МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт Информационных Технологий

Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА №6

по дисциплине
«Проектирование интеллектуальных систем (часть 2/2)»

Студент группы:ИКБО-04-22 __Кликушин В.И._
(Ф. И.О. студента)

Преподаватель __Холмогоров В.В.__
(Ф.И.О. преподавателя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 ПОСТАНОВКА ЗАДАЧИ 5
2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 6
2.1 Кодирование информации в генетических алгоритмах 6
2.2 Классический генетический алгоритм 8
2.3 Операторы генетического алгоритма 10
2.3.1 Селекция 11
2.3.2 Скрещивание 13
2.3.3 Мутация 14
3 ПРАКТИЧЕСКАЯ ЧАСТЬ 15
3.1 Постановка задачи 15
3.2 Реализация генетического алгоритма 16
ЗАКЛЮЧЕНИЕ 20
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 21
ПРИЛОЖЕНИЯ 22

ВВЕДЕНИЕ

Существуют классы оптимизационных задач, решение которых удается находить с помощью достаточно эффективных методов, вполне приемлемых по трудоемкости. Вместе с тем имеются и такие классы оптимизационных задач, решение которых невозможно найти без полного перебора вариантов. В частности, к числу последних относятся многие разновидности задач многокритериальной оптимизации. Известно, что при большой размерности этих задач реализация перебора вариантов практически невозможна из-за чрезвычайно больших временных затрат. Усложнение многомерной целевой функции приводит экспоненциальному росту зависимости размерности задачи от времени вычислений.

Таким образом, оптимизационные задачи могут решаться классическими методами до определенного придела. В этой ситуации альтернативным походом является применение методов, базирующихся на методологии эволюционных вычислений. Этот термин обычно используется для общего описания алгоритмов поиска, оптимизации или обучения, основанных на некоторых формализованных принципах естественного эволюционного отбора.

Высокая согласованность и эффективность работы элементов биологических систем приводила целый ряд исследователей к естественной мысли о возможности использования принципов биологической эволюции для оптимизации важных для приложений систем, природа которых отлична от биологической. Так, в 1975 году была опубликована основополагающая работа Дж. Холланда «Адаптация в естественных и искусственных системах», в которой был предложен генетический алгоритм. В этой работе Холланд ввел формализованный подход для предсказывания качества следующего поколения, известный как теорема схем. Эволюционные вычисления представляет собой одно из направлений искусственного интеллекта, объединяющее компьютерные методы моделирования эволюционных процессов в естественных и

искусственных системах, такие как генетические алгоритмы, эволюционные стратегии, эволюционное программирование и другие эвристические методы.

Следует отметить, что эволюционная стратегия схожа с генетическим алгоритмом, но существует несколько различий. При поиске решения в эволюционной стратегии вначале происходит мутация и скрещивание особей для получения потомков, затем происходит детерминированный отбор без повторений лучших особей из общего поколения родителей и потомков. По мере развития эволюционных стратегий и генетических алгоритмов в течение последних лет существенные различия между ними постепенно уменьшаются.

1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки реализации генетических алгоритмов.

Задачи: создать программную реализацию генетического алгоритма, который способен решать задачу выбранного варианта (для этого она должна быть описана математически), либо находить решение одной из существующих задач или проблем математики, алгоритм должен иметь следующий минимальный стек:

– оптимизированные для решаемой задачи входные данные, образующие начальную популяцию;

– тренировочную и тестовые выборки (может и не быть, нужны для самостоятельного переобучения и отслеживания пользователем степени обучения);

– loss-функцию и метрику точности (может и не быть, если ожидаемые выходные данные неизвестны);

– фитнес-функцию и алгоритм отбора;

– случайное скрещивание и случайную мутацию.

В качестве дополнительных элементов генетического алгоритма реализовать:

– стратегии элитизма, рулетки и/или турнира (наиболее простые из существующих алгоритмов отбора);

– графики истории и результатов обучения;

– сравнительный анализ собственной стратегии из пункта 1 и стратегий из пункта 2 (в том числе между собой);

– графики полученных и ожидаемых (если они есть) результатов.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Генетические алгоритмы – адаптивные методы поиска, которые в последнее время часто используются для решения задач функциональной оптимизации. Они основаны на генетических процессах биологических организмов: биологические популяции развиваются в течение нескольких поколений, подчиняясь законам естественного отбора и по принципу «выживает наиболее приспособленный», открытому Чарльзом Дарвином. Подражая этому процессу генетические алгоритмы способны «развивать» решения реальных задач, если те соответствующим образом закодированы.

Очень важным понятием в генетических алгоритмах является функция приспособленности (fitness-функция), иначе называемая функцией оценки. Она представляет меру приспособленности данной особи в популяции. Эта функция играет важнейшую роль, поскольку позволяет оценить степень приспособленности конкретных особей в популяции и выбрать из них наиболее приспособленные (т. е. имеющие наибольшие значения функции приспособленности) в соответствии с эволюционным принципом выживания «сильнейших» (лучше всего приспособившихся). Функция приспособленности также получила свое название непосредственно из генетики. Она оказывает сильное влияние на функционирование генетических алгоритмов и должна иметь точное и корректное определение. В задачах оптимизации функция приспособленности, как правило, оптимизируется и называется целевой функцией.

На каждой итерации генетического алгоритма приспособленность каждой особи данной популяции оценивается при помощи функции приспособленности, и на этой основе создается следующая популяция особей, составляющих множество потенциальных решений задачи.

2.1 Кодирование информации в генетических алгоритмах

Выбор способа кодирования является одним из важнейших этапов при использовании эволюционных алгоритмов. В частности, должно выполняться следующее условие: должна быть возможность закодировать (с допустимой погрешностью) в хромосоме любую точку из пространства поиска. Невыполнение этого условия может привести как к увеличению времени эволюционного поиска, так и к невозможности найти решение поставленной задачи. Как правило, в хромосоме кодируются численные параметры решения. Для этого возможно использование бинарного и вещественного кодирования.

Бинарное (двоичное, целочисленное) кодирование. В каноническом генетическом алгоритме хромосома представляет собой битовую строку, в которой закодированы параметры решения поставленной задачи. Каждая хромосома может кодировать только один параметр задачи. Но так как в одном генотипе может быть несколько хромосом, то один генотип может закодировать несколько параметров.

Вещественное кодирование (real-coded). При работе с оптимизационными задачами в непрерывных пространствах вполне естественно представлять гены напрямую вещественными числами. В этом случае хромосома есть вектор вещественных чисел. Точность будет определяться исключительно разрядной сеткой той ЭВМ, на которой реализуется алгоритм. Длина хромосомы будет совпадать с длиной вектора-решения оптимизационной задачи, иначе говоря, каждый ген будет отвечать за одну переменную. Генотип объекта становится идентичным его фенотипу.

Таким образом, часто бывает удобнее кодировать в гене не целое число, а вещественное. Это позволяет избавиться от операций кодирования/декодирования, используемых в целочисленном кодировании, а также увеличить точность найденного решения. Алгоритмы вещественного кодирования работают, в общем случае, с непрерывной областью допустимых значений переменных. Данный класс алгоритмов позволяет уменьшить объём вычислительных процедур на каждом шаге эволюции за счёт отсутствия

двоично-десятичных преобразований при расчёте значений функций приспособленности и уменьшения размеров хромосом.

2.2 Классический генетический алгоритм

Методологическая основа генетического алгоритма базируется на теории эволюционного процесса, в котором набор действий повторяется итеративно и продолжается несколько жизненных циклов (поколений), пока не будет выполнен критерий останова алгоритма (Рисунок 2.1).

Рисунок 2.1 - Итерационный процесс генетического алгоритма

Генетический алгоритм начинает работу с некоторого случайного набора исходных решений, который называется популяцией. Каждый элемент из популяции называется хромосомой и представляет некоторое решение проблемы в первом приближении. Хромосома представляет собой строку символов некоторой природы, не обязательно бинарных. Хромосомы эволюционируют на протяжении множества итераций, носящих название поколений (или генераций). В ходе каждой итерации хромосома оценивается с использованием некоторой меры соответствия (англ. fitness function), которую мы будем называть функцией соответствия или fitness-функция. Для создания следующего поколения новые хромосомы, называемые отпрысками, формируются либо путем скрещивания (англ. crossover) двух хромосом - родителей из текущей популяции, либо путем случайного изменения (мутации) одной хромосомы. Новая популяция формируется путем выбора согласно функции соответствия некоторых родителей и отпрысков и удаления оставшихся для того, чтобы сохранять

постоянным размер популяции. Хромосомы с большей функцией соответствия имеют больше шансов быть выбранными (выжить). После нескольких итераций алгоритм сходится к лучшей хромосоме, которая является либо оптимальным, либо близким к оптимальному решению.

Блок схема классического генетического алгоритма представлена на Рисунке 2.2.

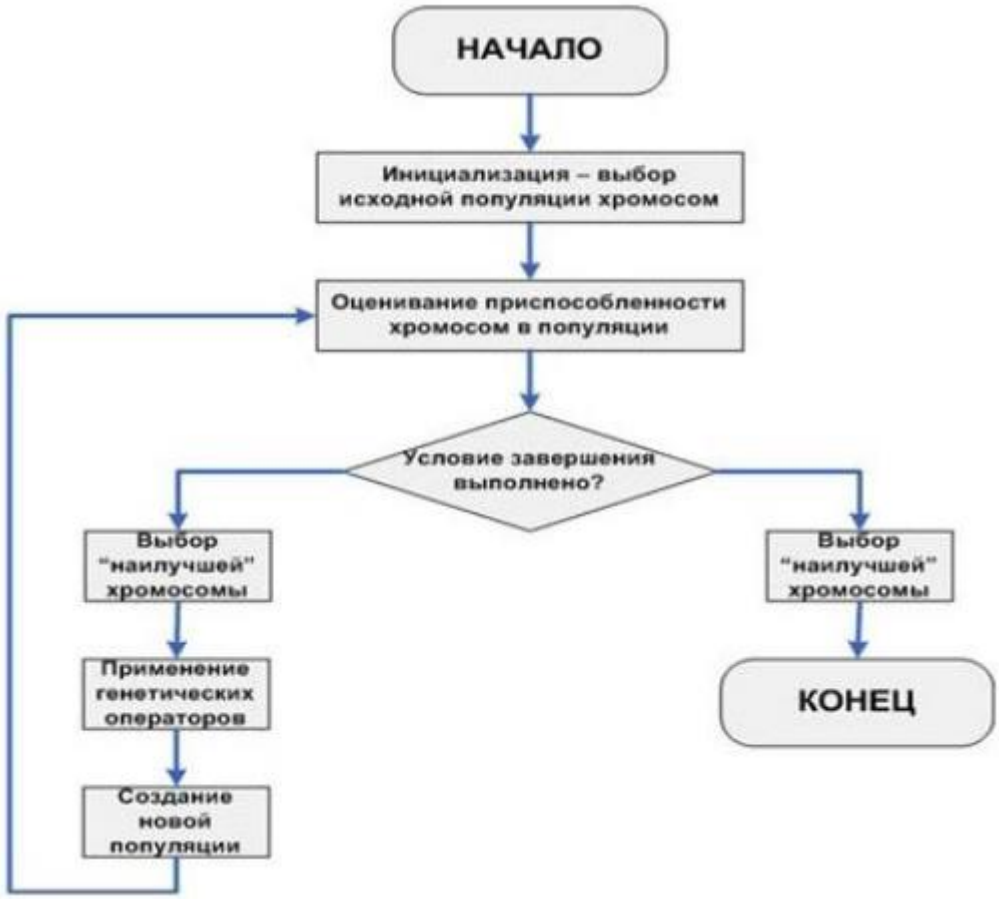

Рисунок 2.2 – Блок-схема генетического алгоритма

Инициализация начальной популяции, формирование исходной популяции, заключается в случайном выборе заданного количества хромосом (особей), представляемых двоичными последовательностями фиксированной длины.

Оценивание приспособленности хромосом в популяции состоит в расчете функции приспособленности для каждой хромосомы этой популяции. Чем больше значение этой функции, тем выше «качество» хромосомы.

Проверка условия остановки алгоритма. Процесс эволюции может продолжаться до бесконечности. Критерием останова может служить заданное количество поколений или схождение (convergence) популяции.

Схождением называется состояние популяции, когда все строки популяции находятся в области некоторого экстремума и почти одинаковы. То есть скрещивание практически никак не изменяет популяции, а мутирующие особи склонны вымирать, так как менее приспособлены. Таким образом, схождение популяции означает, что достигнуто решение близкое к оптимальному. Итоговым решением задачи может служить наиболее приспособленная особь последнего поколения. Также критерием останова может служить заданное количество поколений, исчерпание времени, отпущенного на эволюцию.

Если условие остановки выполнено, то производится переход к завершающему этапу выбора «наилучшей» хромосомы. В противном случае на следующем шаге выполняется операции: репродукции, скрещивания и мутации. Данные операции представлены определенными операторами.

Термины селекция и репродукция в данном контексте используются в качестве синонимов. При этом репродукция в данном случае связывается скорее с созданием копий хромосом родительского пула, тогда как более распространенное содержание этого понятия обозначает процесс формирования новых особей, происходящих от конкретных родителей. Если принимается такое толкование, то операторы скрещивания и мутации могут считаться операторами репродукции, а селекция - отбором особей (хромосом) для репродукции.

2.3 Операторы генетического алгоритма

В каждой генерации генетического алгоритма хромосомы являются результатом применения некоторых генетических операторов. Оператор – это языковая конструкция, представляющая один шаг из последовательности действий или набора описаний алгоритма.

Генетический алгоритм состоит из набора генетических операторов. Генетический оператор по аналогии с оператором алгоритма – средство отображения одного множества на другое. Другими словами, это конструкция, представляющая один шаг из последовательности действий генетического алгоритма. В классическом генетическом алгоритме применяются два основных генетических оператора: оператор скрещивания (crossover) и оператор мутации (mutation).

Однако следует отметить, что оператор мутации играет явно второстепенную роль по сравнению с оператором скрещивания. Это означает, что скрещивание в классическом генетическом алгоритме производится практически всегда, тогда как мутация – достаточно редко.

2.3.1 Селекция

Селекция (Репродукция) – это процесс, в котором хромосомы копируются в промежуточную популяцию (родители) для дальнейшего участия в создании потомков согласно их значениям функции соответствия. При этом хромосомы с лучшими значениями целевой функции имеют большую вероятность попадания одного или более потомков в следующее поколение.

Таким образом, оператор отбор S порождает промежуточную популяцию β_t из текущей популяции P_t путем отбора и генерации новых копий особей (Формула 2.1).

$$P_t \xrightarrow{S} \beta_t \tag{2.1}$$

Текущая популяция альтернативных решений P_t есть множество элементов (Формула 2.2).

$$P_t = \{P_1, P_2, \ldots, P_i, \ldots, P_{N_p}\}, \tag{2.1}$$

где t – номер генерации;

N_p – размер популяции.

Каждый элемент популяции P_i представляется в виде хромосомы (строки), которая есть возможное решение рассматриваемой оптимизационной задачи. Хромосома состоит из конечного числа генов g_n, представляя генотип объекта, т. е. совокупность его наследственных признаков (Формула 2.3).

$$P_i = \{g_1, g_2, \ldots, g_n\} \tag{2.3}$$

При отборе конкурентоспособных особей используют fitness-функцию, как единственно доступный источник информации о качестве решения. Но различные методы отбора родителей по-разному используют эту информацию.

Среди основных видов селекции:

– селекция на основе рулетки (Roulette) – самый простой и наиболее используемый метод отбора. При его реализации каждому элементу в популяции соответствует зона на колесе рулетки, пропорционально соразмерная с величиной целевой функции. Тогда при повороте колеса рулетки каждый элемент имеет некоторую вероятность выбора для селекции. Причем элемент с большим значением целевой функции имеет большую вероятность для выбора;

– турнирная селекция (Tournament) – случайно выбирается указанное число особей, и лучшие элементы в этой группе на основе заданного турнира определяются для дальнейшего генетического поиска;

– селекция на основе заданного распределения (Uniform) – родители выбираются случайным образом согласно заданному распределению и с учетом количества родительских особей и их вероятностей;

– селекция на основе заданного вероятностного распределения (Stochastic uniform) – строится линия, в которой каждому родителю ставится в соответствие её часть определенного размера (в зависимости от вероятности родителя), затем

алгоритм пробегает по линии шагами одинаковой длины и выбирает родителей в зависимости от того, на какую часть линии попал шаг.

Оператор репродукции считается эффективным, если он создает возможность перехода из одной подобласти альтернативных решений области поиска в другую. Это повышает вероятность нахождения глобального оптимума целевой функции.

Кроме описанных, существует большое число других методов селекции, которые можно условно классифицировать на три группы: вероятностные, детерминированные и различные комбинации этих методов.

Выбор случайных и сильно различающихся хромосом повышает генетическое разнообразие популяции, что повышает скорость сходимости генетического алгоритма на начальном этапе оптимизации и позволяет в некоторых случаях выходить из локальных оптимумов.

Основной трудностью решения инженерных оптимизационных задач с большим количеством локальных оптимумов является предварительная сходимость алгоритмов. Другими словами, попадание решения в один, далеко не лучший, локальный оптимум при наличии их большого количества. Различные методы селекции и их модификации как раз и позволяют в некоторых случаях решать проблему предварительной сходимости алгоритмов. Следует отметить, что исследователи генетических алгоритмов все более склоняются к мысли применять комбинированные методы селекции с использованием предварительных знаний о решаемых задачах и предварительных результатах.

2.3.2 Скрещивание

Полученная промежуточная популяция (родители) подвергается операции скрещивания, иногда называемой рекомбинацией или кроссинговера (crossover). Оператор скрещивания (кроссинговер) – это языковая конструкция, позволяющая на основе преобразования (скрещивания) хромосом родителей (или их частей) создавать хромосомы потомков. Существует достаточно

большое число операторов кроссинговера, их структура в основном и определяет эффективность алгоритма.

2.3.3 Мутация

Мутация – позволяет осуществить случайные изменения в позиции хромосомы (генов), тем самым обеспечивается генетическое разнообразие популяции, расширяется пространство поиска наилучших решений. Мутация необходима в ситуациях, когда популяция малочисленна и подвержена преждевременной сходимости (premature convergence). При этом оператор кроссинговера практически не изменяет популяции, т. к. все хромосомы почти одинаковы. В этом случае мутация способна инвертировать «застрявший» ген у одной из хромосом и вновь расширить пространство поиска.

Оператор мутации – языковая конструкция, позволяющая на основе преобразования родительской хромосомы (или ее части) создавать хромосому потомка.

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

3.1 Постановка задачи

Необходимо произвести оптимизацию функции, то есть найти ее глобальный минимум.

Нахождение глобального минимума функции от многих переменных состоит в поиске точки в многомерном пространстве, где значение функции будет минимальным. Сложность этой задачи состоит в том, что функция может содержать множество локальных минимумов, где производная функция равна нулю, но значение функции не является минимальным.

Выбранная функция для оптимизации: функция Гольдшейна-Прайса (Формула 3.1).

$$f(x,y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + \\ +(2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)] . \quad (3.1)$$

Глобальный минимум функции достигается в точке (0; -1) и равен 3. Функция рассматривается на области $-2 \leq x, y \leq 2$.

График оптимизируемой функции представлен на Рисунке 3.1.

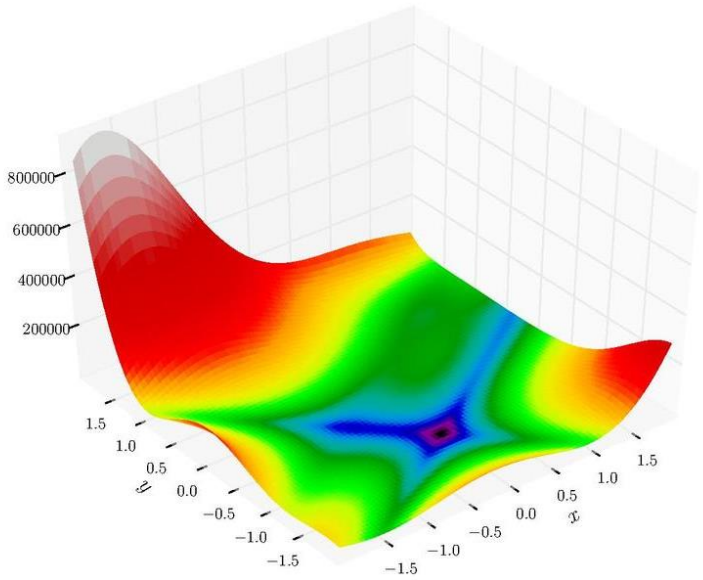

Рисунок 3.1 – График функции Гольдшейна-Прайса

3.2 Реализация генетического алгоритма

Реализация генетического алгоритма состоит из двух взаимодействующих классов: Individual (особь, код файла individual.py представлен в Приложении А) и GeneticAlgorithm (основной класс с написанной логикой алгоритма). Код файла genetic_algorithm.py представлен в Приложении Б.

Для кодирования решений используется вещественное представление, где каждая особь представлена двумя вещественными числами в заданном диапазоне, соответствующими координатам x и y в пространстве поиска.

Начальная популяция создается случайным образом: для каждой особи генерируются значения x и y в заданных границах с использованием равномерного распределения.

В качестве фитнес-функции используется сама функция Гольдштейна-Прайса. Поскольку задача состоит в поиске глобального минимума, меньшие значения функции соответствуют большей приспособленности особи.

Реализованы два метода селекции:

1. Метод рулетки – вероятность выбора особи пропорциональна её приспособленности. Для минимизации функции используется инвертирование значений фитнес-функции.
2. Турнирная селекция – случайно выбирается заданное количество особей, и из них выбирается особь с наилучшим значением фитнес-функции.

Реализованы два оператора скрещивания:

1. Одноточечное скрещивание – обмен координатами x или y между родителями.
2. Арифметическое скрещивание – линейная комбинация координат родителей с коэффициентом $\alpha \in [0, 1]$.

Используется равномерная мутация, при которой координаты особи с заданной вероятностью заменяются на случайные значения в пределах границ поиска.

Реализован механизм элитизма, который сохраняет заданное количество лучших особей в каждом поколении без изменений, гарантируя сохранение найденных хороших решений.

После завершения работы алгоритма строится график сходимости, отображающий изменение лучшего значения фитнес-функции по поколениям. Это позволяет оценить динамику обучения и эффективность алгоритма.

Для отработки алгоритма выбраны гиперпараметры, представленные в Листинге 3.1.

Листинг 3.1 – Гиперпараметры генетического алгоритма

```
bounds = [(-2, 2), (-2, 2)]
ga = GeneticAlgorithm(
    fitness_function=goldstein_price,
    bounds=bounds,
    population_size=50,
    max_generations=100,
    crossover_rate=0.8,
    mutation_rate=0.1,
    selection_method="roulette",
    tournament_size=3,
    elitism_count=2,
    crossover_method="arithmetic",
    mutation_method="uniform",
)
```

Построенный график сходимости представлен на Рисунке 3.1.

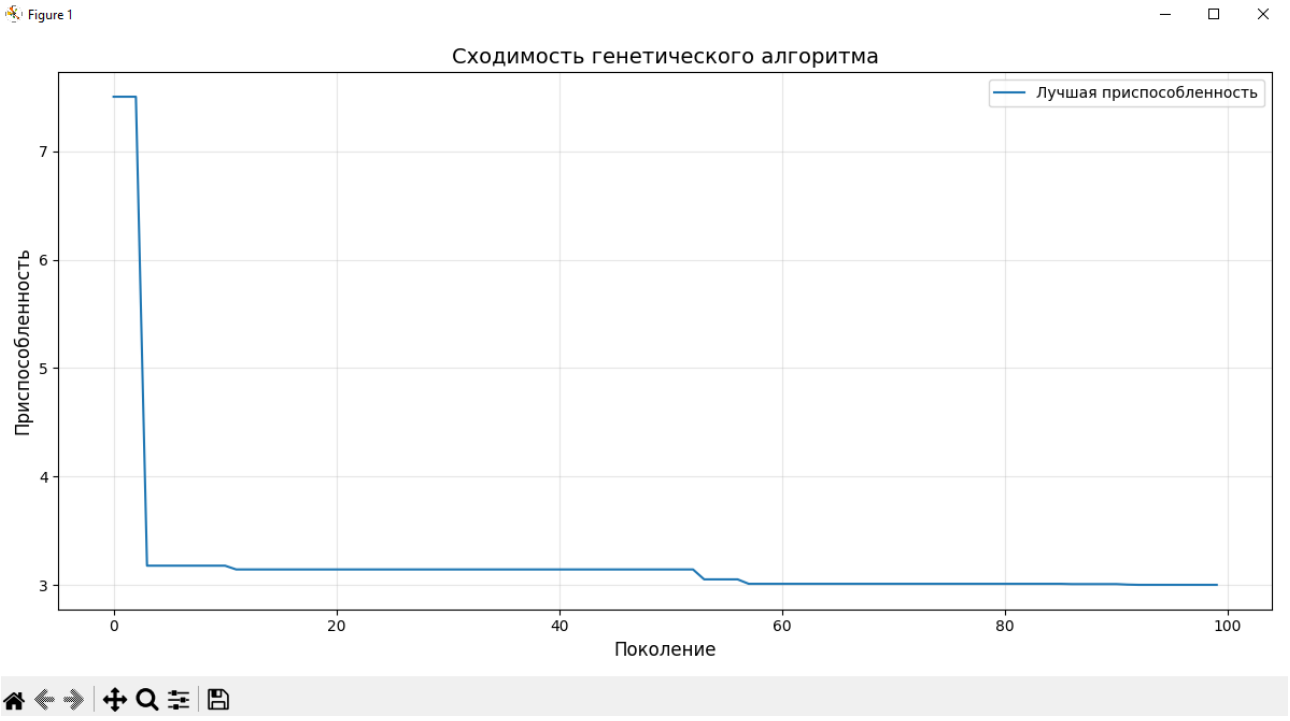

Рисунок 3.1 – Построенный график сходимости

Результат работы алгоритма с заданными параметрами представлен на Рисунке 3.2.

```
Поколение 82: лучшая приспособленность=3.010237, значение функции=3.010237, координаты=(-0.004600, -0.997561)
Поколение 83: лучшая приспособленность=3.010237, значение функции=3.010237, координаты=(-0.004600, -0.997561)
Поколение 84: лучшая приспособленность=3.010237, значение функции=3.010237, координаты=(-0.004600, -0.997561)
Поколение 85: лучшая приспособленность=3.007934, значение функции=3.007934, координаты=(-0.004600, -1.003845)
Поколение 86: лучшая приспособленность=3.007934, значение функции=3.007934, координаты=(-0.004600, -1.003845)
Поколение 87: лучшая приспособленность=3.007934, значение функции=3.007934, координаты=(-0.004600, -1.003845)
Поколение 88: лучшая приспособленность=3.007934, значение функции=3.007934, координаты=(-0.004600, -1.003845)
Поколение 89: лучшая приспособленность=3.007934, значение функции=3.007934, координаты=(-0.004600, -1.003845)
Поколение 90: лучшая приспособленность=3.003118, значение функции=3.003118, координаты=(-0.002707, -1.002515)
Поколение 91: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 92: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 93: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 94: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 95: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 96: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 97: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 98: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Поколение 99: лучшая приспособленность=3.000731, значение функции=3.000731, координаты=(0.000461, -1.001141)
Лучшее решение: x=0.000461, y=-1.001141
Значение функции: 3.000731
```

Рисунок 3.2 – Результат работы алгоритма

График сходимости для алгоритма с турнирным принципом отбора представлен на Рисунке 3.3.

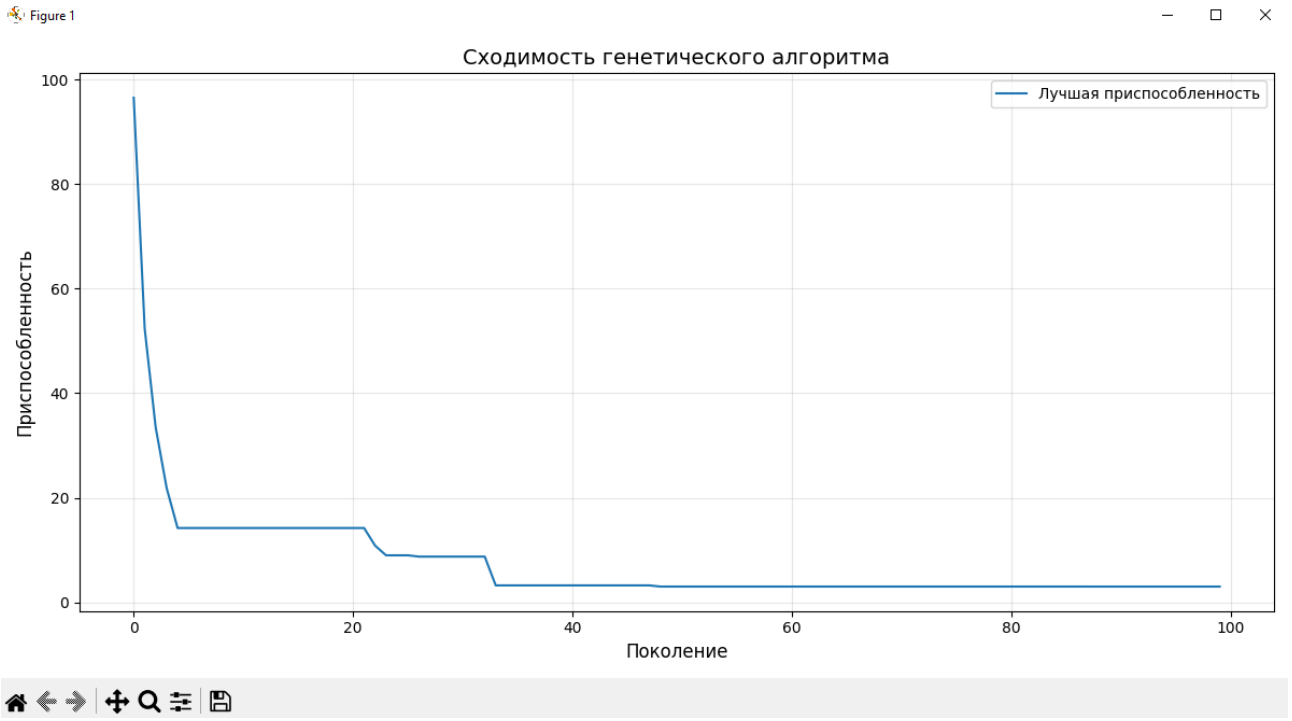

Рисунок 3.3 – График сходимости с турнирным алгоритмом отбора

Результат работы алгоритма с турнирным принципом отбора представлен на Рисунке 3.4.

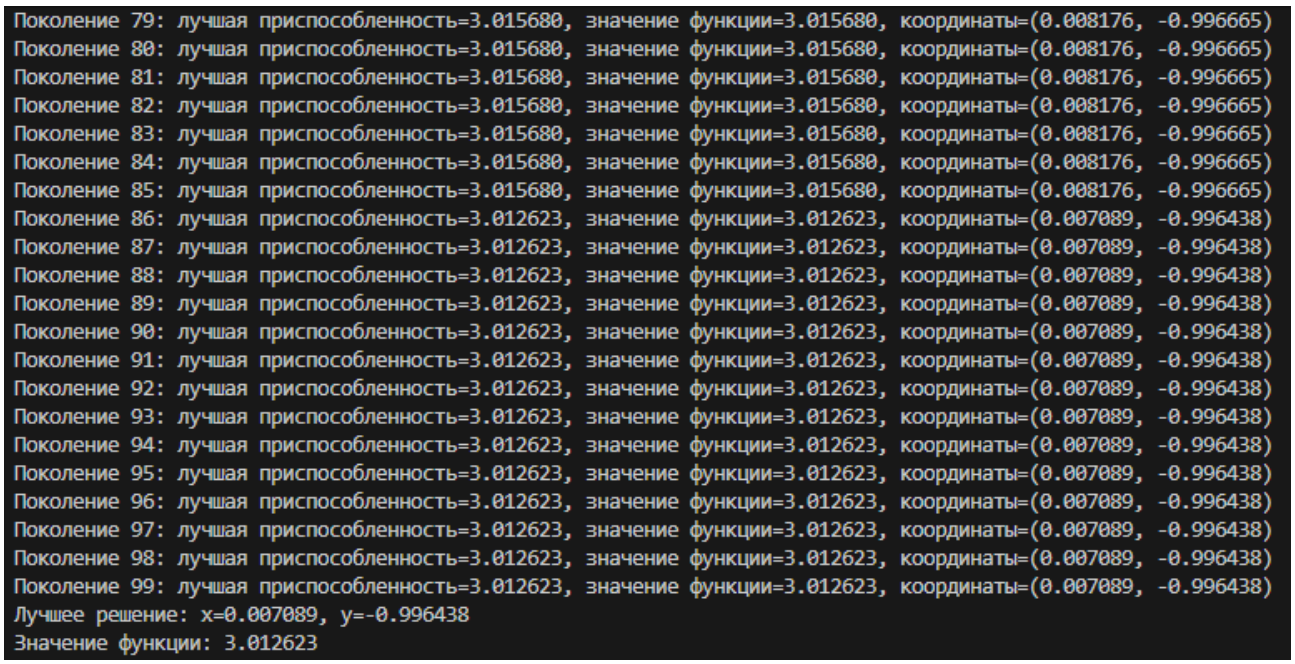

Рисунок 3.4 - Результат работы алгоритма с турнирным принципом отбора

Найденное решение в обоих случаях остается очень близким к глобальному минимуму оптимизируемой функции.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы изучена теория генетических алгоритмов как одного из методов эволюционных вычислений, применяемых для решения сложных оптимизационных задач. Особое внимание уделено принципам работы генетических алгоритмов, включая кодирование решений, функции приспособленности, операторы селекции, скрещивания и мутации.

Результаты проведенных экспериментов показали, что реализованный генетический алгоритм успешно справляется с задачей поиска глобального минимума функции Гольдштейна-Прайса. Обе тестируемые стратегии селекции (рулетка и турнир) демонстрируют устойчивую сходимость к оптимальному решению с высокой точностью.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
2. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
3. Сорокин, А. Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие / А. Б. Сорокин – Москва: Московский технологический университет (МИРЭА), 2019.
4. Генетические алгоритмы — математический аппарат. URL: https://loginom.ru/blog/ga-math (дата обращения: 10.12.2025).

ПРИЛОЖЕНИЯ

Приложение А — Код файла individual.py.

Приложение Б — Код файла genetic_algorithm.py.

Приложение А

Код файла individual.py

Листинг А – Код файла individual.py

```
from typing import Callable

class Individual:
    """Класс, представляющий особь (хромосому) в популяции"""

    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y
        self.fitness = 0.0

    def __repr__(self):
        return f"Individual(x={self.x:.6f}, y={self.y:.6f},
fitness={self.fitness:.6f})"

    def calculate_fitness(self, fitness_func: Callable) -> float:
        """Вычисление приспособленности особи"""
        self.fitness = fitness_func(self.x, self.y)
        return self.fitness
```

Приложение Б

Код файла genetic_algorithm.py

Листинг Б – Код файла genetic_algorithm.py

```
import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from typing import List, Tuple, Callable
from individual import Individual

def goldstein_price(x: float, y: float) -> float:
    """
    Функция Гольдштейна-Прайса для оптимизации.
    Глобальный минимум: f(0, -1) = 3
    """
    term1 = 1 + (x + y + 1) ** 2 * (
        19 - 14 * x + 3 * x**2 - 14 * y + 6 * x * y + 3 * y**2
    )
    term2 = 30 + (2 * x - 3 * y) ** 2 * (
        18 - 32 * x + 12 * x**2 + 48 * y - 36 * x * y + 27 * y**2
    )
    return term1 * term2

class GeneticAlgorithm:
    """Класс генетического алгоритма для оптимизации функции двух переменных"""

    def __init__(
        self,
        fitness_function: Callable,
        bounds: List[Tuple[float, float]],
        population_size: int = 50,
        max_generations: int = 100,
        crossover_rate: float = 0.8,
        mutation_rate: float = 0.1,
        selection_method: str = "tournament",
        tournament_size: int = 3,
        elitism_count: int = 2,
        crossover_method: str = "arithmetic",
        mutation_method: str = "uniform",
        epsilon = 1e8
    ):
        self.fitness_function = fitness_function
        self.bounds = bounds
        self.population_size = population_size
        self.max_generations = max_generations
        self.crossover_rate = crossover_rate
        self.mutation_rate = mutation_rate
        self.selection_method = selection_method
        self.tournament_size = tournament_size
        self.elitism_count = elitism_count
        self.crossover_method = crossover_method
        self.mutation_method = mutation_method
        self.epsilon = epsilon

        self.best_fitness_history = []

        self.best_individual = None
        self.best_fitness_value = float("inf")
```

Продолжение Листинга Б

```
        self.population = self._initialize_population()

    def _initialize_population(self) -> List[Individual]:
        """Инициализация начальной популяции"""
        population = []
        for _ in range(self.population_size):
            x = random.uniform(self.bounds[0][0], self.bounds[0][1])
            y = random.uniform(self.bounds[1][0], self.bounds[1][1])
            individual = Individual(x, y)
            individual.calculate_fitness(self.fitness_function)
            population.append(individual)
        return population

    def _calculate_fitness_all(self):
        """Вычисление приспособленности для всей популяции"""
        for individual in self.population:
            individual.calculate_fitness(self.fitness_function)

    def _roulette_wheel_selection(self) -> Individual:
        """Селекция методом рулетки для минимизации"""
        max_fitness = max(ind.fitness for ind in self.population)
        inverted_fitness = [
            max_fitness - ind.fitness + 1e-10 for ind in self.population
        ]
        total_inverted = sum(inverted_fitness)

        if total_inverted == 0:
            return random.choice(self.population)

        pick = random.uniform(0, total_inverted)
        current = 0

        for i, individual in enumerate(self.population):
            current += inverted_fitness[i]
            if current >= pick:
                return individual

        return self.population[-1]

    def _tournament_selection(self) -> Individual:
        """Турнирная селекция"""
        tournament = random.sample(self.population, self.tournament_size)
        return min(tournament, key=lambda ind: ind.fitness)

    def _select_parent(self) -> Individual:
        """Выбор родителя в зависимости от метода селекции"""
        if self.selection_method == "roulette":
            return self._roulette_wheel_selection()
        elif self.selection_method == "tournament":
            return self._tournament_selection()
        else:
            return self._tournament_selection()

    def _single_point_crossover(
        self, parent1: Individual, parent2: Individual
    ) -> Tuple[Individual, Individual]:
        """Одноточечное скрещивание"""
        if random.random() > self.crossover_rate:
            return (
                Individual(parent1.x, parent1.y),
```

Продолжение Листинга Б

```
                Individual(parent2.x, parent2.y),
            )

        crossover_point = random.randint(0, 1)

        if crossover_point == 0:
            child1 = Individual(parent2.x, parent1.y)
            child2 = Individual(parent1.x, parent2.y)
        else:
            child1 = Individual(parent1.x, parent2.y)
            child2 = Individual(parent2.x, parent1.y)

        return child1, child2

    def _arithmetic_crossover(
        self, parent1: Individual, parent2: Individual
    ) -> Tuple[Individual, Individual]:
        """Арифметическое скрещивание"""
        if random.random() > self.crossover_rate:
            return (
                Individual(parent1.x, parent1.y),
                Individual(parent2.x, parent2.y),
            )

        alpha = random.random()

        x1 = alpha * parent1.x + (1 - alpha) * parent2.x
        x2 = alpha * parent2.x + (1 - alpha) * parent1.x

        y1 = alpha * parent1.y + (1 - alpha) * parent2.y
        y2 = alpha * parent2.y + (1 - alpha) * parent1.y

        child1 = Individual(x1, y1)
        child2 = Individual(x2, y2)

        return child1, child2

    def _uniform_mutation(self, individual: Individual) -> Individual:
        """Равномерная мутация"""
        if random.random() > self.mutation_rate:
            return individual

        if random.random() < 0.5:
            individual.x = random.uniform(self.bounds[0][0], self.bounds[0][1])
        if random.random() < 0.5:
            individual.y = random.uniform(self.bounds[1][0], self.bounds[1][1])

        return individual

    def _crossover(
        self, parent1: Individual, parent2: Individual
    ) -> Tuple[Individual, Individual]:
        """Скрещивание в зависимости от выбранного метода"""
        if self.crossover_method == "single_point":
            return self._single_point_crossover(parent1, parent2)
        elif self.crossover_method == "arithmetic":
            return self._arithmetic_crossover(parent1, parent2)
        else:
            return self._arithmetic_crossover(parent1, parent2)

    def _apply_elitism(self) -> List[Individual]:
```

Продолжение Листинга Б

```
        """Применение элитизма - сохранение лучших особей"""
        sorted_population = sorted(self.population, key=lambda ind: ind.fitness)
        return sorted_population[: self.elitism_count]

    def _update_best_solution(self):
        """Обновление лучшего решения"""
        current_best = min(self.population, key=lambda ind: ind.fitness)
        if current_best.fitness < self.best_fitness_value:
            self.best_fitness_value = current_best.fitness
            self.best_individual = Individual(current_best.x, current_best.y)
            self.best_individual.fitness = current_best.fitness

    def run(self) -> Tuple[Individual, float]:
        """Запуск генетического алгоритма"""
        for generation in range(self.max_generations):

            fitness_values = [ind.fitness for ind in self.population]
            self.best_fitness_history.append(min(fitness_values))

            self._update_best_solution()

            new_population = []

            elites = self._apply_elitism()
            for elite in elites:
                ind = Individual(elite.x, elite.y)
                ind.fitness = elite.fitness
                new_population.append(ind)

            while len(new_population) < self.population_size:
                parent1 = self._select_parent()
                parent2 = self._select_parent()

                child1, child2 = self._crossover(parent1, parent2)

                child1 = self._uniform_mutation(child1)
                child2 = self._uniform_mutation(child2)

                child1.calculate_fitness(self.fitness_function)
                child2.calculate_fitness(self.fitness_function)

                new_population.append(child1)
                if len(new_population) < self.population_size:
                    new_population.append(child2)

            self.population = new_population

            best_ind = min(self.population, key=lambda ind: ind.fitness)
            actual_value = self.fitness_function(best_ind.x, best_ind.y)
            print(
                f"Поколение {generation}: лучшая
приспособленность={best_ind.fitness:.6f}, "
                f"значение функции={actual_value:.6f},
координаты=({best_ind.x:.6f}, {best_ind.y:.6f})"
            )

        self._update_best_solution()

        actual_value = self.fitness_function(
            self.best_individual.x, self.best_individual.y
        )
```

Окончание Листинга Б

```
        print(
            f"Лучшее решение: x={self.best_individual.x:.6f},
y={self.best_individual.y:.6f}"
        )
        print(f"Значение функции: {actual_value:.6f}")

        return self.best_individual, actual_value

    def plot_convergence(self):
        """Построение графика сходимости"""
        plt.figure(figsize=(12, 6))

        generations = list(range(len(self.best_fitness_history)))
        plt.plot(
            generations,
            self.best_fitness_history,
            label="Лучшая приспособленность",
        )
        plt.title("Сходимость генетического алгоритма", fontsize=14)
        plt.xlabel("Поколение", fontsize=12)
        plt.ylabel("Приспособленность", fontsize=12)
        plt.legend()
        plt.grid(True, alpha=0.3)

        plt.tight_layout()
        plt.show()

if __name__ == "__main__":
    bounds = [(-2, 2), (-2, 2)]
    ga = GeneticAlgorithm(
        fitness_function=goldstein_price,
        bounds=bounds,
        population_size=50,
        max_generations=100,
        crossover_rate=0.8,
        mutation_rate=0.1,
        selection_method="roulette",
        tournament_size=3,
        elitism_count=2,
        crossover_method="arithmetic",
        mutation_method="uniform",
    )

    best_individual, best_value = ga.run()

    ga.plot_convergence()

bounds = [(-2, 2), (-2, 2)]
ga = GeneticAlgorithm(
    fitness_function=goldstein_price,
    bounds=bounds,
    population_size=50,
    max_generations=100,
    crossover_rate=0.8,
    mutation_rate=0.1,
    selection_method="tournament",
    tournament_size=3,
    elitism_count=2,
    crossover_method="arithmetic",
    mutation_method="uniform",
)
```