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 №5
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. Реализовать программу на языке Python, использующую эволюционную стратегию для поиска минимума функции axis parallel hyper-ellipsoid;
2. Для $n = 2$ построить визуализацию поверхности и траектории поиска: отображать найденный экстремум и расположение популяции на каждом шаге, обеспечить пошаговый режим;
3. Исследовать влияние основных параметров ЭС (размер популяции, стратегия мутации, вероятность рекомбинации) на скорость сходимости, число поколений и точность результата;
4. Повторить вычислительный эксперимент для $n = 3$ и сопоставить затраты времени и качество найденного решения.

2 Теоретические сведения

2.1 Общие сведения

Эволюционные стратегии (ЭС), также как и генетические алгоритмы, основаны на эволюции популяции потенциальных решений, но, в отличие от них, здесь используются генетические операторы на уровне фенотипа, а не генотипа. Разница в том, что ГА работают в пространстве генотипа — кодов решений, в то время как ЭС производят поиск в пространстве фенотипа — векторном пространстве вещественных чисел.

В ЭС учитываются свойства хромосомы «в целом», в отличие от ГА, где при поиске решений исследуются отдельные гены. В природе один ген может одновременно влиять на несколько свойств организма. С другой стороны, одно свойство особи может определяться несколькими генами. Естественная эволюция основана на исследовании совокупности генов, а не отдельного (изолированного) гена.

В эволюционных стратегиях целью является движение особей популяции по направлению к лучшей области ландшафта фитнесс-функции. ЭС изначально разработаны для решения многомерных оптимизационных задач, где пространство поиска — многомерное пространство вещественных чисел.

Ранние эволюционные стратегии основывались на популяции, состоящей из одной особи, и в них использовался только один генетический оператор — мутация. Здесь для представления особи (потенциального решения) была использована идея, которая заключается в следующем.

Особь представляется парой действительных векторов:

$$v = (\mathbf{x}, \boldsymbol{\sigma}),$$

где $\mathbf{x}$ — точка в пространстве решений и $\boldsymbol{\sigma}$ — вектор стандартных отклонений (вариабельность) от решения. В общем случае особь популяции определяется вектором потенциального решения и вектором «стратегических параметров» эволюции. Обычно это вектор стандартных отклонений (дисперсия), хотя допускаются и другие статистики.

Единственным генетическим оператором в классической ЭС является оператор мутации, который выполняется путём сложения координат вектора-родителя со случайными числами, подчиняющимися закону нормального распределения, следующим образом:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} + \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}),$$

где $\mathcal{N}(\mathbf{0}, \boldsymbol{\sigma})$ — вектор независимых случайных чисел, генерируемых согласно распределению Гаусса с нулевым средним значением и стандартным отклонением $\boldsymbol{\sigma}$. Как видно из приведённой формулы, величина мутации управляется нетрадиционным способом. Иногда эволюционный процесс используется для изменения и самих стратегических параметров $\boldsymbol{\sigma}$, в этом случае величина мутации эволюционирует вместе с искомым потенциальным решением.

Интуитивно ясно, что увеличение отклонения подобно увеличению шага поиска на поверхности ландшафта. Высокая вариабельность способствует расширению пространства поиска и эффективна при нахождении потенциальных зон

(суб)оптимальных решений и соответствует высоким значениям коэффициента мутации. В то же время малые значения вариабельности позволяют сфокусироваться на поиске решения в перспективной области. Стратегические параметры стохастически определяют величину шага поиска: большая вариабельность ведёт к большим шагам.

2.2 Двукратная эволюционная (1+1)-стратегия

Здесь потомок принимается в качестве нового члена популяции (он заменяет своего родителя), если значение фитнесс-функции (целевой функции) на нём лучше, чем у его родителя и выполняются все ограничения. Иначе (если значение фитнесс-функции на нём хуже, чем у родителя), потомок уничтожается и популяция остаётся неизменной.

Алгоритм процесса эволюции двукратной (1+1)-эволюционной стратегии можно сформулировать следующим образом:

1. Выбрать множество параметров $\mathbf{X}$, необходимых для представления решения данной проблемы, и определить диапазон допустимых изменений каждого параметра: $\{x_1^{min}, x_1^{max}\}, \{x_2^{min}, x_2^{max}\}, \ldots, \{x_P^{min}, x_P^{max}\}$. Установить номер поколения $t = 0$; задать стандартное отклонение σ_i для каждого параметра, функцию f, для которой необходимо найти оптимум, и максимальное число поколений k.

2. Для каждого параметра случайным образом выбрать начальное значение из допустимого диапазона: множество этих значений составляет начальную популяцию (из одной особи) $\mathbf{X}^{(t)} = (x_1, x_2, \ldots, x_P)$.

3. Вычислить значение оптимизируемой функции f для родительской особи $F_p = f(\mathbf{X}^{(t)})$.

4. Создать новую особь-потомка: $\mathbf{X}^* = \mathbf{X}^{(t)} + \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma})$.

5. Вычислить значение f для особи-потомка $F_o = f(\mathbf{X}^*)$.

6. Сравнить значения функций f для родителя и потомка; если значение потомка F_o лучше, чем у родительской особи, то заменить родителя на потомка $\mathbf{X}^{(t)} = \mathbf{X}^*$, иначе оставить в популяции родителя.

7. Увеличить номер поколения $t = t + 1$.

8. Если не достигнуто максимальное число поколений $t < k$, то переход на шаг 4, иначе выдать найденное решение $\mathbf{X}^{(t)}$.

Несмотря на то, что фактически здесь популяция состоит из одной особи, рассмотренная стратегия называется двукратной ЭС. Причина в том, что здесь фактически происходит конкуренция потомка и родителя.

2.3 Правило успеха 1/5

Обычно вектор стандартных отклонений $\boldsymbol{\sigma}$ остаётся неизменным в течение всего процесса эволюции. Чтобы оптимизировать скорость сходимости этого процесса, И. Решенберг (основоположник ЭС) предложил правило успеха «1/5».

Смысл его заключается в следующем — правило применяется после каждых k поколений процесса (где k — параметр этого метода):

$$\sigma_i^{(t+1)} = \begin{cases} c_i \cdot \sigma_i^{(t)}, & \text{если } \varphi(k) > 1/5, \\ \sigma_i^{(t)}, & \text{если } \varphi(k) = 1/5, \\ c_d \cdot \sigma_i^{(t)}, & \text{если } \varphi(k) < 1/5, \end{cases}$$

где $\varphi(k)$ — отношение числа успешных мутаций к общему числу произведённых мутаций k (число успехов, делённое на k), которое называется коэффициентом успеха для оператора мутации в течение k последних поколений; величина $c_i > 1$, $c_d < 1$ — регулирует увеличение/уменьшение отклонения мутации.

Обычно на практике оптимальные значения полагают равными следующим величинам: $c_d = 0.82$; $c_i = 1/0.82 = 1.22$. Смысл этого правила в следующем:

- если коэффициент успеха $\varphi(k) > 1/5$, то отклонение $\sigma^{(t+1)}$ увеличивается (мы идём более крупными шагами);
- если коэффициент успеха $\varphi(k) < 1/5$, то отклонение $\sigma^{(t+1)}$ уменьшается (шаг поиска уменьшается).

Таким образом, алгоритм автоматически подстраивает шаг поиска под текущий рельеф функции.

2.4 Многократная эволюционная стратегия

По сравнению с двукратной многократная эволюция отличается не только размером популяции ($N > 2$), но и имеет некоторые дополнительные отличия:

- все особи в поколении имеют одинаковую вероятность выбора для мутации;
- имеется возможность введения оператора рекомбинации, где два случайно выбранных родителя производят потомка по следующей схеме:

  $$x_i^{\text{потомок}} = x_i^{q_i}, \quad i = 1, \ldots, n,$$

  где $q_i = 1$ или $q_i = 2$ (т.е. каждая компонента потомка копируется из первого или второго родителя).

В современной литературе используются следующие обозначения:

- $(1 + 1)$-ЭС — двукратная стратегия (1 родитель производит 1 потомка);
- $(\mu + 1)$-ЭС — многократная стратегия (μ родителей производят 1 потомка);

- $(\mu+\lambda)$-ЭС — μ родителей производят λ потомков и отбор μ лучших представителей производится среди объединённого множества ($\mu + \lambda$ особей) родителей и потомков;
- (μ, λ)-ЭС — μ особей родителей порождает λ потомков, причём $\lambda > \mu$ и процесс выбора μ лучших производится только на множестве потомков.

Следует подчеркнуть, что в обоих последних видах ЭС обычно число потомков существенно больше числа родителей $\lambda > \mu$ (иногда полагают $\lambda/\mu = 7$).

Многочисленные исследования доказывают, что ЭС не менее эффективно, а часто гораздо лучше справляются с задачами оптимизации в многомерных пространствах, при этом более просты в реализации из-за отсутствия процедур кодирования и декодирования хромосом.

3 Особенности реализации

3.1 Структура модулей

- **Модуль `functions.py`**: содержит реализацию тестовой функции axis parallel hyper-ellipsoid и вспомогательные генераторы диапазонов.

- **Модуль `es.py`**: ядро эволюционной стратегии. Определены структуры конфигурации, представление особей и популяции, операторы рекомбинации и мутации.

- **Модуль `experiments.py`**: сценарии серийных экспериментов с переборами параметров и сохранением метрик.

- **Модуль `main.py`**: точка входа для интерактивных запусков с визуализацией.

3.2 Модуль functions.py

Модуль содержит реализацию тестовой функции axis parallel hyper-ellipsoid:

```
1  def axis_parallel_hyperellipsoid(x: Array) -> float:
2      """Axis-parallel hyper-ellipsoid benchmark function.
3
4      Parameters:
5          x: Point in R^n
6
7      Returns:
8          The value of the hyper-ellipsoid function
9      """
10     indices = np.arange(1, x.shape[0] + 1, dtype=np.float64)
11     return float(np.sum(indices * np.square(x)))
```

Функция принимает вектор NumPy произвольной размерности и возвращает скалярное значение фитнеса. Для двумерного случая формула принимает вид $f(x_1, x_2) = x_1^2 + 2x_2^2$, для трёхмерного $f(x_1, x_2, x_3) = x_1^2 + 2x_2^2 + 3x_3^2$.

Также определена вспомогательная функция для генерации симметричных границ:

```
1  def default_bounds(dimension: int,
2                     lower: float = -5.12,
3                     upper: float = 5.12) -> tuple[Array, Array]:
4      """Construct symmetric bounds for each dimension."""
5      x_min = np.full(dimension, lower, dtype=np.float64)
6      x_max = np.full(dimension, upper, dtype=np.float64)
7      return x_min, x_max
```

3.3 Модуль es.py

3.3.1 Структуры данных

Особь представлена классом `Individual`, содержащим координаты решения, стратегические параметры и фитнес:

```
@dataclass
class Individual:
    """Single individual of the evolution strategy population."""
    x: Array        # Coordinates in solution space
    sigma: Array    # Standard deviations for mutation
    fitness: float  # Fitness value

    def copy(self) -> "Individual":
        return Individual(self.x.copy(),
                          self.sigma.copy(),
                          float(self.fitness))
```

Конфигурация эволюционной стратегии задаётся через `EvolutionStrategyConfig`:

```
@dataclass
class EvolutionStrategyConfig:
    fitness_func: FitnessFn
    dimension: int
    x_min: Array
    x_max: Array
    mu: int                         # Number of parents
    lambda_: int                    # Number of offspring
    mutation_probability: float
    initial_sigma: Array | float
    max_generations: int
    selection: Literal["plus", "comma"] = "comma"
    recombination: Literal["intermediate", "discrete",
                           "none"] = "intermediate"
    success_rule_window: int = 10
    success_rule_target: float = 0.2
    sigma_increase: float = 1.22
    sigma_decrease: float = 0.82
    # ... other parameters
```

3.3.2 Рекомбинация

Функция `recombine` реализует выбор родителей и создание базового вектора для потомка:

```
def recombine(parents: Sequence[Individual],
              config: EvolutionStrategyConfig) -> tuple[Array, Array, float]:
    """Recombine parent individuals before mutation.

    Returns:
        Base vector, sigma and the best parent fitness
    """
    if config.recombination == "none":
        parent = random.choice(parents)
```

```
        return parent.x.copy(), parent.sigma.copy(), parent.fitness

    selected = random.choices(parents,
                              k=config.parents_per_offspring)

    if config.recombination == "intermediate":
        x = np.mean([p.x for p in selected], axis=0)
        sigma = np.mean([p.sigma for p in selected], axis=0)
    elif config.recombination == "discrete":
        mask = np.random.randint(0, len(selected),
                                 size=config.dimension)
        x = np.array([selected[mask[i]].x[i]
                      for i in range(config.dimension)])
        sigma = np.array([selected[mask[i]].sigma[i]
                          for i in range(config.dimension)])

    parent_fitness = min(p.fitness for p in selected)
    return x, sigma, parent_fitness
```

Промежуточная рекомбинация усредняет координаты родителей, дискретная копирует каждую координату из случайно выбранного родителя.

3.3.3 Мутация

Оператор мутации использует логнормальное распределение для адаптации стратегических параметров:

```
def mutate(x: Array, sigma: Array,
           config: EvolutionStrategyConfig,
           sigma_scale: float) -> tuple[Array, Array]:
    """Apply log-normal mutation with optional
       per-coordinate masking."""
    global_noise = np.random.normal()
    coordinate_noise = np.random.normal(size=config.dimension)

    # Adapt sigma using log-normal distribution
    sigma_new = sigma * np.exp(config.tau_prime * global_noise +
                               config.tau * coordinate_noise)
    sigma_new = np.clip(sigma_new * sigma_scale,
                        config.sigma_min, config.sigma_max)

    # Apply mutation steps
    steps = np.random.normal(size=config.dimension) * sigma_new

    # Optional per-coordinate mutation probability
    if config.mutation_probability < 1.0:
        mask = np.random.random(config.dimension) < \
               config.mutation_probability
        if not np.any(mask):
            mask[np.random.randint(0, config.dimension)] = True
        steps = steps * mask
        sigma_new = np.where(mask, sigma_new, sigma)

    x_new = np.clip(x + steps, config.x_min, config.x_max)
    return x_new, sigma_new
```

Параметры τ и τ' вычисляются как $\tau = 1/\sqrt{2\sqrt{n}}$ и $\tau' = 1/\sqrt{2n}$, где n — размерность задачи.

3.3.4 Создание потомков

Функция `create_offspring` генерирует λ потомков и отслеживает успешные мутации:

```
def create_offspring(parents: Sequence[Individual],
                     config: EvolutionStrategyConfig,
                     sigma_scale: float) -> tuple[list[Individual],
                                                  list[bool]]:
    """Create offspring and track successful mutations."""
    offspring: list[Individual] = []
    successes: list[bool] = []

    for _ in range(config.lambda_):
        base_x, base_sigma, best_parent_fitness = \
            recombine(parents, config)
        mutated_x, mutated_sigma = \
            mutate(base_x, base_sigma, config, sigma_scale)
        fitness = float(config.fitness_func(mutated_x))
        child = Individual(mutated_x, mutated_sigma, fitness)
        offspring.append(child)
        successes.append(fitness < best_parent_fitness)

    return offspring, successes
```

3.3.5 Селекция

Отбор следующего поколения производится согласно выбранной стратегии:

```
def select_next_generation(parents: list[Individual],
                           offspring: list[Individual],
                           config: EvolutionStrategyConfig) -> list[Individual]:
    """Select next generation according to the strategy."""
    if config.selection == "plus":
        pool = parents + offspring  # (mu + lambda)-strategy
    else:
        pool = offspring            # (mu, lambda)-strategy

    pool.sort(key=lambda ind: ind.fitness)
    next_generation = [ind.copy() for ind in pool[:config.mu]]
    return next_generation
```

3.4 Главная функция алгоритма

Функция `run_evolution_strategy` реализует основной цикл эволюционной стратегии с адаптацией по правилу успеха 1/5:

```
def run_evolution_strategy(config: EvolutionStrategyConfig) -> EvolutionStrategyResult:
    """Main evolution strategy loop with 1/5 success rule."""
```

```
    # Initialize random seed
    if config.seed is not None:
        random.seed(config.seed)
        np.random.seed(config.seed)

    # Initialize population
    parents = [Individual(
        np.random.uniform(config.x_min, config.x_max),
        config.make_initial_sigma(),
        0.0
    ) for _ in range(config.mu)]
    evaluate_population(parents, config.fitness_func)

    sigma_scale = 1.0
    success_window: deque[float] = deque()

    for generation_number in range(1, config.max_generations + 1):
        # Create offspring and track successes
        offspring, successes = create_offspring(parents, config,
                                                sigma_scale)
        success_ratio = sum(successes) / len(successes)
        success_window.append(success_ratio)

        # Apply 1/5 success rule
        if len(success_window) == config.success_rule_window:
            average_success = sum(success_window) / \
                              len(success_window)
            if average_success > config.success_rule_target:
                sigma_scale = min(sigma_scale * config.sigma_increase,
                                  config.sigma_scale_max)
            elif average_success < config.success_rule_target:
                sigma_scale = max(sigma_scale * config.sigma_decrease,
                                  config.sigma_scale_min)
            success_window.clear()

        # Select next generation
        parents = select_next_generation(parents, offspring, config)

        # Check stopping criteria
        # ...

    return EvolutionStrategyResult(...)
```

Правило успеха 1/5 применяется каждые k поколений (по умолчанию $k = 5$): если доля успешных мутаций выше 1/5, масштаб σ увеличивается в 1.22 раза, если ниже — уменьшается в 0.82 раза.

4 Результаты работы

Для демонстрации работы алгоритма была выполнена визуализация процесса оптимизации двумерной функции ($n = 2$) со следующими параметрами:

- $\mu = 20$ – размер популяции родителей.
- $\lambda = 80$ – число потомков ($\lambda = 4\mu$).
- $p_{mut} = 0.7$ – вероятность мутации каждой координаты.
- Промежуточная рекомбинация двух родителей.
- (μ, λ)-селекция: родители полностью заменяются.
- Адаптивное масштабирование шага мутации по правилу успеха 1/5.
- Начальное стандартное отклонение $\sigma_0 = 0.15 \cdot (x_{max} - x_{min})$.

Визуализация воспроизводит поверхность целевой функции и положение популяции на каждом шаге. Пошаговый режим позволяет наблюдать влияние изменения дисперсий: при успешных мутациях облако точек расширяется, при неудачах сжимается вокруг текущего минимума. Популяция постепенно консолидируется вокруг глобального минимума в точке $(0, 0)$.

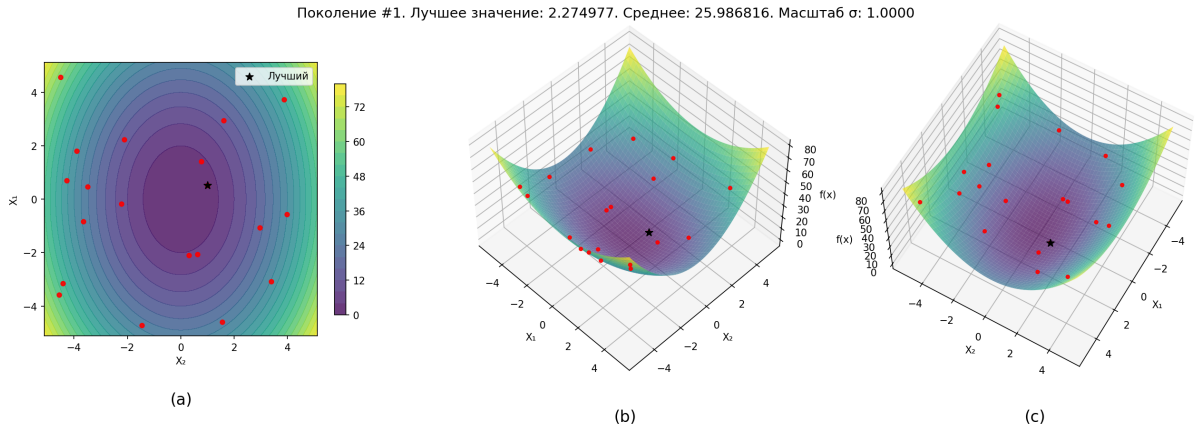

Рис. 1. Поколение 1: начальная популяция и рельеф функции

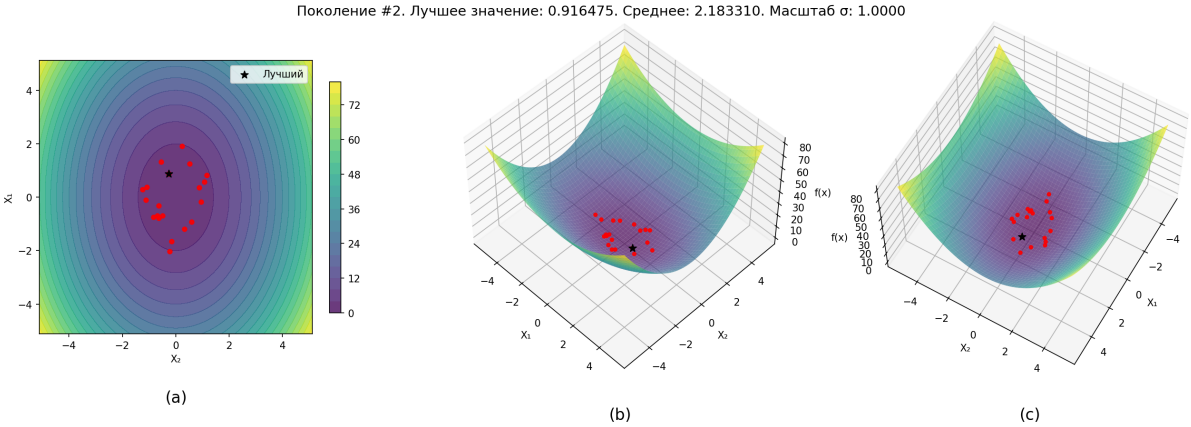

Рис. 2. Поколение 2: адаптация стратегических параметров

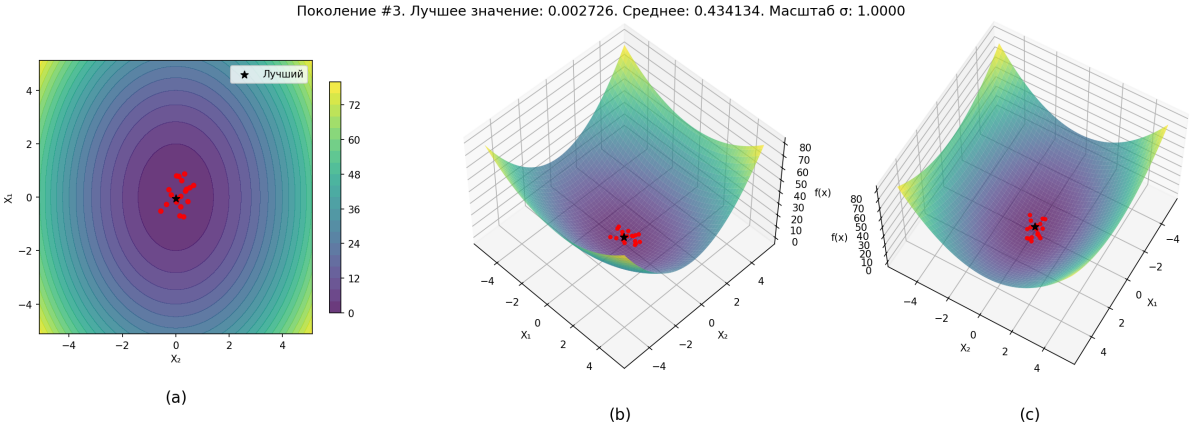

Рис. 3. Поколение 3: фокусировка поиска около минимума

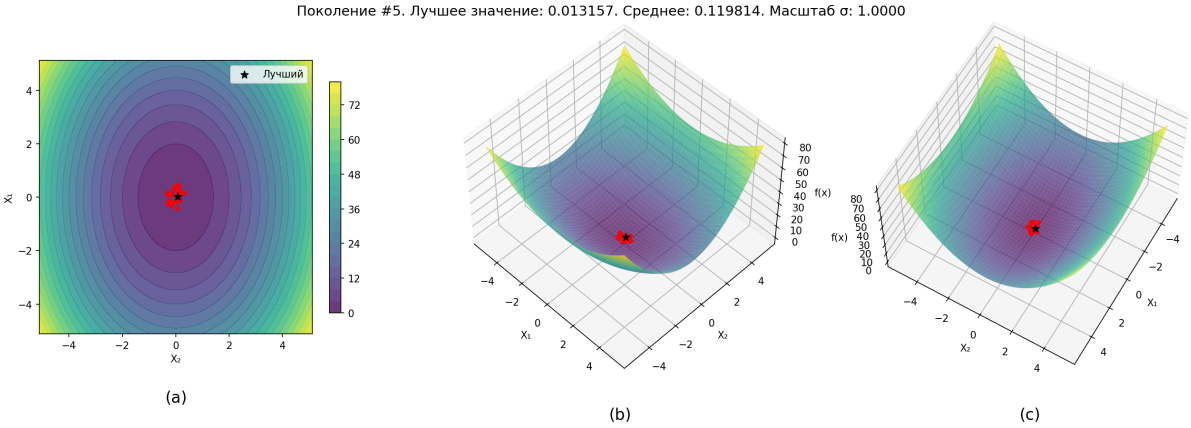

Рис. 4. Поколение 5: сжатие облака решений

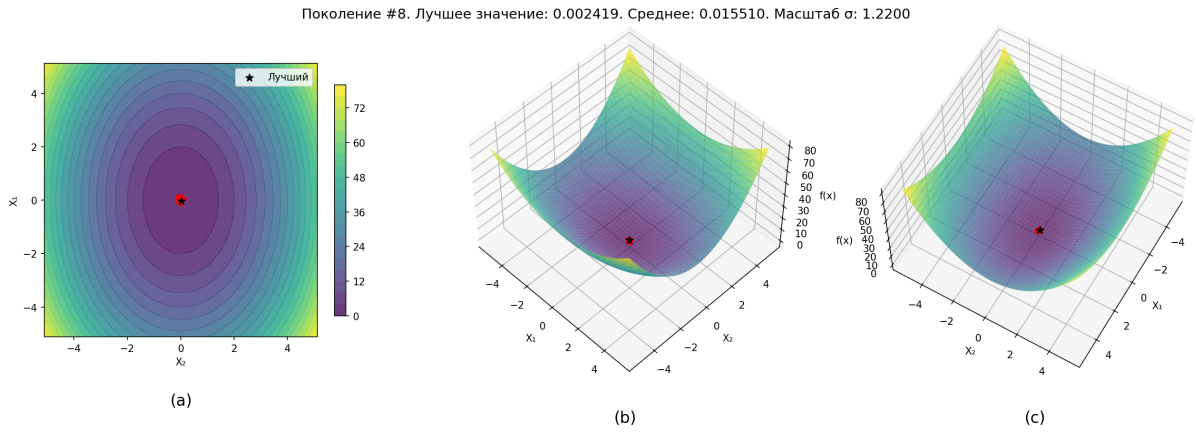

Рис. 5. Поколение 8: стабилизация шага мутации

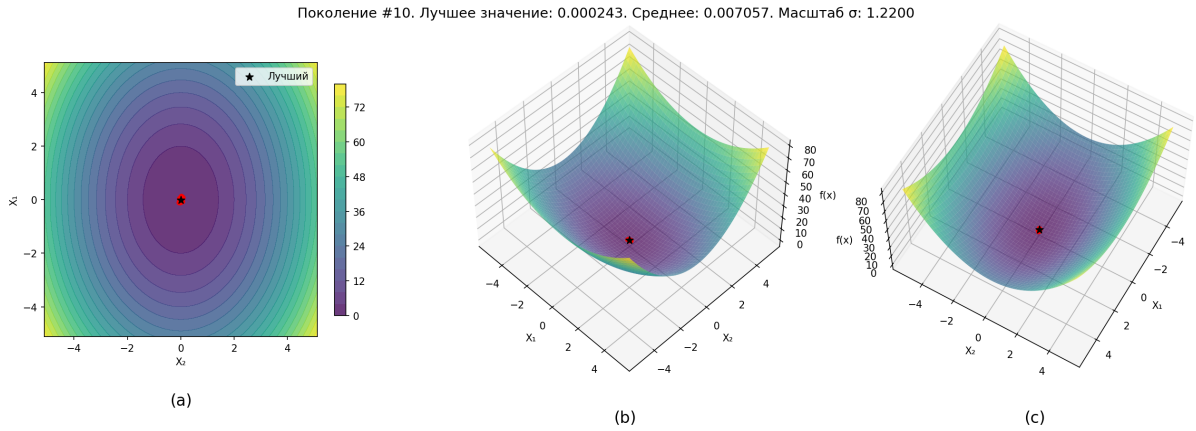

Рис. 6. Поколение 10: движение вдоль долины уровня

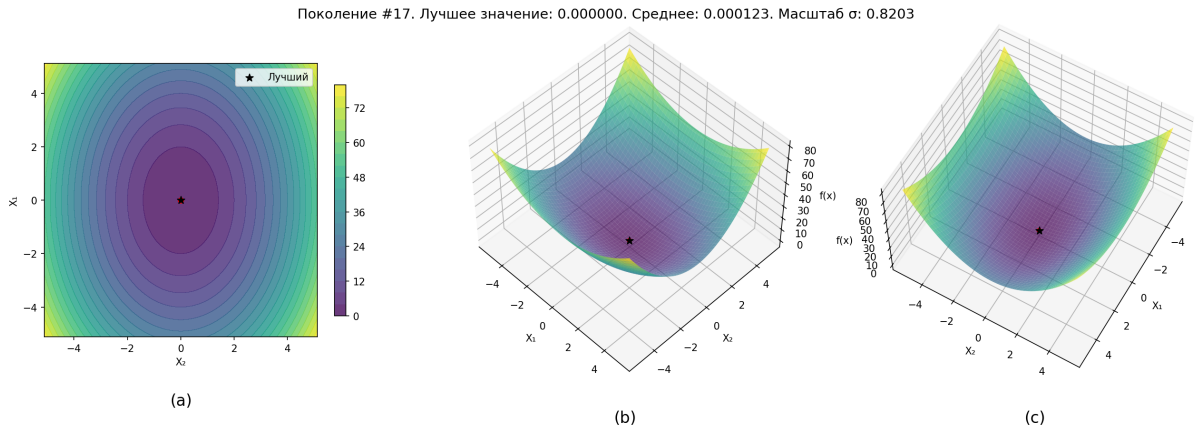

Рис. 7. Поколение 17: окончательная популяция

5 Исследование параметров

В рамках лабораторной работы было проведено исследование влияния размера популяции μ и вероятности мутации p_{mut} на эффективность алгоритма. Для экспериментов использовалась (μ, λ)-стратегия с $\lambda = 5\mu$, промежуточной рекомбинацией и адаптивным масштабированием шага мутации по правилу успеха 1/5.

5.1 Проведение измерений

Для исследования были выбраны следующие значения параметров:

- $\mu = 5, 10, 20, 40$ – размер популяции родителей.
- $p_{mut} = 0.3, 0.5, 0.7, 0.9, 1.0$ – вероятность мутации каждой координаты.
- Количество независимых запусков для усреднения результатов: 5.
- Критерий остановки: достижение порога $f(\mathbf{x}) < 10^{-6}$ или исчерпание лимита 300 поколений.

Результаты измерений представлены в таблицах 1 и 2. В ячейках указано среднее время выполнения в миллисекундах и среднее число поколений до достижения критерия остановки. Лучшие результаты по времени выполнения и по числу поколений выделены жирным цветом.

Таблица 1. Результаты для $n = 2$. Формат: время в мс (число поколений)

$\mu \setminus \mathbf{p_{mut}}$	**0.30**	**0.50**	**0.70**	**0.90**	**1.00**
5	60.6 (37)	35.1 (23)	37.9 (25)	29.2 (20)	**20.4** (17)
10	69.5 (22)	84.1 (28)	61.1 (21)	48.2 (17)	38.1 (16)
20	109.6 (18)	120.4 (20)	107.0 (18)	100.2 (17)	69.4 (15)
40	239.8 (19)	225.9 (19)	199.9 (17)	180.6 (16)	121.4 (**13**)

Таблица 2. Результаты для $n = 3$. Формат: время в мс (число поколений)

$\mu \setminus \mathbf{p_{mut}}$	**0.30**	**0.50**	**0.70**	**0.90**	**1.00**
5	146.0 (88)	212.2 (126)	93.7 (60)	44.8 (29)	**30.3** (25)
10	155.9 (49)	149.3 (48)	88.7 (30)	69.8 (24)	55.7 (23)
20	235.5 (38)	199.0 (32)	157.7 (26)	125.8 (21)	105.9 (21)
40	670.3 (53)	374.2 (31)	311.8 (26)	258.2 (22)	194.0 (**20**)

5.2 Анализ результатов

Анализ экспериментальных данных выявляет следующие закономерности:

- **Влияние вероятности мутации:** Увеличение p_{mut} от 0.3 до 1.0 последовательно улучшает результаты как по времени, так и по числу поколений. Это объясняется тем, что более частая мутация всех координат ускоряет исследование пространства и адаптацию популяции. Лучшие результаты достигаются при $p_{mut} = 1.0$ (мутация всех координат на каждом шаге).

- **Влияние размера популяции:** При малых μ (5-10) алгоритм демонстрирует наименьшее время выполнения и умеренное число поколений. С ростом μ до 40 время увеличивается пропорционально размеру популяции, но число поколений снижается благодаря более широкому охвату пространства поиска. Для двумерной задачи оптимальным является $\mu = 5$, $p_{mut} = 1.0$ (20.4 мс, 17 поколений).

- **Масштабирование на размерность:** При переходе от $n = 2$ к $n = 3$ время выполнения изменяется незначительно (30.3 мс против 20.4 мс для лучшей конфигурации), однако требуется больше поколений (25 против 17). Это связано с усложнением ландшафта целевой функции и необходимостью большего числа итераций для достижения порога 10^{-6}.

- **Эффективность адаптации:** Правило успеха 1/5 обеспечивает автоматическую подстройку масштаба мутации, что позволяет алгоритму быстро сходиться без ручной настройки начального σ. Минимальное число поколений (13 и 20 для $n = 2$ и $n = 3$ соответственно) достигается при больших популяциях ($\mu = 40$) и высокой вероятности мутации ($p_{mut} = 1.0$).

6 Ответ на контрольный вопрос

Вопрос: Что такое направленная мутация?

Ответ: Направленная мутация — это тип мутации, при котором изменения вносятся не случайным образом, а с учётом информации о ландшафте фитнес-функции или направлении улучшения решения. В отличие от обычной (ненаправленной) мутации, которая добавляет случайный шум к параметрам, направленная мутация использует информацию о градиенте функции приспособленности, историю успешных мутаций или другие эвристики, чтобы изменять особь в направлении, с большей вероятностью ведущем к улучшению. Это позволяет ускорить сходимость алгоритма, особенно вблизи оптимума, комбинируя преимущества эволюционного поиска и методов локальной оптимизации.

Заключение

В ходе пятой лабораторной работы реализована программа оптимизации многомерных функций методом эволюционных стратегий. Получены следующие результаты:

1. Изучены теоретические основы $(1+1)$ и популяционных ЭС, включая самонастраивающуюся мутацию и правило успеха 1/5;
2. Разработана модульная Python-реализация с поддержкой визуализации поиска и гибкой конфигурацией стратегических параметров;
3. Проведены вычислительные эксперименты для измерения влияния размера популяции, интенсивности мутации и схемы адаптации на скорость сходимости при $n = 2$ и $n = 3$;
4. Подготовлена инфраструктура для дальнейшего расширения: сохранение историй поколений, экспорт результатов и интерактивный просмотр шагов оптимизации.

Список литературы

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