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

Кафедра «Прикладная математика и информатика»

(наименование кафедры полностью)

01.03.02 Прикладная математика и информатика

(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование

(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Математическое моделирование волноводных структур с заданными свойствами»

Обучающийся Р.В. Пекишев

(И.О. Фамилия) (личная подпись)

Руководитель д.ф-м.н., доцент, С.В. Талалов

(ученая степень (при наличии), ученое звание (при наличии), И.О. Фамилия)

Тольятти, 2024

Аннотация

Выпускная квалификационная работа посвящена математическому моделированию волноводных структур с заданными свойствами. Основное внимание уделено решению обратной задачи Штурма-Лиувилля, которая используется одной из ключевых в математической физике. В работе рассматривается проблема определения потенциала в уравнении Штурма-Лиувилля на основе заданного отрицательно дискретного спектра задачи, которая используется в таких сферах как квантовая механика, оптика и теория волноводов.

Цель исследования – разработка и реализация метода подбора для определения параметров обратной задачи Штурма-Лиувилля в контексте планарных волноводов.

Объект исследования – обратная задача Штурма-Лиувилля.

Предмет исследования – метод подбора для определения коэффициентной функции уравнения по заданному дискретному спектру.

Для поставленной цели выполняется обзор литературы, составление математической модели, программная реализация алгоритма решения, а также тестирование разработанной программы на моделированных данных.

Работа включает теоретическую часть, где рассматриваются основные аспекты задач Штурма-Лиувилля, и практическую часть, где приводится реализация алгоритмов на языке Python с использованием специализированных библиотек для научных вычислений. Результаты исследования позволяют оценить эффективность и точность предложенного подхода в условиях практического применения, что способствует усовершенствованию процессов проектирования и оптимизации оптических устройств на основе волноводов.

Abstract

The graduation qualification work is dedicated to the mathematical modeling of waveguide structures with specified properties. The main focus is on solving the inverse Sturm-Liouville problem, which is one of the key problems in mathematical physics. The work explores the problem of finding the potential in the Sturm-Liouville equation by given negative discrete spectrum of the tusk, which is used in fields such as quantum mechanics, optics, and waveguide theory.

The aim of the research is the development and implementation of a matching method for determining the parameters of the inverse Sturm-Liouville problem in the context of planar waveguides.

The object of the research is the inverse Sturm-Liouville problem. The subject of the research is the matching method for determining the coefficient function of the equation based on the given discrete spectrum.

To achieve the set goal, a literature review, the development of a mathematical model, the software implementation of the solution algorithm, and the testing of the developed program on simulated data are performed.

The work includes a theoretical part, which discusses the main aspects of Sturm-Liouville problems, and a practical part, which presents the implementation of algorithms in Python using specialized libraries for scientific computing. The research results allow for the evaluation of the efficiency and accuracy of the proposed approach under practical application conditions, contributing to the improvement of the design and optimization processes of optical devices based on waveguides.

Содержание

Введение

Решение обратных задач Штурма-Лиувилля представляет собой одно из важнейших направлений в математической физике и приложениях, связанных с изучением свойств дифференциальных уравнений. Специфика этих задач заключается в определении параметров дифференциального уравнения по заданному набору его решений или спектральным характеристикам. В частности, проблема нахождения потенциалов или коэффициентов в уравнении Штурма-Лиувилля по известному дискретному спектру имеет принципиальное значение для многих областей науки и инженерии, включая квантовую механику, оптику и теорию волноводов.

В контексте планарных волноводов, где распространение волн описывается с помощью дифференциальных уравнений Штурма-Лиувилля, задача приобретает особую актуальность. Планарные волноводы, являясь фундаментальным элементом в оптоэлектронике и фотонике, требуют точного моделирования для разработки эффективных оптических устройств. Определение параметров волновода, таких как профиль преломления, на основе известного спектра мод, позволяет значительно усовершенствовать процесс проектирования и оптимизации этих устройств.

Метод подбора, рассматриваемый в данной работе, предлагает эффективный алгоритмический подход к решению обратной задачи Штурма-Лиувилля в условиях ограниченной исходной информации, что часто встречается на практике. Сущность метода заключается в итеративном уточнении параметров дифференциального уравнения с целью минимизации различий между известным дискретным спектром и спектром, получаемым в результате численного решения уравнения с подбираемыми параметрами.

Целью данной выпускной квалификационной работы является реализация и анализ метода подбора для определения параметров обратной задачи Штурма-Лиувилля в контексте планарных волноводов на основе

известного дискретного спектра. Рассмотрение данной темы предполагает не только теоретическую разработку алгоритма, но и его апробацию на реальных или моделированных данных, что позволит оценить эффективность и точность предложенного подхода в условиях практического применения. Для достижения поставленной цели следует выполнить следующие задачи:

- Сделать литературный обзор, в котором отражены: связь теории планарных волноводов со спектральной задачей Штурма-Лиувилля в различных случаях; общая схема решения обратной задачи Штурма-Лиувилля в пространстве $L_2[-1,1]$ методом подбора; основные моменты процедуры минимизации невязки градиентным методом;
- Составить блок-схему решения задачи методом подбора, минимизируя невязку;
- Выполнить программную реализацию исследования, используя различный выбор начальной точки при градиентном минимизации невязки. Интерпретировать результаты;
- Выполнить гладкую аппроксимацию найденной функции (функций) $V(x)$;
- Сделать сравнение результатов при разных ϵ и разных N. Сравнить: сами решения, а также затраченные машинные ресурсы на его нахождение. Исследовать влияние выбора начальной точки алгоритма при различных числах N.

Объектом исследования выпускной квалификационной работы является обратная задача Штурма-Лиувилля.

Предметом ВКР является решение (методом подбора) обратной задачи Штурма-Лиувилля - определение коэффициентной функции уравнения по заданному (отрицательному) дискретному спектру.

Для понимания сути ВКР определим используемые понятия: обратные и некорректные задачи, градиентные метод, планарные волноводы, уравнение Шредингера, метод подбора.

В научных и инженерных исследованиях различие между прямыми и обратными задачами является ключевым для понимания моделирования и анализа систем. Прямая задача состоит в определении результатов или эффектов при известных начальных условиях и параметрах системы. Например, при известной силе, действующей на объект, и его массе можно вычислить ускорение объекта согласно второму закону Ньютона. Решение прямой задачи влечёт за собой прямое приложение теоретических моделей для предсказания результатов.

Обратная задача, напротив, заключается в определении начальных условий или параметров системы, исходя из наблюдаемых результатов. Она гораздо сложнее, поскольку одним и теми же результатами могут обладать различные начальные условия или параметры. Решение обратной задачи часто требует итерационных методов и использования алгоритмов оптимизации для нахождения наиболее вероятных начальных условий, которые привели к наблюдаемым результатам.

Таким образом, основное различие между прямыми и обратными задачами заключается в направлении анализа: прямые задачи движутся от причин к следствиям, в то время как обратные задачи пытаются восстановить причины по имеющимся следствиям.

В контексте определения Жака Адамара корректная обратная задача характеризуется существованием, единственностью устойчивостью и её решения. Единственность означает, что существует только одно решение, соответствующее заданным начальным данным, в то время как устойчивость подразумевает, что небольшие изменения в исходных данных приводят к незначительным изменениям в решении [7]. Некорректная обратная задача не удовлетворяет хотя бы одному из этих критериев, что приводит к неопределённостям или неустойчивости решений.

Рассмотрим метод подбора решения обратной задачи Штурма-Лиувилля и минимизацию невязки градиентным методом.

Общая схема решения обратной задачи Штурма-Лиувилля методом подбора в пространстве $L_2[-1,1]$ представляет собой итерационный процесс, целью которого является нахождение такой функции потенциала q(x), которая позволит наилучшим образом воспроизвести заданный набор собственных значений. Процесс включает в себя следующие ключевые шаги:

Инициализация: начальное приближение функции потенциала $q_0(x)$ выбирается исходя из априорных знаний или простого предположения. Это приближение служит отправной точкой для дальнейших итераций.

Решение прямой задачи: для текущего приближения потенциала $q_k(x)$ решается прямая задача Штурма-Лиувиля, чтобы вычислить собственные значения. Этот шаг требует применения численных методов, таких, как «Метод стрельбы».

Оценка невязки: вычисляется разность (невязка) между собственными значениями, полученными на предыдущем шаге, и заданными собственными значениями. Функционал невязки может быть определен, например, как сумма квадратов этих разностей. Невязка в данном контексте определяется как разность между левой и правой частью дифференциального уравнения, возникающего при подстановке приближённого решения в исходное уравнение. Цель состоит в минимизации этого функционала.

Минимизация невязки: Используя методы оптимизации, такие как градиентный спуск, производится коррекция потенциала $q_k(x)$, чтобы минимизировать функционал невязки. Важным элементом является вычисление градиента функционала по потенциалу и выбор подходящего размера шага для обновления.

Итерация и конвергенция: Процесс повторяется, потенциал $q(x)$ постепенно корректируется на каждом шаге итерации до тех пор, пока не будет достигнута удовлетворительная степень согласованности между вычисленными и заданными собственными значениями, или пока не будут

выполнены другие критерии остановки (например, максимальное количество итераций или достижение заданной точности).

Анализ результатов: после остановки итерационного процесса анализируется полученный потенциал $q(x)$ на предмет его соответствия начальным ожиданиям, физической осмысленности и стабильности решения.

Обратная задача Штурма-Лиувилля является одним из фундаментальных примеров некорректной обратной задачи в математической физике, особенно в контексте квантовой механики и теории волновых процессов. Эта задача заключается в восстановлении потенциала $q(x)$ дифференциального оператора второго порядка на интервале $[a, b]$, исходя из данных рассеяния. В классическом подходе, исходя из известного набора собственных значений и некоторых дополнительных условий, таких как нормировочные коэффициенты, стремятся восстановить функцию потенциала $q(x)$, которая удовлетворяет уравнению Штурма-Лиувилля (2) с граничными условиями 3-го рода (Робена) [20].

$$-y'' + \mathrm{q(x)y} = \lambda y, \qquad x \in [0; \pi], \tag{2}$$

$$\begin{cases} \gamma_1 y'(a) + \beta_1 y(a) = 0, \ \gamma_1^2 + \beta_1^2 \neq 0 \\ \gamma_2 y'(b) + \beta_2 y(b) = 0, \ \gamma_2^2 + \beta_2^2 \neq 0 \end{cases} \tag{3}$$

где q(x) –потенциал;

γ, β –произвольные вещественные числа.

Решение обратной задачи Штурма-Лиувилля позволяет не только определить форму потенциала, но и глубже понять физические свойства системы, например распределение электронных состояний в квантовой яме. Однако из-за её некорректности, небольшие ошибки в данных могут привести к значительным расхождениям в решении, что требует применения методов регуляризации для получения физически осмысленных результатов.

Актуальность теории обратных задач спектрального анализа значительно возросла в свете развития квантовой механики и физики конденсированных сред. Стационарное уравнение Шредингера (4), являющееся частным случаем уравнения Штурма-Лиувилля, играет центральную роль в изучении квантовых систем, позволяя определить энергетические уровни и волновые функции частиц в потенциальных ямах или полях [20]. Решение обратной задачи для уравнения Шредингера (4) дает возможность восстановить потенциальную функцию системы по известным энергетическим спектрам, что критически важно для понимания структуры вещества на микроскопическом уровне [19].

$$\frac{d^2\psi(x)}{dx^2} + (E - u(x)\psi(x)) = 0, \tag{4}$$

где x –декартовая координата частицы;

ψ –волновая функция;

$u(x)$ –потенциальная энергия частицы;

E –энергия частицы.

Кроме того, данная теория находит применение в оптике и теории волноводов, где аналогичные математические модели используются для описания распространения световых волн в средах с переменным показателем преломления. Анализ спектральных данных позволяет определить характеристики оптических волноводов, такие как режимы распространения и дисперсионные свойства, что непосредственно влияет на проектирование высокоэффективных оптических устройств и линий связи. Таким образом, теория применение методов решения обратных задач для уравнения (4) с соответствующими граничными условиями предоставляет мощный инструментарий для глубокого понимания и инженерного проектирования в широком спектре научных и технических приложений.

1 Формулировка математической модели

1.1 Обзор литературы

Развитие обратных и некорректных задач стало значительным направлением в математике и физике XX века, обусловленное потребностью в точных методах решения соответствующих уравнений, не поддающихся стандартным аналитическим подходам. Некорректные задачи, чьи решения не удовлетворяли условиям Коши – существование, единственность и устойчивость, стали особенно актуальны в связи с развитием вычислительной техники и сложных инженерных систем. Советские ученые, такие как Андрей Николаевич Тихонов и Василий Сергеевич Владимиров, внесли значительный вклад в развитие теории и методов решения этих соответствующих спектральных проблем. Их работы заложили основу для методов регуляризации некорректных задач и развития теоретических основ обратных задач, предоставив мощный инструментарий для исследований в различных областях науки и техники.

В монографии С.И. Кабанихина «Обратные и некорректные задачи» [7] представлен всесторонний анализ теоретических и практических аспектов обратных и некорректных задач, которые находят широкое применение в современной науке и технике. Автор систематизировал и детально рассмотрел методы решения, критерии и алгоритмы регуляризации, а также условия корректности для широкого класса соответствующих спектральных проблем. Особое внимание уделено проблематике идентификации параметров, восстановлению функций и решению дифференциальных уравнений в частных производных. Так, С.И. Кабанихин акцентирует важность комплексного подхода, сочетающего теоретическую строгость и практическую направленность, что делает его работу неоценимым ресурсом для специалистов в области прикладной математики, физики и инженерии.

Книга служит фундаментом для понимания основ и последних достижений в области обратных и некорректных задач, подчеркивая их значимость в развитии научных исследований и технологического прогресса.

В труде А. Н. Тихонова и В. Я. Арсенина «Методы решения некорректных задач» [17] освещены ключевые аспекты теории и практического применения методов решения соответствующих уравнений, не удовлетворяющих условиям корректности по Адамару. Авторы подробно анализируют принципы и методы регуляризации, предлагая решения для соответствующих спектральных проблем, которые характеризуются отсутствием устойчивости, единственности или существования решений. В частности, вводится и обосновывается концепция регуляризации Тихонова, которая стала основополагающей в области решения некорректных задач. Книга вводит ряд фундаментальных терминов, таких как «регуляризующий оператор», «априорная информация», «условия стабилизации», раскрывая их значение и применение в контексте некорректных задач. Монография Тихонова и Арсенина служит незаменимым источником для научных работников и специалистов, занимающихся анализом и решением сложных математических и физических задач, демонстрируя глубокое понимание проблематики и предлагая эффективные методы ее решения.

В работе [23] проводится анализ специализированных алгоритмов для решения уравнений Штурма-Лиувилля и Шрёдингера. Автор демонстрирует эффективность численных методов, особенно в контексте квантовой механики и математической физики. Рассматриваются ключевые аспекты точности, стабильности и вычислительной эффективности предложенных подходов, подкрепленные теоретическими обоснованиями и практическими примерами.

Учебное пособие М.М. Карчевского [9] представляет собой глубокий аналитический обзор ключевых уравнений, лежащих в основе математической физики. Автор детализирует методы решения линейных и нелинейных уравнений, включая уравнения в частных производных, и обеспечивает

читателям теоретические основы и практические навыки для понимания и применения в различных научных исследованиях. М.М. Карчевский акцентирует внимание на важности математического аппарата для решения сложных физических задач, делая пособие ценным ресурсом для студентов и специалистов в области математики и физики.

В своей монографии «Обратные задачи Штурма-Лиувилля» Б. М. Левитан [10] представляет анализ обратных спектральных задач, основываясь на классических результатах спектральной теории оператора Штурма–Лиувилля. В. А. Марченко в «Операторы Штурма-Лиувилля и их приложения» [11], акцентирует внимание на операторном подходе к спектральным задачам и их приложениях в математической и теоретической физике. Обе работы вносят значительный вклад в понимание и развитие методов решения обратных задач, предлагая глубокий теоретический анализ и практические алгоритмы для исследования сложных физических систем.

В статье А. А. Егорова [4] рассматривается обратная задача рассеяния лазерного излучения в интегрально-оптическом волноводе с двумерными нерегулярностями на фоне шума. Автор анализирует влияние стохастических флуктуаций на точность реконструкции параметров нерегулярностей. Разработанный метод объединяет теорию рассеяния света с численными алгоритмами для устойчивого решения обратных задач, показывая значимость учета шумовых характеристик для повышения точности измерений.

В исследовании [18] представлен новаторский подход к реконструкции потенциальной функции в обратной задаче Штурма-Лиувилля с использованием частичных данных. Авторы демонстрируют эффективность метода на основе частичных и приближенных данных для восстановления точной формы потенциальной функции, выделяя важность данного подхода для теоретической и прикладной физики.

В монографии О. М. Алифанова и Е.А. Артюхина [1] представлены методы решения, основанные на принципах экстремального поиска решений

для некорректно поставленных задач. Авторы фокусируются на разработке и анализе алгоритмов, применимых в области обратных задач теплообмена, подчеркивая их эффективность в условиях неопределенности и нестабильности данных. Особое внимание уделено методам регуляризации и оптимизации, обеспечивающим устойчивость и точность решений в контексте обратных задач.

В издании А.А. Самарского и П. Н. Вабищевича [14] подробно излагаются современные численные подходы к решению обратных задач. Авторы сосредотачиваются на методах, обеспечивающих высокую точность и устойчивость решений, ключевыми среди которых являются итерационные методы и алгоритмы регуляризации. Особое внимание уделяется адаптации этих методов для конкретных классов соответствующих уравнений, связанных с математической физикой, что делает работу актуальной для исследователей и практиков в данной области.

В статье В. А. Морозова [12] рассматриваются методы регуляризации для стабилизации решений некорректных задач, вводя понятие семейств операторов как инструмента управления процессом регуляризации. А.Б. Бакушинский в работе [2] анализирует применение принципа невязки для оценки точности и устойчивости решений обратных задач, расширяя область его применения и углубляя теоретическое понимание механизмов регуляризации.

В статье [2] освещается углубленный анализ использования принципа невязки для оценки адекватности решений некорректных задач, предлагая новые подходы к его применению для повышения точности. И.В. Емелин и М.А. Красносельский в своей статье [5] фокусируются на методологии определения момента прекращения итераций, что является ключевым для обеспечения устойчивости решений и избежания переобусловленности в процессе регуляризации.

В работе [25] исследует применение функционала наименьших квадратов для решения обратных задач Штурма-Лиувилля, предлагая численный метод, основанный на минимизации ошибок между теоретическими и экспериментальными данными.

В издании [22] освещают теоретические аспекты обратных задач Штурма-Лиувилля и их приложения, разрабатывая методику для анализа и восстановления операторов на основе заданных спектральных данных, что способствует глубокому пониманию механизмов и решениям специфических прикладных задач.

В учебном электронном пособии С. В. Талалова «Обратные и некорректные задачи» [15], представлен краткий курс лекций. В данных лекциях представлена краткая теория и необходимые сведения из функционального анализа, используемые при решении прямых, обратных и некорректных задач. Автор обращает свое внимание на такие термины как метрические и нормированные пространства, линейные операторы в пространствах, прямая и обратная задача Штурма-Лиувилля. Так же С.В. Талалов показывает читателям подробный алгоритм по решению обратных и прямых задач Штурма-Лиувилля, рассматривая разные случаи и виды данной задачи.

1.2 Обобщенная математическая модель на примере планарного световода

Планарный волновод представляет собой устройство, способное направлять волны, в частности световые, в пределах плоскостного слоя материала. Основная особенность такого волновода – его геометрия, которая обеспечивает эффективное удержание и передачу энергии волн в двумерной плоскости.

Планарные волноводы обычно состоят из трех слоев:

Подложка: нижний слой с относительно низким показателем преломления n_2.

Сердечник: центральный слой, обладающий высоким показателем преломления n_1, что необходимо для удержания волн внутри данного слоя посредством полного внутреннего отражения.

Крышка: верхний слой, аналогичный подложке по показателю преломления n_3.

Для описания распространения света в волноводе используются уравнения Максвелла для электромагнитных волн. Основное условие для удержания волны в сердечнике – это выполнение условия полного внутреннего отражения. Это условие соблюдается, когда угол θ падения волны на границу сердечник/крышка или сердечник/подложка превышает критический угол, который можно выразить через показатели преломления материалов [3].

Монохроматический свет в планарном волноводе распространяется вдоль оси волновода, управляемый показателем преломления материала и длиной волны света. Волновой вектор $\vec{k}$ определяет направление и фазовую скорость волны, выражаясь как $\vec{k} = \frac{2\pi}{\lambda} n_1 \hat{k}$ и $k = \frac{2\pi}{\lambda} = \frac{\omega}{c}$ где ω – угловая частота волны, с – скорость света в вакууме. Показатель преломления n_1 и длина волны λ играют ключевые роли в описании волнового процесса. Постоянная распространения β описывает фазовые изменения волны при её движении, что критично для оптических систем. Эта величина определяется как проекция волнового вектора на направление распространения волны вдоль волновода: $\beta = kn_1 \cos(\theta)$, где θ – угол между вектором $\overrightarrow{k}$ и осью волновода. Этот параметр отражает вклад волнового вектора, который направлен непосредственно вдоль длинноволнового направления распространения. Трехслойный планарный волновод представлен на рисунке 1.

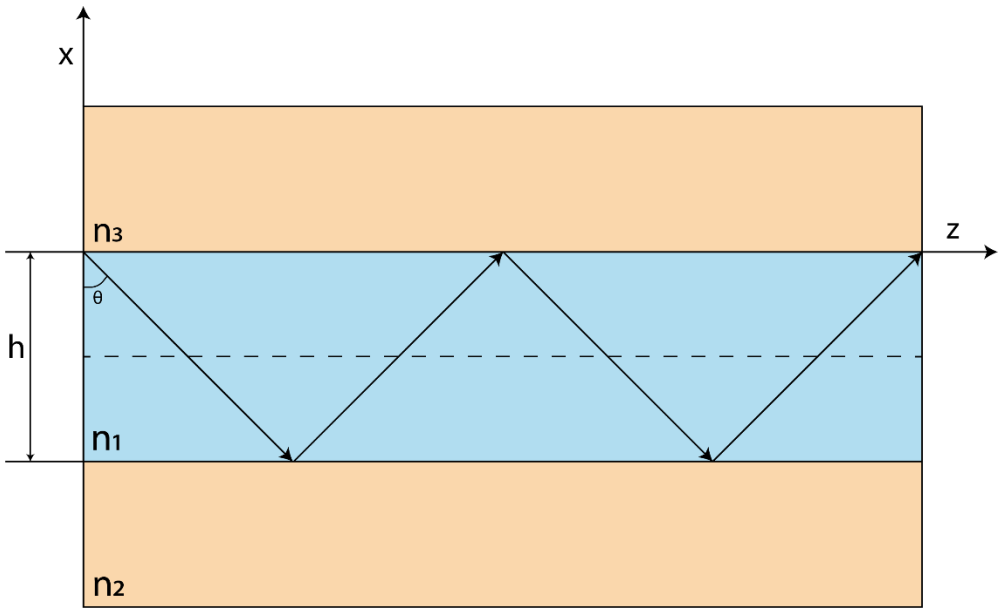

Рисунок 1 - Трёхслойный планарный волновод

В планарном волноводе процесс формирования стоячих волн является результатом попеременного отражения световой волны от подложки и покровного слоя. Эти слои имеют различные показатели преломления по сравнению с сердечником волновода, что способствует полному внутреннему отражению света. Рассмотрим волновод (рисунок 2), ориентированный вдоль оси z, где x представляет поперечное направление. В таком случае световую волну, распространяющуюся в волноводе, можно представить в виде суммы двух волн, одна из которых распространяется вперёд, а другая – назад: $E(x,z) = E_0 sin(k_x x) cos(\beta z) + E_0 sin(k_0 x) sin(\beta z)$, где $E(x,z)$ – амплитуда электрического поля в точке (x, z), E_0 – амплитуда входящей волны, k_x – волновое число в поперечном направлении $x, \beta -$ постоянная распространения вдоль оси z. Стоячие волны образуются при интерференции двух волн одинаковых амплитуд, распространяющихся в противоположных направлениях. В волноводе это условие достигается, когда фазы отражённых волн совпадают. Для стоячих волн условие фазового совпадения выражается как: $\beta z = m\pi, \quad m = 0, 1, 2, \ldots$, где m – целое число, обозначающее порядок

моды. Угол θ, под которым свет попадает в волновод, и величина постоянной распространения β имеют критическое значение для возникновения стоячих волн. Угол θ определяет направление входа света и должен быть достаточным для обеспечения полного внутреннего отражения [6][13]. Стоячие волны в волноводе, известные как волновые моды, представляют собой резонансные состояния, которые возникают на резонансных частотах, где длина волны или её кратные соответствуют размеру системы. Это означает, что волна укладывается в систему целое число раз. Первые четыре волновые моды показаны на рисунке 2.

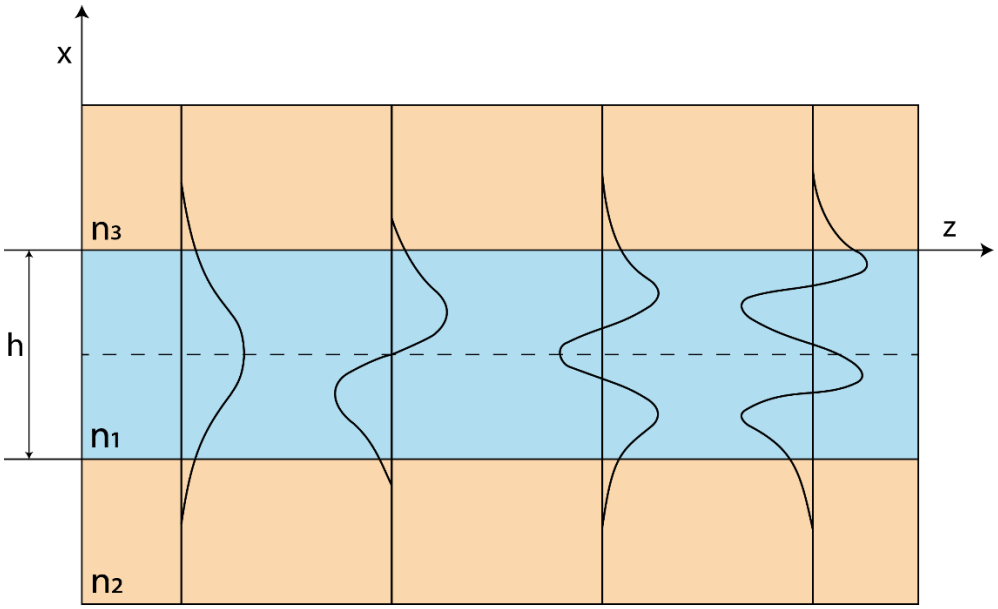

Рисунок 2 – Четыре волновые моды

Уравнение Гельмгольца для двумерного случая включает производные по двум координатам. Исходное уравнение:

$$(\Delta + k^2 {n_1}^2)u = 0, \qquad (5)$$

где $\Delta -$ оператор Лапласа для двух измерений, определяемый как $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$;

$k -$ волновой вектор; $n_1 -$ показатель преломления.

Тогда уравнение Гельмгольца записывается в таком виде:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + k^2 {n_1}^2 u = 0, \quad (6)$$

Для того чтобы привести это уравнение (6) к виду уравнения Штурма-Лиувилля, рассмотрим возможность разделения переменных. Предположим, что решение *u* можно представить как произведение двух функций, каждая из которых зависит только от одной переменной $u(x, y) = X(x)Y(y)$. Подставляя это в уравнение Гельмгольца, получаем:

$$X''(x)Y(y) + X(x)Y''(y) + k^2 n_1^2 X(x)Y(y) = 0 \quad (7)$$

Делим на $X(x)Y(y)$ (предполагая, что X и Y не равны нулю):

$$\frac{X''(x)}{X(x)} + \frac{Y''(y)}{Y(y)} + k^2 n_1^2 = 0 \quad (8)$$

Уравнение (7) можно переписать так, чтобы одна часть зависела только от *x*. Предполагаем, что $\frac{X''(x)}{X(x)} = -\lambda$, тогда уравнение (8) принимает вид:

$$\frac{Y''}{Y} + k^2 {n_1}^2 - \lambda = 0, \quad (9)$$

Перенесем коэффициент λ в правую часть и умножим обе части уравнения на Y:

$$Y'' + k^2 {n_1}^2 Y = \lambda Y. \qquad (10)$$

Далее путем замен $Y" \to -y'', k^2 {n_1}^2 \to q(x),$ получим уравнение Штурма-Лиувилля в виде уравнения (11).

$$-y'' + q(x)y = \lambda y. \qquad (11)$$

Это уравнение представляет собой типичное дифференциальное уравнение Штурма-Лиувилля, в котором функция и её вторая производная связаны линейным соотношением. Это уравнение может быть решено для определенных граничных условий, в зависимости от физической ситуации (например, ограничения на *X* и *Y* на границах области).

Этот подход позволяет анализировать и находить решения в форме стоячих волн или волн, распространяющихся в определенных направлениях, в зависимости от выбора константы λ и свойств среды, описываемых n_1. Подробная теория о планарных волноводах представлена в приложении Б.

1.3 Математическая модель: формулировка прямой задачи Штурма-Лиувилля

Целью прямой задачи Штурма-Лиувилля является нахождение всех значений λ, для которых существует ненулевое решение $u(x)$ удовлетворяющее основному уравнению и граничным условиям.

$$-\frac{\partial^2\psi(x)}{\partial x^2}+u(x)\psi(x)=\lambda\psi(x), \qquad x\in[0;\pi], \tag{12}$$

$$\begin{cases}\gamma_1 y'(a)+\beta_1 y(a)=0, \ \gamma_1^2+\beta_1^2\neq 0\\ \gamma_2 y'(b)+\beta_2 y(b)=0, \ \gamma_2^2+\beta_2^2\neq 0'\end{cases} \tag{13}$$

где $u(x)-$ функция непрерывная на $[0;\pi]$;

$\gamma,\beta-$ произвольные вещественные числа.

Итогом решения задачи Штурма-Лиувилля являются:

Собственные значения ($\boldsymbol{\lambda_n}$):

1. Это набор чисел, при которых существует ненулевое решение уравнения Штурма-Лиувилля, удовлетворяющее заданным граничным условиям.
2. Собственные значения могут быть действительными и дискретными.

Собственные функции (Ψ):

1. Это набор функций, соответствующих собственным значениям, которые удовлетворяют основному уравнению и граничным условиям.
2. Собственные функции могут быть нормированы так, чтобы удовлетворять условию ортонормированности.

Чтобы получить полный спектр необходимо также найти нормировочный коэффициент α:

$$\alpha_n = \int_0^{\pi}\Psi^2(x)dx \tag{14}$$

Метод стрельбы позволяет получить данные рассеяния, варьируя собственное число λ для настройки пристрелки. Изменения λ производятся случайным или последовательным способом, что способствует точности определения параметров рассеяния [15].

В рамках задач Штурма-Лиувилля, когда требуется найти кусочно-постоянное приближение функции $u(x)$, можно воспользоваться следующим подходом. Этот метод обеспечивает аппроксимацию функции с заданной точностью, что важно для точности решений дифференциальных уравнений.

1. Установить начальное количество разбиений *M*, которое должно быть не меньше 1. Значение M определяет, на сколько интервалов будет разделён отрезок [0; π].

2. Делить интервал [0; π] на M равных частей с помощью точек x_i, где $i = 0 \ldots M, x_0 = 0, x_M = \pi,$

3. Определить точки ξ_j как середины каждого отрезка $[x_{j-1};\ x_j]$, где $j = 0 \ldots M,$

4. Построить кусочно-постоянную функцию $u^*(x)$ так, что $u^*(x)$ на каждом интервале равна значению $u(\xi_j)$,

5. Оценить величину δ как норму разности между исходной функцией $u(x)$ и аппроксимированной функцией $u^*(x)$: $\delta = \|u(x) - u^*(x)\|$,

6. Если δ превышает заданную относительную точность ϵ, то увеличить количество разбиений *M* на один и вернуться к шагу 2

7. если $\delta \leq \varepsilon$, завершить процесс, так как достигнута необходимая точность аппроксимации.

Далее перейдем к рассмотрению алгоритма «метод стрельбы» [15]:

1. разделить интервал [0;π] на M равных частей для дискретизации функции и задать шаг изменения собственного числа λ, обозначим этот шаг как Δ. Это позволяет контролировать точность приближения при изменении параметра λ,

2. заменить непрерывную функцию $u(x)$ на интервале [0;π] кусочно-постоянной функцией u^*, что упрощает численные расчёты,

3. выбрать такое число разбиений M, чтобы отношение максимального отклонения аппроксимированной функции от исходной к норме исходной

функции было меньше заданной погрешности ϵ, что гарантирует достаточную точность аппроксимации:

$$\frac{\|u(x) - u^*(x)\|}{\|u(x)\|} < \varepsilon \tag{15}$$

где ε – относительная погрешность.

4. начать с λ, равным минимальному значению функции $u(x)$ на отрезке [0;π], предполагая, что это может обеспечить начальное приближение к одному из собственных значений,

5. Запустить бесконечный цикл для итеративного уточнения значений λ, позволяя алгоритму динамически адаптироваться и уточнять результаты.,

6. Провести итерации от 1 до M с индексом j, на каждом шаге переоценивая функцию и параметры.

7. Вычислить функцию $\Psi(x)$ на каждом интервале, используя текущее значение $\Lambda_j = u_j + \lambda$, и выбрать соответствующую функцию (косинус или гиперболический косинус) в зависимости от знака Λ_j.

8. На первой итерации ($j = 1$) использовать начальные условия для определения констант A и B.:

$$\Psi(0) = 1, \tag{16}$$

$$\Psi'(0) = -Q_0, \tag{17}$$

где Q_0– левое граничное условие.

при $\Lambda_j > 0$:

$$\Psi(x) = A \times cos(\sqrt{\Lambda} * x) + B \times sin(\sqrt{\Lambda} * x) \tag{18}$$

при $\Lambda_j < 0$:

$$\Psi(x) = A \times cosh(\sqrt{\Lambda} * x) + B \times sinh(\sqrt{\Lambda} * x) \quad (19)$$

9. На последующих итерациях решать систему линейных алгебраических уравнений (СЛАУ) для нахождения этих констант, учитывая переходы между различными функциональными формами $\Psi(x)$:

при $\Lambda_{j-1} > 0$ и $\Lambda_j > 0$:

$$A_{j-1} \times \cos\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sin\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = A_j \times \cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (20)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times sin(\sqrt{\Lambda_{j-1}} \times x_{j-1}) + B_{j-1} \times \Lambda_{j-1} \times cos(\Lambda_{j-1} \times x_{j-1}) = -A_j \times \sqrt{\Lambda_j} \times sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right). \quad (21)$$

при $\Lambda_{j-1} > 0$ и $\Lambda_j < 0$:

$$A_{j-1} \times \cos\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sin\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) = A_j \cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (22)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times sin(\sqrt{\Lambda_{j-1}} \times x_{j-1}) + B_{j-1} \times \Lambda_{j-1} \times cos(\Lambda_{j-1} \times x_{j-1}) = -A_j \times \sqrt{\Lambda_j} \times sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right). \quad (23)$$

при $\Lambda_{j-1} < 0$ и $\Lambda_j > 0$:

$$A_{j-1} \times \cosh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) =$$
$$= A_j \times \cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (24)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \Lambda_{j-1} \times cosh(\Lambda_{j-1} \times x_{j-1})$$
$$= \quad (25)$$
$$= -A_j \times \sqrt{\Lambda_j} \times sin\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times cos\left(\sqrt{\Lambda_j} \times x_{j-1}\right).$$

при $\Lambda_{j-1} < 0$ и $\Lambda_j < 0$:

$$A_{j-1} \times \cosh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) =$$
$$= A_j \times \cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right), \quad (26)$$

$$-A_{j-1} \times \sqrt{\Lambda_{j-1}} \times \sinh\left(\sqrt{\Lambda_{j-1}} \times x_{j-1}\right) + B_{j-1} \times \Lambda_{j-1} \times cosh(\Lambda_{j-1} \times x_{j-1})$$
$$= \quad (27)$$
$$= -A_j \times \sqrt{\Lambda_j} \times sinh\left(\sqrt{\Lambda_j} \times x_{j-1}\right) + B_j \times \sqrt{\Lambda_j} \times cosh\left(\sqrt{\Lambda_j} \times x_{j-1}\right).$$

10. вычислить интеграл от квадрата $\Psi(x)$ на каждом интервале для оценки нормированного коэффициента $\alpha(\lambda)$:

$$\alpha(\lambda) = \int_{x_{j-1}}^{x_j} \Psi^2(x)dx \quad (28)$$

11. после выполнения расчётов на текущем интервале увеличить итерационный параметр j на один и вернуться к шагу 7,

12. по завершении всех итераций цикла от 1 до M необходимо рассчитать следующие значения:

$$a = \Psi'(\pi - 0) + Q_\pi(\pi - 0), \quad (29)$$

$$\alpha = \sum_{j=1}^{m} \alpha_j. \quad (30)$$

13. если параметры a на соседних интервалах, a_{i-1} и a_i, имеют разные знаки, то это указывает на наличие собственного значения λ в интервале между λ_{i-1} и λ_i. Собственное значение λ, в этом случае можно найти с помощью линейной интерполяции. Это классический признак изменения знака функции и нахождения корня в пределах данного интервала,

14. проверить, достигнута ли необходимая точность: если относительная ошибка между количеством найденных собственных чисел N и корнем квадратным из последнего найденного собственного числа $\sqrt{\lambda_i}$ меньше заданного порога ϵ, то процесс можно завершить. Формула проверки: $\frac{\|N-\sqrt{\lambda_i}\|}{\|\sqrt{\lambda_i}\|} < \varepsilon$,

15. Если условие завершения цикла не выполнено, увеличить значение λ на шаг Δ и повторить процесс, начиная с пункта 6. Это позволяет динамически адаптировать процесс и продолжить поиск других собственных значений.

1.4 Математическая модель: формулировка обратной задачи Штурма- Лиувилля

Обратная задача Штурма-Лиувилля представляет собой нахождение потенциала $u^*(x)$, с помощью заданных собственных значений и нормировочных коэффициентов и данных, полученных входе решения уравнения (31) для подобранной $u(x)$.

$$-\frac{\partial^2\psi(x)}{\partial x^2} + u^*(x)\psi(x) = \lambda\psi(x), \qquad x \in [0;\pi], \quad (31)$$

где $u^*(x)$ – искомая функция;

λ – собственные числа.

«Поставленную задачу можно решить при помощи оптимизации с применением градиентных методов» [15], целью которых является минимизация разницы между вычисленными и известными спектральными данными.

Начальные данные обратной задачи Штурма-Лиувииля представляют собой полный набор спектральных данных $\{(\xi_i, \beta_i)\}_{i=1}^n$, с помощью которого в дальнейшем будет восстановлена исходная функция $u^*(x)$.

Исходя из априорных знаний о возможной форме $u^*(x)$, выбирается функция $u(x)$, наиболее близкая к $u^*(x)$. Эта подобранная функция $u(x)$ используется получения полного набора спектральных данных $\{(\lambda_i, \alpha_i)\}_{i=1}^n$, после решения прямой задачи Штурма-Лиувилля, они будут использоваться в дальнейшем для минимизации невязки.

Невязка спектральных данных – это расхождение между теоретически ожидаемыми и фактически полученными спектральными значениями. Она используется для оценки точности заданной функции и корректировки $u(x)$. Для минимизации невязки применяется градиент невязки, который будет использоваться для корректировки собственных значений, нормировочных коэффициентов и непосредственно значений подобранной функции. Процесс минимизации будет продолжаться до достижения заданной точности $\Delta\lambda$ для $u(x)$ и $u^*(x)$, что будет указывать на успешное приближение к искомой функции.

Алгоритм решения обратной задачи Штурма-Лиувииля включает в себя следующие шаги:

1) необходимо определить изменение вызванной изменением функции $u(x)$ на δu для уменьшения невязки. Невязку можно приближенно определить как линейную функцию изменений собственных значений [16]:

$$\Delta(u + \delta u) - \Delta(u) = \frac{1}{2}\left[\sum_{k=1}^{N}(\lambda_k + \delta\lambda_k - \xi_K)^2 + \sum_{m=1}^{N}(\alpha_m - \beta_m)^2\right] -$$
$$-\frac{1}{2}\left[\sum_{k=1}^{N}(\lambda_k - \xi_K)^2 + \sum_{m=1}^{N}(\alpha_m - \beta_m)^2\right] \approx \sum_{k=1}^{N}(\lambda_k - \xi_K) \times \delta\lambda_k \approx \quad (32)$$
$$\approx \sum_{k=1}^{n}\frac{1}{\alpha_k}(\lambda_k - \xi_k) \times (\psi_k, \delta u\psi_k),$$

В данном контексте δu указывает на приращение аргумента. Собственные числа, соответствующие подобранной функции, обозначаются λ_k, в то время как собственные числа для искомой функции обозначаются ξ_k. Под собственными функциями подобранной функции понимаются Ψ_k, а нормировочные коэффициенты обозначаются как α_k

2) для каждого отрезка подобранной функции непроходимо вычислить, градиент невязки используя линейную часть приращения невязки из шага 1 :

$$g_m = \sum_{k=1}^{N}\frac{1}{\alpha_k}(\lambda_k - \xi_k) \times (\psi_k, \chi_m\psi_k) \quad (33)$$

При этом χ представляет собой характеристическую функцию отрезка $[x_{i-1}; x_i]$. Собственные числа, относящиеся к подобранной функции, обозначаются как λ_k, а собственные числа искомой функции – как ξ_k.

3) обновить функцию $u(x)$ с использованием вычисленного градиента:

$$u_0(x) \rightarrow u_1(x) = u_0 + \delta u_0, \tag{34}$$

$$\delta u_0 = -\varepsilon \sum_{m=1}^{M} g_m \times \chi_m(x). \tag{35}$$

где ε – коэффициент скорости обучения или шаг градиентного спуска.

4) адаптировать собственные значения λ в соответствии с изменением функции:

$$\lambda_i \rightarrow \lambda_i + \delta\lambda_i, \tag{36}$$

$$\delta\lambda_i = \frac{1}{\alpha_i}(\psi_i, \delta u \psi_i) \tag{37}$$

5) обновить нормировочные коэффициенты α, учитывая взаимодействие между разными собственными функциями:

$$\alpha_i \rightarrow \alpha_i + \sum_{k \neq i} \alpha_k \times |\eta_{ik}|^2, \tag{38}$$

$$\eta_{ik} = \frac{(\Psi_m, u(x) \times \Psi_i)}{\alpha_i(\lambda_i - \lambda_k)}. \tag{39}$$

6) оценить обновленную невязку после коррекции:

$$\Delta(u) = \frac{1}{2}\left(\sum_{k=1}^{N} (\lambda_k - \xi_k)^2 + \sum_{m=1}^{N} (\alpha_m - \beta_m)^2 \right). \tag{40}$$

7) если невязка меньше заданного порога δ, завершить алгоритм. Это указывает на достижение достаточной точности приближения искомой функции $u^*(x)$.

Таким образом, разработана математическая модель для решения обратной задачи Штурма-Лиувилля и описан алгоритм градиентного метода,

применимого на заданном интервале. «Градиентный метод выступает в качестве основного инструмента для итеративного поиска функции $u(x)$, который через последовательные приближения стремится минимизировать различие между известными спектральными данными для исходной функции $u^*(x)$ и данными, полученными для текущего приближения $u(x)$. Этот процесс включает в себя расчёт градиента функционала ошибки по $u(x)$ и обновление $u(x)$ на каждой итерации с целью снижения ошибки» [24]. Представленный подход позволяет систематически улучшать оценку потенциала в рамках установленных моделью предположений.

В первой главе ВКР был проведён всесторонний анализ литературы, посвящённый исследованию обратных и некорректных задач, а также принципам работы планарных волноводов. Были рассмотрены ключевые аспекты задач Штурма-Лиувилля, включая различные методологии, применяемые для их решения. Также в главе представлено описание математических моделей: обобщённая модель планарного волновода, а также математическая модель, основанная на решении прямой и обратной задачи Штурма-Лиувилля в приложении к теории планарного волновода. Описанные модели включают в себя как теоретические основы, так и практические алгоритмы для решения прямых и обратных задач. Результаты обзора литературы и сформулированные математические модели создают необходимую основу для последующих исследований и экспериментальной работы, которая будет описана в следующих разделах работы.

2 Реализация алгоритма решения обратной задачи Штурма-Лиувилля

2.1 Методы исследования

Для выполнения поставленной задачи используется современное оборудование и программное обеспечение. Основу вычислительной мощности обеспечивают процессоры AMD Ryzen 5 2500U и AMD Ryzen 5 4600H, работающие на частоте 3.00 GHz. В совокупности с 16 ГБ оперативной памяти и операционной системой Windows 11, они обеспечивают необходимую производительность и стабильность работы.

Среда разработки представлена Visual Studio Code, популярным инструментом для программирования, который поддерживает множество языков и расширений. В качестве основного языка программирования используется Python версии 3.12.0, что позволяет эффективно решать широкий спектр задач.

Для удобства и выполнения сложный математических операций используются следующие библиотеки: Math, Numpy, Matplotlib, Scipy, Sympy

Numpy предоставляет поддержку больших массивов и матриц, что оптимизирует вычисления. Matplotlib позволяет создавать графики, обеспечивая визуальное представление данных. Scipy используется для научных и технических вычислений, охватывая обработку сигналов и оптимизацию. Sympy подходит для символьных математических расчетов, позволяя точно манипулировать формулами.

Этот набор библиотек представляет собой интегрированное решение для математического моделирования и решения прямых и обратных задач Штурма-Лиувилля. Благодаря своей способности к обработке массивов, символьным вычислениям и оптимизации, эти инструменты позволяют точно и эффективно моделировать физические системы и их свойства. Применение

такого комплекса в математических исследованиях ускоряет процесс решения задач, упрощает анализ данных и способствует более глубокому пониманию динамических систем.

2.2 Аппроксимация исходной функции

Блок-схема аппроксимации функции u(x) изображенная на рисунке 3 включает в себя итерационный цикл аппроксимации, который продолжается до тех пор, пока евклидова норма, измеряющая отклонение, не станет меньше заданной точности. На каждом этапе итерации интервал $[0;\pi]$ делится на M частей. Затем рассчитывается евклидова норма для текущей аппроксимации. Если текущая евклидова норма превышает установленную точность, то количество разбиений увеличивается, и процесс повторяется. Цикл завершается, когда евклидова норма становится меньше или равна заданному порогу точности. Полное описание алгоритма и исходный код программы детально представлены в приложении А.

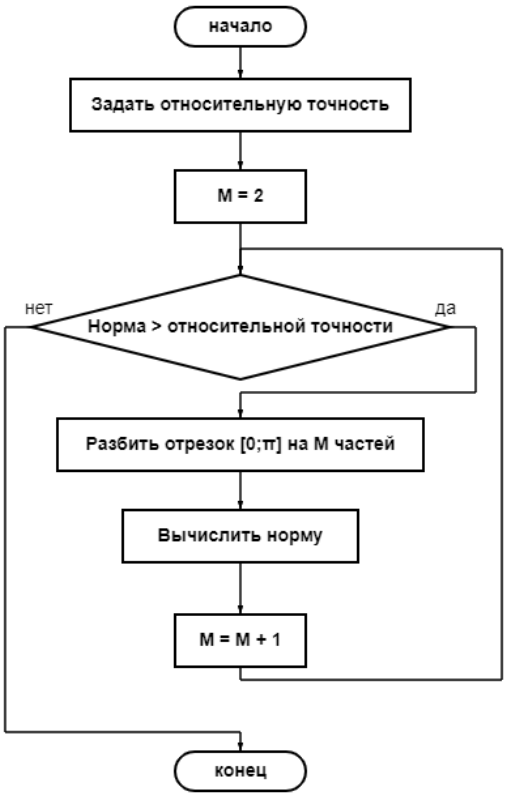

Рисунок 3 – Алгоритм аппроксимации исходной функции

Приведем фрагмент кода программы, представленный на рисунке 4, в котором в цикле отрезок $[0; \pi]$ разбивается на части и вычисляется норма.

```
while delta > e:
  kolIter += 1
  xx = np.linspace(a, b, m)
  fx = y(xx) xjminus1 = xx[0] sum1 = 0
  sum2, err = integrate.quad(f2, a, b)
  for x in xx[1:]:
    Ej = (xjminus1 + x)/2
    v1, err = integrate.quad(f1, xjminus1, x)
    sum1 += v1
    xjminus1 = x
  delta1 = pow(sum1, 1/2)
  delta2 = pow(sum2, 1/2)
  delta = de
```

Рисунок 4 – Фрагмент кода, разбиение отрезка на части и вычисление нормы

Парабола задана на отрезке $[0; \pi]$ с относительной точностью 15%. Итоговые результаты представлены на рисунке 5. Количество разбиений отрезков M составляет 10.

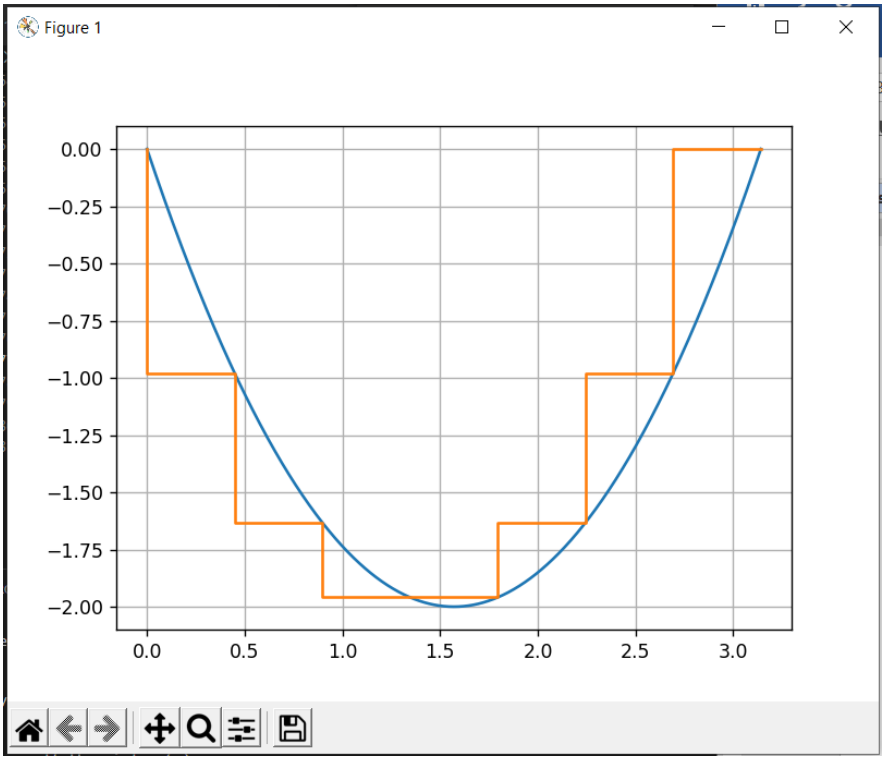

Рисунок 5 – Аппроксимация исходной функции

Таким образом, функция разбивается на M отрезков в соответствии с заданной точностью и выполняется аппроксимация.

2.3 Реализация прямой задачи Штурма-Лиувилля на интервале

Метод стрельбы является численным для. Этот метод включает несколько последовательных этапов, каждый из которых играет ключевую роль в достижении точного решения. Процесс начинается с аппроксимации начальной функции, которая разбивается на соответствующие сегменты. В

ходе этого создаются массивы для хранения данных рассеяния, причем точность решения определяется количеством этих данных. Инициализация начинается с установления начальных значений параметра λ, минимального значения функции, и создания массивов для данных. Затем начинается основной цикл, продолжающийся до выполнения условий завершения. «В рамках этого цикла выполняется итерационный процесс с параметром j, изменяющимся от 1 до M. На каждом шаге определяется знак переменной Λ. В начальной итерации константы A и B вычисляются по уравнениям 14 и 15. В последующих итерациях решение корректируется, и константы A и B определяются через систему линейных уравнений» [15]. В конце каждой итерации алгоритма происходит проверка на наличие решения для найденного значения λ. Если решение есть, то с помощью линейной интерполяции ищется собственное значение между последними найденными значениями. Алгоритм завершает свою работу, когда выполнены все условия завершения, что указывает на нахождение требуемого решения. В противном случае значение λ увеличивается, и процесс повторяется на следующем интервале.

Описанный метод в виде блок-схемы на рисунке 6 использует итеративные циклы и точные вычисления для достижения нужного результата, обеспечивая высокую точность и надежность данных. Полное описание метода и исходный код приводятся в приложении Б, что позволяет понять детали его реализации и применения.

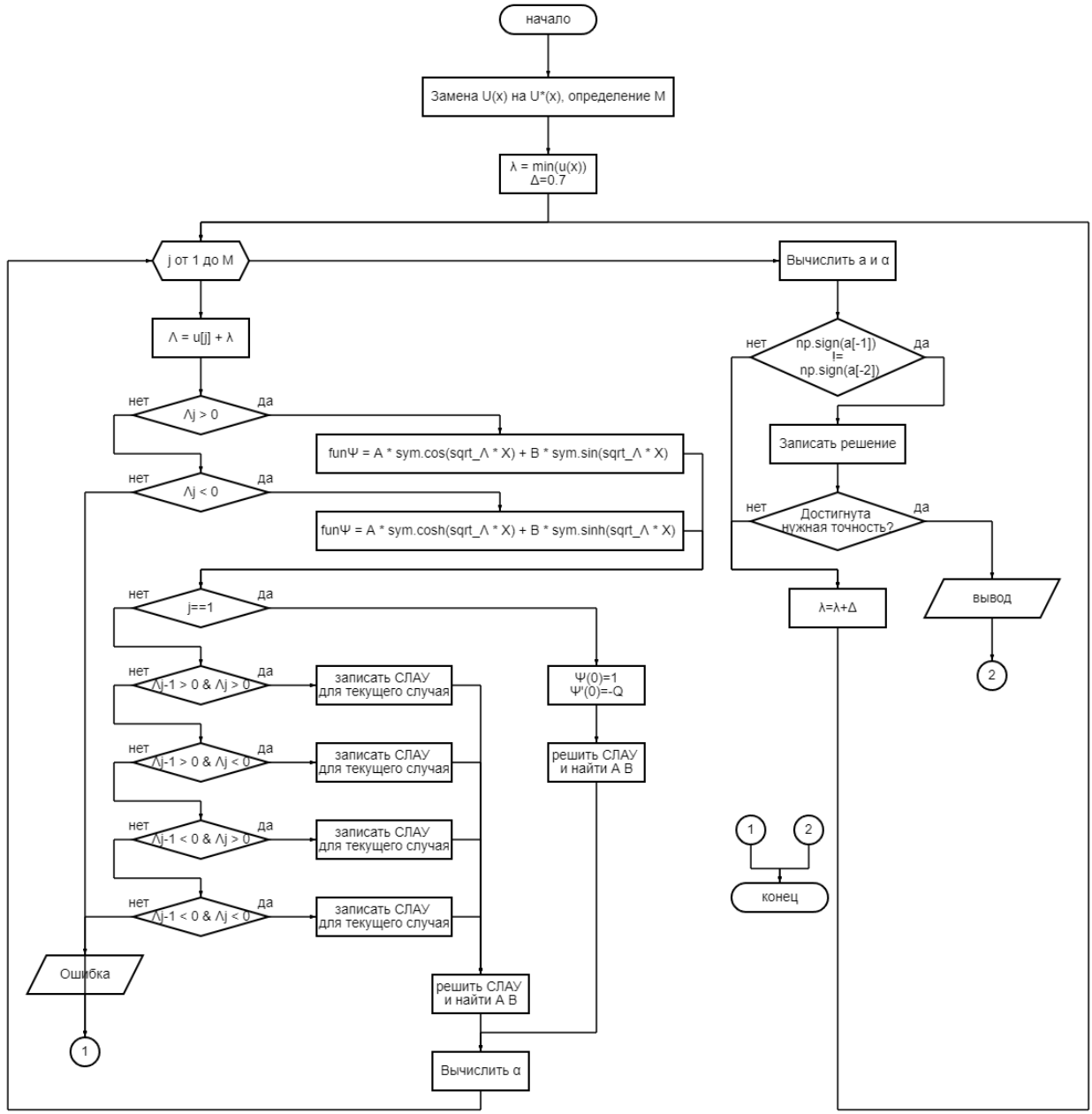

Рисунок 6 – Блок схема алгоритма метода стрельбы

На рисунке 7 приведен фрагмент кода, который определяет вид собственной функции исходя из знака переменной Λ.

```
Λ = u[j] + λ
sqrt_Λ = np.sqrt(abs(Λ))
j_1 = j - 1
A, B, X = symbols('A, B, X')
if Λ > 0:
   funΨ = A * sym.cos(sqrt_Λ × X) + B * sym.sin(sqrt_Λ * X)
      def Ψ(x):
         return A * sym.cos(sqrt_Λ * x) + B * sym.sin(sqrt_Λ * x)
if Λ < 0:
   funΨ = A * sym.cosh(sqrt_Λ * X) + B * sym.sinh(sqrt_Λ * X)
      def Ψ(x):
   return A * sym.cosh(sqrt_Λ * x) + B * sym.sinh(sqrt_Λ * x)
```

Рисунок 7 – Определение собственной функции

Фрагмент кода, в котором определяются константы A и B, на первой итерации алгоритма представлен на рисунке 8.

```
if j == 1:
 A, B, X = symbols('A, B, X')
 funΨ = funΨ - 1
 funΨdx = sym.diff(funΨ, X)
 funΨdx = funΨdx + Q0
 funΨ = funΨ.subs(X, 0)
 funΨdx = funΨdx.subs(X, 0)
 A, B = list(sym.solve([funΨ,  funΨdx], A, B).values())
```

Рисунок 8 – Определение A и B

В случае если итерация алгоритма ≠ 0, то константы находятся с помощью СЛАУ, представленными на рисунке 9, 10.

```
else:
  sqrt_Λ_minus1 = np.sqrt(abs(array_Λ[-1]))
  A, B = symbols('A, B')
  if array_Λ[-1] > 0 and Λ > 0:
    A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A *
sym.cos(sqrt_Λ * x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]),
-array_A[-1] * sqrt_Λ_minus1 * sym.sin(sqrt_Λ_minus1 * x[j_1]) +
array_B[-1] * sqrt_Λ_minus1 * sym.cos(sqrt_Λ_minus1 * x[j_1]) + A *
sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B * sqrt_Λ * sym.cos(sqrt_Λ *
x[j_1])], A, B).values())
  elif array_Λ[-1] > 0 and Λ < 0:
    A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A *
sym.cosh(sqrt_Λ * x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]),
-array_A[-1] * sqrt_Λ_minus1 * sym.sin(sqrt_Λ_minus1 * x[j_1]) +
array_B[-1] * sqrt_Λ_minus1 * sym.cos(sqrt_Λ_minus1 * x[j_1]) + A *
sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) - B * sqrt_Λ * sym.cosh(sqrt_Λ *
x[j_1])], A, B).values())
  elif array_Λ[-1] < 0 and Λ > 0:
    A, B = list(sym.solve([array_A[-1] * sym.cosh(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 * x[j_1]) - A *
sym.cos(sqrt_Λ * x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]),
-array_A[-1] * sqrt_Λ_minus1 * sym.sinh(sqrt_Λ_minus1 * x[j_1]) +
array_B[-1] * sqrt_Λ_minus1 * sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A *
sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B * sqrt_Λ * sym.cos(sqrt_Λ *
x[j_1])], A, B).values())
```

Рисунок 9 – Решение прямой задачи

```
  elif array_Λ[-1] < 0 and Λ < 0:
    A, B = list(sym.solve([array_A[-1] * sym.cosh(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 * x[j_1]) - A *
sym.cosh(sqrt_Λ * x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]),
-array_A[-1] * sqrt_Λ_minus1 * sym.sinh(sqrt_Λ_minus1 * x[j_1]) +
array_B[-1] * sqrt_Λ_minus1 * sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A *
sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) - B * sqrt_Λ * sym.cosh(sqrt_Λ *
x[j_1])], A, B).values())
```

Рисунок 10 – Продолжение решения прямой задачи

В ходе цикла вычисляем нормировочные коэффициенты, реализация представлена на рисунке 11. Далее суммируем результаты и находим коэффициенты α на промежутке, что видно на рисунке 12.

```
def Ψ_2(x):
  return Ψ(x)**2
integr, err = integrate.quad(Ψ_2, x[j_1], x[j])
array_α.append(integr)
```

Рисунок 11 – Пункт 10 в алгоритме «метод стрельбы»

```
x_symbol = sym.Symbol('x')
Ψdx = sym.diff(Ψ(x_symbol))
a_ = float(Ψdx.subs(x_symbol, π) + Qπ * Ψ(π))
a.append(a_)
α_ = sum(array_α)
α.append(α_)
```

Рисунок 12 – Определение нормировочного коэффициента α и параметр a

Для проверки наличия решения в интервале между λ_{i-1} и λ_i, нужно сравнить последние два параметра a, если их знаки отличаются, то решение есть. Реализация на рисунке 13.

```
if (len(a) >= 2):
  left = a[len(a)-2]
  right = a[len(a)-1]
  if sign(left) != sign(right):
    print("λ=", λ, "a=", a_, "α(λ)=", α_)
```

Рисунок 13 – Проверка наличия решения

Условие выхода из цикла программы в виде кода описаны на рисунке 14. Параметр diff рассчитывается по формуле $\frac{\|N-\sqrt{\lambda_i}\|}{\|\sqrt{\lambda_i}\|} < \varepsilon$, где ε – eps_shoot относительная точность, заданная пользователем.

```
if count_λ > 0 and λ > 0:
  diff = abs(
    count_λ+1 - math.sqrt(abs(λ_graph[-1]))) /
math.sqrt((abs(λ_graph[-1])))
  print("λ=", λ, " /a=", a_, " /α(λ)=", α_, " / условие выхода из
цикла (",
          diff, "<=", eps_shoot, ") / количество λ count_λ=",
count_λ)
  if diff <= eps_shoot:
    break
```

Рисунок 14 – Определение условия выхода из цикла

В случае если выхода из цикла не произошло, алгоритм начинается по заново, но увеличивая значение λ. По завершению программы возвращается полный спектр данных для прямой задачи Штурма-Лиувилля. Для проверки алгоритма возьмем функция, представленная параболой на отрезке $[0; \pi]$ с минимумом $y = -3$ и точностью 15% представлен. Собственные значения и нормировочные коэффициенты в таблице 1.

Таблица 1 – Данные рассеяния

№	Собственное число (λ)	Нормировочный коэффициент (α)
1	1.247	1.625
2	6.675	1.944
3	13.577	1.658
4	22.547	1.485
5	33.565	1.496
6	46.526	1.535
7	61.571	1.511
8	78.711	1.461
9	97.596	1.444
10	118.488	1.427

На рисунке 15 представлен график демонстрирующий, как параметр стрельбы изменяется в зависимости от нормировочного коэффициента. Волнообразная форма графика с возрастающей длиной и амплитудой свидетельствует о сложной динамике системы. Красные точки на графике четко показывают найденные собственные значения, что позволяет легко проследить их распределение. Важно отметить, что с увеличением числа найденных собственных значений, разность между последними значениями становится всё больше, что может указывать на увеличение сложности решения или изменения в поведении системы.

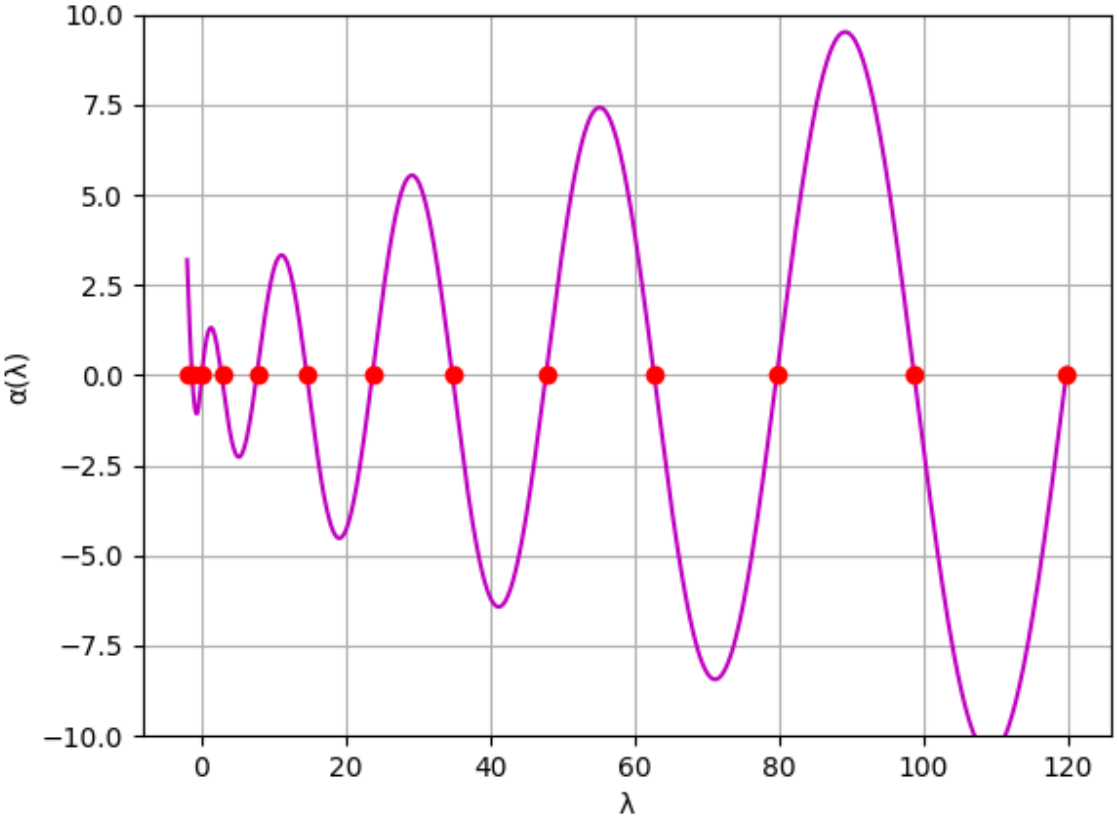

Рисунок 15 – График зависимости параметра стрельбы от нормировочного коэффициента

Так же приведем графики собственных функций от собственных значений. На рисунке 16, видно, что по мере увеличения собственного значения , собственная функция уменьшает амплитуду и увеличивает частоту.

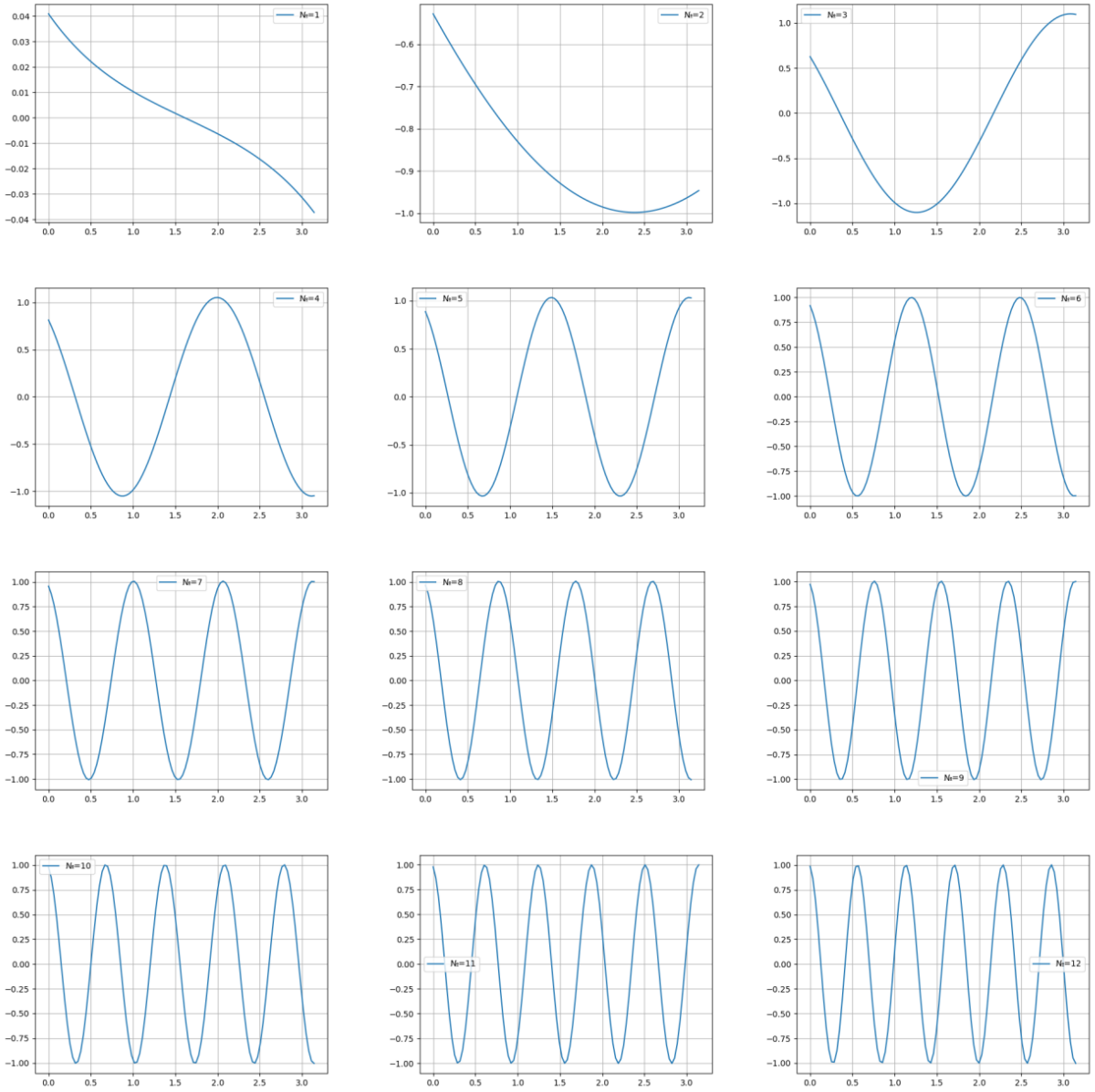

Рисунок 16 – Двенадцать найденных собственных функций

При помощи данной программы был успешно реализован метод «стрельбы» для решения прямой задачи Штурма-Лиувииля на отрезке $[0; \pi]$.

2.4 Реализация обратной задачи Штурма-Лиувилля на интервале

Для решения обратной задачи Штурма-Лиувилля используется следующий алгоритм. Вначале необходимо собрать два набора спектральных данных. Первый набор $(\xi_1, \beta_1, \dots, \xi_n, \beta_n)$ данных задан по условию задачи последовательностью или решением прямой задачи для $u^*(x)$. Второй набор $(\lambda_1, \alpha_1, \dots, \lambda_n, \alpha_n)$ второй набор получается путем решения прямой задачи для подобранной начальной точки $u(x)$. Эти два набора спектральных данных являются основой для анализа и сравнения, что позволяет достичь точного восстановления исходной функции. Спектральные данные первого набора, полученные для функции $u^*(x)$, и второго набора, рассчитанные для функции $u(x)$, дают возможность провести анализ и выявить невязку межу наборами данных. Благодаря этому процессу можно обеспечить корректное и точное восстановление исходной функции. Алгоритм, подробно описанный и представленный в виде блок-схемы на рисунке 17, демонстрирует каждый этап решения обратной задачи.

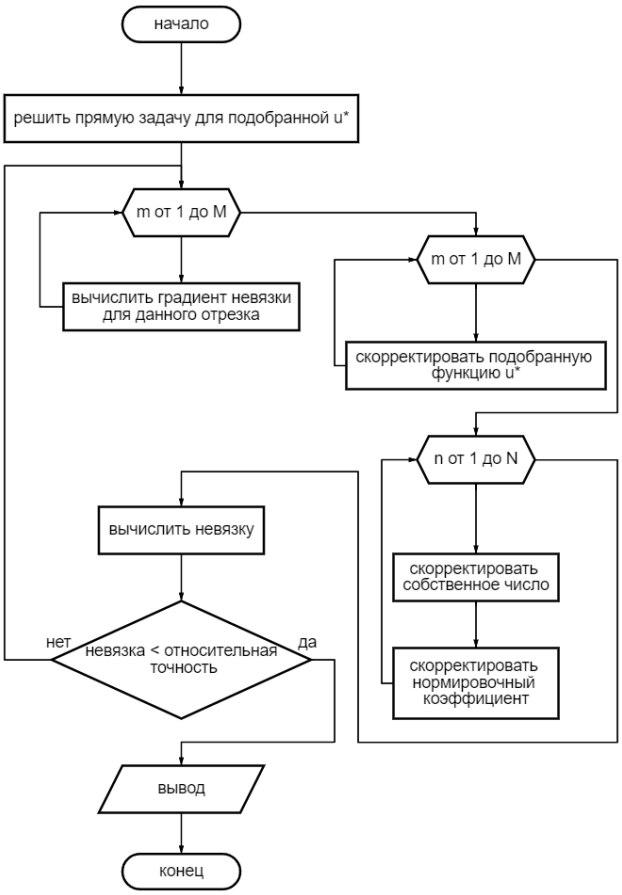

Рисунок 17 – Алгоритм решения обратной задачи

Для нахождения невязки рассчитывается градиент по формуле (33) из алгоритма решения обратной задачи. Градиент строится для каждого интервала кусочно-постоянной функции и с помощью него производиться корректировки для значений $u(x)$. После с помощью вычисленных новых, скорректированных значений функции находим обновленные собственные значения и нормировочные коэффициенты (36) и (38).

Приведенный программный код на рисунке 18 иллюстрирует вычисление градиента невязки. Алгоритм выполняет вычисление суммы согласно формуле (33), приведенной в пункте 2 решения обратной задачи. Каждый отрезок на этом этапе соответствует элементу вектора градиента невязки, что позволяет точно определить и минимизировать отклонения.

```
_g = []
for m in range(1, M):
  _sum = 0
  _rangeX = [arrayX[m-1], arrayX[m]]
  _left = _rangeX[0]
  _right = _rangeX[1]
  def _Xm(_x):
    if (_left  <= _x <= _right):
      return 1
    else:
      return 0
  for k in range(N):
    _Ψ = spectr_arr_2[k].Ψ
    _α = spectr_arr_2[k].α
    _λ = spectr_arr_2[k].λ _ε = arr_ε[k]
    def _fun_integrate(_x):
      return _Ψ(_x) * (_Xm(_x)*_Ψ(_x))
    _integrate, err = integrate.quad(_fun_integrate, a_left,
b_right)
    _sum += (1/_α) * (_λ - _ε) * _integrate
  _g.append(_sum)
Gm = _g
```

Рисунок 18 – Вычисление градиента невязки

На рисунке 19 представлена функция для нахождения приращения значений функции для дальнейшей корректировки подобранной функции $u(x)$, собственных чисел λ и коэффициентов α.

```
def δU(_x):
  _sum = 0
  for m in range(1, M):
    _rangeX = [arrayX[m-1], arrayX[m]]
    _left = _rangeX[0]
    _right = _rangeX[1]
    def _Xm(_x):
      if (_left  <= _x <= _right):
        return 1
      else:
        return 0
    _sum += Gm[m-1] * _Xm(_x)
  return -revers_e *_sum
```

Рисунок 19 – Функция для корректировки подобранной функции

Код, приведенный на рисунке 20, показывает каким образом корректируется значения подобранной функции.

```
for i in range(0, len(arrayU)):
  arrayU[i] = arrayU[i] + δU(arrayX[i])
```

Рисунок 20 – Корректировка подобранной функции

Ниже приведен на рисунке 21 код программы для корректировки собственных значений и нормировочных коэффициентов по формулам (36) и (38) соответственно. При помощи обновленных спектральных данных вычисляется невязка по формуле (40).

```
for i in range(len_λ):
  _Ψ = _spectr[i].Ψ
  _α = arr_α[i]
  _λ = arr_λ[i]
  def _funIntegr_δλ(_x):
    return _Ψ(_x) * (δU(_x)*_Ψ(_x))
  _integr, _ = integrate.quad(_funIntegr_δλ, a_left, b_right,
limit=50)
  _δλ = (1/_α)*(_integr)
  _λ = _λ + _δλ
  arr_λ[i] = _λ
  _sumA = 0
  for k in range(len_λ):
    if (i != k):
      _Ψk = _spectr[k].Ψ
      _αk = arr_α[k]
      _λk = arr_λ[k]
      def _funIntegr_aik(_x):
        return _Ψk(_x) * (U(_x)*_Ψ(_x))
      aik, err = integrate.quad(_funIntegr_aik, a_left, b_right,
limit=50)
      aik = aik/(_αk*(_λ-_λk))
      _sumA += _α * abs(aik)**2
  _α = _α + _sumA
  arr_α[i] = _α
  _ε = arr_ε[i]
  _β = arr_β[i]
  _sum_λ += (_λ - _ε)**2
  _sum_α += (_α - _β)**2
delta_Δ = 1/2*(_sum_λ+_sum_α)
round_N = 3
delta_Δ = round(delta_Δ, round_N)
arr_Δ.append(delta_Δ)
```

Рисунок 21 – Вычисление невязки с обновлёнными значениями

Для успешного решения обратной задачи Штурма-Лиувилля на заданном интервале был реализован соответствующий алгоритм. В процессе выполнения программы промежуточные результаты выводятся в консоль для отслеживания прогресса. После завершения всех вычислений приложение отображает несколько графиков в отдельных окнах. Эти графики включают: подобранную функцию $u(x)$, исходную функцию $u^*(x)$, зависимость

собственных чисел от параметра стрельбы, динамику изменения функции $u(x)$ на различных этапах, а также сравнение графиков функций $u(x)$ и $u^*(x)$ в начальный и конечный моменты.

2.5 Тестирование реализованной программы

Для проверки программы на работоспособность решим обратную задачу Штурма-Лиувилля при следующих условиях: исходные собственные значения были заданы последовательностью $-\frac{5}{\sqrt{n+1}}$. Подобранная функция $u(x) = a \cdot x(x-\pi)$, где a – параметр, зависящий от количества собственных значений n. Количество собственных чисел и нормировочных коэффициентов составляет восемь. В таблице 2 представлены спектральные данные $(\xi_1, \beta_1, \dots, \xi_n, \beta_n)$ для функции $u^*(x)$ и спектральные данные $(\lambda_1, \alpha_1, \dots, \lambda_n, \alpha_n)$ для функции $u(x)$.

Таблица 2 – Данные рассеяния функции $u*(x)$ и $u(x)$

	u(x)*		*u(x)*	
№	ε	β	λ	α
1	-5.0	1.571	-20.682	37.747
2	-3.536	1.571	-15.127	1.004
3	-2.887	1.571	-6.153	0.983
4	-2.5	1.571	-0.301	1.27

Во время работы алгоритма вычисляется невязка. Приведем результаты невязки на каждой итерации для определения правильности работы алгоритма в таблице 3. Как мы можем видеть, с каждой итерацией невязка становится меньше, что говорит о корректной работе программы.

Таблица 3 – Невязка на каждой итерации

№	Невязка
1	163.233
2	137.576
…	…
65	0.956
66	0.863

Результат восстановления функции показан на рисунке 22.

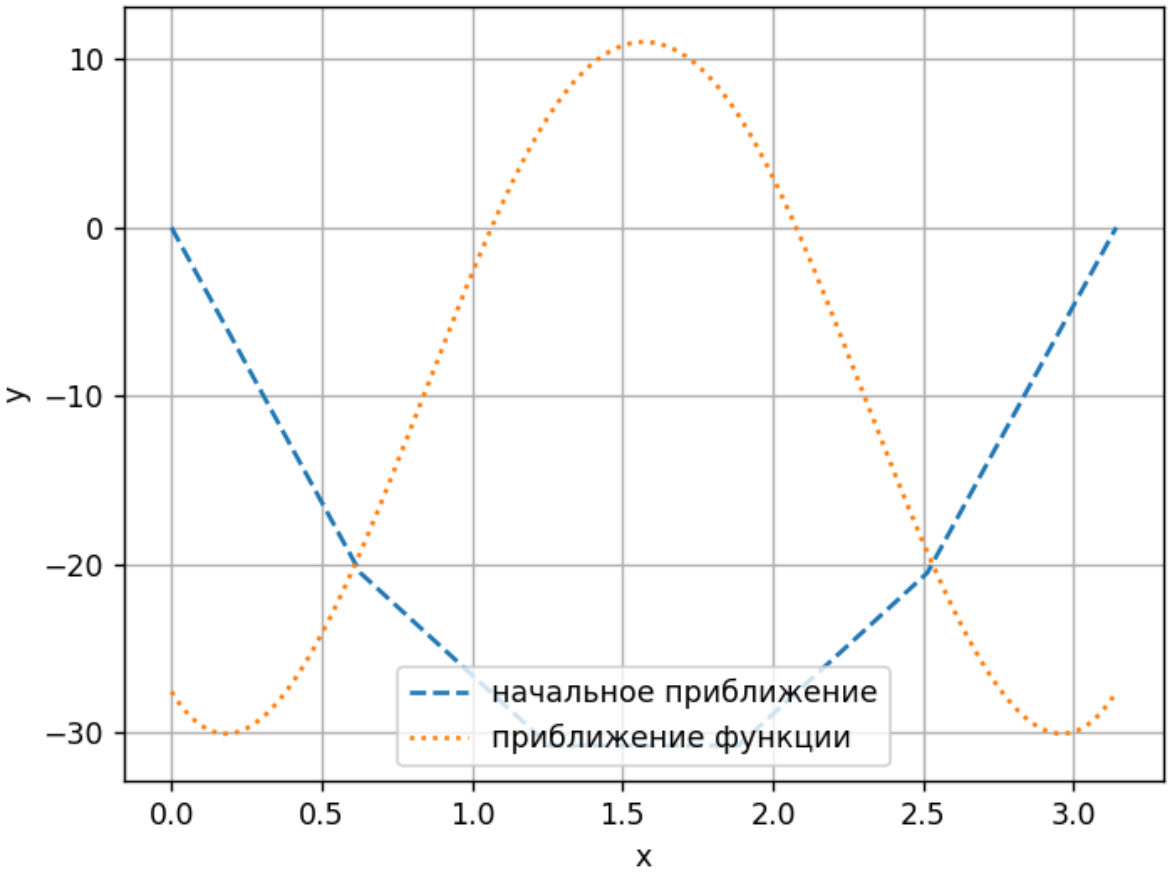

Рисунок 22 – Результат решения обратной задачи

Этапы итеративной корректировки функции представлены на рисунке 23. Нулевая итерация – это начальная точка, итерация 151 финальная.

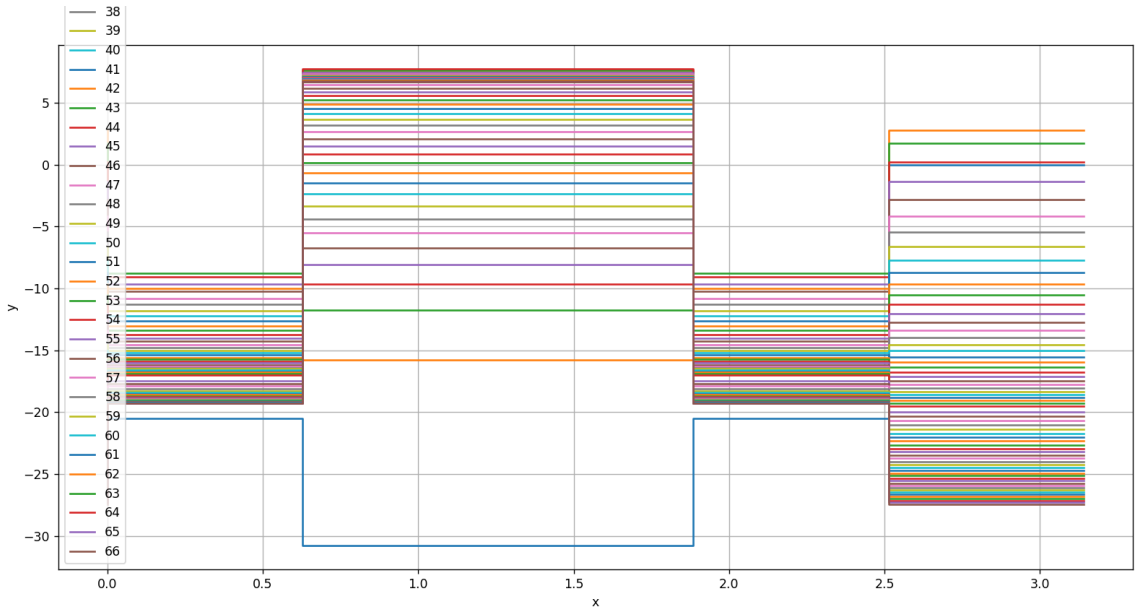

Рисунок 23 – Восстановление неизвестной функции

Возьмем другой вид спектральных данных тестирования программы восстановим коэффициентную функцию, заданную следующим образом: $\xi_n = -b_1 - b_2 n$, где $b_1 = 0.3$, $b_2 = 0.4$. Все нормировочные коэффициенты собственных значений будут равны $\beta_n = \frac{\pi}{2}$. Возьмем первые 5 пар значений ξ_n, β_n, они представлены в таблице 4.

Таблица 4 – Данные рассеяния функции $u*(x)$

№	ξ_n	β_n
1	-0.3	1.571
2	-0.7	1.571
3	-1.1	1.571
4	-1.5	1.571
5	-1.9	1.571

Для данного спектра возьмем такую же функции, как и в первом случае, но с иным параметром a, для лучшего совпадения.

Приведем в таблице 5 спектральные данные функции $u*(x)$ и $u(x)$.

Таблица 5 – Данные рассеяния функции $u*(x)$ и $u(x)$

	u(x)*		*u(x)*	
№	ε	β	λ	α
1	-0.3	1.571	-45.682	1.603
2	-0.7	1.571	-35.126	1.464
3	-1.1	1.571	-20.379	1.429
4	-1.5	1.571	-12.707	1.4495
5	-1.9	1.571	-1.227	1.4283

На рисунке 25 представлен результат работы программы. Первоначальное приближение после корректировки приобрело форму искомой функции.

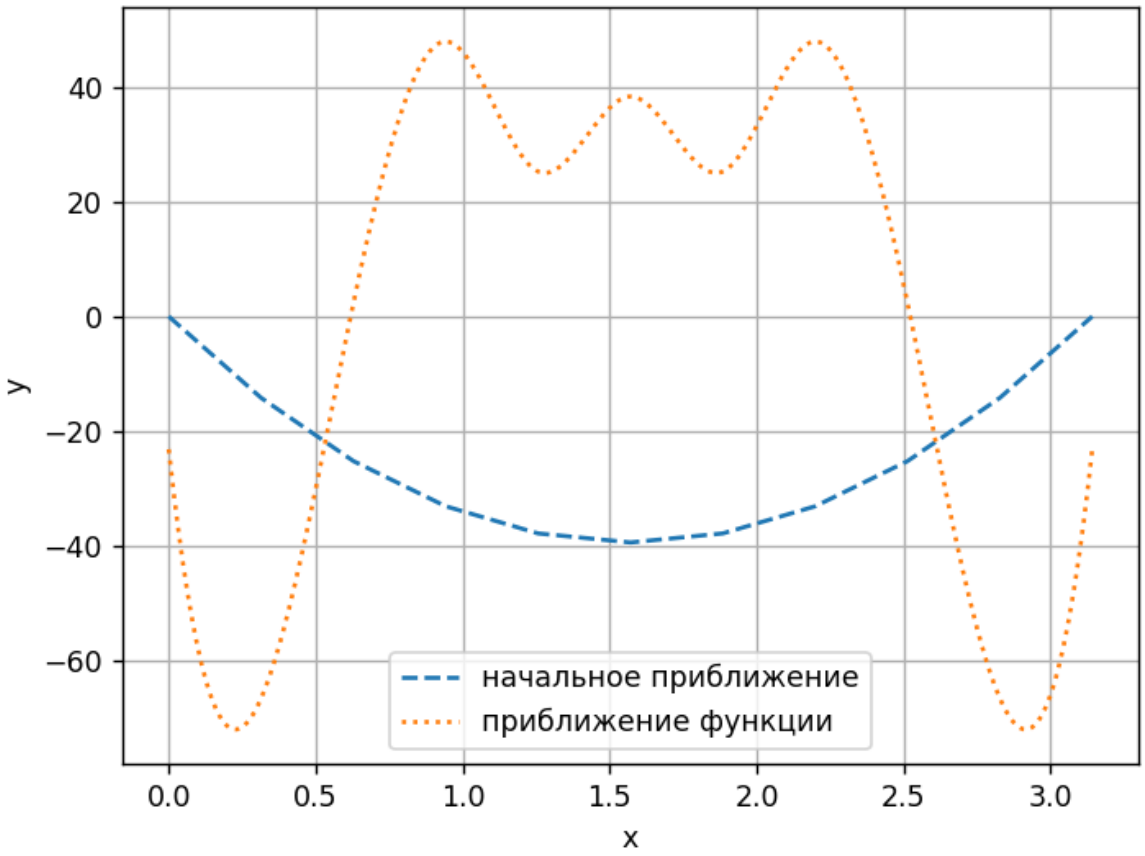

Рисунок 25 – Результат решения обратной задачи

На рисунке 26 представлена корректировка функции $u(x)$ на каждой итерации. Видно, что с каждой итерацией подобранная функция $u(x)$ всё больше приближается к исходной функции $u^*(x)$.

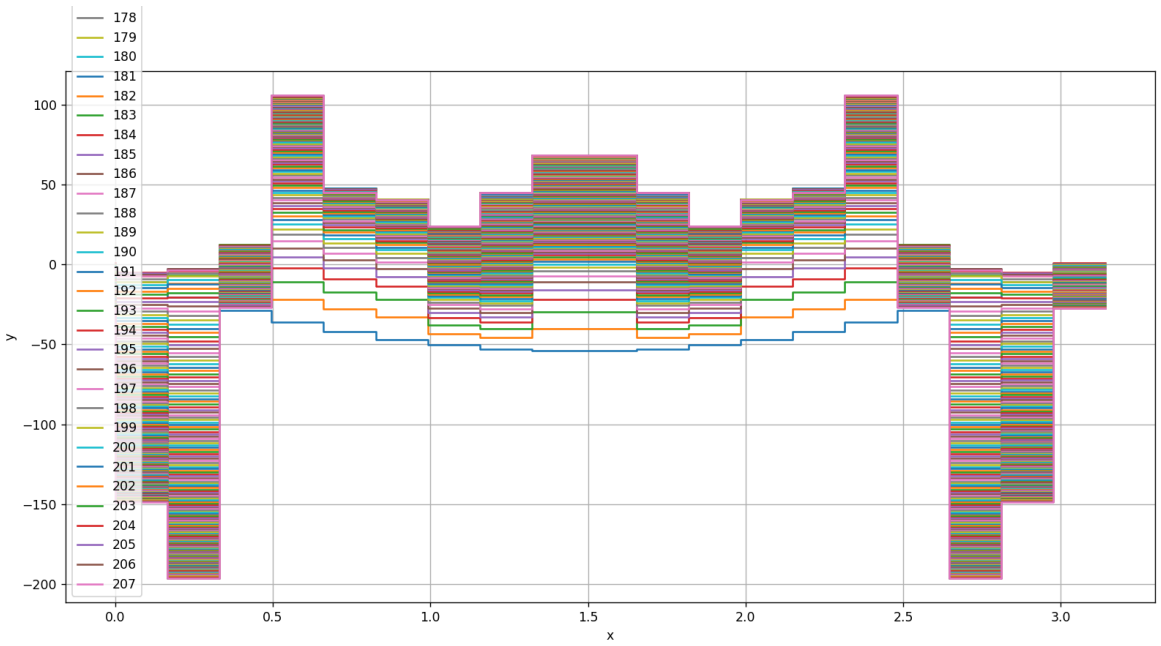

Рисунок 26 – Корректировка подобранной функции

Исходя из полученных данных невязки, которые указаны в таблице 6, можно увидеть, что с каждой итерацией сближения она убывает, что указывает что собственные значения подобранной функции приближаются к заданным значениям.

Таблица 6 – Полученные данные невязки

№ итерации	$\triangle(u)$
1	2655.00
2	2036.402
…	…
207	0.137

Для дальнейшего тестирования возьмем первый вид спектральных данных $\lambda_n = -\frac{5}{\sqrt{n+1}}$, с подобранной функций $u(x) = a \cdot x(x - \pi)$, чтобы установить влияние количества спектральных данных, точность

аппроксимации (количество разбиений M) и скорости градиентного спуска на итоговое восстановление функции. Ниже на рисунке 27 приведены результаты восстановления исходной функции при разных начальных данных.

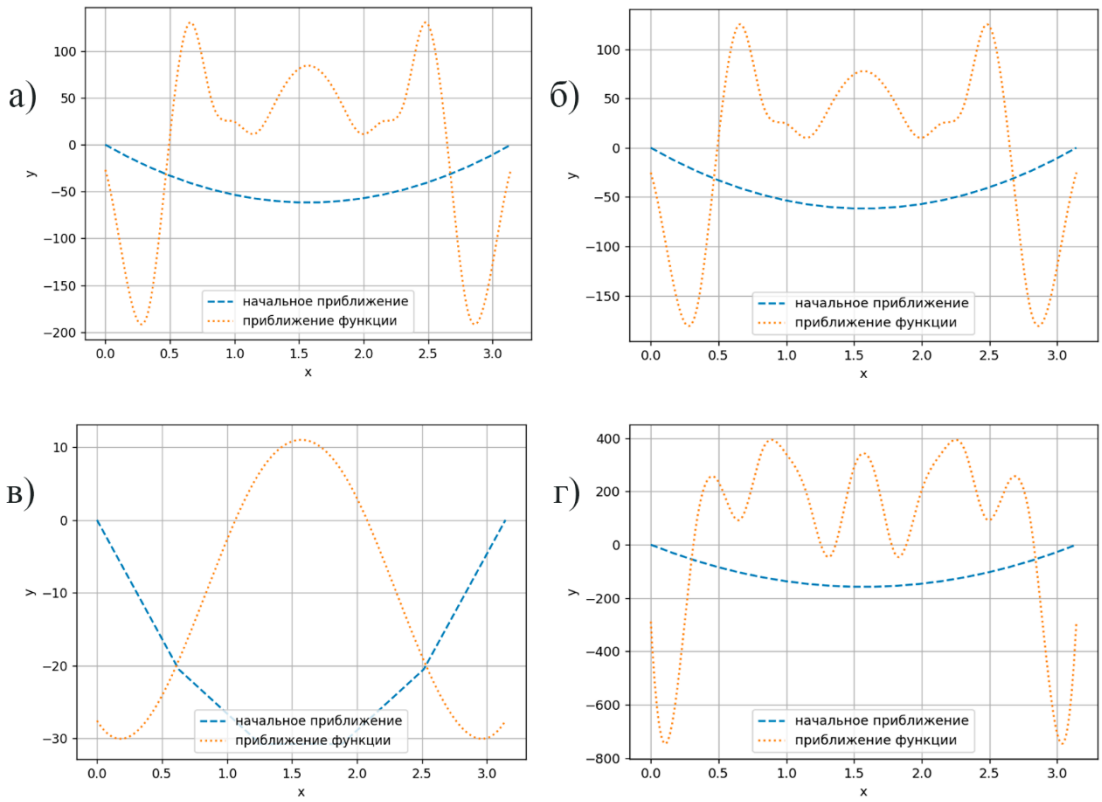

Рисунок 27 – Восстановление исходной функции используя разные начальные данные

Начальные данные для каждого варианта восстановления, представлены в таблице 7.

Таблица 7 – Начальные данные для восстановления функции

Название параметра	График а)	График б)	График в)	График г)
Количество собственных значений	6	6	4	10
Точность аппроксимации/количество разбиений M	10% / 15	10% / 15	20% / 6	5% / 20
Скорость градиентного спуска	1	1.5	0.5	2

Стоит заметить, что при увеличении количества собственных значений, сильно уменьшается минимум подобранной функции. Это связано с увеличением коэффициента «a» для того, чтобы количество отрицательных значений было равным заданному , так как при минимизации невязки $\lambda_n > 0$ не учитываются. В следствии требуется увеличить точность аппроксимации, инче невязка не достигает поставленной точности, но из-за увеличения количества разбиений M, существенно вырастает вычислительная нагрузка. Чтобы компенсировать выросшие затраты на время выполнения, нужно увеличить скорость градиентного спуска, но из-за этого падает точность приближения функции.

В данной главе описаны методы и процесс реализации алгоритма для решения обратной задачи Штурма-Лиувилля.

Был реализован алгоритм аппроксимации исходной функции, где процесс итерационно продолжается до достижения заданной точности. Метод стрельбы, используемый для решения прямой задачи Штурма-Лиувилля, демонстрирует эффективность и точность через итерационные циклы и использование линейных уравнений для нахождения собственных значений.

Реализация обратной задачи включает сбор двух наборов спектральных данных, их анализ и корректировку исходной функции до достижения точного совпадения с заданными параметрами. Алгоритм использует градиентный спуск для минимизации невязки, что позволяет корректно восстанавливать исходную функцию.

Тестирование программы показало её эффективность и точность при различных начальных условиях, продемонстрировав успешное решение задач с различными параметрами аппроксимации и скоростью градиентного спуска. Алгоритм подтвердил свою эффективность в решении данных задач.

Заключение

В процессе выполнения выпускной квалификационной работы были успешно осуществлены все этапы реализации и исследования градиентного метода решения обратной задачи Штурма-Лиувилля на заданном интервале. Были подобраны необходимые инструменты и технологии для разработки программы. Также был описан и запрограммирован алгоритм решения прямой задачи Штурма-Лиувилля с использованием метода стрельбы, а также метод подбора решения для обратной задачи.

В данной выпускной квалификационной работы можно отметить, что комплексный подход к решению обратной задачи Штурма-Лиувилля, включающий использование метода стрельбы и градиентного спуска, позволил достичь результатов в определении параметров волноводов. Применение выбранных алгоритмов и технологий разработки обеспечило точность и стабильность решений. Опыты показали, что тщательная настройка параметров алгоритма, включая начальные условия и скорость итераций, критически важна для оптимизации процесса решения и получения верных результатов. Эти находки подтверждают необходимость дальнейшего исследования влияния различных факторов на точность моделирования в таких задачах, что может способствовать улучшению методик в области физики и инженерии.

По итогу выполнения ВКР все поставленные задачи выполнены, цель достигнута. Выпускная квалификационная работа подтвердила эффективность выбранного градиентного метода и предлагаемых алгоритмов. Результаты работы могут быть полезны для дальнейших исследований и практических приложений в области математической физики и связанных дисциплин.

Список используемой литературы

1. Алифанов О. М., Артюхин Е. А. Экстремальные методы решения некорректных задач и их приложения к обратным задачам теплообмена. – «Наука,» Глав. ред. физико-математической лит-ры, 1988. – 286 с.
2. Бакушинский, А. Б. К распространению принципа невязки. – Журнал вычислительной математики и математической физики, 10(1), 210-213 с.
3. Григорьев Л.В. Кремниевая фотоника. Учебно-методическое пособие по практическим работам. –СПб: Университет ИТМО, 2015. –69 с
4. Егоров А.А. Обратная задача рассеяния лазерного излучения в интегрально-оптическом волноводе с двумерными нерегулярностями при наличии шума. – Сборник научных трудов МНТОРЭС им. А.С. Попова «Интегральная оптика и волноводная оптоэлектроника», 2015. – 30 с.
5. Емелин И. В., Красносельский М. А. Правило останова в итерационных процедурах решения некорректных задач //Автоматика и телемеханика. – 1978. – №. 12. – С. 59-63.
6. Зеленовский П. С. Основы интегральной и волоконной оптики: учебное пособие. Учебное пособие. Екатеринбург.: Издательство Уральского университета, 2019 136 с.
7. Кабанихин С. И. Обратные и некорректные задачи. Учебник для студентов высших учебных заведений. – Новосибирск: Сибирское научное издательство, 2009. – 457 с.
8. Кабанов С. Н. и др. Метод обратной задачи в теории нелинейных волн. Учебное пособие. – Саратовский государственный университет, 2013. – 115 с.
9. Карчевский М. М. Лекции по уравнениям математической физики. Учебное пособие. — 2-е изд., испр. — СПб.: Издательство «Лань», 2016. — 164 с.

10. Левитан Б. М. Обратные задачи Штурма-Лиувилля. – Наука, Главная редакция физико-математической литературы, 1984. – 240 с.

11. Марченко В. А. Операторы Штурма-Лиувилля и их приложения. – Наук. думка, 1975. – 330 с.

12. Морозов В.А. О регуляризирующих семействах операторов. – Изд-во Моск. ун-та Москва, 1967. – 93 с.

13. Никоноров Н. В., Шандаров С. М. Волноводная фотоника. Учебное пособие. Санкт Петербург, ИТМО, 2008. 142 с.

14. Самарский А. А., Вабищевич П. Н. Численные методы решения обратных задач математической физики. – Издательство ЛКИ, 2009. – 480 с.

15. Талалов С.В. Обратные и некорректные задачи. Электронное учебное пособие. ТГУ, 2019. – 61 с.

16. Талалов. С.В. Об одном варианте решения обратной задачи Штурма – Лиувилля градиентными методами. В сборнике: «Материалы IV всероссийской научной конференции с межд. участием «Информационные технологии в моделировании и управлении: подходы, методы, решения». Тольятти, 20 апреля 2022 г. ТГУ, 2022.

17. Тихонов А. Н., Арсенин В. Я. Методы решения некорректных задач. – М.: Наука, 1979. 9. Чередниченко ВГ, 1979. – 283 с.

18. У. – An International Journal of Optimiza- tion and Control: Theories & Applications (IJOCTA). – 2021. – Т. 11. – №. 2. – С. 186-198.

19. Фаддеев Л. Д. Обратная задача квантовой теории рассеяния // Успехи математических наук. – 1959. – Т. 14. – №. 4 (88. – С. 57-119.

20. Чудов Л. А. Обратная задача Штурма-Лиувилля, Матем. сб., 1949, том 25(67), номер 3, 451–456

21. Brown B. M. et al. A Numerical Procedure for the Inverse Sturm-Liouville Operator. – Factorization, Singular Operators and Related Problems. – Springer, Dordrecht, 2003. – С. 55-64.

22. Freiling G., Yurko V. A. Inverse Sturm-Liouville problems and their applications. – Huntington : NOVA Science Publishers, 2001.

23. Ledoux V. Study of special algorithms for solving Sturm-Liouville and Schrodinger equations : дис. – Ghent University, 2007. – 232 с.

24. Philip Sellars, Angelica I. Aviles-Rivero. LaplaceNet: A Hybrid Graph-Energy Neural Network for Deep Semisupervised Classification. –: IEEE, 2024.

25. Röhrl N. A least-squares functional for solving inverse Sturm– Liouville problems. – Inverse Problems. – 2005. – Т. 21. – №. 6. – С. 2009.

Приложение А

Код программы

```
# подключаем библиотеки
import math
import numpy as np
import matplotlib.pyplot as plt
from scipy import integrate
from sympy.core.symbol import symbols
from scipy.interpolate import CubicSpline
from scipy.optimize import fsolve
import sympy as sym
import warnings
import os
import joblib


def save_result(result, cache_file):
    """Сохраняет результат в файл."""
    os.makedirs(os.path.dirname(cache_file), exist_ok=True)
    joblib.dump(result, cache_file)

def load_result(cache_file):
    """Загружает результат из файла, если он существует."""
    if os.path.exists(cache_file):
        return joblib.load(cache_file)
    return None

π = math.pi
Q0 = 0
Qπ = 0
a_left = 0
b_right = π
eps_shoot = 0/100


print('Вид спектра # 1')
print('Заданные собственные значения -5/np.sqrt(n+1) : ',[round(-5/np.sqrt(n+1),3)
for n in range(0,6)])
print('Заданные нормировочные коэффициенты π/2: ',[round(π/2,3) for _ in range(0,6)])
print('Начальная точка: a*x*(x-π)')
print('Рекомендованные параметры: Скорость 1, Кол-во значений 4-6, точность
аппроксимации: 5% - 20%')
print('Вид спектра # 2')
print('Заданные собственные значения -b1-b2n ( по умолчанию b1 = 0.3 b2 = 0.4) :
',[round(-0.3-0.4*n,3) for n in range(0,6)])
print('Заданные нормировочные коэффициенты π/2: ',[round(π/2,3) for _ in range(0,6)])
print('Начальная точка: a*x*(x-π)')
print('Рекомендованные параметры: Скорость 1, Кол-во значений 5,точность
аппроксимации: 5% - 20%')
```

```
# функция аппроксимации
def aprox(delta, e, fun, a, b, m):
    y = np.vectorize(fun, otypes=[float])

    m_res = m
    # если норма станет меньше относительной точности, то выйти из цикла
    while delta > e:
        # функция для нахождения нормы
        def f1(x):
            return pow((abs(fun(x)-fun(Ej))), 2)

        def f2(x):
            return pow((abs(fun(x))), 2)
        # делим отрезок [a:b] на m частей
        xx = np.linspace(a, b, m_res)
        # print('xx',xx)


        fx = y(xx)


        xjminus1 = xx[0]
        sum1 = 0
        #   находим
        sum2, err = integrate.quad(f2, a, b)
        for x in xx[1:]:
            # вычисляем середины соседних отрезков
            Ej = (xjminus1 + x)/2
            # v1 для вычисления нормы
            v1, err = integrate.quad(f1, xjminus1, x)
            sum1 += v1
            xjminus1 = x
        # вычисляем величину дельта, для условия выхода из цикла

        delta1 = pow(sum1, 1/2)
        delta2 = pow(sum2, 1/2)
        delta = delta1/delta2
        m_res = m_res+1


    return m_res-1,xx,fx

class elem_spectr:
    def __init__(self, λ, α, Ψ, A, B):
        """Constructor"""
        self.λ = λ
        self.α = α
        self.Ψ = Ψ
```

```
        self.A = A
        self.B = B

# метод стрельбы
def shoot(fun, eps_shoot, a, b, m, Δ, step, xx, fx):
    print("метод стрельбы")
    fig, ax = plt.subplots()
    ax.grid()
    x = np.linspace(a, b, 1000)
    y = np.vectorize(fun, otypes=[float])
    plt.plot(x, y(x), linestyle='dashed')
    # plt.show()
    plt.xlabel("x")
    plt.ylabel("y")
    # построение кусочной функции
    if step:
        plt.step(xx, fx)
    plt.xlabel("x")
    plt.ylabel("y")
    m = len(xx)
    print("число разбиений отрезка M = ", m)

    # начальные данные
    x = xx
    u = fx
    spectr = []
    u0 = min(u)
    u = [abs(t) for t in u]
    λ_graph = []
    a = []
    α = []
    print("минимум функции u0 =", u0)
    # начать с λ, равным минимальному значению функции u(x)
    λ = u0 + 0.0001
    # Запустить бесконечный цикл для итеративного уточнения значений λ
    while True:

        array_Λ = []
        array_A = []
        array_B = []
        array_α = []

        for j in range(1, m):
            Λ = u[j] + λ
            sqrt_Λ = np.sqrt(abs(Λ))
            j_1 = j - 1
            A, B, X = symbols('A, B, X')
```

```
            if Λ > 0:
                funΨ = A * sym.cos(sqrt_Λ * X) + B * sym.sin(sqrt_Λ * X)
                def Ψ(x):
                    return A * sym.cos(sqrt_Λ * x) + B * sym.sin(sqrt_Λ * x)
            if Λ < 0:
                funΨ = A * sym.cosh(sqrt_Λ * X) + B * sym.sinh(sqrt_Λ * X)
                def Ψ(x):
                    return A * sym.cosh(sqrt_Λ * x) + B * sym.sinh(sqrt_Λ * x)

            # Начальные коэффициенты

            if j == 1:
                A, B, X = symbols('A, B, X')
                funΨ = funΨ - 1
                funΨdx = sym.diff(funΨ, X)
                funΨdx = funΨdx + Q0

                funΨ = funΨ.subs(X, 0)

                funΨdx = funΨdx.subs(X, 0)

                A, B = list(sym.solve([funΨ,  funΨdx], A, B).values())
                array_A.append(A)
                array_Λ.append(Λ)

                array_B.append(B)
        # Сшивание
        # На последующих итерациях решать систему линейных алгебраических уравнений
(СЛАУ) для нахождения этих констант,
            else:
                sqrt_Λ_minus1 = np.sqrt(abs(array_Λ[-1]))
                A, B = symbols('A, B')
                if array_Λ[-1] > 0 and Λ > 0:
                    A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A * sym.cos(sqrt_Λ *
x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]), -array_A[-1] * sqrt_Λ_minus1 *
sym.sin(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 * sym.cos(sqrt_Λ_minus1
* x[j_1]) + A * sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B * sqrt_Λ * sym.cos(sqrt_Λ *
x[j_1])], A, B).values())
                elif array_Λ[-1] > 0 and Λ < 0:
                    A, B = list(sym.solve([array_A[-1] * sym.cos(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sin(sqrt_Λ_minus1 * x[j_1]) - A * sym.cosh(sqrt_Λ *
x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]), -array_A[-1] * sqrt_Λ_minus1 *
sym.sin(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 * sym.cos(sqrt_Λ_minus1
* x[j_1]) + A * sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) - B * sqrt_Λ * sym.cosh(sqrt_Λ *
x[j_1])], A, B).values())
                elif array_Λ[-1] < 0 and Λ > 0:
```

```
                    A, B = list(sym.solve([array_A[-1] * sym.cosh(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 * x[j_1]) - A * sym.cos(sqrt_Λ *
x[j_1]) - B * sym.sin(sqrt_Λ * x[j_1]), -array_A[-1] * sqrt_Λ_minus1 *
sym.sinh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sin(sqrt_Λ * x[j_1]) - B * sqrt_Λ
* sym.cos(sqrt_Λ * x[j_1])], A, B).values())
                elif array_Λ[-1] < 0 and Λ < 0:
                    A, B = list(sym.solve([array_A[-1] * sym.cosh(sqrt_Λ_minus1 *
x[j_1]) + array_B[-1] * sym.sinh(sqrt_Λ_minus1 * x[j_1]) - A * sym.cosh(sqrt_Λ *
x[j_1]) - B * sym.sinh(sqrt_Λ * x[j_1]), -array_A[-1] * sqrt_Λ_minus1 *
sym.sinh(sqrt_Λ_minus1 * x[j_1]) + array_B[-1] * sqrt_Λ_minus1 *
sym.cosh(sqrt_Λ_minus1 * x[j_1]) + A * sqrt_Λ * sym.sinh(sqrt_Λ * x[j_1]) - B *
sqrt_Λ * sym.cosh(sqrt_Λ * x[j_1])], A, B).values())

                array_Λ.append(Λ)
                array_A.append(A)
                array_B.append(B)
                # вычислить интеграл от квадрата Ψ(x) на каждом интервале для оценки
нормированного коэффициента α(λ):
                def Ψ_2(x):
                    return Ψ(x)**2
                integr, err = integrate.quad(Ψ_2, x[j_1], x[j])
                array_α.append(integr)

        x_symbol = sym.Symbol('x')
        Ψdx = sym.diff(Ψ(x_symbol))
        # по завершении всех итераций цикла от 1 до M необходимо рассчитать a:
        a_ = float(Ψdx.subs(x_symbol, π) + Qπ * Ψ(π))
        a.append(a_)
        α_ = sum(array_α)
        α.append(α_)
        λ_graph.append(λ)
        # если параметры a на соседних интервалах, ai-1 и ai, имеют разные знаки, то
это указывает на наличие собственного значения λ в интервале между λ{i-1} и λi.
        if len(a) >= 2 and a[-2] != 0 and np.sign(a[-1]) != np.sign(a[-2]):
          # линейная интерполяция для нахождения собственного значения λ
            λ_ = fsolve(lambda x: [(a[-2] - a[-1]) * x[0] + (λ_graph[-2] - λ_graph[-
1]) * x[1] + (λ_graph[-2] * a[-1] - λ_graph[- 1] * a[-2]), x[1]], [1, 1])[0]
            round_N = 3
            λ_ = round(λ_, round_N)
            α_ = round(α_, round_N)
            spectr.append(elem_spectr(λ_, α_, funΨ, A, B ))
            print( (len(spectr)), ") ", "λ=", λ_, "α=", α_)
        # Проверка точности
        # точность привязана к количеству собственных значений
        if len(spectr) > 0:
            diff = abs( len(spectr)+1 - math.sqrt(abs(λ_graph[-1]))) /
math.sqrt((abs(λ_graph[-1])))
```

```
            if diff <= eps_shoot:
                break
            if len(spectr) >= EPSN:
                break
        λ += Δ
    return spectr, m, λ_graph, α, a, u0

# обратная задача
def reverse(fun_reverse, count_iter,  arr_ε, arr_β, len_λ, xx, fx):
    def U(_x):

      _sum = 0
      for m in range (1,M):
        _rangeX = [arrayX[m-1], arrayX[m]]
        _left = _rangeX[0]
        _right = _rangeX[1]

        def _Xm(_x):
          if (_left <= _x <= _right):
            return 1
          else:
            return 0
        _sum += arrayU[m] * _Xm(_x)
      return _sum


    print('Обратная задача')
    arrayX = np.linspace(a_left, b_right, M)
    arrayU = createFirstU(fun_reverse,M)

    # получение спектра для подобранной функции
    cache_file = 'cache/shoot1.joblib'

    # if load_result(cache_file) is not None:
    #   _spectr, _, _, _, _, _ = load_result(cache_file)
    # else:
    _spectr, _, _, _, _, _ = shoot(fun_reverse, eps_shoot, a_left, b_right, M, Δ,
True, xx, fx)
    #   save_result([_spectr, _, _, _, _, _],cache_file)



    fig, ax = plt.subplots()
    ax.grid()
    arr_Δ = []
    Gm = []
    arr_λ = []
    arr_α = []
```

```
arr_a = []
  # Собственные значения λ > 0 не учитываем по условию
  filtered_spectr = [s for s in _spectr if s.λ <= 0]
  # инициализация значений
  for i in range(len(filtered_spectr)):
      _Ψ = filtered_spectr[i].Ψ
      _A = filtered_spectr[i].A
      _B = filtered_spectr[i].B
      A, B, X, I = symbols('A, B, X, I')
      _Ψ = _Ψ.subs(A, _A)
      _Ψ = _Ψ.subs(B, _B)
      _Ψ = sym.sin(I * X)
      _Ψ = _Ψ.subs(I, i+1)

      _Ψ = sym.lambdify(X, _Ψ)

      filtered_spectr[i].Ψ = _Ψ

      arr_λ.append(filtered_spectr[i].λ)
      # arr_α.append(np.pi/2)
      arr_α.append(filtered_spectr[i].α)
  print("-------------------------")
  print("ИСКОМАЯ ФУНКЦИЯ")
  print("собственные числа. ε =", arr_ε)
  print("нормировочные коэффициенты. β =", arr_β)
  print("-------------------------")
  print("НАЧАЛЬНОЕ ПРИБЛИЖЕНИЕ")
  print("собственные числа. λ =", arr_λ)
  print("нормировочные коэффициенты. α =", arr_α)
  print("-------------------------")

  delta_Δ = 100
  ccc = 0
  while delta_Δ > e_spec and (ccc < count_iter):
    ccc += 1
    ax.grid()
    plt.step(arrayX, arrayU, label=(ccc))
    ax.grid()
    ax.legend(loc="lower left")
    plt.xlabel("x")
    plt.ylabel("y")

      #  вычисления градиента невязки

    _g = []
    for m in range(1, M):
      _sum = 0
      _sum_λ_e = 0
```

```
    _rangeX = [arrayX[m-1], arrayX[m]]

      _left = _rangeX[0]
      _right = _rangeX[1]
      # характеристическая функция
      def _Xm(_x):
        if (_left  <= _x <= _right):
          return 1
        else:
          return 0

      for k in range(len(filtered_spectr)):
        _Ψ = filtered_spectr[k].Ψ
        _α = arr_α[k]
        _λ = arr_λ[k]
        _ε = arr_ε[k]
        # вычисление скалярного произведения векторов
        def _fun_integrate(_x):
          return _Ψ(_x) * (_Xm(_x)*_Ψ(_x))
        _integrate, _ = integrate.quad(_fun_integrate, a_left, b_right,limit=50)
        _sum += (1/ _α) * (_λ - _ε) * _integrate
        # _sum +=  (_λ - _ε) * _integrate
        # _sum_λ_e += (_λ - _ε)**2
      # _g.append(-1*(_sum/np.sqrt(_sum_λ_e)))
      _g.append(_sum)
    Gm = _g
    # Функция для вычисления приращения значений функции
    def δU(_x):
      _sum = 0
      for m in range(1, M):
        _rangeX = [arrayX[m-1], arrayX[m]]
        _left = _rangeX[0]
        _right = _rangeX[1]
        def _Xm(_x):
          if (_left  <= _x <= _right):
            return 1
          else:
            return 0


        _sum += Gm[m-1] * _Xm(_x)
      return -revers_e *_sum
    # корректировка подобранной функции
    arr_δU = []
    for i in range(0, len(arrayU)):
      arrayU[i] = arrayU[i] + δU(arrayX[i])
      arr_δU.append(δU(arrayX[i]))
```

```
    _sum_λ = 0
    _sum_α = 0
    for i in range(len(filtered_spectr)):
      _Ψ = filtered_spectr[i].Ψ
      _α = arr_α[i]
      _λ = arr_λ[i]
      def _funIntegr_δλ(_x):
        return _Ψ(_x) * (δU(_x)*_Ψ(_x))
      _integr, _ = integrate.quad(_funIntegr_δλ, a_left, b_right, )
      _δλ = (1/_α)*(_integr)
      # _δλ = (_integr)
      # print('_δλ',_δλ)
      _λ = _λ + _δλ
      arr_λ[i] = _λ
      _sumA = 0


     for k in range(len(filtered_spectr)):
        if (i != k):
          _Ψk = filtered_spectr[k].Ψ
          _αk = arr_α[k]
          _λk = arr_λ[k]
          def _funIntegr_aik(_x):
            return _Ψk(_x) * (δU(_x)*_Ψ(_x))
          aik, err = integrate.quad(_funIntegr_aik, a_left, b_right, )
          aik = aik/(_αk*(_λ-_λk))
          _sumA += _αk * abs(aik)**2
      # обновление собственных значений и нормировочных коэффициентов
      _α = _α + _sumA
      arr_α[i] = _α

      _ε = arr_ε[i]
      _β = arr_β[i]
      _sum_λ += (_λ - _ε)**2
      _sum_α += (_α - _β)**2
    print('_λ на ', ccc,': ', [round(i,4) for i in arr_λ])
    print('_ε на ', ccc,': ', [round(i,4) for i in arr_ε])
    print('_α на ', ccc,': ', [round(i,4) for i in arr_α])

    delta_Δ = _sum_λ
    round_N = 3
    delta_Δ = round(delta_Δ, round_N)
    print('delta_Δ',delta_Δ, 'шаг', ccc)
    arr_Δ.append(delta_Δ)
    if len(arr_Δ)>1:
      if arr_Δ[-2]-arr_Δ[-1] < 0.1:
        break
  print("Δ итоговая невязка", arr_Δ)
```

```
    return arrayX, arrayU

if __name__ == "__main__":
  warnings.simplefilter("ignore")
  #  начальное M
  M = 3
  # шаг изменения собственного значения в методе стрельбы
  Δ = 0.7
  delta = 1
  x = np.arange(a_left,b_right,0.005,dtype=float)

  def createFirstU(fun, m = M):
    arrX = np.linspace(a_left, b_right, m)
    U = []
    for x in arrX:
        U.append(fun(x))
    # U =  [0] + U[1:-1] + [0]
    return U



# Функция для удобства выбора начальных значений, здесь же и меняются начальные точки
, заданные собственные значения
  def get_inits():
    choose_initial_params = int(input('Введите номер вида спектра: '))
    # choose_initial_params = 1

    if choose_initial_params == 1:

      EPS = int(input('Выберите кол-во собственных значений: '))
      lam = []
      a = 1

      def fun_near (x):
        return (a*x*(x-np.pi))*(np.sin(i*x))**2
      while len(lam) <= EPS and (len(lam) < EPS or lam[-1] > -5):

        for i in range(1, EPS + 1):  # Include EPS in the range
            if len(lam) < EPS:
                integral_value = integrate.quad(fun_near, 0, np.pi)[0]
                new_value = i**2 + (2 / np.pi) * integral_value
                if new_value < 0:
                    lam.append(new_value)
                else:
                    lam = []
                    a += 1
                    break
            elif len(lam) == EPS and lam[-1] > -5:
```

```
            lam = []
            a += 1
            break

     # заданные собственные значения из задания ВКР (I)

    def init_s(n):
      λ_arr_I = []
      α_arr = []
      for i in range(0,n):
        λ_arr_I.append(round(-5/np.sqrt(i+1),3))
        α_arr.append(round(π/2,3))
        # λ_arr_I.append(lam[i])
        # α_arr.append(round(π/2,3))


      return λ_arr_I,α_arr

    def fun(x):
      return (a+1)*x*(x-np.pi)
    print('Коэффициент a: ',a+1)

    ea = float(input('Выберите точность аппроксимации в % (от этого зависит кол-во
M): '))/100
    grad_e = float(input('Скорость градиентного спуска Learning Rate: '))
    e_spec = float(input('Точность невязки : '))
    count_iter = 200

  elif choose_initial_params == 2:
    # заданные собственные значения из задания ВКР (II)
    print(' -b1 - b2 * n ( по умолчанию b1 = 0.3 b2 = 0.4)')
    b1 = float(input('Задайте коэффициенты b1: '))
    b2 = float(input('Задайте коэффициенты b2: '))
    def init_s(n,b1=b1,b2=b2):
      λ_arr_II = []
      α_arr = []
      for i in range(0,n):
        λ_arr_II.append(round(-b1-b2*i,3) )
        α_arr.append(round(π/2,3))


      return λ_arr_II,α_arr
    # Подобранная функция
    EPS = int(input('Выберите кол-во собственных значений: '))

    lam = []
    a = 1
```

```
      def fun_near (x):
        return (a*x*(x-np.pi))*(np.sin(i*x))**2

      while len(lam) <= EPS and (len(lam) < EPS or lam[-1] > -5):
        for i in range(-EPS - 1, -1, 1):
            if len(lam) < EPS:
                integral_value = integrate.quad(fun_near, 0, np.pi)[0]
                new_value = i**2 + (2 / np.pi) * integral_value
                if new_value < 0:
                    lam.append(new_value)
                else:
                    lam = []
                    a += 1
                    break
            elif len(lam) == EPS and lam[-1] > -5:
              lam = []
              a += 1
              break
      print('Коэффициент a: ', a)
      def fun(x):

        return (a)*x*(x-np.pi)
      ea = float(input('Выберите точность аппроксимации в % (от этого зависит кол-во
M): '))/100
      grad_e = float(input('Скорость градиентного спуска Learning Rate: '))
      e_spec = float(input('Точность невязки : '))

      count_iter = 400
    return  fun, init_s, EPS, ea, grad_e,count_iter, e_spec




# получение заданного спектра и др переменных  начальных условий
  func, init_spectr,  EPSN, e, revers_e,count_asd, e_spec = get_inits()
  spectr = []
  M, xx, fx = aprox(delta,e,func,a_left, b_right,M)
  arr_ε , arr_β = init_spectr(EPSN)


  len_λ =  len(arr_ε)

#обратная задача

  arrayX, arrayU = reverse(func, count_asd, arr_ε, arr_β, len_λ, xx, fx)
# построение графиков
  fig, ax_last = plt.subplots()
```

```
 ax_last.grid()
  # подобранная функция
  plotX = np.linspace(a_left, b_right, M)
  plotY = createFirstU(func,M)
  plt.plot(plotX, plotY, label="начальное приближение", linestyle='dashed')
# приближение к начальной функции
# гладкая аппроксимация
  def smooth_approximation(arrayX, arrayY):
    # Проверка, что входные массивы имеют одинаковую длину
    if len(arrayX) != len(arrayY):
        raise ValueError("Array lengths must be equal")

    # Сортировка точек по x для корректной работы сплайн-интерполяции
    sorted_indices = np.argsort(arrayX)
    arrayX_sorted = np.array(arrayX)[sorted_indices]
    arrayY_sorted = np.array(arrayY)[sorted_indices]

    # Создание кубического сплайна
    spline = CubicSpline(arrayX_sorted, arrayY_sorted)

    return spline
  newFun =  smooth_approximation(arrayX, arrayU)
  newFunX = np.linspace(a_left, b_right, num=1000, endpoint=True)
  plt.plot(newFunX, newFun(newFunX), label="приближение функции", linestyle='dotted')
  ax_last.legend()
  plt.xlabel("x")
  plt.ylabel("y")
# конец
  plt.show()
  plt.close()
```

Приложение Б

Теория пленарных волноводов

Планарный волновод представляет собой многослойную структуру, в которой каждый слой имеет свой уникальный показатель преломления. На изображении 1 центральный слой, отмеченный голубым цветом, расположен между подложкой и покрывающим слоем. Центральный слой волновода характеризуется показателем преломления n_1 и толщиной h. Подложка с показателем преломления n_2 находится под волноводным слоем и простирается вниз. Покрывающий слой с показателем преломления n_3 размещен над волноводным слоем и простирается вверх.

Существуют два типа планарных волноводов: пленочные и градиентные. Пленочные волноводы включают однородную пленку постоянной толщины, нанесенную на подложку с более низким показателем преломления. В градиентных волноводах показатель преломления волноводного слоя изменяется вдоль координаты x, увеличиваясь по мере приближения к поверхности волновода.

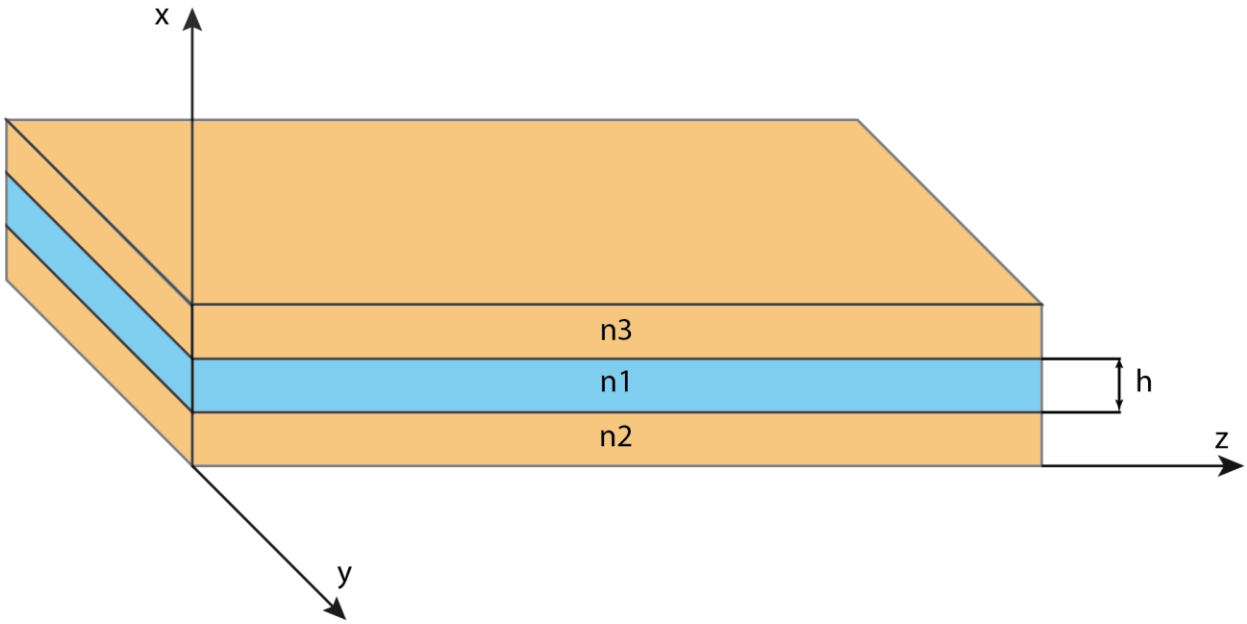

Рисунок 1 – Трехслойный планарный волновод

В рамках электромагнитной теории, электрические и магнитные явления объясняются с помощью четырех фундаментальных уравнений Максвелла:

$$div\,\vec{D} = \rho\,, \tag{1}$$

$$div\vec{B} = 0, \tag{2}$$

$$rot\vec{H} = \vec{j} + \frac{\partial\vec{D}}{\partial t}, \tag{3}$$

$$rot\vec{H} = -\frac{\partial\vec{B}}{\partial t}, \tag{4}$$

где $\vec{D}$ − вектор электрической индукции;

$\vec{B}$ - вектор магнитной индукции;

$\vec{H}$ — напряженность магнитного поля;

$\vec{\jmath}$ — плотность тока;

ρ — объемная плотность заряда.

Добавим к этим уравнениям материальные соотношения:

$$\vec{D} = \epsilon_0\vec{E}\,,\ \vec{B} = \mu_0\vec{H}, \vec{j} = \sigma\vec{E} \tag{5}$$

При распространении электромагнитной волны в среде с однородными свойствами, где $\sigma = 0, \rho = 0, j = 0$, и с постоянными ϵ и μ, уравнения могут быть записаны так:

$$div\vec{E} = 0 \tag{6}$$

$$div\vec{H} = 0 \tag{7}$$

$$rot\vec{H} = \epsilon_0\frac{\partial\vec{E}}{\partial t} \tag{8}$$

$$rot\vec{E} = -\mu_0 \frac{\partial \vec{H}}{\partial t} \tag{9}$$

Применяя операцию ротора к обеим частям уравнения, получаем:

$$rot\,\vec{E} = -\mu_0 \frac{\partial\left(rot\vec{H}\right)}{\partial t} = -\epsilon_0\mu_0 \frac{\partial^2 \vec{E}}{\partial t^2} \tag{10}$$

Так как $rot\,\vec{E} = \nabla \times (\nabla \times \vec{E}) = \nabla\left(\nabla \cdot \vec{E}\right) - \nabla^2\vec{E} = -\nabla^2\vec{E}$, то получаем волновое уравнение для напряженности электрического поля:

$$\nabla^2\vec{E} - \epsilon_0\mu_0 \frac{\partial^2 \vec{E}}{\partial t^2} = 0 \tag{11}$$

Аналогично можно получить волновое уравнение для напряженности магнитного поля:

$$\nabla^2\vec{H} - \epsilon_0\mu_0 \frac{\partial^2 \vec{H}}{\partial t^2} = 0 \tag{12}$$

Проанализируем волновые уравнения, связанные с электрическими и магнитными полями. Начнем с анализа планарного волновода, который характеризуется показателем преломления n_1 и окружен средами с показателями преломления n_2 и n_3. Волновод расположен так, что световая волна движется вдоль оси z, а ось x располагается перпендикулярно к плоскости волновода. Анализируя решение волнового уравнения, видим, что

речь идет о плоской монохроматической волне, в которой присутствуют напряженности электрического и магнитного поля.

$$\vec{E} = \overrightarrow{E_0}(x, y)\exp\big(i(\omega t - \beta z)\big), \tag{13}$$

$$\vec{H} = \overrightarrow{H_0}(x, y)\exp\big(i(\omega t - \beta z)\big), \tag{14}$$

где ω — угловая частота;

β — постоянная распространения волны вдоль направления Z, которая зависит от граничных условий, накладываемых на $\overrightarrow{E_0}$(x, y).

Для волн, перпендикулярных плоскости падения:

$$\beta E_y = \mu_0 H_x, \tag{15}$$

$$-i\beta E_x - \frac{\partial H_z}{\partial x} = i\epsilon_0 \omega E_y, \tag{16}$$

$$\frac{\partial E_y}{\partial x} = -i\mu_0 \omega H_z. \tag{17}$$

Выразим H_x из уравнения (15) и H_z из уравнения (17) и подставим в уравнение (16). Получим волновое уравнение для E_y в виде:

$$\frac{\partial^2 E_y}{\partial x^2} + (\omega^2 \epsilon_0 \mu_0 - \beta^2) E_y = 0 \tag{18}$$

Решением этого уравнения будут монохроматические плоские волны, распространяющиеся вдоль оси x:

$$E_y = E_{y0}\exp(ik_x x), \tag{19}$$

где k_x – волновой вектор распространения волны вдоль оси x;

n – показатель преломления.

Если заменить $k^2n^2 - \beta^2 = \omega^2\epsilon_0\mu_0 - \beta^2$, то получим уравнение Штурма—Лиувилля:

$$\frac{\partial^2 E_y}{\partial x^2} + (k^2n^2 - \beta^2)E_y = 0 \quad (20)$$

Таким образом, уравнения Максвелла могут быть сведены к уравнению Штурма—Лиувилля.