Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Курсовой проект

по курсу «Численные методы»

Студент: Ветренко П. С.
Группа: 80-308б
Преподаватель: Иванов И.Э.
Оценка:

Москва, 2020

1.Постановка задачи:

Разработать программу, осуществляющую численное решение интегральных уравнений Фредгольма второго рода.

2. Теория:

Определение: Интегральное уравнение – уравнение, содержащее искомую функцию под знаком интеграла.

Определение: Интегральное уравнение называется линейным, если оно имеет вид:

$$A(x)f(x) + \int_a^b K(x,\ t)f(t)dt\ = g(x)$$

A(x) – коэффициент, *K(x, t)* – ядро, *g(x)* – свободный член, *f* – искомая функция

Определение: Уравнение Фредгольма II рода – линейное интегральное уравнение, имеющее вид:

$$f(x) - \lambda \int_a^b K(x,\ t)f(t)dt\ = g(x)$$

3. Описание метода решения

Уравнения Фредгольма 2 рода можно решить численно, используя метод квадратур. Для начала построим на отрезке $[a,\ b]$ сетку с узлами $x_0,\ x_1,\ \ldots,\ x_{n-1}$. Для каждого узла сетки запишем уравнение, и в итоге получим систему линейных уравнений:

$$\begin{cases} f(x_0) - \lambda \int_a^b K\left(x_0,\ t\right)f(t)dt\ = g(x_0) \\ \cdots \\ f(x_{n-1}) - \lambda \int_a^b K\left(x_{n-1},\ t\right)f(t)dt\ = g(x_{n-1}) \end{cases}$$

Аппроксимируем интегралы, используя квадратурную формулу трапеций:

$$\int_a^b f(x)dx \approx \frac{1}{2}h\sum_{i=1}^{n}(f_i + f_{i-1}) = h\left(\frac{f(x_0)+f(x_1)}{2} + \ldots + \frac{f(x_{n-1})+f(x_n)}{2}\right) = \frac{h}{2}(f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)$$

), где h – постоянный шаг сетки

После преобразования система будет иметь вид:

$$\begin{cases} f(x_0) - \lambda h \sum_{j=0}^{n-1} w_j K(x_0, x_j) f(x_j) = g(x_0) \\ \cdots \\ f(x_{n-1}) - \lambda h \sum_{j=0}^{n-1} w_j K(x_{n-1}, x_j) f(x_j) = g(x_{n-1}) \end{cases} \quad \begin{cases} w_j = 0.5, \ j = 0, j = n-1 \\ w_j = 1, \ 0 < j < n-1 \end{cases}$$

Запишем матрицу данной системы относительно неизвестных f(x):

$$A = \begin{pmatrix} 1 - \lambda \frac{h}{2} K(x_0, x_0) & \cdots & -\lambda \frac{h}{2} K(x_0, x_{n-1}) \\ \vdots & \ddots & \vdots \\ -\lambda \frac{h}{2} K(x_{n-1}, x_0) & \cdots & 1 - \lambda \frac{h}{2} K(x_{n-1}, x_{n-1}) \end{pmatrix} = \begin{pmatrix} g(x_0) \\ \cdots \\ g(x_{n-1}) \end{pmatrix}$$

Решив систему любым численным методом, получим функцию f(x), заданную таблично в узлах сетки.

Для уравнения Фредгольма первого рода аналогичным образом можно получить систему:

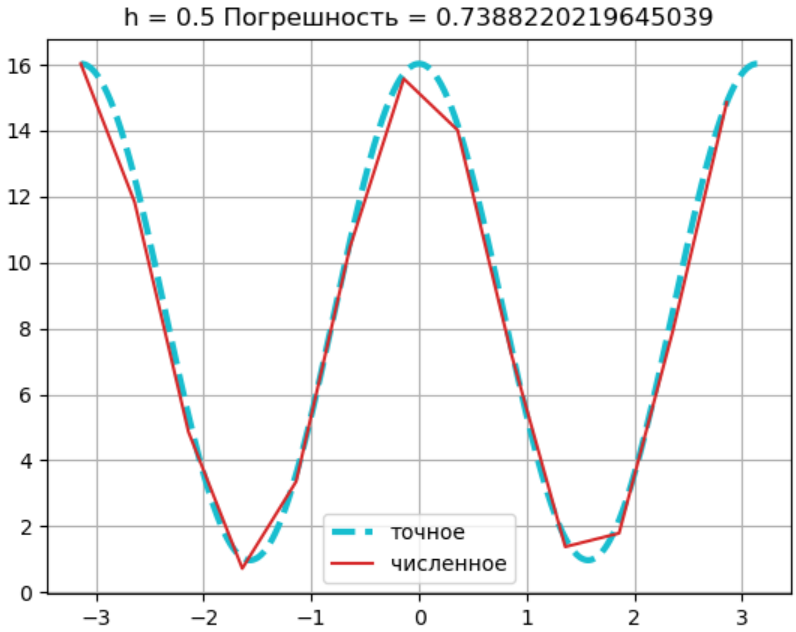
$$A = \begin{pmatrix} \frac{h}{2} K(x_0, x_0) & \cdots & \frac{h}{2} K(x_0, x_{n-1}) \\ \vdots & \ddots & \vdots \\ \frac{h}{2} K(x_{n-1}, x_0) & \cdots & \frac{h}{2} K(x_{n-1}, x_{n-1}) \end{pmatrix} = \begin{pmatrix} g(x_0) \\ \cdots \\ g(x_{n-1}) \end{pmatrix}$$

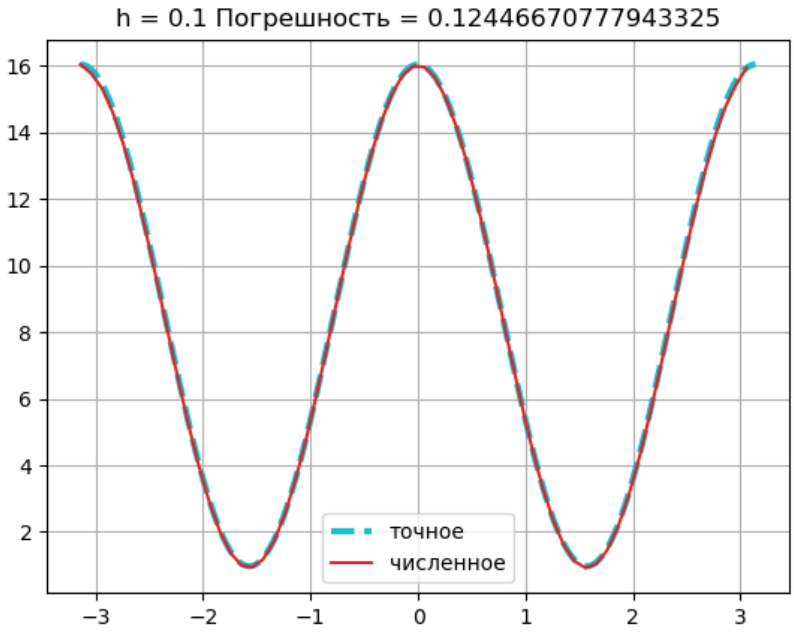
4. Тестовые примеры и результаты:

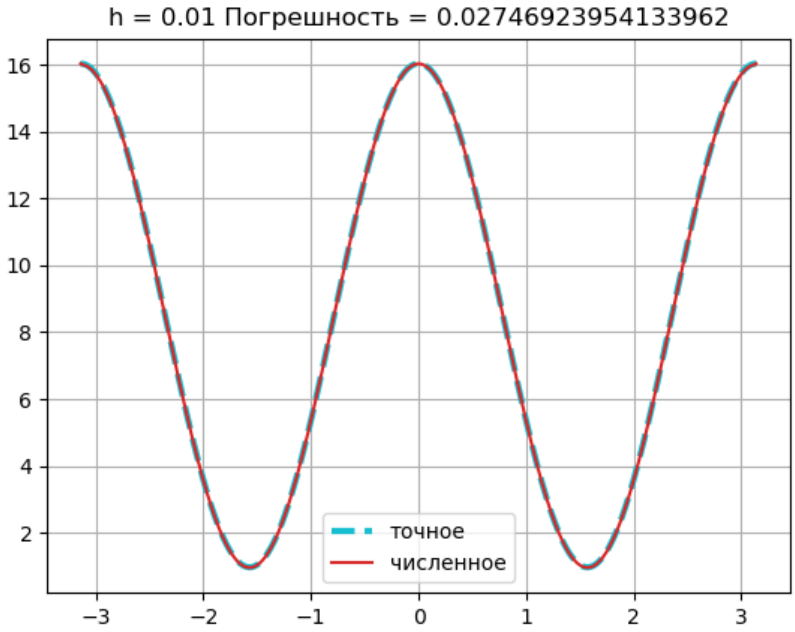
Для расчета погрешности я применяла формулу $\max_{0 \le i \le n-1} \left| f(x_i) - y_i \right|$, где f(x) – точное решение в точке x, а y – численное.

$f(x) - \frac{3}{10}\pi \int_{-\pi}^{\pi} \frac{1}{0.64 cos^2\left(\frac{x+t}{2}\right) - 1} f(t) = 25 - 16 sin^2(x)$ – уравнение Фредгольма 2 рода

$f(x) = \frac{17}{2} + \frac{128}{17}\cos(2x)$ - решение

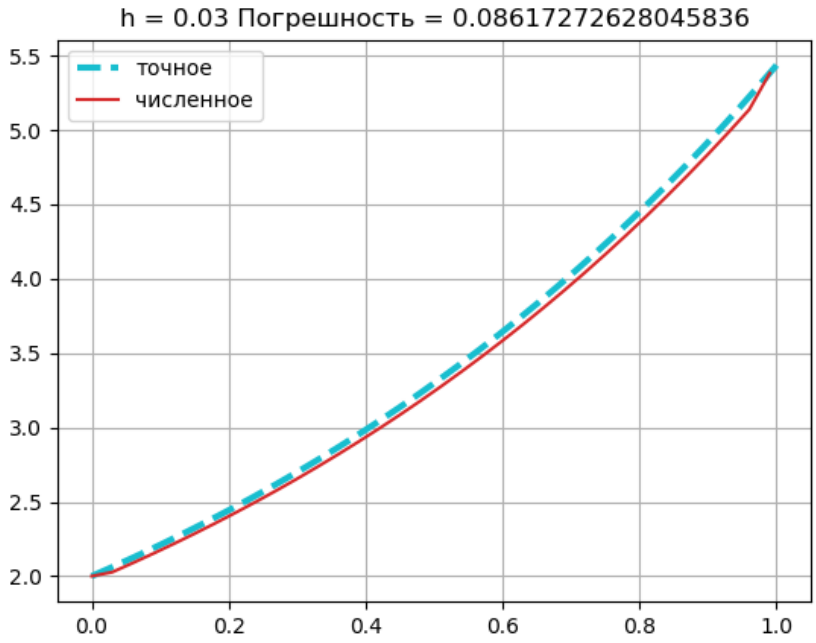

h = 0.5 Погрешность = 0.7388220219645039
точное
численное

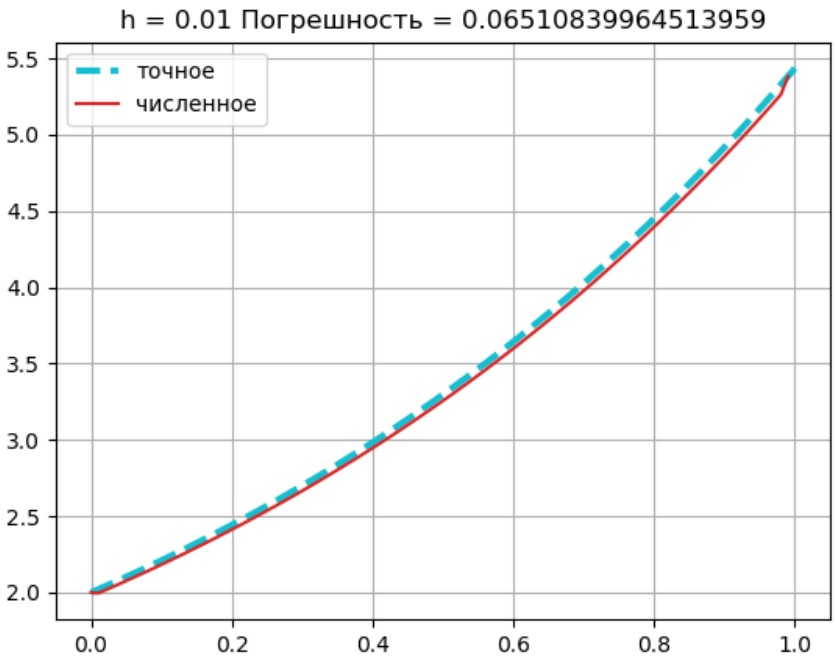

h = 0.1 Погрешность = 0.12446670777943325
точное
численное

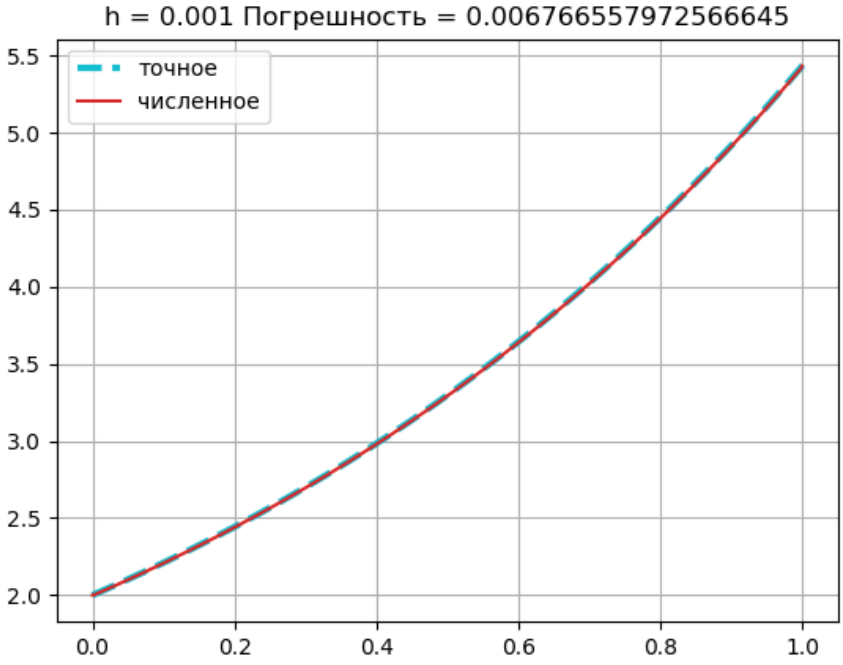


$f(x) - \frac{1}{2}\int_0^1 e^{x-t} f(t)dt = e^x$ – уравнение Фредгольма 2 рода

$f(x) = 2e^x$ - решение







5. Код программы:

```
import numpy
import matplotlib.pyplot as plt


def Fredgolm(K, g, x, h, lamb):
    n = x.size
    A = numpy.zeros((n, n))
    for i in range(n):
        if i == 0:
```

```
            A[i, 0] = 1-0.5*lamb*h*K(x[i], x[0])
        else:
            A[i, 0] = - 0.5 * lamb * h * K(x[i], x[0])

    for i in range(1, n - 1):
        for j in range(n):
            if j == i:
                A[i, j] = 1 - lamb * h * K(x[i], x[j])
            else:
                A[i, j] = - lamb * h * K(x[i], x[j])

    for i in range(n):
        if i == n-1:
            A[i, n-1] = 1-lamb * 0.5 * h * K(x[i], x[n-1])
        else:
            A[i, n - 1] = - lamb * 0.5 * h * K(x[i], x[n - 1])

    return numpy.linalg.solve(A, g(x))

def K(x, t):
    return 1/(0.64*numpy.cos((x+t)/2)**2 - 1)

def g(x):
    return 25 - 16*numpy.sin(x)**2

def f(x):
    return 17/2 + (128/17)*numpy.cos(2*x)

def error(x, y, f):
    max = 0
    for i in range(x.size):
        e = numpy.abs(f(x[i]) - y[i])
        if (e > max):
            max = e
    return max

a = -numpy.pi
b = numpy.pi
lamb = 3/(10*numpy.pi)

h_exact = 0.001
x_exact = numpy.arange(a, b, h_exact)
y_exact = f(x_exact)
plt.plot(x_exact, y_exact, 'C9', label = 'точное', linewidth=3,
linestyle='--')

h_approx = 0.5
x_approx = numpy.arange(a, b, h_approx)
y_approx = Fredgolm(K, g, x_approx, h_approx, lamb)
y_approx[0]=f(x_approx[0])
y_approx[y_approx.size-1] = f(x_approx[x_approx.size-1])
plt.plot(x_approx, y_approx, 'C3', label = 'численное')
```

```
plt.title('h = ' + str(h_approx) + ' Погрешность = ' + str(error(x_approx,
y_approx, f)))
plt.grid(True)
plt.legend()
plt.show()
```

6. Выводы:

Графики наглядно демонстрируют, какова будет точность расчета при различных значениях шага h. Чем меньше длина отрезка интегрирования [a,b], тем короче стоит выбирать шаг. В процессе работы над курсовым проектом я ознакомилась с уравнениями Фредгольма второго рода, стоит отметить, что несмотря на простоту, данный метод обеспечивает достаточно высокую степень точности, что наглядно отражено на графиках.