БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №2

**Решение интегральных уравнений**

Вариант 7

**Выполнила:**

Зуйкевич Лидия

3 курс 7 группа

**Преподаватель:**

Будник А.М.

Минск, 2022

Оглавление

Предварительные вычисления 3

Постановка задачи 4

[1. Метод механических квадратур 5](#_Toc121187455)

Алгоритм решения для ИУФ-2 5

Алгоритм решения для ИУВ-2 6

Листинг программы 7

Вывод программы 10

Выводы 10

[2. Метод последовательных приближений 11](#_Toc121187457)

Алгоритм решения для ИУФ-2 11

Алгоритм решения для ИУВ-2 13

Листинг программы 14

Вывод программы 17

Выводы 17

Предварительные вычисления

Дано интегральное уравнение Фредгольма 2 рода (ИУФ-2):

Параметры:

Проверим условие сходимости метода последовательных приближений для ИУФ-2:

Где M – оценка ядра сверху по 2 переменным:

В нашем случае:

Т.е. условие сходимости не выполняется. Выберем такое, чтобы это условие выполнялось, например, , и будем использовать такое значение во всех заданиях.

Постановка задачи

Даны интегральное уравнение Фредгольма 2 рода (ИУФ-2):

И интегральное уравнение Вольтерра 2-го рода (ИУВ-2):

Параметры:

для ИУФ-2

Необходимо:

1. Методом механических квадратур найти приближенные решения при

N = 10, используя составную формулу трапеций для ИУФ-2 и левых прямоугольников для ИУВ-2.

1. Методом последовательных приближений при n = 5 найти приближенные решения ИУФ-2 и ИУВ-2. Для вычисления интегралов применить квадратурные формулы, указанные в п.1. Оценить погрешность.
2. Провести сравнительный анализ полученных в п. 1 – 2 результатов. Вычислить по каждому из методов решение в точке и объяснить разницу.

# Метод механических квадратур

1. ИУФ-2

Алгоритм решения

Зададим сетку узлов:

И заменим интеграл в уравнении квадратурной формулой:

Теперь запишем приближенное решение по этой формуле в каждой точке сетки:

В результате получим систему вида:

Матрица системы B имеет вид:

Где и в нашем случае:

Используем квадратурную формулу трапеций, тогда:

Для нахождения решения системы воспользуемся методом Гаусса, получим приближенное решение в каждой точке сетки:

Для нахождения приближенного решения в точке воспользуемся формулой:

1. ИУВ-2

Алгоритм решения

Зададим сетку узлов: и запишем уравнение в каждом узле сетки:

Теперь каждый интеграл определенный. Заменяем его квадратурной формулой:

В результате получим систему вида:

Матрица системы B имеет вид:

Матрица нижнетреугольная.

Где и в нашем случае:

Используем квадратурную формулу левых прямоугольников, тогда:

(т.е. матрица с единицами на главной диагонали)

Для нахождения решения системы воспользуемся обратным ходом метода Гаусса, получим приближенное решение в каждой точке сетки:

Для нахождения приближенного решения в точке используем интерполяционный многочлен Лагранжа:

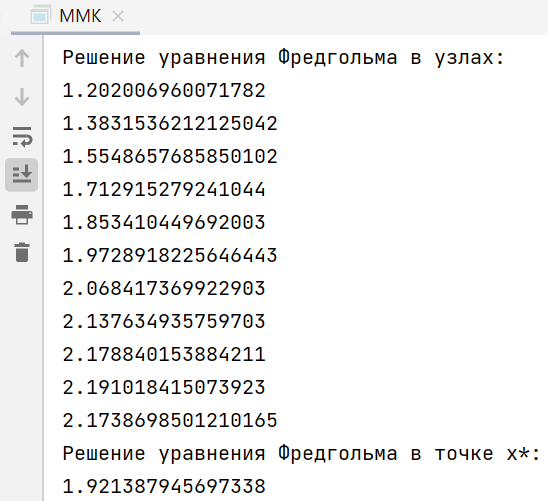
Где:

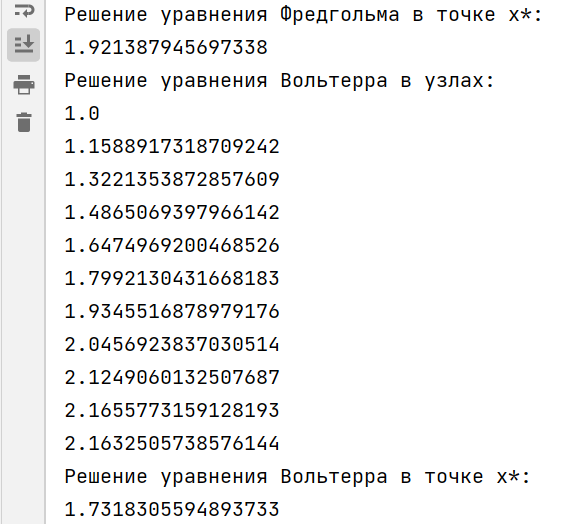
Формула, по которой вычисляем значение в , выглядит так:

# Листинг программы

import java.util.Arrays;  
  
public class MMK {  
 private final double a;  
 private final double b;  
 private final int n;  
 private final double lambda;  
 private double h;  
 private double [][] B;  
 private double[] f;  
 private double[] y;  
 private final double[] x;  
  
 public MMK(double a, double b, int n, double lambda) {  
 this.a = a;  
 this.b = b;  
 this.n = n;  
 this.lambda = lambda;  
 this.h = (b - a) / n;  
 this.B = new double [n + 1][n + 1];  
 this.y = new double [n + 2];  
 this.f = new double [n + 1];  
 this.x = new double[n + 1];  
  
 for(int i = 0; i <= n; i++){  
 x[i] = a + h \* i;  
 }  
 }  
  
 public double K(double x, double s){  
 return Math.*sin*(x + s);  
 }  
  
 public double function(double x){  
 return (Math.*sin*(x) + 1);  
 }  
  
 public void fillDataForFIE(){  
 for(int i = 0; i <= n; i++){  
 f[i] = function(x[i]);  
  
 for(int j = 1; j < n; j++){  
 B[i][j] = - lambda \* h \* K(x[i], x[j]);  
  
 if (j == i){  
 B[i][j]++;  
 }  
 }  
 B[i][0] = - lambda \* (h / 2) \* K(x[i], a);  
 B[i][n] = - lambda \* (h / 2) \* K(x[i], x[n]);  
 }  
 B[0][0]++;  
 B[n][n]++;  
 }  
  
 public void fillDataForVIE(){  
 for (int i = 0; i <= n; i++){  
 Arrays.*fill*(B[i], i, n + 1, 0);  
 }  
 Arrays.*fill*(y, 0);  
  
 for(int i = 0; i <= n; i++){  
 f[i] = function(x[i]);  
 B[i][i] = 1;  
  
 for(int j = 0; j < i; j++){  
 B[i][j] = - lambda \* h \* K(x[i], x[j]);  
 }  
 }  
 }  
  
 public void gaussMethod(){  
 double temp1;  
 double temp2;  
  
 *//прямой ход* for (int k = 0; k <= n; k++){  
 temp1 = B[k][k];  
 for(int j = 0; j <= n; j++) {  
 B[k][j] = B[k][j] / temp1;  
 }  
 f[k] /= temp1;  
 for(int i = k + 1; i <= n; i++) {  
 temp2 = B[i][k];  
 f[i] = f[i] - (f[k] \* temp2);  
 for(int j = 0; j <= n; j++) {  
 B[i][j] = B[i][j] - (B[k][j] \* temp2);  
 }  
 }  
 }  
*//обратный ход* y[n] = f[n];  
 for (int k = n - 1; k >= 0; k--) {  
 y[k] = f[k];  
 for (int j = k + 1; j <= n; j++) {  
 y[k] -= B[k][j] \* y[j];  
 }  
 }  
  
 for (int i = 0; i <= n; i++){  
 System.*out*.println(y[i] + " ");  
 }  
 }  
  
 public void findValueInPointForFIE(double p){  
 for(int i = 1; i < n; i++){  
 y[n + 1] += lambda \* h \* K(p, x[i]) \* y[i];  
 }  
 y[n + 1] += lambda \* (h / 2) \* (K(p, a) \* y[0] + K(p, x[n]) \* y[n]) + function(p);  
  
 System.*out*.println(y[n + 1]);  
 }  
  
 public void findValueInPointForVIE(double p){  
 double w = 1;  
  
 for (int i = 0; i <= n; i++){  
 w \*= (p - x[i]);  
  
 double temp = y[i];  
 for(int j = 0; j <= n; j++){  
 if (i != j){  
 temp /= (x[i] - x[j]);  
 }  
 }  
 temp /= (p - x[i]);  
 y[n + 1] += temp;  
 }  
 y[n + 1] \*= w;  
 System.*out*.println(y[n + 1]);  
 }  
  
 public static void main(String[] args){  
 MMK obj = new MMK(0, Math.*PI* / 2, 10, 0.1);  
 obj.fillDataForFIE();  
  
 System.*out*.println("Решение уравнения Фредгольма в узлах: ");  
 obj.gaussMethod();  
 System.*out*.println("Решение уравнения Фредгольма в точке х\*: ");  
 obj.findValueInPointForFIE(Math.*PI* / 4.4);  
 obj.fillDataForVIE();  
 System.*out*.println("Решение уравнения Вольтерра в узлах: ");  
 obj.gaussMethod();  
 System.*out*.println("Решение уравнения Вольтерра в точке х\*: ");  
 obj.findValueInPointForVIE(Math.*PI* / 4.4);  
 }  
}

Вывод программы





Выводы

Для решения уравнений Фредгольма и Вольтерра использовались квадратурные формулы трапеций и левых прямоугольников, имеющие АСТ 1 и 0 соответственно, с небольшим количеством разбиений N = 10, поэтому очень тчоных результатов ожидать не следует. Система, полученная при решении уравнений, была решена методом Гаусса. Метод Гаусса является прямым методом, поэтому решение системы найдено точно.

Для нахождения значения решения ИУВ-2 в точке был использован интерполяционный многочлен Лагранжа, приближение с помощью которого также имеет свою погрешность.

# Метод последовательных приближений

1. ИУФ-2

Алгоритм решения

Метод последовательных приближений основан на том, что решение уравнения представимо в виде бесконечного ряда:

Подставляем такое выражение в уравнение и приравниваем коэффициенты при одинаковых степенях , находим :

…

…

Возьмем частную сумму ряда и получим формулу для нахождения приближенного решения:

Но мы будем использовать рекуррентную формулу, более удобную для итерации:

Зададим сетку узлов: и заменим интегралы квадратурными формулами, получаем:

Где . В нашем случае:

Задаем , тогда . Т.к. n = 5, по этой формуле для получения приближенного решения надо найти . Данный алгоритм реализован на двумерном массиве, где на каждой итерации одна строка массива содержит значения в узлах сетки, а вторая – значения в узлах.

Используем квадратурную формулу трапеций, тогда:

Для нахождения значения искомой функции в точке добавим ее в конец таблицы и будем считать значения в ней на каждой итерации по тем же формулам, что и в других узлах сетки (только сама точка в квадратурную сумму не включается):

Оценим погрешность найденного решения. Для этого нам нужно оценить и :

( мы изменили, чтобы получить сходимость)

Формула для оценки погрешности:

В нашем случае:

1. ИУВ-2

Алгоритм решения

Для интегрального уравнения Вольтерра рекуррентная формула будет выглядеть так:

Зададим сетку узлов: и запишем приближенное решение в таком виде в каждом узле сетки:

Теперь каждый интеграл определенный. Заменим интегралы квадратурными формулами, получим:

Где . Используем квадратурную формулу левых прямоугольников, тогда:

Получаем формулу:

Для нахождения приближенного решения в точке используем интерполяционный многочлен Лагранжа:

Где:

Формула, по которой вычисляем значение в , выглядит так:

Оценим погрешность найденного решения:

Формула для оценки погрешности:

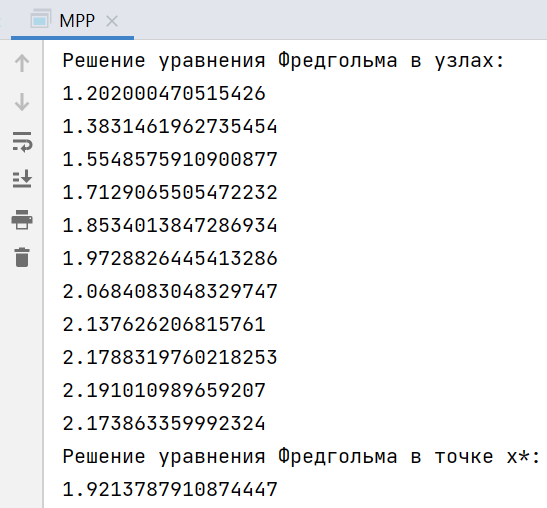
В нашем случае:

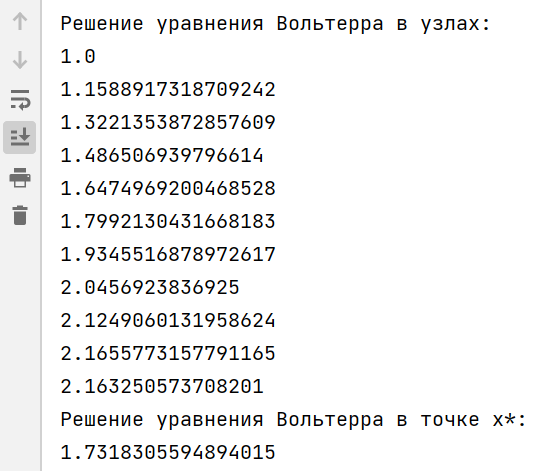
Условие сходимости: , у нас - выполняется.

# Листинг программы

import java.util.Arrays;  
  
public class MPP {  
 private final double a;  
 private final double b;  
 private final int n;  
 private final double lambda;  
 private double h;  
 private double [][] y;  
 private double[] x;  
 final int N = 10;  
  
 public MPP(double a, double b, int n, double lambda) {  
 this.a = a;  
 this.b = b;  
 this.n = n;  
 this.h = (this.b - a) / N;  
 this.y = new double [2][N + 2];  
 this.x = new double [N + 2];  
 this.lambda = lambda;  
 }  
  
 public double K(double k, double s){  
 return Math.*sin*(k + s);  
 }  
  
 public double function(double p){  
 return (Math.*sin*(p) + 1);  
 }  
  
 public void solveFIE(double p){  
 for(int i = 0; i <= N; i++){  
 x[i] = a + h \* i;  
 y[0][i] = function(x[i]);  
 }  
 x[N + 1] = p;  
 y[0][N + 1] = function(x[N + 1]);  
  
 int count = n;  
 int t = 0;  
 int k = 1;  
 int temp;  
 while (count > 0){  
 for(int i = 0; i < N + 2; i++){  
 for(int j = 1; j < N; j++){  
 y[k][i] += lambda \* h \* K(x[i], x[j]) \* y[t][j];  
 }  
 y[k][i] += lambda \* (h / 2) \* (K(x[i], x[0]) \* y[t][0] + K(x[i], x[N]) \* y[t][N]);  
 y[k][i] += function(x[i]);  
 }  
 temp = k;  
 k = t;  
 t = temp;  
 count--;  
  
 Arrays.*fill*(y[k], 0);  
 }  
 for (int i = 0; i < N + 1; i++){  
 System.*out*.println(y[t][i]);  
 }  
 System.*out*.println("Решение уравнения Фредгольма в точке х\*: ");  
 System.*out*.println(y[t][N + 1]);  
 }  
  
 public void solveVIE(double p){  
 Arrays.*fill*(y[0], 0);  
 Arrays.*fill*(y[1], 0);  
  
 for(int i = 0; i <= N; i++){  
 y[0][i] = function(x[i]);  
 }  
  
 int count = n;  
 int t = 0;  
 int k = 1;  
 int temp;  
 while (count > 0){  
 for(int i = 0; i < N + 1; i++){  
 for(int j = 0; j < i; j++){  
 y[k][i] += lambda \* h \* K(x[i], x[j]) \* y[t][j];  
 }  
 y[k][i] += function(x[i]);  
 }  
 temp = k;  
 k = t;  
 t = temp;  
 count--;  
  
 Arrays.*fill*(y[k], 0);  
 }  
 for (int i = 0; i < N + 1; i++){  
 System.*out*.println(y[t][i]);  
 }  
 System.*out*.println("Решение уравнения Вольтерра в точке х\*: ");  
 findValueInPointForVIE(p);  
 }  
  
 public void findValueInPointForVIE(double p){  
 double w = 1;  
  
 for (int i = 0; i <= N; i++){  
 w \*= (p - x[i]);  
  
 double temp = y[1][i];  
 for(int j = 0; j <= N; j++){  
 if (i != j){  
 temp /= (x[i] - x[j]);  
 }  
 }  
 temp /= (p - x[i]);  
 y[1][N + 1] += temp;  
 }  
 y[1][N + 1] \*= w;  
 System.*out*.println(y[1][N + 1]);  
 }  
  
 public static void main(String[] args){  
 MPP obj = new MPP(0, Math.*PI* / 2, 5, 0.1);  
 System.*out*.println("Решение уравнения Фредгольма в узлах: ");  
 obj.solveFIE(Math.*PI* / 4.4);  
 System.*out*.println("Решение уравнения Вольтерра в узлах: ");  
 obj.solveVIE(Math.*PI* / 4.4);  
 }  
}

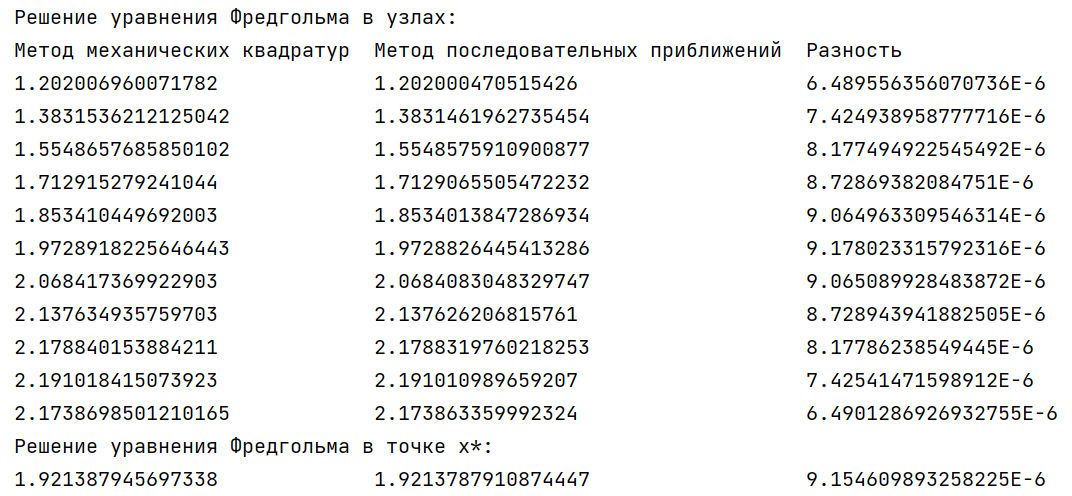
Вывод программы

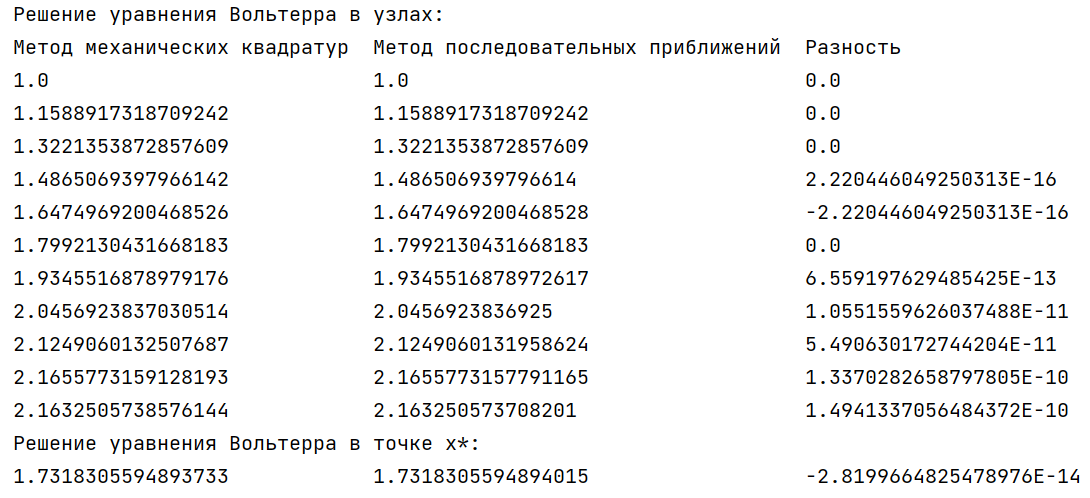




Выводы

Для решений, найденного методом последовательных приближений, по формулам оценили теоретическую погрешность, имеющую порядок для ИУФ и для ИУВ. Значение решения ИУФ в точке нашли, добавив точку в конец массива узлов, т.е. по формулам самого метода, значение решения ИУВ в точке найдено с помощью интерполяционного многочлена Лагранжа. Сравним результаты, полученные методом механических квадратур и методом последовательных приближений:





Значения решения ИУФ в узлах и в точке совпадают до 4 – 5 знаков после запятой, разность между ними имеет порядок . Разность значений в точке имеет тот же порядок, что и разность значений в узлах, т.к. мы использовали формулу самого метода для нахождения значения в точке . Учитывая то, что теоретическая погрешность решения, найденного методом последовательных приближений, имеет порядок , то решения, найденные обоими методами, достаточно точны (имеют погрешность порядка

Значения решения ИУВ в некоторых узлах совпадают, в остальных имеют разность порядка . Разность значений в точке имеет порядок , для нахождения значений в точке мы использовали интерполяционный многочлен Лагранжа, построенный с использованием значений решения в узлах (значения для обоих методов в каждом узле достаточно близки), поэтому и найденные в точке значения отличаются несильно (разность порядка . Несмотря на то, что в методе последовательных приближений было использовано одно и то же n = 5 для ИУФ и ИУВ, решение, найденное для ИУВ, имеет меньшую оценку теоретической погрешности и более близкие к полученным методом механических квадратур результаты, что подтверждает то, что метод последовательных приближений для ИУВ сходится быстрее.