]
        traj_y_sd = [p[2] for p in history_sd]
        plot!(plt2, traj_x_sd, traj_y_sd, marker=:circle, color=:red, linewidth=2,
label="Steepest Descent")
    else
        println("Ошибка: пустая история для метода наискорейшего спуска")
    end

    # Траектория метода сопряженных градиентов (Conjugate Gradient)
    if !isempty(history_cg)
        traj_x_cg = [p[1] for p in history_cg]
        traj_y_cg = [p[2] for p in history_cg]
        plot!(plt3, traj_x_cg, traj_y_cg, marker=:diamond, color=:blue,
linewidth=2, label="Conjugate Gradient")
    else
        println("Ошибка: пустая история для метода сопряженных градиентов")
    end

    # --- Отображение трех графиков на одном полотне ---
    display(plot(plt1, plt2, plt3, layout=(1,3), size=(1500, 500)))
end



# Запуск эксперимента
optimize_func(target_quadratic, "Минимизация квадратичной функции")
readline()
```

3. Результаты



====== Минимизация квадратичной функции ======

Метод сошелся по критериям остановки на 27 итерации

Метод сошелся по градиенту на 11 итерации

Метод сошелся по изменениям координат на 4 итерации

Gradient Descent: k = 27 x_min = [-0.004538456711940925, -2.4412655482112337e-6] f(x_min) = 2.0597601245716587e-5

Steepest Descent: k = 11 x_min = [-1.6117663123145467e-6, -6.480540234462138e-6] f(x_min) = 8.659259410647719e-11

Conjugate Gradient: k = 4 x_min = [-1.670250051067889e-6, -5.397706541574913e-7] f(x_min) = 3.372439951271498e-12

==

4. Выводы

В ходе лабораторной работы были **реализованы и протестированы три метода градиентной оптимизации**:

1. **Метод градиентного спуска (Gradient Descent, GD)**
   - Использует фиксированный шаг α, который уменьшается, если функция не убывает.
   - **Сильная зависимость от выбора шага α**.
   - **Медленная сходимость на сложных функциях** (например, на функции Розенброка).
2. **Метод наискорейшего спуска (Steepest Descent, SD)**
   - Оптимальный шаг α вычисляется **методом золотого сечения**.
   - Улучшенная сходимость по сравнению с обычным градиентным спуском.
   - **Долгие вычисления из-за дополнительной одномерной оптимизации на каждом шаге**.
3. **Метод сопряжённых градиентов (Conjugate Gradient, CG)**
   - Улучшает стандартный градиентный спуск за счёт **использования направлений, учитывающих прошлые итерации**.
   - **Самая быстрая сходимость на квадратичных и гладких нелинейных функциях**.
   - **Минимальное число итераций по сравнению с другими методами.**