Министерство образования Республики Беларусь
Государственное учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ЭВМ

Лабораторная работа № 2

«Операции свертки и корреляции»

Выполнили:
студенты группы 850502
Новиков И.А.
Карпович К.А.

Проверил:
Третьяков А. Г.

Минск 2021

1. Цель работы

Целью работы является:

Изучение операций корреляции и свертки, их основных свойств, а также методики их получения с помощью быстрого преобразования Фурье (БПФ) на основе теорем о корреляции и свертке.

2. Задание:

№ варианта	Заданная функция	N для БПФ
5	y=cos(x)+sin(x)	8

3. Теоретические сведения, необходимые для выполнения лабораторной работы

Свертка – это математический способ комбинирования двух сигналов для формирования третьего сигнала. Это один из самых важных методов ЦОС. Свертка связывает три сигнала: входной сигнал, выходной сигнал и импульсную характеристику.

Корреляция так же, как свертка, использует два сигнала для получения третьего. Этот третий сигнал называется корреляционным сигналом двух входных сигналов.

Теорема свертки

Если *{X(m)}* и *{Y(m)}* – последовательности действительных чисел, для которых $X(m) \leftrightarrow C_x(k)$, $Y(m) \leftrightarrow C_y(k)$, и свертка этих последовательностей определяется как

$$Z(m) = \frac{1}{N} \sum_{N=0}^{N-1} X(h) Y(m-h), \qquad m = \overline{0, N-1}, \tag{2.1}$$

то

$C_z(k) = C_x(k) C_y(k)$.

Доказательство

Вычисляя *Z(m)*, получим

$$C_z(k) = \frac{1}{N} \sum_{m=0}^{N-1} Z(m) W^{km}. \tag{2.2}$$

Подставляя в (2.2) соотношение свертки (2.1), получим

$$C_z(k) = \frac{1}{N^2} \sum_{m=0}^{N-1} \sum_{h=0}^{N-1} x(h) Y(m-h) W^{km} = \frac{1}{N} \sum_{h=0}^{N-1} x(h) \frac{1}{N} \sum_{m=0}^{N-1} Y(m-h) W^{km}.$$

Согласно теореме сдвига, имеем

$$\frac{1}{N}\sum_{m=0}^{N-1} Y(m-h)W^{km} = W^{kh}C_y(k).$$

Таким образом,

$$C_z(k) = C_y(k)\frac{1}{N}\sum_{h=0}^{N-1} x(h)W^{kh} = C_x(k)C_y(k).$$

Эта теорема утверждает, что свертка временных последовательностей эквивалентна умножению их коэффициентов, полученных после дискретного преобразования Фурье.

Теорема корреляции

Если $X(m) \leftrightarrow C_x(k)$ и $Y(m) \leftrightarrow C_y(k)$, а их функция корреляции определяется соотношением

$$\hat{Z}(m) = \frac{1}{N}\sum_{h=0}^{N-1} X(h)Y(m+h), \text{ где } m = \overline{0, N-1}, \qquad (2.3)$$

то $C_{\hat{z}}(k) = \overline{C}_x(k)C_y(k)$,

где $\overline{C}_x(k)$ - комплексное сопряженное $C_x(k)$

Доказательство

По определению имеем

$$C_{\hat{z}}(k) = \frac{1}{N}\sum_{m=0}^{N-1} \hat{Z}(m)W^{km}. \qquad (2.4)$$

Подставляя (2.3) в (2.4) и меняя порядок суммирования, получаем

$$C_{\hat{z}}(k) = \frac{1}{N}\sum_{h=0}^{N-1} X(h)\left\{\frac{1}{N}\sum_{m=0}^{N-1} Y(m+h)W^{km}\right\}$$

Применяя теорему сдвига, будем иметь

$$C_{\hat{z}}(k) = C_y(k)\left\{\frac{1}{N}\sum_{h=0}^{N-1} X(h)W^{-kh}\right\}.$$

Так как $C_x(k) = \frac{1}{N}\sum_{h=0}^{N-1} X(h)W^{kh}$, то $\frac{1}{N}\sum_{h=0}^{N-1} X(h)W^{-kh} = \overline{C}_x(k)$.

Таким образом, $C_{\hat{z}}(k) = C_y(k)\overline{C}_x(k)$.

Если последовательности *{X(m)}* и *{Y(m)}* идентичны друг другу, то

$$C_{\hat{z}}(k) = |C_x(k)|^2, \; k = \overline{0, N-1}.$$

Обратное ДПФ последовательности $C_{\hat{z}}(k)$ есть $\hat{Z}(m) = \sum_{k=0}^{N-1} C_{\hat{z}}(k)W^{-km}$.

Тогда

$$\frac{1}{N}\sum_{h=0}^{N-1} X^2(h) = \sum_{k=0}^{N-1} |C_x(k)|^2,$$ т.е. справедлива теорема Парсеваля.

Матричное представление корреляции и свертки

Если *{X(m)}* и *{Y(m)}* – две *N*-периодические последовательности действительных чисел, то операции корреляции и свертки определяются соответственно как

$$\hat{Z}(m) = \frac{1}{N}\sum_{h=0}^{N-1} X(h)Y(m+h),$$

$$Z(m) = \frac{1}{N}\sum_{h=0}^{N-1} X(h)Y(m-h).$$

В общем виде корреляцию двух последовательностей можно записать как

$$\begin{bmatrix} \hat{Z}(0) \\ \hat{Z}(1) \\ \hat{Z}(2) \\ \vdots \\ \hat{Z}(N-2) \\ \hat{Z}(N-1) \end{bmatrix} = \frac{1}{N}\begin{bmatrix} X(0) & X(1) & X(2) & \cdots & X(N-1) \\ X(N-1) & X(0) & X(1) & \cdots & X(N-2) \\ X(N-2) & X(N-1) & X(0) & \cdots & X(N-3) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ X(2) & X(3) & X(4) & \cdots & X(1) \\ X(1) & X(2) & X(3) & \cdots & X(0) \end{bmatrix}\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ \vdots \\ Y(N-2) \\ Y(N-1) \end{bmatrix}.$$

В свою очередь, соотношение свертки можно записать в общем виде как

$$\begin{bmatrix} Z(0) \\ Z(1) \\ Z(2) \\ \vdots \\ Z(N-2) \\ Z(N-1) \end{bmatrix} = \frac{1}{N}\begin{bmatrix} X(0) & X(1) & X(2) & \cdots & X(N-1) \\ X(1) & X(2) & X(3) & \cdots & X(0) \\ X(2) & X(3) & X(4) & \cdots & X(1) \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ X(N-2) & X(N-1) & X(0) & \cdots & X(N-3) \\ X(N-1) & X(0) & X(1) & \cdots & X(N-2) \end{bmatrix}\begin{bmatrix} Y(0) \\ Y(N-1) \\ Y(N-2) \\ \vdots \\ Y(2) \\ Y(1) \end{bmatrix}.$$

Если последовательности *{X(m)}* и *{Y(m)}* аналогичны друг другу, то $Z(m) = \frac{1}{N}\sum_{h=0}^{N-1} X(h)X(m+h)$, где $m = \overline{0, N-1}$.

Это соотношение определяет автокорреляцию последовательности *{X(m)}*.

С использованием БПФ схема вычислений корреляции будет иметь вид рис.2.1.

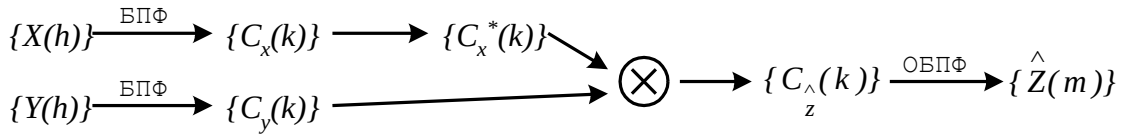

Рис. 1.1. Схема вычисления корреляции

В свою очередь, схему вычисления свертки можно представить как показано на рис.2.2.

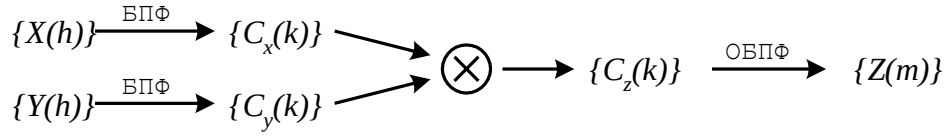

Рис. 1.2. Схема вычисления свертки

4. Результат работы

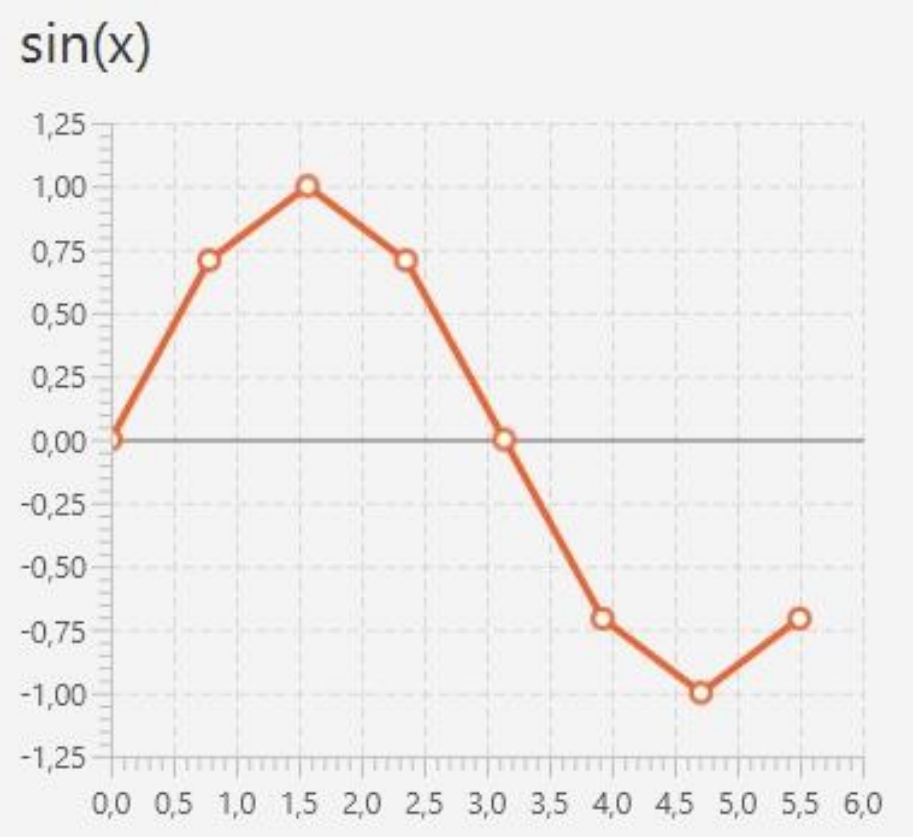

Рис. 1.3. График функции y=sin(x)

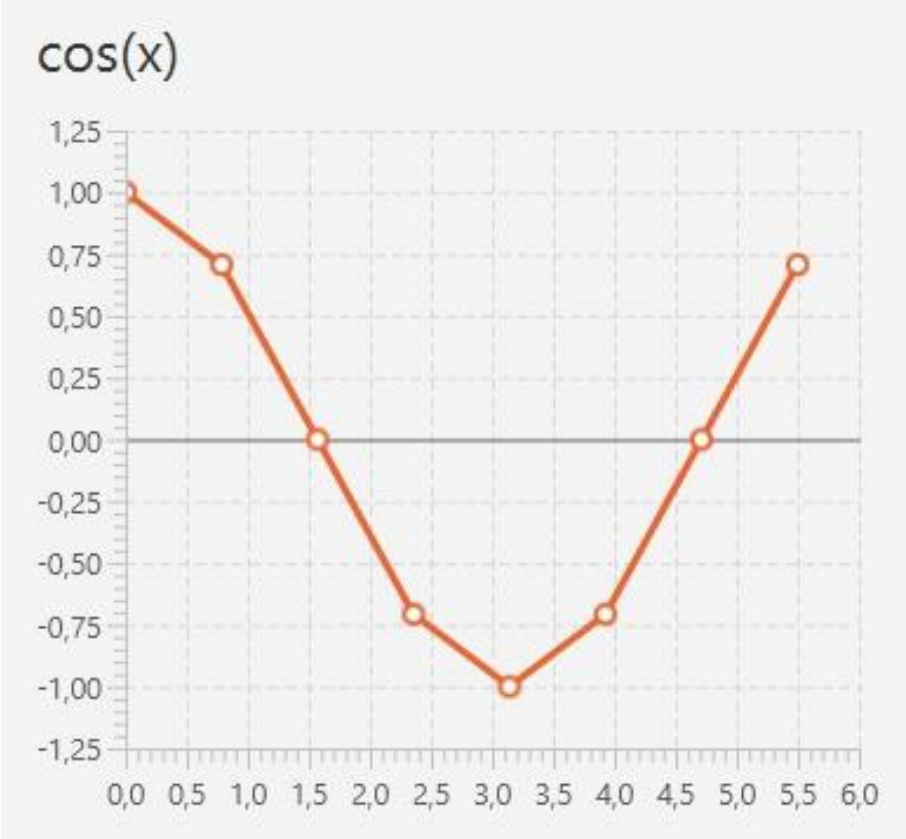

Рис. 1.4. График функции z=cos(x)

Рис. 1.5. Корреляция

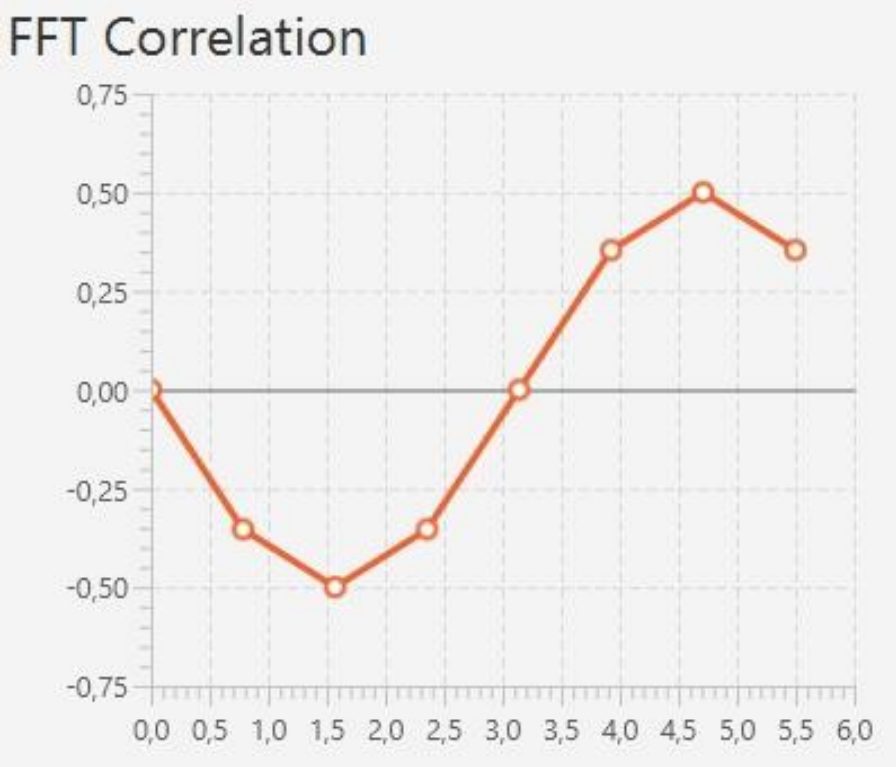

Рис. 1.6. Корреляция БПФ

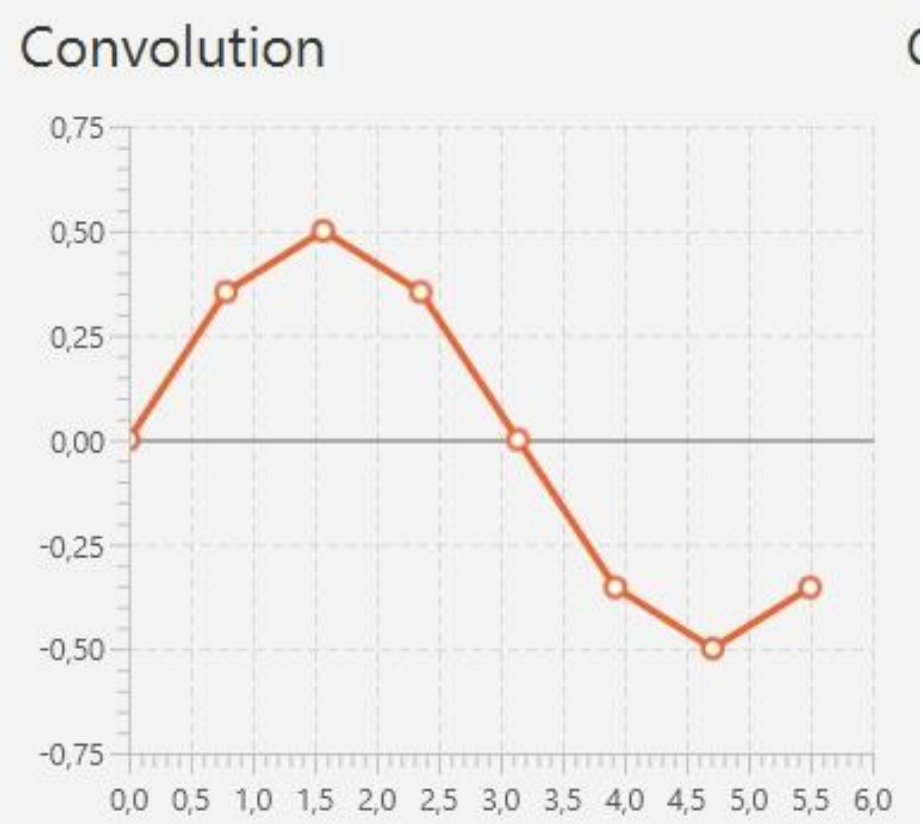

Рис. 1.7. Свертка

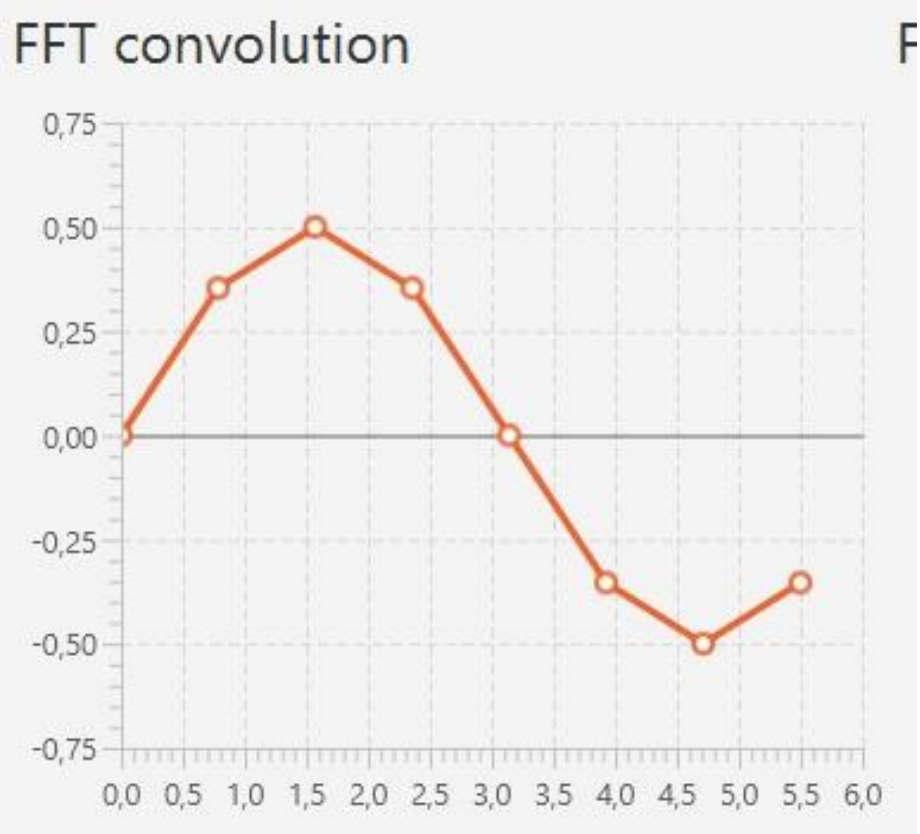

Рис. 1.8. Свертка БПФ

5. Сравнительный анализ

Таблица 1.1

Эффективность БПФ

N	Операций (Корреляция)	Операций (Корреляция БПФ)	Эффективность БПФ
8	128	188	0,68 **:** 1
	Операций (Свертка)	**Операций (Свертка БПФ)**	**Эффективность БПФ**
	128	188	0,68 **:** 1

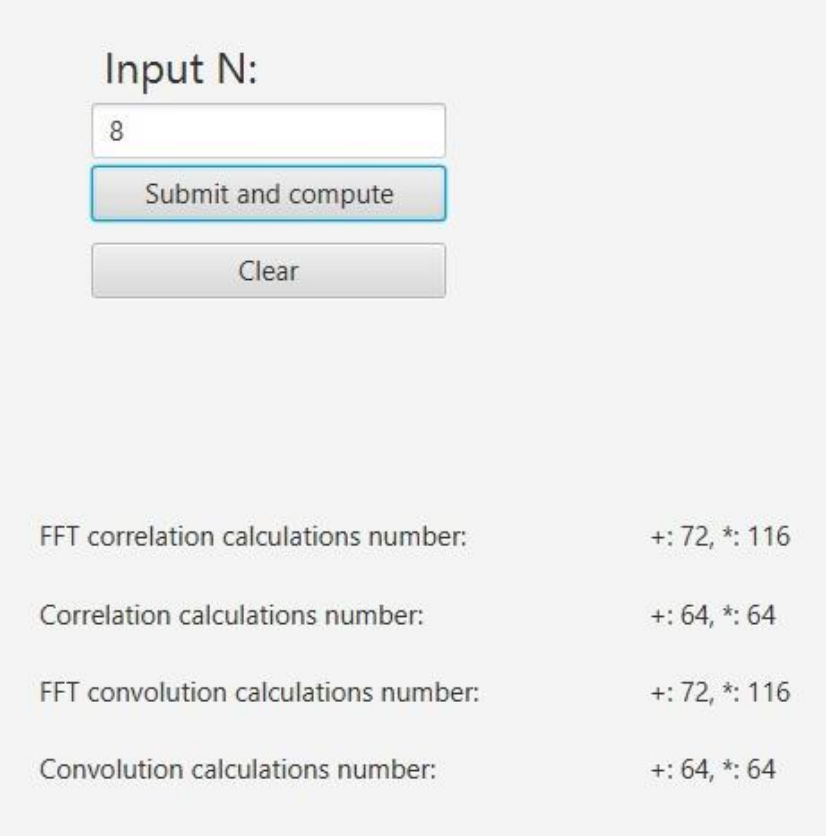

Рис. 1.8. Отображения информации о корреляции и свертке при N = 8

Пример при N=256

N	Операций (Корреляция)	Операций (Корреляция БПФ)	Эффективность БПФ
256	131072	15616	8.39 **:** 1
	Операций (Свертка)	**Операций (Свертка БПФ)**	**Эффективность БПФ**
	131072	188	8.39 **:** 1

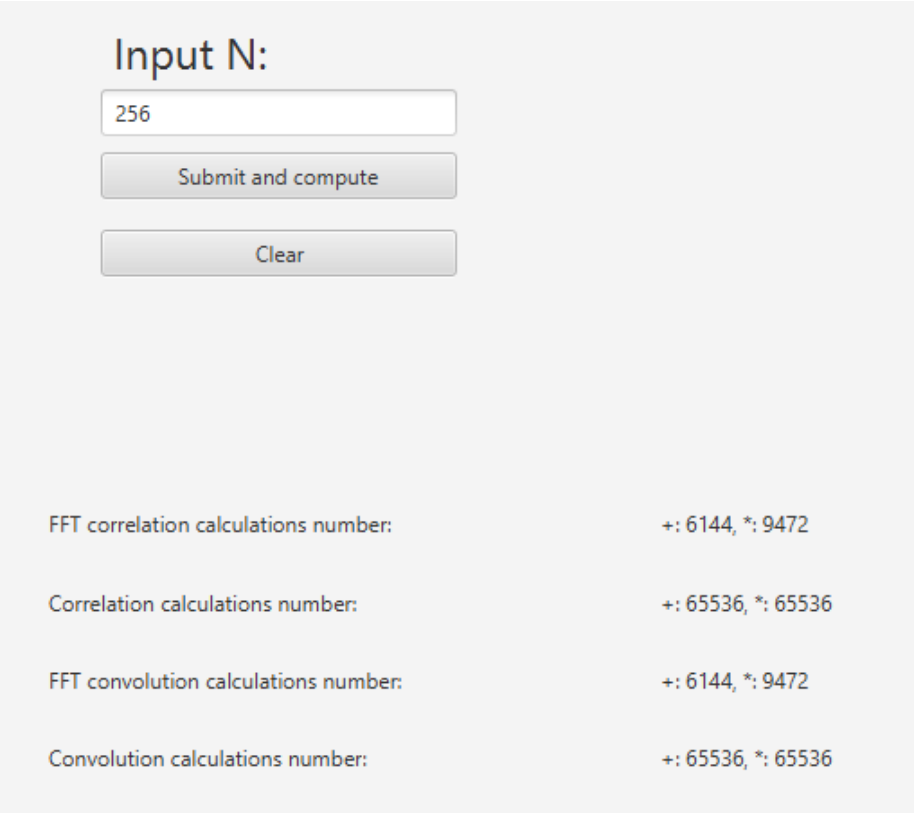

Рис. 1.8. Отображения информации о корреляции и свертке при N = 256

6. Вывод

В методике преобразования Фурье корреляция и свертка - очень важные операции. Свертка связывает три сигнала: входной сигнал, выходной сигнал и импульсную характеристику. Импульсная характеристика – сигнал, с помощью которого описываются системы (пользуясь стратегией импульсного разложения). Импульсное разложение представляет собой способ поточечного анализа сигнала.

Корреляция так же, как свертка, использует два сигнала для получения третьего. Этот третий сигнал называется корреляционным сигналом двух входных сигналов.

Корреляция между двумя непрерывными функциями f(x) и g(x) определяется как:

$$f(x) \circ g(x) = \int_{-\infty}^{+\infty} f(\alpha) g(x + \alpha) d\alpha$$

где α- временная переменная для интегрирования. Корреляция называется автокорреляцией если $f(x) = g(x)$ и взаимной корреляцией в противном случае.

Свертка по определению есть:

$$f(x) * g(x) = \int_{-\infty}^{+\infty} f(\beta) g(x - \beta) d\beta$$

Формы корреляции и свертки похожи, существует только одно различие между ними, показанное выше. При свертке g(x) сперва свертывается по вертикальной оси, а затем перемещается по x для получения g(x- β). Затем эта функция умножается на f(β) и интегрируется.

Вычисление корреляционных функций при помощи БПФ является, особенно для длинных числовых рядов, в десятки и сотни раз более быстрым методом, чем последовательными сдвигами во временной области при больших интервалах корреляции.

ПРИЛОЖЕНИЕ А

Листинг кода

main.java

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;
import java.net.URL;

public class FourierApplication extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        URL fxmlURL = getClass().getClassLoader().getResource("main.fxml");
        if (fxmlURL != null) {
            try {
                Parent root = FXMLLoader.load(fxmlURL);
                primaryStage.setTitle("Fourier transform");
                primaryStage.setScene(new Scene(root));
                primaryStage.show();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Can not load fxml file");
        }
    }
}
```

FourierService.java

```
package com.krealll.fourier.service;

import com.krealll.fourier.model.ComplexNumber;
import com.krealll.fourier.model.OperationCounter;
import com.krealll.fourier.model.TransformParameter;

import static java.lang.Math.*;

public class FourierService {

    public ComplexNumber[] discreteFourierTransform(int dir){
        ComplexNumber[] result = new ComplexNumber[TransformParameter.getN()];
        for (int k = 0; k < TransformParameter.getN() ; k++) {
            result[k] = new ComplexNumber(0,0);
            for (int m = 0; m < TransformParameter.getN() ; m++) {
                double parameter = TransformParameter.PERIOD/ TransformParameter.getN();
                ComplexNumber complexNumber = new ComplexNumber(cos(parameter*k*m),dir *
sin(parameter*k*m));
                double x = TransformParameter.FUNCTION.apply(parameter*m);
                result[k] = result[k].plus(complexNumber.times(x));
                OperationCounter.incAll();
            }
            if (dir == -1) {
                result[k] = result[k].divides(TransformParameter.getN());
            }
        }
        return result;
    }

    public ComplexNumber[] fastFourierTransform(){
        ComplexNumber[] functionNumbers = new ComplexNumber[TransformParameter.getN()];
        for (int i = 0; i < TransformParameter.getN(); i++) {
            functionNumbers[i] = new ComplexNumber(TransformParameter
                    .FUNCTION.apply(TransformParameter.PERIOD*(double)i/
TransformParameter.getN()));
        }
        ComplexNumber[] result = computeFFT(functionNumbers, 1);
        for (int i = 0; i < TransformParameter.getN() ; i++) {
            result[i] = result[i].divides(TransformParameter.getN());
        }
        return result;
    }

    public double[] inverseFourierTransform(ComplexNumber[] numbers){
        double[] result = new double[numbers.length];
        for (int i = 0; i <numbers.length ; i++) {
            result[i] = 0;
        }
        for (int k = 0; k <numbers.length; k++) {
            for (int m = 0; m <numbers.length ; m++) {
                double parameter = TransformParameter.PERIOD/ TransformParameter.getN();
                ComplexNumber W = new ComplexNumber(cos(parameter*(-
m*k)),sin(parameter*(-m*k)));
                result[k] += numbers[m].times(W).re();
            }
        }
```

```
        return result;
    }

    public ComplexNumber[] computeFFT(ComplexNumber[] array, int dir){
        if(array.length==1){
            return array;
        }
        ComplexNumber[] even = new ComplexNumber[array.length/2];
        ComplexNumber[] odd = new ComplexNumber[array.length/2];
        for (int i = 0; i <array.length ; i++) {
            if(i%2==0){
                even[i/2] = array[i];
            } else {
                odd[i/2] = array[i];
            }
        }
        ComplexNumber[] evenResult = computeFFT(even,dir);
        ComplexNumber[] oddResult = computeFFT(odd,dir);
        ComplexNumber[] result = new ComplexNumber[array.length];
        ComplexNumber WN = new
ComplexNumber(cos(TransformParameter.PERIOD/array.length),
                dir * sin(TransformParameter.PERIOD/array.length));
        ComplexNumber w = new ComplexNumber(1.0,0.0);
        for (int i = 0; i <array.length/2; i++) {
            result[i] = evenResult[i].plus(w.times(oddResult[i]));
            result[i+array.length/2] = evenResult[i].minus(w.times(oddResult[i]));
            w = w.times(WN);
            OperationCounter.incAll();
            OperationCounter.incAll();
            OperationCounter.incMul();
        }
        return result;
    }
}
```

DSPService.java

```
package com.krealll.fourier.service;

import com.krealll.fourier.model.ComplexNumber;
import com.krealll.fourier.model.OperationCounter;
import com.krealll.fourier.model.TransformParameter;

import java.util.function.BinaryOperator;
import java.util.function.Function;

public class DSPService {

    private final FourierService fourierService = new FourierService();

    private ComplexNumber[] initComplex(Function<Double,Double> func){
        ComplexNumber[] result = new ComplexNumber[TransformParameter.getN()];
        for (int i = 0; i < TransformParameter.getN(); i++) {
            double x = TransformParameter.PERIOD*(double)i/ TransformParameter.getN();
            result[i] = new ComplexNumber(func.apply(x),0.0);
        }
        return result;
    }

    private double[] init(Function<Double,Double> func){
        double[] result = new double[TransformParameter.getN()];
        for (int i = 0; i < TransformParameter.getN(); i++) {
            double x = TransformParameter.PERIOD*(double)i/ TransformParameter.getN();
            result[i] = func.apply(x);
        }
        return result;
    }


    public double[] computeTransform(BinaryOperator<Integer> func){
        double[] sinNums = init(TransformParameter.SIN_FUNCTION);
        double[] cosNums = init(TransformParameter.COS_FUNCTION);
        double[] result = new double[TransformParameter.getN()];
        for (int m = 0; m < TransformParameter.getN(); m++) {
            double Zn = 0.0;
            for (int h = 0; h < TransformParameter.getN(); h++) {
                Zn += sinNums[h] * cosNums[func.apply(m,h)];
                OperationCounter.incAll();
            }
            result[m] = Zn / TransformParameter.getN();
        }
        return result;
    }

    public double[] computeFFTCorrelation(){
        ComplexNumber[] sinNum = initComplex(TransformParameter.SIN_FUNCTION);
        ComplexNumber[] cosNum = initComplex(TransformParameter.COS_FUNCTION);

        ComplexNumber[] Cx = fourierService.computeFFT(sinNum,-1);
        ComplexNumber[] Cy = fourierService.computeFFT(cosNum,-1);
        ComplexNumber[] Cz = new ComplexNumber[TransformParameter.getN()];
        ComplexNumber[] Z ;
        double[] result = new double[TransformParameter.getN()];
```

```
        for (int i = 0; i < TransformParameter.getN(); i++) {
            ComplexNumber CxConjugate = Cx[i].conjugate();
            Cx[i] = CxConjugate.divides(TransformParameter.getN());
            Cy[i] = Cy[i].divides(TransformParameter.getN());
            Cz[i]=Cx[i].times(Cy[i]);
            OperationCounter.incMul();
        }

        Z = fourierService.computeFFT(Cz,1);
        for (int i = 0; i < TransformParameter.getN(); i++) {
            result[i] = Z[i].re();
        }
        return result;
    }

    public double[] computeFFTConvolution(){
        ComplexNumber[] sinNum = initComplex(TransformParameter.SIN_FUNCTION);

        ComplexNumber[] cosNum = initComplex(TransformParameter.COS_FUNCTION);

        ComplexNumber[] Cx = fourierService.computeFFT(sinNum,-1);

        ComplexNumber[] Cy = fourierService.computeFFT(cosNum,-1);

        ComplexNumber[] Cz = new ComplexNumber[TransformParameter.getN()];
        ComplexNumber[] Z;
        double[] result = new double[TransformParameter.getN()];
        for (int i = 0; i < TransformParameter.getN(); i++) {
            Cx[i] = Cx[i].divides(TransformParameter.getN());
            Cy[i] = Cy[i].divides(TransformParameter.getN());
            Cz[i]=Cx[i].times(Cy[i]);
            OperationCounter.incMul();
        }

        Z = fourierService.computeFFT(Cz,1);

        for (int i = 0; i < TransformParameter.getN(); i++) {
            result[i] = Z[i].re();
        }
        return result;
    }

}
```

TransformParametr.java

```
package com.krealll.fourier.model;

import java.util.function.BinaryOperator;
import java.util.function.Function;
import static java.lang.Math.*;

public class TransformParameter {

    private static int N;
    public static final double PERIOD = 2*PI;
    public static final Function<Double,Double> FUNCTION = (x) -> sin(x) + cos(x);
    public static final Function<Double,Double> SIN_FUNCTION = Math::sin;
    public static final Function<Double,Double> COS_FUNCTION = Math::cos;
    public static final BinaryOperator<Integer> CORRELATION = (x,y)
            -> (x + y + TransformParameter.getN()) % TransformParameter.getN();
    public static final BinaryOperator<Integer> CONVOLUTION = (x,y)
            -> (x - y + TransformParameter.getN()) % TransformParameter.getN();
    private TransformParameter(){}

    public static int getN() {
        return N;
    }

    public static void setN(int n) {
        N = n;
    }
}
```

OperationCounter.java

```
package com.krealll.fourier.model;

public class OperationCounter {
    private static int mulCounter = 0;
    private static int plusCounter = 0;

    public static void incPlus(){
        plusCounter++;
    }

    public static void incMul(){
        mulCounter++;
    }

    public static void incAll(){
        plusCounter++;
        mulCounter++;
    }

    public static void nullAll(){
        mulCounter = 0;
        plusCounter = 0;
    }

    public static void nullPlus(){
        plusCounter = 0;
    }

    public static void nullMul(){
        mulCounter = 0;
    }

    public static int getMulCounter() {
        return mulCounter;
    }

    public static int getPlusCounter() {
        return plusCounter;
    }
}
```

ComplexNumber.java

```
package com.krealll.fourier.model;

public class ComplexNumber {
    private final double re;
    private final double im;

    private static final double DELTA = 0.000001;

    public ComplexNumber(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public ComplexNumber(double real) {
        re = real;
        im = 0.0;
    }

    public double abs() {
        return Math.hypot(re, im);
    }

    public double phase() {

        return Math.atan(im / re);
    }

    public ComplexNumber plus(ComplexNumber b) {
        ComplexNumber a = this;
        double real = a.re + b.re;
        double imag = a.im + b.im;

        return new ComplexNumber(real, imag);
    }

    public ComplexNumber minus(ComplexNumber b) {
        ComplexNumber a = this;
        double real = a.re - b.re;
        double imag = a.im - b.im;

        return new ComplexNumber(real, imag);
    }

    public ComplexNumber times(ComplexNumber b) {
        ComplexNumber a = this;
        double real = a.re * b.re - a.im * b.im;
        double imag = a.re * b.im + a.im * b.re;

        return new ComplexNumber(real, imag);
    }

    public ComplexNumber times(double alpha) {
        return new ComplexNumber(alpha * re, alpha * im);
    }

    public ComplexNumber conjugate() {
```

```
        return new ComplexNumber(re, -im);
    }

    public ComplexNumber reciprocal() {
        double scale = re * re + im * im;
        return new ComplexNumber(re / scale, -im / scale);
    }

    public double re() {
        return re;
    }

    public double im() {
        return im;
    }

    public ComplexNumber divides(int param) {
        return new ComplexNumber(this.re / param, this.im / param);
    }

    public ComplexNumber sin() {
        return new ComplexNumber(Math.sin(re) * Math.cosh(im), Math.cos(re) *
Math.sinh(im));
    }

    public ComplexNumber cos() {
        return new ComplexNumber(Math.cos(re) * Math.cosh(im), -Math.sin(re) *
Math.sinh(im));
    }

    public static ComplexNumber plus(ComplexNumber a, ComplexNumber b) {
        double real = a.re + b.re;
        double imag = a.im + b.im;

        return new ComplexNumber(real, imag);
    }

    private static ComplexNumber normalize(ComplexNumber a) {
        return new ComplexNumber((Math.abs(a.re) < DELTA)?0:a.re, (Math.abs(a.im) <
DELTA)?0:a.im);
    }

    public String toString() {
        if (im == 0) {
            return re + "";
        }
        if (re == 0) {
            return im + "i";
        }
        if (im < 0) {
            return re + " - " + (-im) + "i";
        }
        return re + " + " + im + "i";
    }
}
```

MainController.java

```
package com.krealll.fourier.controller;

import com.krealll.fourier.service.DSPService;
import com.krealll.fourier.model.OperationCounter;
import com.krealll.fourier.model.TransformParameter;
import javafx.fxml.FXML;
import javafx.scene.chart.LineChart;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class MainController  {

    private final static String NUMBER_FORMAT =
"(1|2|4|8|16|32|64|128|256|512|1024|2048|4096|8192|16384|32768|65536)";
    private final static Pattern pattern = Pattern.compile(NUMBER_FORMAT);
    private int displayedTimesCalculations;
    private int displayedPlusCalculations;
    @FXML
    Button computeButton;
    @FXML
    TextField inputN;
    @FXML
    Button clearButton;


    @FXML
    LineChart<Number,Number> cosX;
    @FXML
    LineChart<Number,Number> sinX;
    @FXML
    LineChart<Number,Number> Convolution ;
    @FXML
    LineChart<Number,Number> FFTConvolution;
    @FXML
    LineChart<Number,Number> Correlation;
    @FXML
    LineChart<Number,Number> FFTCorrelation;

    @FXML
    Label FFTCorCalcLabel;
    @FXML
    Label FFTCorCalc;
    @FXML
    Label CorCalcLabel;
    @FXML
    Label CorCalc;
    @FXML
    Label FFTConvCalsLabel;
    @FXML
    Label FFTConvCals;
    @FXML
```

```
    Label ConvCalsLabel;
    @FXML
    Label ConvCals;

    public void initialize(){
        computeButton.setOnMouseClicked((MouseEvent event) ->{
            String someInput = inputN.getText();
            Matcher matcher = pattern.matcher(someInput);
            if(matcher.matches()){
                TransformParameter.setN(Integer.parseInt(someInput));
                ChartCreator.showCos(cosX);
                ChartCreator.showSin(sinX);
                operateConv();
                ConvCals.setText("+: "+displayedPlusCalculations+", *:
"+displayedTimesCalculations);
                ConvCalsLabel.setVisible(true);
                ConvCals.setVisible(true);
                OperationCounter.nullAll();
                operateFFTConv();
                FFTConvCals.setText("+: "+displayedPlusCalculations+", *:
"+displayedTimesCalculations);
                FFTConvCalsLabel.setVisible(true);
                FFTConvCals.setVisible(true);
                OperationCounter.nullAll();
                operateCorr();
                CorCalc.setText("+: "+displayedPlusCalculations+", *:
"+displayedTimesCalculations);
                CorCalcLabel.setVisible(true);
                CorCalc.setVisible(true);
                OperationCounter.nullAll();
                operateFFTCorr();
                FFTCorCalc.setText("+: "+displayedPlusCalculations+", *:
"+displayedTimesCalculations);
                FFTCorCalcLabel.setVisible(true);
                FFTCorCalc.setVisible(true);
                OperationCounter.nullAll();
            }
        });
        clearButton.setOnMouseClicked((MouseEvent event) ->{
            FFTCorCalcLabel.setVisible(false);
            CorCalcLabel.setVisible(false);
            FFTConvCalsLabel.setVisible(false);
            ConvCalsLabel.setVisible(false);
            ConvCals.setVisible(false);
            FFTConvCals.setVisible(false);
            CorCalc.setVisible(false);
            FFTCorCalc.setVisible(false);
            displayedTimesCalculations = 0;
            displayedPlusCalculations = 0;
            cosX.getData().clear();
            sinX.getData().clear();
            Convolution.getData().clear();
            FFTConvolution.getData().clear();
            Correlation.getData().clear();
            FFTCorrelation.getData().clear();
        });
    }

    private void operateConv(){
```

```
            DSPService dspService = new DSPService();
            double[] result = dspService.computeTransform(TransformParameter.CONVOLUTION);
           // logResult(result,"Convolution");
            displayedPlusCalculations = OperationCounter.getPlusCounter();
            displayedTimesCalculations = OperationCounter.getMulCounter();
            ChartCreator.showChart(Convolution,result);
        }
        private void operateFFTConv(){
            DSPService dspService = new DSPService();
            double[] result = dspService.computeFFTConvolution();
            //logResult(result,"FFT Convolution");
            displayedPlusCalculations = OperationCounter.getPlusCounter();
            displayedTimesCalculations = OperationCounter.getMulCounter();
            ChartCreator.showChart(FFTConvolution,result);
        }
        private void operateCorr(){
            DSPService dspService = new DSPService();
            double[] result = dspService.computeTransform(TransformParameter.CORRELATION);
            //logResult(result,"Correlation");
            displayedPlusCalculations = OperationCounter.getPlusCounter();
            displayedTimesCalculations = OperationCounter.getMulCounter();
            ChartCreator.showChart(Correlation,result);
        }

        private void operateFFTCorr(){
            DSPService dspService = new DSPService();
            double[] result = dspService.computeFFTCorrelation();
           // logResult(result,"FFT Correlation");
            displayedPlusCalculations = OperationCounter.getPlusCounter();
            displayedTimesCalculations = OperationCounter.getMulCounter();
            ChartCreator.showChart(FFTCorrelation,result);
        }

        private void logResult(double[] numbers, String name){
            System.out.println(name);
            for (int i = 0; i < TransformParameter.getN() ; i++) {
                System.out.println("Number["+i+"]="+numbers[i]);
            }
        }


    }
```

ChartCreator.java

```
package com.krealll.fourier.controller;

import com.krealll.fourier.model.ComplexNumber;
import com.krealll.fourier.model.TransformParameter;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.ScatterChart;
import javafx.scene.chart.XYChart;
import javafx.scene.chart.XYChart.Series;

public class ChartCreator {



    public static void showChart(LineChart<Number,Number> chart){
        chart.setLegendVisible(false);
        chart.setAnimated(false);
        Series series = new Series();
        for (int i = 0; i < TransformParameter.getN(); i++) {
            double x = TransformParameter.PERIOD*(double)i/ TransformParameter.getN();
            series.getData().add(new XYChart.Data<>(x,
TransformParameter.FUNCTION.apply(x)));
        }
        chart.getData().add(series);
    }

    public static void showSin(LineChart<Number,Number> chart){
        chart.setLegendVisible(false);
        chart.setAnimated(false);
        Series series = new Series();
        for (int i = 0; i < TransformParameter.getN(); i++) {
            double x = TransformParameter.PERIOD*(double)i/ TransformParameter.getN();
            series.getData().add(new XYChart.Data<>(x,
TransformParameter.SIN_FUNCTION.apply(x)));
        }
        chart.getData().add(series);
    }

    public static void showCos(LineChart<Number,Number> chart){
        chart.setLegendVisible(false);
        chart.setAnimated(false);
        Series series = new Series();
        for (int i = 0; i < TransformParameter.getN(); i++) {
            double x = TransformParameter.PERIOD*(double)i/ TransformParameter.getN();
            series.getData().add(new XYChart.Data<>(x,
TransformParameter.COS_FUNCTION.apply(x)));
        }
        chart.getData().add(series);
    }

    public static void showChart(LineChart<Number,Number> chart,double[] numbers){
        chart.setLegendVisible(false);
        chart.setAnimated(false);
        double[] xValues = new double[numbers.length];
        for (int i = 0; i < TransformParameter.getN(); i++) {
            xValues[i] = TransformParameter.PERIOD*(double)i/ TransformParameter.getN();
        }
```

```
        Series series = new Series();
        for (int i = 0; i < TransformParameter.getN(); i++) {
            series.getData().add(new XYChart.Data<>(xValues[i],numbers[i]));
        }
        chart.getData().add(series);
    }

    public static void showAmplitude(LineChart<Number,Number> chart, ComplexNumber[]
numbers){
        chart.setLegendVisible(false);
        chart.setAnimated(false);
        Series series = new Series();
        for (int i = -TransformParameter.getN()+1, j = TransformParameter.getN()-1; i
<0; j--, i++) {
            series.getData().add(new XYChart.Data<>(i,numbers[j].abs()));
        }
        for (int i = 0; i < TransformParameter.getN(); i++) {
            series.getData().add(new XYChart.Data<>(i,numbers[i].abs()));
        }
        chart.getData().add(series);
    }

    public static void showPhase(ScatterChart<Number,Number> chart, ComplexNumber[]
numbers){
        chart.setLegendVisible(false);
        chart.setAnimated(false);
        Series series = new Series();
        for (int i = -TransformParameter.getN()+1, j = TransformParameter.getN()-1; i
<0; j--, i++) {
            series.getData().add(new XYChart.Data<>(i,-numbers[j].phase()));
        }
        for (int i = 0; i < TransformParameter.getN(); i++) {
            series.getData().add(new XYChart.Data<>(i,numbers[i].phase()));
        }
        chart.getData().add(series);
    }


}
```