**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Лабораторная работа № 2

**Решение интегральных уравнений Фредгольма и Вольтера**

Вариант 4 (в)

**Выполнил:**

Дунаев В. А.

3 курс 6 группа

**Преподаватель:**

Будник А.М.

**Минск, 2017**

**Постановка задачи:**

Даны интегральные уравнения Фредгольма и Вольтера:

– ИУ Фредгольма второго рода;

– ИУ Вольтера второго рода.

1.Методом механических квадратур найти решение интегрального уравнения Фредгольма второго рода. Для вычисления интегралов использовать квадратурные формулы средних прямоугольников (n=10).

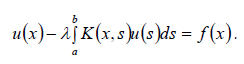
2.Методом последовательных приближений решить интегральное уравнение Фредгольма с точностью ε=.

3.Методом механических квадратур найти решение интегрального уравнения Вольтера второго рода. Для вычисления интегралов использовать квадратурные формулы трапеций (n = 10).

**Краткая теория**

**Метод механических квадратур для ИУФ – 2.**

Рассмотрим интегральное уравнение Фредгольма второго рода:



1

Выберем на отрезке [*a*; *b*] (*n* +1) точек и заменим в уравнении (1) интеграл некоторой квадратурной суммой, причем точки будут ее узлами. Тогда вместо (1) получим равенство:

2

где – коэффициенты выбранной квадратурной формулы, а ρ(*x*) – ее остаток. Соответствующие выражения для них мы получали ранее.

Если теперь в равенстве (2) положить последовательно , то в результате получится система линейных алгебраических уравнений для нахождения точных значений решения в узлах:

3

Остаток ρ(*x*) квадратурной формулы обычно мал по сравнению с самой величиной квадратурной суммы, поэтому, отбрасывая в (3) малые величины λρ(, получим систему линейных алгебраических уравнений

4

Решения системы (4) будут являться некоторыми приближениями к точным решениям задачи (1) в узлах . Найти их можно, используя для этих целей любой (или наиболее подходящий) из методов решения систем линейных алгебраических уравнений (например, метод Гаусса).

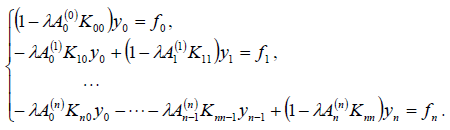
Для КФ средних прямоугольников

**Метод механических квадратур для ИУВ – 2.**

Выберем на отрезке [*a*; *b*] (*n* +1) точек и рассмотрим уравнение в этих точках:

Заменяя интеграл квадратурной суммой и отбрасывая остаток получим:

или в развёрнутом виде:



Матрица данной системы является нижней треугольной. Поэтому решение системы по сути представляет собой обратный ход метода Гаусса.

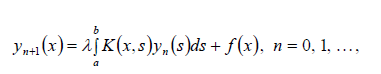
Упростим вычисления. Преобразуем формулу следующим образом:

Очевидно, что такая формула не требует вычисления коэффициентов системы и значения находятся автоматически.

Для КФ трапеций

**Метод последовательных приближений для ИУФ – 2**

Для метода последовательных приближений будем пользоваться формулой:



Заменяя интеграл квадратурной суммой получим:

**Метод последовательных приближений для ИУВ – 2**

В случае уравнения Вольтера перепишется в виде:

*,*

Заменяя интеграл квадратурной суммой получим:

**Ход работы**

1. **Уравнение Фредгольма (метод механических квадратур):**

|  |  |
| --- | --- |
| **Узел** | **Значение** |
| 0.0 | 1.9010870376472133 |
| 0.1 | 2.112087028747158 |
| 0.2 | 2.3710121326057836 |
| 0.3 | 2.6886002669751656 |
| 0.4 | 3.0779801283377184 |
| 0.5 | 3.5552019234800443 |
| 0.6 | 4.139885754431216 |
| 0.7 | 4.856013721102825 |
| 0.8 | 5.732897578320552 |
| 0.9 | 6.806360834476281 |
| 1 | 8.120182790768016 |

1. **Уравнение Вольтерра (метод механических квадратур):**

|  |  |
| --- | --- |
| **Узел** | **Значение** |
| 0.0 | 1.0050251256281406 |
| 0.1 | 1.128678316834707 |
| 0.2 | 1.301503279131756 |
| 0.3 | 1.5312902297748294 |
| 0.4 | 1.8347899316984257 |
| 0.5 | 2.234194397803142 |
| 0.6 | 2.7593123064161307 |
| 0.7 | 3.4508369426198904 |
| 0.8 | 4.3653741577726 |
| 0.9 | 5.583380274320598 |
| 1 | 7.222050374884784 |

1. **Метод последовательных приближений для ИУФ – 2**

|  |  |
| --- | --- |
| **Узел** | **Значение** |
| 0.0 | 1.900989733478848 |
| 0.1 | 2.1119794910100733 |
| 0.2 | 2.3708932850261615 |
| 0.3 | 2.6884689200864846 |
| 0.4 | 3.0778349675761687 |
| 0.5 | 3.5550414960279326 |
| 0.6 | 4.139708454676682 |
| 0.7 | 4.8558177745703315 |
| 0.8 | 5.7326810239113435 |
| 0.9 | 6.806121504841042 |
| 1 | 8.119918290615317 |

Потребовалось 9 итерации для .

1. **Метод последовательных приближений для ИУВ – 2**

|  |  |
| --- | --- |
| **Узел** | **Значение** |
| 0.0 | 1.005025125628125 |
| 0.1 | 1.1286783168344667 |
| 0.2 | 1.3015032791280694 |
| 0.3 | 1.5312902297364868 |
| 0.4 | 1.8347899313897214 |
| 0.5 | 2.234194395725867 |
| 0.6 | 2.759312294133257 |
| 0.7 | 3.4508368765401682 |
| 0.8 | 4.3653738261337445 |
| 0.9 | 5.583378692657079 |
| 1 | 7.2220431060236585 |

Потребовалось 6 итерации для .

**Листинг программы**

**(Метод мех. квадратур для ИУФ-2)**

import javafx.util.Pair;  
import java.util.ArrayList;  
import java.util.List;  
  
public class FredholmEquation {  
  
 private double leftSide;  
  
 private double rightSide;  
  
 private int N;  
  
 private double h;  
  
 private double lambda = 0.5;  
  
 public FredholmEquation(double leftSide, double rightSide, int n) {  
 this.leftSide = leftSide;  
 this.rightSide = rightSide;  
 N = n;  
 this.h = (this.rightSide - this.leftSide) / this.N;  
 }  
  
 private double getA(int index) {  
  
 if (index > 0) {  
 return this.h;  
 } else {  
 return 0;  
 }  
  
 }  
  
 private double K(double x\_i, double s) {  
  
 return Math.*sin*(x\_i + s);  
 }  
  
 private double f(double x\_i) {  
  
 return 1 + Math.*sin*(x\_i);  
 }  
  
 public List<Pair<Double, Double>> solveRightRectangles() throws Exception {  
  
 double[][] sysMatrix = new double[this.N + 1][this.N + 1];  
 double[] x = new double[this.N + 1];  
 double[] f\_i = new double[this.N + 1];  
 for (int i = 0; i <= this.N; i++) {  
 x[i] = this.leftSide + i \* this.h;  
 }  
 for (int i = 0; i <= this.N; i++) {  
  
 f\_i[i] = f(x[i]);  
  
 for (int j = 0; j <= this.N; j++) {  
  
 sysMatrix[i][j] -= this.lambda \* getA(j) \* K(x[i], x[j]);  
 }  
  
 sysMatrix[i][i] += 1;  
  
 }  
 List<Pair<Double, Double>> y = new ArrayList<>(this.N + 1);  
  
 double[] solution = Utils.*theGauss*(sysMatrix, f\_i);  
  
 for (int i = 0; i < solution.length; i++) {  
 y.add(new Pair<>(x[i], solution[i]));  
 }  
  
 return y;  
  
 }  
}

**(Метод мех. квадратур для ИУВ-2)**

import javafx.util.Pair;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class VolterraEquation {  
  
 private double leftSide;  
  
 private double rightSide;  
  
 private int N;  
  
 private double h;  
  
 private double lambda = 0.5;  
  
 public VolterraEquation(double leftSide, double rightSide, int n) {  
 this.leftSide = leftSide;  
 this.rightSide = rightSide;  
 this.N = n;  
 this.h = (this.rightSide - this.leftSide) / this.N;  
  
 }  
  
 private double getA(int i, int limit) {  
 if (i == 0 || i == limit) {  
 return this.h / 2;  
 } else {  
 return this.h;  
 }  
 }  
  
 private double K(double x\_i, double s) {  
  
 return Math.sin(x\_i + s);  
 }  
  
 private double f(double x\_i) {  
  
 return 1 + Math.sin(x\_i);  
 }  
  
 public List<Pair<Double, Double>> solve() {  
  
 List<Pair<Double, Double>> y = new ArrayList<>(this.N + 1);  
 double[] x = new double[this.N + 1];  
  
 for (int i = 0; i <= this.N; i++) {  
 x[i] = this.leftSide + i \* this.h;  
 }  
  
 for (int i = 0; i <= this.N; i++) {  
  
 double y\_i;  
  
 double first = Math.pow((1 - getA(i, i) \* K(x[i], x[i]) \* this.lambda), -1);  
  
 double second = 0;  
  
 for (int j = 0; j < i; j++) {  
  
 second += this.lambda \* getA(j, i) \* K(x[i], x[j]) \* y.get(j).getValue();  
 }  
  
 second = f(x[i]) + second;  
  
 y\_i = first \* second;  
  
 y.add(new Pair<>(x[i], y\_i));  
 }  
  
 return y;  
  
 }  
}

**(Метод последовательных приближений для ИУФ-2)**

import javafx.util.Pair;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class FredholmIterations {  
 private double leftSide;  
 private double rightSide;  
  
 private int N;  
  
 private double h;  
  
 private double lambda = 0.5;  
  
 public FredholmIterations(double leftSide, double rightSide, int n) {  
 this.leftSide = leftSide;  
 this.rightSide = rightSide;  
 N = n;  
 this.h = (this.rightSide - this.leftSide) / this.N;  
 }  
  
 private double getA(int index) {  
  
 if (index > 0) {  
 return this.h;  
 } else {  
 return 0;  
 }  
 }  
  
 private double K(double x\_i, double x\_k) {  
  
 return Math.sin(x\_i + x\_k);  
 }  
  
 private double f(double x\_i) {  
  
 return 1 + Math.sin(x\_i);  
 }  
  
 public List<Pair<Double, Double>> solveRightRectangles() {  
  
 double[] x = new double[this.N + 1];  
 double[] yPrevious = new double[this.N + 1];  
 double[] yCurrent = new double[this.N + 1];  
 for (int i = 0; i <= this.N; i++) {  
 x[i] = this.leftSide + i \* this.h;  
 yCurrent[i] = f(x[i]);  
 }  
 int iterationsCount = 0;  
 do {  
  
 System.arraycopy(yCurrent, 0, yPrevious, 0, this.N + 1);  
  
 for (int i = 0; i <= this.N; i++) {  
  
 double sum = 0;  
 for (int j = 0; j <= this.N; ++j) {  
  
 sum += this.lambda \* getA(i) \* K(x[i], x[j]) \* yPrevious[j];  
 }  
 yCurrent[i] = sum + f(x[i]);  
 }  
  
 iterationsCount++;  
  
 } while (Utils.norm(yCurrent, yPrevious) > 0.001);  
  
 System.out.println(iterationsCount);  
  
 List<Pair<Double, Double>> result = new ArrayList<>(this.N + 1);  
 for (int i = 0; i < yCurrent.length; i++) {  
 result.add(new Pair<>(x[i], yCurrent[i]));  
 }  
 return result;  
 }  
  
}

**(Метод последовательных приближений для ИУВ-2)**

import javafx.util.Pair;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class VolterraIterations {  
 private double leftSide;  
  
 private double rightSide;  
  
 private int N;  
  
 private double h;  
  
 private double lambda = 0.5;  
  
 public VolterraIterations(double leftSide, double rightSide, int n) {  
 this.leftSide = leftSide;  
 this.rightSide = rightSide;  
 N = n;  
 this.h = (this.rightSide - this.leftSide) / this.N;  
 }  
  
 private double getA(int i, int limit) {  
 if (i == 0 || i == limit) {  
 return this.h / 2;  
 } else {  
 return this.h;  
 }  
 }  
  
 private double K(double x\_i, double x\_k) {  
  
 return Math.sin(x\_i + x\_k);  
 }  
  
 private double f(double x\_i) {  
  
 return 1 + Math.sin(x\_i);  
 }  
  
 public List<Pair<Double, Double>> solveRightTrapeeze() {  
  
 double[] x = new double[this.N + 1];  
 double[] yPrevious = new double[this.N + 1];  
 double[] yCurrent = new double[this.N + 1];  
 for (int i = 0; i <= this.N; i++) {  
 x[i] = this.leftSide + i \* this.h;  
 yPrevious[i] = f(x[i]);  
 }  
 int iterationsCounter = 0;  
 do {  
 System.arraycopy(yCurrent, 0, yPrevious, 0, this.N + 1);  
  
 for (int i = 0; i <= this.N; i++) {  
  
 double sum = 0;  
 for (int j = 0; j <= i; ++j) {  
  
 sum += this.lambda \* getA(j, i) \* K(x[i], x[j]) \* yPrevious[j];  
 }  
 yCurrent[i] = sum + f(x[i]);  
 }  
 iterationsCounter++;  
 } while (Utils.norm(yCurrent, yPrevious) > 0.001);  
  
 System.out.println(iterationsCounter);  
  
 List<Pair<Double, Double>> result = new ArrayList<>(this.N + 1);  
 for (int i = 0; i < yCurrent.length; i++) {  
 result.add(new Pair<>(x[i], yCurrent[i]));  
 }  
 return result;  
 }  
  
}