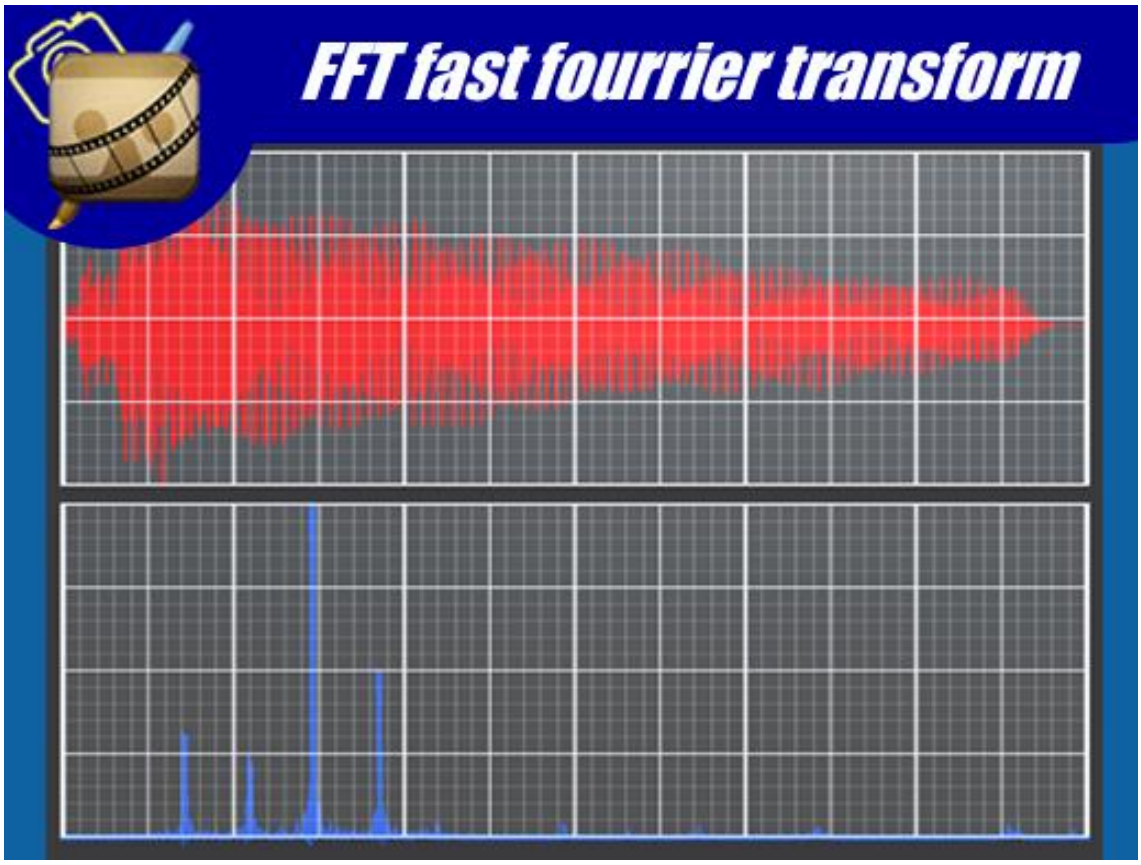


| AUTHOR | Michael Soler |
|---|---|
| CONTACT | michael.soler.beatty@gmail.com |
| Unity Ver. | 2019.1 |

## Index

## 1.Description of the package.

With this package developers will be able to compute the FFT (Fast Fourrier Transform) of a signal/data stream or array of information. The FFT transforms a signal from the time-domain to the frequency domain.

The package contains all the textures, models and scripts shown in the video. For further questions, please contact michael.soler.beatty@gmail.com

## 2.About the FFT (Fast Fourrier Transform)

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence. It converts a signal from its original domain (often time or space) to a representation in the frequency domain. The algorithm that we use in our scripts is the Cooley-Tukey method:

$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{\frac{-2\pi i(2n)k}{\frac{N}{2}}} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{\frac{-2\pi i(2n+1)k}{\frac{N}{2}}} = E_k + e^{\frac{-2\pi ik}{N}} O_k$$

## 3.Scripting

The main script used in this asset is: “FastFourierTransform.cs”.

```
using System;
using UnityEngine;
using System.Numerics;

public class FastFourierTransform : MonoBehaviour
{
    /// <summary>
    /// A fast Fourier transform (FFT) is an algorithm that computes the
discrete Fourier transform (DFT) of a sequence.
    /// It converts a signal from its original domain (often time or space) to
a representation in the frequency domain
    /// </summary>
    ///

    //the FFT returns a complex array of numbers given a input array of complex
numbers.
    public static Complex[] FFT(Complex[] input, bool invert)
    {
        //in case there is only one element
        if (input.Length == 1)
        {
            return new Complex[] { input[0] };
```

```
        }

        //for more elements we need to otain the lenght of the input data
stream
        int length = input.Length;

        //half will be the half of the lenght
        int half = length / 2;

        //this is the result of the FFT
        Complex[] result = new Complex[length];

        // factor that goes in the
        double factorEXP = -2.0 * Math.PI / length;

        //in case we want to invert the factor
        if (invert)
        {
            factorEXP = -factorEXP;
        }


        //
        // Cooley-Tukey algorithm. This is a divide and conquer algorithm that
recursively breaks down a DFT of any composite size N = N1N2 into many smaller
DFTs of sizes N1 and N2,
        // it is divided into even and odd components
        //

        //even
        Complex[] evens = new Complex[half];
        for (int i = 0; i < half; i++)
        {
            evens[i] = input[2 * i];
        }
        //FFT recursive call
        Complex[] evenResult = FFT(evens, invert);

        //odd
        Complex[] odds = evens;
        for (int i = 0; i < half; i++)
        {
            odds[i] = input[2 * i + 1];
        }
        // FFT recursive call
        Complex[] oddResult = FFT(odds, invert);


        // final algorithm
        //          N/2-1                                   N/2-1
        //   FFT_k= SUM  X_2n ·e^(-2*pi*(2n)*k)/(N/2)  +  SUM  X_2n+1 ·e^(-
2*pi*(2n+1)*k)/(N/2)
        //           0                                       0
        //
        // = Even_k + O_k·e^(-2*pi**k)/(N)

        for (int k = 0; k < half; k++)
        {
            double factor_K = factorEXP * k;

            //                      odd part   &  this is the second part that
is added module 1 argument factor_k
```

```
            Complex oddComponent = oddResult[k] * new
Complex(1*Math.Cos(factor_K), 1*Math.Sin(factor_K));

            //first part of the chart
            result[k] = evenResult[k] + oddComponent;
            //second part of the chart
            result[k + half] = evenResult[k] - oddComponent;
        }

        //reutrn the values (complex). To show FFT we need to display module or
"abs" of the complex number
        return result;
    }

    public static Complex[] doubleToComplex(double[] inp)
    {
        Complex[] outp = new Complex[inp.Length];

        //convert to complex number
        for (int ii = 0; ii < inp.Length; ii++)
        {

            outp[ii] = new Complex(inp[ii], 0);
        }

        return outp;
    }


    /// <summary>
    /// maximum and minimum funtions for double arrays
    /// </summary>

    public static double MaxD(double[] inp)
    {
        double outp = -1e10;

        for (int ii = 0; ii < inp.Length; ii++)
        {
            if (inp[ii] > outp)
            {
                outp = inp[ii];
            }
        }

        return outp;
    }

    public static double MinD(double[] inp)
    {
        double outp = 1e10;

        for (int ii = 0; ii < inp.Length; ii++)
        {
            if (inp[ii] < outp)
            {
                outp = inp[ii];
            }
        }

        return outp;
    }
```

```
}
```

To use our scripts and obtain the spectrum of a set of data you must follow these steps.

1. Create the inout and output complex arrays

```
Complex[] inputSignal_Time = new Complex[windowSize];
Complex[] outputSignal_Freq = new Complex[windowSize];
```

2. Transform from double to complex

```
inputSignal_Time =
FastFourierTransform.doubleToComplex(Y_inputValues);
```

3. Obtain the FFT

```
//result is the iutput values once DFT has been applied
outputSignal_Freq =
FastFourierTransform.FFT(inputSignal_Time,false);
```

4. Obtain the module of the complex number

```
Y_output = new double[windowSize];
//get module of complex number
for (int ii = 0; ii < windowSize; ii++)
{
    Y_output[ii] = (double)Complex.Abs(outputSignal_Freq[ii]);
 }
```

## 4. Video tutorial

We have a video tutorial explaining how the scripts and game mechanics works.

https://youtu.be/Cx6V23-IZK4