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 № 15

по курсу «Методы оптимизации»

«Генетический алгоритм с нефиксированным размером популяции для функций нескольких переменных»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать на основе летучки номер 6 генетический алгоритм с нефиксированным размером популяции, используя линейное скрещивание для функций нескольких перменных.

Необходимо сделать качественную визуализацию - цвет точки меняется от времени ее жизни.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```
function rosenbrock(x, y)
    return (1.0 - x)^2 + 100.0 * (y - x^2)^2
end

function rastrigin(x, y)
    return 20 + x^2 + y^2 - 10 * (cos(2    * x) + cos(2    * y))
end

function shvefel(x, y)
    return 418.9829*2 - (x*sin(sqrt(abs(x))) + y*sin(sqrt(abs(y))))
end

using Plots
using Random

#
f(x, y) = rosenbrock(x, y)

#
initial_pop_size = 50
n_generations = 100

x_range = (0.0, 3.0)
y_range = (0.0, 3.0)

tournament_size = 3
max_age = 10

#
mutable struct Indiv
```

```
31      x::Float64
32      y::Float64
33      age::Int
34      new_this_generation::Bool
35  end
36
37  #
38  function tournament_selection(population, fitness_values,
        tournament_size)
39      indices = rand(1:length(population), tournament_size)
40      winner_idx = indices[argmin([fitness_values[i] for i in indices])]
41      return population[winner_idx]
42  end
43
44  #
45  function linear_crossover(parent1, parent2, x_range, y_range)
46      alpha = rand()
47      beta = 1 - alpha
48
49      child_x = alpha * parent1.x + beta * parent2.x
50      child_y = alpha * parent1.y + beta * parent2.y
51
52      child_x = clamp(child_x, x_range[1], x_range[2])
53      child_y = clamp(child_y, y_range[1], y_range[2])
54
55      return (child_x, child_y)
56  end
57
58  population = [Indiv(
59      rand() * (x_range[2] - x_range[1]) + x_range[1],
60      rand() * (y_range[2] - y_range[1]) + y_range[1],
61      0,
62      false
63  ) for _ in 1:initial_pop_size]
64
65  best_xy_per_gen = []
66  best_f_per_gen = []
67  pop_size_per_gen = Int[]
68  all_populations = []
69
70  push!(all_populations, deepcopy(population))
71
72  #
73  for generation in 1:n_generations
74      for ind in population
75          ind.age += 1
```

```
76          ind.new_this_generation = false
77      end
78
79      fitness = [f(ind.x, ind.y) for ind in population]
80      best_idx = argmin(fitness)
81      best_ind = population[best_idx]
82
83      push!(best_xy_per_gen, (best_ind.x, best_ind.y))
84      push!(best_f_per_gen, fitness[best_idx])
85      push!(pop_size_per_gen, length(population))
86
87      #
88      filter!(ind -> ind.age < max_age, population)
89
90      new_Indivs = []
91      n_offspring = 5
92
93      for _ in 1:n_offspring
94          fitness = [f(ind.x, ind.y) for ind in population]
95          parent1 = tournament_selection(population, fitness,
    tournament_size)
96          parent2 = tournament_selection(population, fitness,
    tournament_size)
97
98          if rand() < 0.7
99              child_x, child_y = linear_crossover(parent1, parent2,
    x_range, y_range)
100         else
101             if rand(Bool)
102                 child_x, child_y = parent1.x, parent1.y
103             else
104                 child_x, child_y = parent2.x, parent2.y
105             end
106         end
107
108         #
109         if rand() < 0.1
110             child_x += randn() * 0.5
111             child_y += randn() * 0.5
112             child_x = clamp(child_x, x_range[1], x_range[2])
113             child_y = clamp(child_y, y_range[1], y_range[2])
114         end
115
116         push!(new_Indivs, Indiv(child_x, child_y, 0, true))
117     end
118
```

```
119     append!(population, new_Indivs)
120     push!(all_populations, deepcopy(population))
121 end
122
123 #
124 xs = range(x_range[1], x_range[2], length=100)
125 ys = range(y_range[1], y_range[2], length=100)
126 zs = [f(x, y) for x in xs, y in ys]
127
128 anim = @animate for i in 1:n_generations
129     current_pop = all_populations[i]
130
131     #
132     p1 = contour(xs, ys, zs',
133         title = "                    $(i) -
                ",
134         xlabel = "x",
135         ylabel = "y",
136         fill = true,
137         levels = 30,
138         color = :viridis,
139         legend = :bottomright)
140
141     # 3D
142     p2 = surface(xs, ys, zs',
143         title = "                    $(i) - 3D       ",
144         xlabel = "x",
145         ylabel = "y",
146         zlabel = "f(x,y)",
147         color = :viridis,
148         alpha = 0.7,
149         camera = (30, 30))
150
151     #

152     existing = filter(ind -> !ind.new_this_generation, current_pop)
153     new_inds = filter(ind -> ind.new_this_generation, current_pop)
154
155     scatter!(p1, [ind.x for ind in existing], [ind.y for ind in existing
    ],
156             markersize = 4, alpha = 0.4, color = :grey, label = "
                ")
157
158     scatter!(p1, [ind.x for ind in new_inds], [ind.y for ind in new_inds
    ],
159             markersize = 5, color = :red, label = "          ")
```

```
160
161     best_xy = best_xy_per_gen[i]
162     scatter!(p1, [best_xy[1]], [best_xy[2]],
163              color = :green, markershape = :star, markersize = 7, label
    = "            ")
164
165     #                                                    3D
166     for ind in existing
167         z_val = f(ind.x, ind.y)
168         scatter!(p2, [ind.x], [ind.y], [z_val],
169                  markersize = 3, alpha = 0.4, color = :grey, label = "")
170     end
171
172     for ind in new_inds
173         z_val = f(ind.x, ind.y)
174         scatter!(p2, [ind.x], [ind.y], [z_val],
175                  markersize = 4, color = :red, label = "")
176     end
177
178     best_z = f(best_xy[1], best_xy[2])
179     scatter!(p2, [best_xy[1]], [best_xy[2]], [best_z],
180              color = :green, markershape = :star, markersize = 5, label
    = "")
181
182     #
183     plot(p1, p2, layout = (1, 2), size = (1200, 600))
184 end
185
186 gif(anim, "ga_2d_rosen.gif", fps=10)
187
188 println("                                                    : $(length(
    population))")
```

3 Результаты

Результаты запуска представлены на рисунках 1.

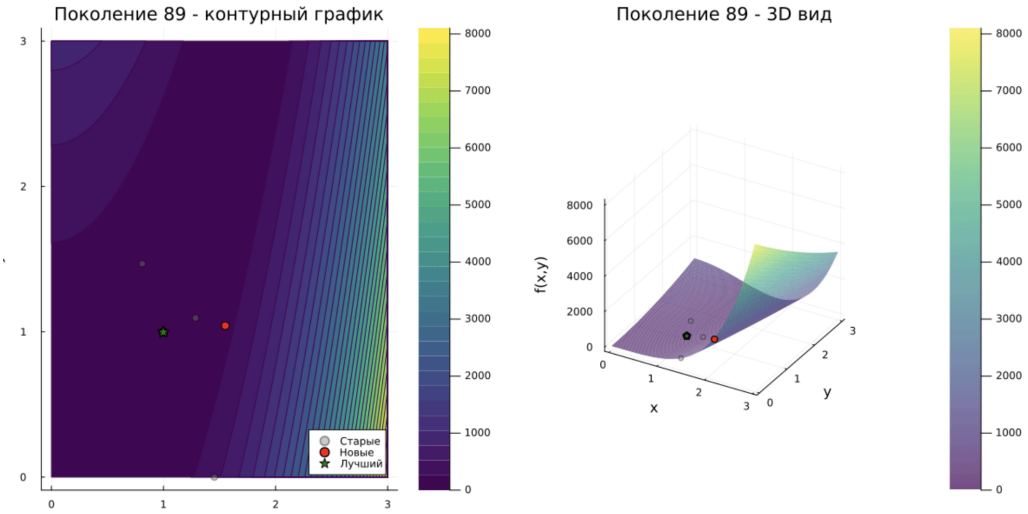

Рис. 1 — Визуализация

4 Выводы

В результате выполнения данного задания был успешно реализован генетический алгоритм с нефиксированным размером популяции и проверена его сходимость на функциях нескольких перменных.