

Contents

1	Stage 1	3
1.1	Source code of the CW1Stage1 class	3
1.2	Usage	5
2	Stage 2	6
2.1	Source code of the CW1Stage2 class	6
2.2	Usage	8
2.3	Answer to the question	8
3	Stage 3	9
3.1	Source code of the CW1Stage3 class	9
3.2	Comments on the spectrums	12
3.2.1	Sin(t)	12
3.2.2	Cos(3t)	13
3.2.3	Sin(t)+Cos(2t)	13
3.2.4	Sin(t)*Cos(2t)	13
3.2.5	Square Wave	15
3.2.6	Square Wave + FFT	15

4	Stage 4	17
4.1	Source code of the CW1Stage3 class	17
4.2	Comments	19
5	Stage 5 and 6	20
5.1	Notes on stages 5 and 6	20

Chapter 1

Stage 1

1.1 Source code of the CW1Stage1 class

Listing 1.1: CW1Stage1.java

```
class CW1Stage1{
    public static void main(String [] args){

        FastFourierTransform fft = new FastFourierTransform ();

        /*
         * For each of the three signals we need
         * - an array of float for the signal itself
         * - an array of float for its spectrum
         */
        float [] signal1 = new float [4];
        float [] spectrum1 = new float [2];

        float [] signal2 = new float [4];
        float [] spectrum2 = new float [2];

        float [] signal3 = new float [8];
        float [] spectrum3 = new float [4];

        /*
         * We now feed the signal's arrays
         */
        signal1 [0] = 0;
        signal1 [1] = 0;
        signal1 [2] = 0;
        signal1 [3] = 0;
```

```

    signal2 [0] = 1;
    signal2 [1] = 1;
    signal2 [2] = 1;
    signal2 [3] = 1;

    signal3 [0] = 1;
    signal3 [1] = 1;
    signal3 [2] = 0;
    signal3 [3] = 0;
    signal3 [4] = 1;
    signal3 [5] = 1;
    signal3 [6] = 0;
    signal3 [7] = 0;

    /*
     * For each signal array we feed the relevant
     * spectrum array with its spectrum
     */

    spectrum1 = fft.fftMag(signal1);
    spectrum2 = fft.fftMag(signal2);
    spectrum3 = fft.fftMag(signal3);

    /*
     * Finally, we display the values
     */
    displaySignalAndSpectrum(signal1, spectrum1);
    displaySignalAndSpectrum(signal2, spectrum2);
    displaySignalAndSpectrum(signal3, spectrum3);
}

/**
 * displays the signal and spectrum arrays
 */
public static void displaySignalAndSpectrum(float [] signal,
                                           float [] spectrum){
    System.out.println("\nSignal");
    for(int i=0;i<signal.length;i++){
        System.out.print(signal[i]+" ");
    }
    System.out.println("\nTransform");
    for(int i=0;i<spectrum.length;i++){
        System.out.print(spectrum[i]+" ");
    }
    System.out.println("\n");
}
}

```

1.2 Usage

Listing 1.2: Output of the execution

```
Signal
0.0  0.0  0.0  0.0
Transform
0.0  0.0

Signal
1.0  1.0  1.0  1.0
Transform
1.0  0.0

Signal
1.0  1.0  0.0  0.0  1.0  1.0  0.0  0.0
Transform
0.5  0.0  0.70710677  0.0
```

Chapter 2

Stage 2

2.1 Source code of the CW1Stage2 class

Listing 2.1: CW1Stage2.java

```
class CW1Stage2{
    public static void main(String [] args){
        FastFourierTransform fft = new FastFourierTransform ();

        /*
         * The two arrays for the signal and its spectrum
         */

        float [] signal = new float [32];
        float [] spectrum = new float [16];

        /*
         * Feeding the array with the signal
         */

        signal[0] = -0.0371f;
        signal[1] = 1.17151f;
        signal[2] = -0.0527f;
        signal[3] = 1.59923f;
        signal[4] = -0.0027f;
        signal[5] = -1.3691f;
        signal[6] = 0.06995f;
        signal[7] = -1.5467f;
        signal[8] = 0.05219f;
        signal[9] = 1.46775f;
        signal[10] = -0.1412f;
```

```

signal[11] = 1.33412f;
signal[12] = 0.21519f;
signal[13] = -1.2873f;
signal[14] = 0.07959f;
signal[15] = -1.3411f;
signal[16] = -0.0570f;
signal[17] = 1.51862f;
signal[18] = -0.0079f;
signal[19] = 1.50113f;
signal[20] = -0.0821f;
signal[21] = -1.3454f;
signal[22] = -0.2069f;
signal[23] = -1.4726f;
signal[24] = -0.2153f;
signal[25] = 1.51804f;
signal[26] = -0.0475f;
signal[27] = 1.41315f;
signal[28] = -0.0336f;
signal[29] = -1.3825f;
signal[30] = 0.02953f;
signal[31] = -1.5112f;

```

```

/*
 * Get the spectrum of the signal and display
 */
spectrum = fft.fftMag(signal);
displaySignalAndSpectrum(signal , spectrum);

```

```

}
public static void displaySignalAndSpectrum(float [] signal ,
                                             float [] spectrum){
    System.out.println("\nSignal");
    for(int i=0;i<signal.length;i++){
        System.out.print(signal[i]+"  ");
    }
    System.out.println("\n\nTransform");
    for(int i=0;i<spectrum.length;i++){
        System.out.print(spectrum[i]+"  ");
    }
    System.out.println("\n");
}
}

```

2.2 Usage

Listing 2.2: Output of the execution

```
Signal
-0.0371 1.17151 -0.0527 1.59923 -0.0027 -1.3691 0.06995 -1.5467
0.05219 1.46775 -0.1412 1.33412 0.21519 -1.2873 0.07959 -1.3411
-0.057 1.51862 -0.0079 1.50113 -0.0821 -1.3454 -0.2069 -1.4726
-0.2153 1.51804 -0.0475 1.41315 -0.0336 -1.3825 0.02953 -1.5112

Transform
0.005309401 0.046879534 0.027470805 0.04465356
0.9941988 0.015258165 0.044472765 0.045014083
0.021029918 0.007935021 0.047579303 0.051182766
1.0205848 0.022527842 0.03420972 0.05714904
```

2.3 Answer to the question

The output from the FFT shows that two frequencies are obviously present.

0.005309401 0.046879534 0.027470805 0.04465356

0.9941988 0.015258165 0.044472765 0.045014083

0.021029918 0.007935021 0.047579303 0.051182766

1.0205848 0.022527842 0.03420972 0.05714904

The frequencies concerned are:

$$f_4 = \frac{n}{N\Delta} = \frac{4}{32 \times \frac{1}{8000}} = 1000Hz$$
$$f_{12} = \frac{n}{N\Delta} = \frac{12}{32 \times \frac{1}{8000}} = 3000Hz$$

And so we have a component of 1000Hz and a higher frequency of component of 3000Hz.

Chapter 3

Stage 3

3.1 Source code of the CW1Stage3 class

Listing 3.1: CW1Stage3.java

```
class CW1Stage3{
    public static void main(String [] args){

        FastFourierTransform fft = new FastFourierTransform ();

        // Number of samples (2^5 = 32)
        int nbSamples = 32;

        // Two Pi
        double twoPi = 2.0; //2*Math.PI;

        /*
         * The arrays for signal and spectrum
         */

        float [] signal1 = new float [nbSamples];
        float [] spectrum1 = new float [nbSamples/2];

        float [] signal2 = new float [nbSamples];
        float [] spectrum2 = new float [nbSamples/2];

        float [] signal3 = new float [nbSamples];
        float [] spectrum3 = new float [nbSamples/2];

        float [] signal4 = new float [nbSamples];
        float [] spectrum4 = new float [nbSamples/2];
    }
}
```

```

float [] signal5 = new float [nbSamples];
float [] spectrum5 = new float [nbSamples/2];

if (args.length < 1) {
    // if no args, print usage and non-zero quit
    printUsage();
    System.exit(1);
}
char choice = args[0].toCharArray()[0];

/*
 * For each sample we feed the signal arrays
 */

for (int i=0; i < nbSamples; i++){

    // 'time' at which the sample is being taken
    double sTime = twoPi*i/nbSamples;

    signal1[i] = (float) Math.sin(sTime);
    signal2[i] = (float) Math.cos(3*sTime);
    signal3[i] = (float) (Math.sin(sTime)
                          + Math.cos(sTime));
    signal4[i] = (float) (Math.sin(sTime)
                          + Math.cos(2*sTime));
    signal5[i] = (float) square(sTime);
}

/*
 * We then feed the the spectrum array with the transform
 * of its signal and display them
 */

/*
 * Selector of signal/spectrum to printed out
 */

switch (choice) {
case 'a':
    System.out.println(" Chosen signal: sin(t)");
    spectrum1 = fft.fftMag(signal1);
    displaySignalAndSpectrum(signal1, spectrum1);
    break;
case 'b':
    System.out.println(" Chosen signal: cos(3t)");

```

```

        spectrum2 = fft.fftMag(signal2);
        displaySignalAndSpectrum(signal2,spectrum2);
        break;
    case 'c':
        System.out.println(" Chosen_signal:_ sin(t)+cos(2t)");
        spectrum3 = fft.fftMag(signal3);
        displaySignalAndSpectrum(signal3,spectrum3);
        break;
    case 'd':
        System.out.println(" Chosen_signal:_ sin(t)*cos(2t)");
        spectrum4 = fft.fftMag(signal4);
        displaySignalAndSpectrum(signal4,spectrum4);
        break;
    case 'e':
        System.out.println(" Chosen_signal:_ Square_Wave");
        spectrum5 = fft.fftMag(signal5);
        displaySignalAndSpectrum(signal5,spectrum5);
        break;
    default:
        System.out.println("Unknown_code");
        printUsage();
    }
}

public static void displaySignalAndSpectrum(float [] signal,
                                           float [] spectrum){

    System.out.println("\n\nSignal\n");
    for(int i=0;i<signal.length;i++){
        System.out.print(signal[i]+"_");
    }
    System.out.println("\n\nTransform");
    for(int i=0;i<spectrum.length;i++){
        System.out.print(spectrum[i]+"_");
    }
    System.out.println("\n");
}

private static void printUsage(){
    System.out.println("Usage:");
    System.out.println("\tjava_CW1Stage3_<signal_code>");
    System.out.println("\tCodes:");
    System.out.println("\t\ta:_ sin(t)");
    System.out.println("\t\tb:_ cos(3t)");
    System.out.println("\t\tc:_ sin(t)+cos(2t)");
    System.out.println("\t\td:_ sin(t)*cos(2t)");
    System.out.println("\t\te:_ square_wave\n");
    System.out.println("\tExample_(for_cos(3t)):\n");
}

```

```

.....\n\t\t\t>_java_CW1Stage3_b_");
}
/**
 *
 * @return the value of the square wave for the given value
 */
public static int square(double f){
    int ii = (int) f;
    if((ii % 2) == 0){
        return 1;
    }else{
        return 0;
    }
}
}

```

3.2 Comments on the spectrums

3.2.1 Sin(t)

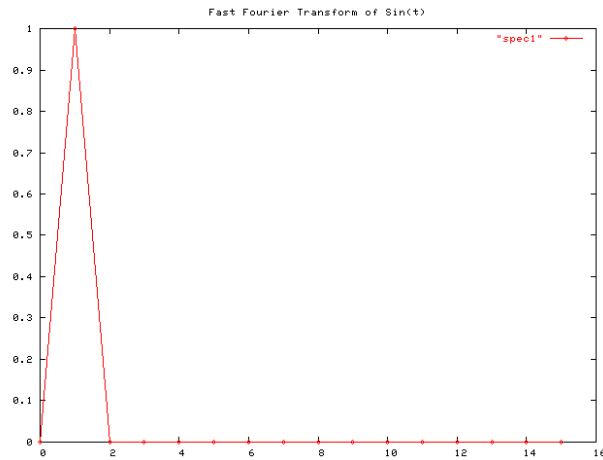


Figure 3.1: Spectrum of $\text{Sin}(t)$

It was obvious to obtain this spectrum with because of the following:

$$\sin(t) = \sin(\mathbf{1} \times t)$$

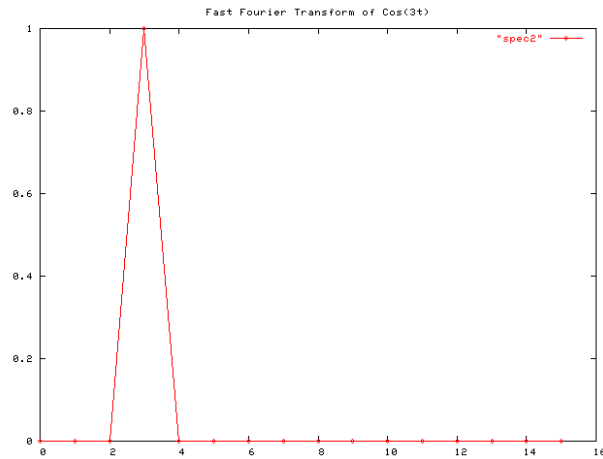


Figure 3.2: Spectrum of $\text{Cos}(3t)$

3.2.2 $\text{Cos}(3t)$

In trigonometry, we know the following formulae:

$$\cos(x) = \sin\left(x + \frac{\pi}{2}\right)$$

So, if $x = 3 \times t$, then we obtain:

$$\cos(3 \times t) = \sin\left(3 \times t + \frac{\pi}{2}\right)$$

$\sin\left(3 \times t + \frac{\pi}{2}\right)$ and $\sin(3 \times t)$ have the same period.

Therefore $\sin(3t)$ and $\cos(3t)$ also have the same period and frequency.

3.2.3 $\text{Sint}(t) + \text{Cos}(2t)$

However $\cos(2 \times t)$ has the same frequency than $\sin(2 \times t)$, we can not here say that $\cos(2 \times t) + \sin(t)$ has the same frequency than $\sin(2 \times t)$ because the phase is different (switched forward by π).

It is however interesting to see that this sum is represented by only one frequency.

3.2.4 $\text{Sin}(t) * \text{Cos}(2t)$

This product is represented by two frequencies: $\sin(t) + \sin(2 \times t)$

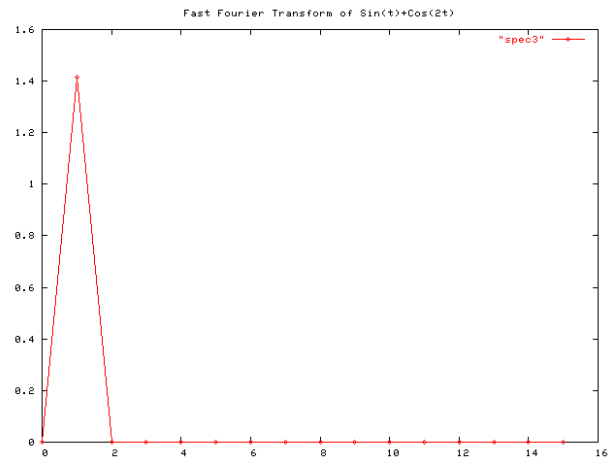


Figure 3.3: Spectrum of $\text{Sin}(t)+\text{Cos}(2t)$

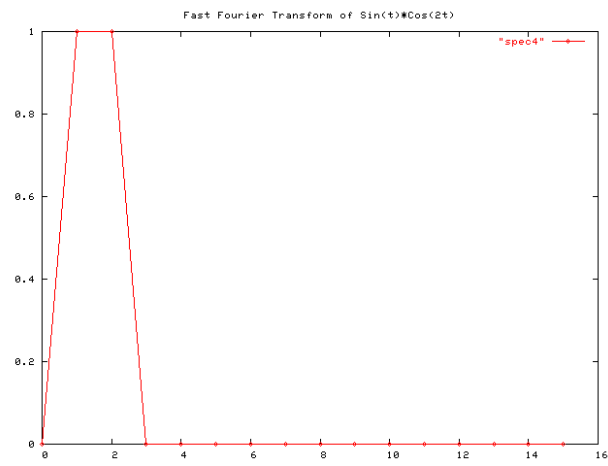


Figure 3.4: Spectrum of $\text{Sin}(t)*\text{Cos}(2t)$

3.2.5 Square Wave

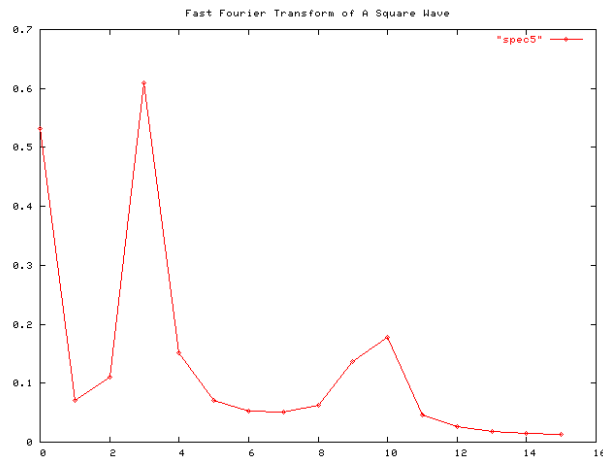


Figure 3.5: Spectrum of a *Square Wave*

The interesting thing is to be able to prove that however a square can not be represented by a *cartesian* function we can still get a close approximation it as show in the next figure.

3.2.6 Square Wave + FFT

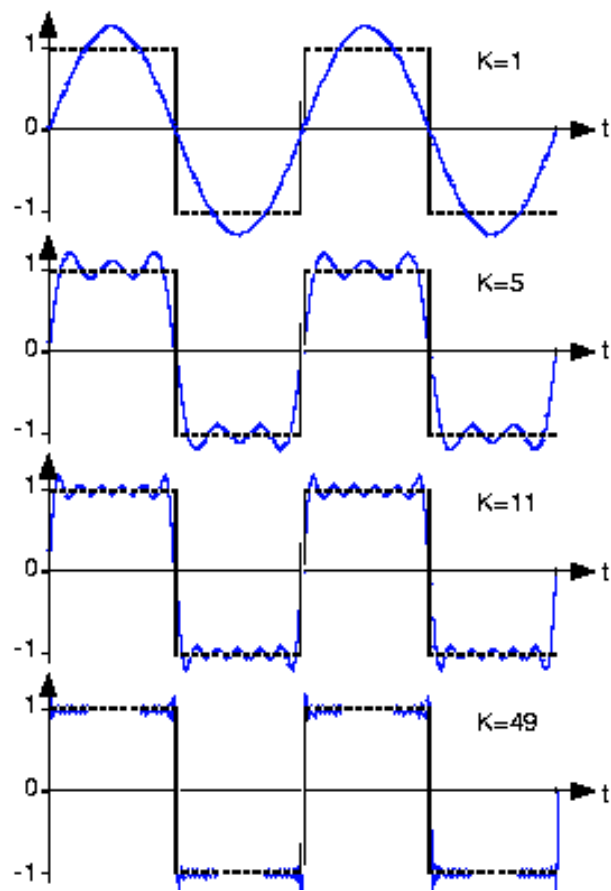


Figure 3.6: Fourier series approximation of a *Square Wave*

Chapter 4

Stage 4

4.1 Source code of the CW1Stage3 class

Listing 4.1: CW1Stage4.java

```
class CW1Stage4{
    public static void main(String [] args){
        FastFourierTransform fft = new FastFourierTransform ();
        int nbSamples = 32;
        float [] signal1 = new float [nbSamples];
        float [] spectrum1 = new float [nbSamples/2];
        float [] signal2 = new float [nbSamples];
        float [] spectrum2 = new float [nbSamples/2];
        float [] signal3 = new float [nbSamples];
        float [] spectrum3 = new float [nbSamples/2];
        float [] signal4 = new float [nbSamples];
        float [] spectrum4 = new float [nbSamples/2];
        float [] signal5 = new float [nbSamples];
        float [] spectrum5 = new float [nbSamples/2];
        double twoPi = 2*Math.PI;

        if(args.length < 2){
            // if no args, print usage and non-zero exit
            printUsage ();
            System.exit (1);
        }
        char choice = args [0].toCharArray () [0];

        for(int i=0; i < nbSamples; i++){
            int noiseFactor = Integer.parseInt (args [1]);
            double sampleTime = Math.random ()*0.1*noiseFactor
                + twoPi*i/nbSamples;
```

```

        signal1[i] = (float) Math.sin(sampleTime);
        signal2[i] = (float) Math.cos(3*sampleTime);
        signal3[i] = (float) (Math.sin(sampleTime)
                               + Math.cos(sampleTime));
        signal4[i] = (float) (Math.sin(sampleTime)
                               + Math.cos(2*sampleTime));
        signal5[i] = (float) square(sampleTime);
    }
    switch(choice){
    case 'a':
        System.out.println(" Chosen signal :
        ----- sin(t)\nNoise_level : "+args[1]);
        spectrum1 = fft.fftMag(signal1);
        displaySignalAndSpectrum(signal1,spectrum1);
        break;
    case 'b':
        System.out.println(" Chosen signal :
        ----- cos(3t)\nNoise_level : "+args[1]);
        spectrum2 = fft.fftMag(signal2);
        displaySignalAndSpectrum(signal2,spectrum2);
        break;
    case 'c':
        System.out.println(" Chosen signal :
        ----- sin(t)+cos(2t)\nNoise_level : "+args[1]);
        spectrum3 = fft.fftMag(signal3);
        displaySignalAndSpectrum(signal3,spectrum3);
        break;
    case 'd':
        System.out.println(" Chosen signal :
        ----- sin(t)*cos(2t)\nNoise_level : "+args[1]);
        spectrum4 = fft.fftMag(signal4);
        displaySignalAndSpectrum(signal4,spectrum4);
        break;
    case 'e':
        System.out.println(" Chosen signal :
        ----- Square_Wave\nNoise_level : "+args[1]);
        spectrum5 = fft.fftMag(signal5);
        displaySignalAndSpectrum(signal5,spectrum5);
        break;
    default:
        System.out.println("Unknown code for signal");
        printUsage();
    }
}

public static void displaySignalAndSpectrum(float[] signal,

```

```

float [] spectrum){
    System.out.println("\nSignal:");
    for(int i=0;i<signal.length;i++){
        System.out.print(signal[i]+" ");
    }
    System.out.println("\n\nTransform:");
    for(int i=0;i<spectrum.length;i++){
        System.out.print(spectrum[i]+" ");
    }
    System.out.println("\n");
}

public static int square(double f){
    int ii = (int) f;
    if((ii % 2) == 0){
        return 1;
    }else{
        return 0;
    }
}

private static void printUsage(){
    System.out.println("Usage:\n\n");
    System.out.println("\tjava CW1Stage4<signal_code><noise_level>\n");
    System.out.println("\tExample_(for_sin(t)_with_noise_level_3):\n\n");
    System.out.println("\t\t\tjava CW1Stage4_a_3\n\n");
}
}
}

```

4.2 Comments

The stage 5 and 6 will show graphically the effect of a noise applied to a wave as well as the result on its spectrum.

Most likely, even if the noise is very high and the wave very distorted, it appears that the spectrum does not change excessively. It is still obvious to find the series for the approximation. This shows that the Fast Fourier Transform is still robust and effective with distorted waves. This tends to prove that, even in a noisy environment, the use of Fourier Transforms for voice recognition should be still quite efficient.

Chapter 5

Stage 5 and 6

5.1 Notes on stages 5 and 6

A class diagram of the most advanced application (stage 6) can be found as an appendice.

For filtering the files types (“.wav” or “.au”) in the file chooser, an external class *Example-Filter* of Sun Microsystems has been added in the current directory because this class does not belong to the standard Java API.

The applications has been designed to be as scalable as possible and allow any lenght, amplitude or frequency of signal as well as any resizing of the frame.

The application can handle all the following audio file format at any frequencies and for *Sun/NeXT audio* and *WAVE audio*:

- little-endian, 8 bit, mono
- little-endian, 16 bit, mono
- little-endian, 8 bit, stereo
- little-endian, 16 bit, stereo

If a stereo file is being opened, then the layout will be updated to allow the display of the two signals (right and left channels) as well as their respective spectrum.

The *frequency*, *samples* and *noise level* can only be applied to signals of the *signal* menu, not to the sound waves.