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 № 14

по курсу «Методы оптимизации»

«Оптимизация функции нескольких переменных муравьиным алгоритмом»

Студент группы ИУ9-81Б Окутин Д.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать муравьиный алгоритм для поиска минимума овражной функции.

2 Реализация

Исходный код программмы представлен в листинге 1.

Листинг 1: code

```
using Plots, Random

function rosenbrock(x::Vector{Float64})
    return (1 - x[1])^2 + 100 * (x[2] - x[1]^2)^2
end

n = 2                       #
domain_min = -5.12          #
domain_max = 5.12           #
B = 11                      #
                (                       : B-1)
N = 30                      #
max_iter = 100              #

alpha = 1.0                #
Q = 20.0                    #

rho = 0.9                  #
tau0 = 1.0                 #


#

boundaries = Vector{Vector{Float64}}(undef, n)
for i in 1:n
    boundaries[i] = collect(range(domain_min, domain_max, length=B))
end
pheromones = [fill(tau0, B - 1) for i in 1:n]

#

function generate_solution(pheromones, boundaries, alpha)
    sol = zeros(n)
    for i in 1:n
```

```
30          tau_vec = pheromones[i]
31          prob = [tau^alpha for tau in tau_vec]
32          prob ./= sum(prob)
33          r = rand(); cumulative = 0.0; chosen_interval = 0
34          for j in 1:length(prob)
35              cumulative += prob[j]
36              if r <= cumulative
37                  chosen_interval = j
38                  break
39              end
40          end
41          chosen_interval = (chosen_interval == 0 ? length(prob) :
    chosen_interval)
42          b_low = boundaries[i][chosen_interval]
43          b_high = boundaries[i][chosen_interval+1]
44          sol[i] = rand() * (b_high - b_low) + b_low
45      end
46      return sol
47  end
48
49  #
50  solutions_history = Vector{Matrix{Float64}}()   # 2D

51  best_history = Vector{Vector{Float64}}()          #

52  best_value_history = Float64[]
53  pheromone_history = Vector{Any}()                 #


54
55  global_best_solution = zeros(n)
56  global_best_value = Inf
57
58  #
59  for iter in 1:max_iter
60      solutions = [generate_solution(pheromones, boundaries, alpha) for k
    in 1:N]
61      costs = [rastrigin(sol) for sol in solutions]
62
63      #              :
                                                  ,

64      if global_best_value < Inf
65          solutions[1] = copy(global_best_solution)
66          costs[1] = global_best_value
67      end
68
```

```
69      #
70      for k in 1:N
71          if costs[k] < global_best_value
72              global_best_value = costs[k]
73              global_best_solution = copy(solutions[k])
74          end
75      end
76
77      #
78      sol_matrix = zeros(N, 2)
79      for k in 1:N
80          sol_matrix[k, :] = solutions[k]
81      end
82      push!(solutions_history, sol_matrix)
83      push!(best_history, copy(global_best_solution))
84      push!(best_value_history, global_best_value)
85      push!(pheromone_history, deepcopy(pheromones))
86
87      #                                          (
                          0)
88      eps = 1e-10
89      for i in 1:n
90          delta_tau = zeros(B - 1)
91          for k in 1:N
92              sol_val = solutions[k][i]
93              for j in 1:(B - 1)
94                  if sol_val >= boundaries[i][j] && sol_val < boundaries[i
        ][j+1]
95                      delta_tau[j] += Q / (costs[k] + eps)
96                      break
97                  elseif j == B - 1 && sol_val == boundaries[i][end]
98                      delta_tau[j] += Q / (costs[k] + eps)
99                      break
100                 end
101             end
102         end
103         for j in 1:(B - 1)
104             pheromones[i][j] = (1 - rho)*pheromones[i][j] + delta_tau[j]
105         end
106     end
107     # println("                    $iter:              L = $(
        global_best_value)")
108 end
109
```

```
#

xgrid = collect(range(domain_min, domain_max, length=100))
ygrid = collect(range(domain_min, domain_max, length=100))
zgrid = [rosenbrock([x, y]) for x in xgrid, y in ygrid]

#

x_centers = [(boundaries[1][i] + boundaries[1][i+1]) / 2 for i in 1:(B
    -1)]
y_centers = [(boundaries[2][i] + boundaries[2][i+1]) / 2 for i in 1:(B
    -1)]
#                    :                                                (
                                                                      )
anim = @animate for i in 1:length(solutions_history)
    sol_mat = solutions_history[i]
    best_sol = best_history[i]

    #                  p1: 2D-
    p1 = contour(xgrid, ygrid, zgrid', fill=false, levels=20,
                 color=:viridis,
                  xlims=(domain_min, domain_max), ylims=(domain_min,
    domain_max),
                  xlabel="x", ylabel="y",
                  title="                    : $i |              : $(round(
    best_value_history[i], digits=2))",
                  aspect_ratio=1)
    scatter!(p1, sol_mat[:,1], sol_mat[:,2], label="                ")
    scatter!(p1, [best_sol[1]], [best_sol[2]], markershape=:circle,
    markersize=10,
             color=:red, label="             ")

    #                  p2: 3D-
    p2 = surface(xgrid, ygrid, zgrid', alpha=0.6,
                  xlims=(domain_min, domain_max), ylims=(domain_min,
    domain_max),                          color=:viridis,
                  zlims=(minimum(zgrid), maximum(zgrid)),
                  xlabel="x", ylabel="y", zlabel="f(x,y)",
                  title="3D                      : $i",
                  aspect_ratio=1)
    z_sol = [rosenbrock([sol_mat[k,1], sol_mat[k,2]]) for k in 1:N]
    scatter3d!(p2, sol_mat[:,1], sol_mat[:,2], z_sol, markerstrokewidth
    =0,
               label="               ")
    scatter3d!(p2, [best_sol[1]], [best_sol[2]], [rosenbrock(best_sol)],
```

```
                    markershape=:circle , markersize=10, color=:red , label="
                    ")


    #                                              : 1               , 3
                        ,                              1200x400 (
                                               )
    plot(p1, p2, layout = (1, 2), legend = :bottomright, size=(1200,400)
    )
end

gif(anim, "ant_colony_combined_3panels_square.gif", fps=10)
println("                                                    3

                   ant_colony_combined_3panels_square.gif")
```

3 Результаты

Результаты запуска представлены на рисунках 1.

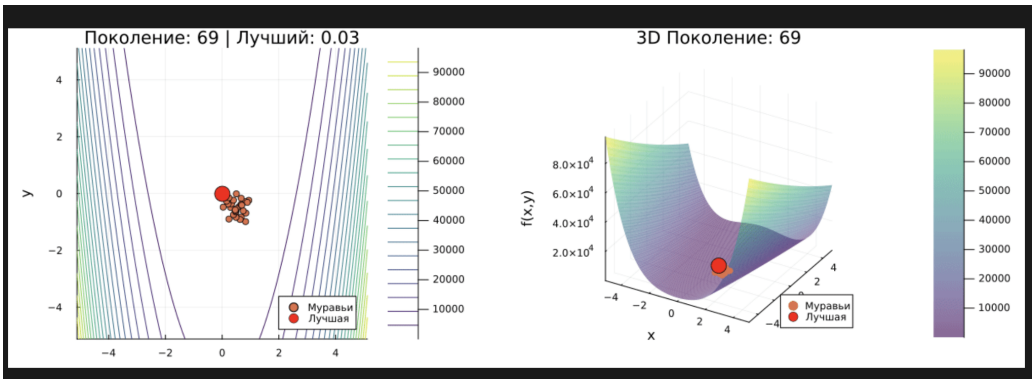

Рис. 1 — Визуализация

4 Выводы

В результате данной лабораторной работы был реализован муравьиный алгоритм. С помощью которого были успешно найдены глобальные минимумы овражных функций.