$Approach\ Algorithm\ of\ Cross\ Correlation$

We agreed to take the different signals used as equal in length.

The covariance of the two vectors in a temporal space are called Cross Correlation and are defined by this formula :

$$(1)\ \Gamma_{AB}(\tau) = A * B^*(-) = \int A(t)B^*(t-\tau)dt$$

And by adding the transform of Fourier, we get the expression :

$$(2)\ \gamma_{AB} = \mathcal{F}[A] \cdot \mathcal{F}[B^*(-)] = a(\nu) \cdot b^*(\nu)$$

For our first native algorithm use the Cross Correlation formula (1) :

Since our program use two tabs of double named sig_1 and sig_2 , the expression is changed as to include the function name, and the name of the length methods :

$$CrossCorrelation_{sig_1,sig_2}\left[Offset\right] = \sum_{i=0}^{length(sig_1)-1} sig_1[i] \cdot sig_2\left[i + Offset\right]$$

The function take as argument sig_1 and sig_2 and return a tabs named $tabFinal$.

Where $Offset$ is defined as the cross-correlation lag.

The Offset play the role of τ and is purpose is to calcul an area of value in order to create a sum that will be put into our tab $tabFinal$ that we will return at the end with give us this final formula where $Offset$ range from $-length(sig_1) + 1$ to $length(sig_1) - 1$, k range from 0 to $length(sig_1) - 1$ and $k + Offset$ must be within the bounds of sig_1 :

$$tabFinal\left[Offset + length(sig_1) - 1\right] = \sum_{k=0}^{length(sig_1)-1} sig_1[k] \cdot sig_2[k + Offset]$$

In sum, we have our first algorithm that is presented as follow with a operation cost of O(n) and with a middle complexity :

```
public static double[] crossCorrelation1(double[] sig1, double[] sig2) {
    int sigLength1 = sig1.length;
    int sigLength2 = sig2.length;
    double[] tabFinal = new double[sigLength1 + sigLength2 - 1];

    for(int i = -sigLength2 + 1; i < sigLength1; ++i) {
        double j = 0.0;

        for(int k = 0; k < sigLength1; ++k) {
            int l = k + i;
            if (l >= 0 && l < sigLength2) {
                j += sig1[k] * sig2[l];
            }
        }

        tabFinal[i + sigLength2 - 1] = j;
    }

    return tabFinal;
}
```

For our second algorithm we will used the expression (2) :

First we need to define, the transform of Fourier in order to understand our expression. The transform of Fourier $\mathcal{F}$ is a operator that transform a integrable function on $\mathbb{R}$ into another function, describing his frequencial spectrum.

$$\mathcal{F}(f) : \xi \mapsto \hat{f} = \int_{-\infty}^{+\infty} f(x) e^{-i\xi x} dx$$

And in order to recollect a usable frequency spectrum from our input sig_1 and sig_2, we need to use the Fourier inverse transformation, describe as this :

$$\mathcal{F}^{-1}(\hat{f})(x) = \int_{\mathbb{R}} \hat{f}(\xi) e^{2i\pi x\xi} d\xi$$

This operator used in our function will make us use complex numbers. Therefore, we will be using complex conjugation in our operations, and further more we will be using the FFT (Fast Fourier Transform) and the $IFFT$ (Inverse Fast Fourier Transform) in order for us to have a

operation cost more close to $O(\mathbb{N}\ log\mathbb{N})$.
So, our function by adding this operator should look like that :

$$FrTrCrossCorrelation(sig_1, sig_2) = \int FFT[A(t)] \cdot FFT[B^*(t-\tau)]dt$$

$$\Leftrightarrow\ FrTrCrossCorrelation = \overline{A(\omega)} \cdot B^*(\omega)$$

$$\Leftrightarrow\ FrTrCrossCorrelation = IFFT(\overline{A(\omega)} \cdot B^*(\omega))$$

By development we have :

$$FrTrCrossCorrelation(sig_1, sig_2) = IFFT\Big(\sum_{k=0}^{length(sig_1)-1} FFT\big(sig_1\big)[k] \cdot \overline{FFT\big(sig_2\big)[k]}\Big)$$

And so the program should look like this :

```
public class SumCrossSpectrum {

    public static Complex computeSumOfCrossSpectrum(double[] sig1, double[]
sig2) {
        // Ensure both signals are the same length
        if (sig1.length != sig2.length) {
            throw new IllegalArgumentException("Input signals must have the
same length.");
        }

        int lengthSig = sig1.length;

        // Fourier Transform
        FastFourierTransformer fft = new
FastFourierTransformer(DftNormalization.STANDARD);
        Complex[] F1 = fft.transform(sig1, TransformType.FORWARD);
        Complex[] F2 = fft.transform(sig2, TransformType.FORWARD);

        // The sum of the cross spectrum
        Complex sum = new Complex(0, 0);
        for (int i = 0; i < lengthSig; i++) {
            sum = sum.add(F1[i].multiply(F2[i].conjugate()));
        }

        return sum;
        }
```

```
}
```