МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»**

Кафедра Математической физики и вычислительной математики

Вариативность алгоритмизации численного решения интегральных уравнений

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы

направление 01.03.02 — Прикладная математика и информатика

механико-математического факультета

Колган Артура Сергеевича

Научный руководитель
доцент, к.ф.-м.н., доцент Д. В. Поплавский

Зав. кафедрой
д.ф.-м.н., профессор В. А. Юрко

Саратов 2024

Введение. Интегральные уравнения являются важным инструментом в математическом моделировании разнообразных процессов, происходящих в природе и технике. Их применение охватывает такие области, как физика, химия, биология, экономика и инженерное дело. Ввиду сложности аналитического решения интегральных уравнений особое значение приобретает разработка эффективных численных методов их решения. Цель работы.Разработка и программная реализация на языке Python вариативного алгоритма численного решения интегральных уравнений методом резольвент, который бы учитывал специфические особенности различных типов интегральных уравнений и условий их применения.

Задачи. Исследование теоретических основ интегральных уравнений и методов их численного решения. Разработка вариативного алгоритма численного решения интегральных уравнений методом резольвент.Реализация разработанного алгоритма в программном обеспечении на языке Python.

Структура бакалаврской работы: Первая глава посвящена теоретической основе интегральных уравнений. Во второй главе рассматриваются численные методы решения интегральных уравнений. В третьей главе представлена вариативность алгоритмизации численного решения интегральных уравнений. В четвёртой - программная реализация и численные эксперименты.

Основное содержание работы. Рассматриваются основные типы линейных интегральных уравнений второго рода:

Фредгольма -

$$x(t) = \lambda \int_a^b Q(t,s)x(s)ds + f(t) \tag{1.2}$$

и Вольтерра —

$$x(t) = \lambda \int_a^t Q(t,s)x(s)ds + f(t). \tag{1.3}$$

Заданная функция $f(t)$ — свободный член - и неизвестная функция $x(t)$ — решение - в этих уравнениях зависят от переменной t, изменяющейся на отрезке $[a,b]$. Функция двух переменных $Q(t,s)$, называемая ядром интегрального уравнения, определяется на множестве точек квадрата $[a,b]\times[a,b]$

в случае интегрального уравнения Фредгольма (рис.1) и треугольника $a \leq s \leq t \leq b$ в случае уравнения Вольтерра (рис.2).

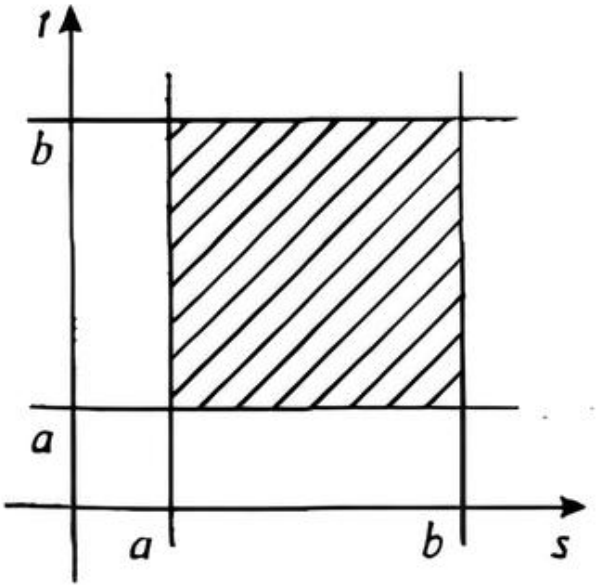

Рисунок 1

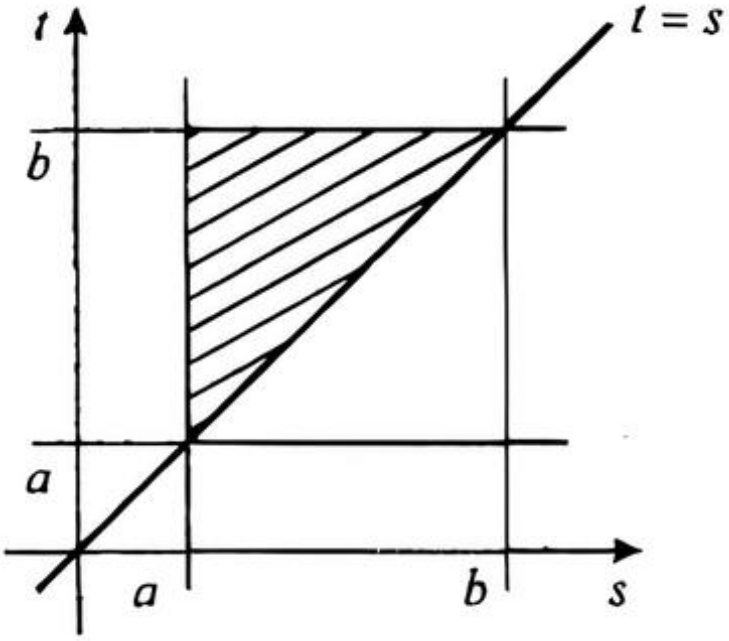

Рисунок 2

Также приведено необходимое определение вырожденного ядра интегрального уравнения, для решения интегральных уравнений.

Определение 1 *Ядро $K(x,t)$ интегрального уравнения (1.2) называется вырожденным, если оно может быть представлено в виде*

$$K(x,t) = \sum_{k=1}^{n} p_k(x) q_k(t). \tag{2.16}$$

В разделе **«Методы численного решения интегральных уравнений»** разбираются такие методы как:

— Квадратурный метод решения интегральных уравнений

— Метод последовательных приближений

— Решение уравнений с вырожденным ядром

— Квадратурно-итерационный метод построения резольвент

Все методы тщательно разбирались с постановки решения, а также были приведены некоторые примеры, показывающие различия в сходимости, в за-

висимости от типа интегрального уравнения.

Как пример решение интегрального уравнения Фредгольма:

$$y(x) = \sin \pi x + \frac{1}{2} \int_0^1 y(t) \mathrm{d}t.$$

и решение интегрального уравнения Вольтерра:

$$y(x) = 1 - \int_0^x (x - t) y(t) \mathrm{d}t$$

(Рисунок 3) - Решение уравнения и сходимость метода для уравнения Фредгольма.
(Рисунок 4) Решение уравнения и сходимость метода для уравнения Вольтерра.

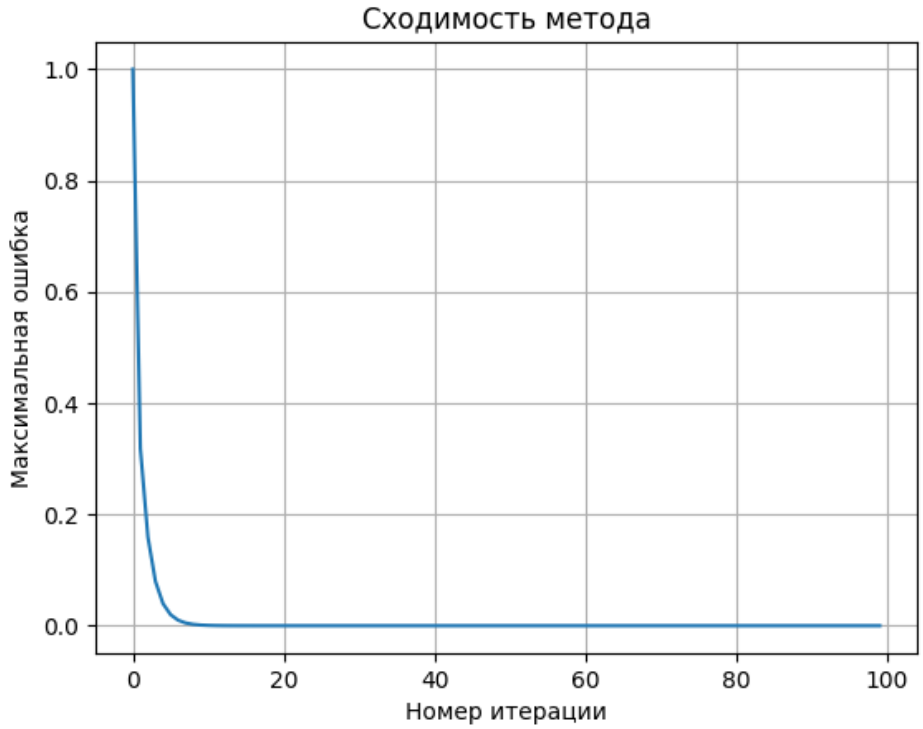

Рисунок 3

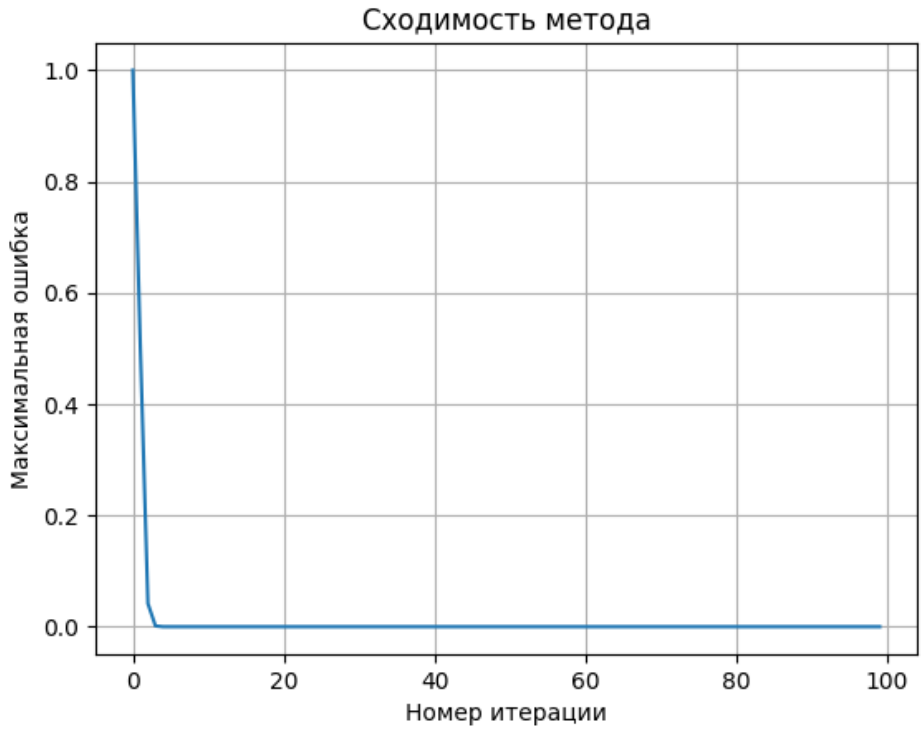

Рисунок 4

Как можно заметить график сходимости метода у этих двух уравнений разный, для уравнения Фредгольма на десятой итерации максимальная ошибка плавно спускается к 0, а у уравнения Вольтера максимальная ошибка с 0,04 резко падает почти до нуля. Связанно это с тем, что структура уравнения Вольтерра, где границы интегрирования зависят от x, может приводить к тому, что ошибка быстро уменьшается на начальных итерациях. Это связано с тем, что на первых итерациях мы вычисляем интеграл по очень маленькому интервалу, что позволяет получить более точное приближение. А также метод трапеций в сочетании с методом последовательных приближений для уравнения Вольтерра дает более быструю сходимость на начальных итерациях.

В разделе **«Вариативность алгоритмизации численного решения интегральных уравнений»** мы рассмотрели описание подходов к вариативности решения интегральных уравнений.

Такие, как:

— Выбор метода решения

— Выбор метода квадратур

— Выбор метода регуляризации

— Адаптивная квадратура

— Комбинирование методов

— Разработка специализированных методов

В разделе **«Программная реализация метода»** были рассмотрены все необходимы шаги, для реализации метода из главы о построении резольвенты квадратурно-итерационным методом. А также пояснили все шаги, реализовав код на примере решения следующего интегрального уравнения:

$$x(t) - 0,5 \int_0^1 \exp^{t-s} x(s)ds = t(1-t)$$

. Программа решает интегральное уравнение второго рода с помощью квадратурно-итерационного метода.

Основные этапы программы:

— Определение функции ядра $\mathbf{Q}(\mathbf{t}, \mathbf{s})$ и правой части $f(t)$ интегрального уравнения.

— Построение квадратурных весов и узлов.

— Проверка спектрального радиуса матрицы для проверки условий сходимости.

— Итерационный процесс для вычисления приближенной резольвенты методом Шульца.

— Решение интегрального уравнения и визуализация результатов.

Шаги программы:

Импортируем библиотеки numpy и matplotlib — библиотеки для работы с числовыми массивами и построения графиков. scipy.sparse.linalg.eigs — функция для вычисления нескольких собственных значений и собственных

векторов разреженных матриц.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse.linalg import eigs
```

Здесь определяется функция ядра $\mathbf{Q}(\mathbf{t}, \mathbf{s}) = \exp^{\mathbf{t}-\mathbf{s}}$

В контексте интегрального уравнения Фредгольма ядро определяет, как входная функция трансформируется в выходную.

```
def Q(t, s):
    return np.exp(t - s)
```

Определение правой части $f(t) = t(1 - t)$
Эта функция представляет известное значение, к которому стремится решение интегрального уравнения.

```
# Определим правую часть f(t)
def f(t):
    return t * (1 - t)
```

Функция для построения вектора квадратурных весов для равномерной сетки. Здесь используется простая формула прямоугольных квадратур.

```
# Функция для построения матрицы квадратурных весов
def build_quadrature_weights(a, b, n):
    return np.ones(n) * (b - a) / n
```

Функция для построения узлов квадратурной сетки, равномерно распределенных на интервале $[a, b]$

```
# Функция для построения узлов квадратурной сетки
def build_quadrature_nodes(a, b, n):
    return np.linspace(a, b, n)
```

Функция для вычисления спектрального радиуса матрицы. Спектральный радиус — это максимальное по модулю собственное значение матрицы. Используется функция eigs из sciру, которая вычисляет несколько собственных значений и соответствующих собственных векторов.

```
# Функция для вычисления спектрального радиуса матрицы
(с использованием scipy.sparse.linalg.eigs)
def spectral_radius(matrix, k=6):
    eigenvalues, _ = eigs(matrix, k=k)
    return max(abs(eigenvalues))
```

Определяются узлы и веса квадратурной сетки на интервале $[a, b]$ с n узлами. Параметр λ используется в интегральном уравнении.

```
# Основная функция для решения интегрального уравнения
def solve_integral_equation(a, b, n,
lambd, tol=1e-10, max_iter=1000):
    nodes = build_quadrature_nodes(a, b, n)
    weights = build_quadrature_weights(a, b, n)
```

Инициализация и построение матрицы $\mathbf{Q}$ размером $n \times n$ по заданному ядру $\mathbf{Q}(\mathbf{t}, \mathbf{s})$

```
 # Инициализация матриц Q
    Q_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            Q_matrix[i, j] = Q(nodes[i], nodes[j])
```

Проверка условия спектрального радиуса для матрицы $\operatorname{diag} \lambda \mathbf{AQ}$, где $\mathbf{A} = \operatorname{diag}(weweights)$. Если условие не выполняется, итерационный процесс не сходится, и функция выдает ошибку.

```
# Проверка условия спектрального радиуса
    if spectral_radius(lambd * np.dot(Q_matrix,
    np.diag(weights))) >= 1:
        raise ValueError("Условие спектрального
```

```
        радиуса не выполнено.
        Не удается вычислить разрешающую способность.")
```

Итерационный процесс Шульца для нахождения обратной матрицы. Начальное приближение $\mathbf{U}_0 = \mathbf{E}$. На каждой итерации вычисляется ошибка, и процесс продолжается до тех пор, пока ошибка не станет меньше заданной точности tol.

```
 # Итерационный процесс для вычисления резольвенты (Шульц)
    E = np.eye(n)
    U_k = E.copy()  # Начальное приближение U_0 = E
    errors = []

    for k in range(max_iter):
        Psi_k = E - (E - lambd * np.dot(Q_matrix,
        np.diag(weights))) @ U_k
        U_k1 = U_k + U_k @ Psi_k

        error = np.linalg.norm(U_k1 - U_k)
        errors.append(error)

        if error < tol:
            break
        U_k = U_k1
```

Вычисление матрицы резольвенты $\mathbf{R} = \mathbf{Q}\mathbf{U}_k$.

```
R_matrix = Q_matrix @ U_k
```

Решение интегрального уравнения. Вектор f вычисляется в узлах квадратурной сетки. Решение уравнения $(\mathbf{E} - \lambda\mathbf{A}\mathbf{R})x = f$ находится с помощью np.linalg.solve.

```
# Решение интегрального уравнения
    f_values = np.array([f(node) for node in nodes])
    x = np.linalg.solve(
```

```
        np.eye(n) - lambd * np.dot(R_matrix,
        np.diag(weights)), f_values
    )

    return nodes, x, R_matrix, errors
```

Основной скрипт для запуска решения интегрального уравнения. Определяются параметры задачи, запускается функция решения, а затем результаты визуализируются с помощью matplotlib. Ошибки на каждой итерации также визуализируются для оценки сходимости итерационного процесса.

```
# Пример использования
a, b = 0, 1  # Интервал интегрирования
n = 10  # Количество узлов квадратурной сетки
lambd = 0.5  # Параметр lambda

try:
    nodes, solution, R_matrix,
    errors = solve_integral_equation(a, b, n, lambd)

    # Визуализация результатов
    plt.figure(figsize=(12, 8))

    # График исходной функции f(t)
    plt.subplot(3, 1, 1)
    t = np.linspace(a, b, 100)
    plt.plot(t, f(t), label="f(t)")
    plt.title("Исходная функция f(t)")
    plt.legend()

    # Решения интегрального уравнения
    plt.subplot(3, 1, 2)
    plt.plot(nodes, solution, "o-", label="Решение x(t)")
    plt.title("Решение интегрального уравнения")
```

```
    plt.legend()


    # График резольвенты R(t, s; lambda) для t = nodes[0]
    plt.subplot(3, 1, 3)
    for i in range(n):
        plt.plot(nodes, R_matrix[i, :],
        label=f"R({nodes[i]:.2f}, s; lambda)")
    plt.title("Резольвента R(t, s; lambda)")
    plt.legend()

    plt.tight_layout()
    plt.show()

    # Вывод матрицы ошибок
    plt.figure(figsize=(8, 6))
    plt.plot(errors, "o-", label="Ошибка на каждой итерации")
    plt.title("Сходимость итерационного процесса")
    plt.xlabel("Номер итерации")
    plt.ylabel("Ошибка")
    plt.legend()
    plt.yscale("log")
    plt.show()

except ValueError as e:
    print(e)
```

Программа выдает следующие результаты

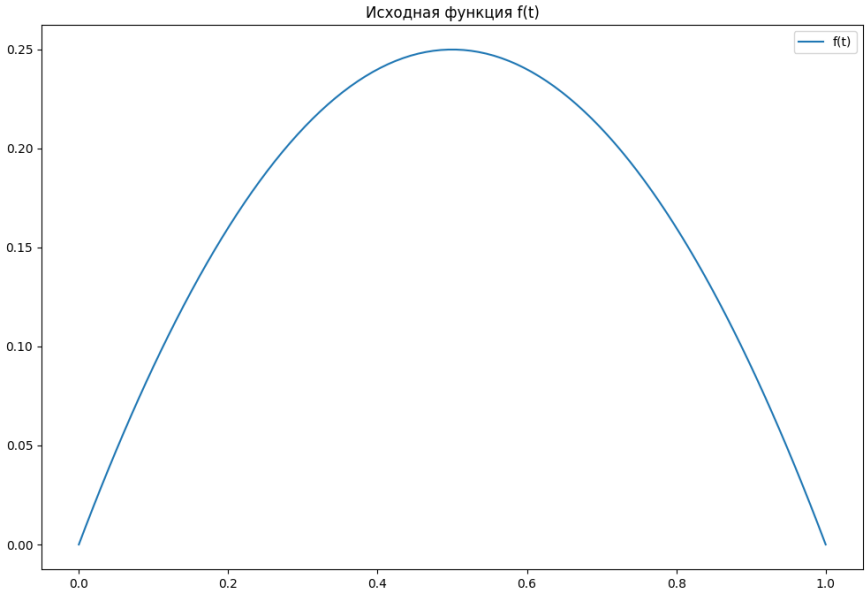

Рисунок 5

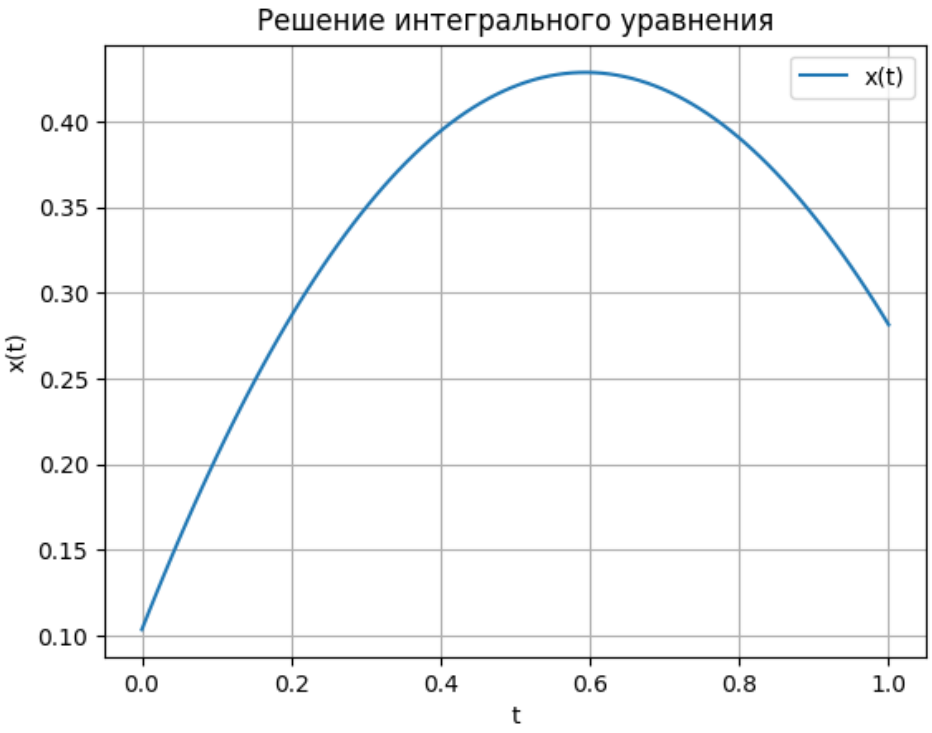

Рисунок 6

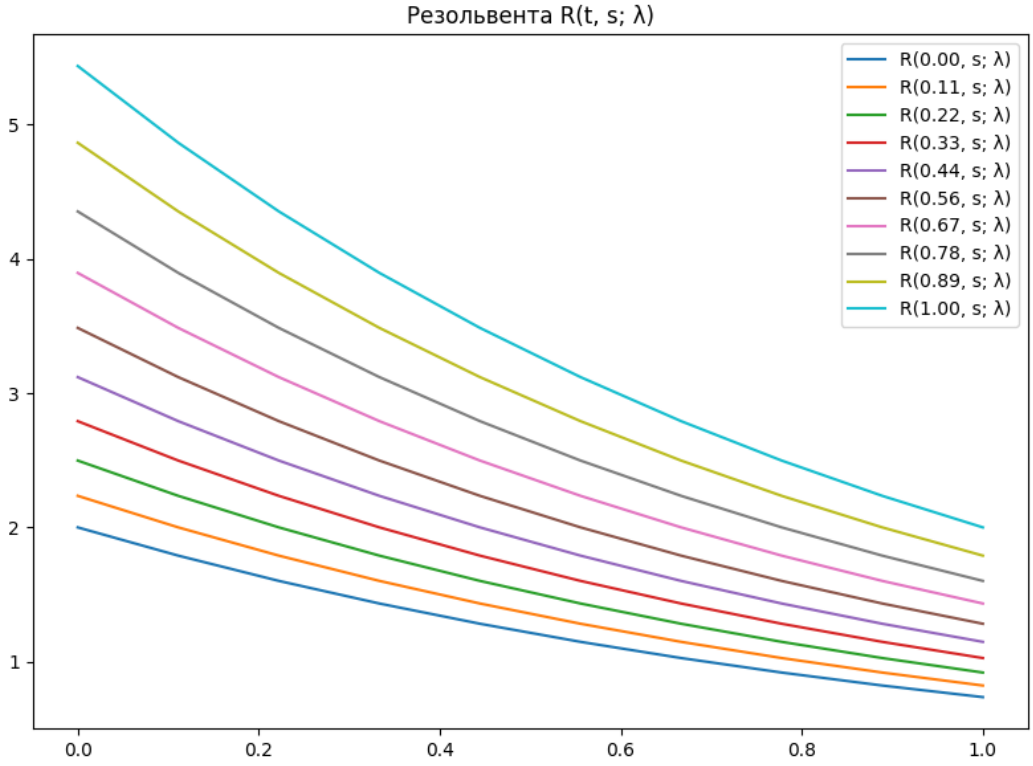

Рисунок 7

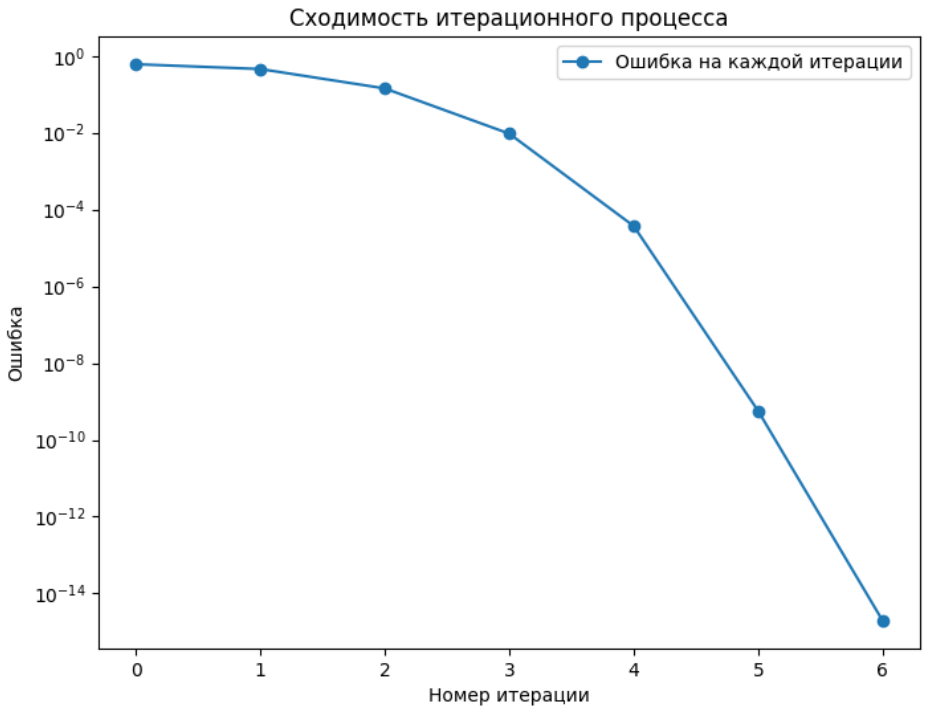

Рисунок 8

Заключение. Таким образом, были рассмотрены основные определения и классификации интегральных уравнений, включая уравнения Фредгольма и Вольтерра, а также уравнения с вырожденным ядром. Это позволило заложить необходимую теоретическую базу для дальнейшего исследования методов их решения. Разработана программная реализация предложенных методов на языке Python, включая создание функций для построения квадратурной сетки и весов, вычисления спектрального радиуса и итерационного процесса для вычисления обратной матрицы.