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»

Летучка № 5

по курсу «Методы оптимизации»

«Реализация генетического алгоритма для функции 1 переменной»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать генетический алгоритм для поиска минимума функции 1 переменной, визуализировать процесс поиска на графиках.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```


using Plots
using Random
using Statistics

function generate_population(size, rrange)
    r1, r2 = rrange
    return [rand() * (r2 - r1) + r1 for _ in 1:size]
end

function mutate(x, mutation_rate, rrange)
    r1, r2 = rrange
    if rand() < mutation_rate
        return rand() * (r2 - r1) + r1
    end
    return x
end

function select(population, fitness, number_of_parents)
    scores = [fitness(x) for x in population]
    sorted_idx = sortperm(scores)
    return population[sorted_idx[1:number_of_parents]]
end

#

function binary_single_point_crossover(parent1, parent2, x_range)
    #

    p1_bits = bitstring(Float64(parent1))
    p2_bits = bitstring(Float64(parent2))

    #
```

```
    crossover_point = rand(1:length(p1_bits))

    #
    c1_bits = p1_bits[1:crossover_point] * p2_bits[crossover_point+1:end]
    c2_bits = p2_bits[1:crossover_point] * p1_bits[crossover_point+1:end]

    #
    child1 = parent1
    child2 = parent2

    try
        child1 = reinterpret(Float64, parse(UInt64, c1_bits, base=2))
        child2 = reinterpret(Float64, parse(UInt64, c2_bits, base=2))
    catch
        #                                                        ,
    end

    #
    if isnan(child1) || isinf(child1)
        child1 = parent1
    end
    if isnan(child2) || isinf(child2)
        child2 = parent2
    end

    #
    child1 = clamp(child1, x_range[1], x_range[2])
    child2 = clamp(child2, x_range[1], x_range[2])

    return child1, child2, crossover_point
end

#
function crossover(parents, number_of_children, crossover_type, x_range)
    children = []
    crossover_points = []  #

    if crossover_type == "average"
        #
```

```
        children = [(rand(parents) + rand(parents)) / 2 for _ in 1:number_of_children]
        crossover_points = zeros(Int, number_of_children) #

    elseif crossover_type == "binary"
        #
        while length(children) < number_of_children
            #
            p1, p2 = rand(parents), rand(parents)

            #
            c1, c2, cp = binary_single_point_crossover(p1, p2, x_range)

            push!(children, c1)
            push!(crossover_points, cp)

            if length(children) < number_of_children
                push!(children, c2)
                push!(crossover_points, cp)
            end
        end
    else
        error(": $crossover_type")
    end

    return children, crossover_points
end

function genetic_algorithm(fitness, generations, population_size, x_range, mutation_rate, crossover_type="average", name="test")
    Random.seed!(123) # seed
    population = generate_population(population_size, x_range)
    best_idx = argmin([fitness(x) for x in population])
    best_solution = population[best_idx]
    best_score = fitness(best_solution)

    #
    best_history = [(best_solution, best_score)]

    #
    anim = Animation()
```

```
    x_vals = range(x_range[1], x_range[2], length=200)
    y_vals = fitness.(x_vals)
    y_min, y_max = minimum(y_vals), maximum(y_vals)
    plot_range = (y_min - 0.1*(y_max-y_min), y_max + 0.1*(y_max-y_min))

    #
    plt = plot(x_vals, y_vals,
                color=:blue, linewidth=2,
                legend=:topright,
                title="                                      -
                        0 - $(crossover_type)                              ",
                xlims=x_range, ylims=plot_range,
                xlabel="X", ylabel="f(X)",
                size=(800, 600), grid=true)

    scatter!(plt, population, fitness.(population),
              color=:lightgreen, alpha=0.7, markersize=4,
              label="                    ")

    scatter!(plt, [best_solution], [best_score],
              color=:red, markersize=8, marker=:star,
              label="                           : x=$(round(best_solution, digits=3))")

    annotate!(plt, x_range[1] + 0.1*(x_range[2]-x_range[1]),
               plot_range[1] + 0.1*(plot_range[2]-plot_range[1]),
               text("f(x) = $(round(best_score, digits=3))", :left, 10))

    frame(anim)

    #
    crossover_stats = []

    for generation in 1:generations
        #
        parents = select(population, fitness, population_size    2)

        #
        children, crossover_points = crossover(parents, population_size - length(parents), crossover_type, x_range)

        #

        if crossover_type == "binary"
            push!(crossover_stats, crossover_points)
        end
```

```
        #
        children = [mutate(c, mutation_rate, x_range) for c in children]

        #
        population = vcat(parents, children)

        #
        current_scores = [fitness(x) for x in population]
        current_best_idx = argmin(current_scores)
        current_best = population[current_best_idx]
        current_score = current_scores[current_best_idx]

        #
        if generation % 10 == 0
            println("\n $generation")
            println(" : ", round(mean(population), digits=2))
            println(" : ", round(current_best, digits=3),
                    " (f = ", round(current_score, digits=3), ")")
            println(" : ", round(std(population), digits=3))

            if crossover_type == "binary"
                avg_crossover_point = round(mean(crossover_points), digits=0)
                println(" : $avg_crossover_point")
            end
        end
#
if current_score < best_score
    best_solution = current_best
    best_score = current_score
    push!(best_history, (best_solution, best_score))

    println(" $generation: = ",
           round(best_solution, digits=4),
           " ( : ", round(best_score, digits=4), ")")
end

#
plt = plot(x_vals, y_vals,
          color=:blue, linewidth=2,
          legend=:topright,
```

```
            title="                                    -
                    $generation",
            xlims=x_range, ylims=plot_range,
            xlabel="X", ylabel="f(X)",
            size=(800, 600), grid=true)

#
scatter!(plt, population, fitness.(population),
          color=:lightgreen, alpha=0.7, markersize=4,
          label="                ")


#
scatter!(plt, [best_solution], [best_score],
          color=:red, markersize=8,
          label="                        : x=$(round(best_solution,
    digits=3))")

frame(anim)
end

#
gif(anim, "genetic_algorithm_$(crossover_type)_$(name).gif", fps=5)
println("                                              :
    genetic_algorithm_$(crossover_type)_$(name).gif")

return best_solution
end

#
fit(x) = (x+1)*(x+2)*((x+3)^2)
# fit(x) = 5 - 24 * x + 17 * x^2 - 11/3 * x^3 + 1/4 * x^4

#
params = (
generations = 100,
population_size = 50,
x_range = (-4.0, 0.0),
mutation_rate = 0.2,
crossover_type = "binary"  # "average"        "binary"
)

#

@time best = genetic_algorithm(fit, params.generations, params.
    population_size,
```

```
                              params.x_range, params.mutation_rate, params.
    crossover_type,"test1")

println("\n\n                                                  :")
println("                                    : x = ", round(best, digits=4),
", f(x) = ", round(fit(best), digits=4))
```

3 Результаты

Результаты запуска представлены на рисунках 1.

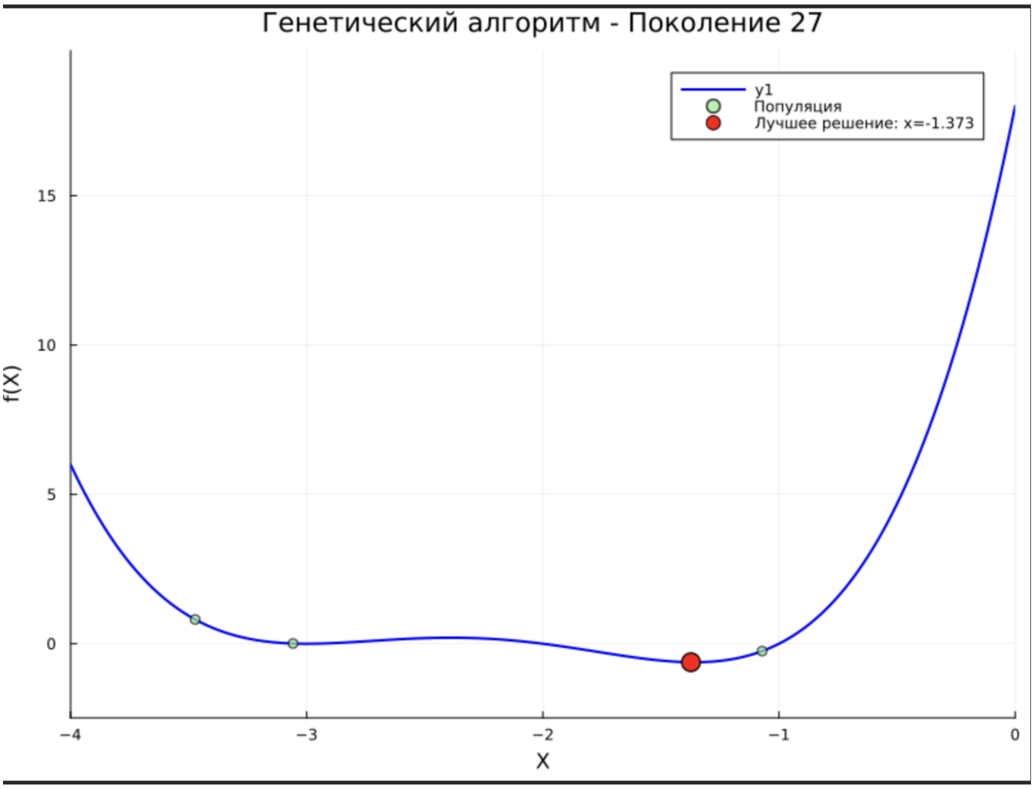

Рис. 1 — Визуализация

4 Выводы

В результате данной лабораторной работы был реализован генетический алгоритм поиска минимума, который отлично продемонстрировал возможность поиска глобального минимума на функциях с ярко выраженными локальными минимумами, которые могли бы сбить, например, градиентный спуск.