МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество «Казахский национальный исследовательский технический университет имени К.И.Сатпаева»

Институт автоматики и информационных технологий

Кафедра «Высшая математика и моделирование»

Сериков Наиль Бауржанович

Решение обратной задачи колебания с использованием нейронных сетей на невесомом квантовом графе типа «звезда»

ДИПЛОМНАЯ РАБОТА

6В06103 – «Математическое и компьютерное моделирование»

Алматы 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество «Казахский национальный исследовательский технический университет имени К.И.Сатпаева»

Институт автоматики и информационных технологий

Кафедра «Высшей математики и моделирования»

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ВМиМ
канд. физ-мат. наук, доцент
_____Тулешева Г. А.
«30» 05 2024 г.

ДИПЛОМНАЯ РАБОТА

На тему: «Решение обратной задачи колебания с использованием нейронных сетей на невесомом квантовом графе типа “звезда”»

по специальности 6В06103 – Математическое и компьютерное моделирование

Выполнил Н.Б. Сериков

Рецензент
PhD, и.о. ассоциированного профессора
факультета математики, физики и
информатики, кафедра математики и
математического моделирования
КазНПУ им. Абая
_____К.М. Шияпов
"28" 05 2024 г.

Научный руководитель
канд. физ-мат. наук
_____ Р.Н. Зимин
"30" мая 2024 г.

Алматы 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РЕСПУБЛИКИ КАЗАХСТАН

Некоммерческое акционерное общество «Казахский национальный исследовательский технический университет имени К.И.Сатпаева»

Институт автоматики и информационных технологий

Кафедра «Высшая математика и моделирование»

6В06103 – «Математическое и компьютерное моделирование»

УТВЕРЖДАЮ
Заведующая кафедрой ВМиМ
канд. физ-мат. наук, доцент
_____ Г.А.Тулешева
«30» 05 2024 г.

ЗАДАНИЕ
на выполнение дипломной работы

Обучающемуся: Сериков Наиль Бауржанович
Тема: «Решение обратной задачи колебания с использованием нейронных сетей на невесомом квантовом графе типа «звезда»»
Утверждена приказом проректора по академической работе: № 548-П/О
от «04» декабря 2024 г.
Срок сдачи законченной работы: «28» мая 2024 г.
Исходные данные к дипломной работе: техническая документация к C++, Tensorflow и Python
Краткое содержание дипломной работы:
а) теоретическая часть;
б) математическая постановка прямой задачи;
в) создание базы данных и предварительная обработка;
г) выбор архитектуры нейронной сети.
Рекомендуемая основная литература: техническая литература по библиотеке TensorFlow, Python и C++, литература, посвященная механическим колебаниям и нейронным сетям

ГРАФИК

подготовки дипломной работы

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю и консультантам	Примечание
1. Теоретическая часть	15.02.24	Выполнено
2. Математическая постановка прямой задачи	27.02.24	Выполнено
3. Создание базы данных и предварительная обработка	20.03.24	Выполнено
4. Выбор архитектуры нейронной сети	18.04.24	Выполнено

Подписи

консультантов и нормоконтролера на законченную дипломный проект
с указанием относящихся к ним разделов проекта

Наименования разделов	Консультанты, И.О.Ф. (уч. степень, звание)	Дата подписания	Подпись
Теоретическая часть	канд. физ.-мат. наук Р.Н. Зимин	15.02.24	
Математическая постановка прямой задачи	канд. физ.-мат. наук Р.Н. Зимин	27.02.24	
Создание базы данных и предварительная обработка	канд. физ.-мат. наук Р.Н. Зимин	20.03.24	
Выбор архитектуры нейронной сети	канд. физ.-мат. наук Р.Н. Зимин	18.04.24	
Нормоконтролер	канд. физ.-мат. наук, Ж.Ж. Шатманов	29.05.2024	

Научный руководитель _____ Р.Н. Зимин

Задание принял к исполнению обучающийся _____ Н.Б. Сериков

Дата «20» января 2024 г

НЕКОММЕРЧЕСКОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО «КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени К.И.САТПАЕВА»

РЕЦЕНЗИЯ

на дипломную работу
Серикова Наиля Бауржановича
6В06103 – Математическое и компьютерное моделирование

На тему: «Решение обратной задачи колебания с использованием нейронных сетей на невесомом квантовом графе типа «звезда»»

ЗАМЕЧАНИЯ К РАБОТЕ

Перед дипломантом стояла задача решения обратная задача для колебания сети, представляющей собой невесомый квантовый граф типа «звезда», соединенных в свободном узле и закрепленных на концах, с использованием нейронных сетей. Дипломная работа состоит из Теоретической части, Основного результата, Заключения, Списка литературы и Приложений.

В качестве основных этапов в ходе решения задачи стоит отметить следующие: математическую постановку прямой задачи колебания невесомого квантового графа типа «звезда», решение прямой задачи и на основании ее решения, построение базы для тренировки нейронных сетей, выбор архитектуры для нейронной сети, а так же ее обучение. Таблицы данных были сгенерированы с использование двух языков программирования Python и C++, при этом учитывались преимущества каждого из них. Была выполнена нормализация данных разными методами для лучшего обучения нейронных сетей.

В ходе решения обратной задачи методами машинного обучения было принято решение построения 2 нейронных сетей, одна искала точку падения и неизвестную точечную массу, а вторая на значениях отклонения нитей на концах, предсказывала номер ребра. Был проведен анализ различных архитектур нейронных сетей в ходе которого выбралась наиболее точная. Проведен анализ полученных результатов. Было отмечено что для второй задачи, определения номера ребра, нейронные сети не так хорошо справляются с задачей. Был построен алгоритм без использования нейронных сетей для решения данной задачи. В работе приведены различные графики, репрезентующие работу нейронных сетей в решении обратной задачи.

Оценка работы

Считаю, что дипломная работа заслуживает оценки «95%», а при успешной защите студент Серикова Н. Б. заслуживает присвоения степени бакалавра по специальности 6В06103 – Математическое и компьютерное моделирование.

Рецензент
И.о. ассоциированного профессора факультета математики, физики и информатики, кафедра математики и математического моделирования, КазНПУ им. Абая, PhD
(должность, уч. степень, звание)
_____ К.М. Шияпов
(подпись)
«30» колы 2023 г.

КАДР САЯСАТЫ БАСҚАРМАСЫНЫҢ ЖЕТЕКШІСІ
УПРАВЛЕНИЯ КАДРОВОЙ ПОЛИТИ
НАО «КазНПУ им. АБ
ПОДПИСЬ

НЕКОММЕРЧЕСКОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО «КАЗАХСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени К.И.САТПАЕВА»

ОТЗЫВ

НАУЧНОГО РУКОВОДИТЕЛЯ

на дипломную работу
Серикова Н. Б.
6В06103 — Математическое и компьютерное моделирование

Тема: Решение обратной задачи колебания с использованием нейронных сетей на невесомом квантовом графе типа «звезда»

Перед Сериковым Н. Б. стояла следующая задача. Имеется пятилучевой квантовый граф (типа «звезда») с закрепленными концами, при этом предполагается, что нити невесомые (не являются струнами). На сеть падает материальная точка с неизвестной массой. Требуется определить нить, на которую упала материальная точка, место падения и массу материальной точки. В качестве инструмента решения было предложено использовать нейронные сети.

Была сформулирована математическая модель. Для обучения искусственной нейронной сети была создана база данных на основе решения параметрического семейства прямых задач колебания сети из нитей. Поскольку в ответе требуется получить данные разного типа — непрерывные (масса и точка падения) и дискретные (номер нити), то, решение задачи искали две нейронные сети: одна определяла массу и точку падения, вторая – искала номер нити, на которую упала точечная масса. Искусственные нейронные сети достаточно хорошо справились с нахождением непрерывных величин и относительно плохо — с нахождением номера нити. Поэтому для нахождения номера нити был построен прямой алгоритм поиска.

Дипломная работа Серикова Наиля Бауржановича состоит трех основных частей: теоретической части, основного результата и заключения.

Считаю, что Сериков Н. Б. справился с поставленной ему задачей, его дипломная работа соответствует выдвигаемым к таким работам требованиям. Считаю, что дипломная работа заслуживает оценки «95», а Сериков Н. Б. — присуждения ему академической степени бакалавра по специальности 6В06103 — «Математическое и компьютерное моделирование».

Научный руководитель
Ассоц. профессор кафедры ВМиМ,
к.ф.-м.н.
(должность, уч. степень, звание)
Зимин Р.Н.
(подпись)
«30» мая 2024 г.

Протокол

о проверке на наличие неавторизованных заимствований (плагиата)

Автор: Сериков Наиль Бауржанович

Соавтор (если имеется):

Тип работы: Дипломная работа

Название работы: Решение обратной задачи колебания с использованием НС на НК графе типа «звезда»

Научный руководитель: Решат Зимин

Коэффициент Подобия 1: 5.9

Коэффициент Подобия 2: 2.2

Микропробелы: 3

Знаки из здругих алфавитов: 0

Интервалы: 0

Белые Знаки: 0

После проверки Отчета Подобия было сделано следующее заключение:

☒ Заимствования, выявленные в работе, является законным и не является плагиатом. Уровень подобия не превышает допустимого предела. Таким образом работа независима и принимается.

☐ Заимствование не является плагиатом, но превышено пороговое значение уровня подобия. Таким образом работа возвращается на доработку.

☐ Выявлены заимствования и плагиат или преднамеренные текстовые искажения (манипуляции), как предполагаемые попытки укрытия плагиата, которые делают работу противоречащей требованиям приложения 5 приказа 595 МОН РК, закону об авторских и смежных правах РК, а также кодексу этики и процедурам. Таким образом работа не принимается.

☐ Обоснование:

Дата *Заведующий кафедрой*

АҢДАТПА

Дипломдық жұмыс ұштарында бекітілген бес бұрышты жұлдыз түріндегі салмақсыз кванттық графтың тербелісіне кері есептің шешімін қарастырады. Жұмыстың мақсаты – «жұлдыз» типті салмақсыз кванттық графикте нейрондық желілерді қолдану арқылы кері тербеліс есебін шешу. Кері есепті шешу үшін нейрондық желілер пайдаланылды. Оларды жаттықтыру үшін графтың ұшындағы табиғи жиілік пен ауытқу бұрыштары кіріске беріліп, соққы нүктесі, массасы және нүкте массасы құлаған жиектің нөмірі шығарылды. Айта кету керек, бастапқы мәселені шешу үшін екі нейрондық желі қолданылды, олардың біреуі әсер ету нүктесі мен массаны, ал екіншісі қабырға санын анықтады.

Нәтижесінде кванттық график бойынша тікелей есеп құрастырылды, математикалық модель құрастырылды, тікелей есептің шешімдерінің деректер базасы генерацияланды және әртүрлі архитектурасы бар нейрондық желілер салынды, олар бұрын жасалған мәліметтер базасында оқытылды. Көрсеткіштерге талдау жүргізілді, нәтижесінде дәлдіктің ең жоғары пайызы бар нейрондық желі архитектурасы таңдалды.

АННОТАЦИЯ

В дипломной работе исследуется решение обратной задачи колебания невесомого квантового графа в виде пятиконечной звезды, закреплённого на концах. Цель работы - решение обратной задачи колебания, с использованием нейронных сетей, на невесомом квантовом графе типа «звезда». Для решения обратной задачи использовались нейронные сети. Для их обучения на вход подавались собственная частота и углы отклонения на концах графа, а на выходе выводились точка падения, масса и номер ребра, на который падала точечная масса. Стоит отметить, что для решения исходной задачи использовались две нейронные сети, одна из которых определяла точку падения и массу, а другая - номер ребра.

В результате была сформулирована прямая задача на квантовом графе, построена математическая модель, сформирована база данных решений прямой задачи и построены нейронные сети с различной архитектурой, которые обучались на сформированной ранее базе данных. Был проведен анализ показателей, в результате чего была выбрана архитектура нейронной сети с наиболее большим процентом точности.

ABSTRACT

The thesis investigates the solution of the inverse problem of oscillation of a weightless quantum graph in the form of a five-pointed star, fixed at the ends. The aim of the work is to solve the inverse oscillation problem using neural networks on a weightless quantum graph of the "star" type. To solve the inverse problem, neural networks were used. For their training, the input consisted of the natural frequency and deflection angles at the ends of the graph, and the output provided the point of impact, mass, and the number of the edge on which the point mass fell. It is worth noting that two neural networks were used to solve the initial problem, one of which determined the point of impact and mass, and the other determined the number of the edge.

As a result, a direct problem on the quantum graph was formulated, a mathematical model was built, a database of solutions to the direct problem was formed, and neural networks with various architectures were constructed and trained on the previously formed database. An analysis of the indicators was carried out, as a result of which the neural network architecture with the highest accuracy percentage was selected.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В наше время искусственный интеллект, в частности машинное обучение, занимает центральное место в жизни человечества, проникая в самые различные аспекты — от финансовых транзакций и электронной коммерции до социальных медиа. Эта технология набирает силу, обеспечивая высокую производительность, гладкость и защищенность процессов. Она непрерывно развивается, открывая двери для разработки передовых систем, в том числе для автоматизации управления и роботизированных домашних помощников. В рамках настоящего дипломного исследования осуществляется анализ применения нейронных сетей для решения проблем, связанных с классической механикой. Главная задача исследования заключается в применении нейронной сети для решения обратных задач, связанных с выявлением массы точки, её расположения на нити, а также определения ребра, исходя из разнообразных параметров, включая период колебаний, углы отклонения, силу натяжения и временные интервалы.

1 Основные теоретические понятия

1.1 Python как инструмент работы с нейронной сетью

Python завоевал заслуженную популярность среди разработчиков и исследователей, работающих с нейронными сетями и искусственным интеллектом. Благодаря своей простоте, мощным библиотекам и активному сообществу, Python стал основным языком программирования для создания, обучения и развертывания нейронных сетей. Мы рассмотрим ключевые причины, по которым Python стал столь популярным в области нейронных сетей, а также обсудим основные библиотеки и инструменты, которые делают его незаменимым инструментом.

Python известен своей лаконичностью и читабельностью. Благодаря этому, код на Python легко писать и читать, что особенно важно в исследовательской работе и в проектах с открытым исходным кодом. Простота синтаксиса позволяет сосредоточиться на решении задач, связанных с моделированием нейронных сетей, вместо того чтобы тратить время на борьбу с синтаксическими особенностями языка.

Python имеет одно из самых активных и поддерживающих сообществ разработчиков. Это означает, что в случае возникновения проблем или вопросов всегда можно найти помощь или готовые решения. Обилие документации, руководств и образовательных ресурсов делает процесс обучения и работы с Python гораздо проще.

Язык программирования Python также обладает богатым набором библиотек, которые значительно упрощают работу с нейронными сетями. Рассмотренные далее библиотеки прямо или косвенно использовались в данной дипломной работе, поэтому остановимся на них подробнее.

1.1.1 TensorFlow

TensorFlow — это мощная и гибкая библиотека для машинного обучения, разработанная и поддерживаемая Google. Она была выпущена в 2015 году и быстро стала одним из самых популярных инструментов для создания и развертывания моделей машинного обучения и глубокого обучения. TensorFlow поддерживает множество алгоритмов и архитектур, от простых линейных моделей до сложных глубоких нейронных сетей. Также данная библиотека предоставляет широкий спектр инструментов и библиотек, позволяющих решать самые разнообразные задачи машинного обучения, от классификации изображений до обработки естественного языка. TensorFlow легко масштабируется для выполнения на разных платформах — от локальных машин до облачных серверов и мобильных устройств. Она имеет огромное и активное сообщество разработчиков и исследователей, что обеспечивает доступ к многочисленным ресурсам, документации, руководствам и примерам. [10] TensorFlow отлично интегрируется с облачными сервисами Google, такими как

Google Cloud AI, что упрощает развертывание и управление моделями в облаке.[4]

1.1.2 Keras

Keras — это высокоуровневый API для построения и обучения моделей глубокого обучения. Изначально разработанный Франсуа Шолле, Keras стал частью экосистемы TensorFlow и предоставляется как интегрированная библиотека в TensorFlow с версии 2.0. Основная цель Keras — упростить процесс создания нейронных сетей, делая его более интуитивно понятным и удобным для пользователей. Данная библиотека обеспечивает простой и четкий API, что позволяет быстро создавать, настраивать и тренировать модели нейронных сетей. Это делает его доступным для широкого круга пользователей, от новичков до экспертов. Модели в Keras строятся из отдельных, легко комбинируемых модулей, таких как слои, функции активации, оптимизаторы и потери. Это позволяет легко адаптировать и расширять функциональность библиотеки.

Хотя Keras тесно интегрирован с TensorFlow, он также поддерживает другие бэкенды, такие как Theano и Microsoft Cognitive Toolkit (CNTK), что делает его гибким в использовании. Также, как часть TensorFlow, Keras полностью совместим с TensorFlow и может использовать его мощные возможности, включая распределенные вычисления и развертывание на мобильных устройствах и в облаке.

1.1.3 Scikit-learn

Scikit-learn был создан как часть проекта SciPy в 2007 году, когда Давид Куртен (David Cournapeau) начал разрабатывать библиотеку во время работы над докторской диссертацией. Изначально библиотека была частью SciPy, но позже выделилась в отдельный проект. В 2010 году scikit-learn был официально выпущен как самостоятельная библиотека, и с тех пор активно развивается и поддерживается сообществом разработчиков и исследователей.

Хотя эта библиотека не специализируется исключительно на нейронных сетях, она предоставляет множество инструментов для машинного обучения, включая простые нейронные сети, и может быть полезна для предварительной обработки данных и базового анализа. Данная библиотека предлагает широкий спектр инструментов и алгоритмов для задач машинного обучения, включая классификацию, регрессию, кластеризацию, уменьшение размерности и предобработку данных.[6]

Помимо всего прочего, scikit-learn предоставляет простой и интуитивно понятный API, что делает его доступным для широкого круга пользователей. Библиотека включает множество алгоритмов для различных задач машинного обучения, что делает её универсальным инструментом для анализа данных. Scikit-learn хорошо интегрируется с другими библиотеками Python, такими как

NumPy, SciPy и Matplotlib, что облегчает работу с данными и визуализацию результатов.

1.1.4 Pandas

Также одним из важных инструментов языка программирования Python является библиотека pandas. Ведь работая напрямую с нейронными сетями и их дальнейшим обучением, требуется также работа с огромным количеством данных и их дальнейший анализ.

Pandas была разработана в 2008 году Уэсом Маккини (Wes McKinney) во время его работы в AQR Capital Management. Целью создания библиотеки было улучшение анализа данных в финансовой отрасли. В 2009 году Уэс Маккини опубликовал первую версию pandas как открытый проект. С тех пор библиотека активно развивается и поддерживается сообществом разработчиков и исследователей.

Pandas предоставляет две основные структуры данных: Series и DataFrame. Series — это одномерный массив, похожий на массив NumPy, но с индексами. Он может содержать данные любого типа (целые числа, строки, числа с плавающей запятой, объекты Python и т. д.). DataFrame — это двумерная структура данных, аналогичная таблице в реляционной базе данных или таблице Excel. Она состоит из строк и столбцов, каждый из которых может содержать данные разных типов.

Pandas предоставляет множество функций и операций для манипуляции данными. Рассмотрим основные из них:

Поддерживает чтение и запись данных из различных источников, таких как CSV, Excel, SQL и другие.

Предоставляет несколько методов для индексации и выборки данных из DataFrame, таких как loc и iloc.

Предоставляет функции для обработки отсутствующих данных, такие как isnull, dropna и fillna.

groupby позволяет группировать данные по определённым столбцам и выполнять агрегатные операции.

Поддерживает слияние и объединение данных из разных источников с помощью функций merge, join и concat.

Предоставляет функции для преобразования данных, такие как apply, map и applymap.[5]

1.2 Основы теории гармонических колебаний

Колебания представляют собой одно из важнейших явлений в механике и физике. Их можно наблюдать в широком диапазоне систем — от простых механических маятников до сложных электромагнитных волн. Понимание колебательных процессов имеет ключевое значение как для теоретической

физики, так и для множества прикладных наук. Возьмем, к примеру, гармонический осциллятор как основную модель, которая предоставляет основу для анализа более сложных систем и играет важную роль в изучении механических колебаний. Тем более, в моем случае колебания имеют гармонический характер, и рассмотрение данной модели будет для меня актуально.

Механические колебания представляют собой периодические движения, которые происходят вокруг положения равновесия. Они могут быть свободными, когда система возвращается к равновесию под действием внутренних сил, или вынужденными, когда система подвергается внешнему воздействию.

Гармонический осциллятор представляет собой простейшую модель колебательной системы, в которой возвращающая сила пропорциональна смещению и направлена в сторону положения равновесия (закон Гука). Примеры таких систем включают:

Маятник: при малых углах отклонения движение маятника можно описать гармоническими колебаниями. При больших углах отклонения движения становятся более сложными.

Пружинный маятник: Масса, подвешенная на пружине, демонстрирует гармонические колебания с периодом, зависящим от жесткости пружины.

Основные характеристики колебаний включают:

Период (T): Время, необходимое для одного полного цикла колебаний. Измеряется в секундах.

Частота (f): Количество полных колебаний в единицу времени. Обратная величина периода, измеряется в герцах (Гц).

Амплитуда (A): Максимальное отклонение системы от положения равновесия. Измеряется в соответствующих единицах длины, угла и т. д.

Фаза (φ): Начальное состояние колебательной системы в момент времени t=0. Измеряется в радианах или градусах.

Выведем уравнение простого гармонического осциллятора. (рис. 1.1)

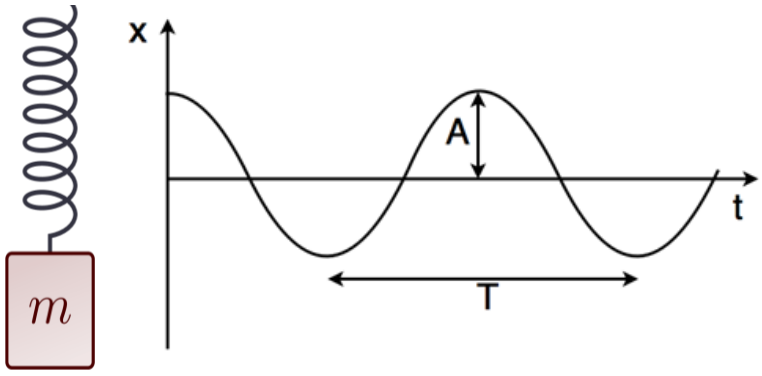


Рисунок 1.1 – График, описывающий простые гармонические колебания относительно x-положения и t – времени на примере пружинного маятника.

Можно сделать вывод что данного рода колебания зависят от коэффициента жесткости пружины(k), массы груза (m) на которую действует сила (F) и положении груза(x). Из второго закона Ньютона и закона Гука учитывая равновесия сил можно сделать вывод что:

$$F = ma = m\frac{d^2x}{dt^2} = mx'' = -kx, \tag{1.1}$$

$$mx'' + kx = 0, \tag{1.2}$$

$$x'' + \frac{kx}{m} = 0, \tag{1.3}$$

$$\omega^2 = \frac{k}{m}, \tag{1.4}$$

где ω — угловая частота системы. Тогда уравнение приобретает вид:

$$x'' + \omega^2 x = 0, \tag{1.5}$$

Для уравнения второго порядка с постоянными коэффициентами решаем характеристическое уравнение:

$$r + \omega^2 = 0, \tag{1.6}$$

$$r^2 = -\omega^2, \tag{1.7}$$

$$r = \pm i\omega, \tag{1.8}$$

Поскольку корни мнимые, общее решение дифференциального уравнения имеет вид:

$$x(t) = C_1 e^{i\,\omega t} + C_2 e^{-i\,\omega t}, \tag{1.9}$$

где C_1и C_2– произвольные константы.

Используем формулы Эйлера подставим в общее решение и получим:

$$x(t) = C_1\big(\cos(\omega t) + isin(\omega t)\big) + C_2\big(\cos(\omega t) - isin(\omega t)\big), \tag{1.10}$$

$$x(t) = (C_1 + C_2)\cos(\omega t) + i(C_1 - C_2)\sin(\omega t), \tag{1.11}$$

Пусть $C_1 + C_2 = A$ и $i(C_1 - C_2) = B$, где A и B - новые произвольные константы. Решение можно также записать в форме одного синуса или косинуса с фазой. В итоге получим решение уравнения (1.12):

$$x(t) = Acos(\omega t + \varphi), \quad (1.12)$$

где A – амплитуда;

φ- начальная фаза.

Гармонический осциллятор является моделью, лежащей в основе множества исследований и приложений. Ее используют для изучения колебательных процессов в квантовой механике, теории поля и других областях физики. Аналитические решения уравнений гармонических колебаний позволяют разрабатывать численные методы для моделирования более сложных систем. Колебания в механике представляют собой одно из ключевых явлений, лежащее в основе множества природных и искусственных процессов. Изучение этих процессов не только способствует развитию теоретической физики, но и имеет огромное практическое значение в инженерных и технологических приложениях.

1.3 Генерация данных-ключевой момент в обучении нейронных сетей

Анализ и сбор данных являются ключевыми элементами в любом проекте, связанном с обучением компьютерных моделей. Эти процессы играют решающую роль в определении успеха проекта, поскольку качество и количество данных напрямую влияют на точность и эффективность обучаемых моделей. Однако не всегда необходимые нам данные лежат на поверхности или доступны для использования. Часто процесс сбора данных требует значительных затрат, будь то временные ресурсы или финансовые инвестиции, и сталкивается с препятствиями из-за внешних обстоятельств, таких как географические ограничения или отсутствие необходимых инструментов для измерения.

Кроме того, нормы защиты конфиденциальности ограничивают способы применения или распространения информационных наборов. В условиях растущей озабоченности по поводу защиты персональных данных и соблюдения нормативных требований, таких как GDPR в Европе или HIPAA в США, многие организации сталкиваются с трудностями при использовании реальных данных для обучения своих моделей. Это приводит к необходимости поиска альтернативных решений для получения достаточного объема данных, которые соответствуют необходимым критериям и не нарушают законы о конфиденциальности.

В машинном обучении генерация данных играет критическую роль в обучении моделей. Для эффективного обучения алгоритмов необходимо большое количество данных, а генерация синтетических данных позволяет расширить существующие наборы данных и улучшить качество моделей. Например, в задачах распознавания изображений или обработки естественного языка, аугментация данных помогает улучшить обобщающую способность моделей. Аугментация данных включает в себя методы, которые позволяют

создавать новые данные на основе уже имеющихся, такие как вращение, изменение масштаба, добавление шума и другие преобразования.

Таким образом, использование синтетических данных не только решает проблемы конфиденциальности и затрат, но и предоставляет дополнительные возможности для улучшения моделей машинного обучения. Это подчеркивает важность дальнейших исследований и разработок в области генерации синтетических данных, чтобы обеспечить их качество и реалистичность, а также максимально эффективно использовать их потенциал в различных приложениях.

1.4 Значимость предварительной обработки данных

Одним из первых и наиболее важных этапов в предварительной обработке данных является их очистка. В реальных наборах данных часто встречаются ошибки, пропущенные значения и дублирующийся записи. Эти недочеты могут негативно сказаться на производительности нейронных сетей. Следующим важным шагом является нормализация или стандартизация данных. Этот процесс необходим для приведения данных к единому масштабу, что особенно важно для нейронных сетей, поскольку это может напрямую влиять на их обучение. Нормализация и стандартизация данных помогают ускорить процесс обучения, поскольку они приводят данные к единообразному масштабу. Это облегчает оптимизацию параметров модели и ускоряет сходимость алгоритмов градиентного спуска.

Стоит также подчеркнуть какими методами нормализации в дальнейшем будут использованы.

Масштабирование по минимуму и максимуму, также известное как метод Min-Max Scaling, является техникой нормализации данных, которая переводит их в определенный интервал, обычно от 0 до 1. Этот метод полезен для приведения различных переменных к одному масштабу, чтобы избежать преобладания одной переменной над другими в алгоритмах машинного обучения, таких как градиентный спуск или кластеризация методом k-средних.

Масштабирование с использованием метода Robust Scaling (Робастное масштабирование) — это метод нормализации данных, который предназначен для устойчивости к выбросам в данных. Он представляет собой альтернативу стандартному масштабированию с помощью среднего и стандартного отклонения, которое чувствительно к выбросам.

Предварительная обработка данных также помогает устранить шум и ошибки в данных, что приводит к более точным моделям. Чистые и хорошо подготовленные данные обеспечивают более эффективное обучение нейронных сетей и снижают вероятность переобучения. Таким образом, этап предварительной обработки данных играет критическую роль в создании качественных и надежных моделей машинного обучения.

2 Основной этап работы

2.1 Постановка прямой задачи

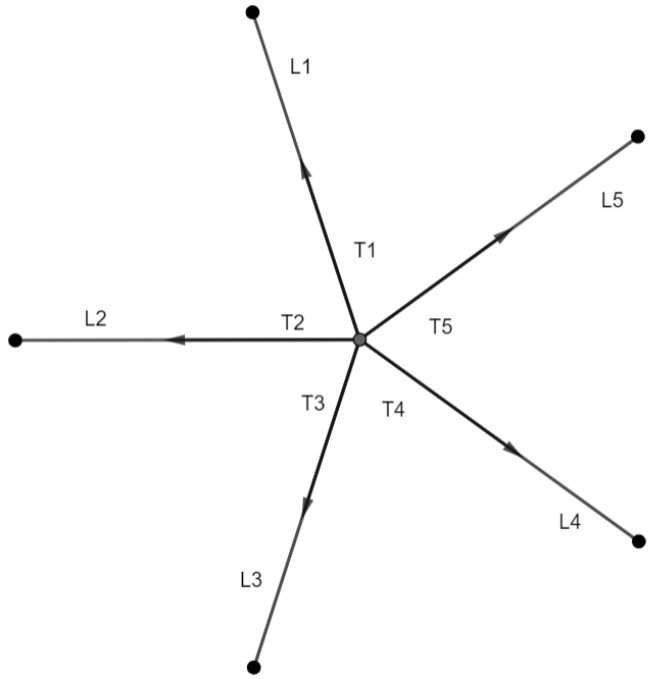


Рисунок 2.1 – Колебательная система в начальный момент времени

Рассматривается механическая система, состоящая из пяти невесомых нитей с фиксированными значениями натяжений и одинаковыми длинами, соединенных в общем узле и закрепленных на границе. На одну из ребер падает груз некоторой массы, тем самым возбуждая свободные колебания нити. В процессе колебаний нити движутся вдоль своей оси, совершая гармонические колебания вокруг своего положения равновесия (Рис. 2.1).

Следует отметить, что в реальных системах нити могут быть изготовлены из различных материалов и иметь разные диаметры, что требует учета дополнительных факторов при моделировании колебаний. Наша модель представлена в упрощенном виде за счет использования невесомых нитей и игнорирования сопротивления среды. Для математического описания модели мы применяем второй закон Ньютона. Когда точечная масса падает на одну из нитей, возникают колебания, так как натяжение нитей противодействует второму закону Ньютона.

Для составления математической модели данной сети требуется знать параметры: длины нитей (L1 L2 L3 L4 L5), натяжения нитей (T1 T2 T3 T4 T5), массу (m), точку падения массы(S), ребро на которое упала точечная масса и углы отклонения на концах ($\alpha, \beta, \gamma, \delta, \varepsilon$).

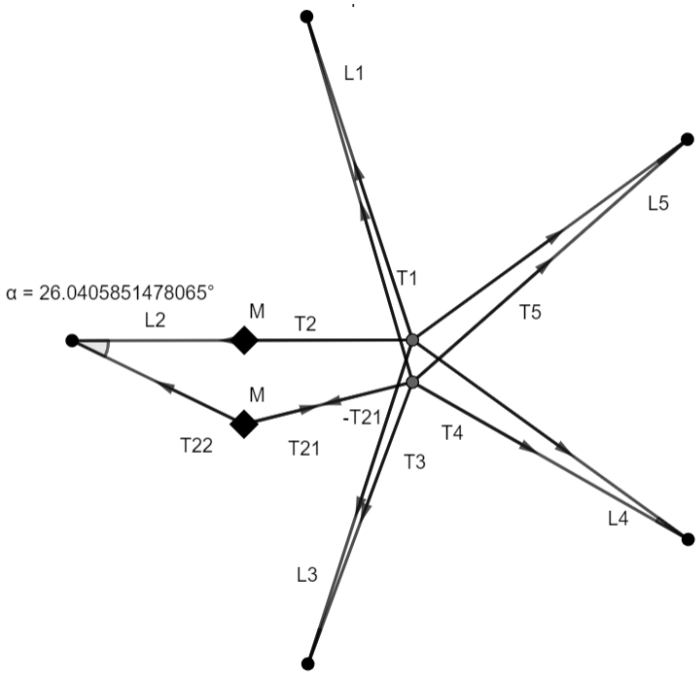


Рисунок 2.2 – Колебательная система после падения массы

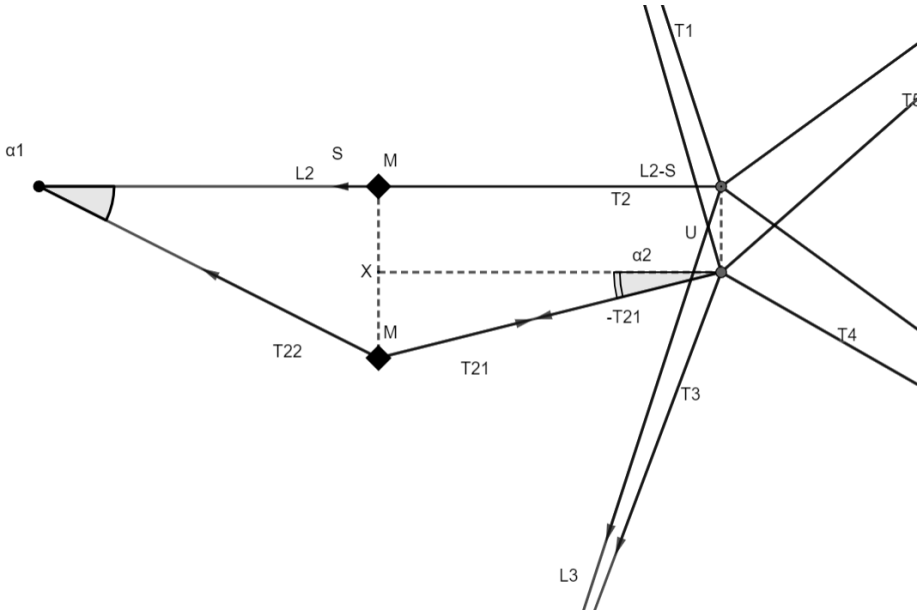


Рисунок 2.3 – Колебательная система после падения массы в увеличенном масштабе

После того как мы определили параметры нашей сети, можем приступить к построению самого уравнения движения. Зная, чему гласит второй и третий закон Ньютона и учитывая движение, которое происходит по направлению оси Оу получаем:

$$m\bar{a} = -(\overline{T_{22}} + \overline{T_{21}}), \quad (2.1)$$

Проецируем силы натяжения нитей на ось Oy:

$$ma = -(T_2 \sin\alpha_1 + T_2 sin\alpha_2), \quad (2.2)$$

За счет того, что происходят малые колебания, синус углов отклонения будет равен тангенсу этих же углов. Следует, что:

$$ma = -(T_2 tan\alpha_1 + T_2 tan\alpha_2), \quad (2.3)$$

$$m\frac{d^2x}{dt^2} = -(T_2 tan\alpha_1 + T_2 tan\alpha_2), \quad (2.4)$$

$$m\frac{d^2x}{dt^2} = -T_2(tan\alpha_1 + tan\alpha_2), \quad (2.5)$$

Из геометрических соображений определим, чему будут равны тангенсы углов отклонения α_1 и α_2 В прямоугольном треугольнике тангенс углов равны положению противолежащего катета к прилежащему катету:

$$m\frac{d^2x}{dt^2} = -\left[T_2\frac{x}{S} + T_2\frac{x-U}{L_2-S}\right], \quad (2.6)$$

$$m\frac{d^2x}{dt^2} + \left[T_2\frac{x}{S} + T_2\frac{x-U}{L_2-S}\right] = 0, \quad (2.7)$$

Вынесем T_2 и поделим обе части на m. Получим:

$$\frac{d^2x}{dt^2} + \frac{T_2}{m}\left[\frac{x}{S} + \frac{x-U}{L_2-S}\right] = 0, \quad (2.8)$$

Вынесем x за скобки и получим привычный вид уравнения:

$$\frac{d^2x}{dt^2} + \frac{T_2}{m}\left[\frac{1}{S} + \frac{1-\frac{U}{x}}{L_2-S}\right]x = 0, \quad (2.9)$$

где m – масса;
S – точка падения массы;
T_2 – натяжение второй нити;
L_2 – длина второй нити;
U – смещение свободного узла;

$x -$ смещение массы по оси Oy;

t – время.

В связи с тем, что у нас в состоянии колебания находится не только та нить, на которую упала масса, а также и другие нити так, как они все закреплены в одном свободном узле, требуется это учесть. Также учитывая, что имеются натяжения на других нитях, сумма нормальных натяжения равна нулю. Следовательно, будет описываться уравнением:

$$\frac{T_1 U}{L_1} + \frac{T_3 U}{L_3} + \frac{T_4 U}{L_4} + \frac{T_5 U}{L_5} + \frac{T_2 (x - U)}{L_2 - S} = 0, \tag{2.10}$$

где T1, T2, T3, T4, T5 – натяжения нитей;
L1, L2, L3, L4, L5 – длины нитей.

$$\begin{cases} \dfrac{d^2 x}{dt^2} + \dfrac{T_2}{m}\left[\dfrac{1}{S} + \dfrac{1 - \dfrac{U}{x}}{L_2 - S}\right] x = 0 \\ \dfrac{T_1 U}{L_1} + \dfrac{T_3 U}{L_3} + \dfrac{T_4 U}{L_4} + \dfrac{T_5 U}{L_5} + \dfrac{T_2 (x - U)}{L_2 - S} = 0 \end{cases} \tag{2.11}$$

Из второго уравнения (2.11) выведем значение U:

$$U\left(\frac{T_1}{L_1} + \frac{T_3}{L_3} + \frac{T_4}{L_4} + \frac{T_5}{L_5} - \frac{T_2}{L_2 - S}\right) = -\frac{T_2 x}{L_2 - S}, \tag{2.12}$$

$$U = -\frac{T_2 x}{(L_2 - S) D}, \tag{2.13}$$

$$D = \left(\frac{T_1}{L_1} + \frac{T_3}{L_3} + \frac{T_4}{L_4} + \frac{T_5}{L_5} - \frac{T_2}{L_2 - S}\right), \tag{2.14}$$

подставляя ее в исходное уравнение, получаем:

$$\frac{d^2 x}{dt^2} + \frac{T_2}{m}\left[\frac{1}{S} + \frac{1}{L_2 - S} + \frac{T_2}{(L_2 - S)^2 D}\right] x = 0, \tag{2.15}$$

2.2 Математическая модель колебания сети

Уравнение (2.15) является обыкновенным дифференциальным уравнением второго порядка моделирующее колебания нашей сети. Тем временем собственная частота будет равна:

$$\omega = \sqrt{\frac{T_2}{m}\left[\frac{1}{S} + \frac{1}{L_2 - S} + \frac{T_2}{(L_2 - S)^2 D}\right]}, \quad (2.16)$$

Для того чтобы решить прямую задачу требуется определить начальные условия чтобы решить задачу Коши. Мы полагаем что скорость в начальный момент времени равна 1 м/с. Так как в начальный момент времени механическая система покоится то перемещение в будет равна 0:

$$\begin{cases} x(0) = 0 \\ x'(0) = 1 \end{cases} \quad (2.17)$$

Поставим общее решение дифференциального уравнения 2 порядка (1.12) в условия Коши (2.17). Получаем:

$$x(0) = A\cos(\omega t + \varphi) = 0, \quad (2.18)$$

$$\cos(\omega t + \varphi) = 0, \quad (2.19)$$

$$\varphi = \frac{\pi}{2} + n\pi \text{ где } n \in \mathbb{Z}, \quad (2.20)$$

Следующим этап определим значение амплитуды, которое потребуется для дальнейших действий. Теперь вычислим производную от общего решения и подставим во второе условие:

$$x'(0) = -A\omega \sin(\omega t + \varphi) = 1, \quad (2.21)$$

$$x'(0) = -A\omega \sin(\varphi) = 1, \quad (2.22)$$

$$-A\omega = 1, \quad (2.23)$$

После чего мы находим чему равна амплитуда:

$$A = -\frac{1}{\omega}, \quad (2.24)$$

Построив математическую модель нашей сети, найдя решение прямой задачи и после нахождения всех нужных параметров, мы можем приступить к созданию базы данных для дальнейшего обучения и тестирования нашей нейронной сети.

2.3 Формирование базы данных решения прямой задачи

На данном этапе возник выбор на каком языке программирования провести создание баз данных. Так как ранее я работал на языке Python я решил сперва сформировать базу данных на этом языке. После того как я написал код для генерации данных, мне пришлось ждать около 25-30 минут чтобы он сгенерировал данные в 250 000 строк по 20 параметрам. На данный курьезный случай есть несколько причин. В первую очередь Python - интерпретируемый язык, что означает, что его код выполняется непосредственно интерпретатором. Это приводит к меньшей скорости выполнения по сравнению с компилируемыми языками, такими как C или C++. Также Python использует GIL, что ограничивает выполнение многопоточных программ. Хотя GIL не является проблемой для ввода-вывода, он ограничивает производительность при вычислительных задачах, которые могут использовать многоядерные процессоры. Помимо этого, Python - язык высокого уровня, который упрощает разработку за счет использования высокоуровневых конструкций. Это делает код более понятным и удобным для разработки, но менее эффективным с точки зрения производительности. Связи с этим было решено сгенерировать базы данных на языке C++, и на то были свои факты. В первую очередь C++ является компилируемым языком. Это означает, что исходный код компилируется в машинный код, который выполняется непосредственно процессором. Это обеспечивает высокую производительность и быструю обработку данных. Также C++ разработчики имеют полный контроль над управлением памятью. Это позволяет эффективно использовать ресурсы системы, минимизировать затраты на выделение и освобождение памяти и оптимизировать работу программы для конкретных задач.

После того как определились с языком программирования приступим к написанию кода для генерации данных. Для начала определимся с начальными параметрами.

Таблица 2.1 — Начальные параметры для генерации данных

T1, T2, T3, T4, T5	Натяжения нитей	5
L1, L2, L3, L4, L5	Длины нитей	10
v	Скорость распространения колебаний	1 м/с
m	Масса	5-15
S	Точка падения	0-10
t	время	0-20 сек
String	Номер нити	1-5

Как видно из таблицы некоторым параметрам было присвоено константное значение, а каким-то случайное в определенном интервале. Данное решение было принято для того, чтобы нейронная сеть могла лучше найти определенную

закономерность и связь между параметрами. Это сыграет роль в дальнейшем при обучении нейронной сети.

```
v = 1                #скорость распространения колебаний

L1 = 10              #длины нитей
L2 = 10
L3 = 10
L4 = 10
L5 = 10

T1 = 5               # натяжение нитей
T2 = 5
T3 = 5
T4 = 5
T5 = 5

for i in range(250000):
    string = random.randint(1,5) #возвращает целочисленное случайное значение
    m = random.uniform(5,15) #возвращает случайное значение с плавающей точкой,масса
    S = random.uniform(0,10) #точка падения массы
    t = random.uniform(0,20)
```

Рисунок 2.4 – Инициализация начальных значений на языке python

```
/*const double pi = std::acos(-1.0);*/
double v = 1.0;
double L1 = 10.0, L2 = 10.0, L3 = 10.0, L4 = 10.0, L5 = 10.0;
double T1 = 5.0, T2 = 5.0, T3 = 5.0, T4 = 5.0, T5 = 5.0;

std::vector<std::vector<double>> data;

for (int i = 0; i < 1000000; ++i) {
    double string = std::round(dis(gen) * 4.0) + 1;
    double m = dis(gen) * 10 + 5;
    double S = dis(gen) * 10.0;
    double t = dis(gen) * 20;
```

Рисунок 2.5 – Инициализация значений на языке C++

Помимо начальных параметров есть также расчетные такие как:

Таблица 2.2 — Расчетные параметры

Alpha	Угол отклонения 2 нити
Beta	Угол отклонения 1 нити

Gamma	Угол отклонения 5 нити
Delta	Угол отклонения 4 нити
Epsilon	Угол отклонения 3 нити
ω	Собственная частота
A	Амплитуда

```
if (string == 1.0) {
    D = (T2 / L2) + (T3 / L3) + (T4 / L4) + (T5 / L5) - (T1 / (L1 - S));
    W = std::sqrt(std::abs((T1 / m) * (1 / S + 1 / (L1 - S) + T1 / ((L1 - S) * D))));
    A = -v / W;
    x = A * std::cos(W * t + 0);
    U = -(T1 * x) / ((L1 - S) * D);
    alpha = std::atan(U / L2);
    beta = std::atan(x / S);
    gamma = std::atan(U / L5);
    delta = std::atan(U / L4);
    epsilon = std::atan(U / L3);
```

Рисунок 2.6 – Часть кода для демонстрации расчетных показателей для 1 нити

После проделанного этапа требовалось написать код для непосредственного генерирования данных. Было решено сформировать таблицу данных на 1000000 строк. Такое количество данных более чем достаточно для обучения и тренировки нашей будущей нейронной сети.

```
6,62283;1,09517;0,909654;4;0,0279243;0,0279243;0,0279243;-0,680473;0,0279243
14,8876;0,824394;9,67427;4;-0,0386616;-0,0386616;-0,0386616;-0,0347598;-0,0386616
10,7583;0,693932;8,77244;5;-0,161444;-0,161444;-0,161444;-0,161444;-0,0942125
10,8298;0,702916;3,55518;2;0,0324062;-0,00730375;-0,00730375;-0,00730375;-0,00730375
11,7013;0,716621;4,35564;1;-0,0349425;0,100601;-0,0349425;-0,0349425;-0,0349425
11,5191;1,00228;6,17782;5;0,178742;0,178742;0,178742;0,178742;-0,153453
13,1732;0,474766;8,98171;3;0,336148;0,336148;0,336148;0,336148;0,226608
7,26049;1,96195;9,8474;4;0,0417445;0,0417445;0,0417445;0,0398059;0,0417445
13,5656;0,679036;4,55099;2;-0,0408562;0,01577;0,01577;0,01577;0,01577
9,10066;0,788264;1,73523;4;-0,000151552;-0,000151552;-0,000151552;0,00201395;-0,000151552
6,1543;0,949528;3,86673;2;-0,195469;0,0526314;0,0526314;0,0526314;0,0526314
8,63971;0,513827;9,04699;2;0,208756;0,300351;0,300351;0,300351;0,300351
14,5505;0,594425;2,83998;1;-0,08981;0,533814;-0,08981;-0,08981;-0,08981
14,3734;0,822954;8,42567;4;0,0578177;0,0578177;0,0578177;0,0254309;0,0578177
5,26564;1,05236;4,19116;3;0,0218288;0,0218288;0,0218288;0,0218288;-0,0688357
12,8142;0,635054;3,05058;5;-0,038158;-0,038158;-0,038158;-0,038158;0,219152
12,1948;0,658164;3,41299;4;0,0788742;0,0788742;0,0788742;-0,361913;0,0788742
5,2487;1,00121;2,22204;3;-0,0288714;-0,0288714;-0,0288714;-0,0288714;0,267796
12,1773;1,02756;8,2648;2;-0,0862223;-0,229408;-0,229408;-0,229408;-0,229408
11,3602;0,767544;1,20861;4;0,0209797;0,0209797;0,0209797;-0,411906;0,0209797
13,4588;0,639;3,79263;5;-0,101647;-0,101647;-0,101647;-0,101647;0,37949
10,4229;0,821427;5,01626;5;0,0682277;0,0682277;0,0682277;0,0682277;-0,134521
8,68512;1,2619;0,43333;5;-0,02256;-0,02256;-0,02256;-0,02256;0,974024
9,74841;0,491996;9,40422;5;0,257466;0,257466;0,257466;0,257466;0,210119
9,89054;0,419241;9,36706;4;-0,14698;-0,14698;-0,14698;-0,117493;-0,14698
5,4076;1,25717;5,58256;5;-0,0271915;-0,0271915;-0,0271915;-0,0271915;0,0373496
```

Рисунок 2.7 – Сформированная таблица данных на языке C++ в формате txt для первой нейронной сети

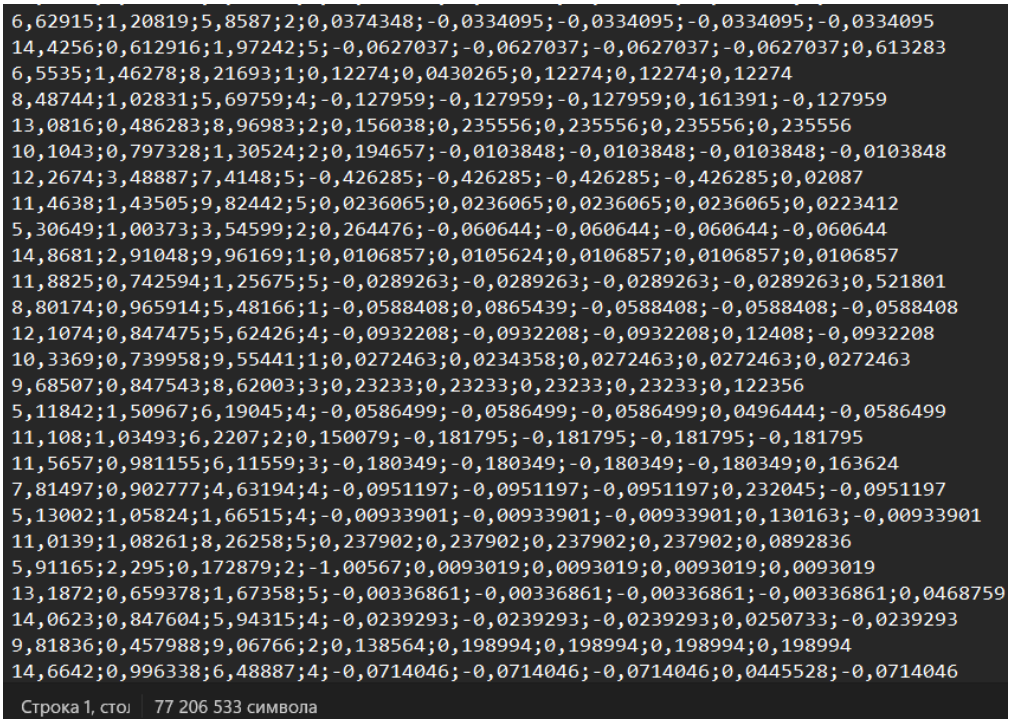


Рисунок 2.8 – Сформированная таблица данных на языке C++ в формате txt для второй нейронной сети

Также стоит отметить, что на данном этапе было решено сформировать две базы данных для двух разных по архитектуре нейронных сетей. Первая нейронная сеть будет определять точку падения и массу груза, в то же время вторая должна определить номер ребра. Также была сформирована третья таблица данных на 50 000 строк данных на языке python с целью узнать на сколько хорошо наша обученная нейронная сеть научилась предсказывать данные.

2.4 Предварительная обработка для улучшения качества данных

После того как база сформирована, её требуется обработать и проанализировать на наличие определённых дефектов или ошибок, которые в дальнейшем могут сказаться на непосредственном обучении. После анализа было выявлено, что данные имели тип «object» при сохранении файла в формате txt. Для дальнейшей работы требуется преобразовать их в тип «float», то есть в значения с плавающей точкой. После сохранения преобразованных данных в DataFrame, мы тем самым подготовили данные для дальнейшей работы.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999999 entries, 0 to 999998
Data columns (total 9 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   mass     999999 non-null  object
 1   fre      999999 non-null  object
 2   pos      999999 non-null  object
 3   string   999999 non-null  int64
 4   alfa     999999 non-null  object
 5   betta    999999 non-null  object
 6   gama     999999 non-null  object
 7   delta    999999 non-null  object
 8   epsilon  999999 non-null  object
dtypes: int64(1), object(8)
memory usage: 68.7+ MB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999999 entries, 0 to 999998
Data columns (total 9 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   mass     999999 non-null  float64
 1   fre      999999 non-null  float64
 2   pos      999999 non-null  float64
 3   string   999999 non-null  int64
 4   alfa     999999 non-null  float64
 5   betta    999999 non-null  float64
 6   gama     999999 non-null  float64
 7   delta    999999 non-null  float64
 8   epsilon  999999 non-null  float64
dtypes: float64(8), int64(1)
memory usage: 68.7 MB
```

Рисунок 2.9 –Данные до и после преобразования.

	mass	fre	pos	string	alfa	betta	gama	delta	epsilon
0	14.80770	1.089930	9.778420	4	0.100289	0.100289	0.100289	0.093512	0.100289
1	7.33265	2.323530	0.133748	4	-0.002435	-0.002435	-0.002435	0.492401	-0.002435
2	10.90890	1.105520	6.380690	3	-0.041025	-0.041025	-0.041025	-0.041025	0.028795
3	13.02450	0.698608	4.623270	3	-0.038166	-0.038166	-0.038166	-0.038166	0.094755
4	7.40549	0.839001	2.337460	1	-0.042543	0.359712	-0.042543	-0.042543	-0.042543
...	...	...	...	...	...	...	...	...	...
999994	13.07170	0.630330	2.395480	1	-0.010645	0.090492	-0.010645	-0.010645	-0.010645
999995	13.15940	0.924522	6.132350	1	-0.190383	0.170256	-0.190383	-0.190383	-0.190383
999996	6.56729	0.631344	9.015220	2	0.162601	0.239337	0.239337	0.239337	0.239337
999997	7.42547	0.838099	3.235610	2	0.209268	-0.040264	-0.040264	-0.040264	-0.040264
999998	7.05380	1.044900	5.304730	4	-0.027860	-0.027860	-0.027860	0.046098	-0.027860

999999 rows × 9 columns

Рисунок 2.10 – База данных сохраненная в DataFrame

Также в ходе анализа данных был выявлен примерный диапазон значений собственной частоты и углов отклонения. Этот шаг был выполнен для дальнейшей правильной нормализации данных. Углы отклонения находились в диапазоне от -1.6 до 1.5. В то же время значения собственной частоты находятся в диапазоне от 0 до 2.

Также по данному графику можно понять, что имеются некоторые выбросы. Когда сталкиваются с аномальными значениями, некоторые стремятся заменить их на более лаконичные и удобные значения, такие как медианные или

средние значения, так как они могут напрямую повлиять на предварительную обработку данных и в дальнейшем на показатели. Но для более реалистичных показателей нам не требуется от них избавляться.

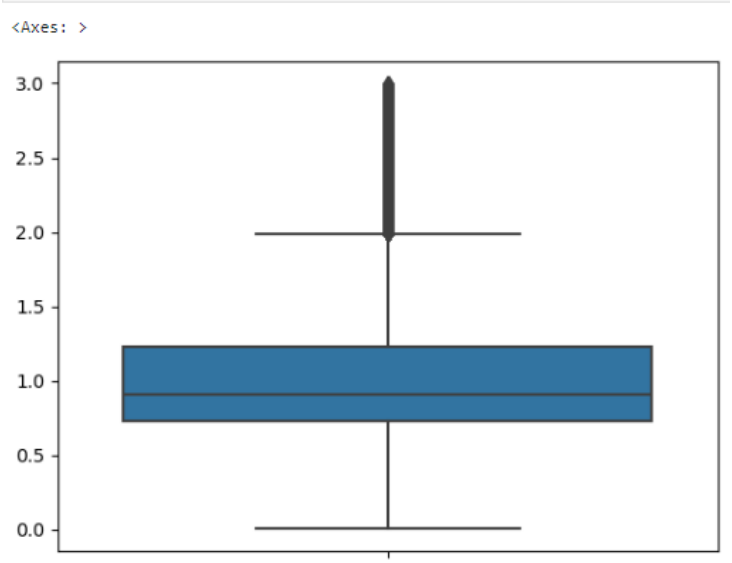


Рисунок 2.11 – Диаграмма размаха собственных значений

Теперь, когда понятно, в каких примерных диапазонах находятся наши данные, требуется провести нормализацию данных. Этот этап нужен для того, чтобы свести данные параметров в один диапазон, так как для нейронной сети будет затруднительно обучаться на дискретных и непрерывных значениях одновременно. Для того чтобы провести грамотную нормализацию данных, нужно в первую очередь поделить данные на входные и выходные. Это делается с целью показать нейронной сети, чего мы ожидаем получить на выходе относительно входных данных. Для первой нейронной сети на вход будут подаваться 6 параметров (собственная частота и углы отклонения), а на выходе мы должны будем получить 2 параметра, такие как точка падения и масса. Для второй нейронной сети и ее собственной сформированной базы данных на вход идут 5 параметров углов отклонения, а на выходе 1 параметр — это номер ребра. Определившись, как мы поделим данные, стоит решить, каким методом мы будем их нормировать. В ходе экспериментов, из неких соображений, было решено нормировать данные собственной частоты в интервале от -1.5 до 1.5, а углы отклонения оставить в исходном диапазоне. Фактически, мы нормировали данные ω, учитывая диапазон углов.

```
input_df2 = df[['alfa','betta','gama','delta','epsilon']]
#normalized_df = input_df.apply(lambda row: (row - row.min()) / (row.max() - row.min()), axis=1)
input_df1 = df[['fre']]
# Создание объекта
scaler = MinMaxScaler(feature_range=(-1.5,1.5))
# Подгонка и преобразование данных
input_df_scaled1 = pd.DataFrame(scaler.fit_transform(input_df1), columns=input_df1.columns)
input_df_scaled = pd.concat([ input_df_scaled1,input_df2],axis=1)
input_df_scaled = input_df_scaled.round(4)
input_df_scaled
```

Рисунок 2.12 – Нормирование значений собственной частоты и дальнейшее объединение входных данных

	fre	alfa	betta	gama	delta	epsilon
0	-0.4111	0.1003	0.1003	0.1003	0.0935	0.1003
1	0.8231	-0.0024	-0.0024	-0.0024	0.4924	-0.0024
2	-0.3955	-0.0410	-0.0410	-0.0410	-0.0410	0.0288
3	-0.8026	-0.0382	-0.0382	-0.0382	-0.0382	0.0948
4	-0.6622	-0.0425	0.3597	-0.0425	-0.0425	-0.0425
...	...	...	...	...	...	...
999994	-0.8710	-0.0106	0.0905	-0.0106	-0.0106	-0.0106
999995	-0.5766	-0.1904	0.1703	-0.1904	-0.1904	-0.1904
999996	-0.8700	0.1626	0.2393	0.2393	0.2393	0.2393
999997	-0.6631	0.2093	-0.0403	-0.0403	-0.0403	-0.0403
999998	-0.4561	-0.0279	-0.0279	-0.0279	0.0461	-0.0279

999999 rows × 6 columns

Рисунок2.13 – Входные данные для первой нейронной сети

Нормализацию выходных данных мы провели также методом Min-Max в интервале от -1.5 до 1.5, так как данный метод неплохо себя показал именно в обучении нейронной сети.

	mass	pos
0	1.44231	1.43352
1	-0.80022	-1.45989
2	0.27267	0.41421
3	0.90735	-0.11301
4	-0.77835	-0.79875
...	...	...
999994	0.92151	-0.78135
999995	0.94782	0.33972
999996	-1.02981	1.20456
999997	-0.77235	-0.52932
999998	-0.88386	0.09141

999999 rows × 2 columns

Рисунок 2.14 – Выходные данные для первой нейронной сети

Что касается нормализации данных для второй нейронной сети, то здесь на вход подаются 5 параметров с значениями углов отклонения, а на выходе 1 параметр с номером ребра. Как и ранее, было решено не нормировать данные углов, а нормировать лишь номер ребра в интервале от -1.5 до 1.5.

	alfa	betta	gama	delta	epsilon
0	-0.0387	-0.0387	-0.0387	-0.0348	-0.0387
1	-0.1614	-0.1614	-0.1614	-0.1614	-0.0942
2	0.0324	-0.0073	-0.0073	-0.0073	-0.0073
3	-0.0349	0.1006	-0.0349	-0.0349	-0.0349
4	0.1787	0.1787	0.1787	0.1787	-0.1535
...	...	...	...	...	...
999994	-0.0305	-0.0212	-0.0305	-0.0305	-0.0305
999995	0.0587	-0.2432	0.0587	0.0587	0.0587
999996	-0.0414	-0.0414	-0.0414	0.5681	-0.0414
999997	-0.0700	-0.0700	-0.0700	-0.0700	0.6117
999998	0.1286	-0.0392	-0.0392	-0.0392	-0.0392

999999 rows × 5 columns

	string
0	0.75
1	1.50
2	-0.75
3	-1.50
4	1.50
...	...
999994	-1.50
999995	-1.50
999996	0.75
999997	0.00
999998	-0.75

999999 rows × 1 columns

Рисунок 2.15 – Входные и выходные данные для второй нейронной сети

2.5 Применение нейронных сетей для решения обратной задачи колебания

Для решения обратной задачи колебаний требуется разработать оптимальную архитектуру нейронной сети. В этом процессе необходимо учесть несколько ключевых факторов, включая выбор типа нейронной сети, количество слоев и нейронов в каждом слое, функцию активации, а также методы регуляризации и оптимизации. В зависимости от характера задачи могут быть использованы различные типы нейронных сетей, такие как полносвязные сети (MLP), сверточные нейронные сети (CNN) для обработки пространственных данных или рекуррентные нейронные сети (RNN) для временных рядов. Мы будем использовать сети прямого распространения. Также эффективная работа модели оценивается на основе ее производительности на данных, которые не использовались в процессе обучения. Это требует грамотного разделения данных на обучающие, валидационные и тестовые наборы. Поэтому первым делом я разделил данные на тестовые и тренировочные.

```
output_train = output_df_scaled[50000:]
input_train = input_df_scaled[500000:]

input_test = input_df_scaled[:500000]
output_test = output_df_scaled[:500000]
```

Рисунок 2.16 – Деление данных на тестовые и тренировочные

Следующим шагом является определение наиболее эффективной архитектуры нейронной сети. Данный шаг является итеративным процессом, включающим множество этапов. Это сложная задача, которая зависит от характера данных и конкретной задачи. Оптимизация архитектуры нейронной сети — это процесс, требующий систематического подхода и экспериментов.

В первую очередь требуется подключить нужные библиотеки.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequentia
from tensorflow.keras.layers import Dense
```

Рисунок 2.17 – Подключение библиотек Tensorflow, Keras

Sequential: это класс из Keras, который используется для создания последовательной модели нейронной сети. Последовательная модель — это модель, где слои идут друг за другом в линейной последовательности. Dense: это класс из Keras, который используется для создания полносвязных (или полностью соединённых) слоёв нейронной сети. В полносвязном слое каждый нейрон соединён со всеми нейронами предыдущего слоя. Так как наша обратная задача колебания была сведена на две нейронные сети, то нам необходимо подобрать оптимальную архитектуру для каждой ситуации по отдельности, так

как если подбирать одну оптимальную архитектуру для двух разных случаев, то один из параметров будет страдать по проценту точности. Возьмем первый случай. Нам требуется нейронная сеть, которая будет предсказывать значения массы и точки падения груза исходя из входных параметров, таких как собственная частота и углы отклонения. Так как на вход подаются 6 параметров, то в первом слое будут 6 нейронов, а на выходе 2 нейрона, так как на выходе мы получаем 2 параметра. Помимо всего прочего, нужно определить количество скрытых слоев и нейронов в каждом слое. После нескольких экспериментов было понятно, что для данной задачи хватит максимум 2 скрытых слоя. При больших количествах скрытых слоев сеть просто переобучалась. Для пояснения: переобучение (overfitting) в нейронных сетях происходит, когда модель слишком хорошо подстраивается под тренировочные данные и теряет способность обобщать знания на новых данных. По определенным признакам можно понять, когда нейронная сеть начинает переобучаться. В первую очередь, если ошибка (loss) на валидационных данных значительно выше, чем на тренировочных данных, то это явный признак переобучения. Помимо всего прочего, если график ошибки на тренировочных данных продолжает снижаться, в то время как на валидационных начинает расти после некоторого количества эпох, то нейронная сеть переобучается. Решив вопрос со скрытыми слоями, требуется теперь определить количество нейронов. Существует правило золотой середины. Суть в том, что все начинается с простых моделей и постепенно увеличивается количество нейронов до тех пор, пока производительность на валидационных данных не перестанет улучшаться. Также можно начинать с количества нейронов, равного средней величине между размером входного и выходного слоев. Также было важно определить такие параметры, как функции активации, функции потери, оптимизаторы. На скрытых слоях я использовал функцию активации tanh. Преимущества его в том, что выходы находятся в диапазоне (-1, 1), решает проблему смещения выходов и его часто используют в скрытых слоях. Функцию потерь я выбрал для начала Mean Squared Error, но также экспериментировал и с другими. MSE измеряет среднеквадратичное отклонение предсказанных значений от истинных. [1]Это наиболее распространенная функция потерь для задач регрессии. Часто используется в прогнозировании. После было выбрано несколько известных оптимизаторов, такие как SGD, Adam, AdaGrad. SGD - базовый оптимизатор, обновляющий веса сети на основе градиента функции потерь с небольшими случайными подвыборками (батчами) данных. Преимущества в том, что ускоряет сходимость, особенно на негладких поверхностях функции потерь. Adam в ту же очередь также широко используется в современных глубоких нейронных сетях, связи с тем, что адаптивно изменяет скорости обучения на основе моментов первого и второго порядков. Данные рекомендации помогли достичь нужных результатов. По началу для эксперимента была построена сеть с одним скрытым слоем. (Рис. 2.18)

```
model = tf.keras.Sequential()
model.add(keras.layers.Dense(units=6, activation='tanh'))
model.add(keras.layers.Dense(units=4, activation='tanh'))
model.add(keras.layers.Dense(units=2, activation='tanh'))
model.compile(optimizer='Adam', loss='mean_squared_error', metrics=['accuracy'])
```

Рисунок 2.18 – Нейронная сеть с 1 скрытым слоем

Точность предсказаний нейронной сети составила 77.33%, что является хорошим показателей. Потери составили 0.3580.

```
Epoch 15/15
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 11s 694us/step - accuracy: 0.7733 - loss: 0.3580
```

Рисунок 2.19 – Точность предсказания массы и точки падения с результатом 77.33%

Также большую роль в хорошей обучаемости сети сыграла правильная предобработка данных. Но для улучшения показателей воспользуемся рекомендациями, описанными ранее. Добавим еще один слой и поменяем количество нейронов. (Рис. 2.20)

```
]: model = tf.keras.Sequential()
   model.add(keras.layers.Dense(units=6, activation='tanh'))
   model.add(keras.layers.Dense(units=6, activation='tanh'))
   model.add(keras.layers.Dense(units=2, activation='tanh'))
   model.add(keras.layers.Dense(units=2, activation='tanh'))
   model.compile(optimizer='Adam', loss='mean_squared_error', metrics=['accuracy'])
```

Рисунок 2.20 – Наиболее оптимальная архитектура нейронной сети

Процент точности нейронной сети (рис. 2.21) составил 91.68%. В то же время значение потерь снизилось до 0.0719. Меньшие значения потерь указывают на то, что модель лучше справляется с предсказанием целевых значений.

```
Epoch 10/10
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 11s 724us/step - accuracy: 0.9168 - loss: 0.0719
```

Рисунок 2.21 – Точность предсказания массы и точки падения с результатом 91.68%

Для примера были также созданы другие архитектуры, которые по точности уступали нейронной сети на (рис. 2.22)

```
model = tf.keras.Sequential()
model.add(keras.layers.Dense(units=6, activation='tanh'))
model.add(keras.layers.Dense(units=6, activation='tanh'))
model.add(keras.layers.Dense(units=4, activation='tanh'))
model.add(keras.layers.Dense(units=2, activation='tanh'))
model.add(keras.layers.Dense(units=2, activation='tanh'))
model.compile(optimizer='Adam', loss='mean_squared_error', metrics=['accuracy'])
```

Рисунок 2.22 – Нейронная сеть с 3 скрытыми слоями

В случае на (рис. 2.22), я увеличил количество слоев до 3. Количество нейронов во 2 слое было выбрано относительно среднего значения между входным и выходным слоями. Явно показано, что точность достигла своего пика на значении 77%, а показатели потерь значительно увеличились, что указывает на то, что обучение проводится хуже по сравнению с предыдущей архитектурой.

```
━━━━━━━━━━ 14s 888us/step - accuracy: 0.7758 - loss: 0.3427

━━━━━━━━━━ 20s 1ms/step - accuracy: 0.7770 - loss: 0.3407
```

Рисунок 2.23 Точность предсказания массы и точки падения с результатом 77.7%

Также были протестированы другие архитектуры с разными функциями активации, оптимизаторами и функциями потерь, но в основном предсказания этих архитектур доходили до пиковых значений точности около 88%. Примерно на 15-20 эпохе точность переставала расти, а потери иногда превышали отметку 1.

Для того, чтобы понять, насколько хорошо наша нейронная сеть предсказывает значения, проведем обучение непосредственно на новых данных. С помощью команды evaluate проведем анализ:

```
model.evaluate(input_test,output_test)

15625/15625 ━━━━━━━━━━━━━━━━━━━━ 8s 519us/step - accuracy: 0.9147 - loss: 0.1526
[0.15342655777931213, 0.9139059782028198]
```

Рисунок 2.24 – Точность предсказания массы и точки падения с результатом 91,47%, потери 0.1256

Как видно, наша первая нейронная сеть работает.

Теперь, с помощью графиков, оценим и проанализируем результаты работы. Для того, чтобы построить графики, нам потребуются нужные библиотеки. Самые популярные из них считаются seaborn и matplotlib.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Рисунок 2.25 – Подключение библиотек

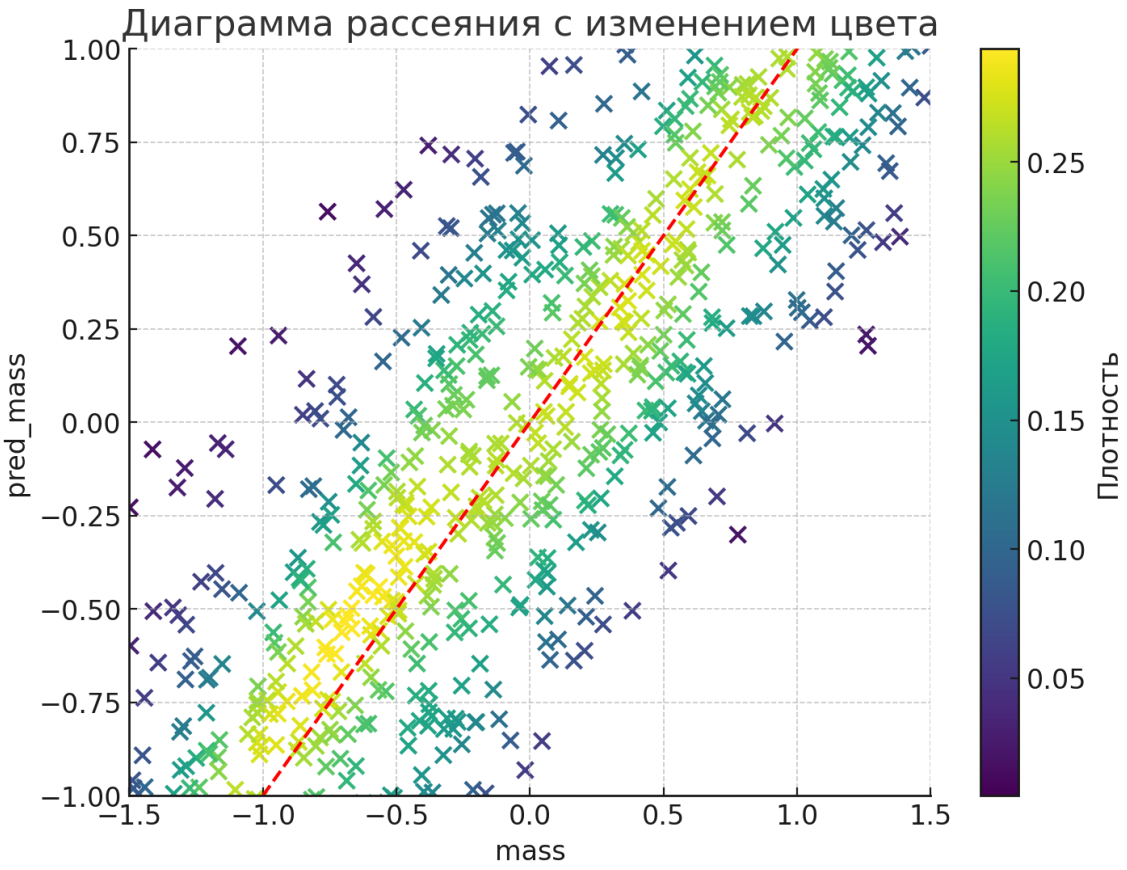


Рисунок 2.26 – Диаграмма рассеяния с изменением цвета

На (рис. 2.26) изображена диаграмма рассеяния с изменением цвета в зависимости от плотности их распределения на предсказанных и реальных значениях массы. Этот тип графика помогает визуализировать, как данные распределены и где наблюдается наибольшая плотность точек. Области с высоким скоплением точек обозначены светлыми цветами (жёлтыми и зелёными). Эти области указывают на значения переменных "mass" и "pred_mass", которые чаще встречаются в наборе данных. Области с низкой плотностью точек обозначены тёмными цветами (синими и фиолетовыми). Можно заметить, что есть диагональная полоса высокой плотности, что может указывать на корреляцию между "mass" и "pred_mass". То есть, по мере увеличения "mass", "pred_mass" также увеличивается, что может свидетельствовать о положительной зависимости между этими переменными.

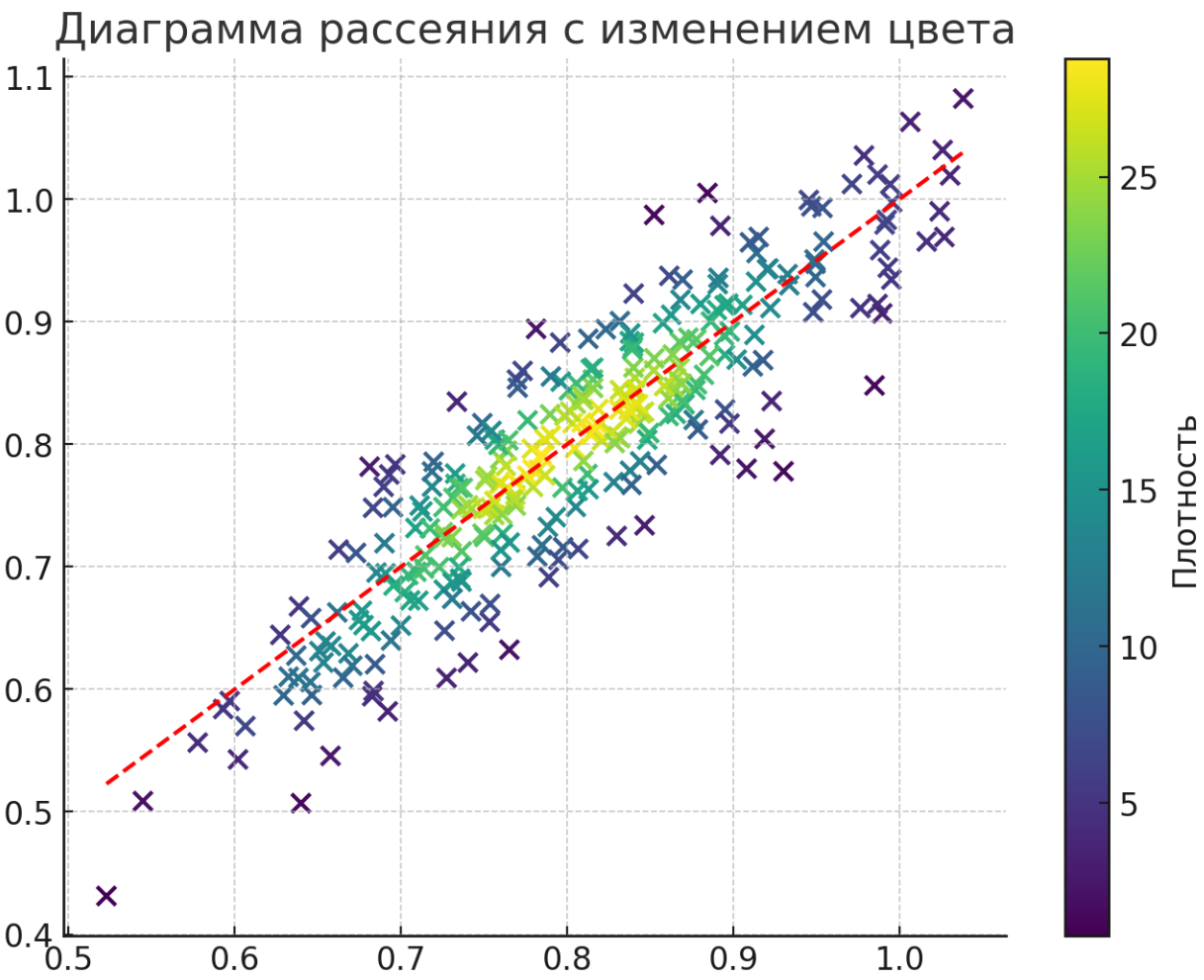


Рисунок 2.27 – Диаграмма рассеяния с изменением цвета

На (рис. 2.27) изображена диаграмма рассеяния с изменением цвета в зависимости от плотности их распределения на предсказанных и реальных значениях точки падения. Видно, что большое скопление происходит в области (0.8; 0.8).

По данным графикам можно сделать вывод, что наша нейронная сеть справляется с задачей, хоть и имеются некоторые выбросы. Возможно, как ранее было сказано, на данные аномальные значения повлияли значения собственной частоты. Тем не менее у нас имеются реальные данные и реальные предсказанные значения с точностью 91.47%. На этом можно перейти к следующей задаче.

2.6 Сложности при обучении второй нейронной сети и пути ее решения

Начав обучать вторую нейронную сеть предсказывать номер ребра, куда упала масса, основываясь на входных данных углов отклонения, я столкнулся с

некоторыми сложностями. В ходе экспериментов для подбора оптимальной архитектуры нейронной сети я смог предсказывать значения ребер с максимальной точностью 24.25%, при этом потери составили 0.0313. О том, что сеть обучается не должным образом, можно судить по графику.

Сама архитектура состояла из 1 скрытого слоя с 3 нейронами и функцией активации tanh. На входном слое было 5 нейронов, так как на вход подавались пять углов отклонения, а на выходе 1 нейрон, то есть номер ребра. Функции активации при этом были выбраны relu, так как в экспериментах она лучше показала себя на входных и выходных слоях.

```
[19]: model = tf.keras.Sequential()
      model.add(keras.layers.Dense(units=5, activation = 'relu'))
      model.add(keras.layers.Dense(units=4, activation = 'tanh'))
      #model.add(keras.layers.Dense(units=2, activation = 'softmax'))
      model.add(keras.layers.Dense(units=1, activation = 'relu'))
      model.compile(optimizer='Adam',loss='mean_squared_error',metrics=['accuracy'])

15625/15625 ━━━━━━━━━━━━━━━━━━━━ 20s 1ms/step - accuracy: 0.2409 - loss: 0.0324
Epoch 10/10
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 19s 1ms/step - accuracy: 0.2425 - loss: 0.0313
```

Рисунок 2.28 – Нейронная сеть для предсказания нити на которую упала масса

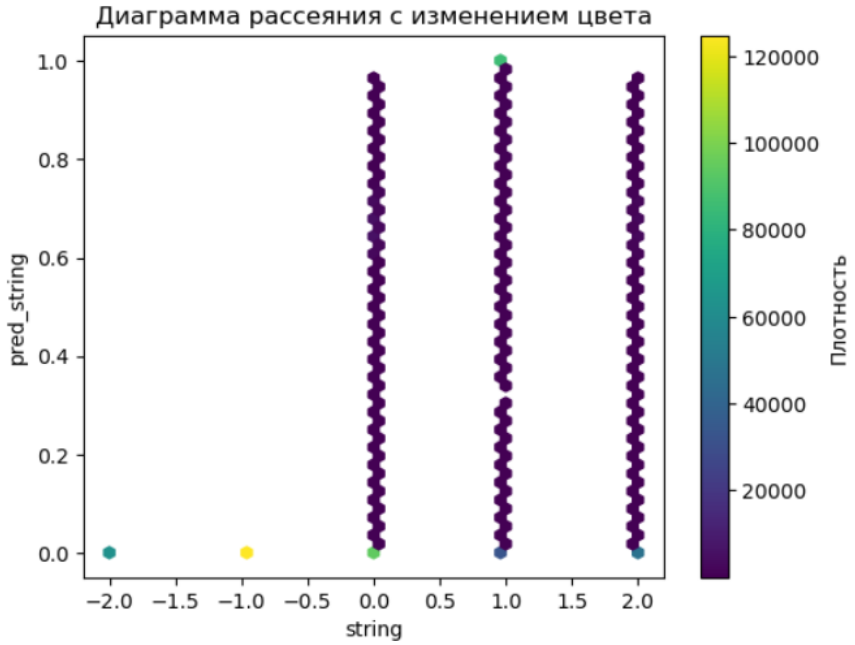
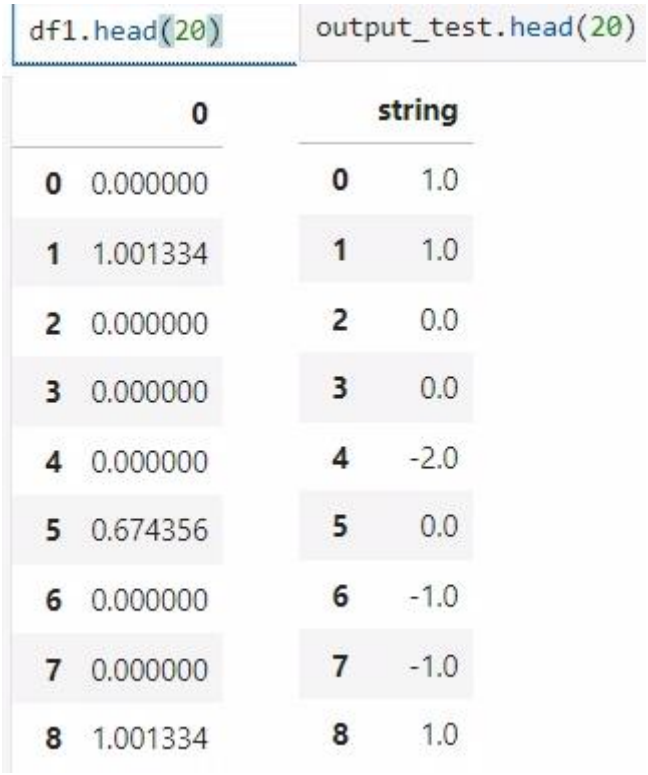


Рисунок 2.29 – Диаграмма рассеяния с изменением цвета предсказанных и реальных значениях string

Возможная проблема, возникшая при предсказывании номера ребра, заключалась в том, что при попытке предсказать дискретные величины, подавая на вход непрерывные данные, сеть до конца не понимала, в чем заключается закономерность. Если с первой нейронной сетью все было понятно, так как и масса, и точка падения имели непрерывные значения, и зависимость между

входными и выходными данными была функциональной, то во второй нейронной сети такой зависимости нет, хотя она все же присутствует.



df1.head(20)

	0
0	0.000000
1	1.001334
2	0.000000
3	0.000000
4	0.000000
5	0.674356
6	0.000000
7	0.000000
8	1.001334

output_test.head(20)

	string
0	1.0
1	1.0
2	0.0
3	0.0
4	-2.0
5	0.0
6	-1.0
7	-1.0
8	1.0

Рисунок 2.30 – Слева предсказанные данные ребер, справа реальные значения

Если присмотреться внимательно к нашей сформированной базе данных, то можно заметить определенную закономерность (рис. 2.31). Один из углов отличается от других, в то время как остальные углы одинаковы. Отсюда следует тот факт, что если один из углов отличается от остальных, то масса упала именно на ту нить, которой принадлежит этот угол. Если угол, который отличается, это alpha, то масса упала на 2 нить; если beta, то на 1; gamma - на 5; delta - на 4; epsilon - на 3. Но у данной закономерности нет определенного функционала, а нити имеют дискретные значения. И на самом деле, подавая на вход непрерывные значения, а на выходе ожидая дискретные, не совсем понятно, как именно нейронная сеть должна предсказывать номер ребра, учитывая, что на выходе мы получаем непрерывные значения (рис.). Тогда возникло решение разделить отрезок [0;1] на 5 равных частей, и если, к примеру, точечная масса упала на первую нить, то мы присваиваем ей непрерывное значение в этом определенном интервале. Для того, чтобы нейронная сеть могла уловить закономерность, было решено присваивать фиксированные значения, а не случайные.

Таблица 2.3 — Деление отрезка (0:1) на 5 равных частей.

(0;0.2)	(0.2;0.4)	(0.4;0.6)	(0.6;0.8)	(0.8;1)
1 нить	2 нить	3 нить	4 нить	5 нить

По такому алгоритму мы получаем новую базу данных со значениями нитей (рис. 2.31). В каком-то роде мы провели нормализацию данных (рис. 2.32).

Тем не менее данный метод не поспособствовал хорошему обучения, а наоборот ухудшил показатели.

string	alfa	betta	gama	delta	epsilon
4	-0.0387	-0.0387	-0.0387	-0.0348	-0.0387
5	-0.1614	-0.1614	-0.1614	-0.1614	-0.0942
2	0.0324	-0.0073	-0.0073	-0.0073	-0.0073
1	-0.0349	0.1006	-0.0349	-0.0349	-0.0349
5	0.1787	0.1787	0.1787	0.1787	-0.1535
...	...	...	...	...	...
1	-0.0305	-0.0212	-0.0305	-0.0305	-0.0305
1	0.0587	-0.2432	0.0587	0.0587	0.0587
4	-0.0414	-0.0414	-0.0414	0.5681	-0.0414
3	-0.0700	-0.0700	-0.0700	-0.0700	0.6117
2	0.1286	-0.0392	-0.0392	-0.0392	-0.0392

Рисунок 2.31 База данных для второй нейронной сети

```
[14]: #def generate_random_value(min_v , max_v):
          #return random.uniform(min_v , max_v)

      def apply_rules(row):
          if row['string'] == 1:
              return 0.124512
          elif row['string'] == 2:
              return 0.345678
          elif row['string'] == 3:
              return 0.587845
          elif row['string'] == 4:
              return 0.785245
          elif row['string'] == 5:
              return 0.985645
          else:
              return np.nan
      random_values = output_df.apply(apply_rules, axis=1)
      result_output_df = pd.DataFrame(random_values)
      columns_h = ['string']
      result_output_df.columns = columns_h
      result_output_df
```

[14]:

	string
0	0.785245
1	0.785245
2	0.587845
3	0.587845
4	0.124512
...	...
999994	0.124512
999995	0.124512
999996	0.345678
999997	0.345678
999998	0.785245

Рисунок 2.32 Новая нормализация выходных данных для второй нейронной сети

```
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 13s 837us/step - accuracy: 0.0000e+00 - loss: 0.0154
Epoch 9/10
 5157/15625 ━━━━━━━━━━━━━━━━━━━━ 8s 765us/step - accuracy: 0.0000e+00 - loss: 0.0155
```

Рисунок 2.33 Точность предсказания номер ребра по новому методу нормализации

Так как данная проблема возникла большей части из-за того, что данные имеют разный характер значений, было решено поэкспериментировать с методами нормализации. В итоге получилось повысить точность предсказаний до 38.08%. (рис 2.34). На тестовых данных процент точности составил 37.59%.

```
Epoch 4/10
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 10s 649us/step - accuracy: 0.3808 - loss: 1.2757
```

Рисунок 2.34 Точность предсказания номер ребра по методу минимаксному нормализации в отрезке (-2:2)

```
model.evaluate(input_test,output_test)
```

```
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 8s 498us/step - accuracy: 0.3773 - loss: 1.2737
[1.2765791416168213, 0.3759920001029968]
```

Рисунок 2.35 Процент точности предсказания на новых данных

В связи с тем, что нейронная сеть попросту не могла обнаружить определенной закономерности в данных, было решено создать код на Python, который будет определять ребро, на которое упала масса. Для этого мы создадим новый DataFrame, куда будем загружать наши новые данные по номеру ребер. Также с помощью dif создаем функцию check_values, которая как раз таки будет определять отличающийся угол и присваивать ему значение ребра (рис. 2.36).

Судя по (рис. 2.37), наш код работает, и тем самым мы доказали нашу гипотезу о закономерности между углами и номерами ребер. На этом этапе работа была завершена.

```
ef check_values(row):
   if row['alfa'] != row['betta'] or row['alfa'] != row['gama'] or row['alfa'] != row['epsilon'] or row['alfa'] != row['delta']:
       if row['alfa'] != row['betta']:
           return 2
       elif row['betta'] != row['gama']:
           return 1
       elif row['gama'] != row['epsilon']:
           return 5
       elif row['delta'] != row['epsilon']:
           return 4
       elif row['epsilon'] != row['delta']:
           return 3
   return None
```

Рисунок 2.36 Функция check_values для определения ребра на которую упала масса

```
new_df[['string']]
```

	string
0	4
1	5
2	2
3	1
4	5
...	...
999994	1

```
df[['string']]
```

	string
0	4
1	5
2	2
3	1
4	5
...	...
999994	1

Рисунок 2.37 Реальные и найденные данные номера ребра

ЗАКЛЮЧЕНИЕ

В дипломной работе исследовалось и анализировалось возможность применения нейронных сетей в решении обратной задачи колебания сети из нитей. Целью исследования было обучить нейронную сеть предсказывать по колебаниям сети, состоящей из пяти невесомых ребер, соединенных общим узлом и закрепленных на границе, такие параметры как масса, точка падения, а также ребро, на которое упала масса. Входящими параметрами являются собственная частота и пять углов отклонения на границах сети.

В ходе выполнения работы, была описана математическая модель сети. Было найдено решение полученной математической модели.

Зная решение, были построены таблицы данных для обучения нейронных сетей с использованием языков программирования Python и C++. Данные были предварительно обработаны для улучшения качества данных и лучшей обучаемости нейронных сетей.

После подготовки данных настал этап обучения нейронной сети для решения обратной задачи колебания. На данном шаге задача свелась к 2 нейронным сетям, одна из которых предсказывает массу и точку падения, другая номер ребра. Первая нейронная сеть хорошо предсказывала значения, но вторая не справлялась со своей задачей. Она предсказывала значения, но с небольшой точностью. Причиной являлось то, что нейронная сеть плохо понимает дискретные значения, и когда на вход подаются параметры с непрерывными значения, то на выходе он также ожидает непрерывные. Тогда было решено свести задачу к написанию программы, которая будет определять номер ребра исходя из входных данных. После ее успешной реализации работа была завершена.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 I. Goodfellow, Y. Bengio, & A. Courville, Deep Learning // MIT Press. — 2016 — С.775.
2 Материалы из интернета — GDPR (General Data Protection Regulation). URL: https://gdpr-info.eu/
3 C. M. Bishop, Pattern Recognition and Machine Learning // Springer. — 2006 — С.738.
4 A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow // O'Reilly Media. — 2019 — С.856.
5 Материалы из интернета — The Pandas Development Team. Pandas Documentation. URL: https://pandas.pydata.org/docs/
6 Wes McKinney, Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython // O'Reilly Media. — 2012 — С.466.
7 J. VanderPlas, Python Data Science Handbook: Essential Tools for Working with Data // O'Reilly Media. — 2016 — С.548.
8 S.Raschka,V.Mirjalili,Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2 // 3rd Edition. Packt Publishing. — 2019 — С.770.
9 Daniel Y.Chen,S.Guido, Pandas for Everyone: Python Data Analysis // Pearson. — 2018 — С.548.
10 Материалы из интернета — TensorFlow Team. TensorFlow Documentation. URL: https://www.tensorflow.org/docs
11 Aurélien. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems // O'Reilly Media. — 2019 — С.856.
12 Материалы из интернета — scikit-learn Developers. Scikit-learn Documentation. URL: https://scikit-learn.org/stable/documentation.html
13 Материалы из интернета — Keras Team. Keras Documentation. URL: https://keras.io

Приложения А

Код для генерации таблицы данных на языке программирования Python

```
import numpy as np
import pandas as pd
import random
import math
df = pd.DataFrame()
F = 0
V = 1
L1 = 10
L2 = 10
L3 = 10
L4 = 10
L5 = 10
T1 = 5
T2 = 5
T3 = 5
T4 = 5
T5 = 5
for i in range(250000):
    string = random.randint(1,5)
    m = random.uniform(5,15)
    S = random.uniform(0,10)
    t = random.uniform(0,20)

    if string == 1:
        D = (T2 / L2) + (T3 / L3) + (T4 / L4) + (T5 / L5) - (T1 / (L1 - S))
        W = math.sqrt(abs((T1 / m) * (1 / S + 1/(L1-S) + T1 / ((L1 - S) * D))))
        A = -V/(W)
        X = A*math.cos((W)*t+F)
        U = -(T1*X)/((L1-S)*D)
        alpha = math.atan(U/L2)
        betta = math.atan(X/S)
        gamma = math.atan(U/L5)
        delta = math.atan(U/L4)
        epsilon = math.atan(U/L3)
    elif string == 2:
        D = (T1 / L1) + (T3 / L3) + (T4 / L4) + (T5 / L5) - (T2 / (L2 - S))
        W = math.sqrt(abs((T2 / m) * (1 / S + 1/(L2-S) + T2 / ((L2 - S) * D))))
        A = -V/(W)
        X = A*math.cos((W)*t+F)
        U = -(T2*X)/((L2-S)*D)
        alpha = math.atan(X/S)
        betta = math.atan(U/L1)
        gamma = math.atan(U/L5)
        delta = math.atan(U/L4)
        epsilon = math.atan(U/L3)
    elif string == 3:
        D = (T1 / L1) + (T2 / L2) + (T4 / L4) + (T5 / L5) - (T3 / (L3 - S))
        W = math.sqrt(abs((T3 / m) * (1 / S + 1/(L3-S) + T3 / ((L3 - S) * D))))
        A = -V/(W)
        X = A*math.cos((W)*t+F)
        U = -(T3*X)/((L3-S)*D)
        alpha = math.atan(U/L2)
        betta = math.atan(U/L1)
        gamma = math.atan(U/L5)
        delta = math.atan(U/L4)
        epsilon = math.atan(X/S)
    elif string == 4:
        D = (T1 / L1) + (T3 / L3) + (T2 / L2) + (T5 / L5) - (T4 / (L4 - S))
        W = math.sqrt(abs((T4 / m) * (1 / S + 1/(L4-S) + T4 / ((L4 - S) * D))))
        A = -V/(W)
        X = A*math.cos((W)*t+F)
        U = -(T4*X)/((L4-S)*D)
        alpha = math.atan(U/L2)
        betta = math.atan(U/L1)
        gamma = math.atan(U/L5)
        delta = math.atan(X/S)
        epsilon = math.atan(U/L3)
    elif string == 5:
        D = (T1 / L1) + (T3 / L3) + (T4 / L4) + (T2 / L2) - (T5 / (L5 - S))
        W = math.sqrt(abs((T5 / m) * (1 / S + 1/(L5-S) + T5 / ((L5 - S) * D))))
        A = -V/(W)
        X = A*math.cos((W)*t+F)
        U = -(T5*X)/((L5-S)*D)
        alpha = math.atan(U/L2)
        betta = math.atan(U/L1)
        gamma = math.atan(X/S)
        delta = math.atan(U/L4)
        epsilon = math.atan(U/L3)
    data_to_append = {'mass':[m],'frequency':[W],'lenght1':[L1],'lenght2':[L2],'lenght3':[L3],'lenght4':[L4],'lenght5':[L5],'tension1':[T1],'tension2':[
    df_to_append = pd.DataFrame(data_to_append)
    df = pd.concat([df, df_to_append], ignore_index=True)
```

Продолжение А

Код для генерации таблицы данных на языке программирования C++

```cpp
#include <iostream>
#include <cmath>
#include <random>
#include <vector>
#include <fstream>

int main() {
    std::locale::global(std::locale(""));
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<double> dis(0.0, 1.0);

    /*const double pi = std::acos(-1.0);*/
    double v = 1.0;
    double L1 = 10.0, L2 = 10.0, L3 = 10.0, L4 = 10.0, L5 = 10.0;
    double T1 = 5.0, T2 = 5.0, T3 = 5.0, T4 = 5.0, T5 = 5.0;

    std::vector<std::vector<double>> data;

    for (int i = 0; i < 1000000; ++i) {
        double string = std::round(dis(gen) * 4.0) + 1;
        double m = dis(gen) * 10 + 5;
        double S = dis(gen) * 10.0;
        double t = dis(gen) * 20;

        double D, W, A, x, U, alpha, beta, gamma, delta, epsilon;

        if (string == 1.0) {
            D = (T2 / L2) + (T3 / L3) + (T4 / L4) + (T5 / L5) - (T1 / (L1 - S));
            W = std::sqrt(std::abs((T1 / m) * (1 / S + 1 / (L1 - S) + T1 / ((L1 - S) * D))));
            A = -v / W;
            x = A * std::cos(W * t + 0);
            U = -(T1 * x) / ((L1 - S) * D);
            alpha = std::atan(U / L2);
            beta = std::atan(x / S);
            gamma = std::atan(U / L5);
            delta = std::atan(U / L4);
            epsilon = std::atan(U / L3);
        }
        else if (string == 2.0) {
            D = (T1 / L1) + (T3 / L3) + (T4 / L4) + (T5 / L5) - (T2 / (L2 - S));
            W = std::sqrt(std::abs((T2 / m) * (1 / S + 1 / (L2 - S) + T2 / ((L2 - S) * D))));
            A = -v / W;
            x = A * std::cos(W * t + 0);
            U = -(T2 * x) / ((L2 - S) * D);
            alpha = std::atan(x / S);
            beta = std::atan(U / L1);
            gamma = std::atan(U / L5);
            delta = std::atan(U / L4);
            epsilon = std::atan(U / L3);
        }
        else if (string == 3.0) {
            D = (T1 / L1) + (T2 / L2) + (T4 / L4) + (T5 / L5) - (T3 / (L3 - S));
            W = std::sqrt(std::abs((T3 / m) * (1 / S + 1 / (L3 - S) + T3 / ((L3 - S) * D))));
            A = -v / W;
            x = A * std::cos(W * t + 0);
            U = -(T3 * x) / ((L3 - S) * D);
            alpha = std::atan(U / L2);
            beta = std::atan(U / L1);
            gamma = std::atan(U / L5);
            delta = std::atan(U / L4);
            epsilon = std::atan(x / S);
        }
        else if (string == 4.0) {
            D = (T1 / L1) + (T3 / L3) + (T2 / L2) + (T5 / L5) - (T4 / (L4 - S));
            W = std::sqrt(std::abs((T4 / m) * (1 / S + 1 / (L4 - S) + T4 / ((L4 - S) * D))));
            A = -v / W;
            x = A * std::cos(W * t + 0);
            U = -(T4 * x) / ((L4 - S) * D);
            alpha = std::atan(U / L2);
            beta = std::atan(U / L1);
            gamma = std::atan(U / L5);
            delta = std::atan(x / S);
            epsilon = std::atan(U / L3);
        }
        else if (string == 5.0) {
            D = (T1 / L1) + (T3 / L3) + (T4 / L4) + (T2 / L2) - (T5 / (L5 - S));
            W = std::sqrt(std::abs((T5 / m) * (1 / S + 1 / (L5 - S) + T5 / ((L5 - S) * D))));
            A = -v / W;
            x = A * std::cos(W * t + 0);
            U = -(T5 * x) / ((L5 - S) * D);
            alpha = std::atan(U / L2);
            beta = std::atan(U / L1);
            gamma = std::atan(U / L5);
            delta = std::atan(U / L4);
            epsilon = std::atan(x / S);
        }
        std::vector<double> row{ m, W,S, string, alpha, beta, gamma, delta, epsilon };
        data.push_back(row);
    }
    /*for (const auto& row : data) {
        std::cout << "m:" << row[0] << "fr:" << row[1]
          << "pos:" << row[12] << "str:" << row[13]
          << "a:" << row[15] << "b:" << row[16] << "g:" << row[17]
          << "d:" << row[18] << "e:" << row[19] << std::endl;
    }*/

    // Открываем файл для записи данных
    std::ofstream file("datafinish2.txt");

    // Проверяем, открылся ли файл успешно
    if (file.is_open()) {
        // Записываем данные в файл
        for (const auto& row : data) {
            for (size_t i = 0; i < row.size(); ++i) {
                // Записываем каждый элемент строки, разделяя точной запятой
                file << row[i];
                // Добавляем точку запятую после каждого элемента, кроме последнего
                if (i < row.size() - 1) {
                    file << ";";
                }
            }
            // Переходим на новую строку после каждой строки данных
            file << "\n";
        }
        // Закрываем файл после записи данных
        file.close();
        std::cout << "Данные успешно записаны в файл datafinish.txt" << std::endl;
    }
    else {
        std::cerr << "Ошибка при открытии файла для записи" << std::endl;
    }
    return 0;
}
```

Продолжение А

Фрагмент кода, который выполняет операцию изменения типа данных из object в float

```
[11]: df['mass'] = df['mass'].astype(float)
[12]: df['fre'] = df['fre'].astype(float)
[13]: df['pos'] = df['pos'].astype(float)
[14]: df['alfa'] = df['alfa'].astype(float)
[15]: df['betta'] = df['betta'].astype(float)
[16]: df['gama'] = df['gama'].astype(float)
[17]: df['delta'] = df['delta'].astype(float)
[18]: df['epsilon'] = df['epsilon'].astype(float)
[19]: df.to_csv('dataform1.csv', index=False)
[20]: df = pd.read_csv('dataform1.csv')
[21]: df.info()
```

Нормализация входных данных для первой нейронной сети

```
input_df2 = df[['alfa','betta','gama','delta','epsilon']]
input_df1 = df[['fre']]
# Создание объекта
scaler = MinMaxScaler(feature_range=(-1.5,1.5))
# Подгонка и преобразование данных
input_df_scaled1 = pd.DataFrame(scaler.fit_transform(input_df1), columns=input_df1.columns)
input_df_scaled = pd.concat([ input_df_scaled1,input_df2],axis=1)
input_df_scaled
```

Нормализация выходных данных для первой нейронной сети

```
output_df = df[['mass','pos']]
scaler = MinMaxScaler(feature_range=(-1.5,1.5))
# Подгонка и преобразование данных
output_df_scaled = pd.DataFrame(scaler.fit_transform(output_df), columns=output_df.columns)
output_df_scaled
```

Архитектура первой нейронной сети

```
output_train = output_df_scaled[500000:]
input_train = input_df_scaled[500000:]

input_test = input_df_scaled[:500000]
output_test = output_df_scaled[:500000]
```

```
model = tf.keras.Sequential()
model.add(keras.layers.Dense(units=6, activation='tanh'))
model.add(keras.layers.Dense(units=6, activation='tanh'))
model.add(keras.layers.Dense(units=2, activation='tanh'))
model.add(keras.layers.Dense(units=2, activation='tanh'))
model.compile(optimizer='Adam', loss='mean_squared_error', metrics=['accuracy'])
```

```
history = model.fit(input_train,output_train, epochs=10)
```

Тестирование модели на тестовых данных

```
model.evaluate(input_test,output_test)
```

```
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 8s 504us/step - accuracy: 0.9085 - loss: 0.1735
[0.1744109243154258, 0.9075400233268738]
```

```
predict_data = model.predict(input_test)
```

```
15625/15625 ━━━━━━━━━━━━━━━━━━━━ 9s 583us/step
```

Создание диаграммы рассеяния

```
hb = plt.hexbin(df2['pos'],df1[1], gridsize=50, cmap='viridis', mincnt=1)
cb = plt.colorbar(hb)
cb.set_label('Плотность')
plt.title('Диаграмма рассеяния с изменением цвета')
plt.xlabel('pos')
plt.ylabel('pred_pos')
plt.show()
```

Продолжение А

Архитектура второй нейронной сети

```
output_train = output_df_scaled[500000:]
input_train = input_df[500000:]

input_test = input_df[:500000]
output_test = output_df_scaled[:500000]
```

```
model = tf.keras.Sequential()
model.add(keras.layers.Dense(units=5, activation = 'relu'))
model.add(keras.layers.Dense(units=4, activation = 'tanh'))
model.add(keras.layers.Dense(units=1, activation = 'relu'))
model.compile(optimizer='Adam',loss='mean_squared_error',metrics=['accuracy'])
```

```
fit_train = model.fit(input_train,output_train, epochs=10)
```

Код для определения нити на которую упала масса

```
def check_values(row):
    if row['alfa'] != row['betta'] or row['alfa'] != row['gama'] or row['alfa'] != row['epsilon'] or row['alfa'] != row['delta']:
        if row['alfa'] != row['betta']:
            return 2
        elif row['betta'] != row['gama']:
            return 1
        elif row['gama'] != row['epsilon']:
            return 5
        elif row['delta'] != row['epsilon']:
            return 4
        elif row['epsilon'] != row['delta']:
            return 3
    return None
```

```
new_df = pd.DataFrame(columns=['string'])
```

```
new_df['string'] = input_df.apply(check_values, axis=1)
```

```
new_df
```

Подключение всех нужных библиотек

```
import pandas as pd
import math
import sklearn
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import random
import matplotlib.pyplot as plt
import seaborn as sns
```