**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** = 0.1;  
  
 **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** K(**double** x\_i, **double** s) {  
  
 **return** Math.*exp*(x\_i + s);  
 }  
  
 **private double** f(**double** x\_i) {  
  
 **return** 1 + Math.*exp*(2 \* x\_i) - Math.*exp*(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** \* **h** \* K(x[i], x[j]);  
 }  
  
 sysMatrix[i][i] += 1;  
  
 }  
 List<Pair<Double, Double>> y = **new** ArrayList<>(**this**.**N**);  
 **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** = 0.1;  
  
 **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.*exp*(x\_i + s);  
 }  
  
 **private double** f(**double** x\_i) {  
  
 **return** 1 + Math.*exp*(2 \* x\_i) - Math.*exp*(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** = 0.1;  
  
  
 **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** K(**double** x\_i, **double** s) {  
  
 **return** Math.*exp*(x\_i + s);  
 }  
  
 **private double** f(**double** x\_i) {  
  
 **return** 1 + Math.*exp*(2 \* x\_i) - Math.*exp*(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** \* **h** \* 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**);  
 **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** = 0.1;  
  
 **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** s) {  
  
 **return** Math.*exp*(x\_i + s);  
 }  
  
 **private double** f(**double** x\_i) {  
  
 **return** 1 + Math.*exp*(2 \* x\_i) - Math.*exp*(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; }}