Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №2.1/2.2

з дисципліни
«Інтелектуальні вбудовані системи»

на тему
«Дослідження алгоритмів перетворення Фур'є»

Виконала:

студентка групи ІП-84
Д'яконенко Дар'я Костянтинівна
Номер заліковки: 8406

Перевірив:

Регіда П. Г.

Київ 2020

Основні теоретичні відомості:

В основі спектрального аналізу використовується реалізація так званого дискретного перетворювача Фур'є (ДПФ) з неформальним (не формульним) поданням сигналів, тобто досліджувані сигнали представляються послідовністю відліків x(k)

$$F_x(p) = \sum_{k=0}^{N-1} x(k) \cdot e^{-jk\Delta tp\Delta\omega}$$

$$\omega \to \omega_p \to p\Delta\omega \to p \qquad \Delta\omega = \frac{2\pi}{T}$$

ДПФ - проста обчислювальна процедура типу звірки (тобто □-е парних множень), яка за складністю також має оцінку N2 + N.

Лістинг програми:

```
using System;
using System.Linq;
using System.Threading;
using System.Windows.Forms;
using System.Diagnostics;
using System.Collections.Generic;
using System.Numerics;

namespace lab1._1
{
    public class Program
    {
```

```
public const int HARMONICS_NUMBER  = 12;
public const int BORDER_FREQUENCY  = 1800;
public const int CALLS_NUMBER  = 64;
static void Main()
{
   Application.EnableVisualStyles();
   Application.SetCompatibleTextRenderingDefault(false);
   Application.Run(new Form1());
}

public static double calculateM(double[] arr)
{
   double sum = 0;

   for (int i = 0; i < arr.Length; i++)
   {
      sum += arr[i];
   }

   return sum / arr.Length;
}

public static double calculateD(double M, double[] arr)
{
   double sum = 0;

   for (int i = 0; i < arr.Length; i++)
   {
      sum += ((arr[i] - M) * (arr[i] - M));
```

```
        }

        return sum / (arr.Length - 1);
    }

    public static double[] generateSignal(int CALLS_NUMBER, int
HARMONICS_NUMBER, int BORDER_FREQUENCY)
    {
        double[] signals = new double[CALLS_NUMBER];
        var rand = new Random();

        for (int i = 1; i <= HARMONICS_NUMBER; i++) {
            double frequency = (i*BORDER_FREQUENCY) /
HARMONICS_NUMBER;
            double amplitude = rand.NextDouble();
            Thread.Sleep(25); // to refresh random seed
            double phase = rand.NextDouble();
            for(int time = 0; time < CALLS_NUMBER; time++)
            {
                double signal = amplitude * Math.Sin((frequency * time) + phase);
                signals[time] += signal;
            }
        }

        return signals;
    }
```

```
public static double[] corFunc(double[] x, double[] y)
{
   double[] corValues = new double[CALLS_NUMBER/2];
   double avgX = x.Average();
   double avgY = y.Average();
   double stdevX = Math.Sqrt(calculateD(avgX,  x));

   int half = CALLS_NUMBER  / 2;
   double v, m, std;
   for(int i = 0; i<half; i++)
   {
      v = 0;
      for(int j = 0;j< x.Length-i; j++)
      {
         v += x[j] * y[j + i];
         m = y.Average();
         std = Math.Sqrt(calculateD(avgY,y));
         corValues[i] = (v / (x.Length  - i) - avgX * m) / (stdevX * std);
      }
   }
   return corValues;
}

public static double[] autoCorFunc(double [] signals)
{
   return corFunc(signals, signals);
}

public static double[] complexity()
```

```
    {
        double[] time = new double[CALLS_NUMBER];
        Stopwatch sw = new Stopwatch();

        for (int i = 0; i < CALLS_NUMBER; i++)
        {
            sw.Start();
            generateSignal(i, HARMONICS_NUMBER,
BORDER_FREQUENCY);
            sw.Stop();
            time[i] = sw.ElapsedMilliseconds/100;
        }
        return time;
    }


    // LAB 2

    public static double[] DFT(double[] data)
    {
        int n = data.Length;
        int m = n;
        double[] real = new double[n];
        double[] imag = new double[n];
        double[] result = new double[m];
        double pi_div = 2.0 * Math.PI / n;
        for (int w = 0; w < m; w++)
        {
            double a = w * pi_div;
```

```
        for (int t = 0; t < n; t++)
        {
            real[w] += data[t] * Math.Cos(a * t);
            imag[w] += data[t] * Math.Sin(a * t);
        }
        result[w] = Math.Sqrt(real[w] * real[w] + imag[w] * imag[w]);
    }
    return result;
}



public static double[] cTor(Complex[] d)
{
    double[] res = new double[d.Length];

    for (int i = 0; i < d.Length; i++)
    {
        double real = d[i].Real;
        double imaginary = d[i].Imaginary;
        res[i] = Math.Sqrt(Math.Pow(real, 2)+Math.Pow(imaginary, 2));
    }
    return res;
}

const double PI = 3.1415926536;

private static int bitReverse(int x, int log2n)
{
```

```
    int n = 0;
    for (int i = 0; i < log2n; i++)
    {
        n <<= 1;
        n |= (x & 1);
        x >>= 1;
    }
    return n;
}

public static Complex[] fft(double[] a, int log2n)
{
    Complex[] A = new Complex[64];

    int n = 64;

    for (int i = 0; i < n; ++i)
    {
        int rev = bitReverse(i, log2n);
        A[i] = a[rev];
    }


    Complex J = new Complex(0, 1);
    for (int s = 1; s <= log2n; ++s)
    {
        int m = 1 << s;
        int m2 = m >> 1;
        Complex w =  new Complex(1, 0);
```

```
            Complex wm = Complex.Exp(J * (PI / m2));
            for (int j = 0; j < m2; ++j)
            {
               for (int k = j; k < n; k += m)
               {

                  Complex t = w * A[k + m2];
                  Complex u = A[k];

                  A[k] = u + t;

                  A[k + m2] = u - t;
               }
               w *= wm;
            }
         }

            return A;
      }



}
}
```

Результати виконання програми:

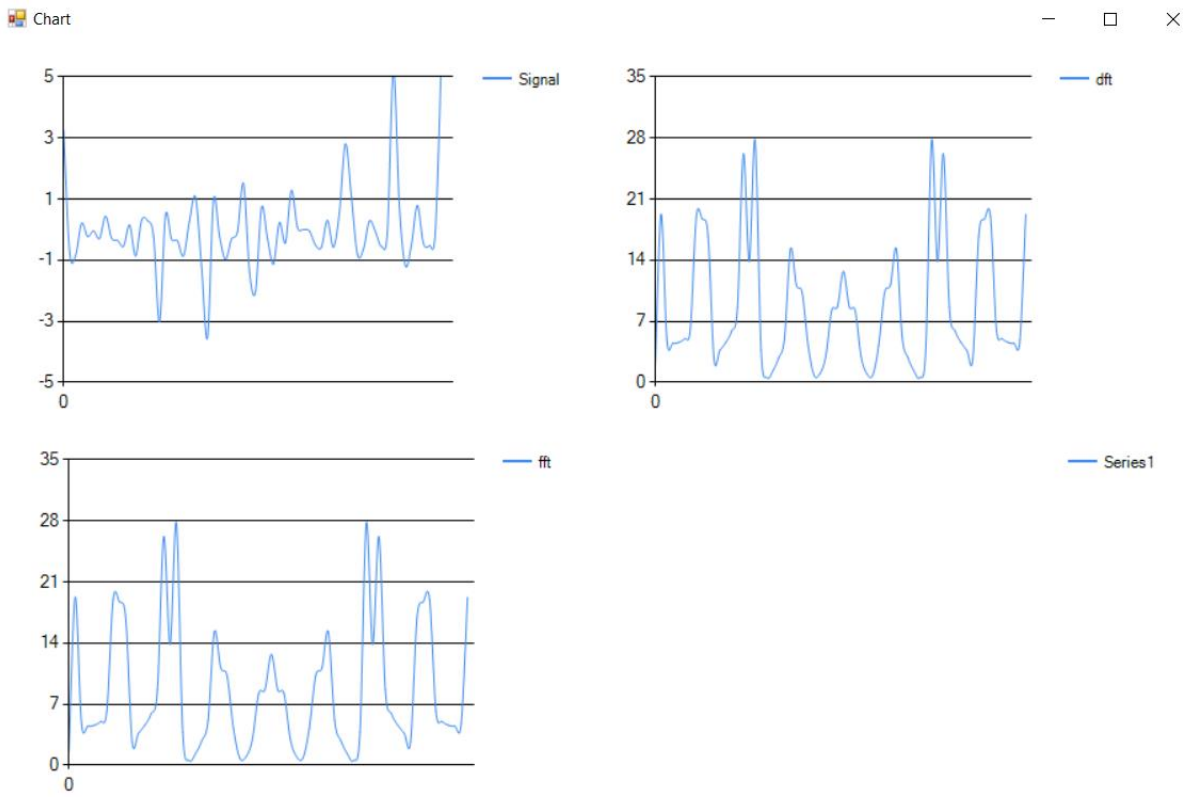
Chart
Signal
5
3
1
-1
-3
-5
0
dft
35
28
21
14
7
0
0
fft
35
28
21
14
7
0
0
Series1