**Содержание**

[Введение……………………………………………………………………………. 7](#_Toc169179542)

[1 Аналитический обзор предметной области……………………………………. 8](#_Toc169179543)

[1.1 Классификация интегральных уравнений 8](#_Toc169179544)

[1.2 Уравнение Фредгольма 2-рода 9](#_Toc169179545)

[1.3 Решение уравнения Фредгольма 2-го рода в общем виде 9](#_Toc169179546)

[1.4 Уравнение Вольтерра 2-го рода 10](#_Toc169179547)

[1.5 Решения уравнения Вольтерра 2-го рода в общем виде 11](#_Toc169179548)

[1.6 Формула Симпсона 12](#_Toc169179549)

[1.7 Постановка задачи 13](#_Toc169179550)

[1.8 Выводы по главе 14](#_Toc169179551)

[2 Алгоритмическое конструирование…………………………………………… 15](#_Toc169179552)

[2.1 Решение уравнения Фредгольма 2-го рода 15](#_Toc169179553)

[2.2 Решение уравнения Вольтерра 2-го рода 17](#_Toc169179554)

[2.3 Выводы по главе 19](#_Toc169179555)

[3 Программное конструирование………………………………………………... 20](#_Toc169179556)

[3.1 Обоснование выбора средств разработки 20](#_Toc169179557)

[3.2 Описание использованных библиотек 21](#_Toc169179558)

[3.3 Описание основных функций 22](#_Toc169179559)

[3.3 Выводы по главе 25](#_Toc169179560)

[4 Тестирование программного средства………………………………………… 26](#_Toc169179561)

[4.1 Описание процесса тестирования 26](#_Toc169179562)

[4.2 Тестирование ПО на конкретном примере 30](#_Toc169179563)

[4.3 Выводы по главе 36](#_Toc169179564)

[Заключение………………………………………………………………………... 37](#_Toc169179565)

[Перечень использованных информационных ресурсов………………………... 39](#_Toc169179566)

[Приложение А – Техническое задание………………………………………….. 40](#_Toc169179567)

[Приложение Б – Исходный код программного средства………………………. 45](#_Toc169179568)

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

ПП.350000.000

Разраб.

Евсюкова С.

Провер.

Никитина А.В.

.

Разработка программного средства для решения интегральных уравнений Фредгольма и Вольтерра методом конечных сумм с помощью формулы Симпсона

Лист.

Листов

53

ДГТУ

Кафедра «ПОВТиАС»

**Введение**

Интегральные уравнения Фредгольма и Вольтерра представляют собой важный класс задач математической физики, теории управления, биологии и других областей науки. Метод конечных сумм является одним из наиболее эффективных и широко используемых подходов.

Для нахождения решения данных уравнений в этой работе используется квадратурная формула Симпсона - один из методов численного интегрирования, позволяющий численно приближенно находить значение определенного интеграла. Применение этой формулы к решению интегральных уравнений Фредгольма и Вольтерра позволяет получить точные и быстрые результаты.

В данной работе рассмотрено применение метода конечных сумм с использованием формулы Симпсона для численного решения интегральных уравнений Фредгольма и Вольтерра. Рассмотрены теоретические основы метода, а также представлены результаты численных экспериментов на различных примерах.

1 Аналитический обзор предметной области

В данном разделе рассматриваются теоретической разбор предметной области и постановка задачи.

1.1 Классификация интегральных уравнений

Интегра́льное уравне́ние — функциональное уравнение, содержащее интегральное преобразование над неизвестной функцией. Если интегральное уравнение содержит также производные от неизвестной функции, то говорят об интегро-дифференциальном уравнении [1]. Такие уравнения подразделюсь на линейные и нелинейные.

Линейные интегральные уравнения – это интегральные уравнения, в которые неизвестная функция входит линейно:

,

где – искомая функция,

, – известные функции,

– параметр.

Функция называется ядром интегрального уравнения. В зависимости от вида ядра и свободного члена линейные уравнения можно разделить ещё на несколько видов [1].

Для нелинейных интегральных уравнений можно придумать немыслимое многообразие нелинейных уравнений, поэтому дать им полную классификацию не представляется возможным. Однако такими уравнениями, имеющие большое теоретическое и прикладное значение, считаются: уравнения Урысона, уравнение Ляпунова – Лихтенштейна, нелинейное уравнение Вольтерра.

1.2 Уравнение Фредгольма 2-рода

Уравнения Фредгольма 2-го рода — это уравнения вида:

,

где – искомая функция,

, – известные функции,

– параметр.

Пределы интегрирования могут быть как конечными, так и бесконечными. Однако переменные должны удовлетворять неравенство: , , а ядро и свободный член должны быть непрерывными: , , либо удовлетворять условиям:

,

,

где , – известные функции.

Ядра, удовлетворяющие последнему условию, называют «фредгольмовыми». Если на , то уравнение называется однородным, иначе оно называется неоднородным интегральным уравнением.

1.3 Решение уравнения Фредгольма 2-го рода в общем виде

Рассмотрим интегральное уравнение Фредгольма второго рода

, (1)

где – неизвестная функция,

, – заданные функции,

– параметр.

Требуется найти решение интегрального уравнения 1 при . Необходимо разбить отрезок на равных частей.

,

где .

Рассмотрим интегральное уравнение 1 в точках деления:

, (2)

где .

Метод конечных сумм заключается в замене определённого интеграла конечной суммой с помощью одной из квадратурных формул:

где .

Разобьём отрезок на равных частей. Получаем:

.

Вводим обозначения:

.

Заменяя определённый интеграл, входящий в равенство 2, конечной суммой с помощью одной из квадратурных формул, получаем:

где .

После вычисления можно записать приближённое выражение:

1.4 Уравнение Вольтерра 2-го рода

Уравнения Вольтерра отличаются от уравнений Фредгольма тем, что один из пределов интегрирования в них является переменным:

,

где .

1.5 Решения уравнения Вольтерра 2-го рода в общем виде

Рассмотрим интегральное уравнение Вольтерра второго рода:

, (5)

где – неизвестная функция,

, – заданные функции,

– параметр,

.

Требуется найти решение интегрального уравнения 5 при . Сведём интегральное уравнение Вольтерра к интегральному уравнению Фредгольма. Для этого вводим

После этого уравнение 4 запишем в виде

. (6)

Для нахождения приближённого решения интегрального уравнения 6 необходимо решить систему линейных алгебраических уравнений

где ,

,

,

,

.

Из равенства 6 получаем

После чего система 6 преобразуется к виду

где .

Здесь , , , – приближение к искомой функции . Решение системы уравнений 3 даёт приближенные значения искомой функции в узлах Приведём уравнение 3 к следующему виду:

подробнее,

Хорошо видно, что матрица коэффициентов этой системы – треугольная.

Первое уравнение содержит одно неизвестное значение , которое сразу же можно найти, второе уравнение содержит только и , поэтому можно найти , и т.д. Таким образом, вычисляем последовательно все значения , которые можно подставить формулу 4 для нахождения решения уравнения.

1.6 Формула Симпсона

Формула Симпсона (также Ньютона-Симпсона) относится к приёмам численного интегрирования. Получила название в честь британского математика Томаса Симпсона (1710—1761).

Суть метода заключается в приближении подынтегральной функции на отрезке интерполяционным многочленом второй степени , то есть приближение графика функции на отрезке параболой. Метод Симпсона имеет порядок погрешности 4 и алгебраический порядок точности 3 [2].

Формулой Симпсона называется интеграл от интерполяционного многочлена второй степени на отрезке :

,

где , и – значения функции в соответствующих точках (на концах отрезка и в его середине).

1.7 Постановка задачи

В результате аналитического обзора была поставлена цель: решить уравнений Фредгольма и Вольтерра 2-го рода аналитически и числено. Разработать программное средство, реализующее численное решение уравнений Фредгольма и Волтерра 2-го рода методом конечных сумм с помощью формулы Симпсона.

Для достижения поставленной цели необходимо решить следующие задачи:

* разработать алгоритмы, реализующие численное решение;
* выбрать среду и язык реализации;
* реализовать подпрограммы: аналитического решения, численного решения, построение графика результата, пользовательский интерфейс;
* провести тестирование ПС на нескольких контрольных примерах;
* в соответствии с отчетом по тестированию выполнить корректировку ПС;
* разработать сопроводительную документацию: техническое задание, отчет.

1.8 Выводы по главе

В данной главе проведён теоретический обзор интегральных уравнений Фредгольма и Вольтерра второго рода. Приведён общий вид решения данных уравнений с помощью метода конечных сумм и квадратурной формулой Симпсона. Данный обзор позволяет алгоритмически сконструировать программное средство для вычисления уравнений Фредгольма и Вольтера второго рода. Однако, при использовании метода конечных сумм и формулы Симпсона для решения уравнений Фредгольма и Вольтерра второго рода необходимо учитывать особенности интегральных ядер, а также обеспечить адекватную аппроксимацию при дискретизации задачи. Важно также оценивать точность полученного численного решения и проводить необходимые проверки на устойчивость метода.

Таким образом, метод конечных сумм и квадратурная формула Симпсона представляют собой полезные инструменты для численного решения уравнений Фредгольма и Вольтерра второго рода, позволяющие получать приближенные решения интегральных уравнений и проводить анализ их свойств с помощью дискретизации и интерполяции.

2 Алгоритмическое конструирование

В данном разделе рассматриваются конкретные решения уравнений Фредгольма и Вольтерра 2-го рода с получением окончательного результата.

2.1 Решение уравнения Фредгольма 2-го рода

Дано интегральное уравнение Фредгольма второго рода:

. (8)

В соответствии с рассмотренным алгоритмом решения уравнения Фредгольма второго рода в предыдущем разделе, решим данное уравнение.

Разобьём отрезок на две части. Получаем ;

; ; ;

; ; .

Рассмотрим уравнение 8 в точках деления :

, (9)

где

Определённый интеграл заменяем конечной суммой, используя формулу Симпсона при . После такой замены равенство 9 примет вид

,

где

Подставляем значения , и получаем систему из трёх уравнений с тремя неестественными , , .

С помощью метода Гаусса решим данную систему и получим следующие результаты:

Подставим результаты в формулу 4:

Получим следующий результат:

.

Данный алгоритм решения наглядно показан на рисунке 1.



Рисунок 1 – Схема алгоритма решения уравнения Фредгольма

2.2 Решение уравнения Вольтерра 2-го рода

Дано интегральное уравнение Вольтерра второго рода:

. (10)

В соответствии с рассмотренным алгоритмом решения уравнения Вольтерра второго рода в предыдущем разделе, решим данное уравнение.

Произведём замену:

Получим следующее уравнение:

. (11)

Разобьём отрезок на две части. Получаем ;

; ; ;

; ; .

Теперь необходимо рассмотреть уравнение 11 в точках деления . Однако стоит вести некоторое условие:

. (12)

Таким образом, будем ссылаться на данной условие при подстановке значений в уравнение 12.

Определённый интеграл заменяем конечной суммой, используя формулу Симпсона при . После такой замены равенство 12 примет вид

,

где

Подставляем значения , и получаем систему из трёх уравнений с тремя неестественными , , .

С помощью метода Гаусса решим данную систему и получим следующие результаты:

Подставим результаты в формулу 4:

Получим следующий результат:

Данный алгоритм решения наглядно показан на рисунке 2.

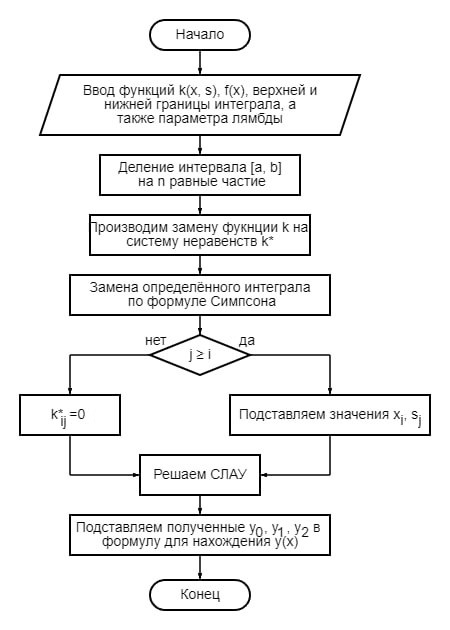


Рисунок 2 – Схема алгоритма решения уравнения Вольтерра

2.3 Выводы по главе

В данной главе рассмотрено частное решение интегральных уравнений Фредгольма и Вольтерра второго рода с помощью схем. Для решения таких уравнений мы использовали метод конечных сумм с помощью формулы Симпсона.

Основой метода является аппроксимация интеграла заданной функции с помощью суммы значений этой функции в заданных точках узлов. Причем для решения интегральных уравнений второго рода используется формула Симпсона, которая аппроксимирует интеграл как сумму значений функции в узлах, взвешенных коэффициентами.

Для реализации метода в алгоритмическом виде построены схемы, отражающие последовательность шагов вычислений. В процессе вычислений использовался итерационный метод, который позволяет приблизить решение с нужной точностью с конечным числом итераций.

Алгоритмическое конструирование с схемами позволяет легко представить и реализовать этот метод в виде программы для численного решения сложных математических задач.

3 Программное конструирование

В данном разделе обоснованы выбор языка программирования, используемого для реализации программы, а также представлены основания выбора среды программирования. Определены и описаны основные подпрограммы разрабатываемого приложения.

3.1 Обоснование выбора средств разработки

Для разработки программного средства взят за основу язык программирования Python.

Python – это язык программирования использующийся в самых разных задачах – от разработки игр и веб-приложений до обучения нейронных сетей. Он имеет простой и понятный синтаксис, который делает код более читаемым и понятным. Это особенно важно для математических вычислений, где ясность и точность кода имеют большое значение. Python обладает обширной экосистемой библиотек, предназначенных для научных вычислений и математического моделирования. Некоторые из наиболее популярных библиотек включают NumPy, matplotlib и pandas. Эти библиотеки предоставляют мощные инструменты для работы с числовыми данными, выполнения математических операций, решения уравнений и построения графиков [4].

В качестве среды разработки выбран Visual Studio Code, так как он позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом.

3.2 Описание использованных библиотек

В процессе программного конструирования выделены библиотеки необходимые для реализации поставленной задачи. Данные библиотеки описание в таблице 1.

Таблица 1 – Используемые библиотеки

|  |  |  |
| --- | --- | --- |
| Название | Описание | Назначение |
| Tkinter | Событийно-ориентированная графическая библиотека на основе средств Tk. Входит в стандартную библиотеку Python. | Используется для создания графического интерфейса пользователя. |
| Numpy | Библиотека с открытым исходным кодом для языка программирования Python. Предоставляет реализации вычислительных алгоритмов (в виде функций и операторов), оптимизированные для работы с многомерными массивами. | Используется для решения СЛАУ. |
| Matplotlib | Библиотека на языке программирования Python для визуализации данных двумерной и трёхмерной графикой. | Используется для вывода на экран графика конечной функции уравнений Фредгольма и Вольтерра. |
| SymPy | Предоставляет возможности компьютерной алгебры в виде отдельного приложения, как библиотека для других приложений или в Интернете как SymPy Live или SymPy Gamma. | Используется для нахождения конечной функции уравнений Фредгольма и Вольтерра. |
| Re | Предоставляет операции сопоставления регулярных выражений. | Используется для обработки строковых значений, содержащих в себе части или полные выражения уравнений Фредгольма и Вольтерра. |

3.3 Описание основных функций

Программное средство состоит из нескольких функций, которые обеспечивают его работоспособность. Такими функциями в данном программном средстве считаются:

*eq\_Fredgolm* – функция, отвечающая за полный расчёт уравнения Фредгольма второго рода. Входные и выходные данные данной функции описаны в таблицах 2 и 3 соответственно.

Таблица 2 – Входные данные функции *eq\_Fredgolm*

|  |  |
| --- | --- |
| Имя переменной | Тип данных |
| k | Заданная функция |
| f | Заданная функция |
| l | Значение параметра |
| a | Нижняя граница определённого интеграла |
| b | Верхняя граница определённого интеграла |

Таблица 3 – Выходные данные функции *eq\_Fredgolm*

|  |  |
| --- | --- |
| Имя переменной | Тип данных |
| main\_equation | Уравнение Фредгольма |
| diap\_parts | Значения интервала, поделённого на две части |
| slau | Система линейных алгебраических уравнений |
| matrix\_answers | Результаты решения СЛАУ |
| result | Конечное выражение |

*eq\_Volterr* –функция, отвечающая за полный расчёт уравнения Вольтерра второго рода. Входные и выходные данные данной функции описаны в таблицах 4 и 5 соответственно.

Таблица 4 – Входные данные функции *eq\_Volterr*

|  |  |
| --- | --- |
| Имя переменной | Тип данных |
| k | Заданная функция |
| f | Заданная функция |
| l | Значение параметра |
| a | Нижняя граница определённого интеграла |
| b | Верхняя граница определённого интеграла |

Таблица 5 – Выходные данные функции *eq\_Volterr*

|  |  |
| --- | --- |
| Имя переменной | Тип данных |
| main\_equation | Уравнение Вольтерра |
| diap\_parts | Значения интервала, поделённого на две части |
| slau | Система линейных алгебраических уравнений |
| matrix\_answers | Результаты решения СЛАУ |
| result | Конечное выражение |

*grafic\_create* –функция, отвечающая за создание графиков результатов уравнения Фредгольма и Вольтерра. Входные и выходные данные данной функции описаны в таблицах 6 и 7 соответственно.

Таблица 6 – Входные данные функции *grafic\_create*

|  |  |
| --- | --- |
| Имя переменной | Тип данных |
| y1 | Результат уравнения Фредгольма |
| y2 | Результат уравнения Вольтерра |
| y3 | Уравнение Фредгольма |
| y4 | Уравнение Вольтерра |
| l | Значение параметра |
| f | Заданная функция |
| a | Нижняя граница определённого интеграла |
| b | Верхняя граница определённого интеграла |

Таблица 7 – Выходные данные функции *grafic\_create*

|  |  |
| --- | --- |
| Имя переменной | Тип данных |
| rate\_res1 | Результат погрешности между уравнением Фредгольма и его результатом решения |
| rate\_res2 | Результат погрешности между уравнением Вольтерра и его результатом решения |

* 1. Выводы по главе

В данной главе разработана программа на языке программирования Python для численного решения уравнений Фредгольма и Вольтерра второго рода с использованием метода конечных сумм и квадратурной формулы Симпсона. Программа предоставляет пользователю возможность задать интегральное уравнение с необходимыми параметрами. Реализация метода конечных сумм позволяет дискретизировать интегральное уравнение, аппроксимировать его с помощью суммирования значений функции на дискретной сетке и численно интегрировать полученное выражение. Этот подход позволяет эффективно решать уравнения Фредгольма и Вольтерра второго рода, учитывая особенности интегральных ядер. Использование квадратурной формулы Симпсона в программе позволяет аппроксимировать интеграл на равномерной сетке с помощью квадратичной интерполяции. Этот метод обеспечивает более точные результаты и уменьшает вычислительную погрешность по сравнению с другими методами численного интегрирования. Важным аспектом программы является проверка правильности решения, оценка точности полученных результатов и устойчивость численных методов при различных входных данных. Это помогает предотвратить ошибки и обеспечить корректное выполнение численного решения уравнений.

Таким образом, разработанная программа на языке программирования Python для решения уравнений Фредгольма и Вольтерра второго рода с использованием метода конечных сумм и квадратурной формулы Симпсона представляет собой удобный инструмент для выполнения численных расчетов и анализа различных интегральных уравнений.

4 Тестирование программного средства

В данном разделе продемонстрированы основные этапы тестирования программного средства.

4.1 Описание процесса тестирования

При запуске программы на экране пользователя выводится окно интерфейса с полями для ввода входных данных для решения уравнений Фредгольма и Вольтерра, что показано на рисунке 3.

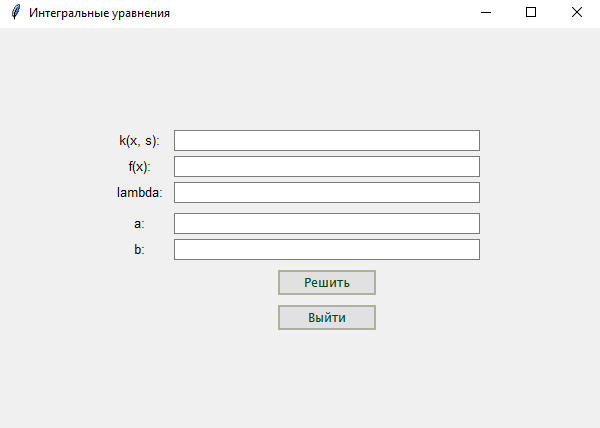


Рисунок 3 – Пользовательский интерфейс

После того как пользователь введёт свои данные в поля интерфейса необходимо будет нажать кнопку «Решить». Если же пользователь захочет выйти из программного средства, то необходимо будет нажать кнопку «Выход», что показано на рисунке 4.

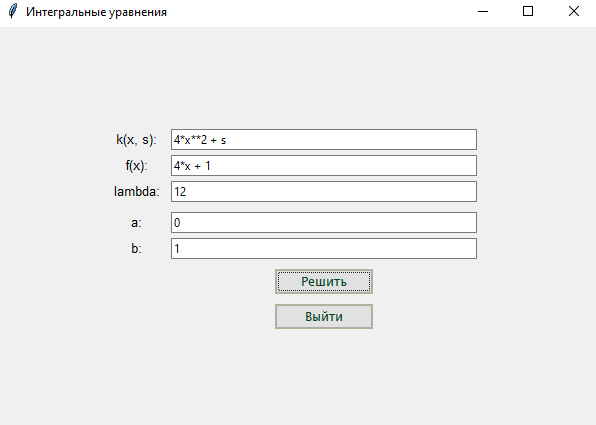


Рисунок 4 – Пользовательский интерфейс с введёнными в поля входными данными

После нажатия на кнопку «Решить» на экран пользователя будет выведено новое окно с графиками уравнений Фредгольма, Вольтерра и функций для нахождения погрешностей, что показано на рисунке 5.

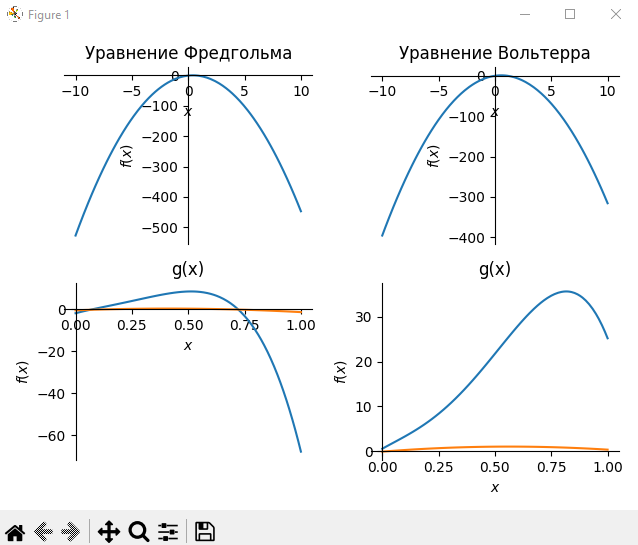


Рисунок 5 – Окно с графиками функция

После закрытия данного окна на экран пользователя будет выведено окно с решением уравнений Фредгольма и Вольтерра, а также погрешность для каждого уравнения, что показано на рисунке 6.

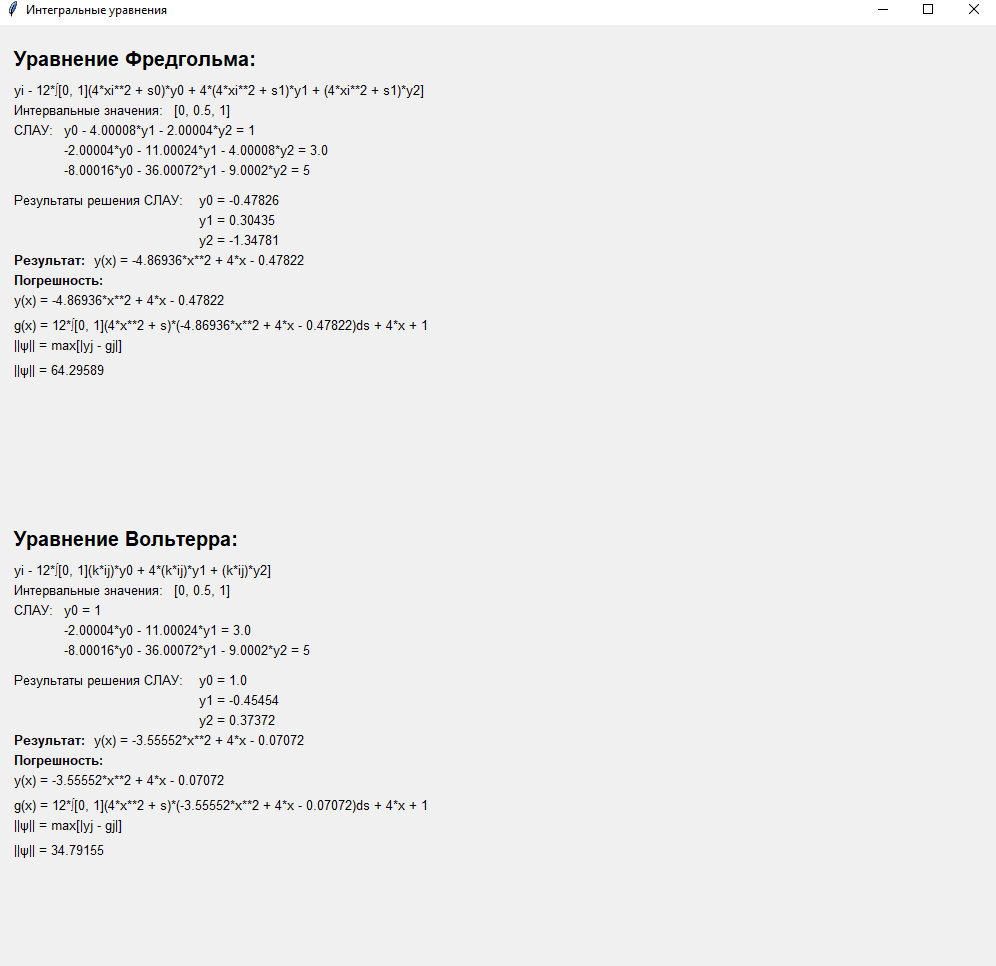


Рисунок 6 – Окно с результатами решения уравнения

В случае, если пользователь введён неверные входные данные или же оставит какие-то поля пустыми, то на экран будет выведено сообщение об ошибке, которое показано на рисунке 7.

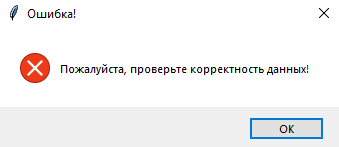


Рисунок 7 – Сообщение об ошибке при неверном вводе данных в пользовательский интерфейс

Если во время вычисления произойдёт непредвиденная ситуация, при которой невозможно продолжить нахождение решения уравнений, то на экран пользователя будет выведено сообщение об ошибке, что показано на рисунке 8.

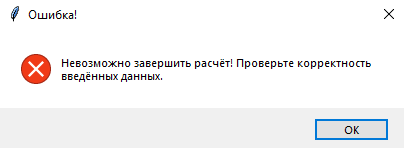


Рисунок 8 – Сообщение об ошибке при проблеме во время нахождения решения уравнения

После закрытия главного окна, либо нажатия на кнопку «Выход» программа полностью завершится. Все дополнительные окна будут закрыты.

4.2 Контрольные примеры

Рассмотрим пример, разобранный во втором разделе, чтобы убедиться в корректности работы программного средства.

Вводим в пользовательский интерфейс входные данные, соответствующие примеру из второго раздела, что показано на рисунке 9.

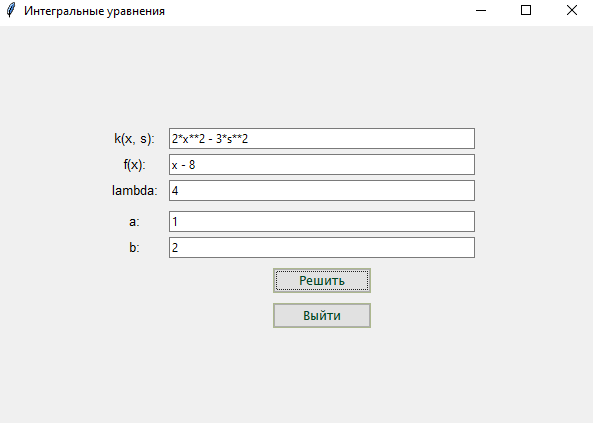


Рисунок 9 – Ввод данных примера 1

После нажатия на кнопку «Решить» на экран будет выведено окно с графиками решения данного примера, что показано на рисунке 10.

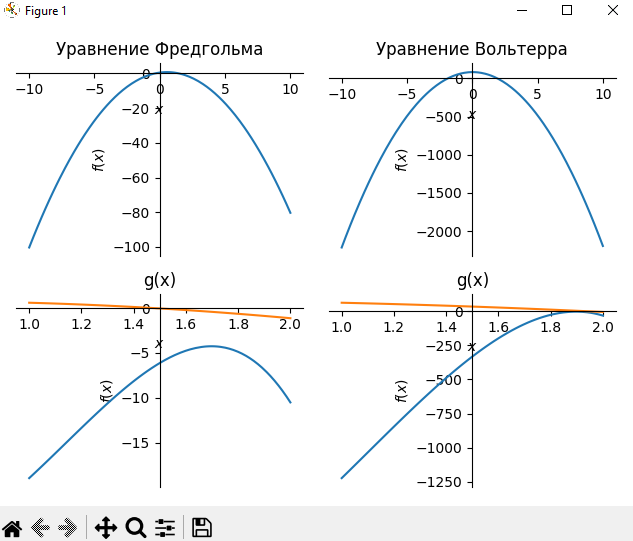


Рисунок 10 – Графики решения примера 1

Закрыв данное окно на экран будет выведено окно с подробным решением уравнений Фредгольма и Вольтерра, что показано на рисунке 11.

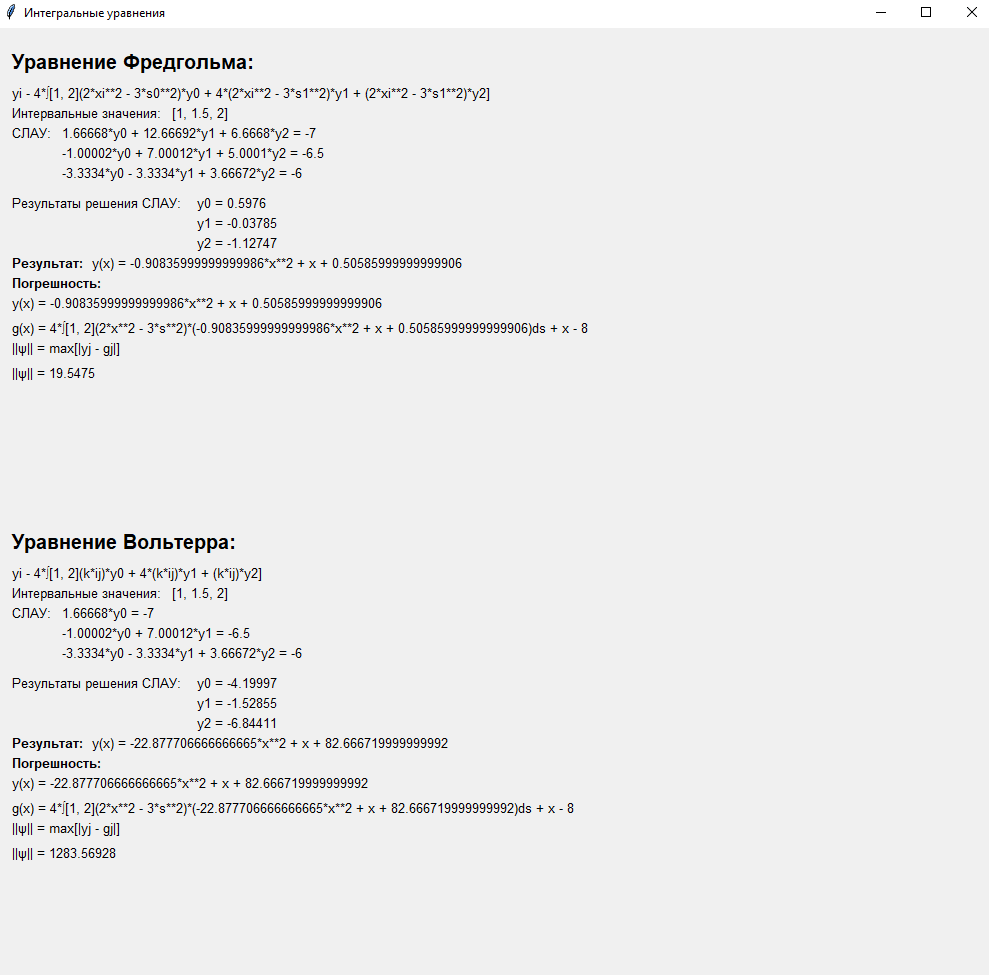


Рисунок 11 – Результат решения примера 1

В результате был получен следующий результат для уравнения Фредгольма:

.

Погрешность для данного уравнения:

.

Результат решения уравнения Вольтерра:

.

Погрешность для данного уравнения:

.

Сравнив результаты программы и результаты ручного счёта, можно сказать, что программа работает корректно.

Рассмотрим другой пример для уточнения корректности работы программы. Взяты следующие входные данные: , , , , , что показано на рисунке 12.

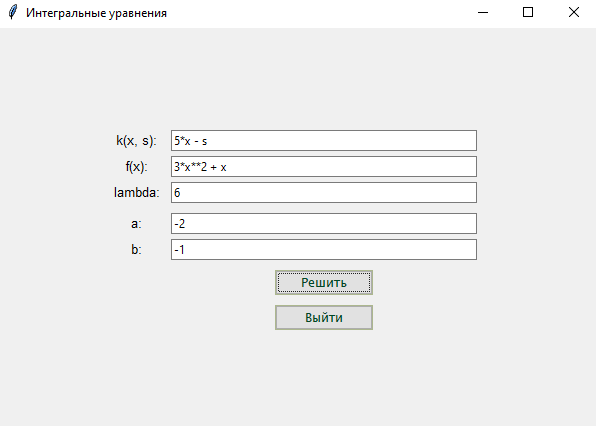


Рисунок 12 – Ввод данных примера 2

После нажатия на кнопку «Решить» на экран будет выведено окно с графиками решения данного примера, что показано на рисунке 13.

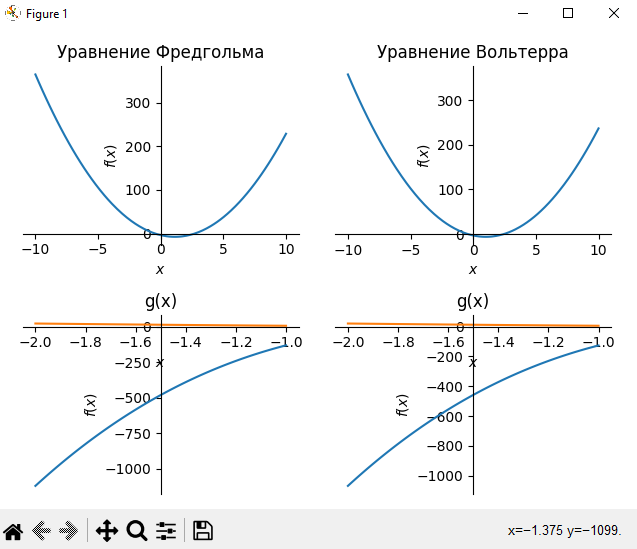


Рисунок 13 – Графики решения примера 2

Закрыв данное окно на экран будет выведено окно с подробным решением уравнений Фредгольма и Вольтерра, что показано на рисунке 14.

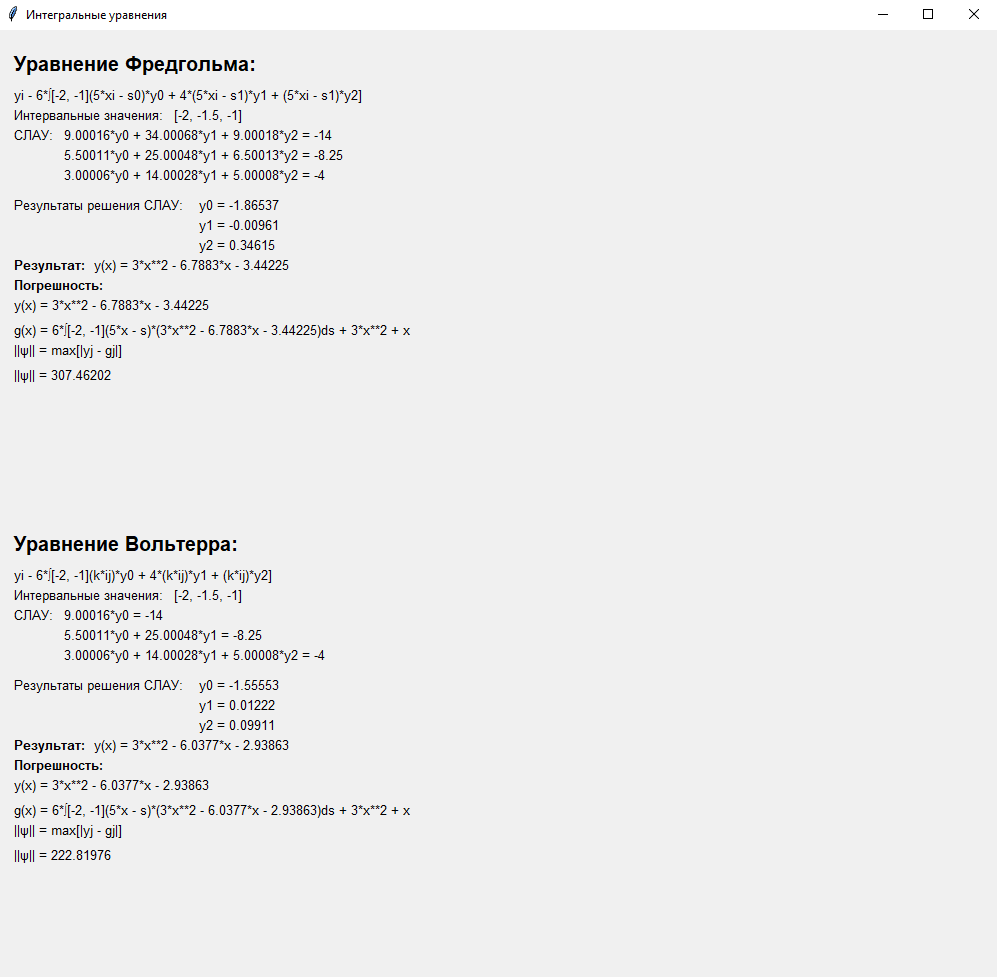


Рисунок 14 – Результат решения примера 2

В результате был получен следующий результат для уравнения Фредгольма:

.

Погрешность для данного уравнения:

.

Результат решения уравнения Вольтерра:

.

Погрешность для данного уравнения:

.

* 1. Выводы по главе

В данной главе для решения интегральных уравнений Фредгольма и Вольтерра второго рода методом конечных сумм с помощью формулы Симпсона программа успешно протестирована. Программное средство было разработано и протестировано на различных тестовых наборах данных. Результаты тестирования показали, что программа корректно решает уравнения и обеспечивает точность результатов. Проведенные эксперименты показали, что метод конечных сумм с использованием формулы трапеции является эффективным и точным способом решения интегральных уравнений Фредгольма и Вольтерра второго рода. Сравнение полученных результатов с аналитическими решениями показало хорошее совпадение. Программное средство обладает удобным интерфейсом и интуитивно понятными возможностями настройки уравнений, граничных условий и параметров метода. Пользователь может легко задать необходимые значения и получить результаты в удобном формате. Программное обеспечение успешно демонстрирует высокую производительность при решении уравнений с различными параметрами и условиями. Время выполнения программы на больших объемах данных остается приемлемым и вычисления проходят эффективно.

Таким образом, результаты тестирования продемонстрировали эффективность, точность и удобство использования разработанного программного средства для решения интегральных уравнений Фредгольма и Вольтерра второго рода методом конечных сумм с применением формулы трапеции. Полученные данные позволяют сделать вывод о успешной реализации и функционировании программы в соответствии с поставленными задачами.

Заключение

Программное средство, способное решать уравнения Фредгольма второго рода и уравнения Вольтерра второго рода, находит применение в различных областях науки и инженерии.

Разработанное ПО может применяться в математическом моделировании и численном решении интегральных уравнений в физике, биологии, экономике и других научных дисциплинах. Решение уравнений Фредгольма и Вольтерра может быть необходимо для анализа сложных систем и прогнозирования их поведения. В обработка сигналов и изображений, где программное обеспечение, способное решать интегральные уравнения, может применяться для улучшения качества обработки сигналов, фильтрации шума, восстановления изображений и других задач. В теория управления и оптимизации, где уравнения Фредгольма и Вольтерра часто возникают при моделировании динамических систем, управлении процессами и оптимизации ресурсов. Программное обеспечение, способное работать с этими уравнениями, может быть использовано для анализа и управления сложными системами. В решении задач обратного распространения, в котором в обработке сигналов, медицинской диагностике, обработке изображений и других областях уравнения Фредгольма и Вольтерра могут возникать при решении задач обратного распространения информации. Программное обеспечение помогает в данном случае восстанавливать исходные данные из полученной информации.

Это лишь некоторые примеры областей, где уравнения Фредгольма второго рода и уравнения Вольтерра второго рода, могут использоваться. В зависимости от конкретных задач и требований, они могут быть применены в различных научных, технических и прикладных областях.

В рамках данной работы, был произведён научный поиск и обзор темы интегральных уравнений Фредгольма и Вольтерра второго рода. Разработано программное средство для решения уравнений Фредгольма и Вольтерра методом конечных сумм с помощью формулы Симпсона. Таким образом поставленная задача была решена и выполнена проверка корректной работоспособности программного средства.

Перечень использованных информационных ресурсов

1. Интегральное уравнение [Электронный ресурс] // Wikipedia: [сайт]. – URL: https://ru.wikipedia.org/wiki/Интегральное\_уравнение (дата обращения: 01.06.2024)
2. Формула Симпсона [Электронный ресурс] // Wikipedia: [сайт]. – URL: https://ru.wikipedia.org/wiki/Формула\_Симпсона (дата обращения: 01.06.2024)
3. Карчевский Е.М. Интегральные уравнения: применение в математическом моделировании, численные методы решения и комплекс программ на языке MatLab: учебное пособие / Е.М. Карчевский. – Казань: Вестфалика, 2022. – 89 с.
4. Любанович Б. Простой Python. Современный стиль программирования: Книга. – «Питер». 2022 г. – 180 с.
5. Михайленко С.В. Численные методы: Учеб. пособие по лабораторному практикуму / С.В. Михайленко., Черноштан Л.И. – Харьков: Харьк. авиац. ин-т, 1984. – 127 с.

Приложение А

Техническое задание

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Профессор каф. «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_\_/Никитина А.В.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2024г. |  |

А.1 Введение

А.1.1 Наименование программного средства

Наименование программного средства – «IntegralEquations».

А.1.2 Область применения

Программное средство может применяться для решения физико-математических задач и последующего использования вычислений.

А.2 Основание для разработки

Разработка ведется на основании документа «Учебный план для студентов ВУЗа» направление 02.03.03 «Математическое обеспечение и администрирование информационных систем» кафедры «Программное обеспечение вычислительное техники и автоматизированных систем» факультета «Информатика и вычислительная техника» Донского Государственного Технического Университета.

А.3 Назначение разработки

А.3.1 Функциональное назначение

Функциональное назначение программного средства заключается в численном решении линейных интегральных уравнений Фредгольма и Вольтерра, построении графиков полученных решений и оценке погрешности численного измерения.

А.3.2 Эксплуатационное назначение

Эксплуатационное назначение состоит в использовании программного средства на персональном компьютере (ПК) с операционной системой Windows.

А.4 Требования к программе

А.4.1 Требования к функциональным характеристикам

Программное средство должно осуществлять следующие функции:

* Решение линейных интегральных уравнений Фредгольма и Вольтерра;
* Вычисление погрешности методом подстановки;
* Построение графиков полученных решений;

А.4.2 Требования к надежности

Для надежной работы программного средства необходимо, чтобы выполнялись следующие условия:

* бесперебойное питание технического средства, на котором находится продукт;
* регулярная проверка программного средства на наличие вирусов;
* отсутствие шума и спокойная обстановка.

А.4.2.1 Входные данные

В качестве входных данных являются функции – ядро и свободная функция, отрезок интегрирования и параметр при определенном интеграле.

А.4.2.2 Выходные данные

Выходные данные предоставляются в виде графиков в программных окнах, а также текста в консоли.

А.4.3 Условия эксплуатации

Для стабильного функционирования и оптимальной работы программного продукта необходимо соблюдение всех требований и правил эксплуатации вычислительной техники. Каких-либо требований к пользователю данного приложения нет.

А.4.4 Требование к составу и параметрам технических средств

В состав технических средств должен входить ПК с операционной системой windows 7 и выше, включающая в себя:

* 64-разрядный процессор;
* оперативная память объемом не менее 1 Гбайт.

Дополнительные требования и ограничения к составу и параметрам технических средств не вводится.

А.4.5 Требования к исходным кодам и языкам программирования

Программное средство должно быть реализовано с применением языков программирования Python. В качестве интегрированной среды разработки программы должен быть использован Visual Studio Code.

Дополнительные требования и ограничения к информационной и программной совместимости не вводятся.

А.4.6 Требования к упаковке и маркировки

К упаковке и маркировки специальных требований не предъявляются.

А.4.7 Требования к транспортировке и хранению

Условия транспортирования, места хранения, условия складирования и сроки хранения в различных условиях должны соответствовать требованиям, предъявляемым к носителям информации, на которых будет содержаться данное программное изделие. Программное средство может храниться на любых цифровых носителях информации (жесткий диск, компакт – диск, флэш накопитель и т. п.).

А.5 Требование к программной документации

Программная документация состоит из следующего:

* титульный лист;
* лист задания на учебную ознакомительную практику;
* пояснительная записка к учебной ознакомительной практике;
* техническое задание по ГОСТ 19.201-78 ЕСПД;
* исходный код программного средства по ГОСТ 19.401-79 ЕСПД.

А.6 Стадии и этапы разработки

Реализация программного средства состояла из следующих этапов:

* постановка задачи (05.02.2024);
* изучение предметной области (16.02.2024 – 25.03.2024);
* алгоритмическое конструирование (01.04.2024 – 10.04.2024);
* программная реализация (27.04.2024 – 01.05.2024);
* тестирование приложения (02.05.2024 – 10.05.2024);
* разработка отчета (12.05.2024 – 06.06.2024).

А.7 Порядок и контроль приемки

Порядок и контроль приемки определяются заведующим кафедрой «ПОВТ и АС», подразумевающие собой демонстрацию показателя владения средствами для разработки программных средств в различных направлениях.

Главным требованием к приемке является наличие корректного работающего программного средства и отчета, предоставленного в печатном виде.

|  |  |
| --- | --- |
| Разработчик технического задания:  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. | /Волкова Эмилия Юрьевна /  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Приложение Б – Исходный код программного средства

***Листинг 1 – Исходный код пользовательского интерфейса***

from tkinter import ttk, Tk, Toplevel, messagebox

import integral\_equations

def get\_input\_data():

try:

k = str(input\_equation\_k.get())

f = str(input\_equation\_f.get())

l = int(lambda\_input.get())

a = int(a\_input.get())

b = int(b\_input.get())

except:

messagebox.showerror("Ошибка!", "Пожалуйста, проверьте корректность данных!")

return 0

get\_results(k, f, l, a, b)

def get\_results(k, f, l, a, b):

try:

main\_eq1, diap, slau1, slau\_answers1, result\_equ1 = integral\_equations.eq\_Fredgolm(k, f, l, a, b)

main\_eq2, diap, slau2, slau\_answers2, result\_equ2 = integral\_equations.eq\_Volterr(k, f, l, a, b)

main\_eq\_f = f'({k})\*({result\_equ1})'

main\_eq\_v = f'({k})\*({result\_equ2})'

fredgolm\_rate, volterr\_rate = integral\_equations.grafic\_create(result\_equ1, result\_equ2, main\_eq\_f, main\_eq\_v, l, f, a, b)

except:

messagebox.showerror("Ошибка!", "Невозможно завершить расчёт! Проверьте корректность введённых данных.")

return 0

output\_window = Toplevel()

output\_window.geometry('%dx%d+%d+%d' % (1000, 950, (window.winfo\_screenwidth()/2) - (1000/2), (window.winfo\_screenheight()/2) - (950/2)))

print\_results(output\_window, 'Фредгольма', main\_eq1, diap, slau1, slau\_answers1, result\_equ1, f"{l}\*∫[{a}, {b}]{main\_eq\_f}ds + {f}", fredgolm\_rate, 15, 20)

print\_results(output\_window, 'Вольтерра', main\_eq2, diap, slau2, slau\_answers2, result\_equ2, f"{l}\*∫[{a}, {b}]{main\_eq\_v}ds + {f}", volterr\_rate, 15, 500)

def print\_results(output\_window, title, main\_eq, diap, slau, slau\_answers, result\_equ, equation, rate, x, y):

output\_fredgolm = ttk.Label(output\_window, text=f'Уравнение {title}: ', font=('arial', 15, "bold"))

output\_fredgolm.place(x=x, y=y)

output\_fredgolm\_equ = ttk.Label(output\_window, text=main\_eq, font=('arial', 10))

output\_fredgolm\_equ.place(x=x, y=y+35)

output\_diap\_title = ttk.Label(output\_window, text='Интервальные значения:', font=('arial', 10))

output\_diap\_title.place(x=x, y=y+55)

output\_diap = ttk.Label(output\_window, text=str(diap), font=('arial', 10))

output\_diap.place(x=x+160, y=y+55)

output\_slau\_title = ttk.Label(output\_window, text='СЛАУ:', font=('arial', 10))

output\_slau\_title.place(x=x, y=y+75)

output\_slau1\_1 = ttk.Label(output\_window, text=slau[0], font=('arial', 10))

output\_slau1\_1.place(x=x+50, y=y+75)

output\_slau1\_2 = ttk.Label(output\_window, text=slau[1], font=('arial', 10))

output\_slau1\_2.place(x=x+50, y=y+95)

output\_slau1\_3 = ttk.Label(output\_window, text=slau[2], font=('arial', 10))

output\_slau1\_3.place(x=x+50, y=y+115)

output\_slau\_answers\_title = ttk.Label(output\_window, text='Результаты решения СЛАУ:', font=('arial', 10))

output\_slau\_answers\_title.place(x=x, y=y+145)

output\_slau\_answers\_1 = ttk.Label(output\_window, text=f'y0 = {slau\_answers[0]}', font=('arial', 10))

output\_slau\_answers\_1.place(x=x+185, y=y+145)

output\_slau\_answers\_2 = ttk.Label(output\_window, text=f'y1 = {slau\_answers[1]}', font=('arial', 10))

output\_slau\_answers\_2.place(x=x+185, y=y+165)

output\_slau\_answers\_3 = ttk.Label(output\_window, text=f'y2 = {slau\_answers[2]}', font=('arial', 10))

output\_slau\_answers\_3.place(x=x+185, y=y+185)

output\_res\_eq\_title = ttk.Label(output\_window, text='Результат:', font=('arial', 10, 'bold'))

output\_res\_eq\_title.place(x=x, y=y+205)

output\_res\_eq = ttk.Label(output\_window, text=f"y(x) = {result\_equ}", font=('arial', 10))

output\_res\_eq.place(x=x+80, y=y+205)

rate\_title = ttk.Label(output\_window, text='Погрешность:', font=('arial', 10, 'bold'))

rate\_title.place(x=x, y=y+225)

rate\_eq1 = ttk.Label(output\_window, text=f"y(x) = {result\_equ}", font=('arial', 10))

rate\_eq1.place(x=x, y=y+245)

rate\_eq2 = ttk.Label(output\_window, text=f"g(x) = {equation}", font=('arial', 10))

rate\_eq2.place(x=x, y=y+270)

rate\_formula = ttk.Label(output\_window, text=f"||ψ|| = max[|yj - gj|]", font=('arial', 10))

rate\_formula.place(x=x, y=y+290)

rate\_ans = ttk.Label(output\_window, text=f"||ψ|| = {round(rate, 5)}", font=('arial', 10))

rate\_ans.place(x=x, y=y+315)

window = Tk()

window.title("Интегральные уравнения")

window.geometry('%dx%d+%d+%d' % (600, 400, (window.winfo\_screenwidth()/2) - (600/2), (window.winfo\_screenheight()/2) - (400/2)))

style\_btn = ttk.Style()

style\_btn.configure("TButton", font=('algerian', 10), foreground="#004524", background="#ACB78E")

style\_label = ttk.Style()

style\_label.configure("TLabel", font=('italic', 10))

style\_check\_btn = ttk.Style()

style\_check\_btn.configure("TCheckbutton", font=('algerian', 10))

frame = ttk.Frame(window)

frame.pack(expand=True)

quation\_k\_txt = ttk.Label(frame, text="k(x, s): ")

quation\_k\_txt.grid(row=2, column=1)

input\_equation\_k = ttk.Entry(frame, width=50)

input\_equation\_k.grid(row=2, column=2, pady=5)

equation\_f\_txt = ttk.Label(frame, text="f(x): ")

equation\_f\_txt.grid(row=3, column=1)

input\_equation\_f = ttk.Entry(frame, width=50)

input\_equation\_f.grid(row=3, column=2)

lambda\_txt = ttk.Label(frame, text="lambda: ")

lambda\_txt.grid(row=4, column=1)

lambda\_input = ttk.Entry(frame, width=50)

lambda\_input.grid(row=4, column=2, padx=5, pady=5)

a\_txt = ttk.Label(frame, text="a: ")

a\_txt.grid(row=5, column=1)

a\_input = ttk.Entry(frame, width=50)

a\_input.grid(row=5, column=2, pady=5)

b\_txt = ttk.Label(frame, text="b: ")

b\_txt.grid(row=6, column=1)

b\_input = ttk.Entry(frame, width=50)

b\_input.grid(row=6, column=2, padx=5)

start\_btn = ttk.Button(frame, text='Решить', command=get\_input\_data)

start\_btn.grid(row=8, column=2, pady=10)

exit\_btn = ttk.Button(frame, text="Выйти", command=quit)

exit\_btn.grid(row=9, column=2)

window.mainloop()

***Листинг 2 – Исходный код решения уравнений Фредгольма и Вольтерра***

from sympy.plotting import plot, PlotGrid, plot\_parametric

import sympy as sp

import numpy as np

import re

def eq\_Fredgolm(k, f, l, a, b):

main\_equation = f"yi - {l}\*∫[{a}, {b}]({(k.replace('x', 'xi')).replace('s', 's0')})\*y0 + 4\*({(k.replace('x', 'xi').replace('s', 's1'))})\*y1 + ({(k.replace('x', 'xi')).replace('s', 's1')})\*y2]" #само уравнение фредгольма

diap\_parts = [a, a+((b-a)/2), b]

k1\_s0 = eval(re.sub('s0', str(diap\_parts[0]), re.sub('x0', str(diap\_parts[0]), (k.replace('x', 'x0')).replace('s', 's0'))))

k1\_s1 = eval(re.sub('s1', str(diap\_parts[1]), re.sub('x0', str(diap\_parts[0]), (k.replace('x', 'x0')).replace('s', 's1'))))

k1\_s2 = eval(re.sub('s2', str(diap\_parts[2]), re.sub('x0', str(diap\_parts[0]), (k.replace('x', 'x0')).replace('s', 's2'))))

k2\_s0 = eval(re.sub('s0', str(diap\_parts[0]), re.sub('x1', str(diap\_parts[1]), (k.replace('x', 'x1')).replace('s', 's0'))))

k2\_s1 = eval(re.sub('s1', str(diap\_parts[1]), re.sub('x1', str(diap\_parts[1]), (k.replace('x', 'x1')).replace('s', 's1'))))

k2\_s2 = eval(re.sub('s2', str(diap\_parts[2]), re.sub('x1', str(diap\_parts[1]), (k.replace('x', 'x1')).replace('s', 's2'))))

k3\_s0 = eval(re.sub('s0', str(diap\_parts[0]), re.sub('x2', str(diap\_parts[2]), (k.replace('x', 'x2')).replace('s', 's0'))))

k3\_s1 = eval(re.sub('s1', str(diap\_parts[1]), re.sub('x2', str(diap\_parts[2]), (k.replace('x', 'x2')).replace('s', 's1'))))

k3\_s2 = eval(re.sub('s2', str(diap\_parts[2]), re.sub('x2', str(diap\_parts[2]), (k.replace('x', 'x2')).replace('s', 's2'))))

y0, y1, y2 = sp.symbols('y0 y1 y2')

vector\_b = [eval(re.sub(f'x{i}', str(diap\_parts[i]), f.replace('x', f'x{i}'))) for i in range(len(diap\_parts))]

equation1 = eval(f"y0 + ({(-1)\*l\*round((b-a)/6, 5)\*k1\_s0})\*y0 + ({(-1)\*l\*round((b-a)/6, 5)\*4\*k1\_s1})\*y1 + ({(-1)\*l\*round((b-a)/6, 5)\*k1\_s2})\*y2")

equation2 = eval(f"y1 + ({(-1)\*l\*round((b-a)/6, 5)\*k2\_s0})\*y0 + ({(-1)\*l\*round((b-a)/6, 5)\*4\*k2\_s1})\*y1 + ({(-1)\*l\*round((b-a)/6, 5)\*k2\_s2})\*y2")

equation3 = eval(f"y2 + ({(-1)\*l\*round((b-a)/6, 5)\*k3\_s0})\*y0 + ({(-1)\*l\*round((b-a)/6, 5)\*4\*k3\_s1})\*y1 + ({(-1)\*l\*round((b-a)/6, 5)\*k3\_s2})\*y2")

slau = [

f"{equation1} = {vector\_b[0]}",

f"{equation2} = {vector\_b[1]}",

f"{equation3} = {vector\_b[2]}"

]

matrix = [

[round(1+(-1)\*l\*round((b-a)/6, 5)\*k1\_s0, 5), round((-1)\*l\*round((b-a)/6, 5)\*4\*k1\_s1, 5), round((-1)\*l\*round((b-a)/6, 5)\*k1\_s2, 5)],

[round((-1)\*l\*round((b-a)/6, 5)\*k2\_s0, 5), round(1+(-1)\*l\*round((b-a)/6, 5)\*4\*k2\_s1, 5), round((-1)\*l\*round((b-a)/6, 5)\*k2\_s2, 5)],

[round((-1)\*l\*round((b-a)/6, 5)\*k3\_s0, 5), round((-1)\*l\*round((b-a)/6, 5)\*4\*k3\_s1, 5), round(1+(-1)\*l\*round((b-a)/6, 5)\*k3\_s2, 5)]

]

matrix\_answers = np.linalg.inv(matrix).dot(vector\_b)

for i in range(len(matrix\_answers)):

matrix\_answers[i] = round(matrix\_answers[i], 5)

k1\_s0\_new = re.sub('s0', str(diap\_parts[0]), re.sub('x0', str(diap\_parts[0]), k.replace('s', 's0')))

k1\_s1\_new = re.sub('s1', str(diap\_parts[1]), re.sub('x0', str(diap\_parts[0]), k.replace('s', 's1')))

k1\_s2\_new = re.sub('s2', str(diap\_parts[2]), re.sub('x0', str(diap\_parts[0]), k.replace('s', 's2')))

y = f"({f}) + {(l/6)}\*({(k1\_s0\_new)})\*({matrix\_answers[0]}) + {(l/6)\*4}\*({(k1\_s1\_new)})\*({matrix\_answers[1]}) + {(l/6)}\*({(k1\_s2\_new)})\*({matrix\_answers[2]})"

result = sp.trigsimp(y)

return main\_equation, diap\_parts, slau, matrix\_answers, result

def eq\_Volterr(k, f, l, a, b):

main\_equation = f"yi - {l}\*∫[{a}, {b}](k\*ij)\*y0 + 4\*(k\*ij)\*y1 + (k\*ij)\*y2]"

diap\_parts = [a, a+((b-a)/2), b]

k1\_s0 = eval(re.sub('s0', str(diap\_parts[0]), re.sub('x0', str(diap\_parts[0]), (k.replace('x', 'x0')).replace('s', 's0'))))

k2\_s0 = eval(re.sub('s0', str(diap\_parts[0]), re.sub('x1', str(diap\_parts[1]), (k.replace('x', 'x1')).replace('s', 's0'))))

k2\_s1 = eval(re.sub('s1', str(diap\_parts[1]), re.sub('x1', str(diap\_parts[1]), (k.replace('x', 'x1')).replace('s', 's1'))))

k3\_s0 = eval(re.sub('s0', str(diap\_parts[0]), re.sub('x2', str(diap\_parts[2]), (k.replace('x', 'x2')).replace('s', 's0'))))

k3\_s1 = eval(re.sub('s1', str(diap\_parts[1]), re.sub('x2', str(diap\_parts[2]), (k.replace('x', 'x2')).replace('s', 's1'))))

k3\_s2 = eval(re.sub('s2', str(diap\_parts[2]), re.sub('x2', str(diap\_parts[2]), (k.replace('x', 'x2')).replace('s', 's2'))))

y0, y1, y2 = sp.symbols('y0 y1 y2')

vector\_b = [eval(re.sub(f'x{i}', str(diap\_parts[i]), f.replace('x', f'x{i}'))) for i in range(len(diap\_parts))]

equation1 = eval(f"y0 + ({(-1)\*l\*round((b-a)/6, 5)\*k1\_s0})\*y0 + ({(-1)\*l\*round((b-a)/6, 5)\*4\*0})\*y1 + ({(-1)\*l\*round((b-a)/6, 5)\*0})\*y2")

equation2 = eval(f"y1 + ({(-1)\*l\*round((b-a)/6, 5)\*k2\_s0})\*y0 + ({(-1)\*l\*round((b-a)/6, 5)\*4\*k2\_s1})\*y1 + ({(-1)\*l\*round((b-a)/6, 5)\*0})\*y2")

equation3 = eval(f"y2 + ({(-1)\*l\*round((b-a)/6, 5)\*k3\_s0})\*y0 + ({(-1)\*l\*round((b-a)/6, 5)\*4\*k3\_s1})\*y1 + ({(-1)\*l\*round((b-a)/6, 5)\*k3\_s2})\*y2")

slau = [

f"{equation1} = {vector\_b[0]}",

f"{equation2} = {vector\_b[1]}",

f"{equation3} = {vector\_b[2]}"

]

matrix = [

[round(1+(-1)\*l\*round((b-a)/6, 5)\*k1\_s0, 5), round((-1)\*l\*round((b-a)/6, 5)\*4\*0, 5), round((-1)\*l\*round((b-a)/6, 5)\*0, 5)],

[round((-1)\*l\*round((b-a)/6, 5)\*k2\_s0, 5), round(1+(-1)\*l\*round((b-a)/6, 5)\*4\*k2\_s1, 5), round((-1)\*l\*round((b-a)/6, 5)\*0, 5)],

[round((-1)\*l\*round((b-a)/6, 5)\*k3\_s0, 5), round((-1)\*l\*round((b-a)/6, 5)\*4\*k3\_s1, 5), round(1+(-1)\*l\*round((b-a)/6, 5)\*k3\_s2, 5)]

]

matrix\_answers = np.linalg.inv(matrix).dot(vector\_b)

for i in range(len(matrix\_answers)):

matrix\_answers[i] = round(matrix\_answers[i], 5)

k1\_s0\_new = re.sub('s0', str(diap\_parts[0]), re.sub('x0', str(diap\_parts[0]), k.replace('s', 's0')))

k1\_s1\_new = re.sub('s1', str(diap\_parts[1]), re.sub('x0', str(diap\_parts[0]), k.replace('s', 's1')))

k1\_s2\_new = re.sub('s2', str(diap\_parts[2]), re.sub('x0', str(diap\_parts[0]), k.replace('s', 's2')))

y = f"({f}) + {(l/6)}\*({(k1\_s0\_new)})\*({matrix\_answers[0]}) + {(l/6)\*4}\*({(k1\_s1\_new)})\*({matrix\_answers[1]}) + {(l/6)}\*({(k1\_s2\_new)})\*({matrix\_answers[2]})"

result = sp.trigsimp(y)

return main\_equation, diap\_parts, slau, matrix\_answers, result

def rate\_result(y1, y2, a, b):

results = []

x = np.arange(a, b, 0.005)

for i in range(len(x)):

results.append(abs(eval(str(y1).replace('x', str(x[i]))) - eval(str(y2).replace('x', str(x[i])))))

return max(results)

def func(x, y):

return str(y).replace('s', str(x))

def grafic\_create(y1, y2, y3, y4, l, f, a, b):

x, s = sp.symbols("x s")

y3 = sp.expand(eval(y3))

y4 = sp.expand(eval(y4))

equation1 = sp.expand(eval(f"{l}\*({sp.integrate(eval(func(s, y3)), (s, a, b))}) + ({f})"))

equation2 = sp.expand(eval(f"{l}\*({sp.integrate(eval(func(s, y4)), (s, a, b))}) + ({f})"))

p1 = plot(y1, show=False, title='Уравнение Фредгольма')

p2 = plot(y2, show=False, title='Уравнение Вольтерра')

p3 = plot(equation1, (x, a, b), show=False, title='g(x)')

p3.append(plot(y1, (x, a, b), show=False)[0])

p4 = plot(equation2, (x, a, b), show=False, title='g(x)')

p4.append(plot(y2, (x, a, b), show=False)[0])

PlotGrid(2, 2, p1, p2, p3, p4)

rate\_res1 = rate\_result(equation1, y1, a, b)

rate\_res2 = rate\_result(equation2, y2, a, b)

return rate\_res1, rate\_res2