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 № 11

по курсу «Методы оптимизации»

«Реализация генетического алгоритма для функции 2 переменных»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать генетический алгоритм для поиска минимума функции 2 переменных, визуализировать процесс поиска на графиках.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```
using Plots
using Random
using Statistics

function binary_to_decimal(binary, range)
    min_val, max_val = range
    decimal = 0
    for (i, bit) in enumerate(binary)
        decimal += bit * 2^(length(binary) - i)
    end

    normalized = decimal / (2^length(binary) - 1)
    return min_val + normalized * (max_val - min_val)
end

function decimal_to_binary(decimal, range, bits_length)
    min_val, max_val = range
    normalized = (decimal - min_val) / (max_val - min_val)
    int_value = round(Int, normalized * (2^bits_length - 1))
    binary = digits(int_value, base=2, pad=bits_length)
    return reverse(binary)
end

function generate_population(size, ranges, bits_per_var)
    population = []
    for _ in 1:size
        x_binary = [rand(0:1) for _ in 1:bits_per_var]
        y_binary = [rand(0:1) for _ in 1:bits_per_var]
        push!(population, (x_binary, y_binary))
    end
    return population
end

function mutate(individual, mutation_rate)
```

```
35      x_binary, y_binary = individual
36      x_mutated = [bit      (rand() < mutation_rate) for bit in x_binary]
37      y_mutated = [bit      (rand() < mutation_rate) for bit in y_binary]
38
39      return (x_mutated, y_mutated)
40  end
41
42  function evaluate_fitness(individual, fitness, ranges, bits_per_var)
43      x_binary, y_binary = individual
44
45      x = binary_to_decimal(x_binary, ranges[1])
46      y = binary_to_decimal(y_binary, ranges[2])
47
48      return fitness(x, y), (x, y)
49  end
50
51  function select(population, fitness_values, number_of_parents)
52      sorted_idx = sortperm(fitness_values)
53      return population[sorted_idx[1:number_of_parents]]
54  end
55
56  function crossover(parents, number_of_children)
57      children = []
58      for _ in 1:number_of_children
59          parent1 = rand(parents)
60          parent2 = rand(parents)
61
62          x_crosspoint = rand(1:length(parent1[1]))
63          y_crosspoint = rand(1:length(parent1[2]))
64
65          child_x = vcat(parent1[1][1:x_crosspoint], parent2[1][
        x_crosspoint+1:end])
66          child_y = vcat(parent1[2][1:y_crosspoint], parent2[2][
        y_crosspoint+1:end])
67
68          push!(children, (child_x, child_y))
69      end
70      return children
71  end
72
73  function genetic_algorithm(fitness, generations, population_size, ranges
        , mutation_rate, bits_per_var=16)
74      Random.seed!(123)
75
76      #
```

```
77      population = generate_population(population_size, ranges,
    bits_per_var)
78
79      fitness_results = [evaluate_fitness(ind, fitness, ranges,
    bits_per_var) for ind in population]
80      fitness_values = [res[1] for res in fitness_results]
81      decoded_values = [res[2] for res in fitness_results]
82
83      best_idx = argmin(fitness_values)
84      best_solution_binary = population[best_idx]
85      best_solution = decoded_values[best_idx]
86      best_score = fitness_values[best_idx]
87
88      x_range, y_range = ranges
89      x_vals = range(x_range[1], x_range[2], length=50)
90      y_vals = range(y_range[1], y_range[2], length=50)
91
92      anim = Animation()
93      anim_contour = Animation()
94
95      z_func = [fitness(x, y) for y in y_vals, x in x_vals]
96
97      z_min = minimum(z_func) - 5
98      z_max = maximum(z_func) + 5
99
100     plt = surface(x_vals, y_vals, z_func,
101                   alpha=0.7,
102                   color=:viridis,
103                   title="Generation 0",
104                   xlabel="X", ylabel="Y", zlabel="f(X,Y)",
105                   zlims=(z_min, z_max))
106
107     scatter!(plt, [p[1] for p in decoded_values], [p[2] for p in
    decoded_values],
108              fitness_values,
109              color=:green, markersize=2, label="                    ")
110
111     scatter!(plt, [best_solution[1]], [best_solution[2]], [best_score],
112              color=:red, markersize=4, marker=:star,
113              label="                          ($(round(best_score, digits
    =4)))")
114
115     frame(anim)
116
117     plt_contour = contour(x_vals, y_vals, z_func,
118                           fill=true,
```

```
                        levels=20,
                        color=:viridis,
                        title="Generation 0 (Contour)",
                        xlabel="X", ylabel="Y")

    #

    scatter!(plt_contour, [p[1] for p in population], [p[2] for p in
    population],
            color=:green, markersize=3, label="                    ")

    #

    scatter!(plt_contour, [best_solution[1]], [best_solution[2]],
            color=:red, markersize=5, marker=:star,
            label="                          ($(round(best_score, digits
    =4)))")

    frame(anim_contour)



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

        children = crossover(parents, population_size - length(parents))

        children = [mutate(child, mutation_rate) for child in children]

        population = vcat(parents, children)

        fitness_results = [evaluate_fitness(ind, fitness, ranges,
    bits_per_var) for ind in population]
        fitness_values = [res[1] for res in fitness_results]
        decoded_values = [res[2] for res in fitness_results]

        current_best_idx = argmin(fitness_values)
        current_best_binary = population[current_best_idx]
        current_best = decoded_values[current_best_idx]
        current_score = fitness_values[current_best_idx]

        if generation % 10 == 0
            println("\nGeneration $generation")
            println("Population stats:")
            println("  Size: ", length(population))
```

```
159             println("  Mean X: ", round(mean([p[1] for p in
    decoded_values]), digits=2))
160             println("  Mean Y: ", round(mean([p[2] for p in
    decoded_values]), digits=2))
161             println("  Best: (", round(current_best[1], digits=2), ", ",
162                     round(current_best[2], digits=2), ")",
163                     " (f = ", round(current_score, digits=2), ")")
164             println("  Diversity X: ", round(std([p[1] for p in
    decoded_values]), digits=3))
165             println("  Diversity Y: ", round(std([p[2] for p in
    decoded_values]), digits=3))
166         end
167
168         if current_score < best_score
169             best_solution_binary = current_best_binary
170             best_solution = current_best
171             best_score = current_score
172         end
173
174         plt = surface(x_vals, y_vals, z_func,
175         alpha=0.7,
176         color=:viridis,
177         title="Generation $generation",
178         xlabel="X", ylabel="Y", zlabel="f(X,Y)",
179         zlims=(z_min, z_max))
180
181         scatter!(plt, [p[1] for p in decoded_values], [p[2] for p in
    decoded_values],
182         fitness_values,
183         color=:green, markersize=2, label="                    ")
184
185         scatter!(plt, [best_solution[1]], [best_solution[2]], [
    best_score],
186         color=:red, markersize=4, marker=:star,
187         label="                          ($(round(best_score, digits=4)))
    ")
188
189         frame(anim)
190
191         #

192         plt_contour = contour(x_vals, y_vals, z_func,
193         fill=true,
194         levels=20,
195         color=:viridis,
196         title="Generation $generation (Contour)",
```

```
197         xlabel="X", ylabel="Y")
198 
199         #
200         scatter!(plt_contour, [p[1] for p in population], [p[2] for p in
        population],
201         color=:green, markersize=3, label="                    ")
202 
203         #
204         scatter!(plt_contour, [best_solution[1]], [best_solution[2]],
205         color=:red, markersize=5, marker=:star,
206         label="                          ($(round(best_score, digits=4)))
        ")
207 
208         frame(anim_contour)
209 
210         println("Generation $generation: New best = (",
211         round(best_solution[1], digits=4), ", ",
212         round(best_solution[2], digits=4), ")",
213         " (Score: ", round(best_score, digits=4), ")")
214             end
215 
216     #
217     gif(anim, "genetic_algorithm_binary_3d.gif", fps=3)
218     # gif(anim_contour, "genetic_algorithm_contour.gif", fps=5)
219     println("                                                :
        genetic_algorithm_binary_3d.gif")
220 
221     final_plt = surface(x_vals, y_vals, z_func,
222                         alpha=0.7,
223                         color=:viridis,
224                         title="Final Result (Binary Encoding)",
225                         xlabel="X", ylabel="Y", zlabel="f(X,Y)")
226 
227     scatter!(final_plt, [best_solution[1]], [best_solution[2]], [
        best_score],
228             color=:red, markersize=6, marker=:star,
229             label="                                    ")
230 
231     contour_plt = contour(x_vals, y_vals, z_func,
232                           title="Contour Plot with Best Solution (Binary
        Encoding)",
233                           xlabel="X", ylabel="Y")
234 
235     scatter!(contour_plt, [best_solution[1]], [best_solution[2]],
236             color=:red, markersize=6, marker=:star,
237             label="                                    ")
```

```
238
239     savefig(final_plt, "final_surface_plot_binary.png")
240     savefig(contour_plt, "final_contour_plot_binary.png")
241
242     println("\nBinary representation of best solution:")
243     println("X: ", join(best_solution_binary[1]), " (", best_solution
        [1], ")")
244     println("Y: ", join(best_solution_binary[2]), " (", best_solution
        [2], ")")
245
246     return best_solution, best_score, best_solution_binary
247 end
248
249
250 function rosenbrock(x, y)
251     return (1 - x)^2 + 100 * (y - x^2)^2
252 end
253
254 #
255 function rastrigin(x, y)
256     return 20 + x^2 + y^2 - 10 * (cos(2   * x) + cos(2   * y))
257 end
258
259 function schwefel(x,y)
260     return 418.9829*2 - (x*sin(sqrt(abs(x))) + y*sin(sqrt(abs(y))))
261 end
262
263
264 fit_function = schwefel
265
266 params = (
267     generations = 100,
268     population_size = 50,
269     ranges = ((-5.0, 5.0), (-5.0, 5.0)),
270     mutation_rate = 0.2,
271     bits_per_var = 16
272 )
273
274 #
275 @time best_solution, best_score, best_binary = genetic_algorithm(
276     fit_function,
277     params.generations,
278     params.population_size,
279     params.ranges,
280     params.mutation_rate,
281     params.bits_per_var
```

```
)

println("\n\nFinal result:")
println("Best solution: (x, y) = (", round(best_solution[1], digits=4),
    ", ",
        round(best_solution[2], digits=4), ")")
println("Function value: f(x, y) = ", round(best_score, digits=4))
println("Binary representation:")
println("X: ", join(best_binary[1]))
println("Y: ", join(best_binary[2]))
```

3 Результаты

Результаты запуска представлены на рисунках 1.

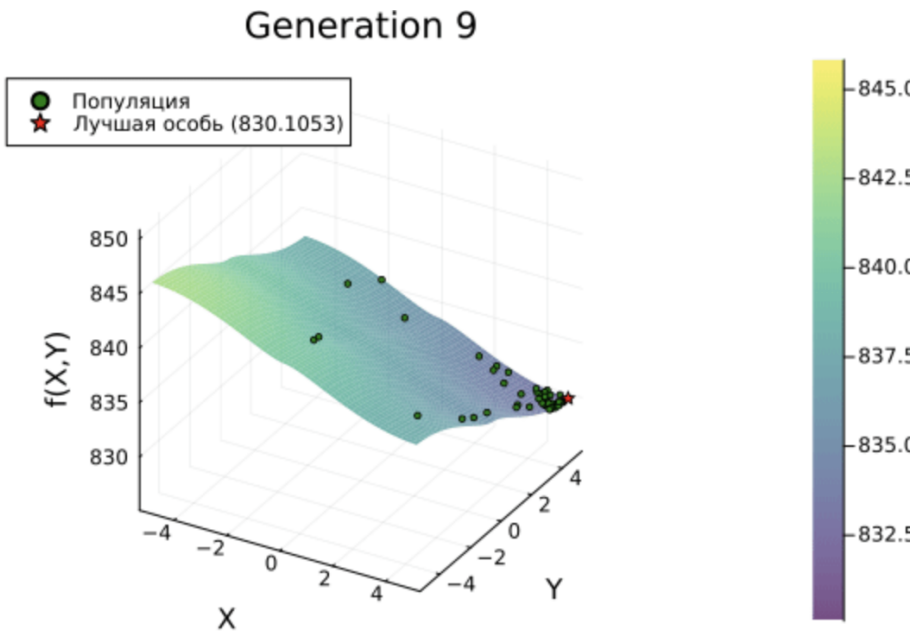

Рис. 1 — Визуализация

4 Выводы

В результате данной лабораторной работы был реализован генетический алгоритм поиска минимума функции нескольких переменных, который отлично продемонстрировал возможность поиска глобального минимума на овражных функциях.