Zadanie 1

Napisz funkcję realizującą Dyskretną Transformatę Fouriera.

```java
public class DFT {

    public List<Pair<Double, Double>> dft(ChartDetails signal) {
        List<Pair<Double, Double>> scores = new ArrayList<>();
        for (int k = 0; k < signal.getValues().size(); ++k) {
            double real = 0.0;
            double imaginary = 0.0;
            for (int n = 0; n < signal.getValues().size(); ++n) {
                double wn = 2 * Math.PI * n * k;
                real += signal.getValues().get(n) * cos(wn / signal.getValues().size());
                imaginary += signal.getValues().get(n) * sin(wn / signal.getValues().size()) * (-1);
            }
            Pair<Double, Double> pair = new Pair<>(real, imaginary);
            scores.add(pair);
        }
        return scores;
    }
```

Zadanie 2

Użyj funkcji z poprzednich zajęć i wyznacz dyskretny sygnał tonu prostego $x(n)$. Wygeneruj wykres dla $n \in < 0; ABC >$, jako parametry inicjalizujące przyjmij: $A = 1.0[V]$, $f = B[Hz]$, $\varphi = C \times \pi[rad]$.
Użyj przekształcenia DFT z zadania pierwszego i dla uzyskanej reprezentacji sygnału $x(n)$ w dziedzinie czasu wyznacz sygnał w dziedzinie częstotliwości $X(k)$.
Oblicz widmo amplitudowe jako $M(k) = \sqrt{Re[X(k)]^2 + Im[X(k)]^2}$
Wartość amplitudy przedstawić w sakli decybelowej $M'(k) = 10 \times \log_{10} M(k)$
Wyznacz skalę częstotliwości $f_k = k \times \frac{f_s}{N}$
Wykreślić wykres widma amplitudowego $M(k)$, (wartosci $f_k$oznaczają częstotliwości prążków widma.

Kod: DFT – widmo amplitudowe

```java
public ChartDetails makeAmplitude(List<Pair<Double, Double>> dft) {
    List<Double> scores = new ArrayList<>();
    for (Pair<Double, Double> pair : dft) {
        double real = pair.getKey();
        double imaginary = pair.getValue();
        double sum = 0;
        sum = sqrt(real * real + imaginary * imaginary);
        scores.add(sum);
    }
    return new ChartDetails( title: "DFT", scores, xAxisTitle: "Frequency", yAxisTitle: "Amplitude");
}
```

Kod: DFT – widmo amplitudowe w skali decybelowej

```
    public ChartDetails decibelScale(ChartDetails amplitude) {
        List<Double> scores = new ArrayList<>();

        for (int i = 0; i < amplitude.getValues().size(); ++i) {
            scores.add(10 * log10(amplitude.getValues().get(i)));
        }

        return new ChartDetails( title: "DFT spectrum with decibel scale", scores, xAxisTitle: "Frequency", yAxisTitle: "Amplitude (Decibel scale)");
    }
}
```

Kod: Skala częstości dla DFT wyrażonego w postaci widma amplitudowego w skali decybelowej liczona jest w momencie tworzenia wykresu

```
private LineChart<NumberAxis, NumberAxis> getLinechart(ChartDetails chartDetails, Pair<NumberAxis, NumberAxis> pairXYAxis, boolean useDec) {
    LineChart<NumberAxis, NumberAxis> lineChart = new LineChart(pairXYAxis.getKey(), pairXYAxis.getValue());
    XYChart.Series<NumberAxis, NumberAxis> dftSeries = new XYChart.Series<>();
    //chartDetails.getValues().forEach((y) -> dftSeries.getData().add(new XYChart.Data(y)));
    int x = 0;
    for (Double value : chartDetails.getValues()) {
        if (useDec) {
            double x1 = value * (7 / chartDetails.getValues().size());
            dftSeries.getData().add(new XYChart.Data(x1, value));
        } else {
            dftSeries.getData().add(new XYChart.Data(x, value));
            ++x;
        }
    }

    lineChart.getData().add(dftSeries);
    lineChart.setCreateSymbols(false);
    lineChart.setTitle(chartDetails.getTitle());
```

Zadanie 4

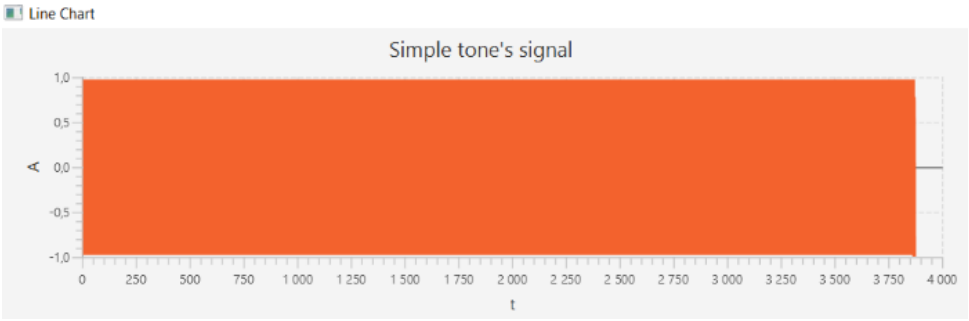
Napisz funkcję realizującą Odwrotną Dyskretną Transformatę Fouriera.
Zweryfikuj poprawność jej działania odwracając sygnał z dziedziny częstotliwości do dziedziny czasu (wykorzystaj sygnał użyty w zadaniu drugim).
Kod: IDFT

```
public class IDFT {

    public ChartDetails idft(List<Pair<Double, Double>> dft) {
        List<Double> scores = new ArrayList<>();

        for (int n = 0; n < dft.size(); n++) {
            double sum = 0;
            for (int k = 0; k < dft.size(); k++) {
                double real = dft.get(k).getKey();
                double imaginary = dft.get(k).getValue();
                double ws = 2 * Math.PI * k * n;
                sum += cos(ws / dft.size()) * real - sin(ws / dft.size()) * imaginary;
            }
            scores.add(sum/dft.size());
        }
        //for s(t)
        //return new ChartDetails("IDFT: ", scores, "t", "A");
        //for others
         return new ChartDetails( title: "IDFT: ", scores, xAxisTitle: "t[s]", yAxisTitle: "Function(t)");
    }
```

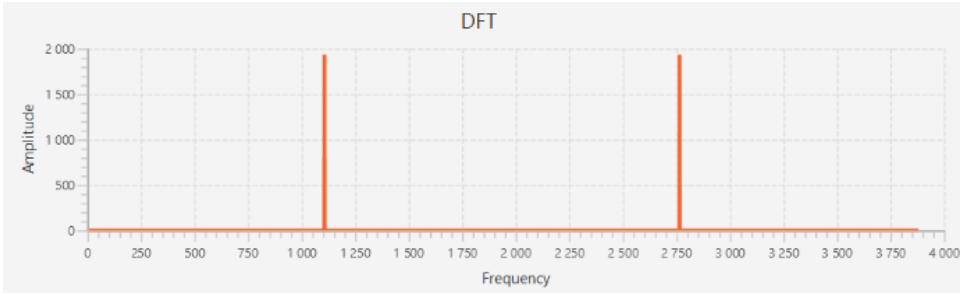
Zadanie 2/Zadanie 4

Użyj funkcji z poprzednich zajęć i wyznacz dyskretny sygnał tonu prostego $x(n)$.
Użyj przekształcenia DFT z zadania pierwszego i dla uzyskanej reprezentacji sygnału $x(n)$ w dziedzinie czasu wyznacz sygnał w dziedzinie częstotliwości $X(k)$.
Zweryfikuj poprawność IDFT odwracając sygnał z dziedziny częstotliwości do dziedziny czasu (wykorzystaj sygnał użyty w zadaniu drugim).

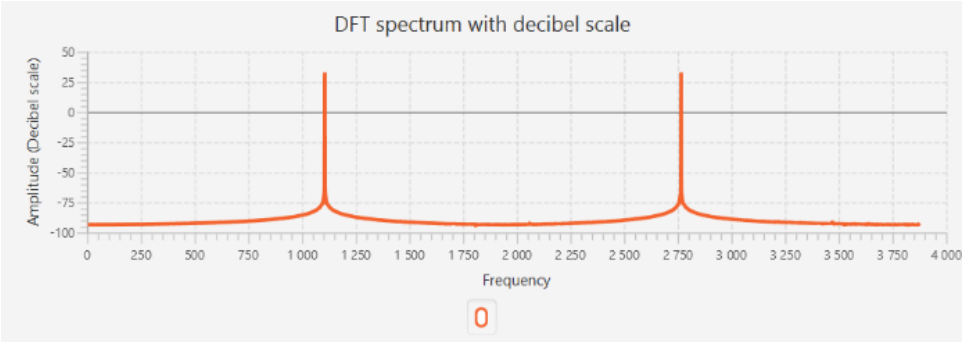
Sygnał tonu prostego (na wejściu)



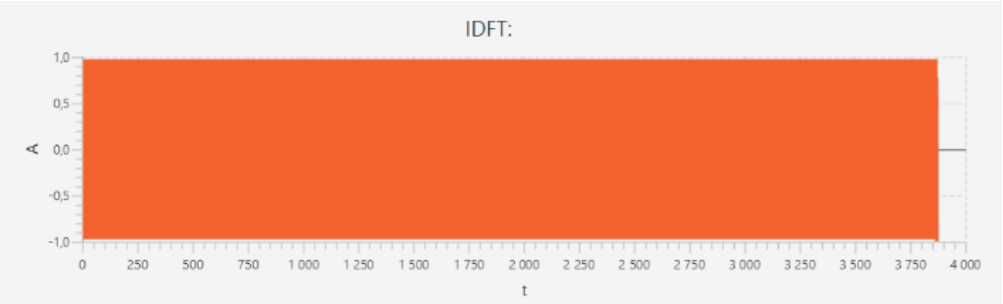
Sygnał tonu prostego – DFT (widmo amplitudowe)



Sygnał tonu prostego – DFT (widmo amplitudowe w skali decybelowej)



Sygnał tonu prostego – IDFT



Kod realizujący tworzenie tonu prostego:

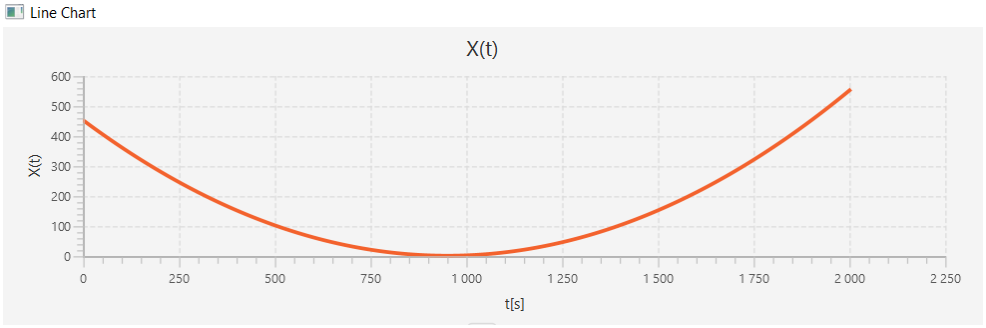
```
    public static ChartDetails makeSampling(double start, double stop, double fs) {
        final List<Double> scores = new ArrayList<>();
        double A = 1.0; //[V]
        double f = b; //[Hz]
        double fi = c * PI; //[rad]
        double s;
        for (double t = start; t <= stop; t += (1 / fs)) {
            s = A * sin(2 * PI * f * t + fi);
            scores.add(s);
        }
        return new ChartDetails( title: "Simple tone's signal", scores, xAxisTitle: "t", yAxisTitle: "A");
    }
}
```

Zadanie 3

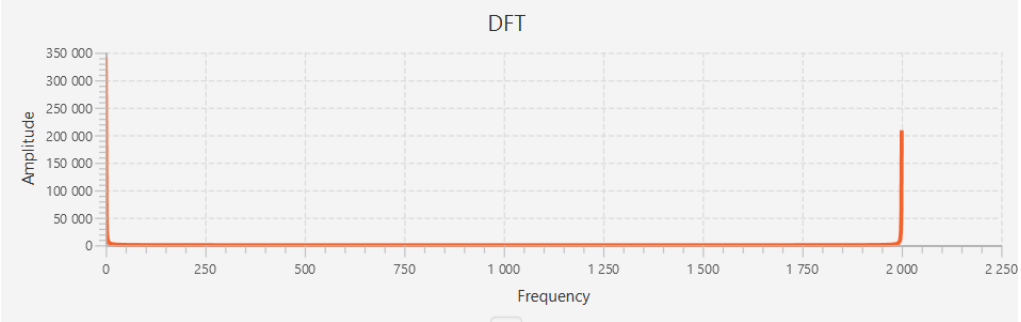
Dla sygnałów uzyskanych na pierwszych laboratoriach obliczyć widma amplitudowe . Należy tak dobrać skale (liniową lub logarytmiczną) osi pionowych i poziomych aby jak najwięcej prążków widma było widocznych na wykresie.

FUNKCJA X(t)

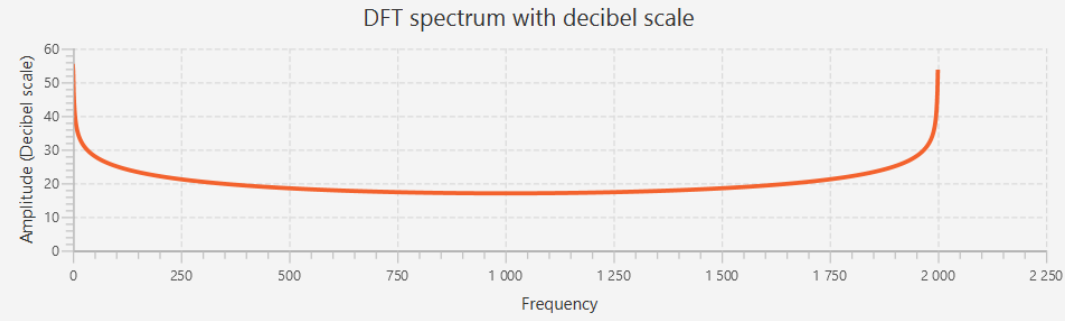
X(t) (na wejściu)



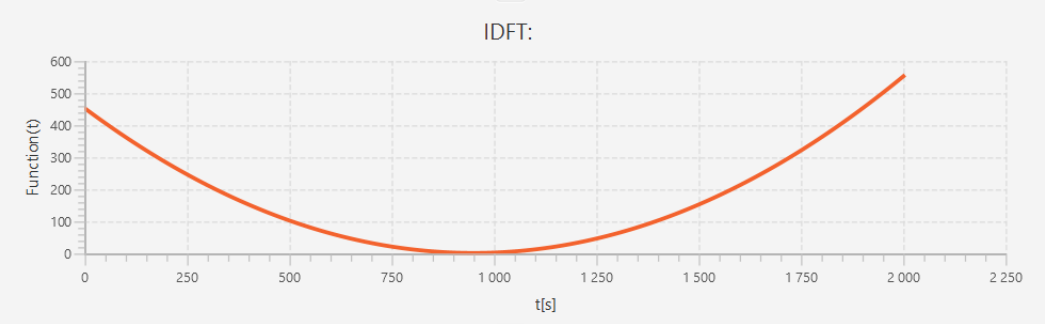
X(t) – DFT (widmo amplitudowe)



X(t) – DFT (widmo amplitudowe w skali decybelowej)



X(t) – IDFT

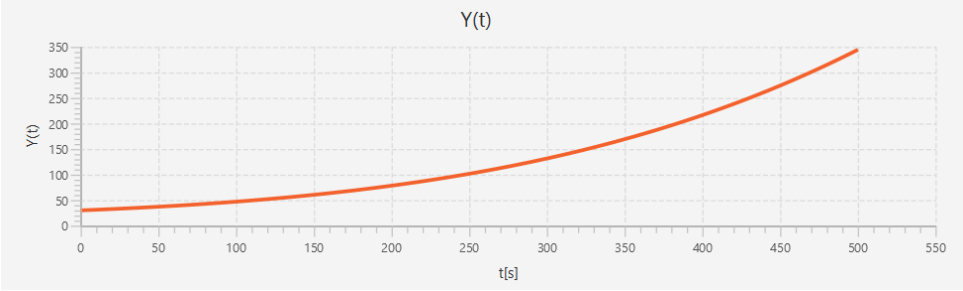


Kod realizujący tworzenie funkcji X(t):

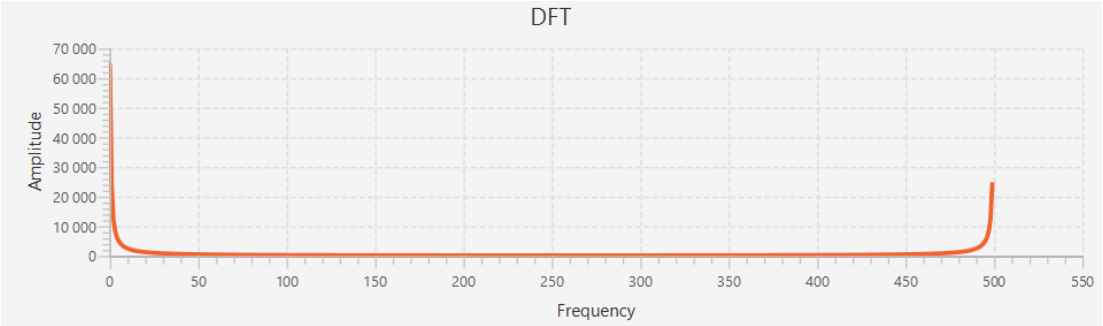
```
public static ChartDetails makeX(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    for (double t = start; t <= stop; t += (1 / step)) {
        double tmp = a * pow(t, 2) + b * t + c;
        scores.add(tmp);
    }
    return new ChartDetails( title: "X(t)", scores, xAxisTitle: "t[s]", yAxisTitle: "X(t)");
}
```

FUNKCJA Y(t)

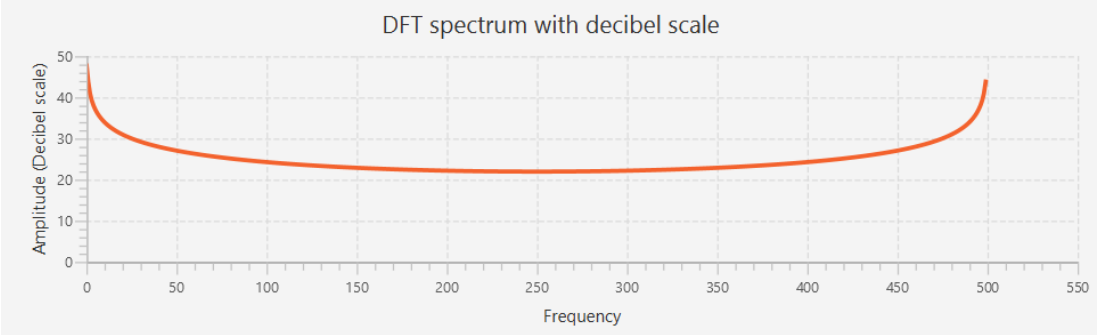
Y(t) (na wejściu)



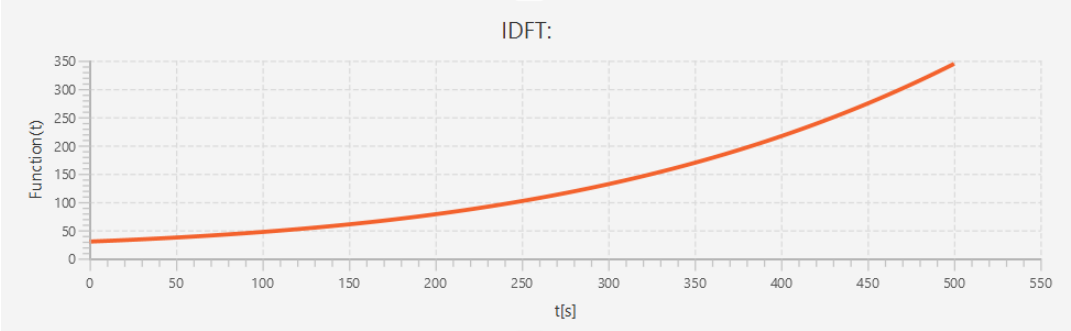
Y(t) – DFT (widmo amplitudowe)



Y(t) – DFT (widmo amplitudowe w skali decybelowej)



Y(t) – IDFT



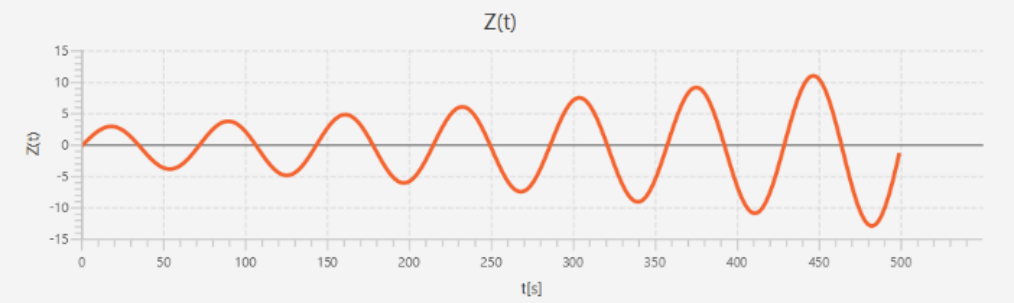
Kod realizujący tworzenie funkcji Y(t):

```
public static ChartDetails makeY(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    for (double t = start; t <= stop; t += (1 / step)) {
        double tmp = 2 * pow(countX(t), 2) + 12 * cos(t);
        scores.add(tmp);
    }
    return new ChartDetails( title: "Y(t)", scores, xAxisTitle: "t[s]", yAxisTitle: "Y(t)");
}
```

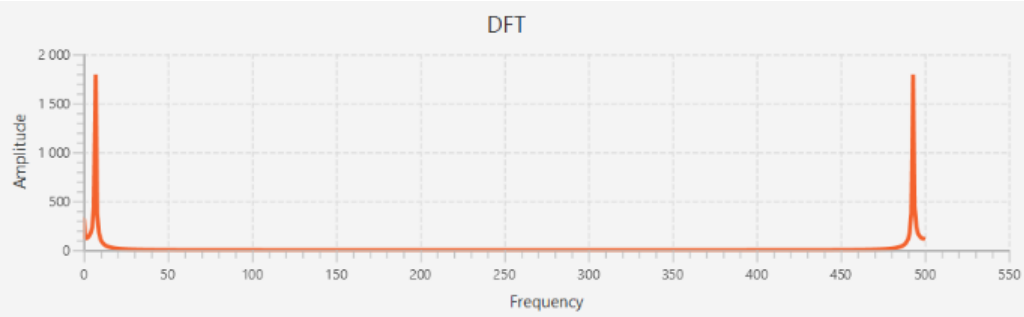
Aleksandra Karasińska
44355

FUNKCJA Z(t)

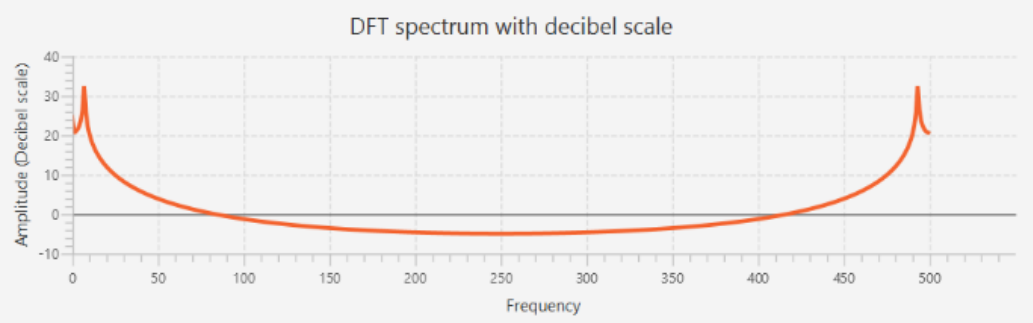
Z(t) (na wejściu)



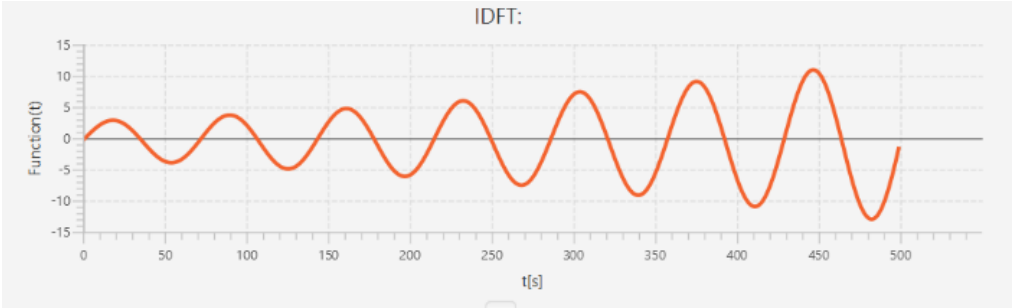
Z(t) – DFT (widmo amplitudowe)



Z(t) – DFT (widmo amplitudowe w skali decybelowej)



Z(t) – IDFT

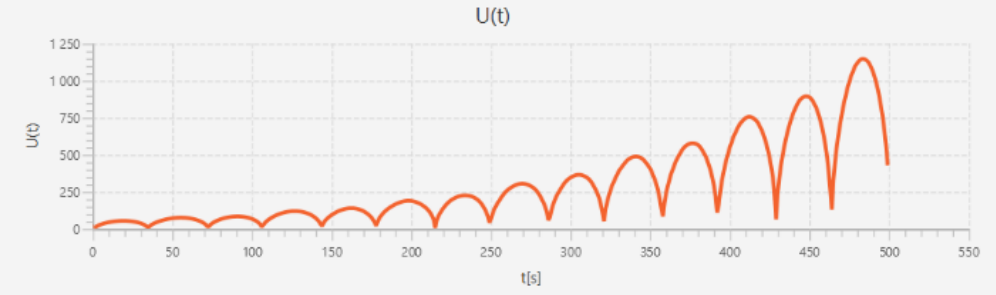


Kod realizujący tworzenie funkcji Z(t):

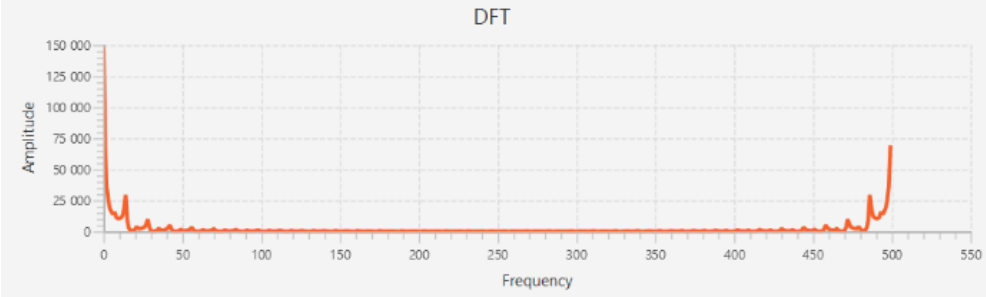
```
private static Double countZ(double t) {...}

public static ChartDetails makeX(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    for (double t = start; t <= stop; t += (1 / step)) {
        double tmp = a * pow(t, 2) + b * t + c;
        scores.add(tmp);
    }
    return new ChartDetails( title: "X(t)", scores, xAxisTitle: "t[s]", yAxisTitle: "X(t)");
}
```

FUNKCJA U(t)

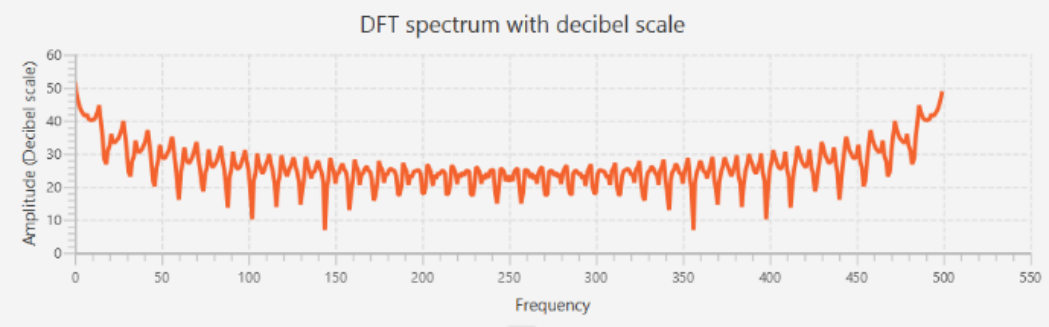
U(t) (na wejściu)



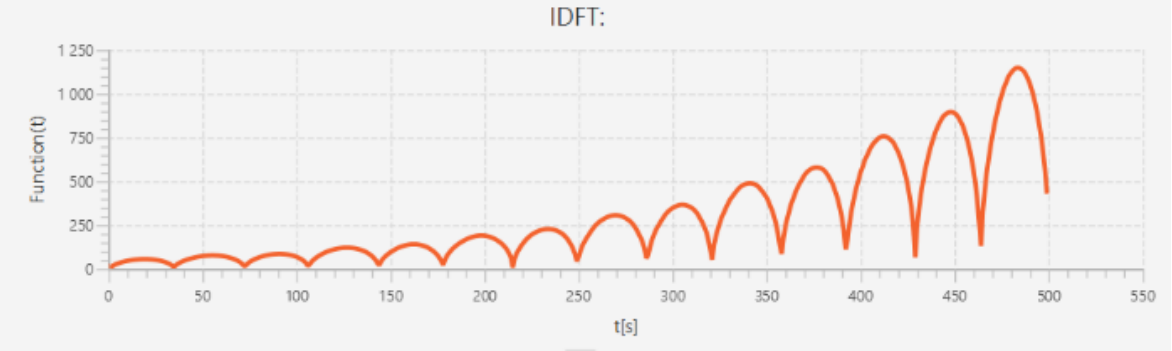
U(t) – DFT (widmo amplitudowe)



U(t) – DFT (widmo amplitudowe w skali decybelowej)



U(t) – IDFT

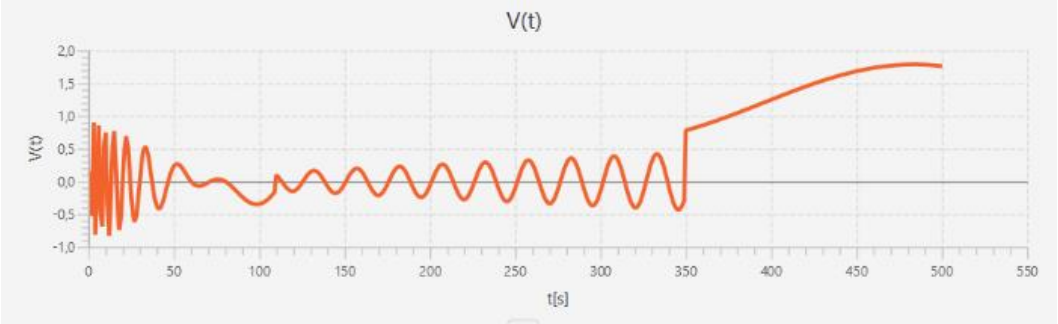


Kod realizujący tworzenie funkcji U(t):

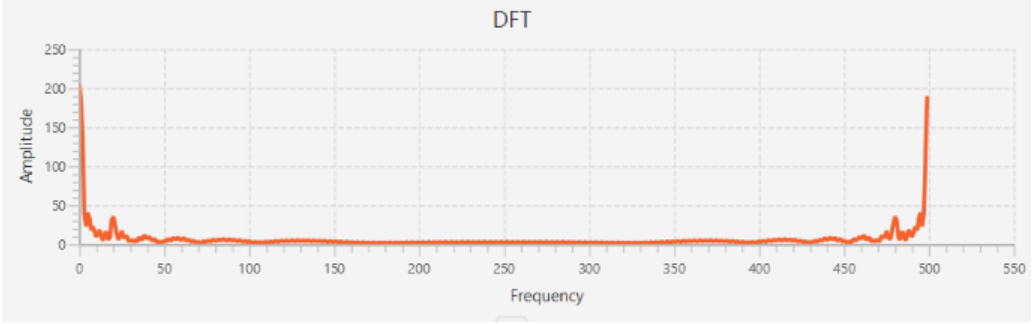
```
public static ChartDetails makeU(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    for (double t = start; t <= stop; t += (1 / step)) {
        double tmp = sqrt(abs(countY(t) * countY(t) * countZ(t))) - 1.8 * sin(0.4 * t * countZ(t) * countX(t));
        scores.add(tmp);
    }
    return new ChartDetails( title: "U(t)", scores, xAxisTitle: "t[s]", yAxisTitle: "U(t)");
}
```

FUNKCJA V(t)

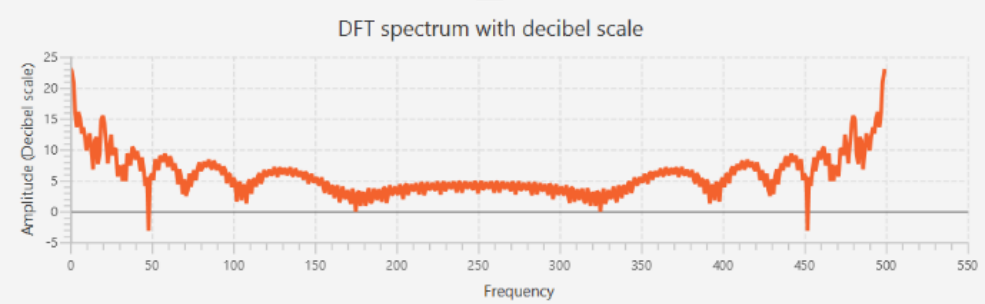
V(t) (na wejściu)



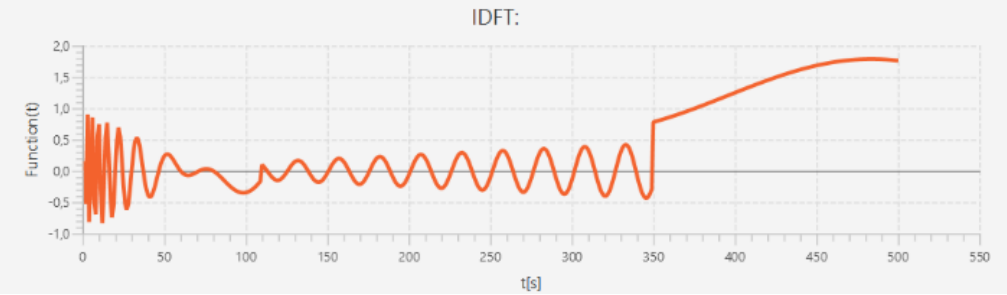
V(t) – DFT (widmo amplitudowe)



V(t) – DFT (widmo amplitudowe w skali decybelowej)



V(t) – IDFT

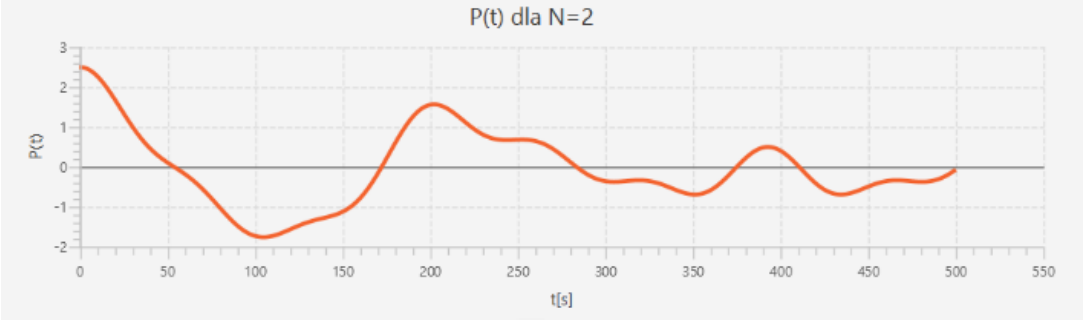


Kod realizujący tworzenie funkcji V(t):

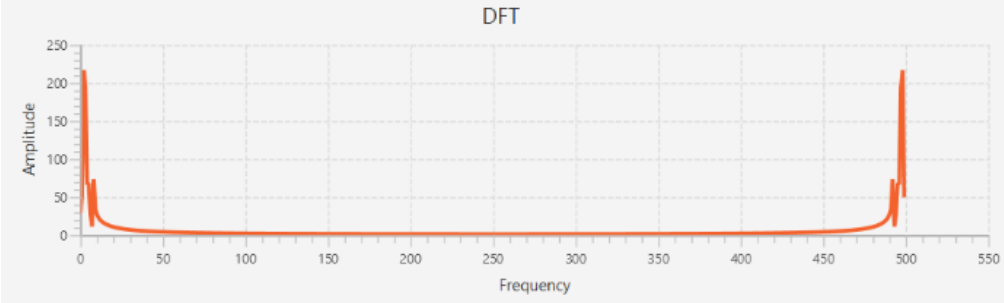
```
public static ChartDetails makeV(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    double tmp;
    for (double t = start; t <= stop; t += (1 / step)) {
        if (t >= 0 && t < 0.22) {
            tmp = (1 - 7 * t) * sin((2 * PI * t * 10) / (t + 0.04));
            scores.add(tmp);
        } else if (t >= 0.22 && t < 0.7) {
            tmp = 0.63 * t * sin(125 * t);
            scores.add(tmp);
        } else if (t <= 1 && t >= 0.7) {
            tmp = pow(t, -0.662) + 0.77 * sin(8 * t);
            scores.add(tmp);
        }
    }
    return new ChartDetails( title: "V(t)", scores, xAxisTitle: "t[s]", yAxisTitle: "V(t)");
}
```

FUNKCJA P(t) dla N = 2

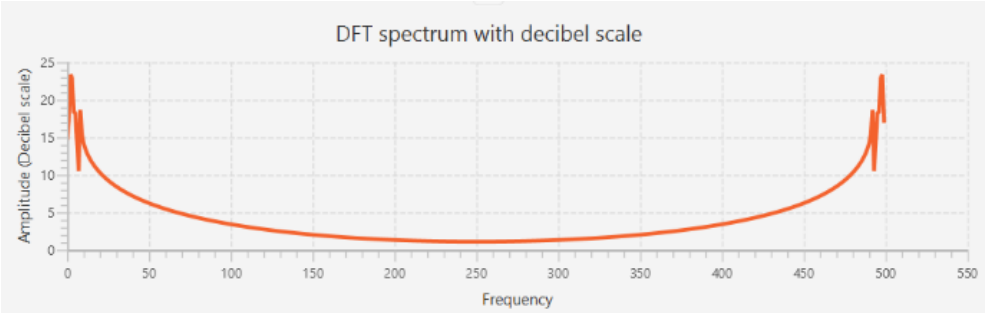
P(t) dla N = 2 (na wejściu)



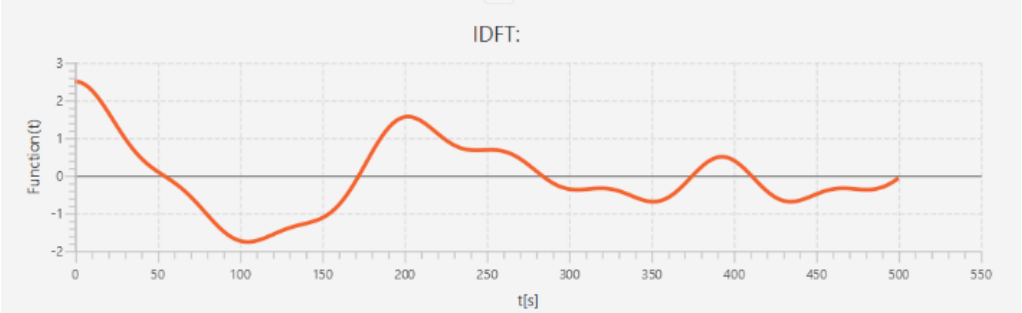
P(t) dla N = 2 – DFT (widmo amplitudowe)



P(t) dla N = 2 – DFT (widmo amplitudowe w skali decybelowej)



P(t) dla N = 2 – IDFT



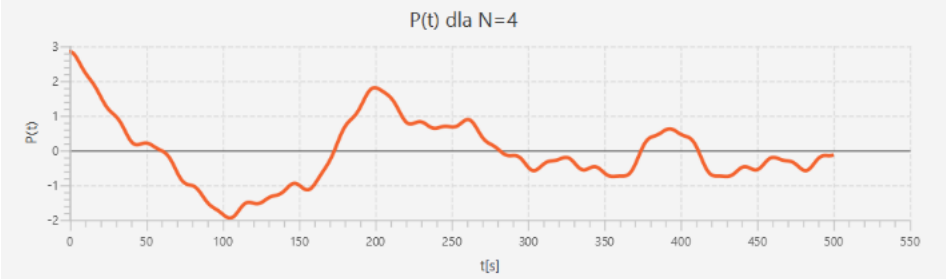
Kod realizujący tworzenie funkcji P(t) dla N = 2:

```
public static ChartDetails makeP2(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    for (double t = start; t <= stop; t += (1 / step)) {
        double tmp = 0;
        for (int n = 1; n <= 2; n++) {
            tmp += (cos(12 * t * n * n) + cos(16 * t * n)) / (n * n);
        }

        scores.add(tmp);
    }
    return new ChartDetails( title: "P(t) dla N=2", scores, xAxisTitle: "t[s]", yAxisTitle: "P(t)");
}
```

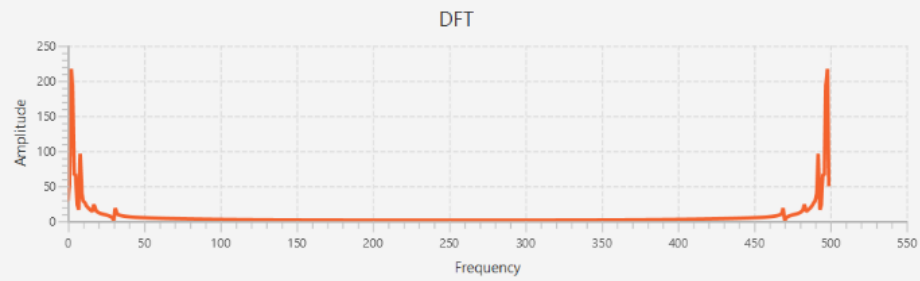
Aleksandra Karasińska
44355

FUNKCJA P(t) dla N = 4

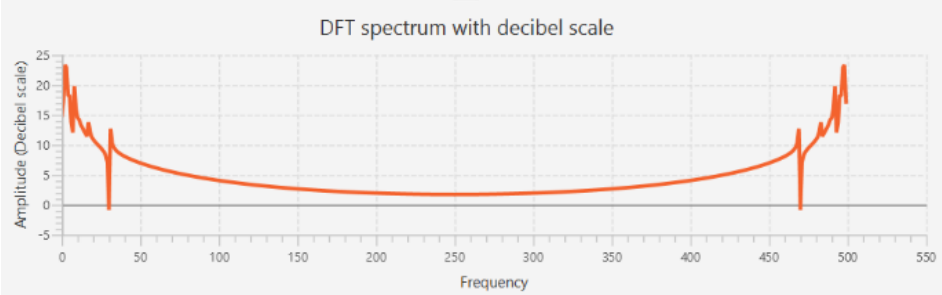
P(t) dla N = 4 (na wejściu)



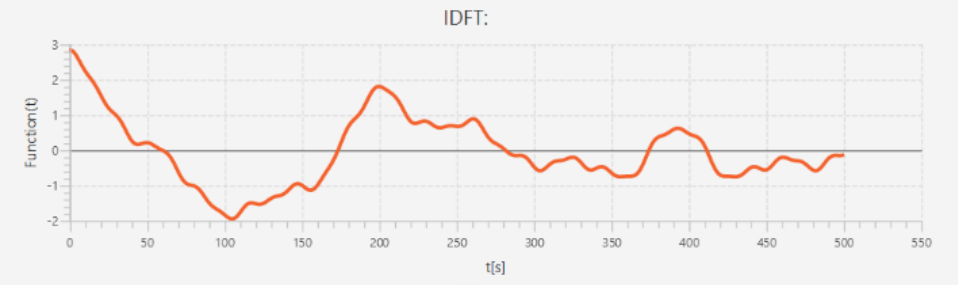
P(t) dla N = 4 – DFT (widmo amplitudowe)



P(t) dla N = 4 – DFT (widmo amplitudowe w skali decybelowej)



P(t) dla N = 4 – IDFT

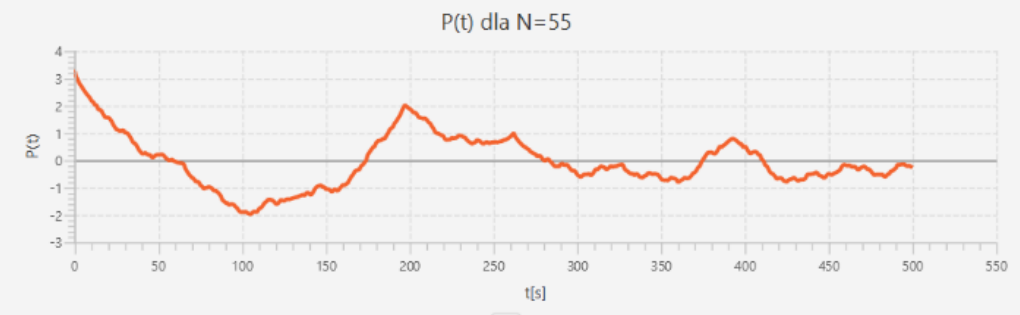


Kod realizujący tworzenie funkcji P(t) dla N = 4:

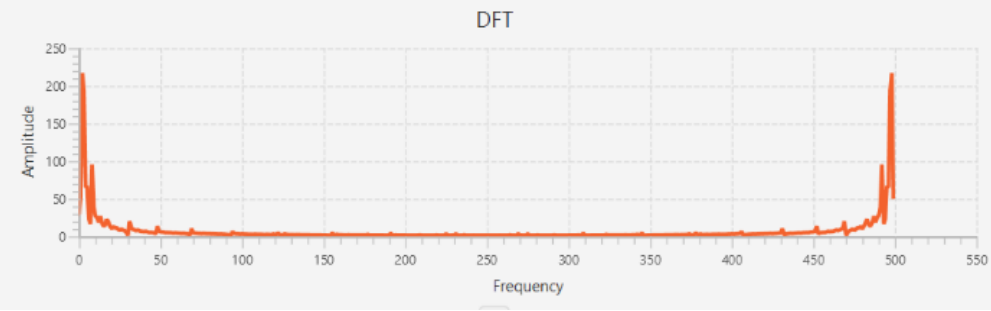
```
public static ChartDetails makeP4(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    for (double t = start; t <= stop; t += (1 / step)) {
        double tmp = 0;
        for (int n = 1; n <= 4; n++) {
            tmp += (cos(12 * t * n * n) + cos(16 * t * n)) / (n * n);
        }

        scores.add(tmp);
    }
    return new ChartDetails( title: "P(t) dla N=4", scores,  xAxisTitle: "t[s]",  yAxisTitle: "P(t)");
}
```

FUNKCJA P(t) dla N = 55

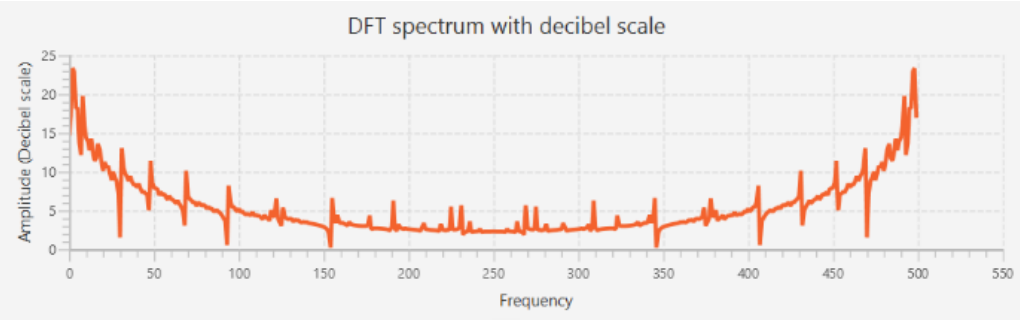
P(t) dla N = 55 (na wejściu)



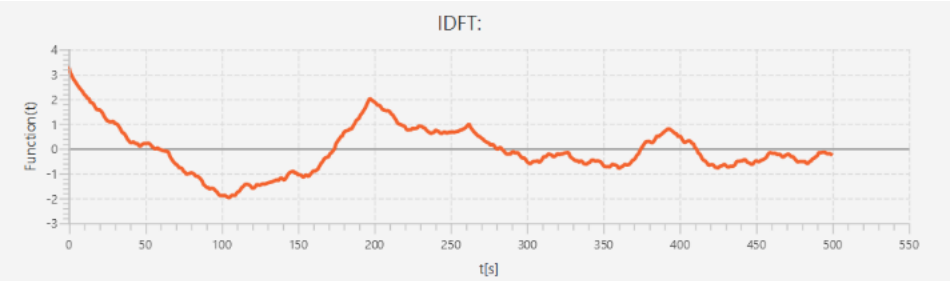
P(t) dla N = 55 – DFT (widmo amplitudowe)



P(t) dla N = 55 – DFT (widmo amplitudowe w skali decybelowej)



P(t) dla N = 55 – IDFT



Kod realizujący tworzenie funkcji P(t) dla N = 55:

```
public static ChartDetails makeP55(double start, double stop, double step) {

    List<Double> scores = new ArrayList<>();
    for (double t = start; t <= stop; t += (1 / step)) {
        double tmp = 0;
        for (int n = 1; n <= 55; n++) {
            tmp += (cos(12 * t * n * n) + cos(16 * t * n)) / (n * n);
        }

        scores.add(tmp);
    }
    return new ChartDetails( title: "P(t) dla N=55", scores,  xAxisTitle: "t[s]",  yAxisTitle: "P(t)");
}
```