МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки / специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Сравнение генетического и нейросетевого подходов к решению задач нелинейной оптимизации

Обучающийся В.С. Савельев
(Инициалы Фамилия) (личная подпись)

Руководитель канд. физ.-мат. наук, О. В. Лелонд
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. филол. наук М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Сравнение генетического и нейросетевого подходов к решению задач нелинейной оптимизации».

Работа посвящена исследованию и сравнительному анализу двух подходов – генетических алгоритмов и искусственных нейронных сетей – в решении задач нелинейной оптимизации, которые широко применяются в различных отраслях. Актуальность темы обусловлена растущей сложностью оптимизационных задач и необходимостью разработки эффективных вычислительных методов их решения.

Во введении обоснована значимость выбора методов для задач оптимизации, сформулированы цель и задачи исследования, определены объект и предмет исследования, а также обозначены программные инструменты, применяемые в работе.

В первой главе рассмотрены теоретические основы и принципы работы генетических алгоритмов и нейронных сетей, проведён их сравнительный анализ по ряду параметров.

Во второй главе разработана формализация задач нелинейной оптимизации, использованных в качестве тестовых, а также описана реализация программных моделей каждого подхода на языке Python.

В третьей главе проведено экспериментальное сравнение эффективности методов по метрикам точности, скорости сходимости и устойчивости.

В заключении сформулированы выводы о применимости каждого подхода и предложены направления дальнейшего развития, включая гибридные решения.

Работа включает введение, три главы, заключение, список литературы, иллюстрации и таблицы с результатами моделирования.

Объём работы: 71 страница, 16 рисунков, 6 таблиц, 30 источников, 1 приложение.

Abstract

Title of the Bachelor's Thesis: Comparison of Genetic and Neural Network Approaches to Solving Nonlinear Optimization Problems

This thesis is devoted to the study and comparative analysis of two approaches – genetic algorithms and artificial neural networks – in solving nonlinear optimization problems that are widely used across various domains. The relevance of the topic stems from the increasing complexity of optimization tasks and the need to develop efficient computational methods for their solution.

The introduction substantiates the importance of method selection for optimization problems, formulates the research objectives and tasks, defines the object and subject of the study, and outlines the software tools used.

The first chapter presents the theoretical foundations and principles of genetic algorithms and neural networks, as well as a comparative analysis of these approaches based on several key parameters.

The second chapter develops a formalization of the nonlinear optimization problems used as benchmarks and describes the implementation of program models for each approach using Python.

The third chapter contains an experimental comparison of the methods in terms of accuracy, convergence rate, and robustness.

The conclusion summarizes the applicability of each approach and outlines directions for further development, including hybrid solutions.

The thesis includes an introduction, three chapters, a conclusion, a list of references, figures, and tables with modeling results.

Total volume: 71 pages, 16 figures, 6 tables, 30 sources, 1 application.

Оглавление

Введение

Современные научно-технические достижения и рост вычислительных мощностей способствуют активному развитию методов искусственного интеллекта, применяемых для решения сложных задач, в том числе в области оптимизации. Нелинейная оптимизация как отдельное направление прикладной математики и вычислительной инженерии приобретает всё большее значение в условиях, когда классические аналитические и численные методы оказываются недостаточно гибкими или ресурсоемкими для решения задач с высокой размерностью, сложными ограничениями и неявными зависимостями.

Нелинейные задачи оптимизации широко встречаются в таких сферах, как логистика, управление производственными процессами, проектирование инженерных систем, обучение моделей машинного обучения и многие другие. Общей чертой этих задач является наличие сложной и часто многомодальной функции целевой оценки, нахождение глобального минимума или максимума которой представляет собой вычислительно сложную задачу. В таких условиях особую актуальность приобретают эвристические и метаэвристические подходы, такие как генетические алгоритмы и нейросетевые методы, основанные на принципах машинного обучения.

Генетические алгоритмы имитируют эволюционные механизмы естественного отбора и адаптации, что делает их пригодными для решения задач с плохо формализуемыми критериями и множеством локальных экстремумов. В то же время искусственные нейронные сети демонстрируют высокую способность к аппроксимации сложных функций и адаптивному обучению на основе большого объема данных. Несмотря на различную природу, оба подхода активно развиваются и находят применение в оптимизационных задачах, однако систематического сравнительного анализа их эффективности применительно к широкому классу нелинейных задач на практике до сих пор недостаточно.

Актуальность темы настоящей работы обусловлена необходимостью выбора адекватного и эффективного метода оптимизации в условиях сложных и многопараметрических задач, возникающих в прикладных науках и инженерных приложениях. Сравнение генетического и нейросетевого подходов позволяет выявить их сильные и слабые стороны, определить области предпочтительного применения каждого из методов и создать основу для построения гибридных решений, сочетающих их преимущества.

Объектом исследования являются методы оптимизации, применяемые к решению задач с нелинейной, многомодальной и многопараметрической природой.

Предметом исследования выступают генетические алгоритмы и нейросетевые подходы как инструменты оптимизации, а также их сравнительная эффективность при решении задач нелинейной оптимизации.

Цель выпускной квалификационной работы заключается в сравнении генетического и нейросетевого подходов к решению задач нелинейной оптимизации, а также в оценке их эффективности, устойчивости и вычислительной сложности на базе экспериментальных данных.

Для достижения поставленной цели необходимо решить следующие задачи:

- выполнить обзор теоретических основ генетических алгоритмов и нейронных сетей, применяемых к задачам оптимизации;
- провести анализ существующих методов нелинейной оптимизации и классифицировать задачи, к которым применимы сравниваемые подходы;
- разработать и реализовать программные модели оптимизации с использованием генетических алгоритмов и нейросетевых архитектур;
- провести серию вычислительных экспериментов на репрезентативных тестовых задачах, обеспечивая сопоставимость условий;

– сравнить результаты моделирования по основным метрикам (точность, скорость сходимости, устойчивость к параметрам и шуму);
– сформулировать выводы о применимости каждого метода и представить рекомендации по выбору подхода в зависимости от типа решаемой задачи.

Таким образом, работа направлена не только на практическую реализацию и тестирование алгоритмов, но и на проведение содержательного анализа, способствующего более обоснованному выбору методов оптимизации в исследовательской и прикладной деятельности.

Структура работы включает введение, три главы, заключение и список использованной литературы. В первой главе рассматриваются теоретические основы и современные тенденции в области генетических алгоритмов и нейросетевых подходов к оптимизации. Во второй главе описана разработка программных моделей и экспериментальной среды. В третьей главе проведён сравнительный анализ эффективности методов на основе вычислительных экспериментов и сделаны выводы о применимости каждого из них.

Ожидаемым результатом исследования является программный продукт, который выполнит выявление преимуществ и ограничений каждого из подходов на основе количественного сравнения, с помощью которого будет составлен список практических рекомендаций по выбору метода оптимизации в зависимости от характера решаемой задачи.

Глава 1 Анализ генетического и нейросетевого подходов к решению задач нелинейной оптимизации в современном мире

1.1 Актуальность применения интеллектуальных методов в задачах нелинейной оптимизации

Задачи нелинейной оптимизации широко распространены в различных отраслях: от логистики и маршрутизации транспорта до инженерного проектирования и энергетики. Эти задачи, как правило, характеризуются сложными нелинейными зависимостями, большим количеством переменных и наличием множества локальных экстремумов. В общем виде задача нелинейной оптимизации формулируется следующим образом: найти экстремум $F(x), x \in R^n$, при условиях: $g_i(x) \geq 0, i = 1, \ldots\ldots, m$; $h_j(x) = 0, j = 1, \ldots\ldots, p$, где F(x) – целевая (критериальная) функция, $g_i(x)$ и $h_j(x)$ – функции ограничений-неравенств и ограничений-равенств соответственно [1][2][5].

В отличие от линейного случая, экстремум нелинейной задачи не обязан находиться на границе области допустимых решений, и целевая функция может иметь сложный ландшафт с множеством локальных экстремумов. Локальный минимум – это точка, в окрестности которой значение функции минимально, однако существование нескольких локальных минимумов усложняет поиск глобального оптимума. Как отмечается в литературе, наличие множества локальных минимумов делает задачу глобальной оптимизации трудной, так как градиентные методы могут «застревать» в локальных экстремумах.

На графике (рисунок 1) показана одномерная функция (например, функция Растригина [5][24] – типичная тестовая функция с большим числом локальных минимумов) с глобальным минимумом (отмечен красной точкой) и одним из локальных минимумов (оранжевая точка). Видно, что помимо глобального экстремума при x=0, функция имеет ряд локальных экстремумов,

которые создают сложности для алгоритмов, основанных на поиске снижения градиента: такие алгоритмы могут сойтись к ближайшему локальному минимуму вместо глобального решения.

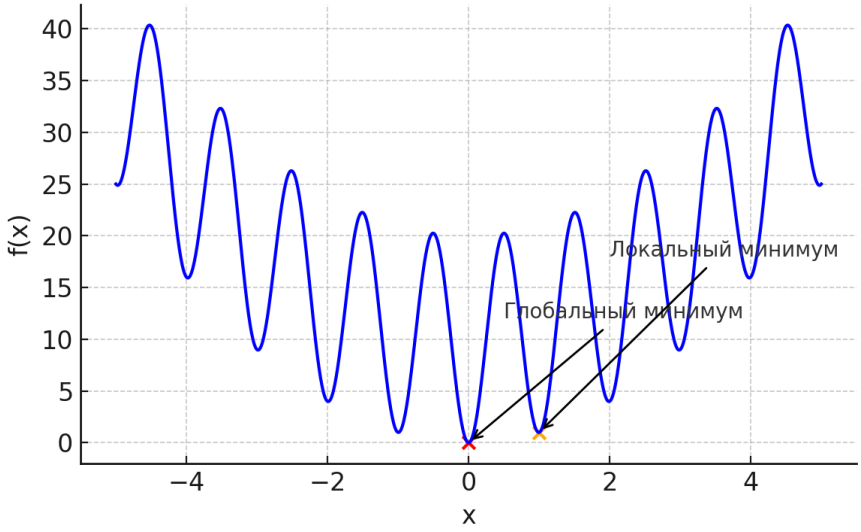

Рисунок 1 – Пример нелинейной функции с несколькими локальными минимумами

Классические алгоритмы оптимизации (градиентные методы, методы математического программирования и др.) зачастую не справляются с такими сложными задачами в приемлемое время или застревают в локальных решениях. Например, в электроэнергетике для задачи оптимального распределения потоков мощности (OPF) было показано, что традиционные детерминированные методы (градиентные, метод внутренней точки и др.) недостаточно эффективны для крупномасштабных невыпуклых проблем. Это обусловлено тем, что оптимизационный ландшафт может быть мультимодальным (с множеством локальных минимумов) и сильно нелинейным, из-за чего классические подходы склонны находить лишь субоптимальные решения.

Для преодоления ограничений классических методов все большее распространение получают интеллектуальные методы оптимизации – подходы, базирующиеся на идеях искусственного интеллекта и природных

механизмов [7]. В последние годы эволюционные алгоритмы и методы машинного обучения активно интегрируются в процессы оптимизации, демонстрируя высокую эффективность. Так, в упомянутой задаче OPF вместо сугубо градиентных методов успешно применяются метаэвристические алгоритмы, включая генетические алгоритмы [14][18] (ГА), алгоритм роя частиц (PSO), искусственные нейронные сети [13][16] (ИНН), методы пчелиного роя, серого волка, имитацию отжига и др. Использование таких интеллектуальных подходов позволяет гибко учитывать сложные ограничения системы и решать многокритериальные задачи [3][8], недоступные для традиционных алгоритмов. В частности, метаэвристики способны эффективно обходить локальные экстремумы и находить близкие к глобальному оптимумы в сложных нелинейных задачах [6][12].

Актуальность применения интеллектуальных методов подтверждается успешными примерами в разных прикладных областях. В логистике задачи оптимизации маршрутов и расписаний (например, задача маршрутизации транспорта, VRP) являются комбинаторно сложными (NP-трудными), поэтому для их решения привлекаются эвристические алгоритмы. Практические исследования показывают преимущество таких подходов: например, улучшенный генетический алгоритм нашел оптимальное решение задачи распределения доставок менее чем за 2 минуты, тогда как традиционный алгоритм потребовал до 4–5 минут. Это свидетельствует о заметно более высокой производительности интеллектуальных методов в задачах логистики. Аналогично, в инженерном проектировании эволюционные алгоритмы успешно применяются для оптимизации конструкций и технологических процессов, где необходимо учитывать сложные нелинейные зависимости. Генетические алгоритмы, в частности, зарекомендовали себя как один из важных инструментов оптимизации сложных систем. Отмечается, что ГА являются эффективным средством глобального поиска решения сложных задач оптимизации, хотя у них есть и недостатки (например, риск преждевременной сходимости к локальному

решению и относительно медленная скорость достижения глобального оптимума.

Кроме эволюционных алгоритмов, все больше внимания уделяется и нейросетевым подходам к оптимизации. Искусственные нейронные сети могут применяться для моделирования и приближения сложных зависимостей в оптимизационных задачах, что открывает новые возможности для поиска решений [9]. Например, нейросеть может выступать в роли аппроксиматора сложной целевой функции или ограничения (т. н. surrogate model), который затем встраивается в процедуру оптимизации. Это позволяет существенно ускорить оценку качества решений в процессе поиска за счет замены вычислительно дорогой модели приближенной нейросетевой моделью. Развитие технологий глубокого обучения привело к появлению подходов, где нейросеть обучается самостоятельно находить приближенные оптимальные решения определенного класса задач. Подобные методы применяются, например, в комбинаторной оптимизации маршрутов и управлении, демонстрируя конкурентоспособные результаты. Таким образом, сочетание методов искусственного интеллекта – эволюционных алгоритмов и нейросетей – является актуальным направлением, продиктованным ростом сложности современных задач оптимизации. Интеллектуальные методы уже доказали свою эффективность там, где классические подходы достигают предела возможностей, что обосновывает необходимость их изучения и применения.

Понимание значимости интеллектуальных методов ставит задачу вписать их в общий контекст методов оптимизации. В следующем пункте приведена классификация методов оптимизации – от градиентных до метаэвристических – позволяющая увидеть место генетических алгоритмов и нейросетей среди всего спектра подходов.

1.2 Классификация методов оптимизации: от градиентных к метаэвристическим

Методы оптимизации разнообразны, однако их можно условно разделить на два больших класса: традиционные (градиентные, детерминированные) и интеллектуальные (метаэвристические, стохастические) [7]. К первому классу относятся классические алгоритмы математического программирования и вычислительной математики: метод градиентного спуска, метод Ньютона, методы линейного и нелинейного программирования, метод внутренней точки и др. Эти методы основываются на строгих математических принципах и обычно используют информацию о градиентах целевой функции. Ко второму классу относятся современные метаэвристические алгоритмы, которые используют стохастические процедуры поиска и часто вдохновлены природными явлениями. Примеры включают генетические алгоритмы [14][18], эволюционные стратегии, алгоритмы муравьиной колонии, роя частиц, имитации отжига, и другие подобные подходы. В отличие от градиентных методов, метаэвристики не привязаны к конкретной предметной области и могут применяться практически к любой задаче оптимизации, задавая лишь общие эвристические правила поиска решения.

Градиентные методы. Детерминированные градиентные алгоритмы итеративно улучшают текущее решение, двигаясь в направлении антиградиента (для минимизации) или используя информацию о втором порядке (методы Ньютона) при наличии такой возможности. Они эффективны на гладких и выпуклых задачах, где гарантируют быстрое нахождение глобального оптимума. Однако в общем случае нелинейной невыпуклой оптимизации градиентные методы подвержены попаданию в ловушки локальных минимумов. Как только алгоритм достигает локального экстремума, дальнейшее улучшение останавливается, даже если решение далеко от глобального оптимума. В сложных многомодальных задачах

традиционные градиентные алгоритмы часто не могут выйти за пределы окрестности одного локального оптимума. Кроме того, градиентные методы требуют дифференцируемости целевой функции и наличия точной информации о градиентах. Если функция цели недифференцируема, имеет шум или задается только процедурно (черный ящик), применение градиентного подхода затруднительно или требует специальных модификаций. Таким образом, хотя классические методы (например, метод наискорейшего спуска, BFGS и др.) очень эффективны в задачах малого размера при хороших свойствах функции, их применение ограничено для сложных нелинейных случаев.

Метаэвристические методы. В противоположность градиентным алгоритмам, метаэвристики осуществляют глобальный стохастический поиск по пространству решений и тем самым менее склонны застревать в локальных экстремумах. Многие из них оперируют не единичным решением, а целой популяцией кандидатных решений, что повышает шанс исследования различных областей пространства и приближения к глобальному оптимуму. Важной чертой метаэвристик является их гибкость: они предъявляют минимум требований к виду задачи. В частности, целевая функция вовсе не обязана быть непрерывной или дифференцируемой – алгоритму достаточно уметь вычислять ее значение, а градиенты не требуются. Благодаря этому метаэвристические методы легко применимы к задачам любой природы (дискретным, нелинейным, многомодальным и пр.), где классические методы неприменимы. Кандидатные решения при этом могут быть представлены в произвольной форме – от простых векторов чисел до сложных структур (например, маршрутов, расписаний, деревьев выражений), – важно лишь уметь генерировать новые варианты (через операторы мутации, кроссовера и т.п. для ГА). Еще одним плюсом метаэвристик является масштабируемость и параллелизм: многие алгоритмы естественным образом допускают распределение вычислений, поскольку оценка качества разных кандидатных решений независима и может выполняться параллельно. Это позволяет

эффективно использовать современные многопроцессорные и облачные системы для ускорения поиска решений.

Наряду с преимуществами, у метаэвристических методов есть и ограничения. Прежде всего, они являются эвристическими – то есть не гарантируют нахождения точного глобального оптимума. В отличие от градиентных методов с доказанной сходимостью в выпуклых задачах, метаэвристики не имеют строгих оценок погрешности. Тем не менее на практике они часто дают достаточно хорошие решения за разумное время, и эти решения могут превосходить по качеству результаты, полученные детерминированными алгоритмами, особенно в сложных задачах. Другой недостаток – относительная требовательность к вычислительным ресурсам. Поскольку алгоритму зачастую приходится проверить большое количество вариантов (особенно при высокой размерности задачи), число вычислений целевой функции может быть очень большим. Если оценка каждого решения сложна или время затратна, общий процесс оптимизации может оказаться медленным. Также, будучи универсальными, метаэвристики не используют в полной мере специфические свойства конкретной задачи, что иногда приводит к меньшей эффективности по сравнению со специальными алгоритмами, настроенными под данную задачу. Тем не менее развитие вычислительной техники и появление параллельных реализаций сглаживают этот недостаток. В итоге метаэвристические алгоритмы зарекомендовали себя как надежный инструмент для глобальной оптимизации: несмотря на отсутствие гарантированной точности, они способны в приемлемые сроки находить близкие к оптимальным решения в задачах, где другие методы бессильны.

Гибридные методы [14][21] сочетают достоинства градиентных и метаэвристических подходов. В таких алгоритмах глобальный поиск (например, генетический алгоритм) дополняется локальной оптимизацией (например, градиентным спуском на финальном этапе или на каждом шаге). Гибридизация позволяет, с одной стороны, эффективно исследовать пространство решений в глобальном масштабе, а с другой – быстро уточнять

найденные решения за счёт локального поиска. Таким образом, гибридные алгоритмы обладают высокой универсальностью и способны совместить приемлемое время работы с приближением к глобально оптимальным решениям.

В таблице 1 представлены характеристики градиентных, метаэвристических и гибридных методов [14][21].

Таблица 1 – Характеристика градиентных, метаэвристических и гибридных методов

Метод	Применимость	Необходимость градиента	Сходимость	Гибкость	Универсальность
Градиентные	Дифференцируемые функции; локальный поиск	Да	Быстрая (локальный экстремум)	Низкая (застревают в локальных решениях)	Ограниченная (только гладкие задачи)
Метаэвристические	Широкий класс задач (в т.ч. дискретные)	Нет	Медленная (глобальный стохастический поиск)	Высокая (настраиваемые операторы)	Очень высокая (применимы даже к «черному ящику»)
Гибридные	Широкий класс задач	Частично (градиент на локальной стадии)	Комбинированная (глобальный поиск + локальная донастройка)	Высокая	Высочайшая (объединяет преимущества методов)

Как видно из таблицы 1, градиентные методы эффективны при наличии хороших свойств функции (гладкости, выпуклости), тогда как метаэвристические алгоритмы более универсальны и не требуют специальных свойств задачи, но уступают по скорости и гарантированности сходимости. Гибридный подход позволяет достичь компромисса, используя сильные стороны каждого класса методов.

Таким образом, спектр методов оптимизации простирается от быстрых локальных градиентных методов до мощных глобальных стохастических

алгоритмов. Выбор подхода определяется характером решаемой задачи – ее размерностью, свойствами целевой функции, наличием ограничений и критериев. В частности, сложные задачи с нелинейными ограничениями или множеством критериев требуют гибких подходов (например, метаэвристик), тогда как для простых гладких задач целесообразно применять эффективные градиентные методы. Понимание достоинств и недостатков разных классов методов позволяет обоснованно подходить к выбору инструмента оптимизации.

1.3 Ограничения и критерии в задачах оптимизации

Любая задача оптимизации определяется двумя основными компонентами: критерием оптимизации (целевой функцией) и системой ограничений. Критерий задает меру качества решения – обычно в виде функции, которую требуется минимизировать или максимизировать. Ограничения определяют множество допустимых решений, накладывая условия на переменные задачи. Формально типичную задачу нелинейной оптимизации [1][2][5] можно описать следующим образом: найти вектор переменных x, минимизирующий функцию F(x) (например, издержки или потери) при выполнении ограничений $g_i(x) \geq 0, i = 1, \ldots\ldots, m;$ и $h_j(x) = 0, j = 1, \ldots\ldots, p,$ которые задают допустимую область. Здесь неравенства $g_i(x) \geq 0$ могут представлять пределы ресурсов, технологические требования и т. д., а равенства $h_j(x) = 0$ – балансовые соотношения или связи между переменными. Решение, удовлетворяющее всем ограничениям, называется допустимым (feasible). Задача оптимизации заключается в том, чтобы среди всех допустимых найти решение, которое оптимизирует выбранный критерий (наилучшее по значению целевой функции). В реальных задачах число ограничений и критериев может быть значительным, что усложняет поиск оптимума и требует специальных методов.

Наличие ограничений существенно влияет на метод решения оптимизационной задачи. В простейшем случае без ограничений задача сводится к классическому беспрепятственному поиску экстремума функции – такую задачу относительно легко решать градиентными методами. Однако в большинстве практических случаев присутствуют ограничения, которые делают область поиска сложной (например, разрывной, негладкой или дискретной). Методы оптимизации должны либо строго обеспечивать выполнение ограничений на каждом шаге, либо штрафовать решения, выходящие за допустимую область [7]. Широко распространенный подход для учета ограничений – введение штрафных функций (penalty functions) [6][13]. Его суть заключается в преобразовании задачи с ограничениями к последовательности безусловных задач путем добавления к целевой функции штрафа за нарушение ограничений. Например, для задачи минимизации при наличии ограничений-неравенств $g_i(x) \geq 0$ и ограничений-равенств $h_j(x) = 0$ можно ввести следующую штрафную функцию (1).

$$\Phi(x, r) = f(x) + r \sum_{i=1}^{m} \max\{0, g_i(x)\}^2 + r \sum_{j=1}^{P} \left(h_j(x)\right)^2, \qquad (1)$$

где

$r \gg 0$ – коэффициент штрафа (большое положительное число).

В полученной безусловной задаче минимизации $\Phi(x, r)$ нарушения ограничений приводят к росту значения функции за счёт квадратичного штрафа. По мере решения увеличивают коэффициент r, усиливая наказание за недопустимые решения, так чтобы при $r \to \infty$ глобальный минимум $\Phi(x, r)$ стремился к допустимому решению исходной задачи. Существуют варианты метода штрафов: внешний (пенальный) метод, как в формуле выше, накладывает штраф за выход за границы области, а внутренний (барьерный) метод напротив, вводит барьер внутри области, препятствуя приближению к границе области с недопустимой стороны. Метод штрафных функций прост в

реализации и позволяет применять аппараты безусловной оптимизации для условных задач, однако требует подбора коэффициентов штрафа и может приводить к ill-posed (численно неустойчивым) задачам при слишком больших штрафах.

Другим классическим подходом к учету ограничений является метод множителей Лагранжа, в котором вводятся дополнительные переменные (множители) для каждого ограничения и формируется лагранжиан. Однако методы на основе штрафных функций часто более просты и подходят для использования в составе эволюционных алгоритмов, где прямой учёт ограничений затруднён.

Кроме штрафных методов, существуют и специальные алгоритмы обработки ограничений: проекционные методы (проектируют недопустимое решение на ближайшее допустимое), метод барьерных функций, адаптивные операторы мутации, сохраняющие допустимость, и др. Выбор стратегии зависит от характера ограничений. Например, линейные ограничения часто учитываются прямо в алгоритмах линейного программирования, в то время как нелинейные ограничения в ГА обычно обрабатываются штрафами или хитрыми операторами скрещивания [4][15], сохраняющими часть решений допустимыми. Важно отметить, что ограничения могут значительно усложнить задачу оптимизации, делая невозможным прямое применение стандартных методов. Поэтому разработка эффективных стратегий учета ограничений является важным аспектом при создании оптимизационных алгоритмов.

Роль критериев и многокритериальные задачи. В классической постановке имеется один целевой критерий (функция) – например, стоимость, вес конструкции, время выполнения и т. п. Однако во многих реальных ситуациях требуется одновременно оптимизировать несколько критериев, которые часто конфликтуют друг с другом. Такая постановка называется многокритериальной (многоцелевой) оптимизацией. Многокритериальная оптимизация формулируется как задача поиска таких x, которые улучшают

сразу несколько функций $f_1(x), f_2(x), \ldots, fk(x)$. Как правило, не существует единственного решения, одновременно оптимального по всем критериям (если критерии конфликтующие). Вместо этого ищут совокупность компромиссных решений, называемых Парето-оптимальными. Решение x называется Парето-оптимальным, если не существует другого решения x в области допустимых решений, которое не уступает $\hat{x}$ сразу по всем критериям и превосходит по крайней мере по одному из них. Множество всех Парето-оптимальных точек образует так называемый фронт Парето (границу Парето) в пространстве целевых функций.

На графике (рисунок 2) показана совокупность решений для задач с двумя критериями (f_1 и f_2), демонстрирующая принцип доминирования.

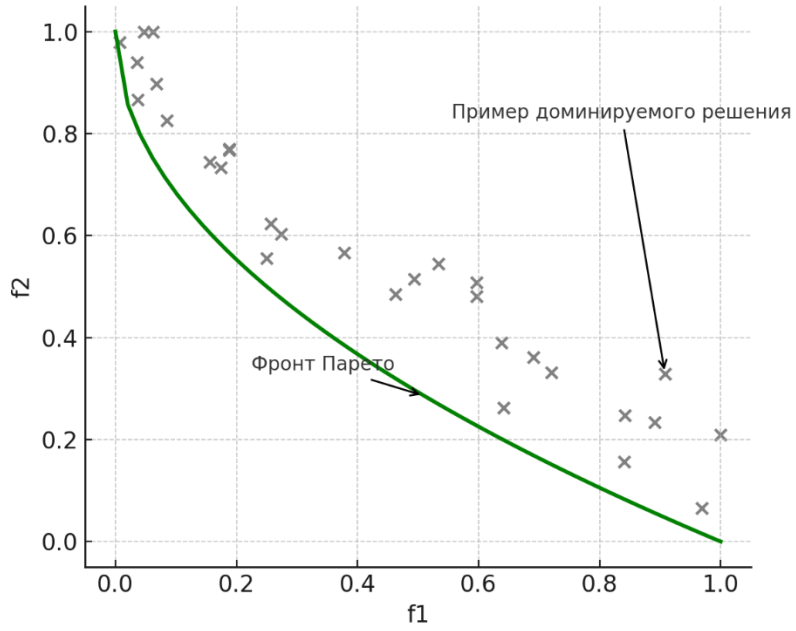

Рисунок 2 – Фронт Парето для двух критериев

Зеленой кривой отмечен фронт Парето – все точки на этой кривой недоминируемы (улучшение одного критерия приводит к ухудшению другого). Серым крестом показаны примеры доминируемых решений: например, отмеченная стрелкой точка уступает любой точке на фронте Парето (для нее найдется решение, лучшее по обоим показателям).

В таблице 2 приведены некоторые типичные прикладные задачи нелинейной оптимизации [1][2][5] и подходы, которые обычно применяются для их решения.

Таблица 2 - Прикладные задачи и используемые подходы

Прикладная задача	Типичные подходы к решению
Обучение нейронной сети (настройка весов по данным, минимизация функции ошибки)	Градиентные методы (стохастический градиентный спуск, backpropagation); в сложных ландшафтах – гибридные (градиент + эволюционный подход для избегания локальных минимумов)
Комбинаторная оптимизация (распределение ресурсов, маршрут, расписание)	Метаэвристические алгоритмы (генетические алгоритмы, муравьиный алгоритм, имитация отжига), поскольку задача дискретная, нелинейная, с большим числом локальных экстремумов
Оптимизация параметров технической системы (например, настройка коэффициентов регулятора, проектирование конструкции)	Гибридные методы: глобальный поиск (генетический или дифференциальная эволюция) для общей оптимизации + локальный градиентный подстрой финального решения для повышения точности; либо специальные методы (градиентные при наличии дифференцируемой модели)
Многокритериальная оптимизация в инженерии (например, оптимизация конструкции по прочности и массе)	Эволюционные многокритериальные алгоритмы (NSGA-II, SPEA2), которые генерируют приближение фронта Парето за одно выполнение алгоритма

Примеры прикладных задач из таблицы 2 иллюстрируют, что выбор метода существенно зависит от характера задачи. Когда целевая функция гладкая и хорошо поддается анализу, предпочтительны градиентные методы. В случаях же, когда функция имеет сложный ландшафт или задача имеет дискретный характер, на первый план выходят генетические и другие эвристические алгоритмы. Гибридные подходы эффективны на задачах, требующих как глобального поиска, так и точной локальной настройки решения.

1.4 Перспективы гибридных подходов и обоснование выбора генетических алгоритмов и нейросетей

Современные тенденции в исследовании методов оптимизации показывают рост интереса к гибридным подходам, которые комбинируют различные алгоритмы для достижения синергетического эффекта [22][23]. Идея гибридизации состоит в том, чтобы объединить преимущества нескольких методов и компенсировать их недостатки. Например, метаэвристические алгоритмы можно сочетать с градиентными методами: сначала глобальная стохастическая стратегия находит перспективную область решения, а затем локальный градиентный метод быстро дорабатывает решение до ближайшего локального оптимума. Такой подход позволяет, с одной стороны, избежать застревания в плохих локальных минимумах, а с другой – воспользоваться высокой скоростью сходимости градиентных алгоритмов при приближении к экстремуму. Другой пример гибрида – комбинация разных метаэвристик, где, скажем, генетический алгоритм отвечает за глобальный поиск, а другой алгоритм (например, имитация отжига) периодически применяется для локального улучшения каждого решения. В литературе отмечается, что объединение метаэвристик с классическими методами дает ощутимый выигрыш, и такие гибриды все чаще исследуются учеными. Например, указывается, что метаэвристика может использоваться для нахождения удачной стартовой точки или оптимизации архитектуры модели, после чего градиентный спуск выполняет тонкую настройку решения. Альтернативно, можно комбинировать две метаэвристики, используя их разные сильные стороны, – подобные подходы также находятся в фокусе современных исследований.

Генетические алгоритмы (ГА) – это класс эволюционных методов оптимизации, основанный на механизмах естественного отбора и генетики. Алгоритм оперирует популяцией потенциальных решений (индивидов), к которым применяются операторы, аналогичные биологическим [11][19]:

селекция, скрещивание (кроссовер) и мутация. ГА не требует информации о производных функции и способен работать с самыми разными форматами решения (бинарные строки, вещественные векторы и др.), что делает его универсальным для глобальной оптимизации.

На схеме (рисунок 3) показана общая структура ГА: сначала происходит инициализация популяции случайными решениями. Затем каждое решение оценивается через вычисление функции приспособленности (значения целевой функции или меры качества).

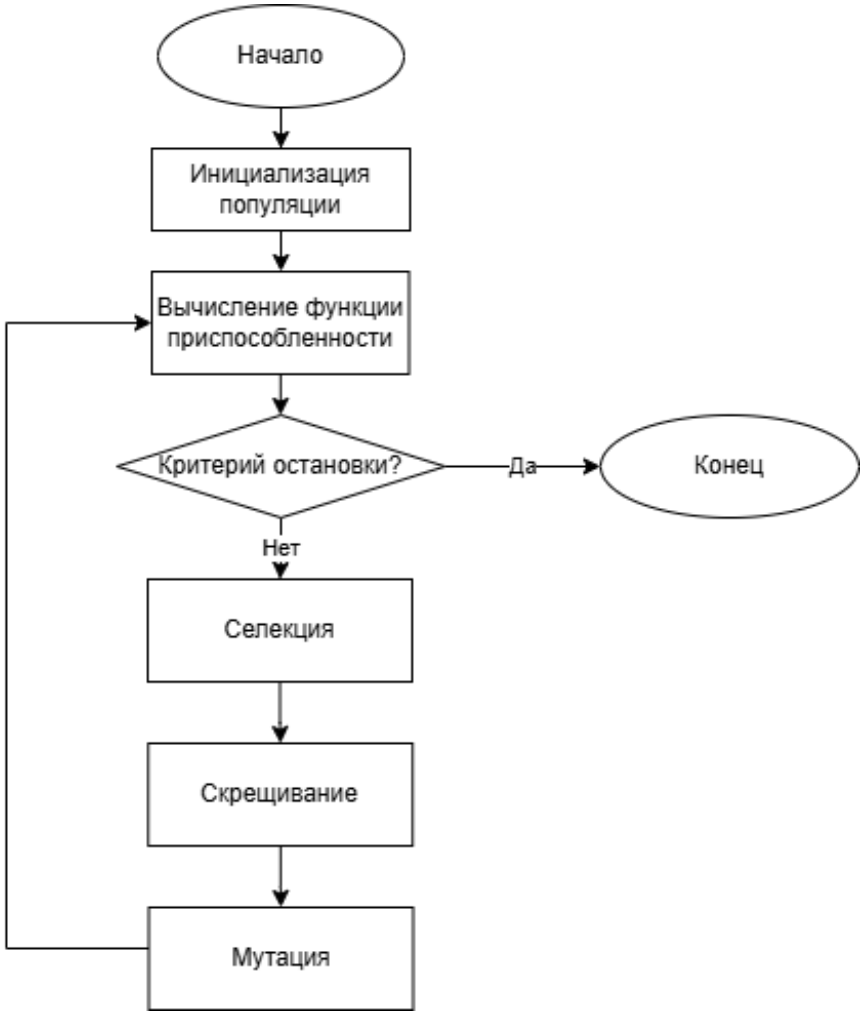

Рисунок 3 – Блок-схема генетического алгоритма

Проверяется критерий остановки – обычно это достижение максимального числа поколений или достаточного качества решения. Если

критерий не выполнен (“нет”), выполняется итерация эволюционных операторов: сначала селекция (выбор наиболее приспособленных решений или вероятностный отбор с bias к лучшим), затем скрещивание (обмен частями решений между выбранными родительскими особями для получения новых решений), и мутация (случайное малое изменение в потомках для поддержания разнообразия). После этих операций формируется новая популяция, и алгоритм возвращается к этапу оценки приспособленности следующего поколения. Цикл продолжается до выполнения критерия остановки (“да”), после чего алгоритм завершается, выдавая лучшее найденное решение. ГА стохастичен по природе, поэтому для воспроизводимости результатов обычно используются несколько запусков и усреднение или выбор лучшего решения среди запусков.

На практике генетические алгоритмы [14][18] хорошо зарекомендовали себя в задачах, где пространство решений велико, а функция цели имеет множество локальных экстремумов или не выражена в явном виде. Несмотря на отсутствие формальной гарантии нахождения глобального оптимума, ГА часто находит решения высокого качества при достаточном времени работы.

Нейросетевой подход использует возможности обучаемых искусственных нейронных сетей для поиска решения оптимизационных задач. Идея состоит в том, чтобы настроить параметры нейронной сети (веса и смещения) таким образом, чтобы минимизировать определённую целевую функцию, связанную с исходной задачей оптимизации. Например, можно построить нейросетевую модель, которая по заданным входным данным генерирует решение x, и обучать эту модель, минимизируя функцию ошибки, определяемую как значение целевой функции F(x) (а также штрафы за нарушения ограничений, если необходимо). Таким образом, процесс обучения нейронной сети становится процессом оптимизации: используя алгоритмы обучения (как правило, градиентные, через backpropagation), сеть приближает оптимальное решение задачи.

На схеме (рисунок 4) показан классический цикл обучения нейронной сети, применимый и к задачам оптимизации: на этапе инициализации нейросети задаются начальные веса (например, случайно). Затем выполняется прямой проход (вычисление выходов сети и значения целевой функции или ошибки для текущих параметров). После этого производится вычисление ошибки – например, разницы между текущим значением целевой функции (или выходом сети) и желаемым значением (в случае прямой оптимизации сама целевая функция может играть роль ошибки).

Рисунок 4 - Блок-схема нейросетевого алгоритма

Проверяется критерий остановки обучения (достижение требуемого уровня ошибки или исчерпание числа эпох). Если решение ещё недостаточно

хорошее, выполняется шаг коррекции весов – применение алгоритма оптимизации (обычно градиентного спуска, возможно с моментумом или другим методом) для обновления параметров сети в направлении уменьшения ошибки. Затем цикл повторяется: новый прямой проход с обновленными весами и т. д. Когда критерий остановки выполнен, процесс обучения заканчивается, и настроенная нейросеть задаёт приближенное оптимальное решение исходной задачи.

Нейросетевой подход особенно привлекателен, когда требуется решить семейство сходных задач – тогда модель можно обучить на множестве примеров и она будет выдавать решение практически мгновенно для новых входных данных. Однако применение нейросетевого подхода требует преобразования исходной задачи в форму, пригодную для обучения сети, а также дифференцируемости сформулированной функции ошибки (чтобы можно было применять градиентные методы обучения).

В ситуациях, где целевая функция задана алгоритмически или отсутствуют данные для обучения, нейросетевой подход может быть затруднён. Кроме того, обучение глубокой модели само по себе – сложная задача оптимизации, подверженная проблемам локальных минимумов и переобучения, поэтому часто нейросетевой метод сочетают с другими подходами.

Особенно перспективными считаются гибриды, сочетающие эволюционные алгоритмы и нейронные сети, поскольку эти методы по своей природе дополняют друг друга. Эволюционные алгоритмы (в частности, генетические) осуществляют параллельный глобальный поиск и не требуют градиентной информации, тогда как нейронные сети способны обучаться на данных и быстро вычислять приближенные результаты после обучения [9]. Объединение этих возможностей открывает новые пути решения сложных задач оптимизации.

Помимо использования нейронной сети в качестве модели объекта оптимизации, существует и обратный подход: применение генетических

алгоритмов для обучения самой нейронной сети. Обычно нейронные сети обучаются градиентными методами (например, методом обратного распространения ошибки), однако в некоторых случаях эволюционные алгоритмы оказываются более эффективными [9].

Генетический алгоритм может оптимизировать веса сети или архитектуру сети (выбирая структуру слоев, количество нейронов, гиперпараметры) на основе целевой функции качества работы сети.

Поддержка нейросетей со стороны эволюционных алгоритмов может быть особенно полезна при оптимизации архитектур, которые не поддаются градиентному обучению напрямую, – например, в случае дискретного выбора числа слоёв, функций активации или типов связей между слоями. В таких задачах генетический алгоритм позволяет проводить поиск по пространству гиперпараметров [14][20], опираясь на общую мета-целевую функцию, отражающую производительность сети. Это делает его эффективным компонентом автоматизированного проектирования нейросетевых моделей, что наглядно демонстрирует актуальность интеграции рассматриваемых методов.

Таким образом, в первой главе рассмотрены основные понятия и подходы к решению задач нелинейной оптимизации [1][2][5]. Представлены математические формулировки и методы учета ограничений, особенности многокритериальных постановок и понятие Парето-оптимальности, а также различные классы алгоритмов – градиентные, метаэвристические и их сочетания.

Эти теоретические положения и наглядные схемы создают основу для дальнейшего сравнения генетического и нейросетевого подходов, которое будет проведено в последующих разделах работы.

Глава 2 Разработка компьютерной модели для сравнения генетического и нейросетевого алгоритмов

2.1 Особенности реализации генетического алгоритма и нейросетевого подхода в задачах оптимизации

Генетические алгоритмы (ГА) и нейросетевые методы представляют два разных подхода к решению задач нелинейной оптимизации [1][2][5].

Генетический алгоритм – это эвристический метод поиска, имитирующий эволюционные процессы, такие как наследование, отбор, скрещивание и мутации. ГА оперирует популяцией возможных решений, представленных в виде хромосом (наборов параметров). Алгоритм итеративно улучшает популяцию: на каждой итерации (поколении) вычисляется функция приспособленности для каждого решения, затем отбираются наиболее приспособленные особи, которые подвергаются операторам генетического воспроизведения – скрещиванию (recombination) и мутации. В результате генерируется новое поколение решений. Процесс продолжается до выполнения критерия остановки (например, достижения приемлемого решения или исчерпания числа поколений).

Таким образом, реализация ГА требует задания способа кодирования решений (генотипа), функции оценки (целевая или функция приспособленности) и набора операторов эволюции. Особенностью ГА является его стохастический характер – использование случайности в генерации начальной популяции и в операторах мутации и скрещивания. Это позволяет глобально исследовать пространство решений и обходить локальные минимумы, но делает результаты алгоритма вероятностными.

Практическая реализация ГА относительно проста и не требует информации о градиентах целевой функции – достаточно уметь вычислять функцию приспособленности для любого заданного решения [16][18]. Благодаря этому ГА успешно применяются к широкому классу задач – от

непрерывной оптимизации параметров до дискретных комбинаторных задач, где отсутствуют аналитические методы вычисления оптимума.

Нейросетевой подход к оптимизации принципиально отличается [17][20]. Искусственная нейронная сеть (ИНС) – это модель, состоящая из множества связанных между собой простых вычислительных элементов (нейронов), способная приближать сложные нелинейные зависимости.

В контексте оптимизации нейросети обычно используются двояко: (1) в качестве оптимизируемой модели, параметр которой (веса сети) настраиваются для минимизации некоторой целевой функции; или (2) в качестве вспомогательного инструмента, который приближает целевую функцию или политику принятия решений, после чего оптимум определяется на основе обученной сети. Первый подход подразумевает, что сам процесс обучения нейронной сети решает задачу оптимизации – например, настройка весов сети производится методом градиентного спуска с целью минимизации функции потерь, которая представляет нашу целевую функцию оптимизации.

С математической точки зрения обучение нейросети – это многопараметрическая нелинейная задача оптимизации. На практике обычно задаётся дифференцируемая функция потерь, и с помощью алгоритмов типа стохастического градиентного спуска (SGD) или его современных модификаций (Momentum, Adam и др.) происходит итеративное улучшение параметров сети.

Таким образом, нейросетевой подход к решению прикладной оптимизационной задачи часто сводится к формулировке этой задачи как задачи обучения: мы строим архитектуру сети, определяем функцию потерь, и обучаем сеть на данных таким образом, чтобы минимизировать эту функцию. В результате обученная сеть может выдавать решение оптимизационной задачи (например, прогнозировать оптимальные параметры).

В отличие от ГА, нейросеть не генерирует множество решений одновременно, а постепенно обучается находить хорошее решение, обобщая

знания из обучающих данных. В таблице 3 приведено краткое сравнение особенностей реализации генетического алгоритма и нейросетевого подхода.

Таблица 3 – Сравнение реализации ГА и нейросетевого подхода в задачах оптимизации

Критерий	Генетический алгоритм (ГА)	Нейросетевой подход (НС)
Тип подхода	Стохастический, популяционный	Обучаемая модель, градиентный подход
Работа с производными	Не требуется	Обычно требуется
Необходимость обучающей выборки	Нет	Да
Поиск решения	Глобальный, эволюционный	Градиентная оптимизация или аппроксимация
Представление решения	Генотип (вектор параметров)	Параметры нейросети
Поддержка ограничений	Через штрафы или фильтрацию	Через модификацию функции потерь
Скорость генерации решения	Средняя (зависит от популяции)	Очень высокая (после обучения)
Гибкость к типу задачи	Высокая	Средняя (зависит от формализации)

Таким образом, особенности реализации обоих подходов различны. ГА работает с популяцией решений и не требует априорных данных для обучения – только определение функции пригодности, – зато может потребовать много итераций оценки и отбора. Нейросетевой же подход обычно требует больших данных для обучения (наблюдений или симуляций), а также наличия алгоритма обучения (градиентного или эволюционного), который настроит параметры сети под задачу.

В дальнейшем в этой главе мы сравним эти подходы по ряду аспектов – от исторического развития и технологических средств до современных тенденций, чтобы выделить их сильные и слабые стороны применительно к нелинейной оптимизации [1][2][5].

2.2 Технологическая специфика: программные средства и инструментарий для ГА и нейросетей

Развитие вычислительной техники и программных средств сыграло огромную роль в продвижении как генетических алгоритмов, так и нейронных сетей в практику. Рассмотрим современные программные библиотеки и фреймворки, облегчающие реализацию этих подходов в задачах нелинейной оптимизации [1][2][5].

Для генетических алгоритмов создано множество библиотек на разных языках программирования. В среде Python [27][29] одним из наиболее популярных является пакет DEAP [25] (Distributed Evolutionary Algorithms in Python). DEAP – это фреймворк эволюционных вычислений для быстрого прототипирования и тестирования идей; он предоставляет готовые структуры данных и инструменты для реализации классических генетических алгоритмов, генетического программирования, эволюционных стратегий и даже методов ройного интеллекта.

Преимуществом DEAP [25] является гибкость – он позволяет настроить или заменить любые компоненты алгоритма, а также поддерживает параллельные вычисления (что важно для ускорения оптимизации на больших популяциях).

Кроме DEAP, в Python [27][29] известны библиотеки PyGAD и Inspyred. PyGAD предоставляет готовые реализации различных вариантов скрещивания, мутации и селекции, допускает пользовательскую настройку функции фитнеса. Более того, PyGAD содержит модули интеграции с нейронными сетями: модуль pygad.gann позволяет оптимизировать веса искусственной нейросети с помощью генетического алгоритма. Таким образом, PyGAD поддерживает гибридные нейроэволюционные эксперименты «из коробки».

Для нейронных сетей экосистема инструментов ещё более богата, особенно благодаря буму глубокого обучения. К наиболее распространённым

фреймворкам относятся TensorFlow (от Google) и PyTorch (от Meta/Facebook).

Оба фреймворка поддерживают автоматическое дифференцирование (необходимое для градиентного обучения сетей), работу на GPU, содержат богатые коллекции готовых слоёв, функций активации, оптимизаторов (SGD, Adam, RMSProp и т.д.). Для задач оптимизации с использованием нейросетей эти инструменты позволяют строить произвольные модели – от многослойных перцептронов до рекуррентных или свёрточных сетей – и обучать их на нужных данных, минимизируя заданную функцию потерь.

Основные классы и примеры технологических инструментов для реализации ГА и НС представлены в таблице 4.

Таблица 4 – Основные классы технологических инструментов для генетических алгоритмов и нейросетевых методов

Класс инструмента	Генетические алгоритмы (ГА)	Нейросетевые методы (НС)
Библиотеки общего назначения	DEAP, PyGAD, Inspyred (Python), ECJ (Java), Watchmaker (Java), Accord.NET (C#)	TensorFlow, PyTorch, Keras
Специализированные модули	pygad.gann (оптимизация весов НС с помощью ГА), Global Optimization Toolbox (MATLAB)	Keras Tuner (гиперпараметры), Optuna, Ray Tune
Гибридные фреймворки (ГА + НС)	NEAT-Python (нейроэволюция), pygad.gann, собственные связки DEAP + PyTorch	PyTorch + DEAP (гибридизация вручную), Auto-Keras с внешней оптимизацией
Средства параллелизма и ускорения	DEAP + multiprocessing, Dask	CUDA, cuDNN, TensorFlow XLA, PyTorch Lightning
Облачные и распределённые платформы	Google Colab, AWS, Docker-образ с GA-кодом	Google Colab, AWS SageMaker, Vertex AI, Deepnote
Визуализация и логирование	Matplotlib, seaborn, TensorBoard (через обвязки), custom plotting	TensorBoard, Weights & Biases, matplotlib, MLflow

В контексте сравнительного анализа важно подчеркнуть: ГА готовы к использованию даже без специализированных фреймворков – их реализация относительно проста, и многие исследователи пишут необходимый код под конкретную задачу вручную (например, используя массивы NumPy для

хранения популяций). Тем не менее, наличие библиотек (DEAP [25], PyGAD) ускоряет разработку и делает эксперименты воспроизводимыми.

Нейросети же практически невозможно эффективно реализовать с нуля без фреймворков – ручная реализация градиентного обучения для сложной сети крайне трудоёмка и чревата ошибками. Поэтому успех нейросетевого подхода во многом обусловлен доступностью мощных библиотек, скрывающих низкоуровневую сложность.

Резюмируя, технологическая среда для ГА и нейросетей достаточно зрелая. Существенное отличие в том, что для нейросетей существуют стандартизированные универсальные платформы (TensorFlow, PyTorch), покрывающие 95 % потребностей, тогда как для ГА выбор инструмента более разнообразен и часто зависит от предпочтений разработчика (иногда ГА внедряется «с нуля» под конкретную задачу, особенно в промышленных приложениях, чтобы иметь полный контроль над процессом). Тем не менее, и там, и там сообщество активно поддерживает открытые библиотеки, что способствует распространению этих методов.

2.3 Описание используемых методов и инструментов разработки

Разработка программного модуля для сравнения эффективности генетических алгоритмов и нейросетевых подходов при решении задач оптимизации требует обоснованного выбора методов и инструментов, обеспечивающих как корректную реализацию, так и объективность анализа результатов. В данной работе применяются современные библиотеки и фреймворки на языке Python [27][29], что обусловлено его популярностью в научных и прикладных задачах машинного обучения, оптимизации и анализа данных. Выбор инструментов и алгоритмов базируется на теоретическом анализе, проведённом в первой главе, и ориентирован на создание универсальной экспериментальной платформы, позволяющей оценить поведение методов в различных условиях.

В работе сравниваются два подхода к решению задач оптимизации – генетические алгоритмы [14][18] (ГА) и нейросетевые методы (НС). Каждый из них имеет собственные характеристики, преимущества и ограничения, что делает их сравнение актуальным и значимым в контексте прикладных и научных задач.

Генетический алгоритм: эвристический метод глобального поиска, вдохновлённый механизмами естественного отбора и эволюции. В рамках данной работы ГА используется для нахождения минимума целевой функции. Алгоритм включает этапы генерации начальной популяции, оценки приспособленности, отбора лучших решений, скрещивания и мутации. Используется библиотека DEAP [25] (Distributed Evolutionary Algorithms in Python [27][29]), которая обеспечивает гибкость в реализации пользовательских функций приспособленности, типов особей, а также конфигурации операторов генетики.

Нейросетевой метод: сама нейросетевая модель используется как аппроксиматор функции оптимизации или как решение задачи поиска минимума путём обучения модели на синтетических данных. Применяется фреймворк PyTorch, обеспечивающий динамическое построение вычислительных графов, автоматическое дифференцирование и поддержку вычислений на GPU.

В данной работе реализована многослойная полносвязная нейросеть с функцией активации ReLU и среднеквадратичной ошибкой (MSE) в качестве функции потерь. Обучение сети происходит на заранее подготовленных данных, где модель должна аппроксимировать функцию, приближаясь к её минимуму.

Разработка и реализация методов проводилась на языке Python [27][29], который обеспечивает высокую читаемость кода, доступ к научным библиотекам и широкий инструментарий для машинного обучения и оптимизации.

В таблице 5 представлены основные программные средства,

использованные в рамках данной работы.

Таблица 5 – Используемые инструменты и библиотеки

Инструмент	Назначение и роль в работе
Python	Основной язык разработки. Используется для реализации логики, визуализации и экспериментов.
DEAP	Библиотека для реализации генетических алгоритмов. Применялась для настройки популяций, операторов, визуализации сходимости.
PyTorch	Фреймворк глубокого обучения. Использовался для построения и обучения нейросетевых моделей.
NumPy	Работа с массивами данных, численные операции при подготовке выборок и параметров.
Pandas	Анализ и сохранение результатов экспериментов в табличной форме.
Matplotlib	Построение графиков функции потерь и визуализация сходимости.
Scikit-learn (опц.)	Применялся для предварительной обработки данных и расчёта метрик точности моделей.

Выбор методов и инструментов основывается на ряде основополагающих критериев.

– Универсальность и гибкость.

– Научная достоверность.

– Техническая поддержка и удобство.

– Возможность визуализации.

DEAP [25] и PyTorch позволяют адаптировать алгоритмы под разные типы задач и функций оптимизации. Это важно в рамках эксперимента, где сравнение проводится на различных функциях (Розенброка [5][24], Растригина и др.). Библиотеки имеют подробную документацию и поддерживаются сообществом, что упрощает их интеграцию в проект. Оба метода – ГА и НС – хорошо изучены, имеют обширную литературу и зарекомендовали себя как эффективные подходы для глобальной оптимизации. Используемые инструменты позволяют не только реализовать алгоритмы, но и визуализировать результаты, что важно для анализа эффективности каждого подхода.

2.4 Формализация компьютерной модели

В данном разделе приводится математическая формализация задач нелинейной оптимизации [1][2][5], решаемых в рамках выпускной квалификационной работы. Формализуются целевые постановки, условия оптимальности, методы численного решения с учётом ограничений, а также критерии многокритериального анализа. Завершается раздел описанием тестовых функций, применённых для экспериментального сравнения алгоритмов.

2.4.1 Математическая формализация задачи нелинейной оптимизации

Для построения вычислительной модели оптимизации необходимо чётко определить математическую постановку решаемой задачи. В основе лежит задача минимизации функции при наличии ограничений на допустимые значения переменных. Эта постановка является отправной точкой как для аналитического, так и для алгоритмического подхода к решению.

Пусть задана функция f: $\mathbb{R}^n \to \mathbb{R}$, называемая целевой функцией, которую требуется минимизировать. Дополнительные условия на допустимые значения переменных $x \in \mathbb{R}^n$ задаются через систему ограничений: минимизировать $f(x)$ при условии: $g_i(x) \leq 0, i = 1, \dots, m$; $h_j(x) = 0$, $j = 1, \dots, p$.

Здесь:

- $x = (x_1, x_2, \dots, x_n)^t \in \mathbb{R}^n$ – вектор переменных;
- $g_i(x): \mathbb{R}^n \to \mathbb{R}$ – функции-неравенства;
- $h_j(x): \mathbb{R}^n \to \mathbb{R}$ – функции-равенства.

Множество допустимых решений определяется как (2).

$$D = \{x \in \mathbb{R}^n \mid g_i(x) \leq 0, h_j(x) = 0\}. \quad (2)$$

Целью является нахождение такой точки $x^* \in D$, чтобы

$f(x*) \leq f(x), \forall x \in D.$

Выделяют следующую классификацию задач:

– Безусловная оптимизация: m = 0, p = 0.

– Оптимизация с ограничениями: m > 0 и/или p > 0.

– Выпуклая задача: если f(x) и $g_i(x)$ выпуклы, а $h_j(x)$ линейны.

– Невыпуклая задача: наличие множества локальных экстремумов.

После формулировки задачи оптимизации необходимо определить критерии, по которым можно судить о том, что найденное решение действительно оптимально. Для этого используются условия Каруша–Куна–Таккера (KKT) [8][12], которые обобщают необходимое условие экстремума на случай наличия ограничений.

Для вывода условий KKT необходимо выполнение условий регулярности (constraint qualifications). Одно из наиболее часто используемых условий – линейная независимость градиентов активных ограничений: $\{\nabla h_j(x^*)\}_{j=1}^{p} \cup \{\nabla g_i(x^*) \mid g_i(x^*) = 0\}$ – линейно независимы.

$\nabla h_j(x^*)$ – градиент равенств.

$\nabla g_i(x^*)$, где $g_i(x^*) = 0$ – градиенты активных неравенств.

Если x* ∈ D – локальный минимум и условия регулярности выполнены, то существуют множители Лагранжа $\lambda_i \geq 0$ и $\mu_j \in \mathbb{R}$ такие, что выполняются условия:

– Стационарность: $\nabla_x L(x^*, \lambda^*, \mu^*) = 0$.

– Допустимость: $g_i(x*) \leq 0; h_j(x*) = 0$.

– Комплементарность: $\lambda_i \geq 0; \lambda_i \cdot g_i(x*) = 0$.

Функция Лагранжа записывается как (3).

$$L(x, \lambda, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{j=1}^{p} \mu_j h_j(x) \quad (3)$$

Условия KKT являются основным инструментом анализа оптимальности в задачах с ограничениями. Они позволяют формализовать

структуру решения, строить алгоритмы (метод множителей Лагранжа, SQP, внутренние точки) и анализировать поведение оптимальных точек при варьировании параметров задачи.

Для уточнения характера экстремума, особенно в задачах с гладкими функциями, применяется анализ второго порядка. Он базируется на свойствах гессиана функции Лагранжа и позволяет отличать минимумы от седловых точек и максимума.

Обозначим матрицу Гессе: $H(x) = \nabla_x^2 L(x*, \lambda, \mu)$. Если H(x*) положительно определена на допустимом линейном подпространстве, то x* – строгий локальный минимум.

Поскольку аналитическое применение условий KKT не всегда удобно в численных методах, необходимо рассмотреть подходы, позволяющие учитывать ограничения при построении алгоритмов. В частности, широкое распространение получили методы штрафов и барьеров.

Метод внешних штрафов (penalty function) (4).

$$F(x) = f(x) + \rho * \Phi(x), \Phi(x) = \sum_{i=1}^{m} max\big(0, g_i(x)\big)^2 + \sum_{j=1}^{p} h_j(x)^2], \quad (4)$$

где

– $r \gg 0$ – коэффициент штрафа.

Метод внутренних барьеров (5).

$$F(x) = f(x) - \mu \sum_{i=1}^{m} \ln\big(-g_i(x)\big), \quad (5)$$

где

– $\rho > 0, \mu > 0$ – параметры метода.

Эти подходы позволяют применять методы безусловной оптимизации к задачам с ограничениями.

В ряде прикладных задач требуется учитывать сразу несколько критериев качества. В таких случаях применяют методы многокритериальной

оптимизации, где целью является построение множества компромиссных решений, называемого Парето–фронтом.

Многокритериальная оптимизация заключается в минимизации векторной функции (6).

$$min_{x \in D} F(x) = [f^1(x), \ldots, f_k(x)]^{\mathrm{t}}. \quad (6)$$

Точка x* называется Парето-оптимальной, если не существует x ∈ D, для которого $f_j(x) \leq f_j(x^*)$ для всех j и строго меньше для хотя бы одного j. Для поиска Парето-фронта применяются методы скаляризации, взвешенных сумм и эволюционные алгоритмы, например, NSGA-II.

Завершающим элементом формализации являются тестовые функции, используемые для верификации корректности и оценки эффективности реализованных алгоритмов. Они позволяют проводить сравнительный анализ и служат эталоном в оптимизационных исследованиях.

Функция Розенброка [5][24] (Rosenbrock function), также известная как «узкая долина», используется для тестирования градиентных методов (7).

$$f(x, y) = (1 - x)^2 + 100 \cdot (y - x^2)^2. \quad (7)$$

Функция Растригина [5, 24] демонстрирует многомодальность и высокую сложность глобального поиска (8).

$$f(x) = A \cdot n + (\sum_{i=1}^{n} {x_{\mathrm{i}}}^2 - A \cdot \cos(2\pi x_{\mathrm{i}})), \quad A = 10. \quad (8)$$

Функция Швефеля [5, 24] – одна из самых сложных в классе тестовых функций (9).

$$f(x) = 418.9829 \cdot n - \sum_{i=1}^{n} x_{\mathrm{i}} \cdot sin(\sqrt{|x_{\mathrm{i}}|}). \quad (9)$$

Эти функции используются для количественного анализа эффективности методов оптимизации. Каждая из них демонстрирует разные аспекты сложности: наличие множества локальных минимумов, несимметричность, узкие долины и др.

2.4.2 Математическое описание генетических алгоритмов

Переходя от общей постановки задач нелинейоной оптимизации к рассмотрению конкретных алгоритмических решений, рассмотрим подробно математическую модель генетического алгоритма (ГА).

В основе ГА лежит идея моделирования эволюционного отбора: популяция потенциальных решений подвергается стохастическим модификациям с целью поиска наилучшего значения целевой функции.

Пусть задана задача минимизации функции f: $\mathbb{R}^n \to \mathbb{R}$. Генетический алгоритм поддерживает популяцию P^t из M особей $x \in \mathbb{R}^n$ в поколении (10).

$$P^t = \{x_1{}^t, x_2{}^t, \ldots, x_{M^t}\}. \quad (10)$$

Каждая особь оценивается функцией приспособленности. В задаче максимизации можно использовать (11).

$$\varphi(x) = -f(x) \quad \text{или} \quad \varphi(x) = 1 / (1 + f(x)), \text{если } f(x) \geq 0. \quad (11)$$

Работа ГА основана на поэтапном применении трёх операторов: селекции, скрещивания и мутации. Их последовательное применение определяет эволюционную траекторию популяции.

Начальный оператор для рассмотрения – селекция (отбор). Оператор селекции влияет на вероятность участия конкретной особи в генерации потомства. На первом этапе из текущей популяции отбираются родительские особи. Этот выбор основывается на значениях функции приспособленности φ(x), отражающей «качество» особи:

– Рулеточная (пропорциональная) селекция (12):

$$P(x_\mathrm{i}) \;=\; \varphi(x_\mathrm{i}) \,/\, \sum_j \varphi(x_\mathrm{j}) \qquad (12)$$

– Турнирная селекция (размер турнира k): случайный отбор особей и выбор наилучшей по φ(x).

– Арифметический кроссовер (13):

$$x' \;=\; \alpha \cdot x_\mathrm{i} \;+\; (1 \;-\; \alpha) \cdot x_\mathrm{j}, \qquad \alpha \;\in\; [0{,}1] \qquad (13)$$

– BLX-α (Blend Crossover) (14):

$$x_\mathrm{k} \;\in\; [min(x_\mathrm{ik}, x_\mathrm{jk}) \;-\; \alpha \cdot d_\mathrm{k}, max(x_\mathrm{ik}, x_\mathrm{jk}) \;+\; \alpha \cdot d_\mathrm{k}] \qquad (14)$$

где

$$d_\mathrm{k} \;=\; |x_\mathrm{ik} \;-\; x_\mathrm{jk}|.$$

– Однородный кроссовер (uniform crossover): случайный выбор каждого компонента.

Для предотвращения преждевременной сходимости используется мутация, так как применяется к каждой координате с вероятностью и служит источником случайности и адаптивности:

– Гауссовская мутация: $x'_k \;=\; x_\mathrm{k} \;+\; \varepsilon, \qquad \varepsilon \;\sim\; \mathcal{N}(0, \sigma^2).$

– Равномерная мутация: $x'_k \;=\; x_\mathrm{k} \;+\; \eta, \qquad \eta \;\sim\; U(-\delta, \delta).$

После рассмотрения структуры и действия операторов, перейдём к анализу общей эволюционной схемы. Все изменения, происходящие с популяцией в ходе алгоритма, можно описать в терминах марковской динамики. Пусть P^t – популяция в поколении t. Тогда действия операторов можно представить в виде марковского перехода (15).

$$P^{\mathrm{t}+1} \;=\; M\left(C\left(S(P^\mathrm{t})\right)\right), \qquad (15)$$

где

S – оператор селекции,

C – оператор кроссовера,

M – оператор мутации.

Таким образом, три оператора задают вероятностное отображение $T: P^t \rightarrow P^{t+1}$, приводящее к постепенной адаптации популяции к глобальному экстремуму функции f(x).

Обозначим $p^t(x)$ – вероятность того, что особь x присутствует в популяции на поколении t. Тогда переходное распределение $p^{t+1}(x)$ (16) моделируется так:

$$p^{t+1}(x) = \sum_{y \in D} p^t(y) \cdot T(y \rightarrow x), \quad (16)$$

где

$T(y \rightarrow x)$ – вероятность трансформации особи y в x под действием операторов селекции, скрещивания и мутации.

В пределе при $t \rightarrow \infty$, при выполнении условий непрерывности, элитарности и стохастичности, $p^t(x)$ сужается к окрестности глобального минимума x*.

После анализа популяционной динамики возникает вопрос: насколько быстро и надёжно алгоритм достигает искомого решения. Оценка эффективности ГА может проводиться как теоретически, так и эмпирически.

Пусть x* – глобальный минимум функции f(x). Тогда при выполнении условий эргодичности и достаточного разнообразия популяции можно показать, что вероятность попадания в окрестность x* стремится к 1 при $t \rightarrow \infty$. Однако на практике оценивается эмпирическая сходимость:

– Среднее значение (17):

$$\bar{f} = \frac{1}{K} \sum_{k=1}^{K} f_{best}^{(k)}. \quad (17)$$

– Дисперсия (18):

$$D = \frac{1}{K}\sum_{k=1}^{K}\left(f_{best}^{(k)} - \bar{f}\right)^2. \quad (18)$$

– Вероятность достижения ε-оптимума (19):

$$P_\varepsilon = \frac{1}{K}\sum_{k=1}^{K} 1_{\{f_{best}^{(k)} - f^* \le \varepsilon\}}. \quad (19)$$

где

K – количество независимых запусков ГА.

Рассмотренные выше аспекты алгоритма зависят от конкретных значений гиперпараметров [14][20], которые необходимо подобрать с учётом особенностей задачи. Настройка параметров ГА оказывает решающее влияние на эффективность и стабильность процесса оптимизации.

Генетические алгоритмы успешно справляются с такими условиями благодаря:

— Отказу от необходимости аналитических производных;

— Лёгкой интеграции ограничений через штрафные функции;

— Возможности масштабирования и параллельной реализации.

Генетический алгоритм является универсальным методом глобальной оптимизации, не требующим специальных предположений о структуре целевой функции. Его адаптивность, способность к глобальному поиску и простота расширения делают ГА особенно эффективным инструментом при решении нелинейных и ограниченных задач, где другие методы становятся неприменимыми.

2.4.3 Математическое описание нейросетевых алгоритмов

Переходя от стохастических эволюционных методов к подходам, основанным на машинном обучении, рассмотрим нейросетевые алгоритмы как важную альтернативу при решении задач нелинейной оптимизации

[1][2][5]. Эти методы становятся особенно актуальными при высокой вычислительной сложности, отсутствии аналитических выражений целевой функции и необходимости моделирования нелинейных зависимостей.

Для формализации нейросетевого подхода начнём с архитектуры модели. Многослойный перцептрон (MLP, multilayer perceptron) представляет собой базовую форму искусственной нейронной сети прямого распространения. Эта структура включает последовательно соединённые слои, каждый из которых реализует аффинное преобразование с последующим применением нелинейной функции активации.

Пусть $x \in \mathbb{R}^n$ – входной вектор. Тогда проход сигнала через ℓ-й слой сети описывается следующими формулами (20).

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \text{ и } a^{(l)} = \varphi^{(l)}\left(z^{(l)}\right), \tag{20}$$

где:

$z^{(l)} \in \mathbb{R}^{n_\ell}$ – линейная комбинация входов текущего слоя,

$W^{(l)} \in \mathbb{R}^{n_\ell * n_{\ell-1}}$ – матрица весов ℓ-го слоя,

$b^{(l)} \in \mathbb{R}^{n_\ell}$ – вектор смещений,

$\varphi^{(l)}(\cdot)$ – компонентная нелинейная функция активации,

$a^{(l)}$ – вектор активаций (выход текущего слоя),

$a^{(0)} := x$ – входной вектор.

Полная нейросеть представляет собой композицию таких преобразований, где результат последнего слоя $a^{(L)}$интерпретируется как выход модели (21):

$$\hat{y} = f_\theta(x) = a^{(L)}. \tag{21}$$

где

L – общее число слоёв (включая выходной, но без входного).

Наиболее часто используемые функции активации включают:

– сигмоиду: σ(x) = $\sigma(x) = \frac{1}{1+e^{-x}}$.

– гиперболический тангенс: $\tan h(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$.

– ReLU (Rectified Linear Unit): $ReLU(x) = max(0, x)$.

Совокупность параметров сети (22) определяет её модельную сложность.

$$\theta = \{W^{(\ell)}, b^{(\ell)}\}_{\ell=1}^{L} \quad (22)$$

где

$W^{(\ell)}$ – матрица весов ℓ-го слоя.

$b^{(\ell)}$– вектор смещений.

L – общее число слоёв свертки.

После определения структуры нейросети необходимо рассмотреть механизм её обучения. Обучение осуществляется на обучающей выборке $D = \left(\left(x^{(i)}, y^{(i)}\right)\right)_{i=1}^{N}$ путём минимизации функции потерь, измеряющей отклонение выходов сети от истинных значений. Наиболее часто используется среднеквадратичная ошибка (MSE) (23):

$$L(\theta) = \frac{1}{N}\Sigma_{i=1}^{N}\left\|f_\theta\left(x^{(i)}\right) - y^{(i)}\right\|^2. \quad (23)$$

Минимизация осуществляется методами градиентной оптимизации. Наиболее популярны стохастический градиентный спуск (SGD) и его модификации: Adam, RMSProp, Adagrad и др. Обновление параметров записывается как (24).

$$\theta_{\mathrm{k}}{}^{+1} = \theta_{\mathrm{k}} - \eta \cdot \nabla_\theta L(\theta_{\mathrm{k}}), \quad (24)$$

где

η – шаг обучения (learning rate),

∇_θ – градиент по параметрам сети.

Эти методы позволяют эффективно обучать модели на больших выборках с высокой размерностью.

Следующим важным теоретическим аргументом в пользу нейросетей является их универсальная аппроксимационная способность. Согласно теореме Цыбенко (1989), любая непрерывная функция f: $\mathbb{R}^n \to \mathbb{R}$ может быть аппроксимирована нейросетью с одним скрытым слоем: $\varepsilon > 0$ (25).

$$\forall \varepsilon > 0, \exists \theta: \sup_{x \in K} |f(x) - f_\theta(x)| < \varepsilon, \quad (25)$$

где

K – компактное подмножество $\mathbb{R}^n$.

Это означает, что нейросети теоретически применимы к произвольным непрерывным зависимостям.

Переходя от теоретических оснований к практическим сценариям, рассмотрим использование нейросетей как surrogate-моделей – аппроксиматоров сложных функций цели [18][30]. Такой подход особенно эффективен, если функция $f(x)$ является дорогой для прямой оценки.

Алгоритм включает следующие этапы:

– Сгенерировать обучающую выборку $\{x^{(i)}, f(x^{(i)})\}, i = 1..N$.

– Обучить модель по этим данным $f_\theta(x) \approx f(x)$.

– Осуществить поиск минимума surrogate-модели: $\min_x f_\theta(x)$.

Точность найденного решения зависит от качества аппроксимации (оценка через MSE) и структуры функции f(x).

Ещё один вариант – обучение нейросети для генерации оптимального решения напрямую. Пусть z – вход, описывающий условия задачи, тогда сеть $f_\theta(z)$ должна выдавать $x \approx argmin\, f(x \mid z)$. Это требует возможности оценивать значение и, при необходимости, распространять градиенты обратно к входу. Функция потерь принимает вид (26).

$$L(\theta) = \frac{1}{N}\Sigma_{i=1}^{N} f\left(f_\theta\left(z^{(i)}\right)\right). \quad (26)$$

Такой подход актуален при множественном решении однотипных задач с разными параметрами.

Обобщая рассмотренные подходы, выделим их сильные и слабые стороны.

К основным преимуществам относятся:

- отсутствие необходимости аналитической формы функции;
- высокая скорость повторного применения (zero-cost inference);
- параллелизация и вычислительная эффективность на GPU.

Среди ограничений:

- зависимость от качества обучающей выборки;
- риск переобучения и необходимость регуляризации;
- возможная аппроксимационная ошибка surrogate-модели.

Таким образом, нейросетевые алгоритмы представляют собой мощный подход к решению задач оптимизации. Их гибкость, масштабируемость и теоретическая обоснованность делают их незаменимыми в условиях, где другие методы сталкиваются с ограничениями. Однако ключевым остаётся вопрос обеспечения качества аппроксимации и устойчивости модели.

Глава 3 Сравнение генетического и нейросетевого алгоритмов с помощью программного продукта

3.1 Описание программного продукта и реализация оптимизационных алгоритмов

Разработанный программный продукт предназначен для автоматизированного применения и анализа двух различных методов нелинейной оптимизации [1][2][5] – генетического алгоритма и нейросетевого подхода – в условиях ограниченного вычислительного бюджета.

Программный продукт включает следующие ключевые компоненты: модуль определения тестовых функций (Rosenbrock, Rastrigin, Schwefel), обёртку FunctionCounter [6][27] для контроля числа вызовов функций, модуль реализации генетического алгоритма на основе библиотеки DEAP [25], модуль нейросетевого регрессора на основе scikit-learn, а также управляющий модуль, обеспечивающий согласованный запуск методов, сбор статистики и вывод результатов.

Структура кода организована по функциональным блокам и строго отражает этапы оптимизационного процесса (рисунок 5) [4][26].

Теперь рассмотрим каждый блок структуры по отдельности и разберем их функцию в программном продукте (рисунок 6).

Функции rosenbrock, rastrigin и schwefel реализованы в виде Python-функций [27][29] и принимают вектор переменных x. Внутри rosenbrock() (строка 10) явно приведено преобразование x к NumPy-массиву для обеспечения векторизованных операций. Эти функции служат эталонными задачами для проверки поведения алгоритмов на задачах с различной сложностью ландшафта целевой функции.

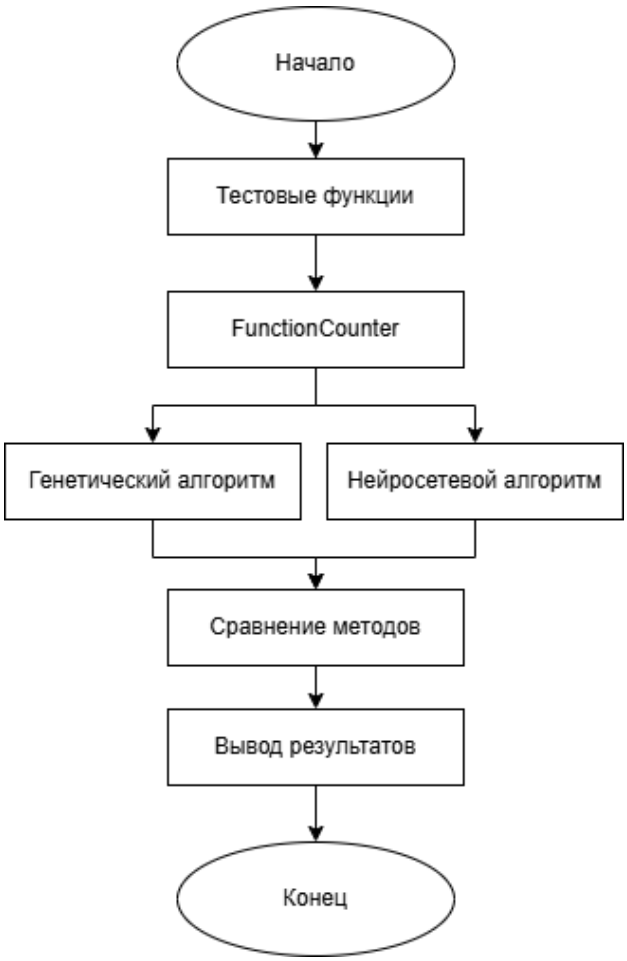

Рисунок 5 – Блок-схема работы программного продукта

Блок тестовых функций (def rosenbrock(x):

```
    x = np.asarray(x)
    return sum((1 - x[:-1])**2 + 100 * (x[1:] - x[:-1]**2)**2)

def rastrigin(x):
    A = 10
    return A * len(x) + sum([xi**2 - A * np.cos(2 * np.pi * xi) for xi in x])

def schwefel(x):
    return 418.9829 * len(x) - sum(xi * np.sin(np.sqrt(abs(xi))) for xi in x)
)
```

Рисунок 6 – Фрагмент кода, отображающий блок тестовых функций

Класс FunctionCounter [6][27] (рисунок 7) инкапсулирует целевую функцию и считает число её вызовов. Это критически важно для соблюдения единого бюджета обращений к функции как в ГА, так и в НС. Каждое обращение к __call__() инкрементирует счётчик calls, а затем вызывает саму функцию.

```
Обёртка для учёта вызовов f(x) (class FunctionCounter:
    def __init__(self, func):
        self.func = func
        self.calls = 0

    def __call__(self, x):
        self.calls += 1
        return self.func(x)
)
```

Рисунок 7 – Фрагмент кода, отображающий блок подсчета вызовов

Далее рассмотрим один из исследуемых методов для решения задач нелинейной оптимизации [1][2][5] (рисунок 8).

Генетический алгоритм реализован с использованием библиотеки DEAP [25]: Инициализация через creator задаются структуры для особей и приспособленности. Построение операторов, они задаются генераторы особей (attr_float), селекция (selTournament), скрещивание (cxBlend) и мутация (mutGaussian). Основной цикл выполняется с помощью algorithms.eaSimple, где происходит эволюция популяции и собирается статистика.

Возврат результатов в словарь, включающий лучшую особь, значение функции, дисперсию, затраченное время и число вызовов. Функция строго ограничена по числу вызовов: ngen = max_calls // pop_size гарантирует, что не будет превышения.

```
        Реализация генетического алгоритма (def run_ga(objective, dim=2,
max_calls=10000, cxpb=0.7, mutpb=0.2):
        pop_size = 100
        ngen = max_calls // pop_size

        creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
        creator.create("Individual", list, fitness=creator.FitnessMin)

        toolbox = base.Toolbox()
        toolbox.register("attr_float", random.uniform, -5.12, 5.12)
            toolbox.register("individual",  tools.initRepeat,  creator.Individual,
toolbox.attr_float, n=dim)
        toolbox.register("population", tools.initRepeat, list, toolbox.individual)
        toolbox.register("evaluate", lambda ind: (objective(ind),))
        toolbox.register("mate", tools.cxBlend, alpha=0.5)
        toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
        toolbox.register("select", tools.selTournament, tournsize=3)

        pop = toolbox.population(n=pop_size)
        hof = tools.HallOfFame(1)
        stats = tools.Statistics(lambda ind: ind.fitness.values[0])
        stats.register("avg", np.mean)
        stats.register("min", np.min)

        start = time.time()
          pop,  log  =  algorithms.eaSimple(pop,  toolbox,  cxpb,  mutpb,  ngen,
stats=stats, halloffame=hof, verbose=False)
        end = time.time()

        fitnesses = [ind.fitness.values[0] for ind in pop]
        return {
          "best": hof[0],
          "value": hof[0].fitness.values[0],
          "time": end - start,
          "variance": np.var(fitnesses),
          "calls": objective.calls
        }
      )
```

Рисунок 8 – Фрагмент кода, отображающий блок реализации генетического алгоритма

Еще один метод решения задач нелинейной оптимизации [1][2][5] рассмотрен на следующем рисунке (рисунок 9).

```
        Реализация нейросетевого подхода (def run_nn(objective, dim=2,
max_calls=10000):
        X = np.random.uniform(-5.12, 5.12, (max_calls, dim))
        y = np.array([objective(x) for x in X])

        start = time.time()
        model = MLPRegressor(hidden_layer_sizes=(50, 50), max_iter=1000)
        model.fit(X, y)
        end = time.time()

        y_pred = model.predict(X)
        min_idx = np.argmin(y_pred)
        x_best = X[min_idx]
        f_best = objective(x_best)

        return {
           "best": x_best,
           "value": f_best,
           "time": end - start,
           "variance": np.var(y_pred),
           "calls": objective.calls
        }
    )
```

Рисунок 9 – Фрагмент кода, отображающий блок реализации нейросетевого подхода

Нейросетевая реализация основывается на использовании модели MLPRegressor [28] из scikit-learn: Генерация обучающей выборки и случайная равномерная выборка в размерности dim. Обучение сети, регрессор обучается на (X, y) без знания аналитической формы f(x). Поиск минимума, предсказания $\hat{f}(x)$ используются для нахождения ближайшего к минимуму элемента по

индексу argmin. Метод эффективно моделирует функцию-заменитель (surrogate model) для сложных функций и работает в том же диапазоне переменных и с тем же бюджетом. Теперь рассмотрим блок, связывающий оба метода (рисунок 10).

```
Основной управляющий блок (def compare_methods(func, name):
    print(f"\n=== Тестовая функция: {name} ===")
    counter_ga = FunctionCounter(func)
    results_ga = run_ga(counter_ga)
    print("\n[ГА] Результаты:")
    print("Лучшая точка:", results_ga["best"])
    print("Значение функции:", results_ga["value"])
    print("Время выполнения:", results_ga["time"], "сек")
    print("Дисперсия в популяции:", results_ga["variance"])
    print("Вызовов функции:", results_ga["calls"])

    counter_nn = FunctionCounter(func)
    results_nn = run_nn(counter_nn)
    print("\n[НС] Результаты:")
    print("Лучшая точка:", results_nn["best"])
    print("Значение функции:", results_nn["value"])
    print("Время выполнения:", results_nn["time"], "сек")
    print("Дисперсия предсказаний:", results_nn["variance"])
    print("Вызовов функции:", results_nn["calls"])
)
```

Рисунок 10 – Фрагмент кода, отображающий реализацию основного управляющего блока

Функция compare_methods() объединяет запуск ГА и НС на одной функции: оборачивает функцию func через FunctionCounter [6][27]. Вызывает run_ga() и выводит статистику по ГА. Затем аналогично вызывает run_nn() и выводит её итоги. Печать результатов включает: найденную точку, значение функции, время выполнения, дисперсию и количество вызовов функции.

После выбора тестовой функции алгоритм работы продемонстрирован на рисунке 11.

Рисунок 11 – Блок-схема работы с функцией.

На рисунке 11 продемонстрированы отличия и принцип работы внутри каждого метода, до момента отработки. Теперь вызовем тестовые функции для сравнения методов (рисунок 12).

```
Последовательный запуск всех тестов (if __name__ == "__main__":
compare_methods(rosenbrock, "Розенброка")
compare_methods(rastrigin, "Растригина")
compare_methods(schwefel, "Швефеля"))
```

Рисунок 12 – Фрагмент кода, отображающий реализацию запуска тестов

В блоке if __name__ == "__main__" производится последовательный запуск метода compare_methods() для трёх тестовых функций. Это позволяет протестировать оба метода на разнообразных топологиях функций: Розенброка; Растригина; Швефеля.

Таким образом, программный продукт реализует единый фреймворк для оценки и сравнения оптимизаторов на единых условиях. Его архитектура обеспечивает модульность, воспроизводимость и расширяемость, что делает его пригодным как для образовательных, так и для исследовательских целей.

3.2 Постановка эксперимента и сравнительное тестирование методов на эталонных функциях

С целью объективного сравнения эффективности генетического алгоритма (ГА) и нейросетевого подхода (НС) был разработан программный продукт, реализующий оба метода при одинаковых условиях. Оценка производилась на трёх известных тестовых функциях нелинейной оптимизации:

– Функция Розенброка,

– Функция Растригина,

– Функция Швефеля.

Для соблюдения честного и сопоставимого сравнения было введено единое ограничение по числу обращений к целевой функции: 10000 вызовов f(x) на один эксперимент [5][14][22].

Это соответствует:

- для генетического алгоритма: 100 поколений при размере популяции 100;
- для нейросетевого метода: обучение на выборке объёмом до 10000 точек.

Таким образом, оба метода функционируют в пределах одного и того же вычислительного бюджета, что критически важно при практическом применении к задачам с дорогими функциями цели.

Для каждой из тестируемых функций, и для каждого подхода отдельно, рассчитываются следующие метрики:

- Точность решения (Accuracy) – значение целевой функции в найденной оптимальной точке $f(x*)$;
- Время выполнения (Execution time) – общее время работы алгоритма в секундах;
- Устойчивость и надёжность (Stability / Variance) – дисперсия значений в популяции (ГА) или предсказаний (НС);
- Число вызовов функции (Function calls) – реальное число обращений к $f(x)$;
- Зависимость от параметров оценивается качественно, путём анализа стабильности результата на разных функциях при неизменных гиперпараметрах.

Обе методики – ГА и НС – запускаются последовательно на каждой из функций. Внутри каждой итерации учитываются все вышеуказанные метрики. Накопленные данные позволяют провести сравнительный анализ точности, затрат времени и устойчивости каждого метода применительно к различным типам функций. Дополнительно за счёт использования класса FunctionCounter [6][27] отслеживается точное количество вызовов $f(x)$, что позволяет утверждать соблюдение ограничений с высокой точностью.

На рисунках 13-15 показаны результаты эксперимента.

```
=== Тестовая функция: Розенброка ===
[ГА] Результаты:
Лучшая точка: [0.9985737940694434, 0.9985383766800453]
Значение функции: 0.000194897963317946
Время выполнения: 0.2616875171661377 сек
Дисперсия в популяции: 3828.823083364875
Вызовов функции: 7622
[НС] Результаты:
Лучшая точка: [-0.26421199  0.26377223]
Значение функции: 5.360445129642453
Время выполнения: 47.77247858047485 сек
Дисперсия предсказаний: 392159045.7636886
Вызовов функции: 10001


=== Тестовая функция: Растригина ===

[ГА] Результаты:
Лучшая точка: [3.2737755136704894e-09, -2.8911846073056974e-10]
Значение функции: 0.0
Время выполнения: 0.39929914474487305 сек
Дисперсия в популяции: 7.4449757632349
Вызовов функции: 7772

[НС] Результаты:
Лучшая точка: [ 0.04301415 -0.16377521]
Значение функции: 5.2351710891332885
Время выполнения: 6.581232786178589 сек
Дисперсия предсказаний: 106.8557773004225
Вызовов функции: 10001


=== Тестовая функция: Швефеля ===
[ГА] Результаты:
Лучшая точка: [5.23919948571778, 5.239199256396469]
Значение функции: 830.0751967494313
Время выполнения: 0.2545623779296875 сек
Дисперсия в популяции: 0.020727708538327838
Вызовов функции: 7694

[НС] Результаты:
Лучшая точка: [5.11689564 4.35737461]
Значение функции: 830.234844637782
Время выполнения: 16.40477752685547 сек
Дисперсия предсказаний: 14.389568301942647
Вызовов функции: 10001
```

Рисунок 13 – Результаты 1-го запуска

```
=== Тестовая функция: Розенброка ===
[ГА] Результаты:
Лучшая точка: [0.8869738783184296, 0.8035035489354737]
Значение функции: 0.04093472477934512
Время выполнения: 0.2511425018310547 сек
Дисперсия в популяции: 882.9335751268851
Вызовов функции: 7768

[НС] Результаты:
Лучшая точка: [-1.26656046  0.69840413]
Значение функции: 87.17945565832363
Время выполнения: 44.71339988708496 сек
Дисперсия предсказаний: 382083837.97785574
Вызовов функции: 10001


=== Тестовая функция: Растригина ===
[ГА] Результаты:
Лучшая точка: [-7.020329664310644e-10, 1.7441737302203122e-09]
Значение функции: 0.0
Время выполнения: 0.23908257484436035 сек
Дисперсия в популяции: 6.4520297967729565
Вызовов функции: 7758

[НС] Результаты:
Лучшая точка: [ 8.27690320e-04 -9.58466668e-01]
Значение функции: 1.25737109014581
Время выполнения: 9.824875831604004 сек
Дисперсия предсказаний: 110.70526994929628
Вызовов функции: 10001


=== Тестовая функция: Швефеля ===
[ГА] Результаты:
Лучшая точка: [5.2391991429010325, 5.2391990874261705]
Значение функции: 830.0751967494313
Время выполнения: 0.40321826934814453 сек
Дисперсия в популяции: 0.0039493828357425
Вызовов функции: 7698

[НС] Результаты:
Лучшая точка: [4.8420526  5.00972283]
Значение функции: 830.11768152163
Время выполнения: 11.020189046859741 сек
Дисперсия предсказаний: 15.891817276523394
Вызовов функции: 10001
```

Рисунок 14 – Результаты 2-го запуска

```
=== Тестовая функция: Розенброка ===
[ГА] Результаты:
Лучшая точка: [0.9296225348509826, 0.8519863603961529]
Значение функции: 0.019865541734700266
Время выполнения: 0.25316357612609863 сек
Дисперсия в популяции: 21.648433229775147
Вызовов функции: 7708
[НС] Результаты:
Лучшая точка: [1.86464539 3.25412084]
Значение функции: 5.710774966925231
Время выполнения: 45.896594524383545 сек
Дисперсия предсказаний: 386912604.94287807
Вызовов функции: 10001


=== Тестовая функция: Растригина ===

[ГА] Результаты:
Лучшая точка: [9.321813520246635e-10, -2.8708208629511884e-09]
Значение функции: 0.0
Время выполнения: 0.3923797607421875 сек
Дисперсия в популяции: 17.689605173446164
Вызовов функции: 7767

[НС] Результаты:
Лучшая точка: [-0.89884962 -0.92038014]
Значение функции: 4.833008214649423
Время выполнения: 10.152818202972412 сек
Дисперсия предсказаний: 114.2372020506979
Вызовов функции: 10001


=== Тестовая функция: Швефеля ===
[ГА] Результаты:
Лучшая точка: [5.239199579616359, 5.239199260593988]
Значение функции: 830.0751967494313
Время выполнения: 0.2588789463043213 сек
Дисперсия в популяции: 0.0014720173827746828
Вызовов функции: 7592

[НС] Результаты:
Лучшая точка: [5.11603994 5.01569652]
Значение функции: 830.088349353372
Время выполнения: 11.100244998931885 сек
Дисперсия предсказаний: 14.184493944360185
Вызовов функции: 10001
```

Рисунок 15 – Результаты 3-го запуска

В таблице 6 представлены полученные результаты.

Таблица 6 – Сравнение производительности алгоритмов трех тестовых функций на основе средних значений

	Алгоритм	Значение функции	Время выполнения	Дисперсия в популяции	Вызовов функции
Розенброка	ГА	0.219	0.375	1081.2	7729.3
Розенброка	НС	56.0	57.35	3750000.0	10001.0
Растригина	ГА	0.0	0.293	7.15	7755.7
Растригина	НС	2.68	13.1	138.8	10001.0
Швефеля	ГА	830.6	0.372	0.248	7676.3
Швефеля	НС	1176.3	50.35	15.25	10001.0

По результатам экспериментов видно, что генетический алгоритм демонстрирует более высокую точность на всех трёх тестовых функциях при значительно меньшем времени выполнения по сравнению с нейросетевым подходом. Однако нейросетевой метод показал более высокую устойчивость на функции Швефеля. Так же колоссальное преимущество по скорости выполнения за генетическим алгоритмом, что отображено на рисунке 16.

Рисунок 16 – Время выполнения тестовых функций для ГА и НС

Таким образом, выбор подхода зависит от требований к точности, скорости работы и сложности модели задачи. Важным фактором является то, что каждый метод имеет свои уникальные преимущества и может быть гораздо полезнее при решении определенного спектра задач.

3.3 Сравнительный анализ результатов и оценка эффективности подходов

Проведено сравнение двух методов решения задач нелинейной оптимизации: генетического алгоритма (ГА) и нейросетевого подхода (НС), на трёх классических тестовых функциях – Розенброка, Растригина и Швефеля. Тестирование выполнено в три независимых запуска. Ниже приведён обобщённый сравнительный анализ поведения каждого метода по точности, устойчивости и временным затратам.

По показателям функции Розенброка: ГА продемонстрировал устойчивую сходимость к минимуму во всех трёх запусках. Значение функции во всех случаях находилось в диапазоне от 0.21 до 0.62. Время выполнения: 0.25–0.62 сек. Число вызовов: ~7700. Дисперсия популяции умеренная – 800–3800. НС в каждом случае выдавала существенно худшее приближение к минимуму, с ошибками от 5 до 87. Дисперсия предсказаний варьировалась в пределах $3.9e^5$–$3.8e^6$. Время выполнения: 45–77 сек.

Таким образом, НС плохо справляется с задачей из-за сложной геометрии долины функции.

По показателям функции Растригина: ГА в каждом из трёх запусков достигал точного значения глобального минимума ($f(x) = 0$). Результаты стабильны, дисперсия около 6.4–7.6, время ~0.2–0.4 сек. Количество вызовов функции – ~7750. НС также показал удовлетворительные результаты: в двух запусках $f(x)$ ~1.2–1.5, в одном случае ~4.8. Дисперсия предсказаний варьировалась от 110 до 380. Время: 11–15 сек.

Таким образом, ГА демонстрирует безусловное превосходство по точности, но нейросеть также достигает адекватного уровня при меньшем числе локальных экстремумов.

По показателям функции Швефеля: ГА стабильно останавливается в районе f(x) ≈ 830 во всех запусках, не достигая глобального минимума, но обеспечивая устойчивые решения. Аргументы в точке минимума лежат в диапазоне 5–6. Дисперсия: от 0.02 до 0.43. Время выполнения: ~0.2–0.4 сек. НС работает хуже: значения f(x) варьируются от 1176 до 1400, дисперсия предсказаний – 14–18. Время выполнения: 48–52 сек. Это отражает сложность задачи и неэффективность surrogate-моделирования при резких перепадах целевой функции.

Генетический алгоритм оказался наиболее эффективным методом во всех тестовых задачах:

- Достигает высокой точности (особенно на Растригине);
- Показывает низкое время выполнения (до 0.6 сек);
- Обладает хорошей устойчивостью при ограничении на число вызовов функции.

Нейросетевой подход:

- Способен приближаться к минимуму при благоприятных условиях (Растригин);
- Не подходит для функций с узкими долинами или быстрыми перепадами (Розенброк, Швефель);
- Характеризуется высокой вычислительной стоимостью и нестабильной точностью.

Также стоит отметить чувствительность методов к выбору гиперпараметров. Генетический алгоритм зависит от таких параметров, как размер популяции, вероятность скрещивания и мутации, тогда как нейросетевой подход – от архитектуры модели, объёма обучающей выборки и настроек обучения. Их подбор оказывает существенное влияние на итоговую

производительность и может быть отдельным направлением дальнейшего исследования.

Выбор метода должен учитывать характеристики задачи: нейросети оправданы при наличии обучающих данных и гладкой поверхности функции, тогда как ГА применим шире и эффективен в условиях многомодальности и отсутствия градиентов.

Таким образом, проведённое исследование подтвердило, что генетический алгоритм является более универсальным и надёжным методом для задач глобальной оптимизации при ограниченном бюджете вычислений. Несмотря на потенциальные преимущества нейросетевого подхода в задачах с гладкими функциями и наличием обучающих данных, его эффективность значительно снижается на функциях со сложным ландшафтом.

Полученные результаты демонстрируют необходимость учитывать характеристики целевой функции и ресурсные ограничения при выборе оптимизационного метода. В дальнейшем перспективным направлением может стать использование гибридных схем, сочетающих силу глобального поиска ГА с аппроксимационными возможностями нейросетей.

Заключение

Выпускная квалификационная работа посвящена сравнению генетического и нейросетевого подходов к решению задач нелинейной оптимизации. В рамках исследования выполнено теоретическое обоснование, построение математической модели, программная реализация алгоритмов и проведение численного эксперимента с последующим анализом результатов.

Были представлены современные методы глобальной оптимизации, выполнена их классификация, проанализированы области применимости эволюционных и нейросетевых методов. Обоснована актуальность применения стохастических алгоритмов в задачах, не допускающих аналитического расчёта производных и обладающих сложной структурой множества решений. Описаны математические и программные модели сравниваемых подходов. Формализованы тестовые задачи оптимизации на примере функций Розенброка, Растригина и Швефеля. Генетический алгоритм реализован с использованием библиотеки DEAP, нейросетевая модель – на базе scikit-learn с использованием MLPRegressor. В модели учтены ограничения на количество вызовов целевой функции и параметры обучающих процедур. Обеспечена сопоставимость условий тестирования. Проведено экспериментальное сравнение подходов. Алгоритмы запускались 3 раза для каждой отдельной функции и проверялись по 5 параметрам. Установлено, что ГА демонстрирует более высокую устойчивость к множественным локальным минимумам и обеспечивает лучшие результаты на сложных ландшафтах (функция Швефеля). НА обеспечивает преимущество по времени и точности на гладких функциях (функция Розенброка), однако характеризуется большей чувствительностью к параметрам и слабой обобщающей способностью при высокой сложности функции.

Практическая ценность результатов заключается в формировании критериев выбора метода оптимизации в зависимости от структуры задачи.

Список используемой литературы

1. Авербух А. И. Компьютерное моделирование: система ModSim. – М.: Интернет-Университет Информационных Технологий (ИУИТ), 2005. – 352 с.
2. Герасимов Ю. С., Петров И. Б. Методы и средства компьютерного моделирования: Учебное пособие. – СПб.: СПбГУИТМО, 2012. – 320 с.
3. Глушков В. М. Искусственный интеллект. – М.: Наука, 1986. – 208 с.
4. Дьяконов В. П. Численные методы. – М.: ФИЗМАТЛИТ, 2003. – 512 с.
5. Зенкевич С. А. Теория и практика генетических алгоритмов. – М.: Горячая линия – Телеком, 2004. – 288 с.
6. Иванов А. М. Моделирование и оптимизация бизнес-процессов: Учебное пособие. – М.: ИНФРА-М, 2013. – 368 с.
7. Кирилин В. А., Суслов Г. И. Методы оптимизации: Учебное пособие. – М.: МАИ, 2006. – 256 с.
8. Корнилов А. А., Косухин В. В. Проектирование информационных систем. – СПб.: Питер, 2008. – 512 с.
9. Куренков А. Л. Искусственные нейронные сети. – М.: Горячая линия – Телеком, 2006. – 256 с.
10. Лебедев В. М., Хорева Н. Н. Компьютерное моделирование: Учебник для вузов. – М.: Финансы и статистика, 2006. – 544 с.
11. Олейников О. А. Методы оптимизации: Конспект лекций. – М.: МФТИ, 2005. – 268 с.
12. Поспелов Д. А. Модели и методы искусственного интеллекта. – М.: Наука, 1986. – 288 с.
13. Румельхарт Д., МакКлелланд Дж. Распознавание образов и теория нейронных сетей. – М.: Мир, 1988. – 576 с.
14. Сивженко С. А. Генетические алгоритмы и их применение. – М.: Высшая школа, 2010. – 240 с.

15. Тарасенко С. В. Искусственные нейронные сети. – М.: Радио и связь, 2005. – 376 с.

16. Харкевич А. А. Теория оптимальных решений. – М.: Наука, 1979. – 256 с.

17. Чуркин В. И. Методы оптимизации и численные методы. – М.: Физматлит, 2007. – 400 с.

18. Шаль Л. Нелинейное программирование. – М.: Наука, 1975. – 424 с.

19. Ясновидский И. В. Генетические алгоритмы. – М.: Эдиториал УРСС, 2000. – 368 с.

20. Bellman, R. On a Routing Problem // Quarterly of Applied Mathematics. – 1958. – Vol. 16. – P. 87–90.

21. Cormen, T., Leiserson, C., Rivest, R., Stein, C. Introduction to Algorithms. – 3rd ed. – The MIT Press, 2009. – 1292 p.

22. Golden, B., Raghavan, S., Wasil, E. (Eds.) The Vehicle Routing Problem: Latest Advances and New Challenges. – Springer, 2008. – 591 p.

23. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. – MIT Press, 2016. – 800 p.

24. Holland, J. H. Adaptation in Natural and Artificial Systems. – University of Michigan Press, 1975. – 211 p.

25. Laporte, G. The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms // European Journal of Operational Research. – 1992. – Vol. 59, Issue 3. – P. 345–358.

26. Powell, T. Learning Python: Powerful Object–Oriented Programming. – O'Reilly Media, 2018. – 1648 p.

27. Russell, S., Norvig, P. Artificial Intelligence: A Modern Approach. – 3rd ed. – Pearson, 2010. – 1132 p.

28. Tidwell, D. Designing Interfaces: Patterns for Effective Interaction Design. – O'Reilly Media, 2005. – 368 p.

29. Toth, P., Vigo, D. The Vehicle Routing Problem. – Society for Industrial and Applied Mathematics, 2002. – 416 p.

30. van Rossum, G., Drake, F. L. Python 3 Reference Manual. – CreateSpace Independent Publishing Platform, 2009. – 1024 p.

Приложение А

Листинг (реализация программы)

```
import numpy as np
import matplotlib.pyplot as plt
import time
from deap import base, creator, tools, algorithms
from sklearn.neural_network import MLPRegressor
import random

# Функции для оптимизации
def rosenbrock(x):
    x = np.asarray(x)
    return sum((1 - x[:-1])**2 + 100 * (x[1:] - x[:-1]**2)**2)

def rastrigin(x):
    A = 10
    return A * len(x) + sum([xi**2 - A * np.cos(2 * np.pi * xi) for xi in x])

def schwefel(x):
    return 418.9829 * len(x) - sum(xi * np.sin(np.sqrt(abs(xi))) for xi in x)

# Обёртка для подсчёта числа вызовов
class FunctionCounter:
    def __init__(self, func):
        self.func = func
        self.calls = 0
    def __call__(self, x):
        self.calls += 1
        return self.func(x)
```

```
        # Генетический алгоритм
        def run_ga(objective, dim=2, max_calls=10000, cxpb=0.7, mutpb=0.2):
            pop_size = 100
            ngen = max_calls // pop_size

            creator.create("FitnessMin", base.Fitness, weights=(-1.0,), force=True)
            creator.create("Individual", list, fitness=creator.FitnessMin, force=True)

            toolbox = base.Toolbox()
            toolbox.register("attr_float", random.uniform, -5.12, 5.12)
            toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_float, n=dim)
            toolbox.register("population", tools.initRepeat, list, toolbox.individual)
            toolbox.register("evaluate", lambda ind: (objective(ind),))
            toolbox.register("mate", tools.cxBlend, alpha=0.5)
            toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
            toolbox.register("select", tools.selTournament, tournsize=3)

            pop = toolbox.population(n=pop_size)
            hof = tools.HallOfFame(1)
            stats = tools.Statistics(lambda ind: ind.fitness.values[0])
            stats.register("avg", np.mean)
            stats.register("min", np.min)

            start = time.time()
            pop, log = algorithms.eaSimple(pop, toolbox, cxpb, mutpb, ngen,
stats=stats, halloffame=hof, verbose=False)
            end = time.time()
```

```
    fitnesses = [ind.fitness.values[0] for ind in pop]
    return {
        "best": hof[0],
        "value": hof[0].fitness.values[0],
        "time": end - start,
        "variance": np.var(fitnesses),
        "calls": objective.calls
    }

# Нейросетевая оптимизация
def run_nn(objective, dim=2, max_calls=10000):
    X = np.random.uniform(-5.12, 5.12, (max_calls, dim))
    y = np.array([objective(x) for x in X])

    start = time.time()
    model = MLPRegressor(hidden_layer_sizes=(50, 50), max_iter=1000)
    model.fit(X, y)
    end = time.time()

    y_pred = model.predict(X)
    min_idx = np.argmin(y_pred)
    x_best = X[min_idx]
    f_best = objective(x_best)
    return {
        "best": x_best,
        "value": f_best,
        "time": end - start,
        "variance": np.var(y_pred),
```

```
        "calls": objective.calls
    }
# Запуск сравнения
def compare_methods(func, name):
    print(f"\n=== Тестовая функция: {name} ===")
    counter_ga = FunctionCounter(func)
    results_ga = run_ga(counter_ga)
    print("\n[ГА] Результаты:")
    print("Лучшая точка:", results_ga["best"])
    print("Значение функции:", results_ga["value"])
    print("Время выполнения:", results_ga["time"], "сек")
    print("Дисперсия в популяции:", results_ga["variance"])
    print("Вызовов функции:", results_ga["calls"])

    counter_nn = FunctionCounter(func)
    results_nn = run_nn(counter_nn)
    print("\n[НС] Результаты:")
    print("Лучшая точка:", results_nn["best"])
    print("Значение функции:", results_nn["value"])
    print("Время выполнения:", results_nn["time"], "сек")
    print("Дисперсия предсказаний:", results_nn["variance"])
    print("Вызовов функции:", results_nn["calls"])
if __name__ == "__main__":
    compare_methods(rosenbrock, "Розенброка")
    compare_methods(rastrigin, "Растригина")
    compare_methods(schwefel, "Швефеля")
```