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 № 6

по курсу «Методы оптимизации»

«Генетический алгоритм с нефиксированным размером популяции»

Студент группы ИУ9-81Б Окутин Д. А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Реализовать на основе летучки номер 5 генетический алгоритм с нефиксированным размером популяции, используя линейное скрещивание.

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

2 Реализация

Параметры алгоритма

- **Целевая функция**:

$$f(x) = (x+1)(x+2)(x+3)^2$$

  Поиск минимума в диапазоне $x \in [-5.0, 1.0]$.

- **Основные параметры**:
  - Начальный размер популяции: 30
  - Число поколений: 50
  - Диапазон x: $[-5.0, 1.0]$
  - Размер турнира: 3
  - Максимальный возраст особи: 20 поколений

- **Операторы**:
  - Вероятность кроссовера: 70%
  - Вероятность мутации: 10% (сила: $\mathcal{N}(0, 0.5)$)
  - Число потомков за поколение: 5

Теоретическая основа

Структура особи

Каждая особь представлена объектом:

- value – значение x
- age – возраст (в поколениях)
- new_this_generation – флаг новизны

Этапы алгоритма

1. **Инициализация**: Случайная генерация начальной популяции в x-диапазоне.
2. **Селекция**: Турнирный отбор:

$$\text{Родитель} = \underset{\text{турнир}}{\arg\min}\, f(x)$$

3. **Кроссовер**: Линейная комбинация родителей:

$$x_{\text{child}} = \alpha x_{\text{parent1}} + (1-\alpha)x_{\text{parent2}}, \quad \alpha \sim U[0,1]$$

4. **Мутация**: Гауссово возмущение:

$$x_{\text{mut}} = x + \mathcal{N}(0, 0.5), \quad \text{с последующей проекцией на } [-5, 1]$$

5. **Age-based отбор**: Удаление особей с возрастом ≥ 20.

Динамика популяции

- Ежегенерационное старение: age$+ = 1$
- Баланс между новыми и старыми особями
- Автоматический контроль размера популяции

Визуализация

- **Цветовая схема**:
  - Серый: Существующие особи (прозрачность $\propto$ оставшемуся времени жизни)
  - Синий: Новые особи
  - Красная звезда: Лучшая особь поколения
- **Анимация**:
  - Сохранение в формате GIF
  - Частота кадров: 10 FPS
  - Диапазон y: $[-5, 20]$

Исходный код

Исходный код программмы представлен в листинге 1.

Листинг 1: методы

```
using Plots
using Random

f(x) = (x+1)*(x+2)*((x+3)^2)

initial_pop_size = 20
n_generations = 150
x_range = (-5.0, 1.0)
tournament_size = 3
max_age = 20

mutable struct Indiv
    value::Float64
    age::Int
    new_this_generation::Bool
end

function linear_crossover(parent1::Float64, parent2::Float64)
    alpha = rand()
    return alpha * parent1 + (1 - alpha) * parent2
```

```
21 end
22
23 function tournament_selection(population, fitness_values,
       tournament_size)
24     indices = rand(1:length(population), tournament_size)
25     winner_idx = indices[argmin([fitness_values[i] for i in indices])]
26     return population[winner_idx]
27 end
28
29 population = [Indiv(rand() * (x_range[2] - x_range[1]) + x_range[1], 0,
       false) for _ in 1:initial_pop_size]
30
31 best_x_per_gen = Float64[]
32 best_y_per_gen = Float64[]
33 pop_size_per_gen = Int[]
34 all_populations = []
35 push!(all_populations, deepcopy(population))
36
37 for generation in 1:n_generations
38     for ind in population
39         ind.age += 1
40         ind.new_this_generation = false
41     end
42
43     values = [f(ind.value) for ind in population]
44     best_index = argmin(values)
45     push!(best_x_per_gen, population[best_index].value)
46     push!(best_y_per_gen, values[best_index])
47     push!(pop_size_per_gen, length(population))
48
49     filter!(ind -> ind.age < max_age, population)
50
51     new_Indivs = []
52     for _ in 1:5
53         parent1 = tournament_selection(population, values,
       tournament_size)
54         parent2 = tournament_selection(population, values,
       tournament_size)
55
56         if rand() < 0.7
57             child_value = linear_crossover(parent1.value, parent2.value)
58         else
59             child_value = rand(Bool) ? parent1.value : parent2.value
60         end
61
62         if rand() < 0.1
```

```
63              child_value += randn() * 0.5
64              child_value = clamp(child_value, x_range[1], x_range[2])
65          end
66
67          push!(new_Indivs, Indiv(child_value, 0, true))
68      end
69      append!(population, new_Indivs)
70      push!(all_populations, deepcopy(population))
71  end
72
73  xs = range(-5, 1, length=400)
74  ys = [f(x) for x in xs]
75  anim = @animate for i in 1:n_generations
76      current_pop = all_populations[i]
77      p = plot(xs, ys, label="f(x)", title="Generation $(i)", xlabel="x",
        ylabel="f(x)", legend=:topright, lw=2, ylims=(-10, 50))
78
79      #
80      for ind in current_pop
81          color = ind.new_this_generation ? :red : :grey
82          alpha = ind.new_this_generation ? 1.0 : (max_age - ind.age)/
        max_age
83          scatter!([ind.value], [f(ind.value)], color=color, alpha=alpha,
        markersize=6, label="")
84      end
85
86      #
87      scatter!([best_x_per_gen[i]], [best_y_per_gen[i]], color=:blue,
        markersize=8, markershape=:star5, label="Best")
88
89      sleep(0.01)
90  end
91
92  gif(anim, "ga_linear_crossover.gif", fps=10)
93  println("Minimum: x = $(best_x_per_gen[end]), f(x) = $(best_y_per_gen[
        end])")
94  println("Final population size: $(length(population))")
```

3 Результаты

Результат запуска представлен на рисунке 1.

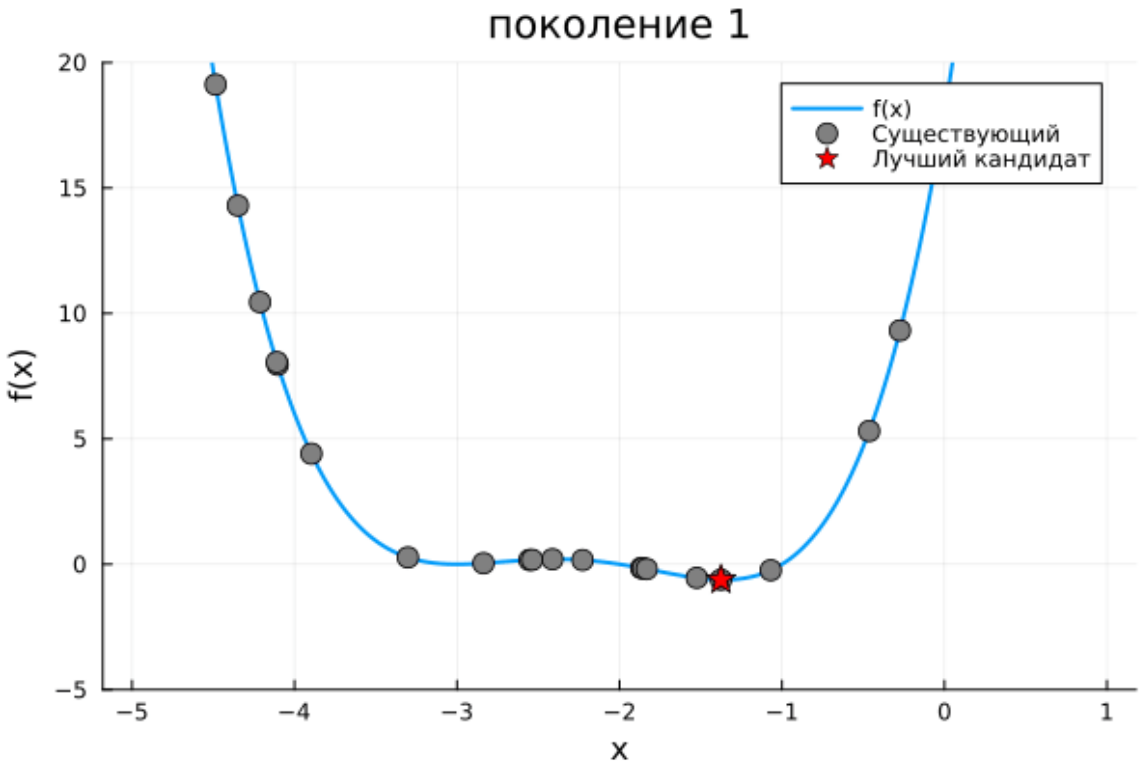

Рис. 1 — Поколение 1

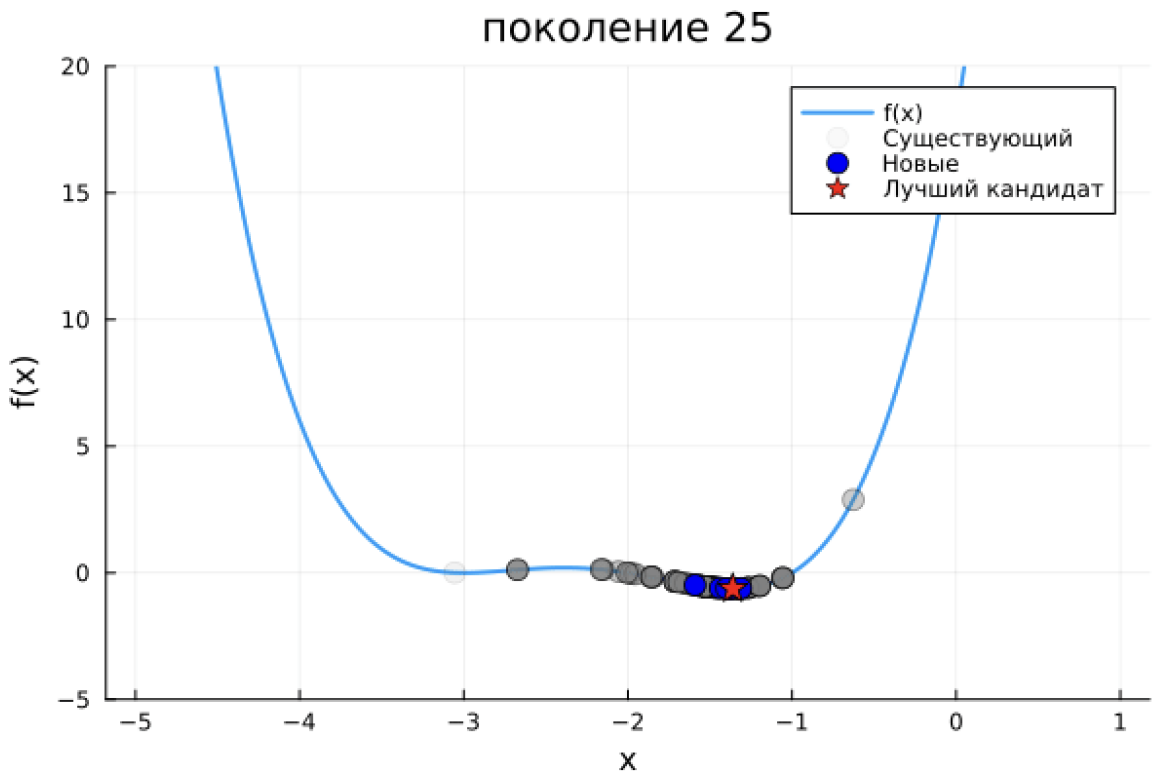

Рис. 2 — Поколение 25

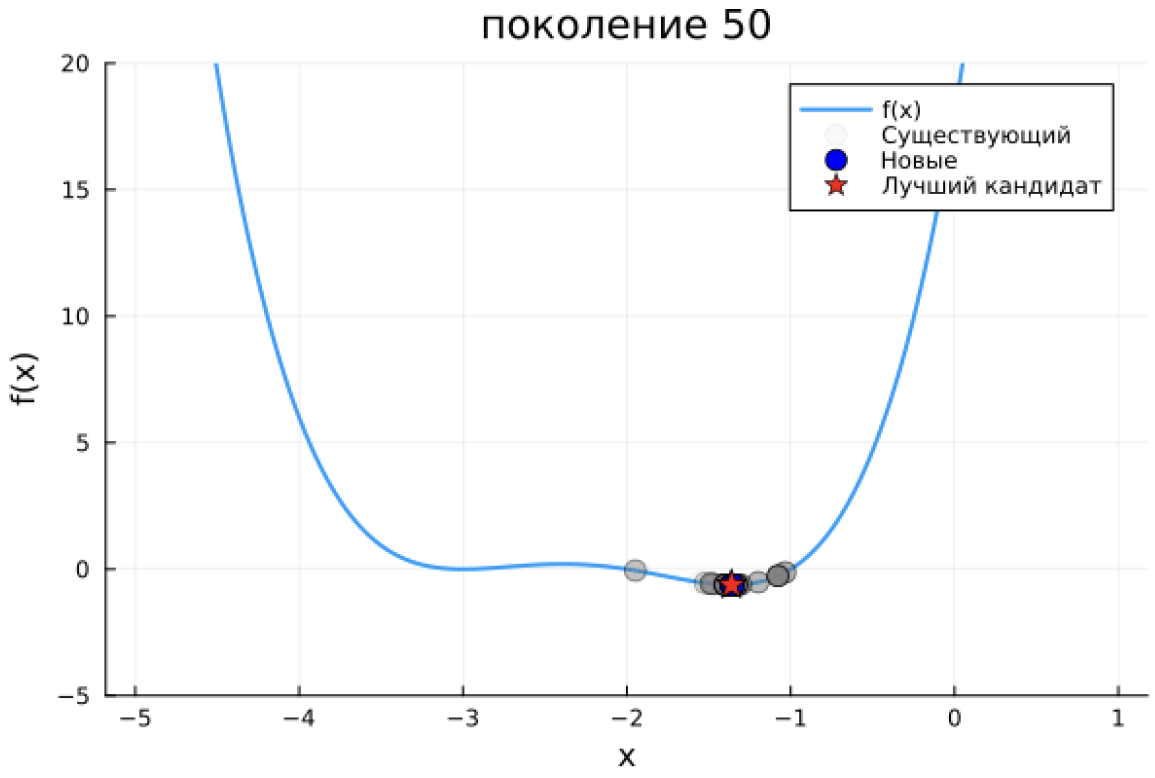

Рис. 3 — Поколение 50

4 Выводы

В результате выполнения данного задания был успешно реализован генетический алгоритм с нефиксированным размером популяции и проверена его сходимость на функции из предыдущих летучек.