МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
ИМЕНИ ПАТРИСА ЛУМУМБЫ»

Факультет физико-математических и естественных наук

Кафедра математического моделирования и искусственного интеллекта

«Допустить к защите»

Заведующий кафедрой
математического моделирования
и искусственного интеллекта
д.ф.-м.н., доцент
_____ М.Д. Малых
«____»_____ 20__г.

Выпускная квалификационная работа бакалавра

Направление 02.03.01 «Математика и компьютерные науки»

ТЕМА: «Обучение нейронных сетей для аппроксимации решений краевых задач с применением неклассических вариационных формулировок»

Выполнил студент **Габриэль Тьерри**
(Фамилия, имя, отчество)

Группа НКНбд-01-20

Студ. билет № 10322204249

Руководитель выпускной
квалификационной работы
Шорохов С.Г. к.ф.-м.н., проф
(Ф.И.О., степень, звание, должность)

_____
(Подпись)

Автор _____
(Подпись)

г. Москва

2024 г.

Федеральное государственное автономное образовательное учреждение
высшего образования
«Российский университет дружбы народов имени Патриса Лумумбы»

АННОТАЦИЯ

выпускной квалификационной работы

Габриэль Тьерри

(фамилия, имя, отчество)

на тему: Обучение нейронных сетей для аппроксимации решений краевых задач с применением неклассических вариационных формулировок

Данная работа исследует новый подход к решению краевых задач через обучение нейронных сетей с использованием нетрадиционных вариационных формулировок. Она показывает, что традиционные методы часто сталкиваются со сложностью и скоростью. Нейронные сети обучаются на данных и законах физики. Мы уделили внимание решению волновых уравнений. Наш метод использует Нейронные сети, называемые физически-информированными нейронными сетями (PINNs). Эти нейронные сети применяют особый процесс обучения. Они учитывают правильные решения и законы физики. Мы проверили наш подход, сравнив его с известными решениями. Результаты очень обнадеживают. Наша нейронная сеть точно предсказывает решения волнового уравнения. Этот метод обладает большим потенциалом для изменения решений очень сложных математических задач. Казалось бы, он может привести к более быстрым и простым решениям краевых задач.

Автор ВКР _____ _____

(Подпись) (ФИО)

Содержание

Обозначения

API: Application Programming Interface

BVP: Boundary Value Problem

FEM: Finite Element Method

FONS: Field-Optimized Neural Substructure

LSTM: Long Short-Term Memory

MAE: Mean Absolute Error

MCR: Monte Carlo Simulation

ML: Machine Learning

MSE: Mean Squared Error

NN: Neural Network

np: NumPy

NPV: Net Present Value

PDE: Partial Differential Equation

PINNs: Physics-Informed Neural Networks

plt: Matplotlib

RNNs: Recurrent Neural Networks

tf: TensorFlow

Введение

Основной целью данного исследования является проверка возможности использования нейронной сети для решения сложной математической задачи, моделируемой краевой задачей, а также эффективности такого подхода при наиболее точном решении задачи и расчете наиболее эффективного процесса. Для этой работы я использовал среду Tensorflow и язык python для обучения нейронной сети аппроксимации решения уравнения в частных производных с помощью метода нейронной сети, основанной на физике (PINNS).

Актуальность работы

Мы рассматриваем новые способы использования нейронных сетей при решении краевых задач неклассическими методами, что сегодня очень актуально, поскольку технологии развиваются очень высокими темпами. По мере развития технологий в каждой области требуются новые, быстрые и оперативные способы расчета. Нейронные сети как часть искусственного интеллекта уже хорошо развиты для удовлетворения этих потребностей. Такая научная и практическая работа в большей степени направлена на совершенствование обычных методов решения таких задач, которые имеют проблемы со сложностью или плохим масштабированием. Тип обучения, который будет проводиться для этих нейронных сетей, будет наилучшим и наиболее эффективным, поскольку он имеет очень большое значение для обучения, проводимого в таких областях, как инженерия и физика. Такой метод, разработанный в этой работе, с практической точки зрения может быть очень полезен для отраслей, в которых важно быстро и правильно решить граничную задачу, например, в машиностроении и финансах. Для такого рода деятельности важно

иметь быстрые и надежные решения из-за необходимости немедленного анализа данных и принятия решений.

Нейронные сети применялись в большинстве предыдущих исследований для решения дифференциальных уравнений, но в большинстве случаев они не полностью учитывали законы физики или не использовали неклассические подходы, как это было в данной статье. Несмотря на то, что теоретические основы были заложены в классических работах фундаментального характера, таких как работы Филипповой и др. (1992) [4] и более поздних работах Сириньяно и Спилиопулоса (2018) [2] а также [5], они еще не применялись на практике. Тема была выбрана в связи с личным и профессиональным интересом автора к вопросу о том, действительно ли передовые технологии искусственного интеллекта могут быть применимы для решения классических математических задач. Это вдохновило меня на развитие этого исследования и поиск новых решений и идей, которые еще не были отработаны в данной области, но которые могли бы революционизировать методы обработки сложных вычислений в науке и технике. Эта статья основана на критической литературе в этой области и дальнейшем использовании и развитии фундаментальной идеи, согласно которой Ядава и др. (2015) [3] рассматривают нейронные сети в контексте дифференциальных уравнений. Эта статья основана на концепции, которая была разработана для того, чтобы проиллюстрировать, как предлагаемые методы применяются на практике, и протестировать их на соответствие известным решениям, показывая, насколько инновационным является это исследование.

Цель работы

Целью данной работы является изучение инновационного подхода, который использует нейронные сети, основанные на физике, для более эффективного решения краевых задач. Интегрируя математические и физические законы непосредственно в обучение этих сетей, этот метод позволяет получать более быстрые и точные решения, чем традиционные вычислительные методы.

Задачи работы

Для достижения цели, в работе решаются следующие задачи:

1. Исследование создания вариационной формулировки уравнений колебаний конечной струны
2. Разработка архитектуры нейронной сети (PINN), которая включает в себя слои, подходящие для понимания и обработки динамики краевых задач, такие как слои долговременной кратковременной памяти (LSTM) и плотные слои.
3. Внедрение специального процесса обучения, который корректировал веса сети в зависимости от ее эффективности в прогнозировании точных решений и соблюдении физических законов, используя методы оптимизации на основе градиента.
4. Использование визуального метода для отображения прогнозов нейронной сети и сравнения их с точными решениями обеспечивает четкую визуальную оценку точности нейронной сети и качества ее аппроксимаций

Методы исследования

В работе были использованы следующие методы исследования:

метод решения волнового уравнения с использованием неклассического принципа.

метод применения архитектуры (PINN) [1] в обучающей модели.

Структура работы

Работа состоит из введения, трех глав, заключения, списка литературы и приложения.

- В первой главе был сделан обзор общего введения в дифференциальные уравнения и представлены методы построения неклассических вариационных постановок краевых задач для уравнений математической физики.
- Вторая глава посвящена выбору архитектуры нейронной сети для аппроксимации решений краевой задачи.
- В третьей главе мы изучаем основные подходы к обучению нейронной сети для аппроксимации решений краевой задачи
- В заключении описываются результаты и делаются выводы о проделанной работе, а также предлагаются способы использования результатов этого исследования.
- В приложении приведены полные перечни программ, написанных в ходе подготовки выпускной квалификационной работы.

1 Введение в дифференциальное уравнение

В этом разделе я расскажу об основных понятиях дифференциальных уравнений. Дифференциальные уравнения являются одним из основных и очень мощных инструментов для описания динамических систем, которые используются в физике, химии, биологии и других науках. Исследователю предоставляется возможность разрабатывать модели, которые по существу имитируют процессы, в которых происходят изменения различных величин во времени и пространстве. Таким образом, знание основных особенностей и типов дифференциальных уравнений, порядка, степени и различий между обыкновенными и дифференциальными уравнениями в частных производных позволяет исследователю правильно ставить задачи и, соответственно, решать их, возникающие в прикладных науках и технике. Классификация уравнений в соответствии с линейностью и типом доступных граничных условий очень полезна при выборе подходящих методов решения, что является важным шагом в процессе нахождения успешного приближения к решениям. Только изучение базовых понятий дифференциальных уравнений может быть полезным для создания прочной теоретической базы, необходимой для наилучшего понимания и разработки неклассических вариационных постановок краевых задач. Этот подход позволяет гораздо более адекватно моделировать сложные физические системы и их взаимодействия, что может быть весьма важно для современной науки и техники.

Определение дифференциального уравнения

Дифференциальное уравнение - это, как мы все знаем, математические выражения, которые связывают функцию с ее производными. Уравнение, учи-

тывающее скорость изменения. Производные описывают скорость изменения функции по отношению к одной или нескольким переменным. Дифференциальные уравнения используются для описания физических явлений таким образом, что приходится рассматривать величины в соотношении с другими величинами, которые постоянно меняются. Дифференциальные уравнения используются для анализа решений, удовлетворяющих уравнениям, и для изучения свойств решений.

В чем польза дифференциального уравнения?

Наиболее важной частью всего дифференциального уравнения является возможность нахождения такой функции во всей области, которая описывает экспоненциальный рост или затухание функции со временем. Это дает очень хорошее преимущество, заключающееся в способности предсказывать окружающую нас природу.

1.1 Структура дифференциального уравнения

Порядок дифференциального уравнения

Применяемый в большинстве областей, таких как физика, химия, биология и экономика, порядок дифференциального уравнения имеет наивысшую производную, присутствующую в дифференциальном уравнении. Короче говоря, это говорит вам о том, насколько глубоки изменения; вот почему в большинстве уравнений, чем выше порядок, тем сложнее поведение или система, которые вы пытаетесь описать.

1. **Дифференциальное уравнение первого порядка**

$$\frac{dy}{dx} + y = e^x$$

Это уравнение включает первую производную от y, делая его дифференциальным уравнением первого порядка.

2. **Дифференциальное уравнение второго порядка**

$$\frac{d^2y}{dx^2} - 3\frac{dy}{dx} + 2y = 0$$

Это уравнение включает вторую производную от y, делая его дифференциальным уравнением второго порядка.

3. **Дифференциальное уравнение третьего порядка**

$$\frac{d^3y}{dx^3} + 3\frac{d^2y}{dx^2} - y = \sin(x)$$

Наличие третьей производной от y делает это дифференциальное уравнение третьего порядка.

Степень дифференциального уравнения

Степень дифференциального уравнения относится к степени старшей производной, когда исходное уравнение представлено в виде уравнения с полиномиальной производной. Проще говоря, степень указывает нам высший порядок дифференцирования в нашем дифференциальном уравнении. Это мера сложности уравнения, основанная на показателе степени старшей производной, в то время как все четко и без помех со стороны этих математических операций.

1. **Дифференциальное уравнение первой степени**

$$\frac{d^2y}{dx^2} + \frac{dy}{dx} - y = 0$$

Наивысшая производная ($\frac{d^2y}{dx^2}$) возведена в первую степень, что делает это уравнение уравнением первой степени.

2. **Дифференциальное уравнение второй степени**

$$\left(\frac{d^2y}{dx^2}\right)^2 + \frac{dy}{dx} = x$$

Здесь квадрат второй производной делает это дифференциальное уравнение уравнением второй степени.

3. **Дифференциальное уравнение третьей степени**

$$\left(\frac{dy}{dx}\right)^3 - y = 0$$

Первая производная возведена в куб, что делает уравнение уравнением третьей степени.

Порядок дает представление о сложности уравнения с точки зрения операций исчисления, в то время как степень дает представление о алгебраической сложности уравнения.

1.2 Классификация дифференциальных уравнений

Дифференциальные уравнения можно классифицировать различными способами на основе их характеристик, которые включают природу вовлеченных производных, тип содержащихся в них функций и их линейность.

1. Обыкновенные дифференциальные уравнения (ОДУ) / дифференциальные уравнения с частными производными (ДУЧП)

- **Обыкновенные дифференциальные уравнения (ОДУ)** включают функции одной независимой переменной и их производные. Пример:

$$\frac{dy}{dx} + y = e^x$$

- **дифференциальные уравнения с частными производными (ДУЧП)** включают функции нескольких независимых переменных и их частные

производные. Пример:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

2. Линейные / нелинейных дифференциальных уравнений

- **Линейные дифференциальные уравнения**, где зависимая переменная и ее производные появляются линейно. Пример:

$$a_n(x)\frac{d^n y}{dx^n} + \cdots + a_1(x)\frac{dy}{dx} + a_0(x)y = g(x)$$

- **Нелинейные дифференциальные уравнения** включают нелинейные члены зависимой переменной или ее производных. Пример:

$$y\frac{d^2 y}{dx^2} + (\frac{dy}{dx})^2 = 0$$

3. Однородные / неоднородных дифференциальных уравнений

- **Однородные дифференциальные уравнения** имеют ноль с одной стороны уравнения, когда все члены перенесены на одну сторону. Они зависят только от зависимой переменной и ее производных.

- **Неоднородные дифференциальные уравнения** содержат члены, которые являются функциями независимых переменных или констант, помимо зависимой переменной и ее производных.

4. Автономные / неавтономных дифференциальных уравнений

- **Автономные дифференциальные уравнения** не включают явно независимую переменную. Пример:

$$\frac{dy}{dx} = f(y)$$

- **Неавтономные дифференциальные уравнения** явно зависят от независимой переменной. Пример:

$$\frac{dy}{dx} = xy + e^x$$

1.3 Дифференциальные уравнения в физике

Дифференциальное уравнение - это наиболее общий инструмент физики для описания и прогнозирования поведения физической системы во времени или пространстве. На самом деле, это достигается за счет того, что он позволяет физикам писать формулы, говорящие: "Если мы знаем, как выглядит эта система и как она меняется сейчас, мы можем абсолютно точно предсказать, что произойдет дальше". Эта компетенция делает его довольно универсальным для решения широкого спектра реальных физических явлений.

Моделирование физических систем

Волновые уравнения: Дифференциальные уравнения также описывают волны, будь то звуковые волны, водные волны или световые волны. Волновое уравнение помогает нам понять, как волны распространяются и взаимодействуют со своей средой.

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$$

уравнения колебаний струны

Волновое уравнение имеет решающее значение в физике для моделирования динамических явлений, где энергия передается через колебания в среде. Оно применяется к звуковым волнам, электромагнитным волнам, сейсмическим

волнам и другим. Классический пример, демонстрирующий принципы волнового уравнения, — это колеблющаяся струна.

Рассмотрим струну, натянутую между двумя фиксированными точками. При возмущении струна колеблется, и эти колебания распространяются вдоль нее. Поведение этих колебаний может быть описано волновым уравнением, специально адаптированным для одномерной струны под напряжением:

$$\frac{\partial^2 y}{\partial t^2} = c^2 \frac{\partial^2 y}{\partial x^2}$$

где $y(x,t)$ представляет смещение струны от равновесия в позиции x и времени t, а c — скорость волны на струне, определяемая как:

$$c = \sqrt{\frac{T}{\mu}}$$

Здесь T — напряжение в струне, а μ — линейная массовая плотность.

Волновое уравнение для колеблющейся струны выводится, рассматривая силы на малом элементе струны, применяя второй закон Ньютона. Общее решение включает синусоидальные функции, выраженные как:

$$y(x,t) = \sum_{n=1}^{\infty} \left(A_n \cos(\omega_n t) + B_n \sin(\omega_n t)\right) \sin\left(\frac{n\pi x}{L}\right)$$

с параметрами:

- L — длина струны,
- A_n и B_n — константы, определяемые начальными условиями,
- $\omega_n = \frac{n\pi c}{L}$ — угловая частота n-го режима,

- n — положительное целое число.

Струна имеет фиксированные концы, что приводит к граничным условиям:

$$y(0,t) = 0 \quad \text{и} \quad y(L,t) = 0$$

Эти условия обеспечивают нулевое смещение на обоих концах струны во все времена.

Физические следствия и применения

- **Музыкальные инструменты:** В гитарах и скрипках звук в основном производится за счет колебаний струн. Тембр звука зависит от возбужденных гармоник.
- **Инженерия:** Понимание колебаний струн помогает в проектировании конструкций с кабелями и в разработке различных датчиков и исполнительных механизмов.
- **Образование:** Эта проблема широко используется для обучения частным дифференциальным уравнениям, рядам Фурье и задачам с граничными значениями в физике и математике.

Применение волнового уравнения к колеблющимся струнам проливает свет на ряд физических явлений и подчеркивает важность волновой механики в различных научных и инженерных дисциплинах.

1.4 Неклассические вариационные принципы

Неклассические вариационные формулировки для задач, обусловленных границами, являются обобщениями классических вариационных методов для сложных граничных условий, задач с нелокальными членами или когда ограничения или направляющие условия не являются строго локальными. Нелокальные вариационные формулировки могут быть необходимы для физических систем с особыми свойствами или взаимодействиями, например, для неоднородных, анизотропных материалов или систем с особыми физическими взаимодействиями.

Базовые понятия

Классическая вариационная формулировка - это определение экстремума для представления энергии системы. Эти вариационные формулировки могут содержать производные более высокого порядка, нелокальные члены или нестандартные граничные условия.

Формулирование проблемы

Для определения неклассической вариационной формулировки следуйте этим шагам:

Спецификация функционала

Определите функционал, который должен быть экстремизирован, который может включать нелокальные члены или вклады от границы:

$$J[u] = \int_\Omega F(x, u, \nabla u, \nabla^2 u, \ldots)\, dx + \int_{\partial\Omega} G(x, u, \nabla u, \ldots)\, ds$$

где Ω — домен, а $\partial\Omega$ — граница.

Граничные и начальные условия

Укажите граничные и начальные условия, которые могут включать производные или интегральные ограничения:

$$u|_{\partial\Omega} = g(x), \quad \frac{\partial u}{\partial n}|_{\partial} = h(x), \quad \int_{\partial\Omega} u\, ds = C$$

Уравнения Эйлера-Лагранжа

Получены уравнения Эйлера-Лагранжа для функционала, которые могут включать производные высших порядков или нелокальные члены

Применение этих формулировок обычно требует численных методов, таких как методы конечных элементов, которые могут потребовать адаптации для работы с неклассическими членами.

Неклассические вариационные формулировки позволяют более точно описывать физические явления в системах, где традиционные предположения не выполняются, обеспечивая надежную основу для решения сложных задач с граничными условиями в математической физике.

1.5 Обзор численных методов решения дифференциальных уравнений в частных производных

Численные методы решения дифференциальных уравнений в частных производных необходимы во многих областях, где аналитические решения невозможны. Ключевые методы включают методы конечных разностей (MCR),

которые аппроксимируют производные на сетке путем преобразования NPV в алгебраические уравнения; методы конечных элементов (FEM), которые делят область на дискретные элементы, идеально подходят для сложных геометрий; методы конечных объемов (MCR), которые сохраняют потоки через контрольные объемы, являются подходит для применения в соответствии с законами сохранения; спектральные методы, использующие преобразования для получения высокоточных решений; и метод прямых (ML), который дискретизирует пространственную область для преобразования метода расчета в обыкновенные дифференциальные уравнения. В последнее время методы, основанные на нейронных сетях, особенно на физических нейронных сетях (FONS), получили широкое распространение благодаря их способности аппроксимировать сложные многомерные решения дифференциальных уравнений путем интеграции физических законов в процесс обучения. Этот инновационный подход обещает повысить точность и эффективность численного моделирования, возможно, переосмыслив традиционные методы в будущем. В следующей главе мы подробно рассмотрим применение нейронных сетей для решения дифференциальных уравнений.

2 Нейронные сети для решения дифференциальных уравнений

2.1 Введение в нейронные сети и дифференциальные уравнения

Дифференциальные уравнения играют важную роль в моделировании различных процессов, изменяющихся во времени или пространстве, во многих областях математики, инженерии и естественных наук. Они могут описывать, например, рост населения или движение жидкости. Во многих случаях уравнения такого типа идеально решаются аналитически. Однако в реальности многие реальные задачи не могут быть решены таким образом, поэтому их необходимо решать приближенно численно. Новым перспективным направлением в решении таких сложных задач является применение нейронных сетей. Для этого необходимо сначала понять основы работы нейронных сетей.

История о нейронной сети

Концепция нейронных сетей имеет богатую историю, которая началась в середине 20-го века и была вдохновлена желанием создать машины, способные имитировать человеческий интеллект. Нейронные сети - это упрощенная, искусственная версия человеческого мозга, предназначенная для обучения на основе опыта. Впервые они были изобретены в 1940-х годах, вдохновленные тем, как работают нейроны в нашем мозге. Основная идея возникла в 1950-х годах, когда появилось устройство, называемое персептроном, - это была очень ранняя модель, способная запоминать простые паттерны. Ситуация изменилась в 1980-х годах, когда исследователи выяснили, как объединить эти перцептроны в так называемые многослойные сети, что позволило им изучать гораздо более сложные вещи. Этому способствовали усовершенство-

ванные методы обучения и более мощные компьютеры. Сегодня нейронные сети превратились в то, что мы называем глубоким обучением, позволяя использовать такие технологии, как распознавание речи в смартфонах, помогая врачам диагностировать заболевания и даже управлять автономными автомобилями. Они продолжают становиться умнее, изучая все новые и новые технологии.

Что такое нейронная сеть?

Нейронная сеть - это вычислительная модель, описывающая структуру и функции человеческого мозга. Как правило, она состоит из наборов взаимосвязанных узлов, называемых искусственными нейронами, сгруппированных в слои, которые обрабатывают и передают информацию на пути к получению конечного результата. Если бы целью было имитировать процесс принятия решений человеческим мозгом, то именно это и делает нейронная сеть. Это вычислительная система, которая пытается изучить основные взаимосвязи, лежащие в основе набора данных. При этом кто-то пытается смоделировать процесс работы человеческого мозга.

Основные принципы работы нейронных сетей

Она состоит из слоев взаимосвязанных узлов, или "нейронов каждый из которых выполняет простые вычисления. Данные, поступающие в сеть, проходят через эти уровни, и каждый узел выполняет математическое преобразование. Выходные данные каждого узла зависят от этих преобразований и силы связей (весов) между узлами, которые настраиваются во время обучения, чтобы свести к минимуму разницу между выходными данными сети и желаемым результатом.

2.2 Ключевые компоненты архитектуры нейронной сети

Давайте рассмотрим различные компоненты нейрона, чтобы сделать их более понятными. Таким образом, мы можем рассматривать его как крошечную технологическую единицу, очень похожую на рабочего на сборочной линии, каждый из которых выполняет небольшую задачу, чтобы внести свой вклад в достижение большего результата.

1. **Входной слой:** Этот слой принимает необработанные данные. Каждый нейрон во входном слое представляет одну из характеристик входных данных. Представьте, что нейрон - это человек, готовый принимать посылки (входные данные). Каждая посылка имеет вес, который указывает на ее важность. Некоторые посылки являются тяжелыми (важные исходные данные), а некоторые - легкими (менее важные исходные данные).

2. **Веса и смещения:** это параметры нейронной сети, которые изучаются в процессе обучения. Каждое соединение между нейронами имеет соответствующий вес, а каждый нейрон - соответствующее смещение. Вес каждой посылки регулирует влияние входных данных. В нейроне это числовые значения, которые масштабируют получаемые входные данные. Если посылка (входные данные) тяжелая (имеют большой вес), это означает, что она более важна для вычислений, которые будет выполнять нейрон. Это постоянное значение, добавляемое к сумме взвешенных входных данных и определяющее начальную точку активации.

3. **Скрытые слои:** это слои между входным и выходным слоями. Нейронные сети могут иметь один или несколько скрытых слоев. Каждый слой состоит из нейронов, которые принимают значения из предыдущего слоя,

применяют вес и смещение, а затем передают результат через функцию активации. Функция активации вносит нелинейность в выходные данные нейрона, что позволяет нейронным сетям моделировать сложные взаимосвязи.

4. **Функции активации:** Эти функции применяются к выходным сигналам нейронов для придания модели нелинейных свойств. Общие функции активации включают сигмовидную мышцу, tanh и ReLU (выпрямленную линейную единицу). Рассматривайте это как процесс принятия решений. На основе общего веса от суммированных входных данных и смещения функция активации определяет, насколько активным должен быть нейрон. Это может быть похоже на принятие решения о том, кричать ли, говорить нормально или шептать, в зависимости от важности полученного сообщения. Общие функции активации включают sigmoid (которая выбирает между двумя состояниями, очень похожими на "да"/ "нет"), tanh (которая может выбирать диапазон от отрицательного до положительного) и ReLU (которая решает, является ли сообщение достаточно важным, чтобы его вообще передавать).

5. **Прямое распространение:** На этом этапе данные перемещаются с входного уровня через скрытые слои на выходной уровень, где делаются прогнозы.

6. **Обратное распространение:** Это фаза обучения, на которой нейронная сеть обновляет свои параметры (веса и смещения). Сеть вычисляет ошибку в своих прогнозах (разницу между прогнозируемыми и фактическими значениями), и эта ошибка распространяется обратно по сети, позволяя обновлять веса с помощью методов оптимизации, таких как градиентный

спуск.

7. **Функция потерь:** Эта функция измеряет, насколько хорошо работает нейронная сеть. Она вычисляет разницу между фактическими и прогнозируемыми результатами. Общие функции потерь включают в себя среднеквадратичную ошибку для задач регрессии и потерю перекрестной энтропии для задач классификации. Корректируя веса и смещения с помощью обучения, нейронная сеть учится делать все более точные прогнозы. Архитектура и глубина сети могут быть адаптированы к конкретным приложениям, начиная от простых задач, таких как линейная регрессия, и заканчивая сложными задачами, такими как распознавание изображений, обработка естественного языка и многое другое.

8. **Функция суммирования:** нейрон суммирует все взвешенные входные данные, аналогично суммированию общего веса всех полученных посылок. Эта общая сумма будет определять следующее действие нейрона.

9. **Выходной сигнал:** Наконец, нейрон посылает свой выходной сигнал, который может быть сигналом другим нейронам, расположенным ниже по потоку в сети. Количество нейронов в выходном слое зависит от типа решаемой задачи (например, один нейрон для бинарной классификации, несколько для мультиклассовой классификации или несколько для регрессии). Это похоже на передачу решения или фрагмента обработанной информации. Чтобы наглядно представить это, представьте себе группу работников (нейронов), участвующих в эстафете. Каждый работник получает набор взвешенных посылок (входных данных), суммирует их с учетом личных предпочтений, принимает решение (активирует) и затем передает результат следующему работнику в очереди. Этот процесс

продолжается до тех пор, пока не будет принято окончательное решение (вывод сети) на основе всей полученной и обработанной информации.

Стандартные архитектуры нейронных сетей

1. **Персептрон:** Персептрон является самой базовой структурой в нейронных сетях. По сути, это тип нейронной сети, которая получает множество входных данных, выполняет определенные математические процедуры с этими входными данными и генерирует выходные данные. Работая с вектором входных данных с реальным значением, он выполняет линейное смешивание каждого атрибута с заданным весом. Затем эти взвешенные входные данные объединяются в единичное значение, которое затем обрабатывается с помощью функции активации. Затем эти отдельные блоки персептрона интегрируются для создания более обширной структуры искусственной нейронной сети.

2. **Сети прямой связи:** Сети прямой связи подобны прямым каналам передачи информации в нейронных сетях. Это тип архитектуры нейронных сетей, в которой данные перемещаются в одном направлении, от входного уровня через скрытые слои (если таковые имеются) к выходному уровню, без каких-либо циклов обратной связи. Представьте это как улицу с односторонним движением для передачи данных. Каждый слой нейронов в сети получает входные данные от предыдущего слоя, обрабатывает их с помощью ряда математических операций, а затем передает результат на следующий уровень. Этот процесс продолжается до тех пор, пока данные не дойдут до уровня вывода, где генерируется окончательный результат или прогноз. По сути, сети прямой связи предназначены для передачи

данных вперед по сети без каких-либо обратных или циклических подключений.

3. **Рекуррентные нейронные сети (RNNs):** Стандартная настройка для глубокого обучения имеет определенный размер входных данных, который может быть ограничивающим при работе с переменными размерами входных данных. Кроме того, в модели отсутствует память о прошлых входных данных, поэтому каждое решение основывается исключительно на текущих входных данных. Вот тут-то и вступают в игру рекуррентные нейронные сети (RNN). Они превосходно обрабатывают последовательности данных в качестве входных данных, что делает их идеальными для таких задач, как обработка естественного языка (NLP), анализ настроений и фильтрация спама. Они также отлично подходят для решения задач, связанных с временными рядами, таких как прогнозирование продаж и тенденций на фондовом рынке.

4. **Рекуррентные нейронные сети (RNNs):** Рекуррентные нейронные сети (RNN) обладают замечательной способностью сохранять то, чему они научились ранее, и использовать это для прогнозирования будущего. Они являются чемпионами по запоминанию в мире нейронных сетей. Итак, вот как они работают: вы передаете им последовательные данные, например, последовательность шагов или событий, и по мере обработки каждого фрагмента данных они обновляют свое внутреннее состояние, что-то вроде записей в журнале. Затем эта скрытая информация о состоянии передается обратно в модель, что-то вроде перелистывания страниц этого журнала. И на каждом этапе RNN выдает какие-то результаты, основанные на том, что он видел до сих пор. Это похоже на то, что мозг

учится на опыте прошлого, чтобы принимать более разумные решения в будущем!

5. **Сеть долговременной кратковременной памяти (LSTM):** Сети долговременной кратковременной памяти (LSTM) представляют собой специализированный тип рекуррентной архитектуры нейронных сетей, предназначенный для преодоления проблемы исчезающего градиента и эффективного моделирования долгосрочных зависимостей в последовательных данных. Традиционные рекуррентные нейронные сети (RNN) часто испытывают трудности с фиксацией долгосрочных зависимостей из-за уменьшения влияния градиентов во время обучения, что приводит к забыванию более ранней информации. LSTM устраняет это ограничение, внедряя более сложную структуру ячеек памяти, состоящую из различных элементов, включая элементы ввода, забывания и вывода. Эти элементы регулируют поток информации через ячейку, позволяя LSTM выборочно сохранять или отбрасывать информацию с течением времени.

   На каждом временном шаге LSTM получает входные данные и обновляет свое внутреннее состояние на основе текущего входного сигнала, предыдущего состояния и памяти, хранящейся в ячейке. Элемент ввода управляет степенью, в которой новая информация включается в состояние ячейки, элемент забывания регулирует степень, в которой существующая информация сохраняется или отбрасывается, а элемент вывода определяет информацию, которая должна быть выведена из ячейки.

   Ключевое новшество LSTMS заключается в их способности поддерживать постоянный поток ошибок на протяжении всего процесса обучения, тем самым обеспечивая эффективное изучение долгосрочных зависимостей.

Благодаря явному моделированию памяти в сетевой архитектуре LSTM превосходно справляются с задачами, связанными с последовательными данными, такими как обработка естественного языка, распознавание речи и прогнозирование временных рядов.

2.3 Использование нейронных сетей для решения дифференциальных уравнений

Традиционные методы, такие как метод конечных элементов или численное интегрирование, хорошо работают, но часто имеют ограничения, особенно при обработке нелинейных или динамических систем, которые меняются в реальном времени. Нейронные сети предлагают увлекательную альтернативу. Настроив нейронную сеть на минимизацию разницы между её предсказаниями и фактическими поведениями, описываемыми дифференциальными уравнениями, исследователи могут обучить её решать эти уравнения при заданных условиях.

Преимущества метода нейронных сетей для решения дифференциальных уравнений

мы можем перечислить несколько важных преимуществ использования нейросетевых методов для ДУ. Например, это способность быть гибким и обобщенным, поскольку нейронная сеть учится решать задачи на основе общих правил, полученных на основе данных; это позволяет эффективно работать с нерегулярными областями или уникальными граничными условиями. В контексте многомерности нейронные сети лучше справляются с задачами большой размерности благодаря тому, что они не сталкиваются с “проклятием” размерности. Адаптивность нейронных сетей может быть объяснена способ-

ностью адаптировать архитектуру к новым данным или изменениям в динамике системы, что не требует полной перестройки модели. Кроме того, простота интеграции эмпирических данных в процесс принятия решений делает NN идеальным решением для задач моделирования, основанного на данных. Наконец, благодаря способности быстро переключаться между вычислениями решения, хорошо обученная сеть обладает потенциалом для вычислений в реальном времени; она подходит для динамического моделирования в инженерных и активных финансовых стратегиях

Основные функции активации

Тип функции активации является наиболее фундаментальным выбором для нейронной сети; он напрямую влияет на ее поведение и возможности через выходные данные узлов нейронной сети. Во-первых, это линейные функции активации; они являются самыми простыми и в них выходные данные пропорциональны входным данным. Эти функции активации полезны для случаев, когда зависимость линейна; однако они не работают для сложных нелинейных случаев. Функция активации “знак” - это еще одна линейная функция, которая вычисляет и выдает значение -1 или 1; они лучше всего подходят для бинарной классификации, но плохо подходят для решения дифференциальных уравнений из-за резкости изменений. Кроме того, существуют функции активации сигмовидной формы, которые плавно преобразуют входные данные в выходные в диапазоне от 0 до 1, что позволяет наилучшим образом работать с вероятностями, а также обеспечивает постепенное обучение и адаптацию, поскольку это важно для точного решения дифференциальных уравнений. Наконец, однако, пошаговые функции активации, которые имитируют двоичный переключатель и имеют значения 0 или 1 на выходе в зависимости от

количества входных данных по сравнению с пороговым значением, незаменимы для двоичных решений, для других задач они не столь полезны, поскольку нет необходимости в тонком подходе. Нейронные сети представляют собой революционное развитие вычислительных методов для дифференциальных уравнений и мощную альтернативу традиционным численным подходам. Его производные свойства и способность делать обобщения на основе имеющихся данных особенно важны в тех случаях, когда точное решение дифференциального уравнения невозможно из-за его сложности.

2.4 Использование методов глубокого обучения для анализа одномерного волнового уравнения на основе нестандартного вариационного принципа.

Изучим создание вариационной формулировки уравнений колебаний конечной струны.

$$u_{tt} - u_{xx} = 0 \tag{1}$$

в квадратной области $\Omega = [0, \pi] \times [0, \pi]$. Здесь и далее буквенные нижние индексы будут означать дифференцирование по соответствующей переменной.

Граница $\partial\Omega$ области Ω является объединением четырех отрезков $\gamma_1, \gamma_2, \gamma_3, \gamma_4$:

$$\partial\Omega = \gamma_1 \cup \gamma_2 \cup \gamma_3 \cup \gamma_4$$

где отрезки $\gamma_1, \gamma_2, \gamma_3, \gamma_4$ задаются следующими соотношениями

$$\gamma_1 = \{(x,t) \mid 0 \leqslant x \leqslant \pi, t = 0\}$$

$$\gamma_2 = \{(x,t) \mid x = \pi, 0 \leqslant t \leqslant \pi\}$$

$$\gamma_3 = \{(x,t) \mid 0 \leqslant x \leqslant \pi, t = \pi\}$$

$$\gamma_4 = \{(x,t) \mid x = 0, 0 \leqslant t \leqslant \pi\}$$

Зададим следующие граничные условия на $\partial\Omega$:

$$\begin{cases} u|_{\gamma_1} = 0, & 0 \leqslant x \leqslant \pi, t = 0, \\ u_x|_{\gamma_2} = -\sin t, & x = \pi, 0 \leqslant t \leqslant \pi, \\ u_t|_{\gamma_2} = 0, & x = \pi, 0 \leqslant t \leqslant \pi, \\ u_x|_{\gamma_3} = 0, & 0 \leqslant x \leqslant \pi, t = \pi, \\ u_t|_{\gamma_3} = -\sin x, & 0 \leqslant x \leqslant \pi, t = \pi, \\ u_x|_{\gamma_4} = \sin t, & x = 0, 0 \leqslant t \leqslant \pi, \\ u_t|_{\gamma_4} = 0, & x = 0, 0 \leqslant t \leqslant \pi \end{cases}$$

Непосредственной подстановкой можно убедиться, что уравнению (1) с граничными условиями (2) удовлетворяет функция

$$u(x,t) = \sin x \sin t = \frac{1}{2}(\cos(x-t) - \cos(x+t)) \tag{3}$$

Построим для краевой задачи (1)-(2) неклассический вариационный принцип, следуя подходу В.М.Шалова (В. М. Шалов, "Принцип минимума квадратичного функционала для гиперболического уравнения Дифференц. уравнения, 1:10 (1965), 1338-1365).

Метод построения вариационной формулировки для волнового уравнения, предложенный В.М.Шаловым, состоит в построении двух векторных операторов $\mathbf{A}$ и $\mathbf{B}$, таких, что оператор $\mathbf{A}$ является $\mathbf{B}$-симметричным:

$$\langle \mathbf{A}u, \mathbf{B}v \rangle = \langle \mathbf{B}u, \mathbf{A}v \rangle \forall u, v$$

и B-положительным:

$$\langle \mathbf{A}u, \mathbf{B}u \rangle > 0 \forall u \neq 0$$

$\langle \mathbf{A}u_n, \mathbf{B}u_n \rangle \to 0, n \to \infty \Rightarrow \|u_n\| \to 0, n \to \infty,$

причем краевая задача для уравнения колебаний струны должна записываться в виде

$$\mathbf{A}u = \mathbf{f}$$

тогда функционал для вариационной задачи имеет вид

$$D[u] = \langle \mathbf{A}u, \mathbf{B}u \rangle - 2\langle \mathbf{f}, \mathbf{B}u \rangle$$

Следуя подходу В.М. Шалова, перейдем от независимых переменных (x, t) к независимым переменным (ξ, η), где

$$\xi = x + t, \eta = x - t$$

в которых уравнение колебаний струны (волновое уравнение) (1) примет вид

$$u_{\xi\eta} = 0 \tag{4}$$

точное частное решение (3) примет вид:

$$u(\xi, \eta) = \frac{1}{2}(\cos\eta - \cos\xi) \tag{5}$$

область Ω примет вид ромба Ω' с вершинами в точках $\Gamma'_0(0,0), \Gamma'_1(\pi,\pi)$, $\Gamma'_2(2\pi,0), \Gamma'_3(\pi,-\pi)$.

Обратный переход от переменных (ξ,η) к переменным (x,t) задается формулами

$$x = \frac{1}{2}(\xi+\eta), t = \frac{1}{2}(\xi-\eta)$$

Граница $\partial\Omega'$ области Ω' состоит из четырех отрезков $\gamma'_1, \gamma'_2, \gamma'_3, \gamma'_4$:

$$\partial\Omega' = \gamma'_1 \cup \gamma'_2 \cup \gamma'_3 \cup \gamma'_4$$

где отрезок γ'_1 соединяет точки $\Gamma'_0(0,0)$ и $\Gamma'_1(\pi,\pi)$, отрезок γ'_2 соединяет точки $\Gamma'_1(\pi,\pi)$ и $\Gamma'_2(2\pi,0)$, отрезок γ'_3 соединяет точки $\Gamma'_2(2\pi,0)$ и $\Gamma'_3(\pi,-\pi)$, отрезок γ'_4 соединяет точки $\Gamma'_3(\pi,-\pi)$ и $\Gamma'_0(0,0)$.

Для рассматриваемой нами области Ω' компоненты внешней нормали $\vec{n} = (n_1, n_2)$ к границе $\partial\Omega'$ на различных участках границы $\gamma'_1, \gamma'_2, \gamma'_3, \gamma'_4$ вычисляются по формулам:

$$\vec{n}\,(\gamma'_1) = \frac{1}{\sqrt{2}}(-1,1), \vec{n}\,(\gamma'_2) = \frac{1}{\sqrt{2}}(1,1), \vec{n}\,(\gamma'_3) = \frac{1}{\sqrt{2}}(1,-1), \vec{n}\,(\gamma'_4) = \frac{1}{\sqrt{2}}(-1,-1)$$

Построим векторные операторы $\mathbf{A}$ и $\mathbf{B}$ следующим образом. Первую компоненту векторного оператора $\mathbf{A}$, действующую на Ω', определим формулой

$$(\mathbf{A}u)_1 = u_{\xi\eta}$$

Соответствующую первую компоненту вспомогательного (симметризующего) векторного оператора $\mathbf{B}$ на Ω' определим равенством

$$(\mathbf{B}v)_1 = \int_{\xi}^{\gamma_1' \cup \gamma_4'} v_\eta(\zeta, \eta) d\zeta + \int_{\eta}^{\gamma_1' \cup \gamma_2'} v_\xi(\xi, \tau) d\tau$$

Здесь точка $(\xi, \eta) \in \Omega'$, интегрирование по переменной ζ ведется при фиксированном значении η от значения ξ до соответствующего значения на одной из кривых γ_1' или γ_4' (в зависимости от значения η), а интегрирование по переменной τ ведется при фиксированном значении ξ от значения η до соответствующего значения на одной из кривых γ_1' или γ_2'.

Вторые компоненты векторных операторов $\mathbf{A}$ и $\mathbf{B}$, действующие на отрезке границы γ_1', определим как тождественные отображения:

$$(\mathbf{A}u)_2 = u, (\mathbf{B}v)_2 = v$$

Третьи компоненты векторных операторов $\mathbf{A}$ и $\mathbf{B}$, действующие на отрезке границы γ_2', определим формулами:

$$(\mathbf{A}u)_3 = u_\eta, (\mathbf{B}v)_3 = -n_1(\gamma_2') \int_{\xi}^{\gamma_1'} v_\eta(\zeta, \eta) d\zeta$$

Четвертые компоненты векторных операторов $\mathbf{A}$ и $\mathbf{B}$, действующие на отрезке границы γ_3', определим формулами:

$$(\mathbf{A}u)_4 = u_\eta, (\mathbf{B}v)_4 = -n_1(\gamma_3') \int_{\xi}^{\gamma_4'} v_\eta(\zeta, \eta) d\zeta$$

Пятые компоненты векторных операторов $\mathbf{A}$ и $\mathbf{B}$, также действующие на отрезке границы γ_3', определим формулами:

$$(\mathbf{A}u)_5 = u_\xi, (\mathbf{B}v)_5 = -n_2(\gamma_3') \int_{\eta}^{\gamma_2'} v_\xi(\xi, \tau) d\tau$$

Последние, шестые компоненты векторных операторов A и B, действующие на отрезке границы γ_4', определим формулами:

$$(\mathbf{A}u)_6 = u_\xi, (\mathbf{B}v)_6 = -n_2\left(\gamma_4'\right)\int_\eta^{\gamma_1'} v_\xi(\xi,\tau)d\tau$$

Итак, векторные операторы A и B равны

$$\mathbf{A}u = \begin{bmatrix} u_{\xi\eta} \\ u \\ u_\eta \\ u_\eta \\ u_\xi \\ u_\xi \end{bmatrix}, \mathbf{B}v = \begin{bmatrix} \int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta)d\zeta + \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau)d\tau \\ v \\ -n_1\left(\gamma_2'\right)\int_\xi^{\gamma_1'} v_\eta(\zeta,\eta)d\zeta \\ -n_1\left(\gamma_3'\right)\int_\xi^{\gamma_4'} v_\eta(\zeta,\eta)d\zeta \\ -n_2\left(\gamma_3'\right)\int_\eta^{\gamma_2'} v_\xi(\xi,\tau)d\tau \\ -n_2\left(\gamma_4'\right)\int_\eta^{\gamma_1'} v_\xi(\xi,\tau)d\tau \end{bmatrix}$$

Векторный оператор $\mathbf{A}$ является дифференциальным и определен на декартовом произведении $\Omega'\times\gamma_1'\times\gamma_2'\times\gamma_3'\times\gamma_3'\times\gamma_4'$, векторный оператор $\mathbf{B}$ является интегродифференциальным и также определен на $\Omega'\times\gamma_1'\times\gamma_2'\times\gamma_3'\times\gamma_3'\times\gamma_4'$.

Также для построения вариационной формулировки требуется вектор-функция $\mathbf{f}$, также определенная на $\Omega'\times\gamma_1'\times\gamma_2'\times\gamma_3'\times\gamma_3'\times\gamma_4'$, и равная

$$\mathbf{f} = \frac{1}{2}\sin\xi\begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

Скалярное произведение векторов $\mathbf{A}u$ и $\mathbf{B}v$ вычисляется путем перемножения соответствующих компонент, вычисления интегралов по соответствующим множествам и суммирования полчившихся значений согласно формуле

$$\langle \mathbf{A}u, \mathbf{B}v\rangle = \int_{\Omega'} (\mathbf{A}u)_1(\mathbf{B}v)_1 d\xi d\eta + \int_{\gamma_1'} (\mathbf{A}u)_2(\mathbf{B}v)_2 ds + \int_{\gamma_2'} (\mathbf{A}u)_3(\mathbf{B}v)_3 ds +$$
$$+ \int_{\gamma_3'} (\mathbf{A}u)_4(\mathbf{B}v)_4 ds + \int_{\gamma_3'} (\mathbf{A}u)_5(\mathbf{B}v)_5 ds + \int_{\gamma_4'} (\mathbf{A}u)_6(\mathbf{B}v)_6 ds$$

Вычислим первый интеграл $\int_{\Omega'}(\mathbf{A}u)_1(\mathbf{B}v)_1 d\xi d\eta$, используя формулу интегрирования по частям для многомерного случая:

$$\int_{\Omega'} (\mathbf{A}u)_1(\mathbf{B}v)_1 d\xi d\eta = \int_{\Omega'} u_{\xi\eta} \left[\int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta + \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau \right] d\xi d\eta$$

$$\int_{\Omega'} u_{\xi\eta} \int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta d\xi d\eta = \int_{\Omega'} (u_\eta)_\xi \int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta d\xi d\eta$$

$$= \int_{\partial\Omega'} u_\eta \int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta n_1 ds - \int_{\Omega'} u_\eta \left(\int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta \right)_\xi d\xi d\eta$$

$$= \int_{\partial\Omega'} u_\eta \int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta n_1 ds + \int_{\Omega'} u_\eta v_\eta d\xi d\eta \qquad (1)$$

$$\int_{\Omega'} u_{\xi\eta} \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau d\xi d\eta = \int_{\Omega'} (u_\xi)_\eta \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau d\xi d\eta$$

$$= \int_{\partial\Omega'} u_\xi \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau n_2 ds - \int_{\Omega'} u_\xi \left(\int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau \right)_\eta d\xi d\eta$$

$$= \int_{\partial\Omega'} u_\xi \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau n_2 ds + \int_{\Omega'} u_\xi v_\xi d\xi d\eta$$

Здесь n_1, n_2 - компоненты внешней нормали $\vec{n} = (n_1, n_2)$ к границе $\partial\Omega'$ области Ω'. При вычислении криволинейных интегралов по контуру $\partial\Omega'$ внутренний интеграл $\int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta$ обращается в нуль на $\gamma_1' \cup \gamma_4'$, а внутренний интеграл $\int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau$ - на $\gamma_1' \cup \gamma_2'$, поэтому получаем

$$\int_{\Omega'}(\mathbf{A}u)_1(\mathbf{B}v)_1 d\xi d\eta = \int_{\Omega'}\left(u_\xi v_\xi + u_\eta v_\eta\right) d\xi d\eta + \int_{\gamma_2'\cup\gamma_3'} u_\eta \int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta n_1 ds + \\ + \int_{\gamma_3'\cup\gamma_4'} u_\xi \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau n_2 ds$$

Преобразуем приведенные выше криволинейные интегралы по кривым $\gamma_2' \cup \gamma_3'$ и $\gamma_3' \cup \gamma_4'$:

$$\int_{\gamma_2'\cup\gamma_3'} u_\eta \int_\xi^{\gamma_1'\cup\gamma_4'} v_\eta(\zeta,\eta) d\zeta n_1 ds + \int_{\gamma_3'\cup\gamma_4'} u_\xi \int_\eta^{\gamma_1'\cup\gamma_2'} v_\xi(\xi,\tau) d\tau n_2 ds = \\ \int_{\gamma_2'} u_\eta \int_\xi^{\gamma_1'} v_\eta(\zeta,\eta) d\zeta n_1 ds + \int_{\gamma_3'} u_\eta \int_\xi^{\gamma_4'} v_\eta(\zeta,\eta) d\zeta n_1 ds + \\ \int_{\gamma_3'} u_\xi \int_\eta^{\gamma_2'} v_\xi(\xi,\tau) d\tau n_2 ds + \int_{\gamma_4'} u_\xi \int_\eta^{\gamma_1'} v_\xi(\xi,\tau) d\tau n_2 ds = \\ \int_{\gamma_2'} u_\eta \int_\xi^{\gamma_1'} v_\eta(\zeta,\eta) d\zeta n_1 ds + \int_{\gamma_3'} \left[u_\eta \int_\xi^{\gamma_4'} v_\eta(\zeta,\eta) d\zeta n_1 + u_\xi \int_\eta^{\gamma_2'} v_\xi(\xi,\tau) d\tau n_2 \right] ds + \\ \int_{\gamma_4'} u_\xi \int_\eta^{\gamma_1'} v_\xi(\xi,\tau) d\tau n_2 ds \tag{2}$$

Итак, окончательно получаем

$$\langle \mathbf{A}u, \mathbf{B}v\rangle = \int_{\Omega'} (u_\xi v_\xi + u_\eta v_\eta)\, d\xi d\eta + \int_{\gamma_2'} u_\eta \int_\xi^{\gamma_1'} v_\eta(\zeta,\eta) d\zeta n_1 ds +$$
$$+ \int_{\gamma_3'} \left[u_\eta \int_\xi^{\gamma_4'} v_\eta(\zeta,\eta) d\zeta n_1 + u_\xi \int_\eta^{\gamma_2'} v_\xi(\xi,\tau) d\tau n_2 \right] ds +$$
$$+ \int_{\gamma_4'} u_\xi \int_\eta^{\gamma_1'} v_\xi(\xi,\tau) d\tau n_2 ds +$$
$$+ \int_{\gamma_1'} uv ds - \int_{\gamma_2'} u_\eta n_1 \int_\xi^{\gamma_1'} v_\eta(\zeta,\eta) d\zeta ds - \int_{\gamma_3'} u_\eta n_1 \int_\xi^{\gamma_4'} v_\eta(\zeta,\eta) d\zeta ds -$$
$$- \int_{\gamma_3'} u_\xi n_2 \int_\eta^{\gamma_2'} v_\xi(\xi,\tau) d\tau ds - \int_{\gamma_4'} u_\xi n_2 \int_\eta^{\gamma_1'} v_\xi(\xi,\tau) d\tau ds \tag{3}$$

В этом выражении все криволинейные интегралы, кроме интеграла по γ_1', сокращаются и формула для скалярного произведения приобретает следующий простой вид:

$$\langle \mathbf{A}u, \mathbf{B}v\rangle = \int_{\Omega'} (u_\xi v_\xi + u_\eta v_\eta)\, d\xi d\eta + \int_{\gamma_1'} uv ds$$

из которого вытекает B-симметричность оператора A. Далее имеем

$$\langle \mathbf{A}u, \mathbf{B}u\rangle = \int_{\Omega'} \left(u_\xi^2 + u_\eta^2\right) d\xi d\eta + \int_{\gamma_1'} u^2 ds$$

откуда, очевидно, вытекает вытекает B-положительность оператора $\mathbf{A}$.

Функционал для вариационной задачи принимает вид

$$D[u] = \langle \mathbf{A}u, \mathbf{B}u\rangle - 2\langle \mathbf{f}, \mathbf{B}u\rangle = \int_{\Omega'} \left(u_\xi^2 + u_\eta^2\right) d\xi d\eta + \int_{\gamma_1'} u^2 ds +$$
$$+ \int_{\gamma_2'} n_1 \sin\xi \int_\xi^{\gamma_1'} u_\eta(\zeta,\eta) d\zeta ds - \int_{\gamma_3'} n_1 \sin\xi \int_\xi^{\gamma_4'} u_\eta(\zeta,\eta) d\zeta ds +$$
$$+ \int_{\gamma_3'} n_2 \sin\xi \int_\eta^{\gamma_2'} u_\xi(\xi,\tau) d\tau ds + \int_{\gamma_4'} n_2 \sin\xi \int_\eta^{\gamma_1'} u_\xi(\xi,\tau) d\tau ds$$

Такой вид функционала неудобен для использования в машинном обучении, поэтому преобразуем повторные интегралы в двойные.

Область Ω' можно рассматривать как объединение четырех подобластей $\Omega'_1, \Omega'_2, \Omega'_3, \Omega'_4$:

$$\Omega' = \Omega'_1 \cup \Omega'_2 \cup \Omega'_3 \cup \Omega'_4$$

Перейдем в криволинейных интегралах к переменным ξ, η и интегрированию по подобластям $\Omega'_1, \Omega'_2, \Omega'_3, \Omega'_4$ с учетом того, что дифференциал длины дуги равен $ds = \sqrt{1 + (\eta_\xi)^2} d\xi = \sqrt{2} d\xi$ или $ds = \sqrt{1 + (\xi_\eta)^2} d\eta = \sqrt{2} d\eta$ и компоненты внешней нормали $\vec{n} = (n_1, n_2)$ на участках границы $\partial\Omega'$ вычисляются следующим образом:

$$\vec{n}\,(\gamma'_1) = \frac{1}{\sqrt{2}}(-1, 1), \vec{n}\,(\gamma'_2) = \frac{1}{\sqrt{2}}(1, 1), \vec{n}\,(\gamma'_3) = \frac{1}{\sqrt{2}}(1, -1), \vec{n}\,(\gamma'_4) = \frac{1}{\sqrt{2}}(-1, -1)$$

При вычислении криволинейного интеграла вдоль γ'_2 используем параметризацию по переменной η, а именно, $\xi = 2\pi - \eta, 0 \leqslant \eta \leqslant \pi, ds = \sqrt{2} d\eta$, тогда

$$\int_{\gamma'_2} \sin\xi n_1\,(\gamma'_2) \int_\xi^{\gamma'_1} u_\eta(\zeta, \eta) d\zeta ds = \int_0^\pi \sqrt{2} \sin(2\pi - \eta) \frac{1}{\sqrt{2}} \int_{2\pi-\eta}^{\gamma'_1} u_\eta(\zeta, \eta) d\zeta d\eta =$$
$$= \int_0^\pi \sin\eta \int_{2\pi-\eta}^{\eta} u_\eta(\xi, \eta) d\xi d\eta = \int_0^\pi \int_\eta^{2\pi-\eta} \sin\eta u_\eta(\xi, \eta) d\xi d\eta =$$
$$= \int_{\Omega'_1 \cup \Omega'_2} \sin\eta u_\eta(\xi, \eta) d\xi d\eta$$

При вычислении первого криволинейного интеграла вдоль γ'_3 также используем параметризацию по переменной η, а именно, $\xi = \eta + 2\pi, -\pi \leqslant \eta \leqslant$

$0, ds = \sqrt{2}d\eta$, тогда

$$-\int_{\gamma_3'} \sin\xi n_1(\gamma_3') \int_{\xi}^{\gamma_4'} u_\eta(\zeta,\eta)d\zeta ds = -\int_{-\pi}^{0} \sqrt{2}\sin(\eta+2\pi)\frac{1}{\sqrt{2}}$$
$$\int_{\eta+2\pi}^{\gamma_4'} u_\eta(\zeta,\eta)d\zeta d\eta = \int_{-\pi}^{0} \sin\eta \int_{\eta+2\pi}^{-\eta} u_\eta(\xi,\eta)d\xi d\eta =$$
$$= \int_{-\pi}^{0}\int_{-\eta}^{\eta+2\pi} \sin\eta u_\eta(\xi,\eta)d\xi d\eta = \int_{\Omega_3'\cup\Omega_4'} \sin\eta u_\eta(\xi,\eta)d\xi d\eta$$

При вычислении второго криволинейного интеграла вдоль γ_3' используем параметризацию по переменной ξ, а именно, $\eta = \xi - 2\pi, \pi \leqslant \xi \leqslant 2\pi, ds = \sqrt{2}d\xi$, тогда

$$\begin{aligned}\int_{\gamma_3'} \sin\xi n_2(\gamma_3') \int_{\eta}^{\gamma_2'} u_\xi(\xi,\tau)d\tau ds &= \int_{\pi}^{2\pi} \sqrt{2}\sin\xi\frac{-1}{\sqrt{2}}\int_{\xi-2\pi}^{\gamma_2'} u_\xi(\xi,\tau)d\tau d\xi \\ &= -\int_{\pi}^{2\pi}\sin\xi\int_{\xi-2\pi}^{2\pi-\xi} u_\xi(\xi,\eta)d\eta d\xi \\ &= -\int_{\pi}^{2\pi}\int_{\xi-2\pi}^{2\pi-\xi}\sin\xi u_\xi(\xi,\eta)d\eta d\xi \\ &= \int_{\Omega_2'\cup\Omega_3'}\sin\xi u_\xi(\xi,\eta)d\xi d\eta\end{aligned} \tag{4}$$

При вычислении криволинейного интеграла вдоль γ_4' также используем параметризацию по переменной ξ, а именно, $\eta = -\xi, 0 \leqslant \xi \leqslant \pi, ds = \sqrt{2}d\xi$, тогда

$$\begin{aligned}\int_{\gamma_4'} \sin\xi n_2\left(\gamma_4'\right) \int_{\eta}^{\gamma_1'} u_\xi(\xi,\tau)d\tau ds &= \int_0^{\pi} \sqrt{2}\sin\xi \frac{-1}{\sqrt{2}} \int_{-\xi}^{\gamma_1'} u_\xi(\xi,\tau)d\tau d\xi \\ &= -\int_0^{\pi} \sin\xi \int_{-\xi}^{\xi} u_\xi(\xi,\eta)d\eta d\xi \\ &= -\int_0^{\pi} \int_{-\xi}^{\xi} \sin\xi u_\xi(\xi,\eta)d\eta d\xi \\ &= -\int_{\Omega_1'\cup\Omega_4'} \sin\xi u_\xi(\xi,\eta)d\xi d\eta\end{aligned} \tag{5}$$

Итак, получаем в переменных ξ, η функционал для вариационной задачи

$$D[u] = \int_{\Omega'} \left(u_\xi^2 + u_\eta^2\right) d\xi d\eta + \int_{\gamma_1'} u^2 ds + \int_{\Omega_1'\cup\Omega_2'} \sin\eta u_\eta(\xi,\eta)d\xi d\eta +$$
$$+ \int_{\Omega_3'\cup\Omega_4'} \sin\eta u_\eta(\xi,\eta)d\xi d\eta - \int_{\Omega_2'\cup\Omega_3'} \sin\xi u_\xi(\xi,\eta)d\xi d\eta - \int_{\Omega_1'\cup\Omega_4'} \sin\xi u_\xi(\xi,\eta)d\xi d\eta =$$
$$= \int_{\Omega'} \left(u_\xi^2 + u_\eta^2\right) d\xi d\eta + \int_{\gamma_1'} u^2 ds + \int_{\Omega'} \sin\eta u_\eta(\xi,\eta)d\xi d\eta - \int_{\Omega'} \sin\xi u_\xi(\xi,\eta)d\xi d\eta =$$
$$= \int_{\Omega'} \left(u_\xi^2 + u_\eta^2 + \sin\eta u_\eta - \sin\xi u_\xi\right) d\xi d\eta + \int_{\gamma_1'} u^2 ds$$

Вернемся в первоначальные переменные (x,t). Согласно формуле замены переменных в двойном интеграле

$$\int_{\Omega'} f(\xi,\eta)d\xi d\eta = \int_{\Omega} f(\xi(x,t),\eta(x,t)) \left|\frac{\partial(\xi,\eta)}{\partial(x,t)}\right| dxdt =$$
$$= \int_{\Omega} f(x+t,x-t) \left|\det\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\right| dxdt = 2\int_{\Omega} f(x+t,x-t)dxdt$$
$$u_\xi = \frac{\partial}{\partial\xi} u(x(\xi,\eta),t(\xi,\eta)) = u_x\frac{\partial x}{\partial\xi} + u_t\frac{\partial t}{\partial\xi} = \frac{1}{2}\left(u_x + u_t\right)$$
$$u_\eta = \frac{\partial}{\partial\eta} u(x(\xi,\eta),t(\xi,\eta)) = u_x\frac{\partial x}{\partial\eta} + u_t\frac{\partial t}{\partial\eta} = \frac{1}{2}\left(u_x - u_t\right)$$

Поэтому получаем в исходных переменных (x,t) следующий функционал

$$D[u] = \int\limits_{\Omega} 2\left(\frac{1}{4}(u_x+u_t)^2 + \frac{1}{4}(u_x-u_t)^2 + \frac{1}{2}\sin(x-t)(u_x-u_t) - \frac{1}{2}\sin(x+t)(u_x+u_t)\right) dxdt +$$

$$+\int\limits_{\gamma_1} u^2 ds = \int\limits_{\Omega} \left(u_x^2 + u_t^2 + u_x(\sin(x-t) - \sin(x+t)) - u_t(\sin(x-t) + \sin(x+t))\right) dxdt + \int\limits_{\gamma_1} u^2 ds =$$

$$= \int\limits_{\Omega} \left(u_x^2 + u_t^2 - 2u_x \cos x \sin t - 2u_t \sin x \cos t\right) dxdt + \int\limits_{\gamma_1} u^2 ds$$

Нейросетевая аппроксимация решения краевой задачи для уравнения колебаний струны

Рассмотрим краевую задачу (1)-(2) для уравнения колебаний струны, допускающую неклассический (неэйлеров) вариационный принцип с функционалом

$$D[u] = \int\limits_{\Omega} \left(u_x^2 + u_t^2 - 2u_x \cos x \sin t - 2u_t \sin x \cos t\right) dxdt + \int\limits_{\gamma_1} u^2 ds$$

$$\begin{cases} u\big|_{\gamma_1} = 0, & 0 \leqslant x \leqslant \pi, t = 0, \\ u_x\big|_{\gamma_2} = -\sin t, & x = \pi, 0 \leqslant t \leqslant \pi, \\ u_t\big|_{\gamma_2} = 0, & x = \pi, 0 \leqslant t \leqslant \pi, \\ u_x\big|_{\gamma_3} = 0, & 0 \leqslant x \leqslant \pi, t = \pi, \\ u_t\big|_{\gamma_3} = -\sin x, & 0 \leqslant x \leqslant \pi, t = \pi, \\ u_x\big|_{\gamma_4} = \sin t, & x = 0, 0 \leqslant t \leqslant \pi, \\ u_t\big|_{\gamma_4} = 0, & x = 0, 0 \leqslant t \leqslant \pi \end{cases}$$

Решение $u(x,t)$ этой краевой задачи может быть аппроксимировано глубокой искусственной нейронной сетью с выходом $f(x,t;\boldsymbol{\theta})$, где x,t - входные значения, а $\boldsymbol{\theta}$ - вектор параметров (весов и смещений) нейронной сети:

$$u(x,t) \approx f(x,t;\boldsymbol{\theta})$$

При обучении нейронной сети в качестве функционала потерь (ошибки) может быть использован функционал невязки (residuals functional)

$$\mathcal{L}_R(f) = \|f_{tt} - f_{xx}\|_\Omega^2 + \|f\|_{\gamma_1}^2 + \|f_x + \sin t\|_{\gamma_2}^2 + \|f_t\|_{\gamma_2}^2 +$$
$$+ \|f_x\|_{\gamma_3}^2 + \|f_t + \sin x\|_{\gamma_3}^2 + \|f_x - \sin t\|_{\gamma_4}^2 + \|f_t\|_{\gamma_4}^2$$

где $\|f\|_X^2 = \int_X |f(x)|^2 \rho(x) dx, \rho(x)$ - плотность некоторого распределения вероятности на X (например, равномерного распределения).

При использовании такого функционала в качестве функционала потерь (ошибки) при обучении нейронной сети, аппроксимирующей решение рассматриваемой краевой задачи, интегралы, входящие в функционал, как правило, подсчитываются методом интегрирования Монте-Карло, а производные f_x, f_t и т.д. неизвестной функции $f(x,t)$ вычисляются при помощи алгоритма автоматического дифференцирования.

Метод интегрирования Монте-Карло позволяет приближенно вычислять интегралы

$$I = \int_\Omega f(\mathbf{x}) \rho(\mathbf{x}) d\mathbf{x}$$

где $\rho(\mathbf{x})$ - некоторое вероятностное распределение в области Ω, в виде конечных сумм

$$\hat{I} = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_i)$$

где $\{\mathbf{x}_i\}_{i=1}^n$ - случайная выборка с распределением $\rho(\mathbf{x})$. Такая аппроксимация является корректной и дает полезные результаты, так как

- по сильному закону больших чисел $\hat{I} \underset{n\to\infty}{\longrightarrow} I$ с вероятностью единица
- ошибку аппроксимации можно измерить и контролировать:

$$\mathrm{Var}(\hat{I}) = \frac{1}{n(n-1)} \sum_{i=1}^{n} \left(f\left(\mathbf{x}_i\right) - \hat{I} \right)^2$$

При использовании функционала невязки $\mathcal{L}_R(f)$ нужно вычислять восемь интегралов, для чего необходимо строить случайные выборки из различных областей (многообразий), и вычислять частные производные до второго порядка включительно.

Общая схема алгоритма обучения нейронной сети с функционалом невязки

1. Выбрать начальное множество параметров нейронной сети $\boldsymbol{\theta}_0$ и начальную скорость обучения α_0
2. Сгенерировать случайные выборки для области Ω и временной/пространственной границы $\partial\Omega$, а именно

- сгенерировать случайную выборку $\{(x_{i_0}, t_{i_0})\}_{i_0=1}^{n_0}$ из n_0 точек из области Ω с распределением ν_0
- сгенерировать случайную выборку $\{(x_{i_1}, t_{i_1})\}_{i_1=1}^{n_1}$ из n_1 точек из фрагмента границы $\gamma_1 \subset \partial\Omega$ с распределением ν_1
- сгенерировать случайную выборку $\{(x_{i_2}, t_{i_2})\}_{i_2=1}^{n_2}$ из n_2 точек из фрагмента границы $\gamma_2 \subset \partial\Omega$ с распределением ν_2
- сгенерировать случайную выборку $\{(x_{i_3}, t_{i_3})\}_{i_3=1}^{n_3}$ из n_3 точек из фрагмента границы $\gamma_3 \subset \partial\Omega$с распределением ν_3

- сгенерировать случайную выборку $\{(x_{i_4}, t_{i_4})\}_{i_4=1}^{n_4}$ из n_4 точек из фрагмента границы $\gamma_4 \subset \partial\Omega$ с распределением ν_4

3. Вычислить функционал невязки $\mathcal{L}_R(f)$ для сгенерированных случайных выборок, объединенных в мини-батч s_k :

$$\mathbf{s}_k = \left\{ \{(x_{i_0}, t_{i_0})\}_{i_0=1}^{n_0}, \{(x_{i_1}, t_{i_1})\}_{i_1=1}^{n_1}, \{(x_{i_2}, t_{i_2})\}_{i_2=1}^{n_2}, \{(x_{i_3}, t_{i_3})\}_{i_3=1}^{n_3}, \{(x_{i_4}, t_{i_4})\}_{i_4=1}^{n_4} \right\}$$

$$L_0(f, \mathbf{s}_k; \boldsymbol{\theta}_k) = \frac{1}{n_0} \sum_{i_0=1}^{n_0} \left(f_{tt}(x_{i_0}, t_{i_0}; \boldsymbol{\theta}_n) - f_{xx}(x_{i_0}, t_{i_0}; \boldsymbol{\theta}_n) \right)^2$$

$$L_1(f, \mathbf{s}_k; \boldsymbol{\theta}_k) = \frac{1}{n_1} \sum_{i_1=1}^{n_1} \left(f(x_{i_1}, t_{i_1}; \boldsymbol{\theta}_n) \right)^2$$

$$L_2(f, \mathbf{s}_k; \boldsymbol{\theta}_k) = \frac{1}{n_2} \sum_{i_2=1}^{n_2} \left(\left(f_x(x_{i_2}, t_{i_2}; \boldsymbol{\theta}_n) + \sin t_{i_2} \right)^2 + \left(f_t(x_{i_2}, t_{i_2}; \boldsymbol{\theta}_n) \right)^2 \right)$$

$$L_3(f, \mathbf{s}_k; \boldsymbol{\theta}_k) = \frac{1}{n_3} \sum_{i_3=1}^{n_3} \left(\left(f_x(x_{i_3}, t_{i_3}; \boldsymbol{\theta}_n) \right)^2 + \left(f_t(x_{i_3}, t_{i_3}; \boldsymbol{\theta}_n) + \sin x_{i_3} \right)^2 \right)$$

$$L_4(f, \mathbf{s}_k; \boldsymbol{\theta}_k) = \frac{1}{n_4} \sum_{i_4=1}^{n_4} \left(\left(f_x(x_{i_4}, t_{i_4}; \boldsymbol{\theta}_n) - \sin t_{i_4} \right)^2 + \left(f_t(x_{i_4}, t_{i_4}; \boldsymbol{\theta}_n) \right)^2 \right)$$

$$\mathcal{L}_R(f, \mathbf{s}_k; \boldsymbol{\theta}_k) = \sum_{j=0}^{4} L_j(f, \mathbf{s}_k; \boldsymbol{\theta}_k)$$

4. Выполнить один или несколько шагов градиентного спуска с мини-батчем (случайными точками) $\mathbf{s}_k$ при помощи адаптивного алгоритма Adam (или иного алгоритма обучения нейронной сети) со скоростью обучения α_k (скорость обучения α_k обновляется автоматически на каждом шаге):

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla_{\boldsymbol{\theta}} \mathcal{L}_R(f, \mathbf{s}_k; \boldsymbol{\theta}_k)$$

5. Повторять шаги 2-4 до тех пор, пока изменение параметров нейронной сети $\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|$ не станет достаточно малой величиной

3 Основные подходы к обучению нейронной сети для аппроксимации решений краевой задачи

3.1 создание архитектуры нейронной сети и ее обучение

На начальном этапе программы, чтобы убедиться, что библиотека нейронной сети обновлена, скрипт начинает с обновления пакета tf-keras — усовершенствованной версии Keras, интегрированной в TensorFlow. Это обновление важно, так как оно обеспечивает доступ к последним функциям и оптимизациям, доступным в библиотеке. После настройки среды скрипт импортирует несколько ключевых библиотек Python, каждая из которых выполняет свою роль в рабочем процессе. TensorFlow (tf) используется для создания и тренировки модели нейронной сети. NumPy (np) применяется для численных операций, в частности, для создания массивов и выполнения матричных операций. А Matplotlib (plt) — библиотека для визуализации данных и результатов модели.

```
%%capture
!pip install --upgrade tf-keras



import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
tf.__version__
```

Листинг 1: часть кода №1

Стратегия выборки данных тщательно разработана для питания нейронной сети разнообразием точек данных. Домен для волнового уравнения аккуратно разделен на внутренние и граничные сегменты, с тысячей точек, отобранных внутри и пятьюстами вдоль границы. Это обеспечивает всестороннее исследование пространства решений. Пространственные (x) и временные (t) диапазо-

ны установлены от нуля до пи, что соответствует математическим свойствам изучаемого волнового уравнения. Точки внутри домена генерируются случайным образом, позволяя модели изучать сложные динамики в различных условиях. На границах значения t фиксированы, что закрепляет решение за известными условиями на этих краях, что критически важно для точного захвата поведения волны в начале или в конце временного домена.

```
# Sampling
n_interior = 1000
n_boundary = 500
xlow, xhigh = 0, np.pi
tlow, thigh = 0, np.pi

def sampler():
  """Sample the function's domain (space-time)"""
  # Interior domain
  x0 = np.random.uniform(low=xlow, high=xhigh, size=[n_interior, 1])
  t0 = np.random.uniform(low=tlow, high=thigh, size=[n_interior, 1])

  # Boundary domain
  x1 = np.random.uniform(low=xlow, high=xhigh, size=[n_boundary, 1])
  t1 = tlow * np.ones((n_boundary, 1))

  return x0, t0, x1, t1
```

Листинг 2: часть кода №2

Чтобы установить ориентир для оценки предсказаний нейронной сети, точное или аналитическое решение волнового уравнения определяется с помощью тригонометрических функций. Функция «exact-solution» вычисляет «np.sin(x) * np.sin(t)», что математически представляет решение волнового уравнения при определенных условиях. Это известное решение служит критическим эталоном, чтобы убедиться, что результаты модели находятся на правильном пути и научно точны.

```
# Exact solution
```

```
def exact_solution(x, t):
    """Analytical solution for the PDE"""
    return np.sin(x)*np.sin(t)
```

Листинг 3: часть кода №3

Сердцем обучения нейронной сети является ее способность минимизировать ошибки, что обеспечивается функцией потерь. Эта функция рассчитывает два типа ошибок: одну из внутренней части домена и другую из границы. Используя «tf.GradientTape» TensorFlow, который записывает операции для автоматического дифференцирования, функция потерь фиксирует, насколько хорошо результаты сети соответствуют ожидаемым градиентам (производным) волнового уравнения — мера точности в воспроизведении динамики, заданной дифференциальным уравнением. Кроме того, она гарантирует, что модель придерживается правильных граничных условий, фундаментального аспекта при работе с дифференциальными уравнениями, где решение должно удовлетворять определенным условиям на краях домена.

```
# Loss function
def loss(model, x0, t0, x1, t1):
  """Compute total loss for training the NN"""
  # Loss term #0: average L2-norm of wave PDE differential operator
  # function value and derivatives at sampled points
  with tf.GradientTape() as tU0:
    U0 = model(x0, t0)
  U0x, U0t = tU0.gradient(U0, [x0, t0])
  L0 = tf.reduce_mean(tf.square(U0x) + tf.square(U0t) - \
                      2.*U0x*tf.math.cos(x0)*tf.math.sin(t0) - \
                      2.*U0t*tf.math.sin(x0)*tf.math.cos(t0))
  # Loss term #1: average L2-norm of initial condition (on gamma1)
  U1 = model(x1, t1)
  L1 = tf.reduce_mean(tf.square(U1))

  return L0 + L1, L0, L1
```

Листинг 4: часть кода №4

Архитектура нейронной сети разработана для обработки сложностей входных данных и необходимых вычислений для точного предсказания поведения

волнового уравнения. Класс Model охватывает эту архитектуру, построенную с использованием API Keras от TensorFlow, который упрощает определение слоев и операций. Сеть состоит из нескольких слоев LSTM (Long Short-Term Memory) и плотных слоев, настроенных последовательно. LSTM особенно хорошо справляются с обработкой последовательностей и могут эффективно захватывать временные паттерны, что делает их идеальными для задач, связанных с уравнениями, зависящими от времени. Входы в модель — это пространственные и временные координаты, которые объединяются и обрабатываются через сеть для предсказания поведения волнового уравнения в этих точках.

```
# Model
class Model(tf.keras.Model):
  def __init__(self):
    super(Model, self).__init__()
    self.model_layers = tf.keras.Sequential([
      tf.keras.layers.Input(shape=(100, 1)),
      tf.keras.layers.Dense(100),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.Dense(1)
    ])

  def call(self, x, t):
    return self.model_layers(tf.concat([x, t], 1))

model = Model()
```

Листинг 5: часть кода №5

Мы решили запустить нашу модель на 100 циклов, известных как эпохи, каждый цикл обрабатывает данные в 50 шагах. Каждая эпоха представляет собой полный проход через весь набор данных, и каждый шаг включает в себя пакет данных, обрабатываемых нейронной сетью. Скорость обучения,

установленная на уровне 0.001, определяет темп, с которым модель обновляет свои веса в ответ на наблюдаемые ошибки во время тренировки. Это тонкая настройка: слишком быстро, и модель может пропустить оптимальное решение; слишком медленно, и это может занять слишком много времени для сходимости или модель может застрять, не достигнув лучшего решения. В качестве оптимизатора выбран Adam за его эффективность в обработке разреженных градиентов на зашумленных проблемах.

```
# Hyperparameters
epochs = 100
steps = 50
learning_rate = 0.001
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
```

Листинг 6: часть кода №6

Мы определили массив «loss-arr» для сохранения значений потерь каждой эпохи. Это не только служит записью динамики обучения, но и основой для последующей визуализации кривой обучения, предоставляя информацию о том, насколько быстро и эффективно модель улучшается со временем. Изучая эти данные, мы можем видеть, улучшается ли модель и как быстро это происходит.

```
loss_arr = np.zeros((2, epochs))
```

Листинг 7: часть кода №7

Как работает обучение: В каждом цикле обучения выбираются свежие точки данных, чтобы модель имела всестороннее представление о проблеме. Эти точки преобразуются в формат, с которым TensorFlow, наша библиотека машинного обучения, может эффективно работать. На каждом шаге цикла модель анализирует данные, делает предположение и затем проверяет, насколько ее предположение отличается от правильного ответа. Эта проверка ошибок

выполняется с помощью чего-то, что называется функцией потерь, которая измеряет два типа ошибок: насколько хорошо предположения модели соответствуют правилам волнового уравнения и насколько точно они соответствуют граничным условиям нашей задачи. Затем оптимизатор использует эти измерения для небольшой корректировки настроек модели, стремясь уменьшить ошибки в будущих предположениях. Эта корректировка основана на градиентах, которые являются математическим способом указания «направлений», говорящих модели, как изменить ее настройки, чтобы приблизиться к правильным ответам.

```
# Train
for i in range(epochs):
  # We sample on each iteration
  xt_arr = sampler()
  xt_var = [tf.Variable(k, dtype=tf.float32) for k in xt_arr]

  for _ in range(steps):
    with tf.GradientTape() as tape:
      # Calculate the loss
      loss_vals = loss(model, *xt_var)
    grads = tape.gradient(loss_vals[0], model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))

  print('{:3} -> {:8.5f} ={:12.9f} +{:8.5f}'.format(
    i,
    loss_vals[0].numpy(),
    loss_vals[1].numpy(),
    loss_vals[2].numpy()
    ))
  loss_arr[0,i] = i
  loss_arr[1,i] = loss_vals[0].numpy()
```

Листинг 8: часть кода №8

Функция visualize-loss изящно олицетворяет суть визуального анализа данных, графически представляя эволюцию потерь в процессе обучения нейронной сети на протяжении эпох. Начиная с создания фигуры для графика, она

определяет четкое полотно высокого разрешения размером 16 на 10 дюймов, что гарантирует видимость даже самых тонких нюансов данных. Центральный элемент функции — это построение графика, на котором эпохи коррелируют с соответствующими значениями потерь, выделенными ярко-красной линией, что не только эффективно выделяет данные, но и помогает в визуальном различении.

```
def visualize_loss(loss_arr):
  plt.figure(figsize=(16, 10), dpi=200)
  plt.plot(loss_arr[0,:], loss_arr[1,:], color="tab:red")

  plt.yticks(fontsize=12, alpha=.7)
  # plt.ylim(ymin=loss_arr[1,:].min(),ymax=np.quantile(loss_arr[1,:],0.95))
  plt.xlabel(r"Stages", fontsize=15, labelpad=10)
  plt.ylabel(r"Error", fontsize=15, labelpad=20)
  plt.title("Loss Function in LSTM Training", fontsize=22)
  plt.grid(axis='both', alpha=.3)

visualize_loss(loss_arr)
```

Листинг 9: часть кода №9

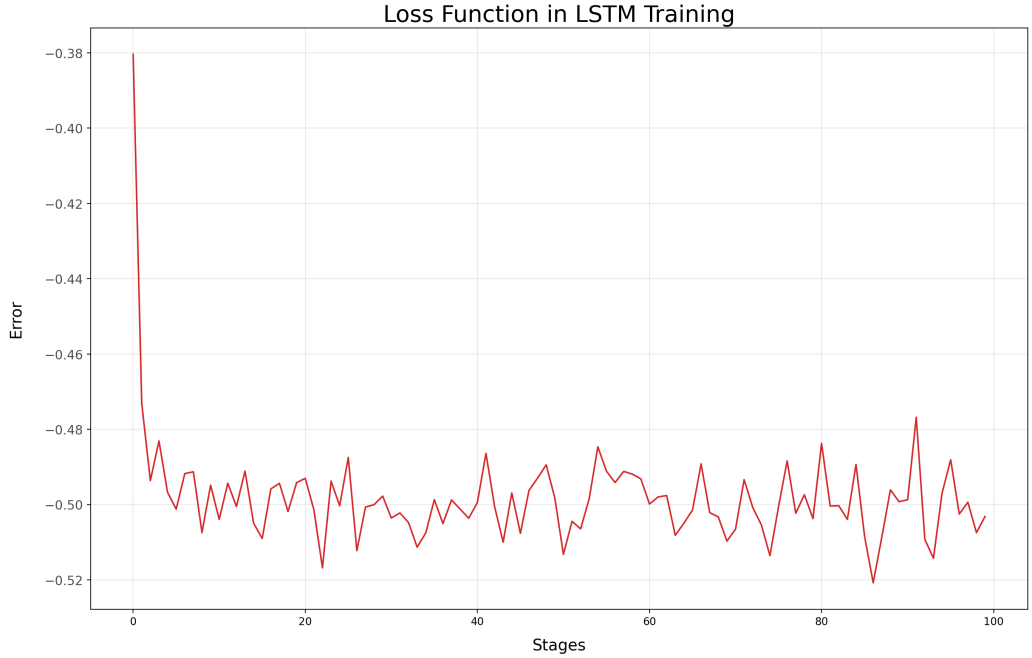

Рис. 1:

На графике под названием "Функция потерь при обучении LSTM"показано, как изменяется ошибка нейронной сети с долговременной и кратковременной памятью (LSTM) во время её обучения на протяжении 100 этапов, которые можно интерпретировать как эпохи или итерации.

Основные наблюдения: Быстрое падение в начале: График показывает резкое снижение ошибки в самом начале, начиная чуть ниже -0.38 и быстро падая до приблизительно -0.50. Это предполагает, что модель быстро усвоила значительное количество информации о базовых паттернах в данных на начальных этапах обучения, что привело к быстрому улучшению производительности.

Постепенная стабилизация: После начального резкого падения, уровень ошибок начинает стабилизироваться. Он колеблется в более узком диапазоне (около -0.50 до -0.52). Эта стабилизация указывает на то, что основные корректировки обучения произошли и последующее обучение дальше уточняет веса и смещения модели для оптимизации производительности.

Колебания: В течение всего обучения ошибка демонстрирует некоторые колебания. Эти взлеты и падения предполагают, что модель время от времени исследует потенциально лучшие конфигурации, что может временно увеличить ошибку, прежде чем она снова стабилизируется.

Общая тенденция к снижению: Несмотря на колебания, общая тенденция графика остаётся направленной вниз, хотя и незначительно, особенно после начального падения. Это указывает на постоянное, хотя и меньшее, улучшение точности модели.

Быстрое начальное уменьшение ошибки ожидаемо и желательно, так как это показывает, что модель способна эффективно учиться на наборе данных. Последующие меньшие колебания вокруг постепенно уменьшающегося уров-

ня ошибок предполагают, что модель тонко настраивает свои параметры для адаптации к сложностям данных.

3.2 Визуализация решения

```
n_plot = 51
x_plot = np.linspace(xlow, xhigh, n_plot)
t_plot = np.linspace(xlow, xhigh, n_plot)

def mesh_values(x_plot, t_plot):
  u_value_mesh = np.zeros([n_plot, n_plot])
  for i in range(n_plot):
    for j in range(n_plot):
      u_value_mesh[i, j] = exact_solution(x_plot[j], t_plot[i])
  # fit
  x_mesh, t_mesh = np.meshgrid(x_plot, t_plot)
  x_plot2 = np.reshape(x_mesh, [n_plot**2,1])
  t_plot2 = np.reshape(t_mesh, [n_plot**2,1])
  fit_u_value = model(x_plot2, t_plot2)
  fit_u_value = np.squeeze(fit_u_value, axis=2)
  fit_u_value = tf.convert_to_tensor(np.sum(fit_u_value, axis=1).reshape(-1, 1))
  fit_u_value_mesh = np.reshape(fit_u_value, [n_plot, n_plot])

  return x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh


def evaluate(u_value_mesh, fit_u_value_mesh):
  # accuracy check
  mean_abs_err = np.sum(np.abs(u_value_mesh - fit_u_value_mesh))/n_plot**2
  mean_sq_err = np.sum(np.abs(u_value_mesh - fit_u_value_mesh)**2)/n_plot**2
  r_sq = 1 - np.sum((u_value_mesh - fit_u_value_mesh)**2) / n_plot**2/np.var(u_value_mesh)

  return mean_abs_err, mean_sq_err, r_sq

x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh = mesh_values(x_plot, t_plot)
evaluate(u_value_mesh, fit_u_value_mesh)
```

Листинг 10: часть кода №10

Фрагмент кода описывает шаги по созданию визуальной и аналитической среды для понимания поведения системы, моделируемой через волновое уравнение, на пространственной и временной сетке. Процесс начинается с опреде-

ления n_plot как 51, что создает сетку 51x51 точек как в пространственном (x_plot), так и во временном (t_plot) измерениях. Эти точки равномерно распределены между xlow и xhigh, а также tlow и thigh соответственно, используя np.linspace, что обеспечивает детальную сетку для исследования динамики уравнения.

В функции mesh_values мы вычисляем точные решения в каждой точке сетки с помощью функции exact_solution. Это решение сохраняется в двумерном массиве u_value_mesh, который представляет собой теоретические или известные значения в каждой точке сетки. Сетка формируется путем объединения x_plot и t_plot с использованием np.meshgrid, а координаты затем преобразуются для соответствия требованиям входных данных нейронной сети. Модель предсказывает поведение в этих точках, и предсказания преобразуются обратно в двумерную сетку, fit_u_value_mesh, которая параллельна структуре точных решений.

Функция evaluate вычисляет три ключевых статистических показателя для количественной оценки точности предсказаний модели по сравнению с точными решениями: среднюю абсолютную ошибку (MAE), среднеквадратичную ошибку (MSE) и коэффициент детерминации (R-квадрат). Эти метрики предоставляют четкое числовое представление о том, насколько близко выходные данные модели соответствуют ожидаемым результатам, по сути оценивая способность модели воспроизводить реальные явления на основе входных данных и ее обучения.

```
def visualize_error(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

  ax1.set_title("Relative Error of LSTM Approximation", fontsize=14)
  ax1.pcolormesh(
```

```
    t_mesh, x_mesh, np.abs(u_value_mesh - fit_u_value_mesh),
    shading='auto', cmap = "rainbow"
  )

  ax2.set_title("Absolute Error of LSTM Approximation", fontsize=14)
  ax2.pcolormesh(
    t_mesh, x_mesh, np.minimum(np.where(u_value_mesh==0.,1., \
    np.abs(1 - np.divide(fit_u_value_mesh, u_value_mesh))),1.),
    shading='auto', cmap = "rainbow"
  )

  plt.plot()


visualize_error(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh)
```

Листинг 11: часть кода №11

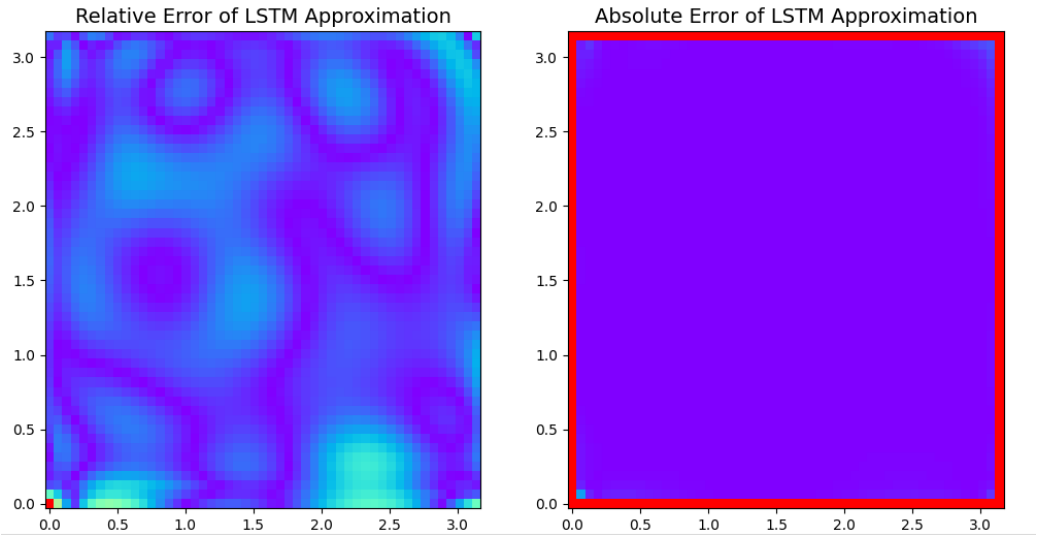

Рис. 2:

Функция visualize_error разработана для иллюстрации различий между предсказанными и фактическими решениями волнового уравнения, выполненными моделью LSTM, используя подход визуализации с двумя панелями. Функция принимает четыре параметра: x_mesh и t_mesh, которые представляют собой пространственные и временные координаты на сетке, а также u_value_mesh и fit_u_value_mesh, массивы фактических и предсказанных значений соответственно.

При вызове этой функции создается фигура с двумя сюжетными областями, каждая размером 12 на 6 дюймов для четкой визуализации. Первая сюжетная область, помеченная как "Относительная ошибка аппроксимации LSTM использует цветную сетку для изображения относительных различий между фактическими и предсказанными значениями. Для визуального различия различных степеней ошибки по всей сетке применяется яркая цветовая карта "радуга где абсолютные различия рассчитываются и отображаются.

Вторая сюжетная область сосредоточена на "Абсолютной ошибке аппроксимации LSTM где ошибки рассчитываются таким образом, чтобы нормализовать ошибки в позициях, где фактическое значение равно нулю, чтобы избежать бесконечных или неопределенных значений. Этот график также использует цветовую карту "радуга обеспечивая согласованную визуальную интерпретацию, которая помогает сравнивать узоры ошибок между двумя графиками.

С помощью plt.plot() функция в конечном итоге отображает графики, хотя эта конкретная строка не вносит вклад в функциональность, поскольку plt.show() или явная функция сохранения были бы более подходящими для отображения или сохранения фигуры.

Функция visualize_solutions создана для того, чтобы предоставить всестороннее и наглядное сравнение между точными решениями волнового уравнения в частных производных и приближениями, выполненными моделью LSTM, все это отображается в трехмерном формате. Функция начинается с создания фигуры, которая включает в себя два 3D графика, расположенных рядом, каждый из которых занимает холст размером 16 на 8 дюймов. Эта

просторная настройка гарантирует, что каждый график поверхности четко виден и легко анализируется.

На первом графике под названием "Поверхность точного решения волнового уравнения PDE"функция визуализирует фактические, теоретические значения решения с использованием сетки, определенной t_mesh и x_mesh, где u_value_mesh обеспечивает значения по оси z. Используется метод plot_surface для отображения поверхности, применяется цветовая карта cm.jet для иллюстрации различий на поверхности, с настройками для минимальной ширины линии и стандартным цветом краев для четкости без визуального шума.

Второй график под названием "Приближение LSTM поверхности решения волнового уравнения"повторяет настройки первого, но вместо этого отображает предсказания модели. Здесь fit_u_value_mesh предоставляет значения по оси z, указывая, насколько близко выходы модели LSTM соответствуют реальному ландшафту данных. Применяются те же визуальные настройки, обеспечивая согласованный вид между двумя графиками и позволяя легко визуально сравнивать и оценивать точность предсказаний модели по отношению к точному решению.

```
def visualize_solutions(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), subplot_kw={"projection": "3d"})

  ax1.set_title("Exact Wave PDE Solution Surface")
  ax1.plot_surface(
    t_mesh, x_mesh, u_value_mesh, rstride=1, cstride=1, linewidth=.05,
    cmap=cm.jet, antialiased=False, edgecolors='gray'
  )

  ax2.set_title("LSTM Approximation of Wave PDE Solution Surface")
  ax2.plot_surface(
    t_mesh, x_mesh, fit_u_value_mesh, rstride=1, cstride=1, linewidth=.05,
    cmap=cm.jet, antialiased=False, edgecolors='gray'
  )
```

```
15
16 visualize_solutions(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh)
```

Листинг 12: часть кода №12

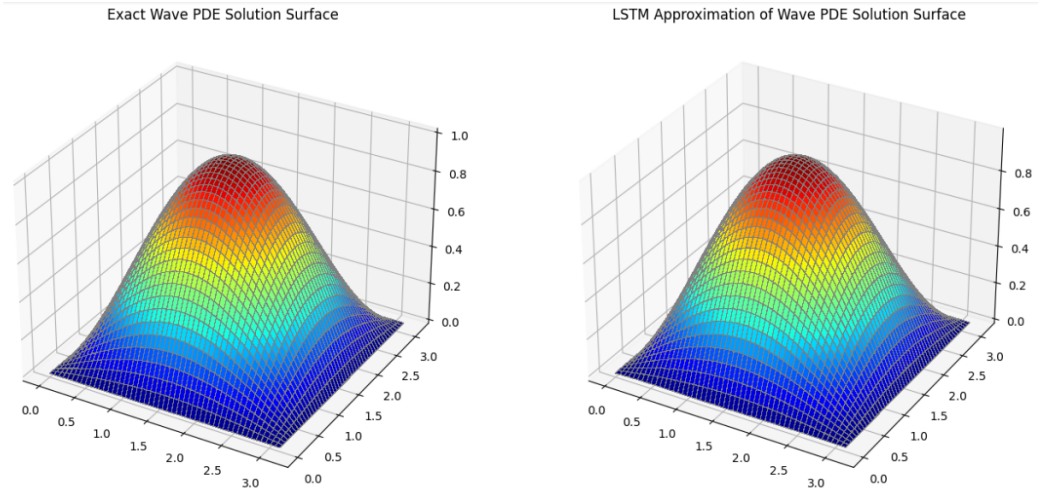

Рис. 3:

На этих двух трехмерных поверхностных графиках мы можем проанализировать сравнение между точными решениями волнового уравнения в частных производных и приближениями, созданными нейронной сетью LSTM. Каждый график отображает поведение волны в пространственно-временной сетке.

Левый график - Поверхность точного решения волнового уравнения: Этот график четко представляет теоретические или точные решения волнового уравнения. Цвета от красного до синего обозначают различные значения решения по сетке, иллюстрируя пик в центре, который постепенно уменьшается к краям. Точная форма и плавные переходы градиента указывают на математическую точность решения.

Правый график - Приближение LSTM поверхности решения волнового уравнения: Этот график показывает приближения модели LSTM для того же волнового уравнения. Цвета следуют аналогичному узору точного решения, что указывает на эффективность модели в захвате общей динамики волнового

уравнения. Поверхность тесно отражает левый график, что предполагает высокую степень точности в предсказаниях модели. Однако тонкие различия в контуре и интенсивности цвета могут выделить области, где приближение LSTM отклоняется от точного решения.

решениями и предсказаниями модели LSTM для волнового уравнения. Сначала она настраивает пространство размером 8x8 дюймов для графика, которое достаточно велико, чтобы четко видеть детали в трех измерениях. Затем на этом холсте готовится 3D-сюжет.

Основная часть этой функции использует ax.plot_surface для построения пространственной (x_mesh) и временной (t_mesh) сеток, а также абсолютных различий между точными решениями и предсказаниями модели (np.abs (u_value_mesh - fit_u_value_mesh)). Это показывает ошибку в виде поверхности на графике.

Для большей четкости деталей график использует специальные настройки, которые отображают каждую точку данных и делают линии тонкими. Применяется цветовая схема, которая изменяется от холодных к теплым цветам, что помогает интуитивно выделить области с большими различиями (ошибками). Края определены серым цветом, чтобы сделать формы и контуры на графике более заметными.

```
def visualize_error_3d(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh):
  figure = plt.figure(figsize=(8, 8))
  ax = figure.add_subplot(projection='3d')

  ax.plot_surface(
    t_mesh, x_mesh, np.abs(u_value_mesh - fit_u_value_mesh),
    rstride=1, cstride=1, linewidth=0.05, cmap=cm.jet,
    antialiased=False, edgecolors='gray'
```

```
  )

visualize_error_3d(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh)
```

Листинг 13: часть кода №13

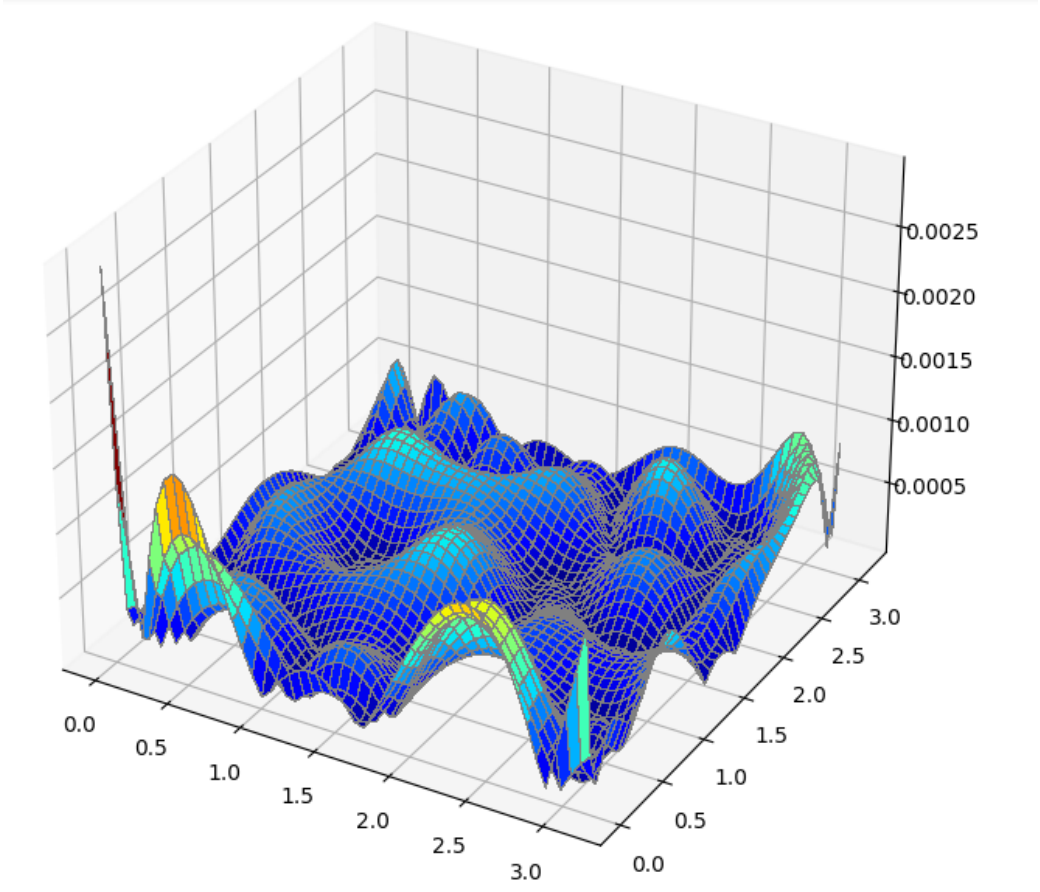

Рис. 4:

Оси x, y и z указывают размеры, в пределах которых происходят эти изменения, а ось z количественно определяет величину измеряемого параметра. Использование синей цветовой карты, которая переходит в более светлые оттенки к вершинам, подчеркивает увеличение значений. Этот цветовой переход упрощает визуальный осмотр местоположения самых высоких и самых низких значений в пространственной области. Данные показывают изменчивость по всему пространству без простого, однородного узора, что предполагает нелинейную связь внутри отображенных двух измерений. Несколько вершин

особенно выделяются своей высотой, особенно на левой стороне графика.

Заключение

В этой работе мы смогли изучить инновационный подход к решению задач BVP путем обучения нейронной сети с помощью неклассических вариационных формулировок, продемонстрировав огромные изменения в вычислительной науке. Экспериментальные результаты, полученные в результате внедрения и тестирования процедур решения, показали, что нейронная сеть может быть хорошим инструментом для решения задач BVP. Трудно переоценить, что сеть вполне достаточно аппроксимировала решение волнового уравнения при наличии высокой точности, низкой частоты ошибок и соответствия аналитического решения во всей предметной области. Более того, такой способ визуализации предсказаний модели и их ошибок позволяет интуитивно понять, в каких областях проблемы модель работает хорошо, а в каких - неадекватно. Такой подход к решению сложных математических задач можно будет изменить, упростив тем самым решение нелинейных задач или задач со слишком большим количеством переменных. Следующие шаги могут включать тестирование этого метода на более широком спектре уравнений и эксперименты с различными типами сетевых структур и процессами обучения для оптимизации.

Список литературы

[1] Zheyuan Hu, Zekun Shi, George Em Karniadakis, and Kenji Kawaguchi. Hutchinson trace estimation for high-dimensional and high-order physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 424:116883, 2024.

[2] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[3] Neha Yadav, Anupam Yadav, Manoj Kumar, et al. *An introduction to neural network methods for differential equations*, volume 1. Springer, 2015.

[4] Владимир Михайлович Филиппов, Владимир Михайлович Савчин, and Сергей Геннадьевич Шорохов. Вариационные принципы для непотенциальных операторов. *Итоги науки и техники. Серия «Современные проблемы математики. Новейшие достижения»*, 40(0):3–176, 1992.

[5] В М Шалов. Принцип минимума квадратичного функционала для гиперболического уравнения. *Дифференциальные уравнения*, 1(10):1338–1365, 1965.

[6] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, Vol.378, 2019, pp. 686-707.

[7] J. Han, J. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Journal of Proceedings of the National Academy of

Sciences, vol. 115, August 2018, pp. 8505-8510.

[8] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, DeepXDE: A Deep Learning Library for Solving Differential Equations, SIAM Review, vol. 63, 2021, pp.208-228.

[9] S. Pal, Numerical Methods: Principles, Analyses and Algorithms, Oxford University Press, Oxford, 2009.

[10] R. S. Beidokhti, A. Malek, Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques. J. Franklin Inst. 346, 898–913 (2009).

[11] I. G. Tsoulos, D. Gavrilis, E. Glavas, Solving differential equations with constructed neural networks. Neurocomputing 72, 2385–2391 (2009).

[12] K. S. Mcfall, J. R. Mahan, Artificial neural network method for solution of boundary value problem with exact satisfaction of arbitrary boundary conditions. IEEE Trans. Neural Netw. 20(8), 1221–1233 (2009).

[13] H. Alli, A. Ucar, Y. Demir, The solutions of vibration control problem using artificial neural networks. J. Franklin Inst. 340, 307–325 (2003).

[14] J. M. Zurada, Introduction to Artificial Neural Systems, Jaico Publishing House, St. Paul, 2001.

[15] Shih-Chia Huang, Trung-Hieu Le, Introduction to TensorFlow 2, Principles and Labs for Deep Learning, Academic Press, 2021, pp. 1-26.

[16] С. Л. Соболев, Некоторые применения функционального анализа в математической физике. ЛГУ, 1950.

[17] Р. Варга, Функциональный анализ и теория аппроксимации в численном анализе. М: Мир, 1974. 126 с. (РЖМат, 1975, ЗБ834К).

[18] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci., 115 (34) (2018), pp. 8505-8510.

[19] K. Kawaguchi, L. P. Kaelbling, Y. Bengio, Generalization in Deep Learning, Cambridge University Press, 2022.

[20] C. Beck, E. Weinan, A. Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, J. Nonlinear Sci., 29 (2019), pp. 1563-1619.

[21] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput., 9(8) (1997), pp. 1735-1780.

[22] I. Lagaris, A. Likas, D. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw., 11 (5) (2000), pp. 1041-1049.

[23] D. Bertsekas, J. Tsitsiklis, Gradient convergence in gradient methods via errors, SIAM J. Optim., 10 (3) (2000), pp. 627-642.

[24] Rong Gao, Kexin Hua, A numerical method for solving uncertain wave equation, Chaos, Solitons & Fractals, Volume 175, Part 1, 2023, 113976.

[25] Johannes D. Schmid, Philipp Bauerschmidt, Caglar Gurbuz, Martin Eser, Steffen Marburg, Physics-informed neural networks for acoustic boundary admittance estimation, Mechanical Systems and Signal Processing, Volume 215, 2024, 111405

Приложение

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

from matplotlib import cm
tf.__version__

# Sampling
n_interior = 1000
n_boundary = 500
xlow, xhigh = 0, np.pi
tlow, thigh = 0, np.pi

def sampler():
  """Sample the function's domain (space-time)"""
  # Interior domain
  x0 = np.random.uniform(low=xlow, high=xhigh, size=[n_interior, 1])
  t0 = np.random.uniform(low=tlow, high=thigh, size=[n_interior, 1])

  # Boundary domain
  x1 = np.random.uniform(low=xlow, high=xhigh, size=[n_boundary, 1])
  t1 = tlow * np.ones((n_boundary, 1))

  return x0, t0, x1, t1

# Exact solution
def exact_solution(x, t):
    """Analytical solution for the PDE"""
    return np.sin(x)*np.sin(t)

# Loss function
def loss(model, x0, t0, x1, t1):
  """Compute total loss for training the NN"""
  # Loss term #0: average L2-norm of wave PDE differential operator
  # function value and derivatives at sampled points
  with tf.GradientTape() as tU0:
    U0 = model(x0, t0)
  U0x, U0t = tU0.gradient(U0, [x0, t0])
  L0 = tf.reduce_mean(tf.square(U0x) + tf.square(U0t) - \
                      2.*U0x*tf.math.cos(x0)*tf.math.sin(t0) - \
                      2.*U0t*tf.math.sin(x0)*tf.math.cos(t0))
  # Loss term #1: average L2-norm of initial condition (on gamma1)
  U1 = model(x1, t1)
  L1 = tf.reduce_mean(tf.square(U1))

```

```
  return L0 + L1, L0, L1


# Model
class Model(tf.keras.Model):
  def __init__(self):
    super(Model, self).__init__()
    self.model_layers = tf.keras.Sequential([
      tf.keras.layers.Input(shape=(100, 1)),
      tf.keras.layers.Dense(100),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.LSTM(100, return_sequences=True),
      tf.keras.layers.Dense(1)
    ])

  def call(self, x, t):
    return self.model_layers(tf.concat([x, t], 1))

model = Model()


# Hyperparameters
epochs = 100
steps = 50
learning_rate = 0.001
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

loss_arr = np.zeros((2, epochs))


# Train
for i in range(epochs):
  # We sample on each iteration
  xt_arr = sampler()
  xt_var = [tf.Variable(k, dtype=tf.float32) for k in xt_arr]

  for _ in range(steps):
    with tf.GradientTape() as tape:
      # Calculate the loss
      loss_vals = loss(model, *xt_var)
    grads = tape.gradient(loss_vals[0], model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))

  print('{:3} -> {:8.5f} ={:12.9f} +{:8.5f}'.format(
    i,
    loss_vals[0].numpy(),
```

```
    loss_vals[1].numpy(),
    loss_vals[2].numpy()
    ))
  loss_arr[0,i] = i
  loss_arr[1,i] = loss_vals[0].numpy()


def visualize_loss(loss_arr):
  plt.figure(figsize=(16, 10), dpi=200)
  plt.plot(loss_arr[0,:], loss_arr[1,:], color="tab:red")

  plt.yticks(fontsize=12, alpha=.7)
  # plt.ylim(ymin=loss_arr[1,:].min(),ymax=np.quantile(loss_arr[1,:],0.95))
  plt.xlabel(r"Stages", fontsize=15, labelpad=10)
  plt.ylabel(r"Error", fontsize=15, labelpad=20)
  plt.title("Loss Function in LSTM Training", fontsize=22)
  plt.grid(axis='both', alpha=.3)

visualize_loss(loss_arr)


n_plot = 51
x_plot = np.linspace(xlow, xhigh, nplot)
t_plot = np.linspace(xlow, xhigh, nplot)

def mesh_values(x_plot, t_plot):
  u_value_mesh = np.zeros([n_plot, n_plot])
  for i in range(n_plot):
    for j in range(n_plot):
      u_value_mesh[i, j] = exact_solution(x_plot[j], t_plot[i])
  # fit
  x_mesh, t_mesh = np.meshgrid(x_plot, t_plot)
  x_plot2 = np.reshape(x_mesh, [n_plot**2,1])
  t_plot2 = np.reshape(t_mesh, [n_plot**2,1])
  fit_u_value = model(x_plot2, t_plot2)
  fit_u_value = np.squeeze(fit_u_value, axis=2)
  fit_u_value = tf.convert_to_tensor(np.sum(fit_u_value, axis=1).reshape(-1, 1))
  fit_u_value_mesh = np.reshape(fit_u_value, [n_plot, n_plot])

  return x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh


def evaluate(u_value_mesh, fit_u_value_mesh):
  # accuracy check
  mean_abs_err = np.sum(np.abs(u_value_mesh - fit_u_value_mesh))/n_plot**2
  mean_sq_err = np.sum(np.abs(u_value_mesh - fit_u_value_mesh)**2)/n_plot**2
  r_sq = 1 - np.sum((u_value_mesh - fit_u_value_mesh)**2) / n_plot**2/np.var(u_value_mesh)

```

```
  return mean_abs_err, mean_sq_err, r_sq


x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh = mesh_values(x_plot, t_plot)
evaluate(u_value_mesh, fit_u_value_mesh)


def visualize_error(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

  ax1.set_title("Relative Error of LSTM Approximation", fontsize=14)
  ax1.pcolormesh(
    t_mesh, x_mesh, np.abs(u_value_mesh - fit_u_value_mesh),
    shading='auto', cmap = "rainbow"
  )

  ax2.set_title("Absolute Error of LSTM Approximation", fontsize=14)
  ax2.pcolormesh(
    t_mesh, x_mesh, np.minimum(np.where(u_value_mesh==0.,1., \
    np.abs(1 - np.divide(fit_u_value_mesh, u_value_mesh))),1.),
    shading='auto', cmap = "rainbow"
  )

  plt.plot()


visualize_error(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh)

def visualize_solutions(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), subplot_kw={"projection": "3d"})

  ax1.set_title("Exact Wave PDE Solution Surface")
  ax1.plot_surface(
    t_mesh, x_mesh, u_value_mesh, rstride=1, cstride=1, linewidth=.05,
    cmap=cm.jet, antialiased=False, edgecolors='gray'
  )

  ax2.set_title("LSTM Approximation of Wave PDE Solution Surface")
  ax2.plot_surface(
    t_mesh, x_mesh, fit_u_value_mesh, rstride=1, cstride=1, linewidth=.05,
    cmap=cm.jet, antialiased=False, edgecolors='gray'
  )

visualize_solutions(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh)


def visualize_error_3d(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh):
  figure = plt.figure(figsize=(8, 8))
```

```
  ax = figure.add_subplot(projection='3d')

  ax.plot_surface(
    t_mesh, x_mesh, np.abs(u_value_mesh - fit_u_value_mesh),
    rstride=1, cstride=1, linewidth=0.05, cmap=cm.jet,
    antialiased=False, edgecolors='gray'
  )

visualize_error_3d(x_mesh, t_mesh, u_value_mesh, fit_u_value_mesh)
```

Листинг 14: Python code for generating samples