МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Лабораторная работа №1
по дисциплине
«Генетические алгоритмы»
Вариант 18

Студент,
группы 5130201/20101 _____ Тищенко А. А.

Преподаватель _____ Большаков А. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

Содержание

1 Постановка задачи

В данной работе были поставлены следующие задачи:

- Изучить теоретический материал;
- Ознакомиться с вариантами кодирования хромосомы;
- Рассмотреть способы выполнения операторов репродукции, кроссинговера и мутации;
- Выполнить индивидуальное задание на любом языке высокого уровня с необходимыми комментариями и выводами

Индивидуальное задание вариант 18:

Дано: Функция одной переменной $f(x) = \frac{\sin(x)}{x^2}$; промежуток нахождения решения $x \in [3.1, 20.0]$.

Требуется:

1. Разработать простой генетический алгоритм для нахождения минимума данной функции в заданном промежутке;
2. Исследовать зависимость времени поиска, числа поколений (генераций) и точности нахождения решения от числа особей в популяции и вероятности кроссинговера и мутации;
3. Вывести на экран график функции с указанием найденного экстремума для каждого поколения;
4. Сравнить найденное решение с действительным.

Ограничения:

1. $\min f(x) = \frac{\sin(x)}{x^2} \approx -0.04957$
2. Точность решения составляет 3 знака после запятой.

2 Теоретические сведения

Генетиче ские алгоритмы (ГА) используют принципы и терминологию, заимствованные у биологической науки – генетики. В ГА каждая особь представляет потенциальное решение некоторой проблемы. В классическом ГА особь кодируется строкой двоичных символов – хромосомой, каждый бит которой называется геном. Множество особей – потенциальных решений составляет популяцию. Поиск (суб)оптимального решения проблемы выполняется в процессе эволюции популяции - последовательного преобразования одного конечного множества решений в другое с помощью генетических операторов репродукции, кроссинговера и мутации.

Предварительно простой ГА случайным образом генерирует начальную популяцию стрингов (хромосом). Затем алгоритм генерирует следующее поколение (популяцию), с помощью трех основных генетических операторов:

1. Оператор репродукции (ОР);
2. Оператор скрещивания (кроссинговера, ОК);
3. Оператор мутации (ОМ).

ГА работает до тех пор, пока не будет выполнено заданное количество поколений (итераций) процесса эволюции или на некоторой генерации будет получено заданное качество или вследствие преждевременной сходимости при попадании в некоторый локальный оптимум. На Рис. 1 представлен простой генетический алгоритм.

Рис. 1. Простой генетический алгоритм

2.1 Основная терминология в генетических алгоритмах

Ген – элементарный код в хромосоме s_i, называемый также знаком или детектором (в классическом ГА $s_i = 0, 1$).

Хромосома – упорядоченная последовательность генов в виде закодированной структуры данных $S = (s_1, s_2, \ldots, s_n)$, определяющая решение (в простейшем случае двоичная последовательность – стринг, где $s_i = 0, 1$).

Локус – местоположение (позиция, номер бита) данного гена в хромосоме.

Аллель – значение, которое принимает данный ген (например, 0 или 1).

Особь – одно потенциальное решение задачи (представляемое хромосомой).

Популяция – множество особей (хромосом), представляющих потенциальные решения.

Поколение – текущая популяция ГА на данной итерации алгоритма.

Генотип – набор хромосом данной особи. В популяции могут использоваться как отдельные хромосомы, так и целые генотипы.

Генофонд – множество всех возможных генотипов.

Фенотип – набор значений, соответствующий данному генотипу. Это декодированное множество параметров задачи (например, десятичное значение x, соответствующее двоичному коду).

Размер популяции N – число особей в популяции.

Число поколений – количество итераций, в течение которых производится поиск.

Селекция – совокупность правил, определяющих выживание особей на основе значений целевой функции.

Эволюция популяции – чередование поколений, в которых хромосомы изменяют свои признаки, чтобы каждая новая популяция лучше приспосабливалась к среде.

Фитнесс-функция – функция полезности, определяющая меру приспособленности особи. В задачах оптимизации она совпадает с целевой функцией или описывает близость к оптимальному решению.

2.2 Генетические операторы

2.2.1 Оператор репродукции

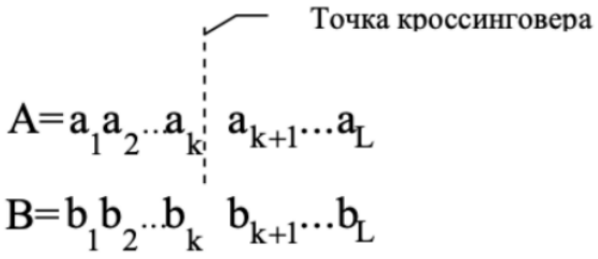
Репродукция – процесс копирования хромосом в промежуточную популяцию для дальнейшего "размножения" в соответствии со значениями фитнесс-функции. В данной работе рассматривается метод колеса рулетки. Каждой хромосоме соответствует сектор, пропорциональный значению фитнесс-функции. Хромосомы с большим значением имеют больше шансов попасть в следующее поколение.

2.2.2 Оператор скрещивания (кроссинговера)

Одноточечный кроссинговер выполняется следующим образом:

1. Из промежуточной популяции выбираются две хромосомы (родители).

2. Определяется случайная точка скрещивания $k \in [1, n-1]$, где n – длина хромосомы.



3. Две новые хромосомы (потомки) формируются путём обмена подстрок после точки k.

$$A' = a_1 a_2 \ldots a_k \; b_{k+1} \ldots b_L$$

$$B' = b_1 b_2 \ldots b_k \; a_{k+1} \ldots a_L$$

2.2.3 Оператор мутации

Мутация применяется с малой вероятностью $P_M \approx 0.001$:

1. В хромосоме $A = a_1 a_2 \ldots a_n$ выбирается случайная позиция k.
2. Ген a_k инвертируется: $a'_k = \neg a_k$.

3 Особенности реализации

В рамках работы создана мини-библиотека `gen.py` для экспериментов с простым генетическим алгоритмом (ГА) на одной переменной. Второй модуль `expirements.py` организует серийные эксперименты (перебор параметров, форматирование и сохранение результатов).

- **Кодирование особей**: каждая хромосома — список битов фиксированной длины L. Длина L подбирается автоматически так, чтобы шаг дискретизации по x был не хуже заданной точности (например, 10^{-3}). Декодирование — линейное в $[x_{\min}, x_{\max}]$. За это отвечают:
  - `bits_for_precision(x_min: float, x_max: float, digits_after_decimal: int) -> int`
  - `decode_bits_to_x(bits: list[int], x_min: float, x_max: float) -> float`
  - `random_bits(L: int) -> list[int]`
- **Фитнесс и минимум/максимум**: целевая функция передаётся параметром. Для режима *минимизации* используется внутреннее преобразование при селекции (инвертирование знака/сдвиг), что позволяет единообразно применять рулетку при отрицательных значениях. За это отвечают:
  - `eval_population(population: list[list[int]], x_min: float, x_max: float, fitness_func(x: float) -> float) -> (list[float], list[float])`
  - Логика режима минимизации в `genetic_algorithm(config: GARunConfig) -> GARunResult`
- **Селекция (рулетка)**: вероятности нормируются после сдвига на минимальное значение в поколении (устойчиво к отрицательным фитнессам). Функция: `reproduction(population: list[list[int]], fitnesses: list[float]) -> list[list[int]]`.
- **Кроссинговер**: одноточечный, попарно по перемешанной популяции. При нечётном размере последняя особь скрещивается со случайной другой особью (возможно повторное участие в кроссинговере одной из особей). Пары скрещиваются с вероятностью p_c, иначе родители добавляются в новую популяцию без изменений. Функции:
  - `crossover_pair(p1: list[int], p2: list[int], pc: float) -> (list[int], list[int])`
  - `crossover(population: list[list[int]], pc: float) -> list[list[int]]`
- **Мутация**: с вероятностью p_m *на хромосому* инвертируется ровно один случайный бит. Функция: `mutation(chrom: list[int], pm: float) -> None`.

- **Критерий остановки**: поддержаны критерии по дисперсии значений фитнесс функции в популяции и по среднему значению (настраиваемые пороги), но для экспериментов в данной лабораторной работе используется только критерий по среднему значению. Хранится история лучших особей и всех популяций по поколениям. Проверка выполняется внутри функции:

  `genetic_algorithm(config: GARunConfig) -> GARunResult` (используются поля `variance_threshold`, `fitness_avg_threshold` из `GARunConfig`).

- **Визуализация**: снимок поколения включает график целевой функции, точки всей текущей популяции, лучшую особь (выделена красным цветом), а также историю лучших по предыдущим поколениям (выделены оранжевым цветом). Функция: `plot_generation_snapshot(... ) -> str`.

- **Измерение времени**: длительность вычислений возвращается в миллисекундах; для сравнения параметров в серии экспериментов введён единый формат вывода. Значение доступно как `GARunResult.time_ms` из `genetic_algorithm(config: GARunConfig) -> GARunResult`.

- **Файловая организация**: по желанию сохраняются изображения поколений в `results/`. Серии экспериментов пишут результаты иерархически: `experiments/`N`/pc_`p`/pm_`m`/` с агрегированной таблицей в `CSV` на уровне популяции. Задействованные функции:

  - `clear_results_directory(results_dir: str) -> None`
  - `run_single_experiment(pop_size: int, pc: float, pm: float) -> (float, int)`
  - `run_experiments_for_population(pop_size: int) -> PrettyTable`

В модуле `expirements.py` задаётся целевая функция (`target_function(x: float) -> float` $(\sin(x)/x^2)$) и другие параметры для экспериментов. Серийные запуски и сохранение результатов реализованы в функциях:

- `run_single_experiment(pop_size: int, pc: float, pm: float) -> (float, int)`,
- `run_experiments_for_population(pop_size: int) -> PrettyTable`,
- `main()`.

4 Результаты работы

На Рис. 2– 9 представлены результаты работы генетического алгоритма со следующими параметрами:

- $N = 15$ – размер популяции.
- $p_c = 0.5$ – вероятность кроссинговера.
- $p_m = 0.01$ – вероятность мутации.
- -0.049 – минимальное среднее значение фитнесс функции по популяции для остановки алгоритма. Настоящий минимум функции равен примерно -0.04957.

С каждым поколением точность найденного минимума становится выше. На втором поколении популяция еще распределена по всей области поиска решения. На поколении 8 особи «конденсируются» вблизи нужного нам экстремума, а на поколении 14 достигается заданная точность нахождения минимума функции.

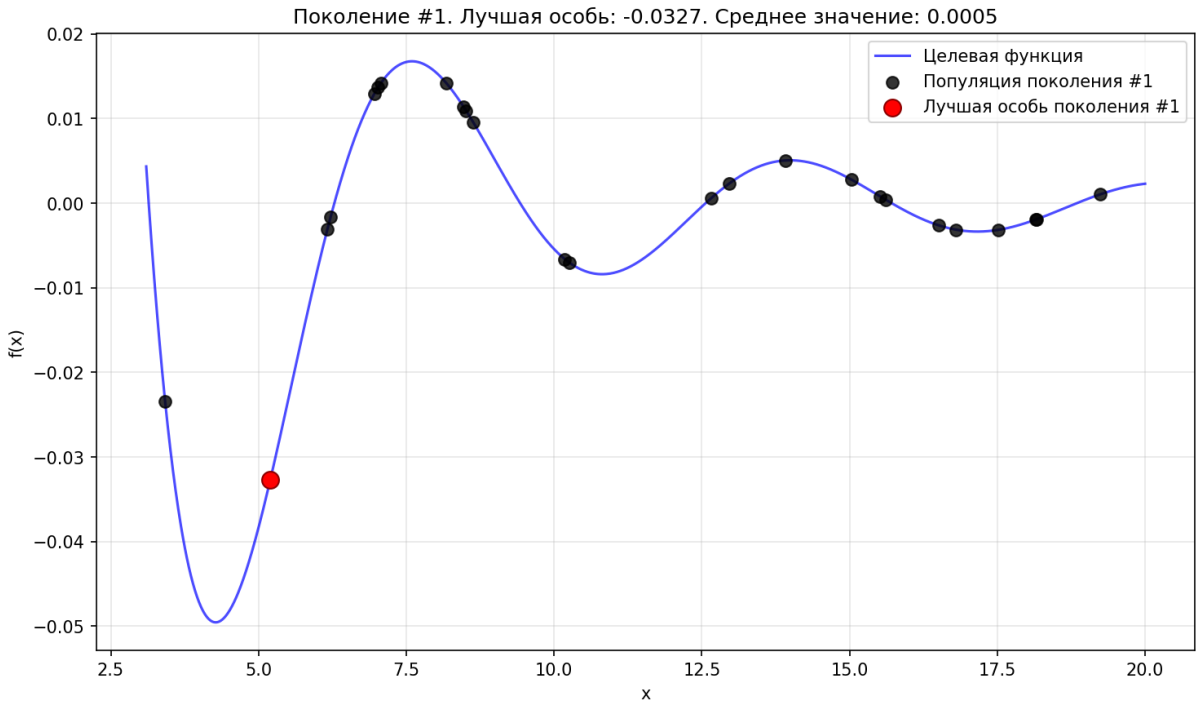


Рис. 2. График целевой функции и популяции поколения №1

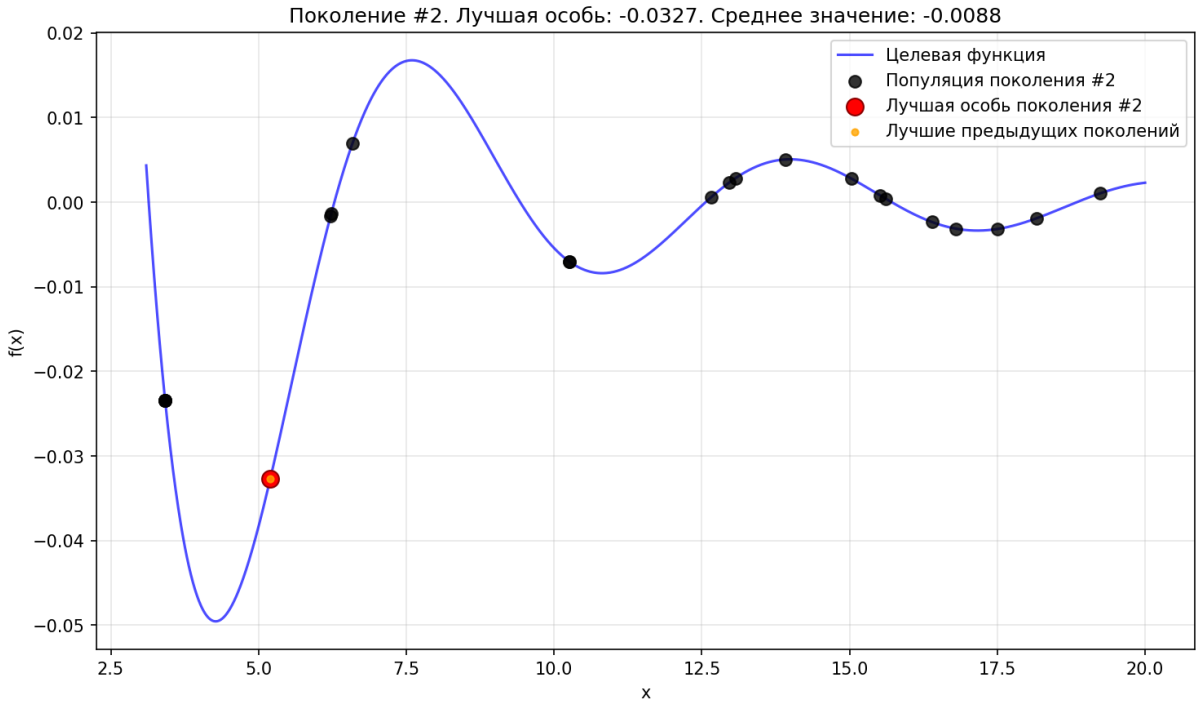


Рис. 3. График целевой функции и популяции поколения №2

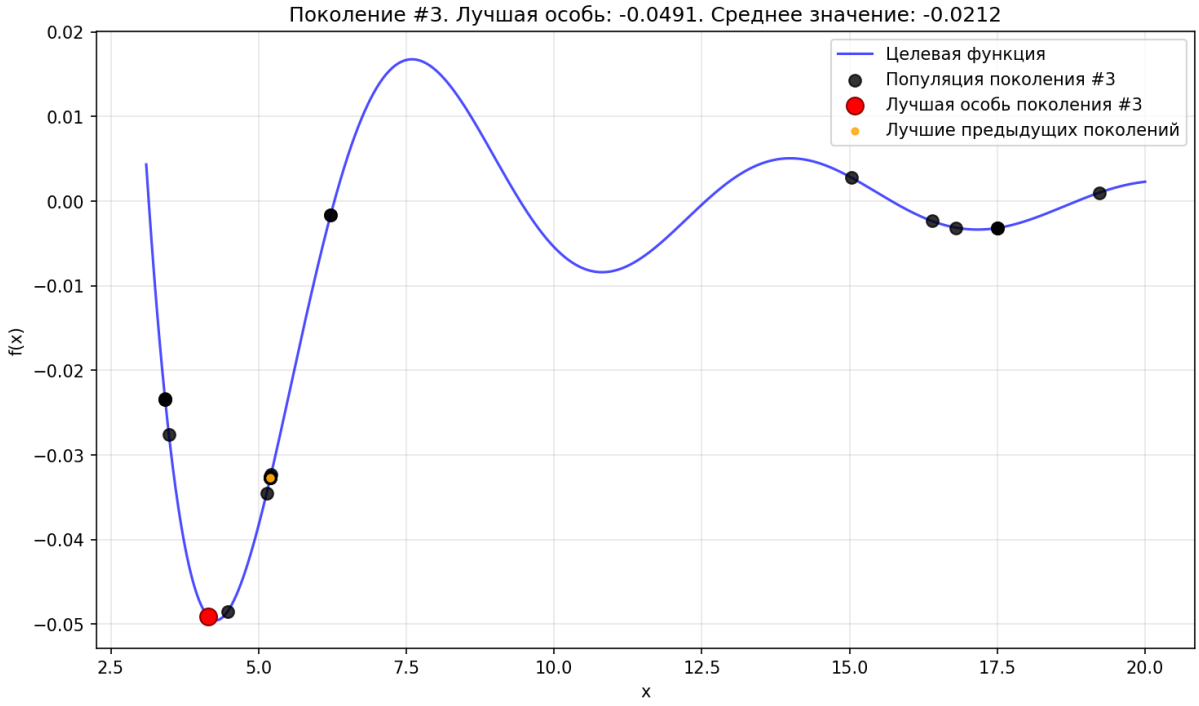


Рис. 4. График целевой функции и популяции поколения №3

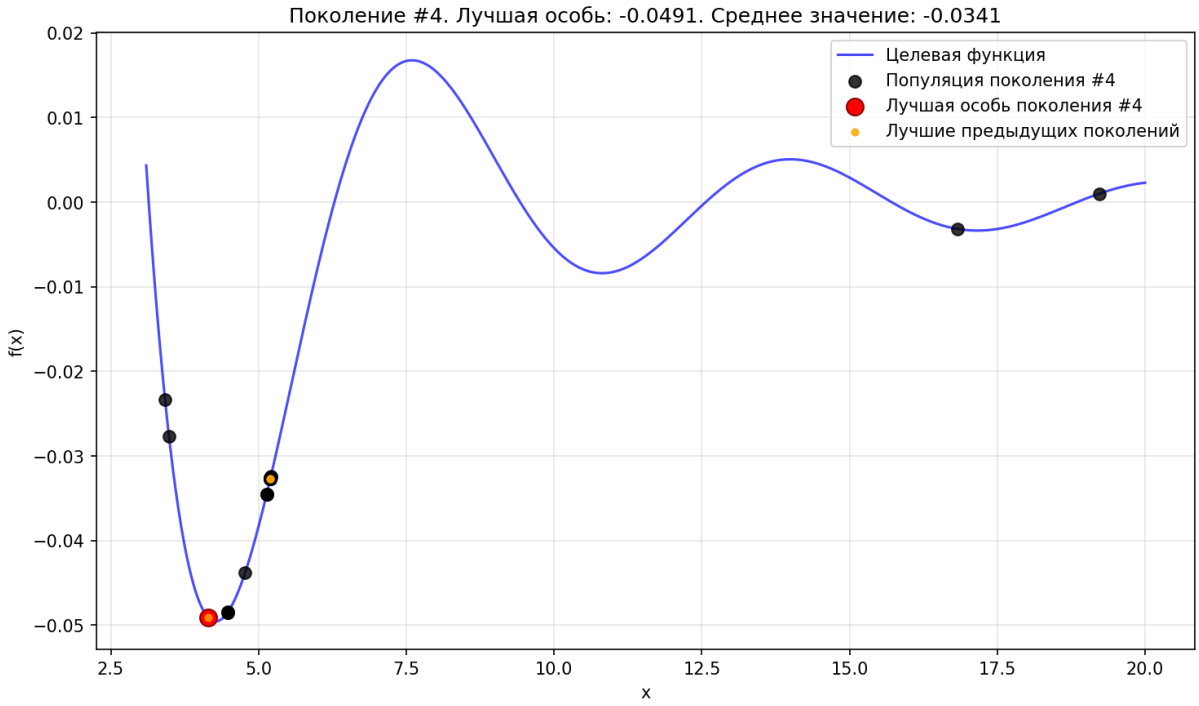


Рис. 5. График целевой функции и популяции поколения №4

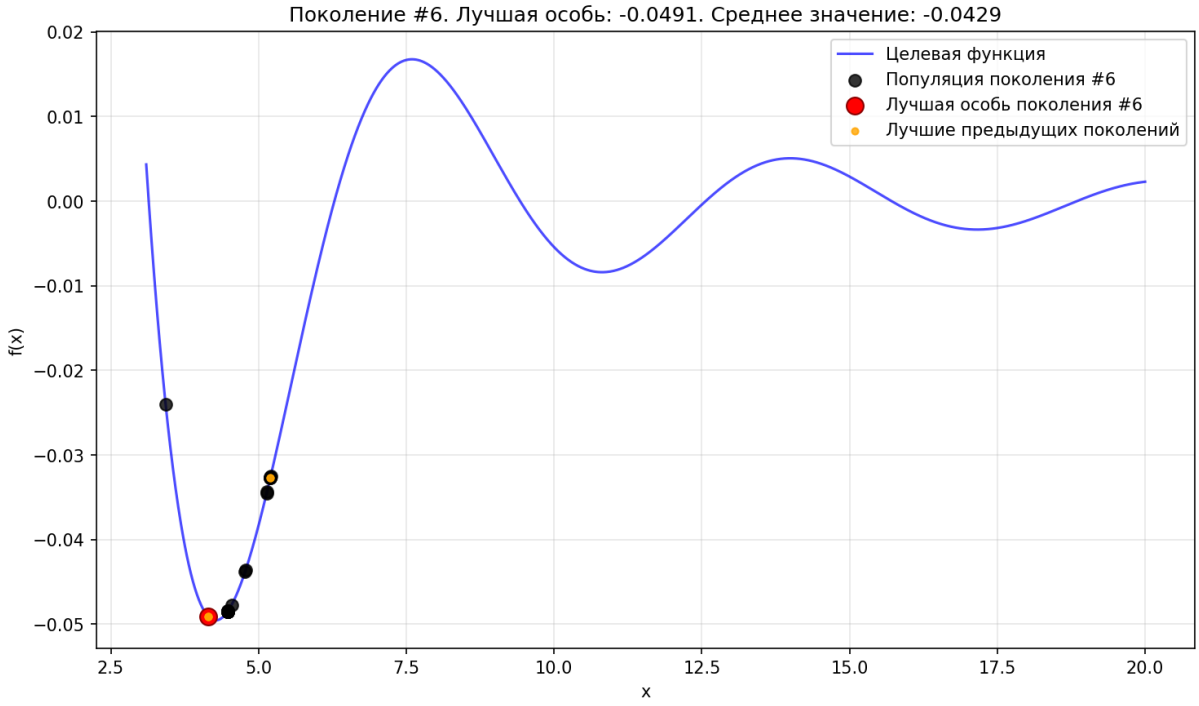


Рис. 6. График целевой функции и популяции поколения №6

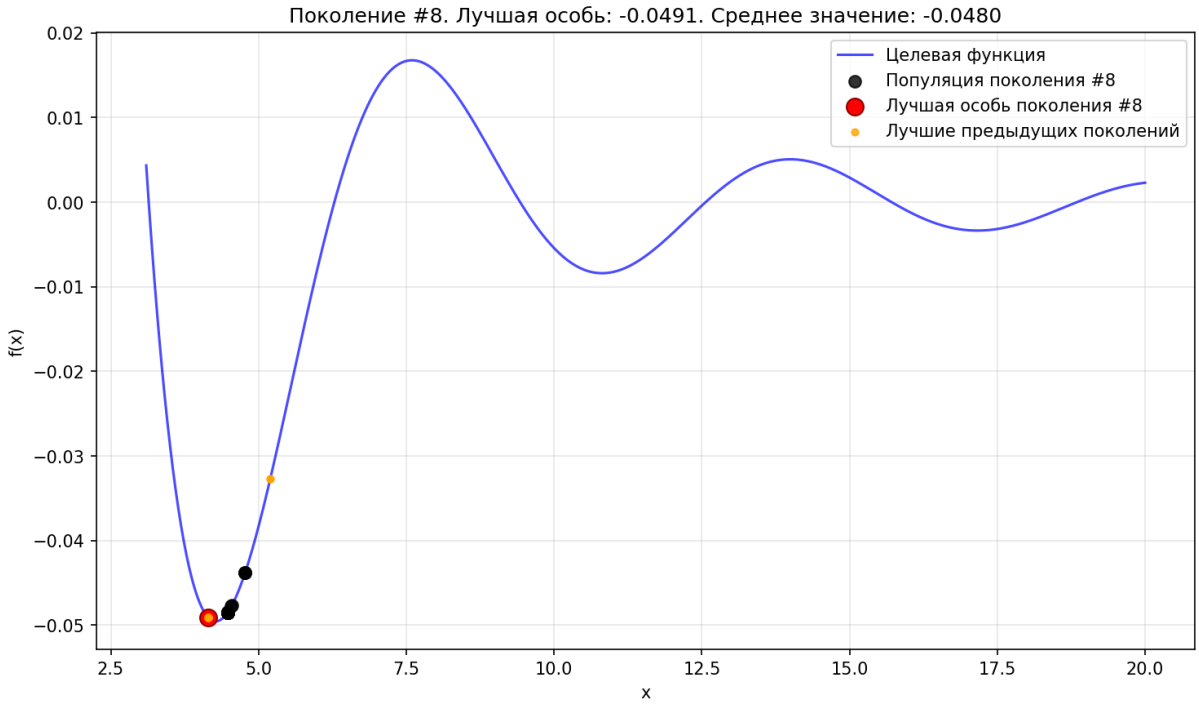


Рис. 7. График целевой функции и популяции поколения №8

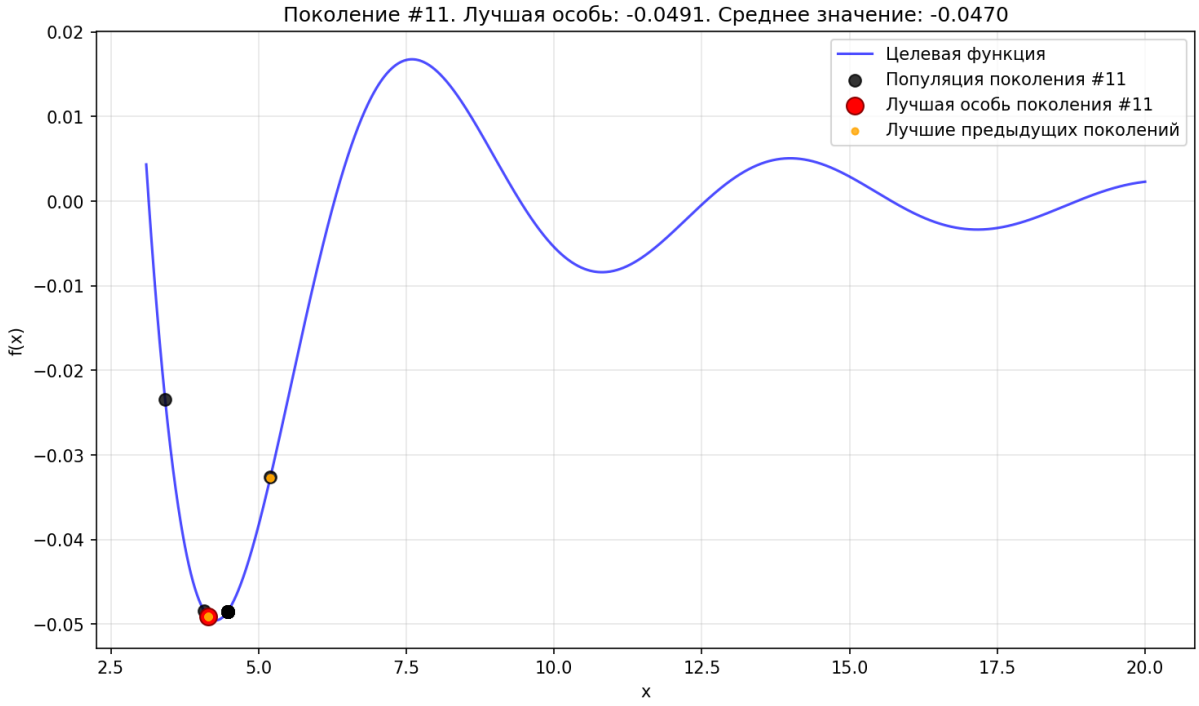


Рис. 8. График целевой функции и популяции поколения №11

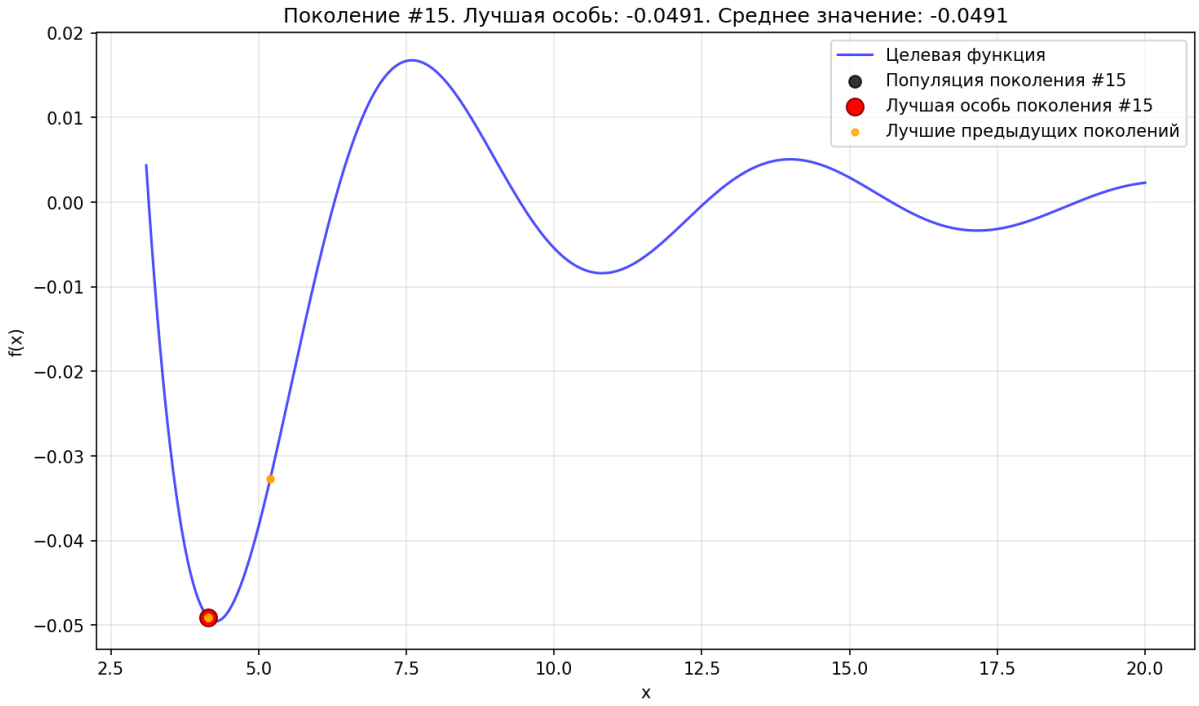


Рис. 9. График целевой функции и популяции поколения №15

5 Исследование реализации

5.1 Проведение измерений

В рамках лабораторной работы необходимо было исследовать зависимость времени выполнения задачи и количества поколений от популяции и вероятностей кроссинговера и мутации хромосомы

Для исследования были выбраны следующие значения параметров:

- $N = 10, 25, 50, 100$ – размер популяции.
- $p_c = 0.3, 0.4, 0.5, 0.6, 0.7, 0.8$ – вероятность кроссинговера.
- $p_m = 0.001, 0.01, 0.05, 0.1, 0.2$ – вероятность мутации.

Результаты измерений представлены в таблицах 1–4. В ячейках указано усредненное время в миллисекундах нахождения минимума функции. В скобках указано усредненное количество поколений, за которое было найдено решение. Каждый эксперимент запускался 30 раз. Если в ячейке стоит прочерк, то это означает, что решение не было найдено за 200 поколений. Лучшее значение по времени выполнения для каждого размера популяции выделено жирным шрифтом.

Таблица 1. Результаты для $N = 10$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.001**	**0.010**	**0.050**	**0.100**	**0.200**
0.3	11.5 (167)	8.4 (123)	5.4 (78)	4.9 (71)	3.3 (48)
0.4	10.1 (144)	7.1 (104)	6.3 (92)	4.7 (67)	4.7 (67)
0.5	11.4 (168)	7.7 (112)	5.4 (79)	6.1 (83)	**3.1 (44)**
0.6	11.0 (160)	6.7 (97)	4.9 (70)	4.7 (67)	5.3 (74)
0.7	12.1 (174)	9.3 (135)	3.7 (52)	4.7 (67)	6.5 (92)
0.8	8.7 (126)	8.3 (119)	3.9 (57)	7.9 (113)	4.4 (61)

Таблица 2. Результаты для $N = 25$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.001**	**0.010**	**0.050**	**0.100**	**0.200**
0.3	14.7 (111)	8.2 (62)	4.9 (37)	4.7 (35)	8.7 (63)
0.4	12.8 (95)	7.3 (54)	4.7 (35)	4.3 (32)	8.2 (61)
0.5	10.5 (78)	5.4 (40)	**2.2 (16)**	5.5 (40)	12.1 (89)
0.6	14.0 (103)	6.5 (48)	3.4 (25)	4.0 (30)	14.0 (87)
0.7	11.5 (84)	6.2 (46)	3.0 (22)	3.2 (24)	11.6 (83)
0.8	9.2 (64)	5.8 (41)	2.5 (18)	3.0 (22)	11.2 (78)

Таблица 3. Результаты для $N = 50$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.001**	**0.010**	**0.050**	**0.100**	**0.200**
0.3	6.1 (26)	5.2 (22)	6.3 (26)	11.6 (48)	40.2 (147)
0.4	6.1 (26)	4.5 (19)	5.2 (22)	9.8 (40)	37.2 (149)
0.5	10.5 (44)	4.9 (20)	7.6 (28)	17.1 (65)	36.2 (144)
0.6	7.5 (31)	**4.6 (19)**	5.6 (23)	18.8 (76)	42.0 (158)
0.7	7.6 (31)	4.7 (20)	7.6 (31)	13.9 (55)	34.3 (136)
0.8	10.8 (44)	5.0 (21)	6.1 (24)	13.9 (56)	36.5 (145)

Таблица 4. Результаты для $N = 100$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.001**	**0.010**	**0.050**	**0.100**	**0.200**
0.3	**7.6 (16)**	9.5 (21)	29.0 (60)	62.9 (128)	–
0.4	8.0 (17)	9.6 (21)	31.5 (68)	56.6 (120)	–
0.5	9.1 (20)	9.2 (20)	22.6 (48)	59.4 (124)	–
0.6	17.8 (38)	12.3 (26)	30.0 (64)	61.1 (128)	95.3 (196)
0.7	10.0 (22)	14.3 (31)	30.3 (64)	49.1 (103)	–
0.8	16.4 (34)	12.1 (25)	31.4 (64)	54.9 (114)	–

5.2 Анализ результатов

Ключевые наблюдения:

- При небольших популяциях ($N = 10$) повышение p_m ускоряет поиск; наилучшее время при $p_c = 0.5$, $p_m = 0.2$ (3.1 мс, 44 пок.).
- Для $N = 25$ оптимум при умеренной мутации $p_m \in [0.05, 0.10]$; минимум времени при $p_c = 0.5$, $p_m = 0.05$ (2.2 мс, 16 пок.) — лучшее среди всех экспериментов.
- Для $N = 50$ лучшее время при $p_c = 0.6$, $p_m = 0.01$ (4.6 мс, 19 пок.). Слишком большая мутация ($p_m = 0.2$) резко ухудшает результаты.
- Для $N = 100$ оптимальны низкие p_m; лучший результат при $p_c = 0.3$, $p_m = 0.001$ (7.6 мс, 16 пок.). При $p_m = 0.2$ решение часто не находится за 200 поколений.
- Рост N не гарантирует ускорения: число поколений может уменьшаться, но суммарное время часто растёт из-за большей стоимости одной итерации.

Практические выводы:

- Для умеренных затрат времени и стабильной сходимости разумно выбирать $N \approx 25$–50, $p_c \approx 0.5$–0.6, $p_m \approx 0.01$–0.05.
- Оптимальное p_m снижается с ростом N: при малых популяциях полезна более агрессивная мутация, при больших — слабая.
- Слишком большие значения p_m и p_c могут разрушать хорошие решения и ухудшать сходимость; стоит избегать $p_m \geq 0.2$ и высоких p_c при больших N.

6 Ответ на контрольный вопрос

Вопрос: Какую роль в ГА играет оператор репродукции (ОР)?

Ответ: Оператор репродукции (ОР) в ГА играет роль селекции. Он выбирает наиболее приспособленных особей для дальнейшего участия в скрещивании и мутации. Это позволяет сохранить наиболее приспособленные особи и постепенно улучшить популяцию.

Заключение

В ходе первой лабораторной работы:

1. Был изучен теоретический материал, основная терминология ГА, генетические операторы, использующиеся в простых ГА;
2. Реализована программа на языке Python для нахождения минимума заданной функции;
3. Проведено исследование зависимости времени выполнения программы и поколения от мощности популяции и коэффициентов кроссинговера и мутации.

Список литературы

[1] Методические указания по выполнению лабораторных работ к курсу «Генетические алгоритмы», 119 стр.