Universidad Católica
San Pablo

Transformada de Fourier

Docente: Rensso Victor Hugo Mora Colque

Integrantes:

Frank Roger Salas Ticona

Sebastian Gonzalo Postigo Avalos

Luis Fredy Huachaca Vargas

Angel Josue Loayza Huarachi

UCSP- Universidad Católica San Pablo
Junio de 2022

Índice general

Capítulo 1

Introducción

La transformada rápida de Fourier descompone una función de espacio o tiempo en las frecuencias que la forman, el resultado de esto es una función sinusoidal con valores complejos que representa la función original. La nueva función nos permite modificar y reconocer partes de la función original fácilmente en comparación de la función original. Al mismo tiempo, existe la inversa de Fourier para devolver a la función a su estado original. [3]

Capítulo 2

Conocimientos previos

Una lista de conocimientos previos para poder comprender como funciona la transformada de fourier en un nivel mas técnico.

2.1. Función sinusoidal

Los limites de una función sinusoidal es [1 -1] ya que $-1 < sen(x) < 1$, y forma parte de un ciclo con un periodo de 2π, por lo que siempre se esta repitiendo constantemente y sin cambios.[5]

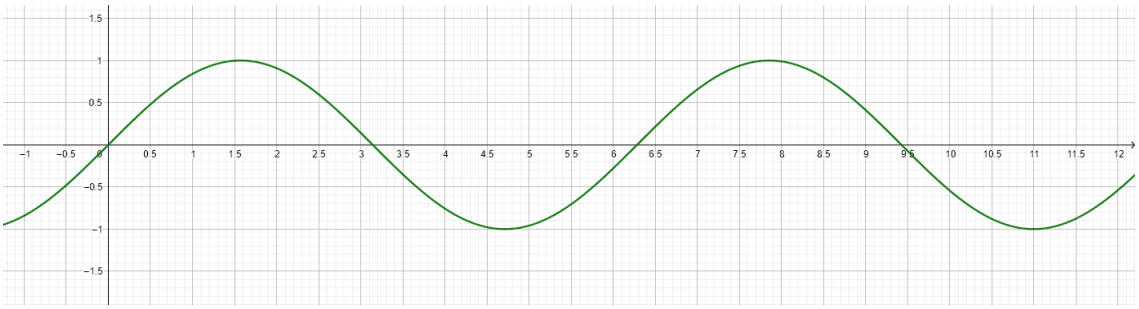

Figura 2.1: Función seno(x) en Geogebra

Lo mas importante de esta función, es que podemos modificar su amplitud, periodo y traslación, con esto podemos formar funciones como:

2.2. Números complejos

Ya que fourier nos sirve para representar una función de reales con una de números complejos, es importante saber algunas de sus propiedades. Para empezar un

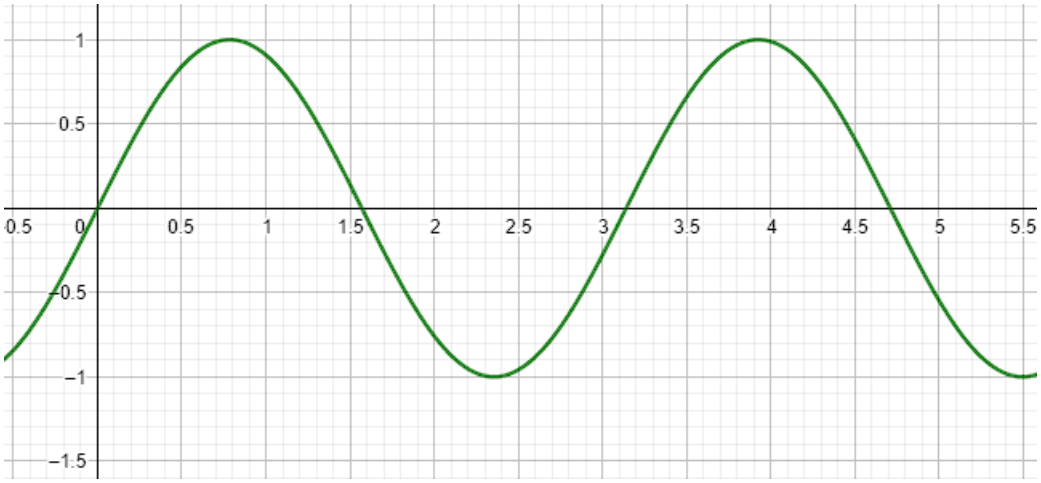

Figura 2.2: Periodo: seno(2x) en geogebra

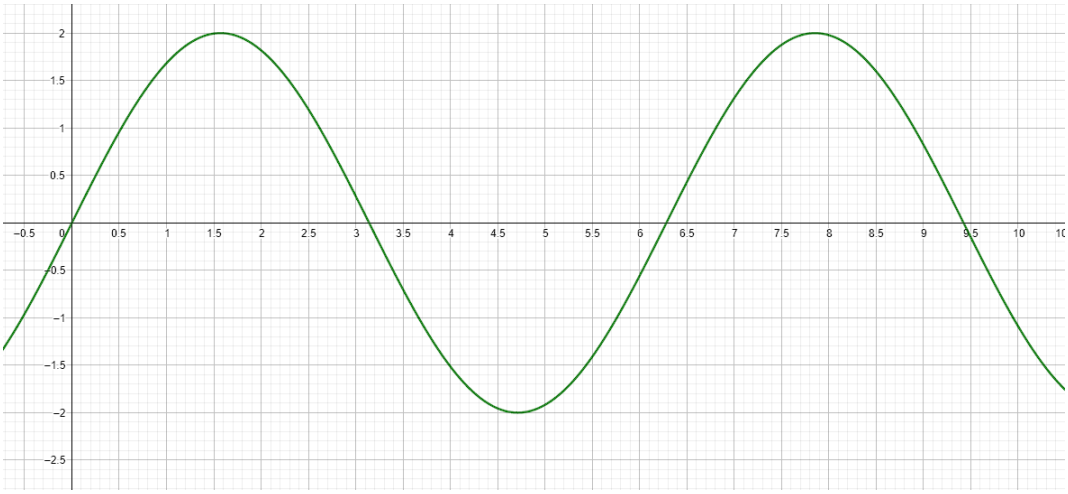

Figura 2.3: Amplitud: 2*seno(x) en geogebra

complejo es representado de la siguiente manera $z = a + ib$ donde a es el numero real y b es el numero imaginario, con esto podemos sacar las siguientes propiedades:

1. La magnitud de un complejo, representa a la amplitud de un sinusoidal complejo. Magnitud = $|a + b|$

2. con la ecuación $arctan(\frac{b}{a})$ podemos hallar el ángulo.

2.3. Formula de Euler

La formula de euler al usar e y potenciarlo a cualquier numero multiplicado por i siempre devolverá un punto en la circunferencia de un circulo con los limites [1, i, -1, -i]. También, sabemos que el numero multiplicado por i sera el numero de unidades que recorre en la circunferencia empezando en el punto 1 llendo en el sentido del reloj hasta llegar al punto e^{ix}, por lo que para completar una rotación en este circulo, x tendría que ser 2π. [4] [1]

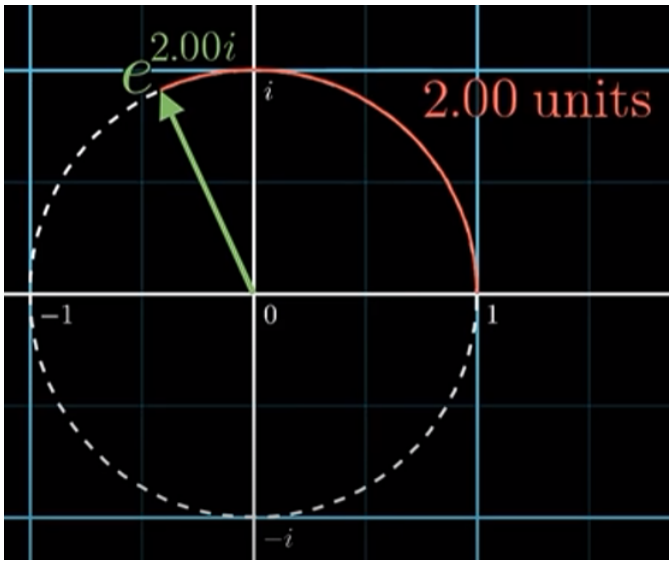

Figura 2.4: Identidad de euler.

Al mismo tiempo, por facilidad podemos escribir la formula de la siguiente manera:

$$e^{ix} = cos(x) + isen(x)$$

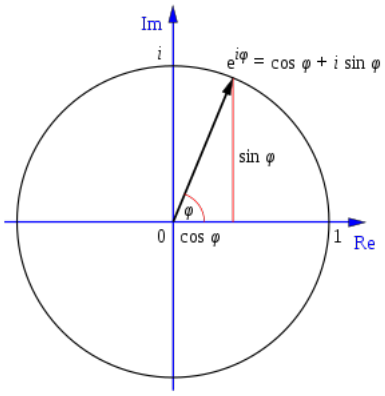

Figura 2.5: Formula de Euler.

2.4. Raíces de unidad

Las raíces de unidad son números complejos que al potenciarlos a un numero positivo, resultan en 1. [2] Se representan de la siguiente manera:

$$
\begin{aligned}
Z^n &= 1 \\
Z &= (cos(2\pi k) + isen(2\pi k))^{\frac{n}{1}} \ donde\ k > -1 \\
Z &= e^{2\pi k/n}
\end{aligned}
$$

Las raíces de unidad $w^n = (e^{2\pi i/n})^n$ formaran parte de un polígono con n lados en un circulo de unidad.

2.5. Diagrama de la mariposa

El diagrama de la mariposa en el contexto de FFT, se encarga de juntar pequeños DFTs de tamaño r(radix), que han sido ejecutados m veces, que antes serán multiplicados por la raíz de unidad w_N^k que sera igual a $e^{\frac{-2\pi ik}{n}}$, por ejemplo:

El diagrama de la mariposa recibe dos inputs (x0, x1) y dará dos outputs(y0, y1) por lo que tendremos las siguientes ecuaciones:

$$
\begin{aligned}
y0 &= x0 + x1w_n^k \\
y1 &= x0 - x1w_n^k
\end{aligned}
$$

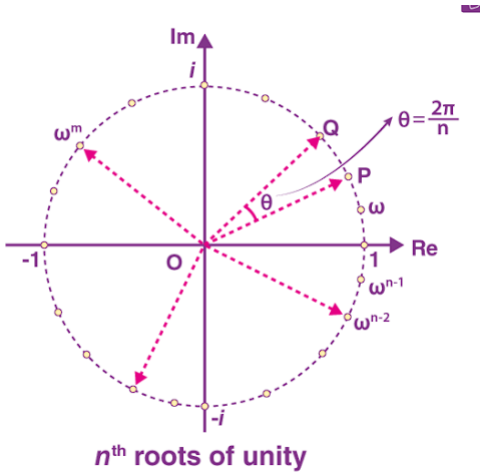

Figura 2.6: Raíces de unidad.

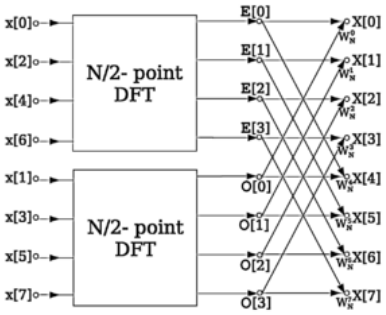

Figura 2.7: Diagrama de la mariposa

2.6. Como encontramos el Espectro

Cuando un numero infinito de ondas sinusoidales con amplitud muy pequeña, se juntan, el resultado puede verse como una onda, que podemos ver y medir. Como un conjunto de hojas de ancho cero.2.8

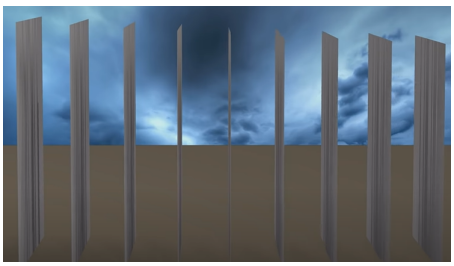

Figura 2.8: Hojas separadas

De la misma forma el volumen de cada hoja de papel es cero, cuando tenemos un infinito numero de hojas juntas, podremos medir la densidad de los objetos, la densidad en algunas partes serán mayores que la densidad en otra partes del proyecto.2.9

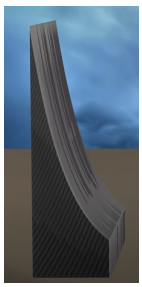

Figura 2.9: Hojas separadas

De la misma forma, cuando juntamos un numero infinito de ondas sinusoidales de amplitudes infinitamente pequeñas, podemos medir la densidad de las frecuencias, y esta densidad de frecuencias serán mas grandes que otras frecuencias cercanas. Esto es a lo que nos referimos como el espectro de frecuencia de una forma de onda.2.10

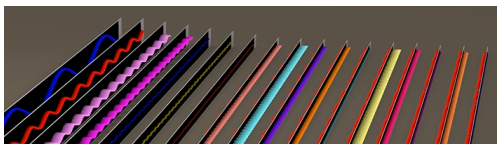

Figura 2.10: Frecuencias mas altas que otras

Capítulo 3

Formulas y Algoritmos

3.1. Transformada Directa de Fourier

Las ecuaciones principales de la transformada de fourier son:

$$X_k = \sum_{n=0}^{N-1} x_n * e^{-i2\pi kn/N}$$

Que representan la transformada de fourier normal y la inversa respectivamente, como podemos ver lo que hace es darnos una sumatoria de la función original multiplicado por la identidad de euler que tiene como valores exponenciales, -i porque tiene que ir al sentido opuesto del reloj, 2π (que nos da toda la circunferencia) y los demás valores representan la frecuencia y el tiempo que le vamos a dar a nuestra transformada. Como se había mencionado antes también se puede representar de la siguiente manera:

$$X_k = \sum_{n=0}^{N-1} x_n \times (cos(-i2\pi kn/N) + iseno(-i2\pi kn/N))$$

3.2. Inversa de Fourier

La inversa de Fourier es la misma formula que la misma transformada explicada anteriormente, la diferencia radíca en que se debe normalizar y esto se hace dividiendo entre N, y cambiando el signo del exponente.

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} X_k * e^{i2\pi kn/N}$$

De la misma forma que con la formula del DFT podemos representar esta formula de la inversa a través de una suma de senos y cosenos.

$$x_n = \frac{1}{N}\sum_{n=0}^{N-1} X_k \times (cos(i2\pi kn/N) + iseno(i2\pi kn/N))$$

3.3. Inversa de Fourier vs Transformada de Fourier

Cabe resaltar que la transformada de Fourier junto a su inversa guardan relación entre si. Pues luego de utilizar como base la representación integral de Fourier para sacar su transformada, esta nos daría una función de omega, que sera la misma transformada de Fourier, pero ahora dependiendo de la frecuencia. Mientras que la transformada inversa, resultara en una función en base al tiempo. En conclusión, este par de transformadas nos llevan desde la representación en el tiempo a una de la frecuencia (transformada de Fourier); y también desde una representación de la frecuencia al tiempo (transformada inversa de Fourier)

3.4. FFT (Cooley Tukey)

Es la implementación más usada del FFT. Una forma sencilla de entenderlo es a través de la multiplicación de matrices generada por el algoritmo del DFT original. Por motivos de ilustración consideremos una aplicación de Fourier para 1024 datos. I es una matriz identidad y D es una matriz diagonal que toma los valores de ω elevados a cierto exponente dependiendo de su posición en la matriz:

$$\begin{aligned}
\hat{x} &= F_{1024}x \\
\hat{x} &= \begin{bmatrix} I & -D \\ I & -D \end{bmatrix} \begin{bmatrix} F_{512} & 0 \\ 0 & F_{512} \end{bmatrix} \begin{bmatrix} X_{pares} \\ X_{impares} \end{bmatrix} \\
F_{512_1} &= \begin{bmatrix} I & -D \\ I & -D \end{bmatrix} \begin{bmatrix} F_{256} & 0 \\ 0 & F_{256} \end{bmatrix} \begin{bmatrix} X_{máltiplos\ de\ 4} \\ X_{2\ +\ múltiplos\ de\ 4} \end{bmatrix} \\
F_{512_2} &= \begin{bmatrix} I & -D \\ I & -D \end{bmatrix} \begin{bmatrix} F_{256} & 0 \\ 0 & F_{256} \end{bmatrix} \begin{bmatrix} X_{1+múltiplos\ de\ 4} \\ X_{3\ +\ múltiplos\ de\ 4} \end{bmatrix} \\
&\ldots
\end{aligned}$$

Al continuar de esta forma estamos descomponiendo Fourier en matrices de lado N/2, como podemos observar la naturaleza de este algoritmo es recursiva, y en cada

paso de la recursión divide N/2 y la matriz de los datos X con los cuales se tiene que multiplicar avanza de impar e impar pero viendolo de manera un poco más general, avanza la cantidad que divide.

3.5. DIT FFT (Decimation in Time)

Sacaremos la cantidad de bits que tiene el tamaño de nuestro vector y haremos bit-reversal a todo nuestro vector, si al revertir un numero este es mayor que el original se hará un intercambio entre las posición original y la revertida en el vector, haremos esto para poder aplicar el diagrama de la mariposa.

luego hallaremos el ángulo $= -2\pi/N$ para usar la identidad de euler $cos(angulo)+iseno(angulo)$ y aplicaremos el diagrama de la mariposa para hacerlo todo en un mismo vector y que la función no sea recursiva para ahorrar recursos. Despues, aplicaremos filtros y aplicaremos inversa de fourier y por ultimo, guardaremos el archivo en un nuevo archivo .wav.

Capítulo 4

Código

Para procesar el audio usamos la librería AudioFile que lee archivos .wav, luego creamos un vector de complejos donde guardaremos toda la data del audio y volveremos el tamaño de nuestro vector a la potencia de 2 mas cercana usando nuestra función findNextPowerOf2() que se encarga de encontrar la siguiente potencia de 2 mas cercana, y llenaremos de ceros el resto del vector hasta llegar a esa potencia.

4.1. FFT (Cooley Tukey)

```
void CooleyTukeyFFT::FFT(SampleArray& values)
{
        const size_t N = values.size();
        if (N <= 1)
            return;

        SampleArray evens = values[std::slice(0, N / 2, 2)];
        SampleArray odds = values[std::slice(1, N / 2, 2)];

        FFT(evens);
        FFT(odds);

        for (size_t i = 0; i < N / 2; i++) {
                ComplexVal index = std::polar(1.0, -2 * PI * i / N) * odds[i];
                values[i] = evens[i] + index;
                values[i + N / 2] = evens[i] - index;
        }
}
```

4.2. DIT FFT (Decimation in Time)

```
void fft(bool invert) {
        int reversal = 0;
        for (int i = 0; i < n; i++) {
                reversal = bit_reversal(i);
                if (i < reversal)
                    swap(audio_data[i],audio_data[reversal]);
                }

        for (int len = 2; len <= n; len <<= 1) {
                double ang = 2 * PI / len;
                if (!invert)
                        ang = ang * -1;
                complex<double> euler_w(cos(ang), sin(ang));

                for (int i = 0; i < n; i += len) {
                        complex<double> temp(1);
                        for (int j = 0; j < len / 2; j++) {
                        complex<double> x0 = audio_data[i + j];
                        complex<double> x1 = audio_data[i + j + len / 2] * temp;
                        audio_data[i + j] = x0 + x1;
                        audio_data[i + j + len / 2] = x0 - x1;
                        temp *= euler_w;
                        }
                        }
                }

        if (invert) {
                for (complex<double>& x : audio_data)
                        x /= n;
        }
}
```

Capítulo 5

Código en Github

El código puede ser analizado en su totalidad en el siguiente repositorio:

Link del repositorios.

Bibliografía

[1] YouTube, 2018. URL: https://www.youtube.com/watch?v=spUNpyF58BY&ab_channel=3Blue1Brown.

[2] Admin. *Nth root of unity - definition, properties, examples*. 2021. URL: https://byjus.com/maths/nth-root-of-unity.

[3] *An interactive guide to the Fourier transform*. URL: https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/.

[4] *Fórmula de Euler para números complejos*. URL: http://www.disfrutalasmatematicas.com/algebra/formula-euler.html.

[5] *Funciones Trigonomtricas*. URL: https://calculo.cc/temas/temas_bachillerato/primero_ciencias_sociales/funciones_elementales/teoria/seno.html.