Министерство образования и науки Российской Федерации
Федеральное государственное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»
Институт ИВТ
Кафедра МКМ

КУРСОВАЯ РАБОТА
по дисциплине «Численные методы»
на тему:
«Решение интегральных уравнений методом квадратур»

Группа: А-05-19
Выполнила: Гусамова Камила Линаровна
Проверила: Амосова Ольга Алексеевна

Москва 2021

Оглавление

1. Задача

1. 1. Постановка задачи

Найти решение интегрального уравнения:

$$u(x) - \lambda \int_a^b K(x,t)u(t)dt = f(x), x \in [a,b]$$

методом квадратур для каждого значения λ из указанного отрезка $[\alpha, \beta]$. При каждом значении λ построить график найденного решения и вычислить площадь полученной криволинейной трапеции. Определить, при каком значении λ, площадь трапеции максимальна.

1.2. Исходные данные

Ядро	Отрезки $[a,b]$ $[\alpha,\beta]$	Метод аппроксимации функции f(x)	Квадратурные формулы	Решение СЛАУ
$\frac{1}{1+e^{\lvert x-t\rvert}}$	[0,3] [1,2]	Метод наименьших квадратов в классе полиномов второй степени	Метод левых прямоугольников	Метод минимальных поправок

Функция f(x)

0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3
8	7.482	7.098	6.813	6.602	6.446	6.331	6.245	6.128	6.124	5.82

2. Теория

2.1. Уравнение Фредгольма второго рода

Уравнения, в которых искомая функция входит под знак интеграла, принято называть интегральными уравнениями.

Уравнение

$$u(x) - \lambda \int_a^b K(x,t)u(t)dt = f(x), x \in [a,b] \quad (1)$$

называется уравнением Фредгольма второго рода.

Функции $K(x,t)$ (*ядро интегрального уравнения*) и $f(x)$ (*правая часть интегрального уравнения*) считаются заданными. λ – числовой параметр.

Под решением интегрального уравнения понимается функция $u(x)$, подстановка которой в уравнение обращает его в тождество.

2.2. Метод квадратур

Одним из наиболее простых и распространенных методов решения интегрального уравнения Фредгольма второго рода является *метод квадратур*, основанный на аппроксимации значений интегрального оператора с использованием одной из квадратурных формул.

Пусть за основу взята квадратурная формула

$$\int_a^b g(t)dt \approx \sum_{j=0}^{N} c_j g(x_j) \quad (2)$$

с узлами x_j (где $a \le x_0 < x_1 < \ldots < x_N \le b$) и весами $c_j > 0$.

Используя формулу (2) с $g(t) = K(x,t)u(t)$ для приближенного вычисления значения интегрального оператора, имеем

$$\int_a^b K(x,t)u(t)dt \approx \sum_{j=0}^{N} c_j K(x,x_j)u(x_j)$$

и от уравнения (1) мы приходим к приближенному равенству

$$u(x) = \lambda \sum_{j=0}^{N} c_j K(x, x_j) u(x_j) + f(x), x \in [a, b]$$

Функцию u^N, удовлетворяющую уравнению

$$u^N(x) = \lambda \sum_{j=0}^{N} c_j K(x, x_j) u^N(x_j) + f(x), x \in [a, b] \quad (3)$$

примем за приближенное решение уравнения (1).

Если уравнение (3) имеет решение, то его можно найти следующим образом. Подставляя в (3) значения $x = x_i$ для $i = 0, \dots, N$, приходим к системе линейных алгебраических уравнений

$$u_i = \lambda \sum_{j=0}^{N} c_j K_{ij} u_j + f_i, i = 0, \dots, N \quad (4)$$

относительно вектора неизвестных $u_N = (u_0, u_1, \dots, u_N)^T$, где $u_j = u^N(x_j)$. Здесь и ниже используются обозначения $K_{ij} = (x_i, x_j)$ и $f_i = f(x_i)$.

Обратим внимание на то, что решение уравнения (3) однозначно определяется своими значениями $u_j = u^N(x_j)$ в узлах $x_j, 0 \le j \le N$. Поэтому уравнение (3) однозначно разрешимо тогда и только тогда, когда однозначно разрешима система (4).

Если ввести матрицу B_N с элементами $b_{ij} = \lambda c_j K_{ij}, 0 \le i, j \le N$ и вектор $f_N = (f_0, f_1, \dots, f_N)^T$, то систему (4) можно записать в матричном виде:

$$u_N = B_N u_N + f_N \quad (5)$$

Эту же систему можно записать в виде

$$A_N u_N = f_N \quad (6)$$

где $A_N = E_N - B_N$, а E_N – единичная матрица.

2.3. Единственность решения

Пусть ядро K удовлетворяет условию

$$\|K\| = \sqrt{\int_a^b\int |K(x,t)|^2 dxdt} < 1$$

Тогда при любой правой части f интегральное уравнение имеет единственное решение, причем справедлива оценка

$$\|u\| \le C_2\|f\| \quad (7)$$

с постоянной $C_2 = \frac{1}{1-\|K\|}$.

2.4. Связь невязки и погрешности

Введем интегральный оператор к:

$$\kappa u(x) = \int_a^b K(x,t)u(t)ds$$

Запишем интегральное уравнение (1) в виде

$$u = \kappa u + f, \quad (8)$$

Пусть u – точное решение интегрального уравнения (8), а $\tilde{u}$ – некоторое приближенное решение того же уравнения. Определим невязку

$$r[\tilde{u}] = \tilde{u} - \kappa\tilde{u} - f, \quad (9)$$

отвечающую этому приближенному решению.

Заметим, что точному решению u отвечает нулевая невязка:

$$r[u] = u - \kappa u - f = 0 \quad (10)$$

Вычитая из равенства (9) равенство (8), убеждаемся в том, что погрешность $u - \tilde{u}$ является решением уравнения того же, вида что и (7), но с невязкой $r[\tilde{u}]$ в роли правой части f:

$$u - \tilde{u} \equiv \kappa(u - \tilde{u}) + r[\tilde{u}]$$

Если справедливо неравенство (7), то погрешность $u - \tilde{u}$ можно оценить через невязку следующим образом:

$$\|u - \tilde{u}\| \le C_2\|r[\tilde{u}]\| \quad (11)$$

Таким образом, если невязка, отвечающая приближенному решению $\tilde{u}$, достаточно мала, то малой будет и погрешность $u - \tilde{u}$.

2.5. Метод наименьших квадратов

Требуется найти многочлен $P_m = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m = \sum_{j=0}^{m} a_j x^j$ заданной степени m такой, чтобы величина среднеквадратичного отклонения

$$\sigma(P_m, f) = \sqrt{\frac{1}{n+1}\sum_{i=0}^{n} (P_m(x_i) - f_i)^2}$$

была минимальной.

Запишем *нормальную систему метода наименьших квадратов*:

$$\sum_{j=0}^{m} a_j \sum_{i=0}^{n} x_i^{k+j} = \sum_{i=0}^{n} f_i x_i^k \,, k = 0,1,\ldots,m$$

Перепишем систему в более удобном для практического использования виде:

$$\begin{cases} s_0 a_0 + s_1 a_1 + s_2 a_2 + \ldots + s_m a_m = b_0 \\ s_1 a_0 + s_2 a_1 + s_3 a_2 + \ldots + s_{m+1} a_m = b_1 \\ \ldots \\ s_m a_0 + s_{m+1} a_1 + s_{m+2} a_2 + \ldots + s_{2m} a_m = b_m \end{cases}$$

где $s_k = \sum_{i=0}^{n} x_i^k$, $b_k = \sum_{i=0}^{n} b_i x_i^k$.

2.6. Формула левых прямоугольников

Элементарная формула

$$I_i = \int_{x_{i-1}}^{x_i} f(x)dx \approx f_{i-1} h$$

Составная формула

$$I = \int_{a}^{b} f(x)dx \approx f_0 h + f_0 h + \ldots + f_{n-1} h = h \sum_{i=1}^{n} f_{i-1}$$

2.7. Метод минимальных поправок

Неявный итерационный метод

$$x^{(k+1)} = x^{(k)} + \tau_{k+1} w^{(k)}, Bw^{(k)} = b - Ax^{(k)}, k = 0,1,\ldots$$

называется *методом минимальных поправок*. В этом методе итерационные параметры вычисляются по формуле

$$\tau_{k+1} = \frac{\left(Aw^{(k)}, w^{(k)}\right)}{\left(B^{-1}Aw^{(k)}, Aw^{(k)}\right)}, k = 0,1,\dots,$$

а очередное значение поправки по формуле

$$w^{(k+1)} = w^{(k)} - \tau_{k+1}B^{-1}Aw^{(k)}$$

3. Алгоритм

3.1. Основные функции

Перейдем к реализации алгоритма и решению поставленной задачи.

Подключим необходимые библиотеки:

```
import numpy as np
from scipy import integrate
import matplotlib.pyplot as plt
```

Напишем функции для аппроксимации методом наименьших квадратов, получения коэффициентов формулы левых прямоугольников и решения СЛАУ методом минимальных поправок:

```
#функция, возвращающая коэффициенты многочлена МНК
def LeastSquaresCoefs(x, y, m):
    a = np.zeros((m + 1, m + 1))
    b = np.zeros(m + 1)
    s = np.zeros(2 * m + 1)

    for i in range(len(x)):
        buffer = y[i]
        for j in range(m + 1):
                b[j] += buffer
                buffer *= x[i]
        buffer = 1
        for j in range(2 * m + 1):
            s[j] += buffer
            buffer *= x[i]

    for i in range(m + 1):
        for j in range(m + 1):
            a[i, j] = s[i + j]

    return np.linalg.solve(a, b)

#функция, возвращающая значение многочлена МНК в точке
def LeastSquares(coef, x0):
    m = len(coef) - 1
    polynom = coef[m]
```

```
    for j in range(m):
        polynom = polynom * x0 + coef[m - j - 1]
    return polynom
```

```
#коэффициенты формулы левых прямоугольников
def LeftRiemannSum(j):
    if j == n - 1:
        return 0
    else:
        return h
```

```
#метод минимальных поправок
def Solve(A, b, eps):
    n = A.shape[0]
    B = np.zeros((n, n))
    for i in range(A.shape[0]):
        B[i][i] = A[i][i]
    x = np.zeros(n)
    fl = True

    while fl:
        r = A @ x - b
        w = np.linalg.inv(B) @ r
        v = np.linalg.inv(B) @ A @ w
        t = np.dot((A @ w), w) / np.dot(v, (A @ w))
        x_prev = x
        x = np.linalg.inv(B) @ (B @ x - t * r)
        fl = np.linalg.norm(x - x_prev) > eps

    return x
```

3.2. Метод и проверка решения

Теперь напишем функцию решения интегрального уравнения:

```
#норма ядра -- используется для проверки единственности решения и
нахождения связи невязки и погрешности
def KNorm(a, b, K):
    return np.sqrt(integrate.dblquad(lambda x, t: np.abs(K(x, t)) ** 2,
a, b, lambda x: a, lambda x: b)[0])

#Евклидова норма вектора
def EuclideanNorm(x):
    s = 0
    n = x.shape[0]
    for i in range(n):
        s += x[i] ** 2
    return np.sqrt(s)
```

```
#решение интегрального уравнения Фредгольма второго рода
```

```
def IntegralEquation(K, a, b, h, param = 1, f = None):
    #составляем матрицы для СЛАУ
    n = int((b - a) / h) + 1
    x = np.linspace(a, b, n)

    #задаем матрицу B
    B = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            B[i][j] = param * LeftRiemannSum(j) * K(x[i], x[j])
    E = np.eye(len(B))

    #вычисляем матрицу A
    A = E - B

    #задаем данные для правой части системы
    x_data = np.array([0, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7,
3])
    y_data = np.array([8, 7.482, 7.098, 6.813, 6.602, 6.446, 6.331,
6.245, 6.128, 6.134, 5.82])

    #составляем вектор правой части:
    #если f != None, то F -- функция, заданная аналитически
    F = np.zeros(n)
    for i in range(n):
        if f is None:
            F[i] = LeastSquares(LeastSquaresCoefs(x_data, y_data, 2),
x[i])
        else:
            F[i] = f(x[i])

    #решаем СЛАУ методом минимальных поправок с заданной точностью
    u = Solve(A, F, eps)

    #вычисляем норму ядра
    k_norm = KNorm(a, b, K)
    #вычисляем коэффициент C2
    C2 = 1 / (1 - k_norm)

    #задаем вектор невязки, интеграл будем вычислять, используя формулу
левых прямоугольников
    r = np.zeros(n)
    for i in range(n):
        r[i] = u[i] - F[i] - param * sum([LeftRiemannSum(j) * K(x[i],
x[j]) * u[j] for j in range(n)])

    #вычислим Евклидову норму для полученного вектора невязки
    res = C2 * EuclideanNorm(r)

    return u, res
```

4. Тестовые примеры

4.1. Тестовый пример 1

$$u(x) - \frac{1}{2}\int_0^1 xsu(t)dt = \frac{5}{6}x$$

Решим уравнение *методом последовательных приближений*:

Если в уравнении числовой параметр λ удовлетворяет условию

$$|\lambda| \le \frac{1}{B}, B^2 = \iint_a^b |K(x,t)|^2\, dxdt,$$

то уравнение имеет единственное решение. В этом случае оно может быть найдено методом последовательных приближений.

Выбирая произвольным образом нулевое приближение $u_0(x)$, можно построить последовательность функций $\{u_n(x)\}$ с помощью рекуррентной формулы

$$u_n(x) = f(x) + \lambda \int_a^b K(x,t)u_{n-1}(t)dt$$

Функции $u_n(x)$ $(n = 1, 2, \dots)$ рассматриваются как приближения к искомому решению уравнения. Данная последовательность $\{u_n(x)\}$ сходится равномерно на [a, b] к точному решению, т.е. $u(x) = \lim\limits_{n\to\infty} u_n(x)$.

В качестве нулевого приближения возьмем свободный член $f(x)$ данного уравнения, т.е. $u_0(x) = f(x) = \frac{5}{6}x$.

Строим последовательность функций $\{u_n(x)\}$:

$$u_1(x) = \frac{5}{6}x + \frac{1}{2}\int_0^1 xt\frac{5}{6}tdt = \frac{5}{6}x + \frac{5}{36}x = \frac{35}{36}x = \frac{5}{6}x\left(1 + \frac{1}{6}\right)$$

$$u_2(x) = \frac{5}{6}x + \frac{1}{2}\int_0^1 xt\frac{35}{36}tdt = \frac{5}{6}x + \frac{35}{216}x = \frac{215}{216}x = \frac{5}{6}x\left(1 + \frac{1}{6} + \frac{1}{36}\right)$$

$$u_3(x) = \frac{5}{6}x + \frac{1}{2}\int_0^1 xt\frac{215}{216}tdt = \frac{5}{6}x + \frac{215}{1296}x = \frac{215}{216}x$$
$$= \frac{5}{6}x\left(1 + \frac{1}{6} + \frac{1}{36} + \frac{1}{216}\right)$$
$$\dots$$
$$u_n(x) = \frac{5}{6}x\left(1 + \frac{1}{6} + \frac{1}{6^2} + \frac{1}{6^3} + \dots + \frac{1}{6^n}\right) = \frac{5}{6}x\sum_{k=0}^{n}\frac{1}{6^k}$$

Отсюда точное решение $u(x)$ определяется как предел

$$u(x) = \lim_{n\to\infty}\frac{5}{6}x\sum_{k=0}^{n}\frac{1}{6^k} = \frac{5}{6}x\frac{6}{5} = x$$

Найдем численное решение тестового примера 1, используя написанные функции. Построим графики решений и абсолютной погрешности, вычислим площади полученных криволинейных трапеций.

```
#тестовый пример 1
a = 0
b = 1
h = 0.01
n = int((b - a) / h) + 1
param = 0.5
eps = 1e-6

def Solution1(x): #точное решение
    return x
def K1(x, t): #ядро
    return x * t
def f1(x): #правая часть
  return (5 / 6) * x

x_data = np.linspace(a, b, n)
y_data, r_data = IntegralEquation(K1, a, b, h, param, f1)
```

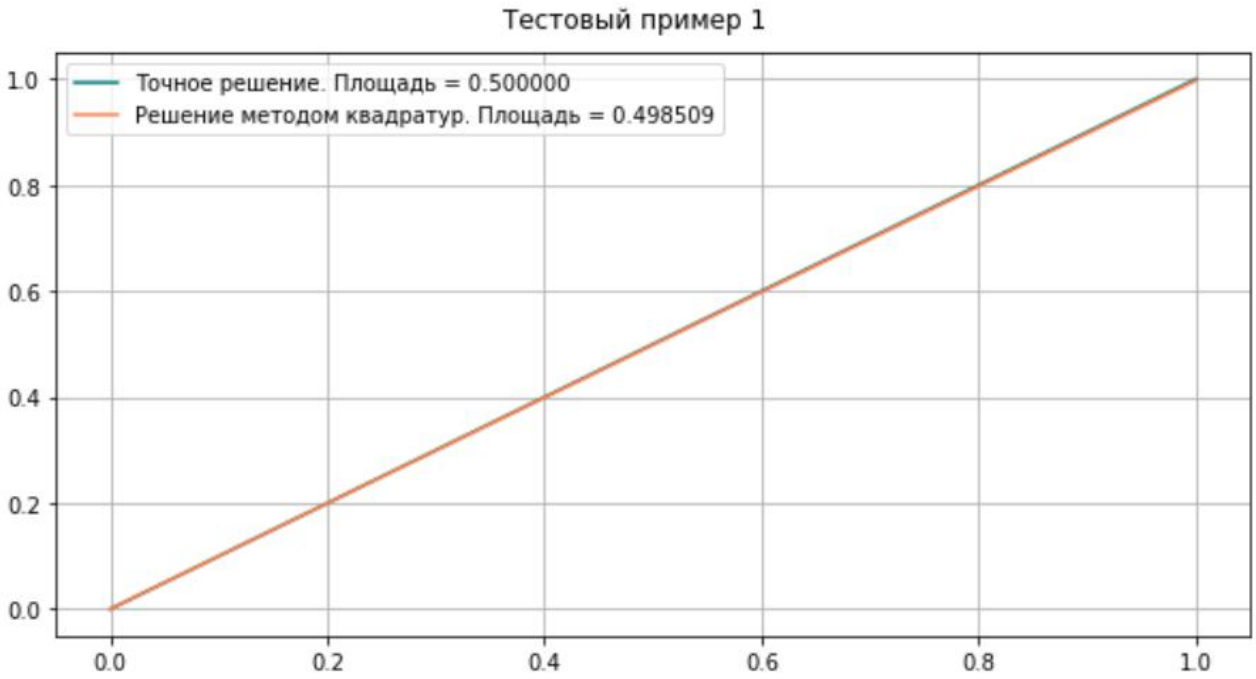

Рис. 1. График решений тестового примера 1

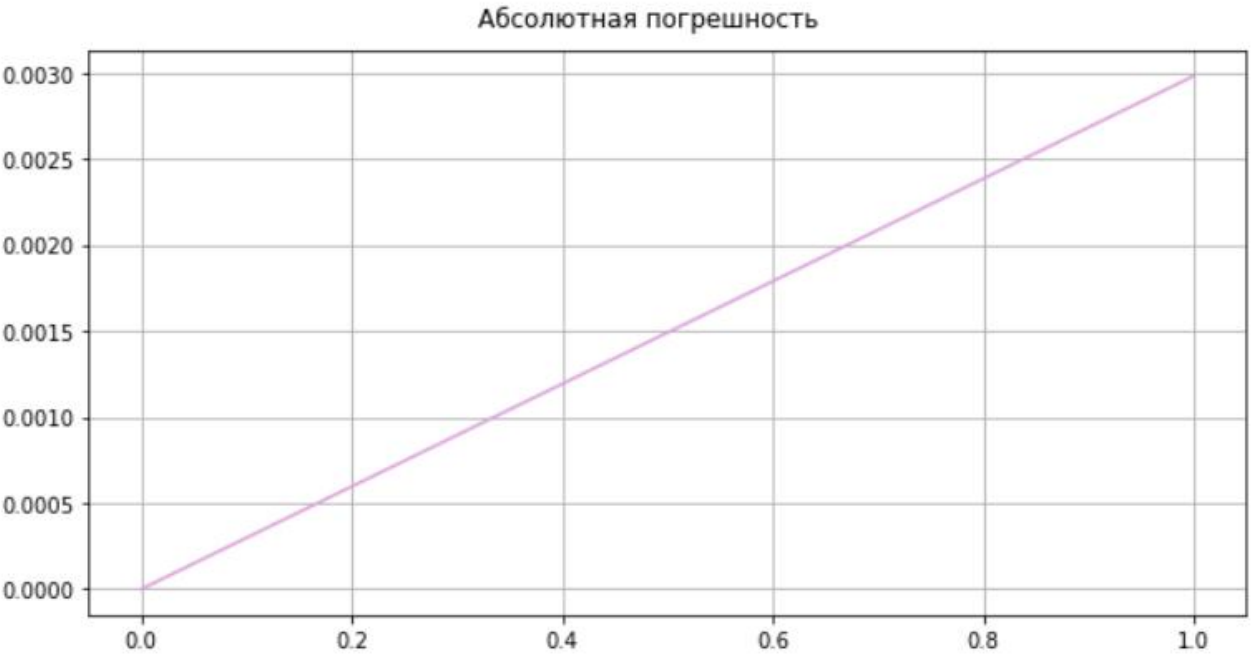

Рис. 2. График абсолютной погрешности приближенного решения тестового примера 1

4.2. Тестовый пример 2

$$u(x) - \frac{1}{2}\int_0^1 u(t)dt = sin(\pi x)$$

Решим уравнение методом последовательных приближений:

В качестве нулевого приближения возьмем свободный член $f(x)$ данного уравнения, т.е. $u_0(x) = f(x) = sin(\pi x)$.

Строим последовательность функций $\{u_n(x)\}$:

$$u_1(x) = sin\pi x + \frac{1}{2}\int_0^1 sin(\pi t)dt = sin(\pi x) + \frac{1}{\pi}$$

$$u_2(x) = sin\pi x + \frac{1}{2}\int_0^1 \left(sin(\pi t) + \frac{1}{\pi}\right)dt = sin(\pi x) + \frac{1}{\pi} + \frac{1}{2\pi}$$

$$u_3(x) = sin\pi x + \frac{1}{2}\int_0^1 \left(sin(\pi x) + \frac{1}{\pi} + \frac{1}{2\pi}\right)dt = sin(\pi x) + \frac{1}{\pi} + \frac{1}{2\pi} + \frac{1}{4\pi}$$

…

$$u_n(x) = sin(\pi x) + \frac{1}{\pi} + \frac{1}{2\pi} + \frac{1}{4\pi} + \ldots + \frac{1}{2^{n-1}\pi} = sin(\pi x) + \frac{1}{\pi}\sum_{k=0}^{n-1}\frac{1}{2^k}$$

Отсюда точное решение $u(x)$ определяется как предел

$$u(x) = \lim_{n\to\infty}\left(sin(\pi x) + \frac{1}{\pi}\sum_{k=0}^{n-1}\frac{1}{2^k}\right) = sin(\pi x) + \frac{2}{\pi}$$

Найдем численное решение тестового примера 2, используя написанные функции. Построим графики решений и абсолютной погрешности, вычислим площади полученных криволинейных трапеций.

```
#тестовый пример 2
a = 0
b = 1
h = 0.01
n = int((b - a) / h) + 1
param = 0.5
eps = 1e-6

def Solution2(x): #точное решение
    return np.sin(np.math.pi * x) + 2 / (np.math.pi)
```

```
def K2(x, t): #ядро
    return 1
def f2(x): #правая часть
    return np.sin(np.math.pi * x)

x_data = np.linspace(a, b, n)
y_data, r_data = IntegralEquation(K2, a, b, h, param, f2)
```

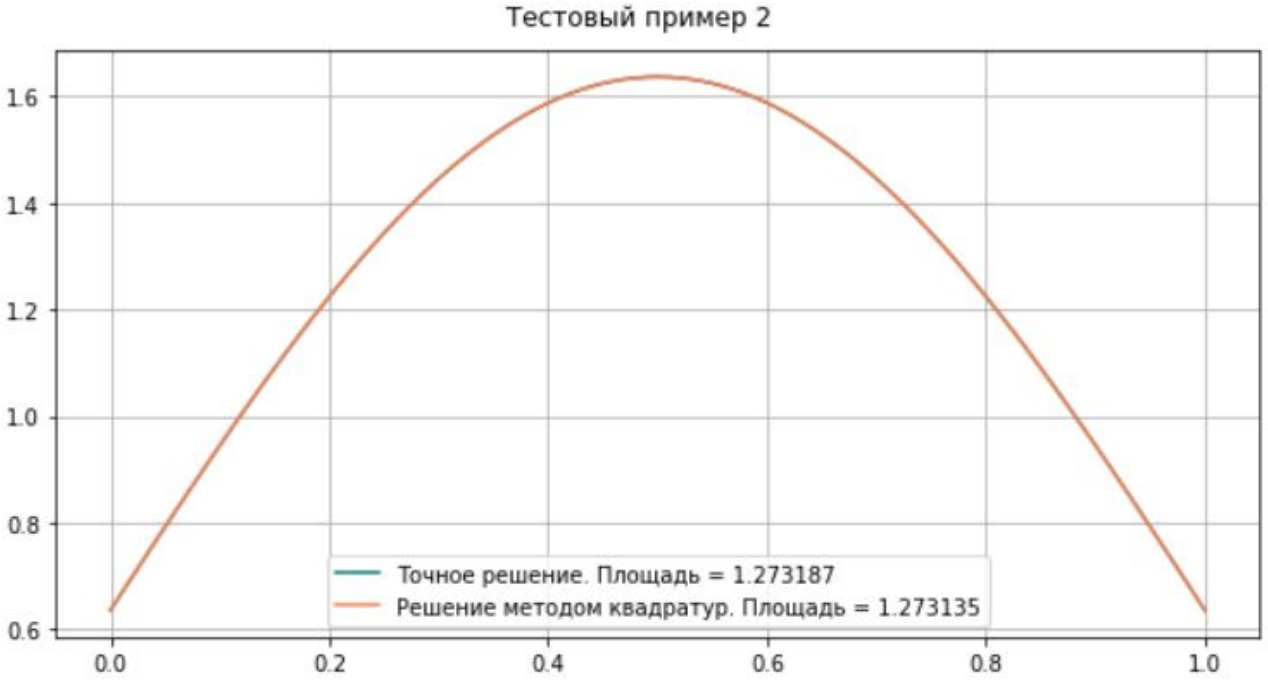

Рис. 3. График решений тестового примера 1

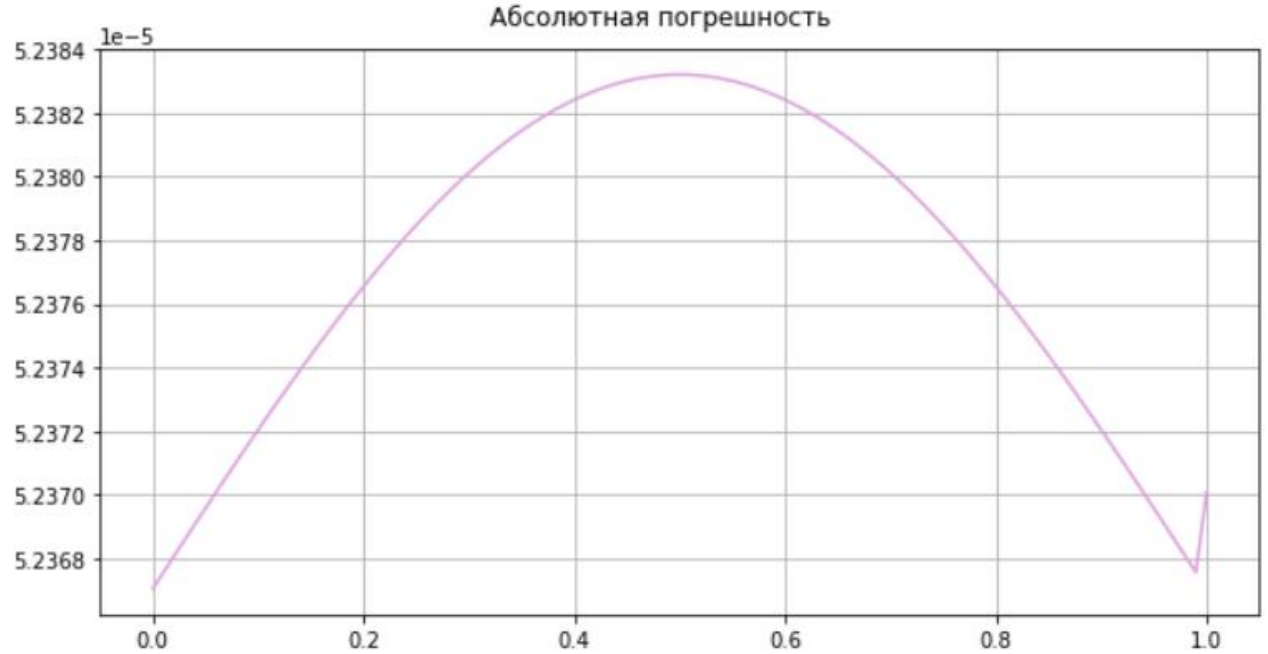

Рис. 4. График абсолютной погрешности приближенного решения тестового примера 1

Графики решений обоих тестовых примеров совпадают и накладываются друг на друга, площади трапеций примерно одинаковы. Абсолютные погрешности малы. Это свидетельствует о верной работе алгоритма – можно перейти к численному решению поставленной задачи.

5. Решение задачи

Теперь решим исходную задачу

$$u(x) - \lambda \int_0^3 \frac{1}{1 + e^{|x-t|}} u(t)dt = f(x), x \in [0, 3], \lambda \in [1, 2]$$

Функция f(x)

0	0.3	0.6	0.9	1.2	1.5	1.8	2.1	2.4	2.7	3
8	7.482	7.098	6.813	6.602	6.446	6.331	6.245	6.128	6.124	5.82

```
#исходные данные
a = 0
b = 3
h = 0.01
n = int((b - a) / h) + 1
eps = 1e-6
alpha = 1
beta = 2

def K(x, t): #ядро
    return 1 / (1 + np.exp(abs(x - t)))

x_data = np.linspace(a, b, n)
y_data, r_data = IntegralEquation(K, a, b, h)

print("Оценка погрешности: ||u - u*|| <=", r_data)
```

```
Оценка погрешности: ||u - u*|| <= 4.4088042281908743e-07
```

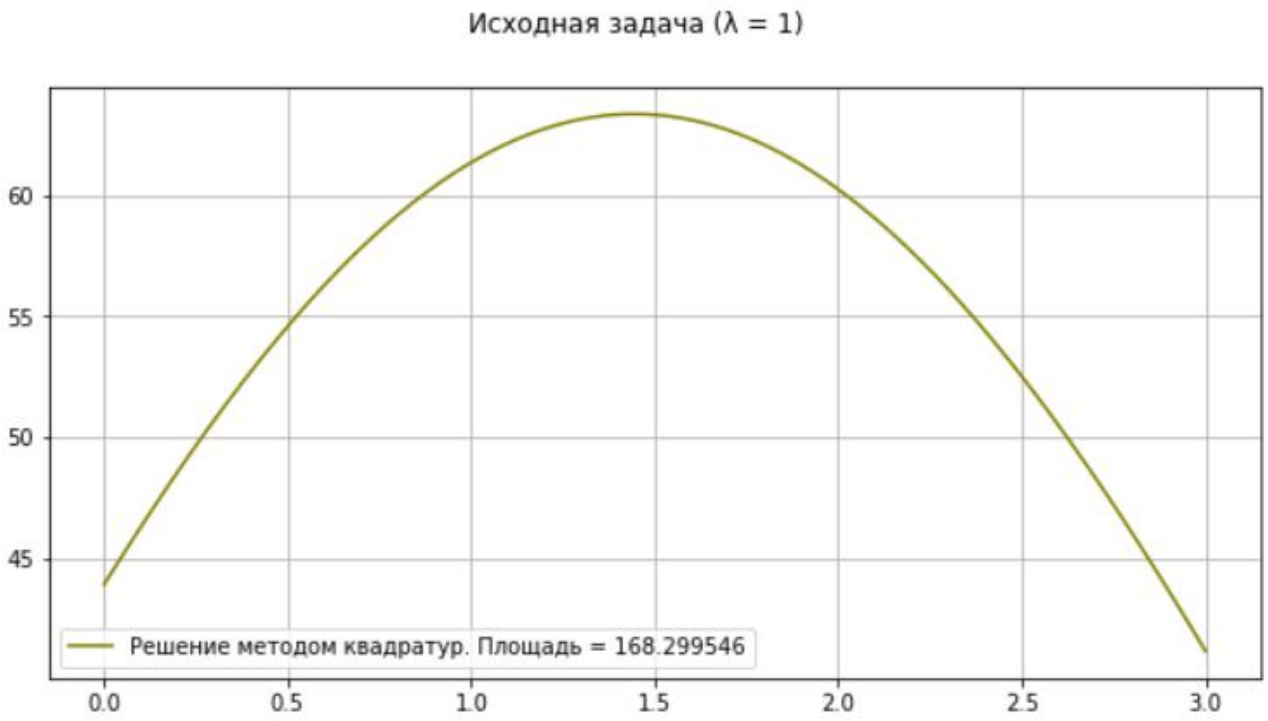

Рис. 5. График решения задачи при $\lambda = 1$

Оценка погрешности численного решения задачи при $\lambda = 1$:

$$\|u - \tilde{u}\| \leq 4.4088 \cdot 10^{-7}$$

Можно сделать вывод о том, что погрешность численного решения мала.

Решим задачу при разных значениях λ

```
params = np.linspace(alpha, beta, 6) #1 1.2 1.4 1.6 1.8 2
for i in range(params.shape[0]):
    for j in range(params.shape[1]):
        y_data, r_data = IntegralEquation(K, a, b, h, params[i][j])

print(i, ",", j, "Оценка погрешности: ||u - u*|| <=", r_data)
```

```
0, 0 Оценка погрешности: ||u - u*|| <= 4.4088042281908743e-07
0, 1 Оценка погрешности: ||u - u*|| <= 38.621656241177085
1, 0 Оценка погрешности: ||u - u*|| <= 4.8685081457552765e-05
1, 1 Оценка погрешности: ||u - u*|| <= 1.7189690566186077e-05
2, 0 Оценка погрешности: ||u - u*|| <= 5.150020853684956e-05
2, 1 Оценка погрешности: ||u - u*|| <= 0.000863423832123382
```

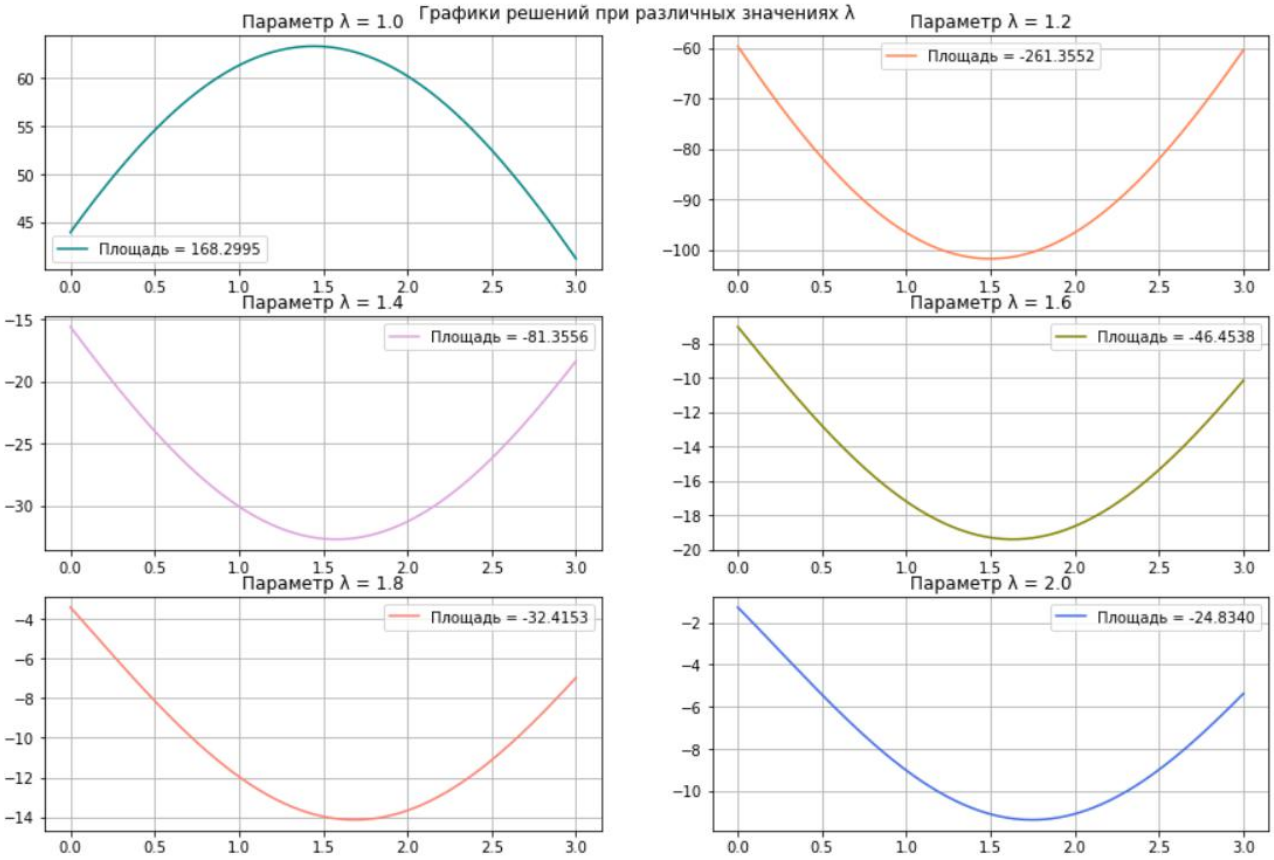

Рис. 6. Графики решений задачи при разных значениях λ

Оценка погрешности численного решения задачи при разных значениях λ:

$$\lambda = 1: \|u - \tilde{u}\| \leq 4.4088 \cdot 10^{-7}$$
$$\lambda = 1.2: \|u - \tilde{u}\| \leq 38.6217$$
$$\lambda = 1.4: \|u - \tilde{u}\| \leq 4.8685 \cdot 10^{-5}$$
$$\lambda = 1.6: \|u - \tilde{u}\| \leq 1.7190 \cdot 10^{-5}$$
$$\lambda = 1.8: \|u - \tilde{u}\| \leq 5.1500 \cdot 10^{-5}$$
$$\lambda = 2: \|u - \tilde{u}\| \leq 0.0008634$$

Площадь трапеции максимальна при значении $\lambda = 1$.

6. Вывод

Используемые методы и корректно построенный алгоритм позволили решить задачу с хорошей точностью, о чем свидетельствует оценка погрешностей полученных численный решений, а также проведенные вычислительные эксперименты.

Список литературы

1. Амосова О.А. Лекции по численным методам. – М., 2020-2021.

2. Амосов А.А. Вычислительные методы [Текст]: Учебное пособие / А.А. Амосов, Ю.А. Дубинский, Н.В. Копченова – Изд. 2-е, стер. – М.: Лань, 2014.

3. Попов В.А. Сборник задач по интегральным уравнениям. – Казань, 2006.

Листинг программы

```
import numpy as np
from scipy import integrate
import matplotlib.pyplot as plt

#функция, возвращающая коэффициенты многочлена МНК
def LeastSquaresCoefs(x, y, m):
    a = np.zeros((m + 1, m + 1))
    b = np.zeros(m + 1)
    s = np.zeros(2 * m + 1)

    for i in range(len(x)):
        buffer = y[i]
        for j in range(m + 1):
                b[j] += buffer
                buffer *= x[i]
        buffer = 1
        for j in range(2 * m + 1):
            s[j] += buffer
            buffer *= x[i]

    for i in range(m + 1):
        for j in range(m + 1):
            a[i, j] = s[i + j]

    return np.linalg.solve(a, b)

#функция, возвращающая значение многочлена МНК в точке
def LeastSquares(coef, x0):
    m = len(coef) - 1
    polynom = coef[m]
    for j in range(m):
        polynom = polynom * x0 + coef[m - j - 1]
    return polynom

#метод минимальных поправок
def Solve(A, b, eps):
    n = A.shape[0]
    B = np.zeros((n, n))
    for i in range(n):
```

```
            B[i][i] = A[i][i]
        x = np.zeros(n)
        fl = True

        while fl:
            r = A @ x - b
            w = np.linalg.inv(B) @ r
            v = np.linalg.inv(B) @ A @ w
            t = np.dot((A @ w), w) / np.dot(v, (A @ w))
            x_prev = x
            x = np.linalg.inv(B) @ (B @ x - t * r)
            fl = np.linalg.norm(x - x_prev) > eps

        return x

    #коэффициенты формулы левых прямоугольников
    def LeftRiemannSum(j):
        if j == n - 1:
            return 0
        else:
            return h

    #норма ядра -- используется для проверки единственности решения и
    нахождения
    #связи невязки и погрешности
    def KNorm(a, b, K):
        return np.sqrt(integrate.dblquad(lambda x, t: np.abs(K(x, t)) ** 2, a,
    b, lambda x: a, lambda x: b)[0])

    #Евклидова норма вектора
    def EuclideanNorm(x):
        s = 0
        n = x.shape[0]
        for i in range(n):
            s += x[i] ** 2
        return np.sqrt(s)

    #решение интегрального уравнения Фредгольма второго рода
    def IntegralEquation(K, a, b, h, param = 1, f = None):
        #составляем матрицы для СЛАУ
        n = int((b - a) / h) + 1
        x = np.linspace(a, b, n)

        #задаем матрицу B
        B = np.zeros((n, n))
        for i in range(n):
            for j in range(n):
                B[i][j] = param * LeftRiemannSum(j) * K(x[i], x[j])
        E = np.eye(len(B))

        #вычисляем матрицу A
```

```
    A = E - B

    #задаем данные для правой части системы
    x_data = np.array([0, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3])
    y_data = np.array([8, 7.482, 7.098, 6.813, 6.602, 6.446, 6.331, 6.245,
6.128, 6.134, 5.82])

    #составляем вектор правой части:
    #если f != None, то F -- функция, заданная аналитически
    F = np.zeros(n)
    for i in range(n):
        if f is None:
            F[i] = LeastSquares(LeastSquaresCoefs(x_data, y_data, 2),
x[i])
        else:
            F[i] = f(x[i])

    #решаем СЛАУ методом минимальных поправок с заданной точностью
    u = Solve(A, F, eps)
    #u = np.linalg.solve(A, F)

    #вычисляем норму ядра
    k_norm = KNorm(a, b, K)
    #вычисляем коэффициент C2
    C2 = 1 / (1 - k_norm)

    #задаем вектор невязки, интеграл будем вычислять, используя формулу
    #левых прямоугольников
    r = np.zeros(n)
    for i in range(n):
        r[i] = u[i] - F[i] - param * sum([LeftRiemannSum(j) * K(x[i],
x[j]) * u[j] for j in range(n)])

    #вычислим Евклидову норму для полученного вектора невязки
    res = C2 * EuclideanNorm(r)

 return u, res

 #тестовый пример 1
 a = 0
 b = 1
 h = 0.01
 n = int((b - a) / h) + 1
 param = 0.5
 eps = 1e-6

 def Solution1(x): #точное решение
     return x
 def K1(x, t): #ядро
     return x * t
 def f1(x): #правая часть
```

```
    return (5 / 6) * x

x_data = np.linspace(a, b, n)
y_data, r_data = IntegralEquation(K1, a, b, h, param, f1)

fig, axs = plt.subplots(1, 1, figsize=(10,5))
axs.plot(x_data, Solution1(x_data), color='teal', label=f'Точное
решение. Площадь = {np.trapz(Solution1(x_data), x_data):.6f}')
axs.plot(x_data, y_data, color='coral', label=f'Решение методом
квадратур. Площадь = {np.trapz(y_data, x_data):.6f}')
plt.suptitle('Тестовый пример 1', y=0.94)
plt.legend()
plt.grid()

fig, axs = plt.subplots(1, 1, figsize=(10,5))
axs.plot(x_data, np.abs(Solution1(x_data) - y_data), color='plum')
plt.suptitle('Абсолютная погрешность', y=0.94)
plt.grid()

#тестовый пример 2
a = 0
b = 1
h = 0.01
n = int((b - a) / h) + 1
param = 0.5
eps = 1e-6

def Solution2(x): #точное решение
    return np.sin(np.math.pi * x) + 2 / (np.math.pi)
def K2(x, t): #ядро
    return 1
def f2(x): #правая часть
    return np.sin(np.math.pi * x)

x_data = np.linspace(a, b, n)
y_data, r_data = IntegralEquation(K2, a, b, h, param, f2)

fig, axs = plt.subplots(1, 1, figsize=(10,5))
axs.plot(x_data, Solution2(x_data), color='teal', label=f'Точное
решение. Площадь = {np.trapz(Solution2(x_data), x_data):.6f}')
axs.plot(x_data, y_data, color='coral', label=f'Решение методом
квадратур. Площадь = {np.trapz(y_data, x_data):.6f}')
plt.suptitle('Тестовый пример 2', y=0.94)
plt.legend()
plt.grid()

fig, axs = plt.subplots(1, 1, figsize=(10,5))
axs.plot(x_data, np.abs(Solution2(x_data) - y_data), color='plum')
plt.suptitle('Абсолютная погрешность', y=0.94)
plt.grid()
```

```
 #исходные данные
 a = 0
 b = 3
 h = 0.01
 n = int((b - a) / h) + 1
 eps = 1e-6
 alpha = 1
 beta = 2

 def K(x, t): #ядро
     return 1 / (1 + np.exp(abs(x - t)))

 x_data = np.linspace(a, b, n)
 y_data, r_data = IntegralEquation(K, a, b, h)

 fig, axs = plt.subplots(1, 1, figsize=(10,5))
 area = np.trapz(y_data, x_data)
 axs.plot(x_data, y_data, color='olive', label=f'Решение методом
квадратур. Площадь = {np.trapz(y_data, x_data):.6f}')
 plt.suptitle('Исходная задача (λ = 1)', y=0.98)
 plt.legend()
 plt.grid()

 print("Оценка погрешности: ||u - u*|| <=", r_data)

 params = np.linspace(alpha, beta, 6) #1 1.2 1.4 1.6 1.8 2

 colors = ('teal', 'coral', 'plum', 'olive', 'salmon', 'royalblue',
'pink', 'grey', 'darkorchid', 'turquoise')

 params = np.reshape(params, (-1, 2))
 colors = np.reshape(colors, (-1, 2))
 areas = []

 fig, axs = plt.subplots(3, 2, figsize=(15,10))
 for i in range(params.shape[0]):
     for j in range(params.shape[1]):
         y_data, r_data = IntegralEquation(K, a, b, h, params[i][j])
         areas.append(np.trapz(y_data, x_data))
         axs[i][j].plot(x_data, y_data, color=colors[i][j],
label=f'Площадь = {areas[-1]:.4f}')
         axs[i][j].set_title(f'Параметр λ = {params[i][j]}')
         axs[i][j].legend()
         axs[i][j].grid()

         print(i, ",", j, "Оценка погрешности: ||u - u*|| <=", r_data)

 plt.suptitle('Графики решений при различных значениях λ', y=0.91)
```