Московский авиационный институт

(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Курсовая работа по дисциплине “Численные методы”

Вариант 15. Численное решение интегральных уравнений Фредгольма 2-го рода.

Студент: Блинов М.А.

Группа: 8О-408Б

Преподаватель: Ревизников Д. Л.

Пивоваров Д. Е.

Дата:

Оценка:

Москва, 2024

СОДЕРЖАНИЕ:

1. ЗАДАНИЕ

Постановка задачи: Написать программу, которая для введенной функции решает интегральное уравнение Фредгольма 2-го рода.

2. МЕТОД РЕШЕНИЯ

Линейное уравнение Фредгольма II рода имеет следующий вид:

$$y(x) - \lambda \int_a^b K(x,s)y(s)ds = f(x), \quad x \in [a,b]$$

Здесь $y(x)$ — неизвестная функция, $K(x,s)$ — ядро интегрального уравнения, $f(x)$ — свободный член (правая часть) интегрального уравнения. Для удобства анализа в интегральном уравнении по традиции принято выделять числовой параметр λ, который называют параметром интегрального уравнения.

Найдем приближенное решение уравнения методом квадратур. Построим на отрезке $[a,b]$ сетку с узлами $x_1, x_2, ..., x_n$.

Запишем уравнение (1) в узлах сетки:

$$y\left(x_i\right) - \lambda \int_a^b K\left(x_i,s\right) y(s)ds = f\left(x_i\right), \quad i = 1,2,\ldots,n$$

Аппроксимируем интегралы в равенствах (2) конечными суммами с помощью одной из квадратурных формул:

$$y_i - \lambda \sum_{j=1}^n A_j K_{ij} y_j = f_i, \quad i = 1,2,\ldots,n$$

Здесь $y_i = \tilde{y}\left(x_i\right)$, $f_i = f(x_i)$, $K_{ij} = K(x_i,x_j)$,

$\tilde{y}$ — приближение к искомой функции y, A_j — веса квадратурной формулы.

Решение системы уравнений дает приближенные значения искомой функции в узлах x_i . По ним с помощью интерполяции можно построить приближенное решение интегрального уравнения на всем отрезке $[a, b]$.

Пусть $\lambda = 1$, а сетка $x_1, x_2, ..., x_n$ — равномерная с шагом h. Используем квадратурную формулу трапеций. Тогда система линейных алгебраических уравнений (3) примет следующий вид:где $w_1 = w_n = 1/2$, $w_j = 1$ при $j = 2, 3, ..., n - 1$.

3. РЕЗУЛЬТАТЫ

```
# Параметры
h = np.pi / 10
a = -np.pi
b = np.pi
_lambda = 3 / (10 * np.pi)

# Ядро уравнения
def K(x1, s):
    return 1 / (0.64 * (np.cos((x1 + s) / 2)) ** 2 - 1) * _lambda

# Правая часть уравнения
def f(x1):
    return 25 - 16 * (np.sin(x1)) ** 2

# Точное решение
def y_exact(x1):
    return 17 / 2 + 128 / 17 * np.cos(2 * x1)
```

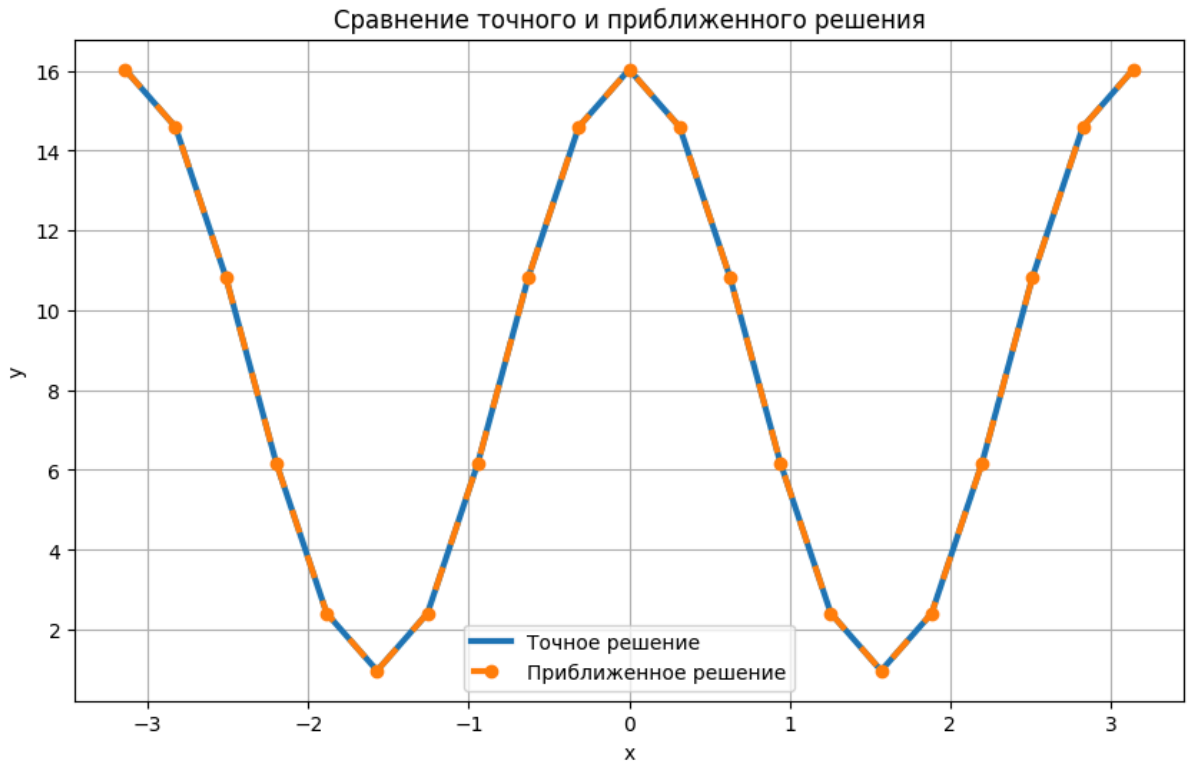

Рисунок 1 - График, отображающий точные и численные значения решений на интервалах x для примера 1

```
(venv) ➜  cw git:(blinov-cw) ✗ python main.py
Точное решение [16.02941176 14.59142208 10.82671619  6.17328381  2.40857792  0.97058824
  2.40857792  6.17328381 10.82671619 14.59142208 16.02941176 14.59142208
 10.82671619  6.17328381  2.40857792  0.97058824  2.40857792  6.17328381
 10.82671619 14.59142208 16.02941176]
Метод квадратур [16.02941186 14.59142217 10.82671629  6.1732839   2.40857802  0.97058833
  2.40857802  6.1732839  10.82671629 14.59142217 16.02941185 14.59142217
 10.82671629  6.1732839   2.40857802  0.97058833  2.40857802  6.1732839
 10.82671629 14.59142217 16.02941186]
```

Рисунок 2 - консольное взаимодействие программы с пользователем

	x	y_exact	y_approx
0	-3.141593	16.029412	16.029412
1	-2.827433	14.591422	14.591422
2	-2.513274	10.826716	10.826716
3	-2.199115	6.173284	6.173284
4	-1.884956	2.408578	2.408578
5	-1.570796	0.970588	0.970588
6	-1.256637	2.408578	2.408578
7	-0.942478	6.173284	6.173284
8	-0.628319	10.826716	10.826716
9	-0.314159	14.591422	14.591422
10	0.000000	16.029412	16.029412
11	0.314159	14.591422	14.591422
12	0.628319	10.826716	10.826716
13	0.942478	6.173284	6.173284
14	1.256637	2.408578	2.408578
15	1.570796	0.970588	0.970588
16	1.884956	2.408578	2.408578
17	2.199115	6.173284	6.173284
18	2.513274	10.826716	10.826716
19	2.827433	14.591422	14.591422
20	3.141593	16.029412	16.029412

Таблица 1 - сравнение точного решения и метода квадратур

4. ИСХОДНЫЙ КОД

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

def fredholm_method(K, f, a, b, h, iterations=1000, tol=1e-6):

    # Разделяем область интегрировани [a, b] на h частей
    x = np.arange(a, b + h, h)

    # Определяем количество отрезков
    n = len(x)

    # Создаем матрицу размера n x n, состоящую из 0
    A = np.zeros((n, n))
    for i in range(n):
        A[i, 0] = -h * 0.5 * K(x[i], x[0])
        for j in range(1, n-1):
            A[i, j] = -h * K(x[i], x[j])
        A[i, n-1] = -h * 0.5 * K(x[i], x[n-1])
        A[i, i] += 1

    B = np.array([f(xi) for xi in x]).reshape(-1, 1)

    # Начальное приближение
    x_0 = np.zeros_like(B)

    # Метод простых итераций
    x_new = x_0.copy()
    for k in range(iterations):
        for i in range(n):
            # Вычисляем новое значение для x_i
            x_new[i] = (B[i] - np.dot(A[i], x_0) + A[i, i] * x_0[i]) /
A[i, i]

        # Проверяем условие сходимости
        if np.linalg.norm(x_new - x_0) < tol:
            return x_new

        # Обновляем предыдущее приближение
        x_0 = x_new.copy()

    return x_new


# Параметры
```

```
h = np.pi / 10
a = -np.pi
b = np.pi
_lambda = 3 / (10 * np.pi)

# Ядро уравнения
def K(x1, s):
    return 1 / (0.64 * (np.cos((x1 + s) / 2)) ** 2 - 1) * _lambda

# Правая часть уравнения
def f(x1):
    return 25 - 16 * (np.sin(x1)) ** 2

# Точное решение
def y_exact(x1):
    return 17 / 2 + 128 / 17 * np.cos(2 * x1)

x_values = np.arange(a, b + h, h)

# Решение уравнения
y_approx = fredholm_method(K, f, a, b, h)
exact = y_exact(x_values)


print("Точное решение", y_exact(np.arange(a, b + h, h)))
print("Метод квадратур", y_approx.flatten())
```

5. ПРОИЗВОДИТЕЛЬНОСТЬ

В данном разделе я исследовал время работы программы при изменении точности вычислений (параметр h).

Интервалы	10	100	1000	10000
Время, ms	1.37	82.4	244.5	1532.43

6. ВЫВОД

В результате работы над курсовой работой мною были изучены методы реализации интегральных уравнений Фредгольма 2-го рода. Из всех методов мне приглянулся метод квадратур: его простоты и следствию из определения самого интеграла; возможность регулировать временные затраты.