**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

Лабораторная №2

**Интегральные уравнения**

Вариант 2

**Выполнил:**

Кендысь Алексей Максимович

студент 3 курса, 7 группы,

специальность

“прикладная математика”

**Преподаватель:**

Доцент кафедры вычислительной

математики ФПМИ,

А.М. Будник

Минск, 2022

**Содержание:**

Постановка задачи 2

Метод механических квадратур 2

Краткие теоретические сведения 2

Листинг программы 3

Результаты 4

Выводы 5

Метод последовательных приближений 5

Краткие теоретические сведения 5

Листинг программы 6

Результаты 8

Выводы 8

Постановка задачи

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

- ИУФ-2

- ИУВ-2

Необходимо:

1. Методом механических квадратур найти приближённые решения при , используя составную формулу правых прямоугольников для ИУФ-2 и составную формулу левых прямоугольников для ИУВ-2.
2. Методом последовательных приближений при найти приближённые решения ИУФ-2 и ИУВ-2. Для вычисления интегралов применять квадратурные формулы, указанные в п.1. Оценить погрешность.
3. Провести сравнительный анализ полученных в п.п.1,2 результатов. Вычислить по каждому методу решение в точке и объяснить разницу.

Метод механических квадратур

Краткие теоретические сведения

*ИУФ-2:*

Для ИУФ-2 интеграл в уравнении заменяется на составную квадратурную формулу правых прямоугольников на равномерном разбиении отрезка с количеством разбиений :

где и, используя составную формулу правых прямоугольников,

В полученное выражение подставляем последовательно Получаем систему:

*,* где – вектора*,*

– матрица ,

где

Учитывая вид матрицы , система решается для матрицы размерности (матрица без первой строки и первого столбца), откуда находятся Для решения системы используется метод Гаусса. Для нахождения используется формула .

Для нахождения значения в точке используется формула , в которую подставляем точку. Получаем:

*ИУВ-2:*

Для ИУВ-2 сразу последовательно подставляются те же , а дальше каждый интеграл в уравнении заменяется на соответствующую составную квадратурную формулу левых прямоугольников.

При этом для составной формулы левых прямоугольников:

Получаем систему с нижнетреугольной матрицей, решение которой получаем обратным ходом метода Гаусса:

При этом учитываем, что , и для нахождения используем формулу .

Для нахождения значения в точке используется интерполяционный многочлен Лагранжа:

И для приближённого значения в точке берётся .

Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double x) {  
 return 1 + x;  
 }  
}  
  
class K {  
 public static double getValue(double x, double s) {  
 return 1 / (x + s + 1);  
 }  
}  
  
class IntegralApprox {  
 public final static int *N* = 10;  
 private final double a;  
 private final double[] akLRS;  
 private final double[] akRRS;  
 private final double[] xk;  
 private final double[] fk;  
 private final double h;  
 private final double x;  
 private double fAtX;  
 private final static double *CONST\_FOR\_X* = 2.2;  
  
 public IntegralApprox(double a, double b) {  
 this.a = a;  
 this.akLRS = new double[*N* + 1];  
 this.akRRS = new double[*N* + 1];  
 this.xk = new double[*N* + 1];  
 this.fk = new double[*N* + 1];  
  
 this.x = (a + b) / *CONST\_FOR\_X*;  
 this.h = (b - a) / *N*;  
  
 this.setXk();  
 this.setFk();  
 this.setFAtX();  
 this.setAkRRS();  
 this.setAkLRS();  
 }  
  
 public double[] getAkLRS() {  
 return this.akLRS;  
 }  
  
 public double[] getAkRRS() {  
 return this.akRRS;  
 }  
  
 public double[] getXk() {  
 return this.xk;  
 }  
  
 public double[] getFk() {  
 return this.fk;  
 }  
  
 public double getFAtX() {  
 return this.fAtX;  
 }  
  
 public double getX() {  
 return this.x;  
 }  
  
 private void setXk() {  
 for (int i = 0; i < *N* + 1; i++) {  
 this.xk[i] = this.a + i \* this.h;  
 }  
 }  
  
 private void setAkLRS() {  
 for (int i = 0; i < *N*; i++) {  
 this.akLRS[i] = this.h;  
 }  
  
 this.akLRS[*N*] = 0.;  
 }  
  
 private void setAkRRS() {  
 this.akRRS[0] = 0.;  
  
 for (int i = 1; i < *N* + 1; i++) {  
 this.akRRS[i] = this.h;  
 }  
 }  
  
 private void setFk() {  
 for (int i = 0; i < *N* + 1; i++) {  
 this.fk[i] = F.*getValue*(this.xk[i]);  
 }  
 }  
 private void setFAtX() {  
 this.fAtX = F.*getValue*(this.x);  
 }  
}  
  
class ResultOutput {  
 public static void outRes(int iN, double[] ykFred, double[] ykVolt, double yAtXFred, double yAtXVolt) {  
 Formatter fmt = new Formatter();  
 fmt.format("Полученное приближённое решение yk для уравнения Фредгольма:\n");  
 for (int i = 0; i < iN + 1; i++) {  
 fmt.format("%.7f\n", ykFred[i]);  
 }  
 fmt.format("Значение в точке x\*: ");  
 fmt.format("%.7f\n", yAtXFred);  
 fmt.format("Полученное приближённое решение yk для уравнения Вольтерра:\n");  
 for (int i = 0; i < iN + 1; i++) {  
 fmt.format("%.7f\n", ykVolt[i]);  
 }  
 fmt.format("Значение в точке x\*: ");  
 fmt.format("%.7f\n", yAtXVolt);  
 System.*out*.println(fmt);  
 }  
}  
  
class MMQ {  
 private final int iN;  
 private final double lambda;  
 private final double[] akFred;  
 private final double[] akVolt;  
 private final double[] xk;  
 private final double[] fk;  
 private final double[] ykFred;  
 private final double[] ykVolt;  
 private final double x;  
 private final double fAtX;  
 private double yAtXFred;  
 private double yAtXVolt;  
  
 public MMQ(IntegralApprox ia, double lambda) {  
 this.akFred = ia.getAkRRS();  
 this.akVolt = ia.getAkLRS();  
 this.xk = ia.getXk();  
 this.fk = ia.getFk();  
 this.x = ia.getX();  
 this.fAtX = ia.getFAtX();  
 this.iN = IntegralApprox.*N*;  
  
 this.lambda = lambda;  
  
 this.ykFred = new double[this.iN + 1];  
 this.ykVolt = new double[this.iN + 1];  
 }  
  
 public void Fredholm() {  
 double[][] systemMatrix;  
 systemMatrix = new double[this.iN][this.iN];  
 this.setSystemMatrix(systemMatrix);  
  
 double[] fkCopy = new double[this.iN];  
 System.*arraycopy*(fk, 1, fkCopy, 0, fkCopy.length);  
  
 double[] yk1 = Gauss.*solve*(systemMatrix, fkCopy);  
 System.*arraycopy*(yk1, 0, this.ykFred, 1, yk1.length);  
 this.ykFred[0] = this.calcYk0Fred();  
  
 this.yAtXFred = this.calcYAtXFred();  
 }  
  
 public void Volterra() {  
 for (int i = 0; i < this.iN + 1; i++) {  
 for (int k = 0; k < i; k++) {  
 this.ykVolt[i] += this.akVolt[k] \* K.*getValue*(this.xk[i], this.xk[k]) \* this.ykVolt[k];  
 }  
  
 this.ykVolt[i] \*= this.lambda;  
 this.ykVolt[i] += this.fk[i];  
  
 if (i != this.iN) {  
 this.ykVolt[i] /= 1. - (this.lambda \* this.akVolt[i] \* K.*getValue*(this.xk[i], this.xk[i]));  
 }  
 }  
  
 this.yAtXVolt = this.calcYAtXVolt();  
 }  
  
 public void outRes() {  
 ResultOutput.*outRes*(this.iN, this.ykFred, this.ykVolt, this.yAtXFred, this.yAtXVolt);  
 }  
  
 private void setSystemMatrix(double[][] matrix) {  
 for (int i = 0; i < this.iN; i++) {  
 for (int j = 0; j < this.iN; j++) {  
 matrix[i][j] = - this.lambda \* this.akFred[j + 1] \* K.*getValue*(this.xk[i + 1], this.xk[j + 1]);  
 if (i == j) {  
 matrix[i][j]++;  
 }  
 }  
 }  
 }  
  
 private double calcYk0Fred() {  
 double res = 0;  
 for (int i = 1; i < this.iN + 1; i++) {  
 res += this.akFred[i] \* K.*getValue*(this.xk[0], this.xk[i]) \* this.ykFred[i];  
 }  
  
 res \*= this.lambda;  
 res += this.fk[0];  
 return res;  
 }  
  
 private double calcYAtXFred() {  
 double res = 0.;  
  
 for (int i = 1; i < this.iN + 1; i++) {  
 res += this.akFred[i] \* K.*getValue*(this.x, this.xk[i]) \* this.ykFred[i];  
 }  
  
 res \*= this.lambda;  
 res += this.fAtX;  
 return res;  
 }  
  
 private double calcYAtXVolt() {  
 Interpolation in = new Interpolation(this.xk, this.ykVolt);  
 return in.getValue(this.x);  
 }  
}  
  
public class Main {  
 private final static double *A* = 0.;  
 private final static double *B* = 1.;  
 private final static double *LAMBDA* = 1.;  
  
 public static void main(String[] args) {  
 IntegralApprox ia = new IntegralApprox(*A*, *B*);  
  
 MMQ mmq = new MMQ(ia, *LAMBDA*);  
 mmq.Fredholm();  
 mmq.Volterra();  
 mmq.outRes();  
 }  
}

Результаты

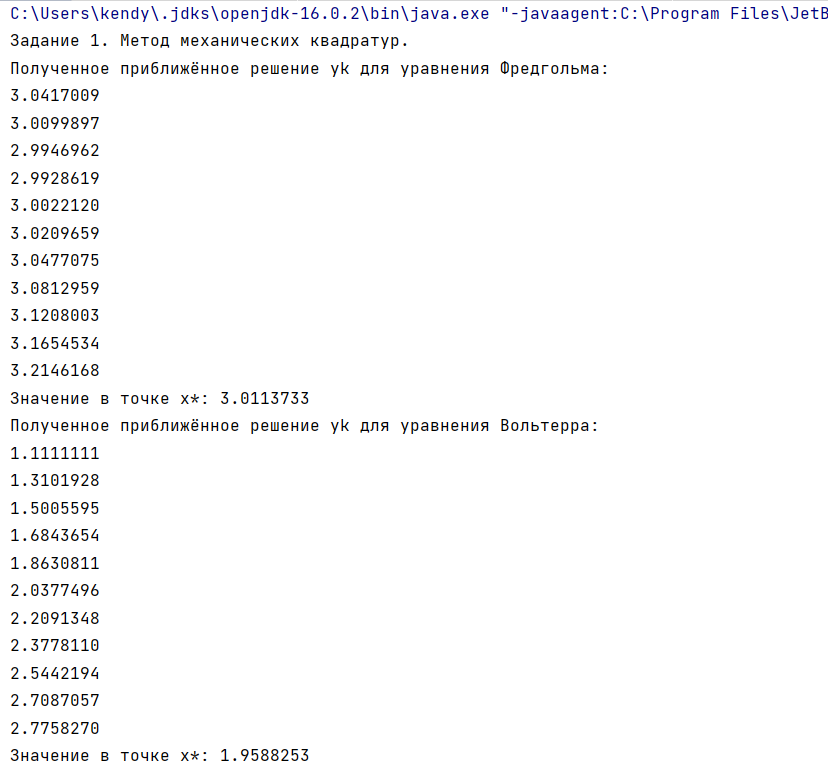


График по точкам для ИУФ-2 (красным отмечено полученное значение в точке ):

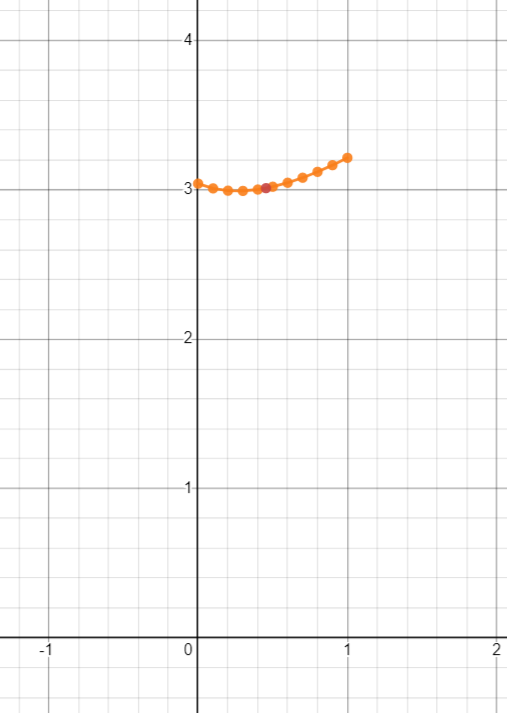
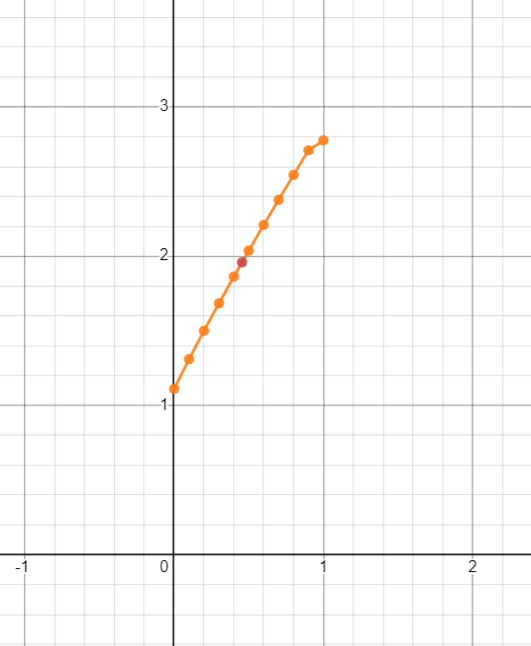


График по точкам для ИУВ-2:



Выводы

Для приближения интегралов были использованы составные квадратурные формулы левых и правых прямоугольников с количеством разбиений = 10. АСТ формул = 0. При этом в прошлой лабораторной при исследовании этих формул для некоторого интеграла для достижения точности потребовалось 8 миллионов разбиений, а в нашем случае . Следовательно, от метода следует ожидать соответствующую погрешность.

Отметим, что для решения системы уравнений для ИУФ-2 использовался метод Гаусса, который является прямым методом, т.е. решение системы получено точно. Система разрешима, когда её определитель не равен 0. Определитель будет равен 0, когда – собственное значение интегрального уравнения. При вычислениях такого поведения не наблюдалось.

Также стоит заметить, что для вычисления значения в точке для ИУФ-2 использовалось приближённое выражение для искомой функции – , которое метод позволяет найти. Для ИУВ-2, в свою очередь, такой возможности нет, поэтому используется интерполирование с помощью многочлена Лагранжа, которое также имеет собственную погрешность.

Метод последовательных приближений

Краткие теоретические сведения

*ИУФ-2:*

Для метода последовательных приближений искомая функция представляется в виде бесконечного ряда: . Далее ряд подставляется в уравнение и приравниваются коэффициенты при одинаковых степенях .

*...*

Тогда приближённое решение ищется в виде:

где . В нашем случае , т.е. требуется найти

Существует также и второй алгоритм, эквивалентный данному:

Для такого вида алгоритма требуется искать .

Используем вторую форму алгоритма. Первая форма была приведена для помощи в оценке погрешности ниже.

При этом для вычисления интегралов используем квадратурные формулы, которые приводили в методе механических квадратур. Для ИУФ-2 используется формула правых прямоугольников, алгоритм перепишется в виде:

Т.к. решение требуется найти в узлах, то при реализации хранится двумерный массив со значениями и . Также при программировании учитывается, что .

Для нахождения значения в точке используется последняя приведённая формула:

*Оценка погрешности для ИУФ-2:*

Для оценки погрешности существует формула:

где ,

Формула справедлива для

В нашем случае . Т.е. формулу применить не можем. Оценим погрешность применения метода для нашего уравнения вручную, используя первую форму алгоритма:

на ,

на ,

Аналогично

Тогда получаем:

*ИУВ-2:*

Метод последовательных приближений во второй форме для ИУВ-2 записывается в виде:

Подставляя узлы и приближая интегралы по составной квадратурной формуле левых прямоугольников, получаем окончательную формулу:

Коэффициенты и узлы определяются по тем же формулам, которые были описаны в методе механических квадратур.

Для нахождения значения в точке используется интерполяционный многочлен Лагранжа, также приведённый в методе механических квадратур.

*Оценка погрешности для ИУВ-2:*

Для оценки погрешности существует формула:

где ,

Формула справедлива для

В нашем случае . Тогда по формуле получаем:

.

Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double x) {  
 return 1 + x;  
 }  
}  
  
class K {  
 public static double getValue(double x, double s) {  
 return 1 / (x + s + 1);  
 }  
}  
  
class IntegralApprox {  
 public final static int *N* = 10;  
 private final double a;  
 private final double[] akLRS;  
 private final double[] akRRS;  
 private final double[] xk;  
 private final double[] fk;  
 private final double h;  
 private final double x;  
 private double fAtX;  
 private final static double *CONST\_FOR\_X* = 2.2;  
  
 public IntegralApprox(double a, double b) {  
 this.a = a;  
 this.akLRS = new double[*N* + 1];  
 this.akRRS = new double[*N* + 1];  
 this.xk = new double[*N* + 1];  
 this.fk = new double[*N* + 1];  
  
 this.x = (a + b) / *CONST\_FOR\_X*;  
 this.h = (b - a) / *N*;  
  
 this.setXk();  
 this.setFk();  
 this.setFAtX();  
 this.setAkRRS();  
 this.setAkLRS();  
 }  
  
 public double[] getAkLRS() {  
 return this.akLRS;  
 }  
  
 public double[] getAkRRS() {  
 return this.akRRS;  
 }  
  
 public double[] getXk() {  
 return this.xk;  
 }  
  
 public double[] getFk() {  
 return this.fk;  
 }  
  
 public double getFAtX() {  
 return this.fAtX;  
 }  
  
 public double getX() {  
 return this.x;  
 }  
  
 private void setXk() {  
 for (int i = 0; i < *N* + 1; i++) {  
 this.xk[i] = this.a + i \* this.h;  
 }  
 }  
  
 private void setAkLRS() {  
 for (int i = 0; i < *N*; i++) {  
 this.akLRS[i] = this.h;  
 }  
  
 this.akLRS[*N*] = 0.;  
 }  
  
 private void setAkRRS() {  
 this.akRRS[0] = 0.;  
  
 for (int i = 1; i < *N* + 1; i++) {  
 this.akRRS[i] = this.h;  
 }  
 }  
  
 private void setFk() {  
 for (int i = 0; i < *N* + 1; i++) {  
 this.fk[i] = F.*getValue*(this.xk[i]);  
 }  
 }  
 private void setFAtX() {  
 this.fAtX = F.*getValue*(this.x);  
 }  
}  
  
class ResultOutput {  
 public static void outRes(int iN, double[] ykFred, double[] ykVolt, double yAtXFred, double yAtXVolt) {  
 Formatter fmt = new Formatter();  
 fmt.format("Полученное приближённое решение yk для уравнения Фредгольма:\n");  
 for (int i = 0; i < iN + 1; i++) {  
 fmt.format("%.7f\n", ykFred[i]);  
 }  
 fmt.format("Значение в точке x\*: ");  
 fmt.format("%.7f\n", yAtXFred);  
 fmt.format("Полученное приближённое решение yk для уравнения Вольтерра:\n");  
 for (int i = 0; i < iN + 1; i++) {  
 fmt.format("%.7f\n", ykVolt[i]);  
 }  
 fmt.format("Значение в точке x\*: ");  
 fmt.format("%.7f\n", yAtXVolt);  
 System.*out*.println(fmt);  
 }  
}  
  
class FPI {  
 private final static int *N* = 5;  
 private final int iN;  
 private final double lambda;  
 private final double[] akFred;  
 private final double[] akVolt;  
 private final double[] xk;  
 private final double[] fk;  
 private double[] ykFred;  
 private double[] ykVolt;  
 private final double x;  
 private final double fAtX;  
 private double yAtXFred;  
 private double yAtXVolt;  
  
 public FPI(IntegralApprox ia, double lambda) {  
 this.akFred = ia.getAkRRS();  
 this.akVolt = ia.getAkLRS();  
 this.xk = ia.getXk();  
 this.fk = ia.getFk();  
 this.x = ia.getX();  
 this.fAtX = ia.getFAtX();  
 this.iN = IntegralApprox.*N*;  
  
 this.lambda = lambda;  
 }  
  
 public void Fredholm() {  
 double[] y1 = new double[this.iN + 1];  
 double[] y2 = new double[this.iN + 1];  
  
 System.*arraycopy*(fk, 0, y1, 0, fk.length);  
  
 for (int i = 0; i < *N*; i++) {  
 this.calcY2Fred(y1, y2);  
 if (i != *N* - 1) {  
 System.*arraycopy*(y2, 0, y1, 0, y2.length);  
 }  
 }  
  
 this.ykFred = y2;  
 this.yAtXFred = this.calcYAtXFred(y1);  
 }  
  
 public void Volterra() {  
 double[] y1 = new double[this.iN + 1];  
 double[] y2 = new double[this.iN + 1];  
  
 System.*arraycopy*(fk, 0, y1, 0, fk.length);  
  
 for (int i = 0; i < *N*; i++) {  
 this.calcY2Volt(y1, y2);  
 System.*arraycopy*(y2, 0, y1, 0, y2.length);  
 }  
  
 this.ykVolt = y2;  
  
 this.yAtXVolt = this.calcYAtXVolt();  
 }  
  
 public void outRes() {  
 ResultOutput.*outRes*(this.iN, this.ykFred, this.ykVolt, this.yAtXFred, this.yAtXVolt);  
 }  
  
 private void calcY2Fred(double[] y1, double[] y2) {  
 for (int j = 0; j < this.iN + 1; j++) {  
 y2[j] = 0.;  
 for (int k = 1; k < this.iN + 1; k++) {  
 y2[j] += this.akFred[k] \* K.*getValue*(this.xk[j], this.xk[k]) \* y1[k];  
 }  
  
 y2[j] \*= this.lambda;  
 y2[j] += fk[j];  
 }  
 }  
  
 private double calcYAtXFred(double[] y1) {  
 double res = 0.;  
 for (int k = 1; k < this.iN + 1; k++) {  
 res += this.akFred[k] \* K.*getValue*(this.x, this.xk[k]) \* y1[k];  
 }  
  
 res \*= this.lambda;  
 res += this.fAtX;  
 return res;  
 }  
  
 private void calcY2Volt(double[] y1, double[] y2) {  
 y2[0] = fk[0];  
 for (int j = 1; j < this.iN + 1; j++) {  
 y2[j] = 0.;  
 for (int k = 0; k < j; k++) {  
 y2[j] += this.akVolt[k] \* K.*getValue*(this.xk[j], this.xk[k]) \* y1[k];  
 }  
  
 y2[j] \*= this.lambda;  
 y2[j] += fk[j];  
 }  
 }  
  
 private double calcYAtXVolt() {  
 Interpolation in = new Interpolation(this.xk, this.ykVolt);  
 return in.getValue(this.x);  
 }  
}  
  
public class Main {  
 private final static double *A* = 0.;  
 private final static double *B* = 1.;  
 private final static double *LAMBDA* = 1.;  
  
 public static void main(String[] args) {  
 IntegralApprox ia = new IntegralApprox(*A*, *B*);  
  
 System.*out*.println("Задание 2. Метод последовательных приближений.");  
  
 FPI fpi = new FPI(ia, *LAMBDA*);  
 fpi.Fredholm();  
 fpi.Volterra();  
 fpi.outRes();  
 }  
}

Результаты

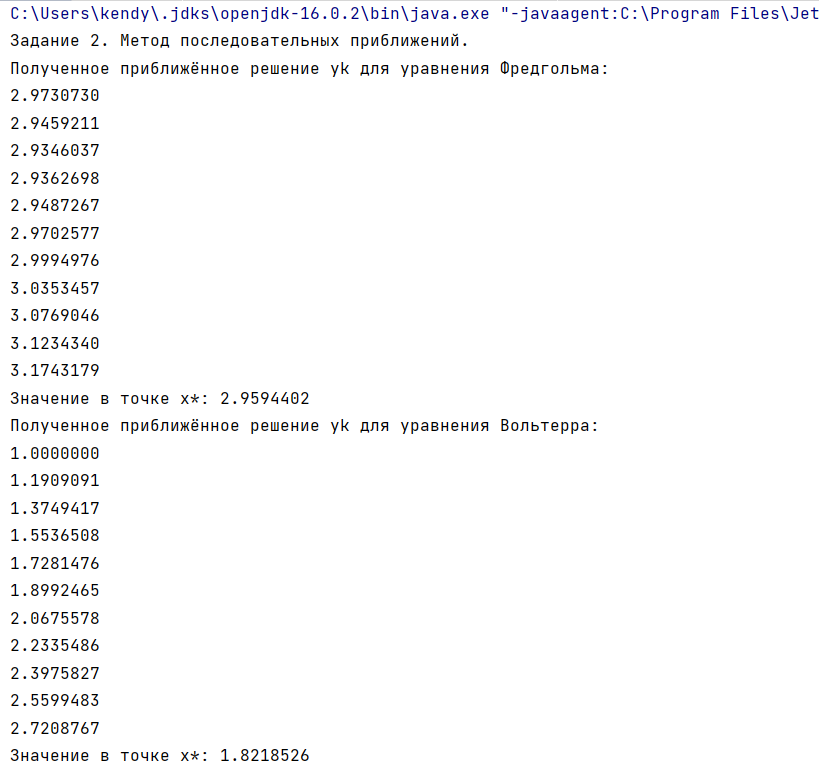


График по точкам для ИУФ-2:

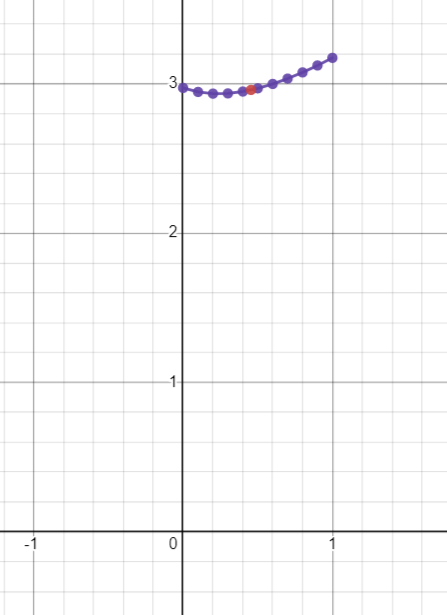
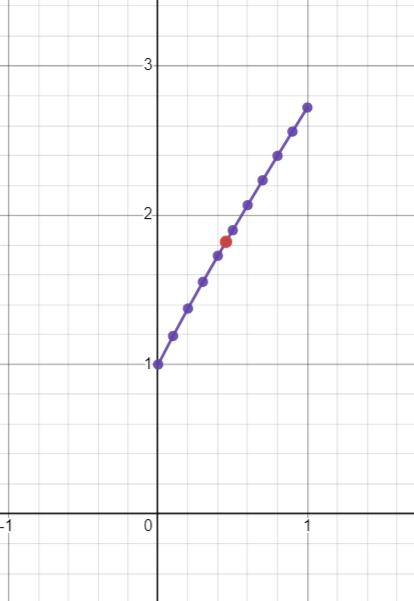
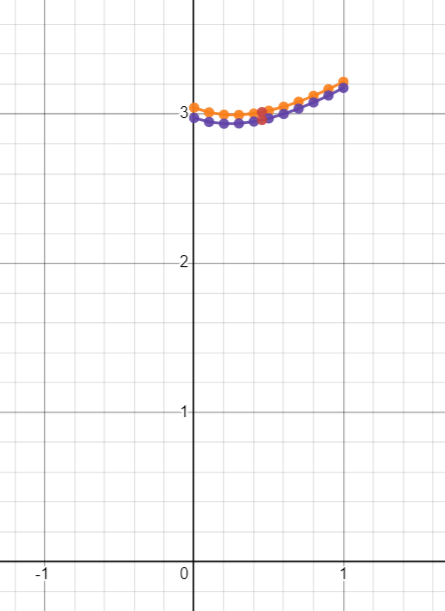


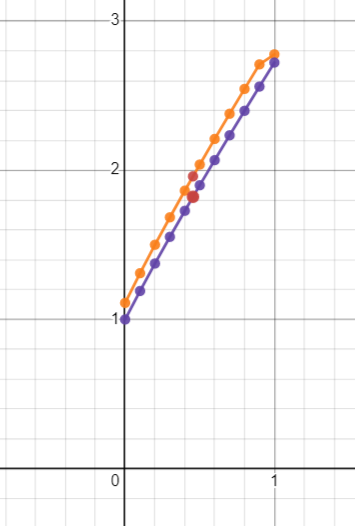
График по точкам для ИУВ-2:



Сравнение метода механических квадратур и метода последовательных приближений для ИУФ-2:



Сравнение метода механических квадратур и метода последовательных приближений для ИУВ-2:



Разница значений в точке для ИУФ-2:

 - ММК

 - МПП

Разница значений в точке для ИУВ-2:

 - ММК

 - МПП

Выводы

Из полученных значений и графиков видно, что результаты вышли похожими для обоих методов для соответствующих уравнений. Тем не менее, значения в узлах отличаются уже на второй значащей цифре (в частности, для значения в точке ). Это объясняется тем, что помимо погрешности самих методов (теоретическую погрешность метода последовательных приближений мы смогли оценить), влияет и погрешность квадратурных формул, с помощью которых приближались интегралы.

Как уже говорилось ранее, используемые формулы правых и левых прямоугольников имеют АСТ = 0, что сильно сказывается на результате. Также используется довольно малое количество разбиений (снова напомним, что в прошлой лабораторной при исследовании этих формул для достижения точности потребовалось 8 миллионов разбиений, в нашем случае ).

При оценке погрешности метода последовательных приближений можно заметить, что для ИУВ-2 метод, вообще говоря, сходится быстрее, т.к. в оценке присутствует факториал. Действительно, для ИУВ-2 мы получили , а для ИУФ-2 . Погрешность же метода механических квадратур определяется погрешностью используемой квадратурной формулы.