БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №2

Численные методы решения интегральных уравнений

Вариант 12

Выполнил:
Черепенников Р.М.
3 курс 8 группа

Преподаватель:
Будник А.М.

2021

1. Постановка задачи

Даны интегральные уравнения Фредгольма и Вольтерра 2-го рода при следующих значениях их параметров:

$K(x,s)$	$f(x)$	λ	$[a,b]$
e^{x-s}	$e^{x-2}(e-1)$	0.3	$[0,\frac{\pi}{2}]$

При таком значении λ метод простых итераций для ИУФ-2 не сходится, поэтому заменим

2. Метод механических квадратур

2.1. Теория

ИУФ-2

Дано интегральное уравнение Фредгольма 2-го рода:

$$u(x) - \lambda \int_a^b K(x,s)u(s)ds = f(x) \tag{1}$$

Для решения необходимо найти значение $u(x)$ в узлах $x_i, i = \overline{0,n}$. Вместо интеграла подставим квадратурную сумму и получим:

$$y(x_i) - \lambda \sum_{k=0}^{n} A_k K(x_i,x_k)y(x_k) = f(x_i), \qquad i = \overline{0,n}$$

Здесь и далее рассматривается функция $y(x)$ которая является приближением искомой функции $u(x)$.

Введем обозначения $y_i = y(x_i),\ K_{ik} = K(x_i,x_k),\ f_i = f(x_i)$. Тогда систему можно записать следующим образом:

$$y(x_i) - \lambda \sum_{k=0}^{n} A_k K(x_i,x_k)y(x_k) = f(x_i)\,, \qquad i = \overline{0,n}$$

Или, в матричном виде:

$$\begin{bmatrix} 1-\lambda A_0 K_{00} & \cdots & -\lambda A_n K_{0n} \\ \vdots & \ddots & \vdots \\ -\lambda A_0 K_{n0} & \cdots & 1-\lambda A_n K_{nn} \end{bmatrix} \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix}$$

Решив данную систему получим значения функции в узлах x_i. Чтобы найти значение функции в произвольной точке можно проинтерполировать функцию по имеющимся значениям либо применить формулу

$$y(x) = \lambda \sum_{k=0}^{n} A_k K(x,x_k)y_k + f(x_k), \tag{2}$$

ИУВ-2

Дано интегральное уравнение Вольтерра 2-го рода:

$$u(x) - \lambda \int_a^x K(x,s)u(s)ds = f(x) \tag{3}$$

Метод механических квадратур для уравнения Вольтерра можно применить, сведя уравнение Вольтерра к уравнению Фредгольма с помощью замены

$$\widetilde{K}(x,s) = \begin{cases} K(x,s), & s \le x \\ 0, & s > x \end{cases}$$

Повторяя действия, описанные для уравнения Фредгольма, получаем следующую систему:

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -\lambda A_0 K_{10} & 1-\lambda A_1 K_{11} & 0 & \cdots & 0 \\ 0 & \vdots & \ddots & \vdots & 0 \\ -\lambda A_0 K_{n0} & \cdots & \cdots & \cdots & 1-\lambda A_n K_{nn} \end{bmatrix} \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix}$$

Приближенное решение в любой точке находится с помощью формулы (2).

2.2. Листинг программы

```
import numpy as np

def get_quad_coefs(a, b, N, method):
    h = (b-a)/N
    coefs = []

    if method == 'r_rectangles':
        coefs.append(0)
        for i in range(N):
            coefs.append(h)

    elif method == 'simpson':
        for i in range(N+1):
            if(i == 0) or i == N:
                coefs.append(h/3)
            elif i%2 == 0:
                coefs.append(2*h/3)
            elif i%2 == 1:
                coefs.append(4*h/3)

    return np.array(coefs)
```

```
# Метод механических квадратур


def mech_quad(K, f, lmb, a, b, N, method):
    h = (b-a)/N
    x = np.array([a + h*i for i in range(N+1)])


    coefs = get_quad_coefs(a, b, N, method)


    A = np.zeros((N+1, N+1)) # матрица системы
    b = np.zeros((N+1, 1)) # неоднородность системы
    for i in range(N+1):
        b[i] = f(x[i])
        for j in range(N+1):
            A[i][j] = 1 - lmb* coefs[j]*K(x[i], x[i]) if i == j else -
lmb*coefs[j]*K(x[i], x[j])
    u =np.linalg.solve(A,b).flatten()
    return x, u, coefs


def get_solution_at_point(x,u,quad_coefs, lmb,K, f, x0):
    u_0 = 0.0
    for x_i, y_i, A_i in zip(x, u, quad_coefs):
        u_0 += lmb * A_i * K(x0, x_i) * y_i


    u_0 += f(x0)
    return u_0
print('ИУф-2')
solution = mech_quad(K, f, lmb, a, b, 10, 'simpson')
for i in range(solution[0].shape[0]):
    print(f'u({solution[0][i]:.2f}) = {solution[1][i]:.6f}')


print(f'u(a+b/2.2) = u({(a+b)/2:.2f})={get_solution_at_point(solution[0],
solution[1], solution[2], lmb, K, f, (a+b)/2):.6f}')
print()
print('ИУВ-2')
solution = mech_quad(K_volt, f, lmb, a, b, 10, 'r_rectangles')
for i in range(solution[0].shape[0]):
    print(f'u({solution[0][i]:.2f}) = {solution[1][i]:.6f}')
```

```
print(f'u(a+b/2.2) = u({(a+b)/2:.2f})={get_solution_at_point(solution[0],
solution[1], solution[2], lmb, K_volt, f, (a+b)/2):.6f}')
```

2.3. Полученные результаты

```
ИУф-2
u(0.00) = 0.275879
u(0.16) = 0.322803
u(0.31) = 0.377708
u(0.47) = 0.441952
u(0.63) = 0.517123
u(0.79) = 0.605080
u(0.94) = 0.707998
u(1.10) = 0.828420
u(1.26) = 0.969325
u(1.41) = 1.134196
u(1.57) = 1.327110
u(a+b/2.2) = u(0.79)=0.605080

ИУВ-2
u(0.00) = 0.232544
u(0.16) = 0.276440
u(0.31) = 0.328621
u(0.47) = 0.390652
u(0.63) = 0.464392
u(0.79) = 0.552052
u(0.94) = 0.656258
u(1.10) = 0.780134
u(1.26) = 0.927394
u(1.41) = 1.102450
u(1.57) = 1.310551
u(a+b/2.2) = u(0.79)=0.552052
```

3 Метод последовательных приближений

3.1. Теория

ИУФ-2

Рассматривается уравнение (1).

Решение будем искать в следующем виде:

$$y_n(x) = \sum_{i=0}^{n} \lambda^i \phi_i(x)$$

где $\phi_i(x)$ находятся следующим образом:

$$\phi_0(x) = f(x)$$

$$\phi_i(x) = \int_a^b K(x,s)\phi_{i-1}(s)dx\,, \qquad i = \overline{1,n}$$

Заменяя интеграл квадратурной суммой получим

$$\phi_0(x) = f(x)$$

$$\phi_i(x) = \sum_{k=0}^{N} A_k K(x, x_k)\phi_{i-1}(x_k), \qquad i = \overline{1,n}$$

Применяя формулу (2) можно найти решение в любой точке.

ИУВ-2

Рассматривается уравнение (3).

Решение будем искать в следующем виде:

$$y_n(x) = \sum_{i=0}^{n} \lambda^i \phi_i(x)$$

где $\phi_i(x)$ находятся следующим образом:

$$\phi_0(x) = f(x)$$

$$\phi_i(x) = \int_a^x K(x,s)\phi_{i-1}(s)dx, \qquad i = \overline{1,n}$$

Заменяя интеграл квадратурной суммой получим

$$\phi_0(x) = f(x)$$

$$\phi_i(x_j) = \sum_{k=0}^{j} A_k^j K(x_j, x_k)\phi_{i-1}(x_k), \qquad i = \overline{1,n}\ j = \overline{0,N}$$

Применяя следующую формулу можно найти решение в любой точке:

$$y_n(x) = \lambda \sum_{x_k < x} A_k K(x, x_k) y_{nk} + f(x_k)$$

3.2. Листинг программы

```
import numpy as np

def get_quad_coefs(a, b, N, method):
    h = (b-a)/N
    coefs = []

    if method == 'r_rectangles':
        coefs.append(0)
        for i in range(N):
            coefs.append(h)
```

```
    elif method == 'simpson':
        for i in range(N+1):
            if(i == 0) or i == N:
                coefs.append(h/3)
            elif i%2 == 0:
                coefs.append(2*h/3)
            elif i%2 == 1:
                coefs.append(4*h/3)

    return np.array(coefs)

def iterations_method(K, f, lmb, a, b, N_splits, N_it, method):
    h = (b-a)/N_splits
    x = np.array([a + h*i for i in range(N_splits+1)])
    coefs = get_quad_coefs(a, b, N_splits, method)
    phis = np.zeros((N_it+1, N_splits+1)) # (i, j) элемент y_i(x_j)

    for i in range(N_it+1):
        for j in range(N_splits+1):
            if i == 0:
                phis[i][j] = f(x[j])
            else:
                phis[i][j] = np.sum([ coefs[k]*K(x[j], x[k])*phis[i-1][k] for
k in range(N_splits + 1)])

    u = np.zeros((N_splits + 1, ))
    for i in range(N_splits + 1):
        for j in range(N_it + 1):
            u[i] += lmb**j * phis[j][i]
    return x, u, coefs

print('ИУф-2')
solution = iterations_method(K, f, lmb, a, b, 10,5 ,'simpson')
for i in range(solution[0].shape[0]):
    print(f'u({solution[0][i]:.2f}) = {solution[1][i]:.6f}')
```

```
print(f'u(a+b/2.2) = u({(a+b)/2:.2f})={get_solution_at_point(solution[0],
solution[1], solution[2], lmb, K, f, (a+b)/2):.6f}')

print()

print('ИУВ-2')

solution = iterations_method(K_volt, f, lmb, a, b, 10,5 ,'r_rectangles')

for i in range(solution[0].shape[0]):

    print(f'u({solution[0][i]:.2f}) = {solution[1][i]:.6f}')


print(f'u(a+b/2.2) = u({(a+b)/2:.2f})={get_solution_at_point(solution[0],
solution[1], solution[2], lmb, K, f, (a+b)/2):.6f}')

print()
```

3.3. Полученные результаты

```
ИУф-2
u(0.00) = 0.275875
u(0.16) = 0.322798
u(0.31) = 0.377703
u(0.47) = 0.441946
u(0.63) = 0.517116
u(0.79) = 0.605071
u(0.94) = 0.707987
u(1.10) = 0.828408
u(1.26) = 0.969310
u(1.41) = 1.134179
u(1.57) = 1.327090
u(a+b/2.2) = u(0.79)=0.605079

ИУВ-2
u(0.00) = 0.232544
u(0.16) = 0.276440
u(0.31) = 0.328621
u(0.47) = 0.390652
u(0.63) = 0.464392
u(0.79) = 0.552052
u(0.94) = 0.656258
u(1.10) = 0.780134
u(1.26) = 0.927394
u(1.41) = 1.102450
u(1.57) = 1.310551
u(a+b/2.2) = u(0.79)=0.597530
```

4. Сравнительный анализ полученных результатов

Рассмотрим погрешность метода последовательных приближений для ИУФ-2 и ИУВ-2.

ИУФ-2: $|\varepsilon_n(x)| \leq L\frac{q^{n+1}}{1-q}$

ИУВ-2: $|\varepsilon_n(x)| \leq L\frac{q^{n+1}}{(n+1)!} * \frac{1}{1-\frac{q}{n+2}}$

где $q = |\lambda| M(b-a),\ |f(x)| \le L, |K(x,s)| \le M$

Причем метод последовательных приближений сходится только при $q \le 1$

Для нашей задачи мы получили следующие оценки:

ИУФ-2: $|\varepsilon_n(x)| \le$ 0.852067

ИУВ-2: $|\varepsilon_n(x)| \le$ 0.00031916

Сравнивая полученные значения для ММП и ММК можно сделать вывод, что задача решается верно, так как для различных методов получаем близкие значения как в узлах, так и в еще одной точке.

К недостаткам МПП можно отнести то, что в этом методе происходит накопление погрешности. Так как ϕ_i вычисляется приближенно, а далее на основе этого приближения вычисляется ϕ_{i+1}.

Так же для ИУФ-2 метод последовательных приближений может быть применен, из-за этого даже пришлось заменить исходные данные для нашей задачи. К плюсам же МПП в сравнении с ММК можно отнести меньшую трудоемкость, так как при использовании ММК необходимо решать СЛАУ.