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 №2
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

Индивидуальное задание вариант 18:

Дано: Функция Axis parallel hyper-ellipsoid function.

Общая формула для n-мерного случая:

$$f(\mathbf{x}) = \sum_{i=1}^{n} i \cdot x_i^2$$

где $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, область определения $x_i \in [-5.12, 5.12]$ для всех $i = 1, \ldots, n$.

Для двумерного случая (n=2):

$$f(x, y) = 1 \cdot x^2 + 2 \cdot y^2 = x^2 + 2y^2$$

область нахождения решения $x \in [-5.12, 5.12], y \in [-5.12, 5.12]$.

Глобальный минимум: $f(\mathbf{x}) = 0$ в точке $x_i = 0$ для всех $i = 1, \ldots, n$. Для двумерного случая: $\min f(x, y) = f(0, 0) = 0$.

Требуется:

1. Создать программу, использующую генетический алгоритм для нахождения минимума данной функции;
2. Для n=2 вывести на экран график функции с указанием найденного экстремума и точек популяции. Предусмотреть возможность пошагового просмотра процесса поиска решения;
3. Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма: числа особей в популяции, вероятности кроссинговера и мутации;
4. Повторить процесс поиска решения для n=3, сравнить результаты и скорость работы программы.

2 Теоретические сведения

Генетические 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. Однако при работе с оптимизационными задачами в непрерывных пространствах вполне естественно представлять гены напрямую вещественными числами. В этом случае хромосома есть вектор вещественных чисел (real-coded алгоритмы). Их точность определяется исключительно разрядной сеткой ЭВМ. Длина хромосомы совпадает с длиной вектора-решения оптимизационной задачи, каждый ген отвечает за одну переменную. Генотип объекта становится идентичным его фенотипу.

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. Может быть представлена как двоичная последовательность (где $s_i = 0, 1$) или как вектор вещественных чисел (real-coded представление).

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 Операторы кроссинговера для real-coded алгоритмов

Оператор скрещивания непрерывного ГА (кроссовер) порождает одного или нескольких потомков от двух хромосом. Требуется из двух векторов вещественных

чисел получить новые векторы по определённым законам. Большинство real-coded алгоритмов генерируют новые векторы в окрестности родительских пар.

Пусть $C_1 = (c_{11}, c_{21}, \ldots, c_{n1})$ и $C_2 = (c_{12}, c_{22}, \ldots, c_{n2})$ – две хромосомы, выбранные оператором селекции для скрещивания.

Арифметический кроссовер (arithmetical crossover): создаются два потомка $H_1 = (h_{11}, \ldots, h_{n1})$, $H_2 = (h_{12}, \ldots, h_{n2})$, где:

$$h_{k1} = w \cdot c_{k1} + (1 - w) \cdot c_{k2}$$

$$h_{k2} = w \cdot c_{k2} + (1 - w) \cdot c_{k1}$$

где $k = 1, \ldots, n$, w – весовой коэффициент из интервала $[0; 1]$.

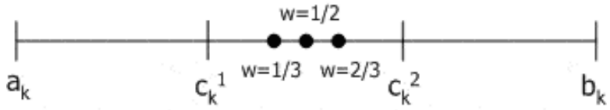

Рис. 2. Арифметический кроссовер

Геометрический кроссовер (geometrical crossover): создаются два потомка $H_1 = (h_{11}, \ldots, h_{n1})$, $H_2 = (h_{12}, \ldots, h_{n2})$, где:

$$h_{k1} = (c_{k1})^w \cdot (c_{k2})^{(1-w)}$$

$$h_{k2} = (c_{k2})^w \cdot (c_{k1})^{(1-w)}$$

где w – случайное число из интервала $[0; 1]$.

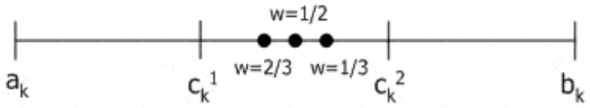

Рис. 3. Геометрический кроссовер

Смешанный кроссовер (BLX-alpha crossover): генерируется один потомок $H = (h_1, \ldots, h_k, \ldots, h_n)$, где h_k – случайное число из интервала $[c_{min} - I \cdot \alpha, c_{max} + I \cdot \alpha]$, $c_{min} = \min(c_{k1}, c_{k2})$, $c_{max} = \max(c_{k1}, c_{k2})$, $I = c_{max} - c_{min}$.

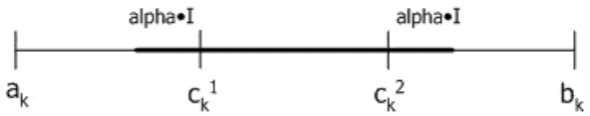

Рис. 4. Смешанный кроссовер

SBX кроссовер (Simulated Binary Crossover): кроссовер, имитирующий двоичный, разработанный в 1995 году исследовательской группой под руководством K. Deb'a. Моделирует принципы работы двоичного оператора скрещивания, сохраняя важное свойство – среднее значение функции приспособленности остаётся неизменным у родителей и их потомков.

Создаются два потомка $H_k = (h_{1k}, \ldots, h_{jk}, \ldots, h_{nk})$, $k = 1, 2$, где:

$$h_{j1} = 0.5[(1 + \beta_k)c_{j1} + (1 - \beta_k)c_{j2}]$$

$$h_{j2} = 0.5[(1 - \beta_k)c_{j1} + (1 + \beta_k)c_{j2}]$$

где $\beta_k \geq 0$ – число, полученное по формуле:

$$\beta_k = \begin{cases} (2u)^{\frac{1}{n+1}}, & \text{при } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n+1}}, & \text{при } u > 0.5 \end{cases}$$

где $u \in (0, 1)$ – случайное число, распределённое по равномерному закону, $n \in [2, 5]$ – параметр кроссовера. Увеличение n повышает вероятность появления потомка в окрестности родителей.

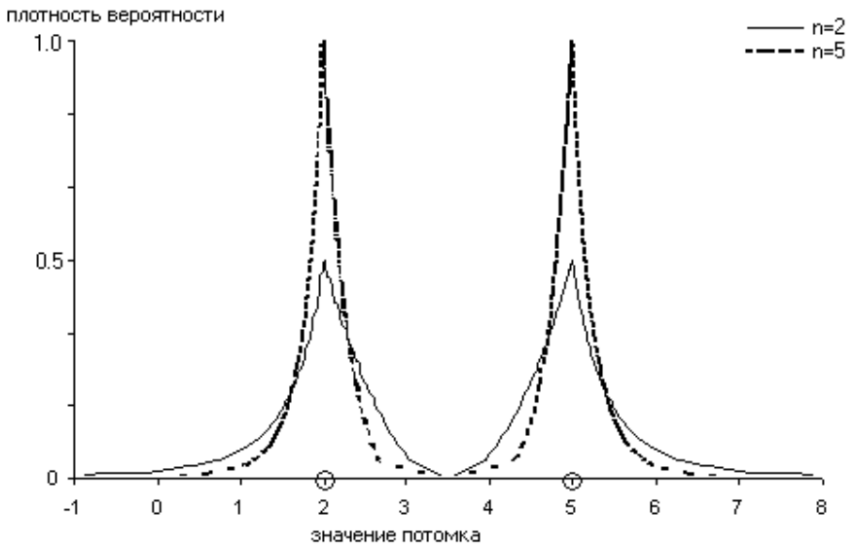

Рис. 5. SBX кроссовер

2.2.3 Операторы мутации для real-coded алгоритмов

В качестве оператора мутации наибольшее распространение получили: случайная и неравномерная мутация.

Случайная мутация (random mutation): ген, подлежащий изменению, принимает случайное значение из интервала своего изменения.

Неравномерная мутация (non-uniform mutation): из особи случайно выбирается точка c_k с разрешёнными пределами изменения $[c_{kl}, c_{kr}]$. Точка меняется на:

$$c'_k = \begin{cases} c_k + \Delta(t, c_{kr} - c_k), & \text{при } a = 1 \\ c_k - \Delta(t, c_k - c_{kl}), & \text{при } a = 0 \end{cases}$$

где a – случайно выбранное направление изменения, $\Delta(t, y)$ – функция, возвращающая случайную величину в пределах $[0, y]$ таким образом, что при увеличении t

среднее возвращаемое значение уменьшается:

$$\Delta(t, y) = y \cdot r \cdot \left(1 - \frac{t}{T}\right)^b$$

где r – случайная величина на интервале $[0, 1]$, t – текущая эпоха работы генетического алгоритма, T – общее разрешённое число эпох алгоритма, b – задаваемый пользователем параметр, определяющий степень зависимости от числа эпох.

3 Особенности реализации

В рамках работы создана мини-библиотека `gen.py` для экспериментов с real-coded генетическим алгоритмом для многомерных функций. Второй модуль `expirements.py` организует серийные эксперименты (перебор параметров, форматирование и сохранение результатов).

- **Кодирование особей**: каждая хромосома представлена как `np.ndarray` вещественных чисел (`Chromosome = NDArray[np.float64]`). Длина хромосомы соответствует размерности задачи оптимизации. Популяция – список хромосом (`Population = list[Chromosome]`). Инициализация случайными векторами в заданном диапазоне:
  - `initialize_population(pop_size: int, x_min: Chromosome, x_max: Chromosome) -> Population`
- **Фитнесс и минимум/максимум**: целевая функция принимает хромосому (вектор) и возвращает скалярное значение фитнесса. Для режима минимизации используется внутреннее преобразование при селекции (сдвиг на минимальное значение), что позволяет применять рулетку при отрицательных значениях:
  - `eval_population(population: Population, fitness_func: FitnessFn) -> Fitnesses`
  - 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: Population, fitnesses: Fitnesses) -> Population`.
- **Кроссинговер**: реализованы арифметический и геометрический кроссоверы для real-coded алгоритмов. Кроссинговер выполняется попарно по перемешанной популяции с вероятностью p_c. Функции:
  - `arithmetical_crossover_fn(p1: Chromosome, p2: Chromosome, w: float) -> tuple[Chromosome, Chromosome]`
  - `geometrical_crossover_fn(p1: Chromosome, p2: Chromosome, w: float) -> tuple[Chromosome, Chromosome]`
  - `crossover(population: Population, pc: float, crossover_fn: CrossoverFn) -> Population`
- **Мутация**: случайная мутация – с вероятностью p_m на хромосому изменяется один случайно выбранный ген на случайное значение из допустимого диапазона. Функции:
  - `build_random_mutation_fn(x_min: Chromosome, x_max: Chromosome) -> MutationFn`

  - `mutation(population: Population, pm: float, mutation_fn: MutationFn) -> Population`

- **Критерий остановки**: поддерживается критерий по среднему значению фитнесс-функции в популяции и максимальному количеству поколений. Хранится история всех поколений. Проверка выполняется в функции:

  `genetic_algorithm(config: GARunConfig) -> GARunResult`.

- **Визуализация**: для двумерных функций реализованы 3D-графики поверхности и 2D-контурные графики с отображением популяций. Функции:

  - `plot_fitness_surface(fitness_func: FitnessFn, x_min: Chromosome, x_max: Chromosome, ax: Axes3D)`
  - `plot_fitness_contour(fitness_func: FitnessFn, x_min: Chromosome, x_max: Chromosome, ax: Axes)`
  - `save_generation(generation: Generation, history: list[Generation], config: GARunConfig)`

- **Измерение времени**: длительность вычислений возвращается в миллисекундах как часть `GARunResult.time_ms`.

- **Файловая организация**: результаты экспериментов сохраняются иерархически в структуре `experiments/N/` с таблицами результатов в формате CSV. Задействованные функции:

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

В модуле `expirements.py` задаётся целевая функция axis parallel hyper-ellipsoid: $f(x, y) = x^2 + 2y^2$ и параметры экспериментов. Серийные запуски и сохранение результатов реализованы в функциях `run_single_experiment`, `run_experiments_for_population` и `main`.

4 Результаты работы

На Рис. 6–14 представлены результаты работы генетического алгоритма со следующими параметрами:

- $N = 25$ – размер популяции.
- $p_c = 0.5$ – вероятность кроссинговера.
- $p_m = 0.01$ – вероятность мутации.
- Алгоритм останавливался, если лучшее значение фитнеса не изменялось 10 поколений подряд.
- Использован арифметический кроссовер для real-coded хромосом.

Популяция постепенно консолидируется вокруг глобального минимума в точке $(0, 0)$. Лучшая особь была найдена на поколнении №9 (см. Рис. 11), но судя по всему она подверглась мутации или кроссинговеру, поэтому алгоритм не остановился. На поколении №19 (см. Рис. 14) было получено значение фитнеса 0.0201, которое затем повторялось в следующих 10 поколениях. Алгоритм остановился на поколлении №29. На графиках показаны 2D-контурный график (a) и 3D-поверхность целевой функции с точками популяции текущего поколения (b) и (c).

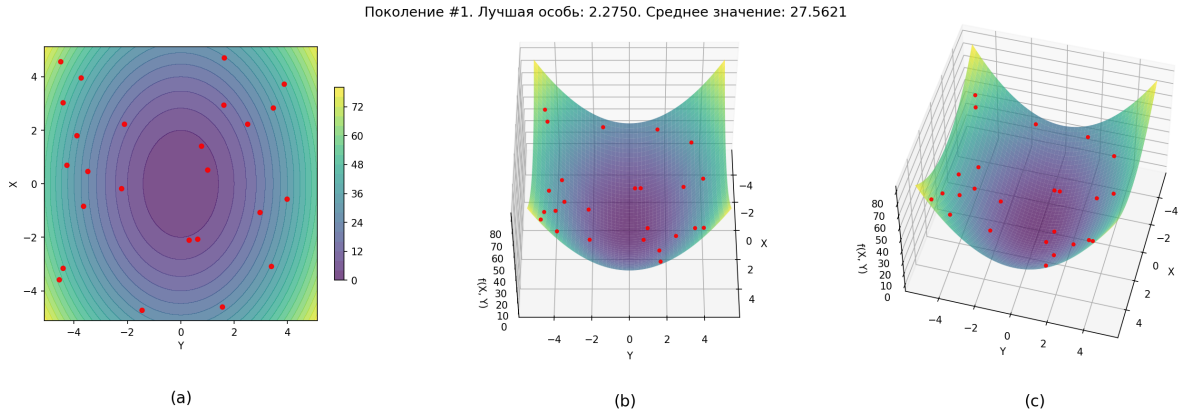

Рис. 6. График целевой функции и популяции поколения №1

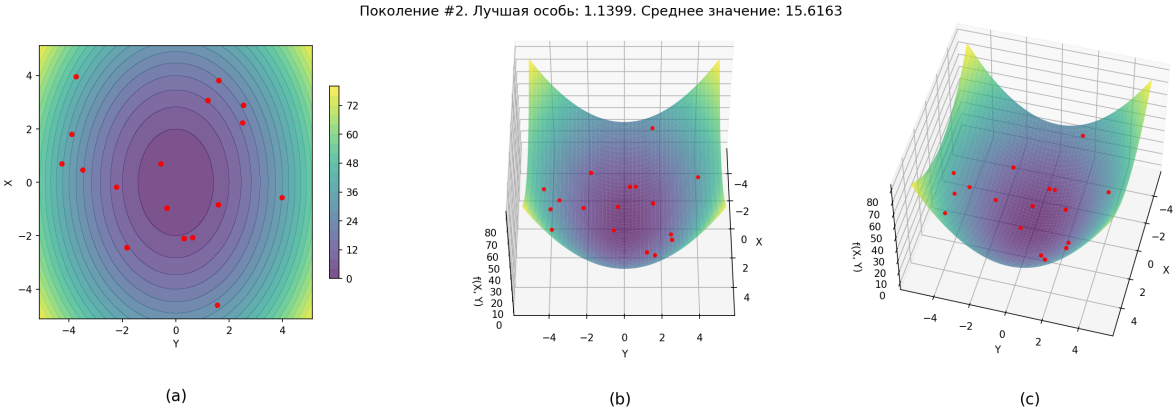

Рис. 7. График целевой функции и популяции поколения №2

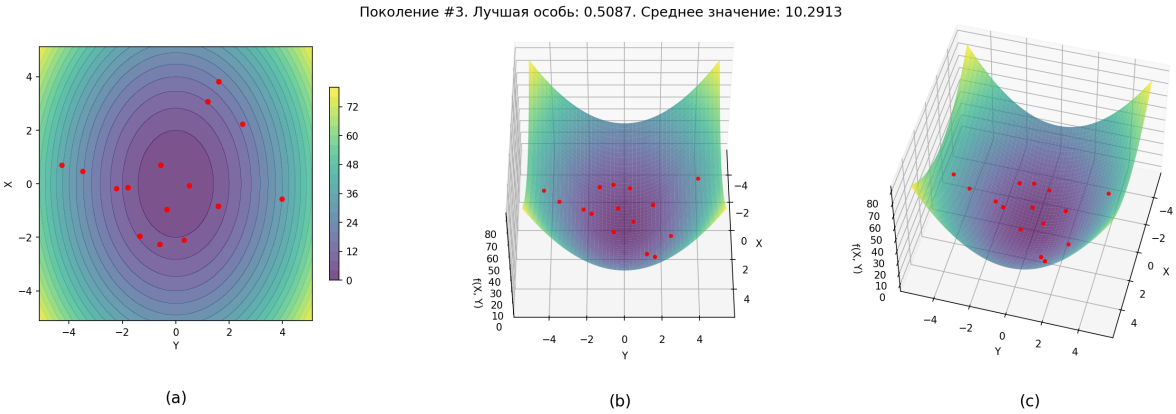

Рис. 8. График целевой функции и популяции поколения №3

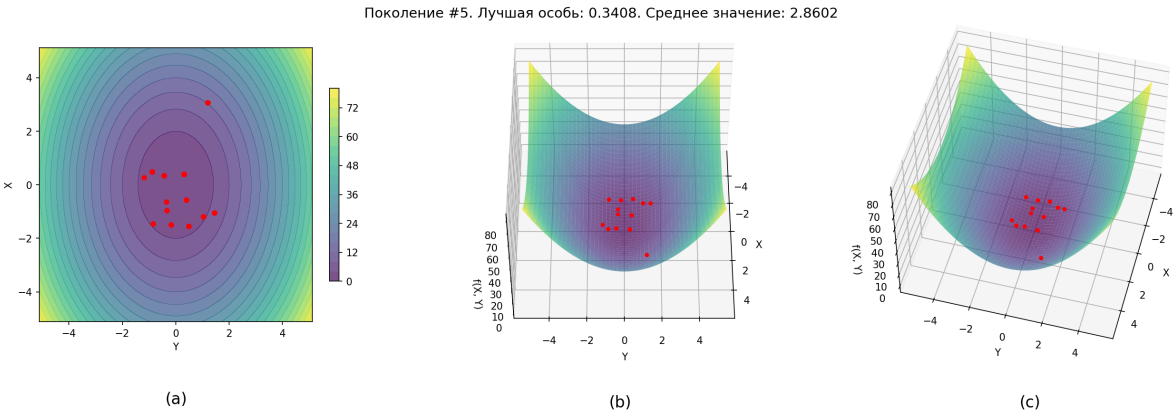

Рис. 9. График целевой функции и популяции поколения №5

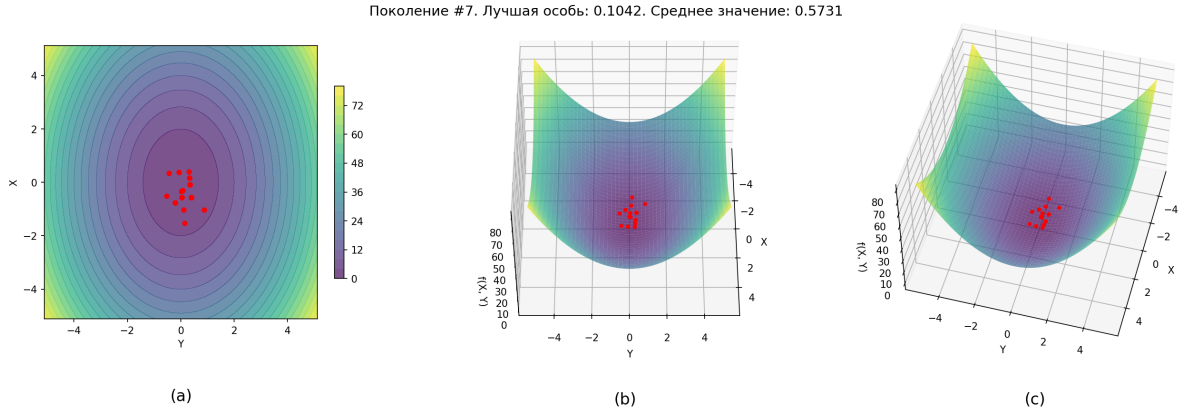

Рис. 10. График целевой функции и популяции поколения №7

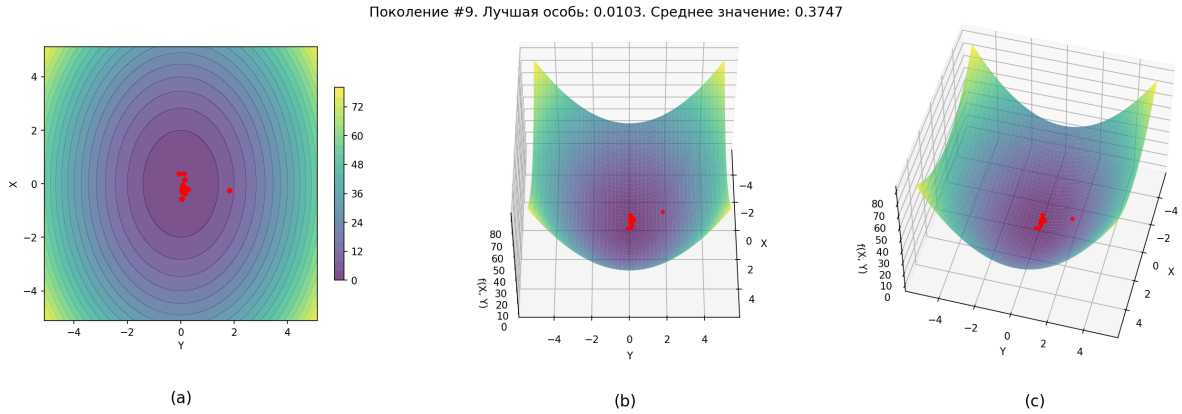

Рис. 11. График целевой функции и популяции поколения №9

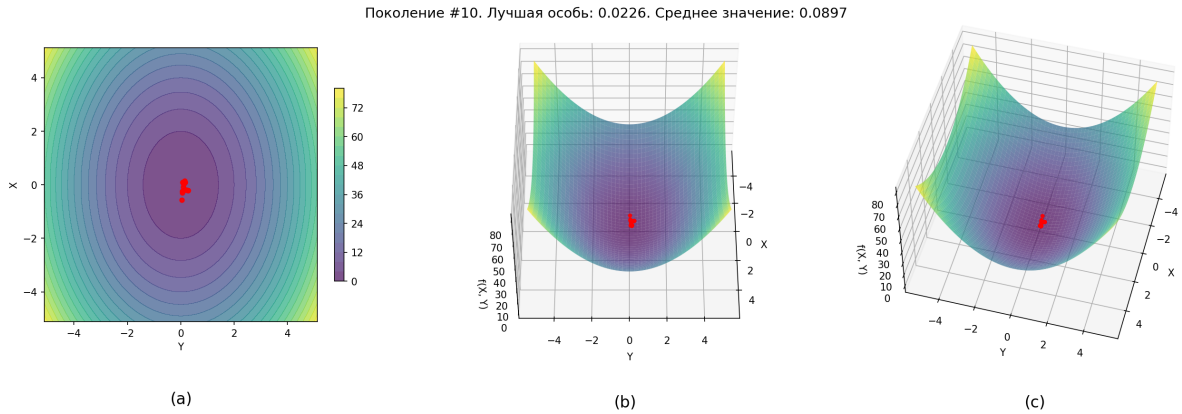

Рис. 12. График целевой функции и популяции поколения №10

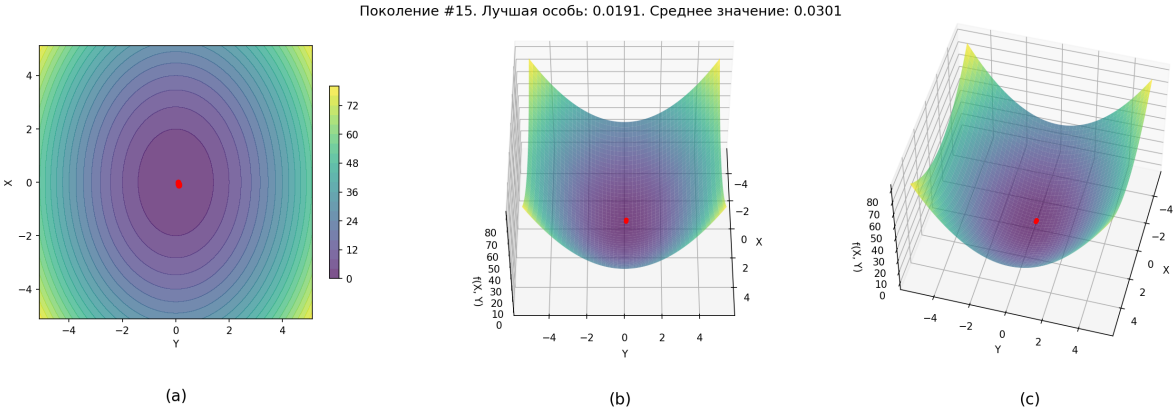

Рис. 13. График целевой функции и популяции поколения №15

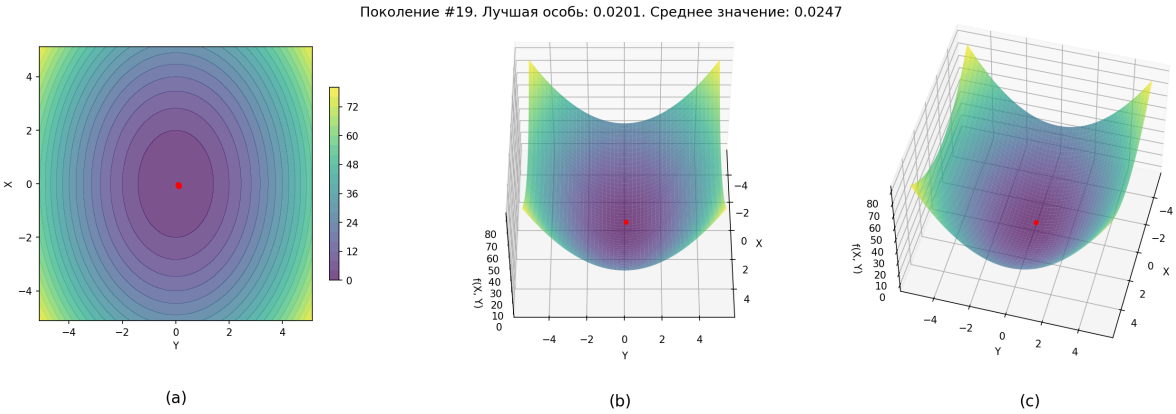

Рис. 14. График целевой функции и популяции поколения №19

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.

Измерения были проведены для двух критериев остановки:

- Лучшее значение фитнеса не изменялось 10 поколений.
- Лучшее значение фитнеса достигло заданного значения 0.005.

Результаты для первого критерия остановки

Результаты измерений представлены в таблицах 1–4. В ячейках указано время в миллисекундах нахождения минимума функции. В скобках указано количество поколений, за которое было найдено решение. Во второй строке указано усреднённое по всем запускам лучшее значение фитнеса. Если в ячейке стоит прочерк, то это означает, что решение не было найдено за 200 поколений. Лучшее значение по времени выполнения и по значению фитнеса для каждого размера популяции выделено цветом и жирным шрифтом.

Таблица 1. Результаты для $N = 10$

$P_c \setminus P_m$	0.001	0.010	0.050	0.100	0.200
0.3	1.3 (13) 0.36281	1.7 (18) 7.55685	1.2 (13) 1.55537	**1.0 (11)** 1.78411	9.4 (87) 0.04271
0.4	1.3 (14) 0.03913	1.6 (17) 0.02868	1.3 (13) 0.36232	2.1 (20) 0.10641	—
0.5	1.4 (15) 0.87081	1.7 (18) 1.71634	2.3 (21) 0.10401	3.4 (25) 0.00461	—
0.6	2.8 (19) 0.06375	1.8 (13) 0.72202	2.9 (22) 0.01473	3.4 (25) 0.01162	29.4 (184) **0.00033**
0.7	1.5 (15) 1.25409	2.3 (22) 8.67464	1.9 (18) 0.13319	8.6 (66) 0.00078	8.9 (48) 0.11136
0.8	1.9 (15) 3.10415	1.4 (13) 1.09275	2.1 (19) 0.43094	6.4 (54) 0.00191	—

Таблица 2. Результаты для $N = 25$

$P_c \setminus P_m$	0.001	0.010	0.050	0.100	0.200
0.3	3.0 (18) 0.16836	**2.2 (13)** 0.04190	4.7 (27) 0.00544	—	—
0.4	4.1 (24) 0.00808	4.6 (26) 0.01101	5.8 (31) 0.02330	3.8 (19) 0.05414	—
0.5	3.1 (17) 0.05259	5.0 (26) 0.47018	27.8 (138) **0.00024**	14.5 (67) 0.00312	—
0.6	6.1 (31) 0.01033	6.8 (34) 0.00148	—	—	—
0.7	4.1 (21) 0.00107	3.2 (16) 0.32522	—	—	—
0.8	23.9 (109) 0.00352	15.8 (72) 0.11662	28.3 (123) 0.00038	—	—

Таблица 3. Результаты для $N = 50$

$P_c \setminus P_m$	0.001	0.010	0.050	0.100	0.200
0.3	14.9 (51) 0.05874	19.3 (59) **0.00003**	36.7 (113) 0.00190	—	—
0.4	12.5 (40) 0.01955	**5.6 (18)** 0.00022	—	—	—
0.5	65.0 (195) 0.04790	26.4 (78) 0.01673	—	—	—
0.6	16.4 (47) 0.00329	18.5 (50) 0.00065	—	—	—
0.7	51.0 (137) 0.00120	59.3 (158) 0.00010	—	—	—
0.8	48.8 (126) 0.01393	67.6 (172) 0.00650	—	—	—

Таблица 4. Результаты для $N = 100$

$P_c \setminus P_m$	0.001	0.010	0.050	0.100	0.200
0.3	24.2 (44) 0.00110	17.9 (32) 0.00113	**17.6 (29)** 0.00193	—	—
0.4	30.7 (51) 0.00173	—	—	—	—
0.5	27.4 (43) 0.00016	—	—	—	—
0.6	20.4 (31) 0.00115	129.8 (186) 0.00025	—	—	—
0.7	115.4 (162) 0.00002	—	—	—	—
0.8	106.5 (143) **0.00001**	—	—	—	—

Результаты для второго критерия остановки

Результаты измерений представлены в таблицах 5–8.

Таблица 5. Результаты для $N = 10$

$P_c \setminus P_m$	0.001	0.010	0.050	0.100	0.200
0.3	—	—	—	15.6 (155) 0.00063	7.8 (69) 0.00409
0.4	—	—	—	8.9 (81) **0.00038**	**4.6 (40)** 0.00317
0.5	—	—	8.7 (85) 0.00199	—	16.5 (140) 0.00453
0.6	—	—	—	8.9 (77) 0.00310	14.3 (117) 0.00082
0.7	—	—	8.2 (70) 0.00089	5.6 (49) 0.00431	7.1 (58) 0.00047
0.8	—	19.7 (180) 0.00397	—	5.0 (42) 0.00494	5.5 (44) 0.00357

Таблица 6. Результаты для $N = 25$

$P_c \setminus P_m$	0.001	0.010	0.050	0.100	0.200
0.3	1.1 (7) 0.00277	30.0 (173) **0.00059**	—	2.2 (12) 0.00191	30.2 (139) 0.00200
0.4	1.8 (10) 0.00384	—	12.2 (63) 0.00164	6.6 (33) 0.00354	18.5 (82) 0.00224
0.5	—	—	12.5 (58) 0.00233	2.3 (11) 0.00196	17.1 (73) 0.00116
0.6	—	30.9 (151) 0.00265	36.7 (175) 0.00146	10.0 (46) 0.00449	5.7 (23) 0.00281
0.7	1.1 (6) 0.00472	—	0.8 (4) 0.00233	3.9 (17) 0.00112	**0.3 (2)** 0.00371
0.8	—	—	10.3 (43) 0.00137	7.7 (32) 0.00379	10.5 (41) 0.00155

Таблица 7. Результаты для $N = 50$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.001**	**0.010**	**0.050**	**0.100**	**0.200**
0.3	3.7 (12) 0.00354	3.4 (9) 0.00075	23.7 (73) 0.00467	4.9 (14) 0.00043	2.1 (6) 0.00029
0.4	3.6 (12) 0.00270	4.2 (13) 0.00061	9.2 (25) 0.00251	18.2 (51) 0.00490	6.6 (16) 0.00063
0.5	4.0 (10) 0.00099	48.8 (141) 0.00324	3.8 (11) 0.00087	14.7 (39) **0.00017**	1.2 (3) 0.00115
0.6	1.6 (5) 0.00070	51.6 (139) 0.00217	4.7 (13) 0.00294	2.6 (7) 0.00397	11.5 (27) 0.00053
0.7	—	—	2.6 (7) 0.00144	3.5 (9) 0.00182	**1.1 (3)** 0.00072
0.8	4.1 (11) 0.00240	3.5 (8) 0.00380	2.5 (6) 0.00422	2.7 (7) 0.00126	4.3 (10) 0.00060

Таблица 8. Результаты для $N = 100$

$\mathbf{P_c} \setminus \mathbf{P_m}$	**0.001**	**0.010**	**0.050**	**0.100**	**0.200**
0.3	9.3 (17) 0.00451	6.0 (11) 0.00344	10.0 (17) 0.00343	5.3 (8) 0.00046	9.8 (14) 0.00412
0.4	5.7 (9) **0.00005**	8.4 (14) 0.00108	3.5 (6) 0.00254	4.0 (6) 0.00186	6.5 (9) 0.00283
0.5	3.8 (6) 0.00019	4.9 (8) 0.00103	3.6 (6) 0.00260	11.1 (16) 0.00204	7.5 (10) 0.00374
0.6	—	6.5 (10) 0.00107	3.6 (5) 0.00079	**0.9 (2)** 0.00324	10.1 (13) 0.00044
0.7	1.7 (3) 0.00106	6.6 (10) 0.00489	4.1 (6) 0.00031	12.4 (16) 0.00240	4.8 (6) 0.00276
0.8	5.0 (7) 0.00387	58.4 (77) 0.00453	7.8 (10) 0.00259	11.2 (13) 0.00210	6.1 (7) 0.00493

5.2 Анализ результатов

Обоснование применения двух критериев остановки

В исследовании использовались два различных критерия остановки алгоритма, поскольку критерий по количеству поколений (отсутствие улучшения в течение 10 поколений) не всегда обеспечивал достижение достаточно хороших значений фитнеса, особенно для малых популяций. Это делало некорректным сравнение эффективности различных комбинаций параметров только по времени выполнения. Введение второго критерия (достижение фитнеса 0.005) позволило получить более объективную оценку скорости нахождении качественных решений.

Первый критерий остановки (отсутствие улучшения в течение 10 поколений)

При использовании первого критерия остановки наблюдаются следующие закономерности:

- **Малые популяции** ($N = 10$)**:** Оптимальный баланс достигается при умеренных значениях параметров. Лучший результат по времени показывает комбинация $p_c = 0.3$, $p_m = 0.1$ (1.0 мс, 11 поколений), однако лучшее значение фитнеса достигается при $p_c = 0.6$, $p_m = 0.2$ (0.00033). Качество решений существенно варьируется.

- **Средние популяции** ($N = 25$)**:** Демонстрируют высокую эффективность при низких значениях мутации. Минимальное время выполнения достигается при $p_c = 0.3$, $p_m = 0.01$ (2.2 мс, 13 поколений), а наилучший фитнес — при $p_c = 0.5$, $p_m = 0.05$ (0.00024).

- **Большие популяции** ($N = 50, 100$)**:** Характеризуются критической чувствительностью к высоким значениям мутации и демонстрируют заметное улучшение качества фитнеса. Для $N = 50$ лучшие результаты при $p_c = 0.4$, $p_m = 0.01$ (5.6 мс по времени) и $p_c = 0.3$, $p_m = 0.01$ (фитнес 0.00003). Для $N = 100$ работают только комбинации с очень низкой мутацией, но обеспечивают отличное качество (фитнес до 0.00001).

- **Проблема сходимости:** С увеличением размера популяции значительно возрастает количество комбинаций параметров, не обеспечивающих сходимость за 200 поколений, особенно при $p_m \geq 0.05$.

Второй критерий остановки (достижение фитнеса 0.005)

Использование фиксированного порога фитнеса демонстрирует принципиально иную картину и подтверждает правильность введения альтернативного критерия:

- **Инверсия требований к мутации:** В отличие от первого критерия, здесь малые популяции требуют более высоких значений мутации для достижения целевого фитнеса. Для $N = 10$ большинство комбинаций с $p_m \leq 0.01$ вообще не достигают порога, что подтверждает проблему качества при первом критерии.
- **Лучшие результаты больших популяций:** Популяции $N = 50$ и $N = 100$ показывают отличные результаты — достижение высокого качества за минимальное время: $N = 50$ при $p_c = 0.7$, $p_m = 0.2$ (1.1 мс, 3 поколения) и $N = 100$ при $p_c = 0.6$, $p_m = 0.1$ (0.9 мс, 2 поколения).

6 Ответ на контрольный вопрос

Вопрос: Опишите понятие «оптимизационная задача».

Ответ: Оптимизационная задача — это математическая задача, в которой требуется найти такие значения переменных, при которых некоторая функция, называемая целевой, принимает наибольшее или наименьшее значение. При этом искомые значения должны удовлетворять определённым условиям или ограничениям, задающим допустимую область решений. Цель оптимизации заключается в выборе наилучшего варианта среди множества возможных с точки зрения заданного критерия эффективности.

Такие задачи широко применяются в науке, технике, экономике и управлении для рационального распределения ресурсов, минимизации затрат или максимизации прибыли. В зависимости от формы целевой функции и ограничений оптимизационные задачи могут быть линейными, нелинейными, дискретными или непрерывными. Их решение позволяет принимать обоснованные решения и повышать эффективность различных процессов и систем.

Заключение

В ходе второй лабораторной работы была успешно решена задача оптимизации функции Axis parallel hyper-ellipsoid function с использованием генетических алгоритмов:

1. Изучен теоретический материал о real-coded генетических алгоритмах и различных операторах кроссинговера и мутации;
2. Создана программная библиотека на языке Python с реализацией арифметического и геометрического кроссоверов, случайной мутации и селекции методом рулетки;
3. Проведено исследование влияния параметров ГА на эффективность поиска для популяций размером 10, 25, 50 и 100 особей;

Список литературы

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