

Proyecto

Mariela López Jarquín

20 de enero de 2021

Resumen

A continuación se muestra la programación del algoritmo para calcular la transformada de Fourier, la DFT (Transformada de Fourier Discreta) y FFT (Transformada rápida de Fourier).

1. Transformada de Fourier Discreta

La transformada discreta de Fourier o DFT es un tipo de transformada discreta utilizada en el análisis de Fourier. Transforma una función matemática en otra, obteniendo una representación en el dominio de la frecuencia, siendo la función original una función en el dominio del tiempo.

La secuencia de N números complejos x_0, \dots, x_{N-1} se transforma en la secuencia de N números complejos X_0, \dots, X_{N-1} mediante la DFT con la fórmula

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-\frac{i2\pi}{N}kn} \quad k = 0, \dots, N-1$$

1.1. Problema computacional.

Objetivo: Calcular la transformada de Fourier de una función.

Entrada: El número n representa la cantidad de muestras.

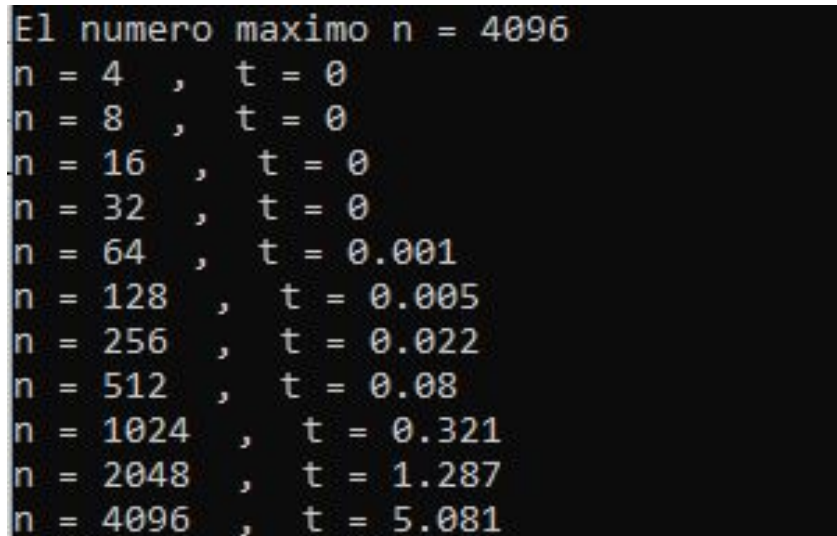
Salida: El tiempo de ejecución de cada n muestra.

1.2. Algoritmo.

Para el algoritmo se utilizó $f(x) = 2\sin(2\pi x) + 5\cos(2\pi x)$ y también se usó la librería `complex` para manipular números complejo. El C++ tiene una potente librería standard para numeros complejos. Se puede utilizar incluyendo en la librería del programa `include <complex>`

1.3. Instancia del problema.

Como prueba de escritorio, se seleccionó la siguiente instancia del problema. Entrada: 4096. La salida del programa se observa en la Figura 1.



```
El numero maximo n = 4096
n = 4 , t = 0
n = 8 , t = 0
n = 16 , t = 0
n = 32 , t = 0
n = 64 , t = 0.001
n = 128 , t = 0.005
n = 256 , t = 0.022
n = 512 , t = 0.08
n = 1024 , t = 0.321
n = 2048 , t = 1.287
n = 4096 , t = 5.081
```

Figura 1: Ejecución del programa.

2. Transformada Rápida de Fourier

Conocida por la abreviatura FFT es un algoritmo eficiente que permite calcular la transformada de Fourier discreta y su inversa. Su importancia radica en el hecho que elimina una gran parte de los cálculos repetitivos a que está sometida la DFT, por lo tanto se logra un cálculo más rápido.

2.1. Problema computacional.

Objetivo: Calcular la transformada de fourier mediante Transformada Rápida de Fourier.

Entrada: El valor de n para la cantidad de muestra.

Salida: El tiempo de ejecución de cada n muestra.

2.2. Algoritmo.

Al igual que el DFT se utilizó la misma función de prueba y mismas librerías junto con los metodos de la clase Complex para la programación del algoritmo FFT.

2.3. Instancia del problema.

Como prueba de escritorio, se seleccionó la siguiente instancia del problema. Entrada: 8192, el cual es significativamente mayor que la instancia en el DFT ya que el algoritmo FFT es mucho más eficiente. La salida del programa se observa en la Figura 2.

```
num. maximo n = 8192
n = 4 , t = 0
n = 8 , t = 0
n = 16 , t = 0.001
n = 32 , t = 0
n = 64 , t = 0
n = 128 , t = 0.001
n = 256 , t = 0.001
n = 512 , t = 0.002
n = 1024 , t = 0.004
n = 2048 , t = 0.008
n = 4096 , t = 0.017
n = 8192 , t = 0.036
```

Figura 2: Ejecución del programa.

3. Comparación entre DFT y FFT

Comparando los datos de tiempo de ejecución por cada n muestra entre el DFT y FFT notamos una gran diferencia entre tiempos de ejecución, lo cual tiene sentido ya que el algoritmo de DFT se utilizan dos ciclos for uno dentro de otro de tamaño de iteración n . Por otra parte el algoritmo de FFT se programo de tal manera en el que se dividian los datos y se utilizaba la recursividad para tener una mayor eficiencia. Podemos apreciar las comparaciones de los datos en la Figura 3 y en la Figura 4.

4. Conclusiones.

La Transformada Discreta de Fourier es util para el análisis de señales de tiempo discreto y dominio finito pero resulta muy ineficiente al tratar con muestras grandes y por ello es indicado utilizar FFT el

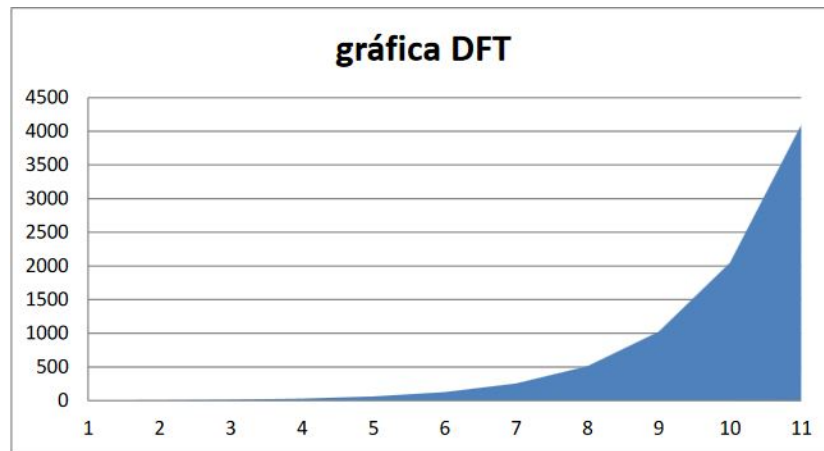


Figura 3: Ejecución del programa.

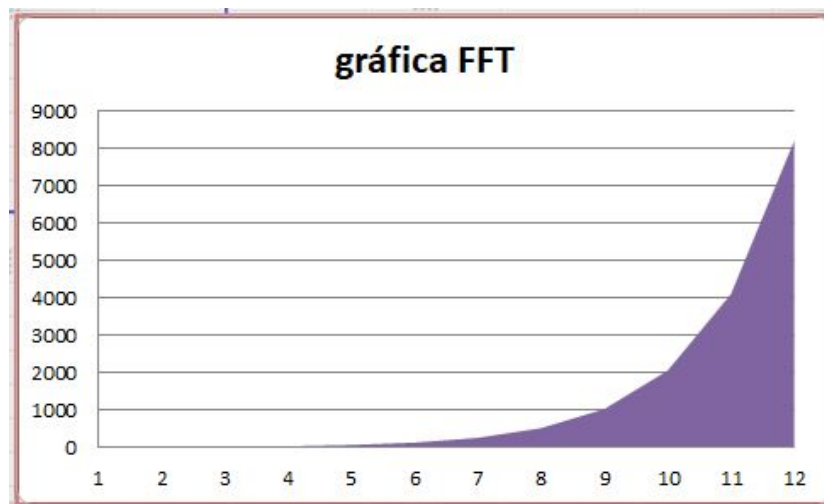


Figura 4: Ejecución del programa.

cual es exageradamente mas eficiente que el DFT.

A. Código fuente del Ejercicio 1.

```
#include <complex>
#include <iostream>
#include <valarray>
```

```

#include <time.h>
#include<fstream>

using namespace std;

int n = 0;

//se utilizo complex
typedef complex<double> Complex;

typedef valarray<Complex> CArray;

double funcion(double);

template <class TI>
void obtener_datos(TI data[])
{
    for (int i = 0; i < n; i++)
    {
        data[i] = funcion(i);
    }
}

// la funci n DFT.
void DFT(CArray &, Complex[]);

int main()
{
    cout << "El_numero_maximo_n=_";
    int p;
    cin >> p;
    fstream myfile;
    myfile.open("DFT_I.txt", fstream::out);
    clock_t t;
    for (int j = 4; j <= p; j +=2)

    {

```

```

        n = j;
        Complex *test = new Complex[n];
        obtener_datos(test);
        CArray data(test, n);
        t = clock();
        DFT(data, test);
        t = clock() - t;
        cout << "n=" << n << ", t=" << ((float)t) / CLOCKS_PER_SEC << endl;
        myfile << "{" << n << ", " << ((float)t) / CLOCKS_PER_SEC << "}" << endl;
        delete [] test;
    }
    myfile.close();

    return 0;
}

double funcion(double x)
{
    return 2 * sin(2 * M_PI / n * x) + 5 * cos(2 * M_PI / n * x);
}

void DFT(CArray &x, Complex test[])
{
    for (int i = 0; i < n; i++)
    {
        double suma_R = 0;
        double suma_i = 0;
        for (int k = 0; k < n; k++)
        {
            suma_R += test[k].real() * cos(2 * M_PI / n * k * i);
            suma_i -= test[k].real() * sin(2 * M_PI / n * k * i);
        }
        x[i].real(suma_R);
        x[i].imag(suma_i);
    }
}

```

B. Código fuente del Ejercicio 2.

```
#include <complex>
#include <iostream>
#include <valarray>
#include <time.h>
#include <fstream>

using namespace std;

int n = 8;

typedef complex<double> Complex;

typedef valarray<Complex> CArray;

double funcion(double);

template <class T> //identificador de tipo: T
void obtener_datos(T data[])
{
    for (int i = 0; i < n; i++)
    {
        data[i] = funcion(i);
    }
}

void FFT(CArray &);

int main()
{
    cout << "num. _maximo_n_=" ;
    long long int NN;
    cin >> NN;
    fstream myfile;
    myfile.open("FFT.txt", fstream::out);
    clock_t t;
```



```

for (long int j = 4; j <= NN; j *= 2)
{
    n = j;
    Complex *test = new Complex[n];
    obtener_datos(test);
    CArray data(test, n);

    t = clock();
    FFT(data);
    t = clock() - t;
    cout << "n=" << n << ", t=" << ((float)t) / CLOCKS_PER_SEC << endl;
    myfile << "{" << n << ", " << ((float)t) / CLOCKS_PER_SEC << "}" << endl;
    delete [] test;
}
myfile.close();

return 0;
}

```

```

double funcion(double x)
{
    return 2 * sin(2 * M_PI / n * x) + 5 * cos(2 * M_PI / n * x);
}

```

// Definimos la estructura de la funcion FFT.

```

void FFT(CArray &x)
{
    const size_t n = x.size();
    if (n <= 1)
        return;

    CArray even = x[slice(0, n / 2, 2)];
    CArray odd = x[slice(1, n / 2, 2)];

    // Recursividad
    FFT(even);

```

```

FFT(odd);

// Combinamos
for (size_t k = 0; k < n / 2; ++k)
{
    Complex t = polar(1.0, -2 * M_PI * k / n) * odd[k];
    x[k] = even[k] + t;
    x[k + n / 2] = even[k] - t;
}
}

```

Referencias

- [1] Dra. María del Pilar Gómez, Procesamiento digital de señales, Coordinación de computación, Versión: 11 de Octubre 2017.
- [2] Ana Martínez Manzano, Transformada rápida de Fourier Implementación y algunas aplicaciones, Facultad de Matemáticas, 2018.
- [3] Jimmy Alexander Cortés Osorio, Jairo Alberto Mendoza Vargas, José A. Muriel Escobar, “Alternativa al Análisis en Frecuencia de la FFT”, Universidad Tecnológica de Pereira. ISSN 0122-1701, No 44, Abril de 2010.
- [4] URL: https://www.tutorialspoint.com/mfc/mfc_array.htm(visitado 26/11/2020).
- [5] URL: <https://www.uv.es/diazj/complex.pdf> (visitado 29/12/2020).
- [6] URL: <https://codingornot.com/cc-plantillas-templates-en-c> (visitado 29/12/2020).
- [7] URL: <http://www.cplusplus.com/forum/beginner/91382/> (visitado 19/01/2020).
- [8] URL: <https://stackoverflow.com/questions/15231466/whats-the-difference-between-pi-and-m-pi-in-objc> (visitado 19/01/2020).