Transformada de Fourier y su aplicaci ón en tratamiento de audio

Hecho por:

- javier Oswaldo Alvarez Reyes
- Rony Rodrigo Sicos Barrera
- Pablo Cesar Yucra Ccolqque

Transformada de Fourier

Se la deduce a partir de la Serie Compleja de Fourier, considerando el periodo T de la función f(t) tiende a infinito:

1. Serie compleja de Fourier

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left[a_n \cos\left(\frac{n\pi t}{L}\right) + b_n sen\left(\frac{n\pi t}{L}\right)\right]$$

Transformada de Fourier

Su resultado es una función compleja que depende de la frecuencia angular w.

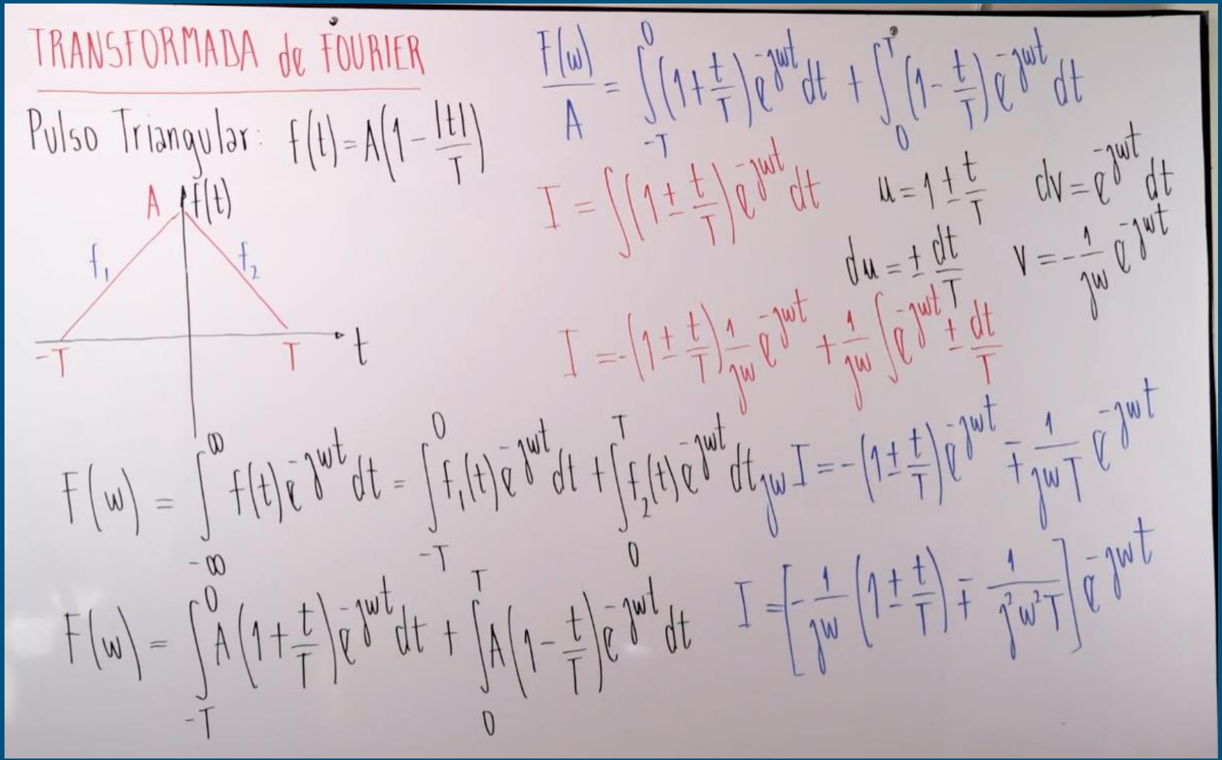
Descomponiendo F(w), podemos obtener los diagramas de fase y magnitud en función de w, que se denomina ESPECTRO DE MAGNITUD Y FASE.

Si f(t) tiene simetría par, su transformación es real.

Si f(t) tiene simetría impar, su transformada es imaginaria.

$$F(w) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j \cdot \omega \cdot t} \cdot dt$$

Transformada de Fourier



Transformada de Fourier

TRANSFORMADA de FOURIER

$$I = \left[-\frac{1}{jw}\left(1 \pm \frac{t}{T}\right) \mp \frac{1}{j^2w^2T}\right] e^{-jwt}$$

$$\frac{F(w)}{A} = \int_{-T}^{0}\left(1+\frac{t}{T}\right)e^{-jwt}\,dt + \int_{0}^{T}\left(1-\frac{t}{T}\right)e^{-jwt}\,dt$$

$$\frac{F(w)}{A} = \left.\left[-\frac{1}{jw}\left(1+\frac{t}{T}\right) - \frac{1}{j^2w^2T}\right]e^{-jwt}\right|_{-T}^{0} + \left.\left[-\frac{1}{jw}\left(1-\frac{t}{T}\right)+\frac{1}{j^2w^2T}\right]e^{-jwt}\right|_{0}^{T}$$

$$jw\frac{F(w)}{A} = \left[-(1) - \frac{1}{jwT}\right]e^{0} + \left[(1-1)+\frac{1}{jwT}\right]e^{jwT} + \left[-1(1-1)+\frac{1}{jwT}\right]e^{-jwT} - \left[-1(1)+\frac{1}{jwT}\right]e^{0}$$

$$\frac{jw\cdot F(w)}{A} = -1 - \frac{1}{jwT} + \frac{1}{jwT}e^{jwT} + \frac{1}{jwT}e^{-jwT} + 1$$

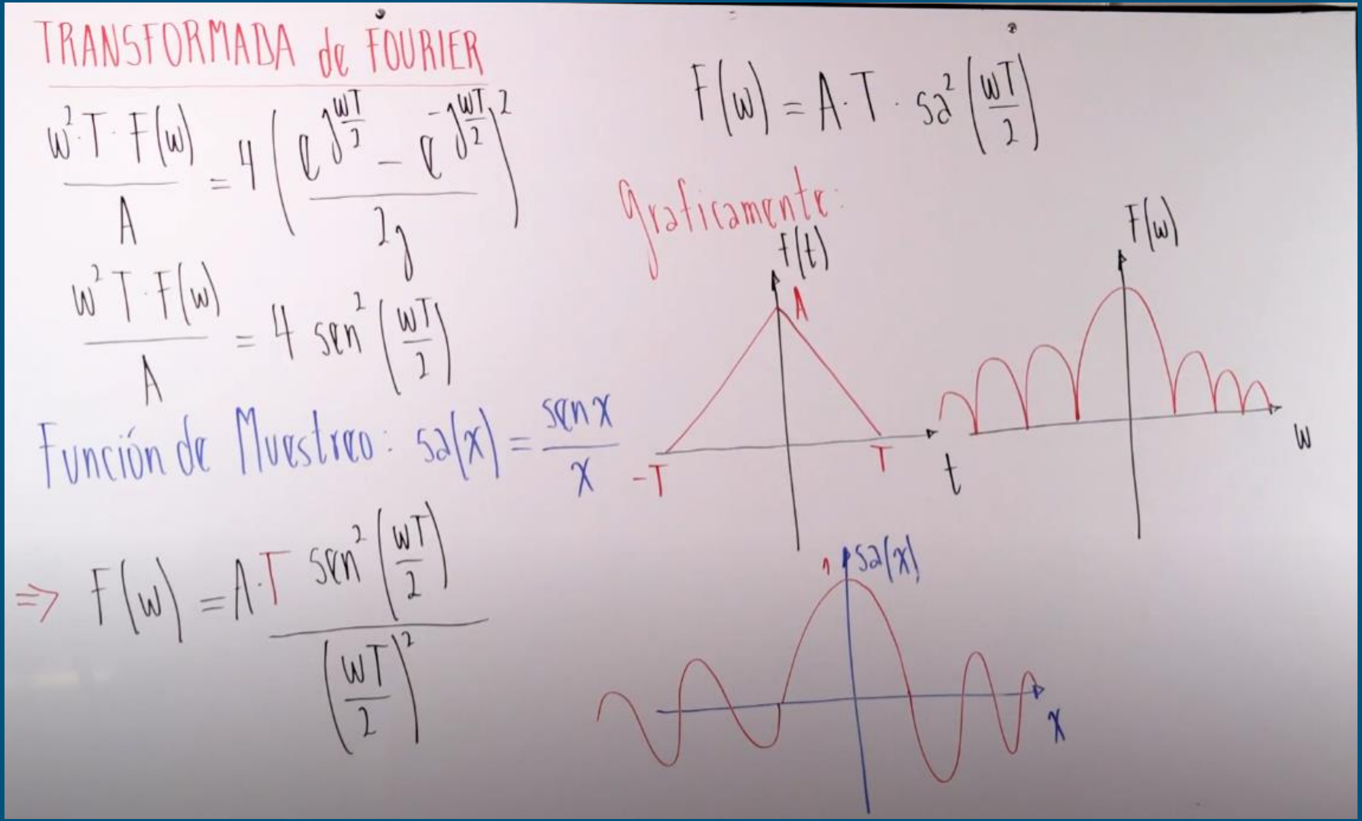
Transformada de Fourier

$$\frac{j\omega \cdot F(\omega)}{A} = \cancel{-1} - \cancel{\frac{1}{j\omega T}} + \frac{1}{j\omega T} e^{j\omega T} + \frac{1}{j\omega T} e^{-j\omega T} \cancel{+1} - \cancel{\frac{1}{j\omega T}}$$

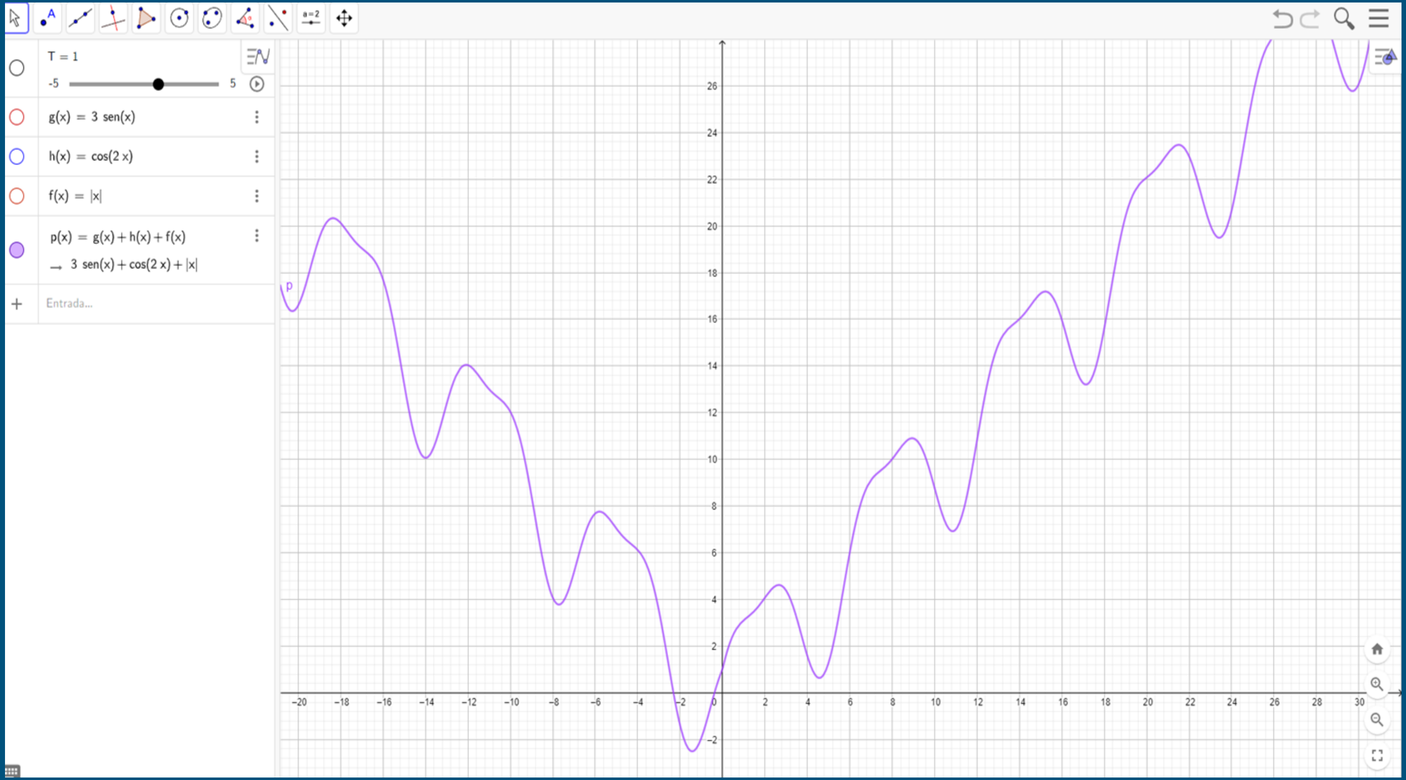
$$\frac{j^2\omega^2 T \cdot F(\omega)}{A} = -2 + e^{j\omega T} + e^{-j\omega T} = \left(e^{j\frac{\omega T}{2}} - e^{-j\frac{\omega T}{2}}\right)^2$$

$$\frac{\omega^2 T \cdot F(\omega)}{A} = \left(\frac{e^{j\frac{\omega T}{2}} - e^{-j\frac{\omega T}{2}}}{2j}\right)^2 \cdot 4$$

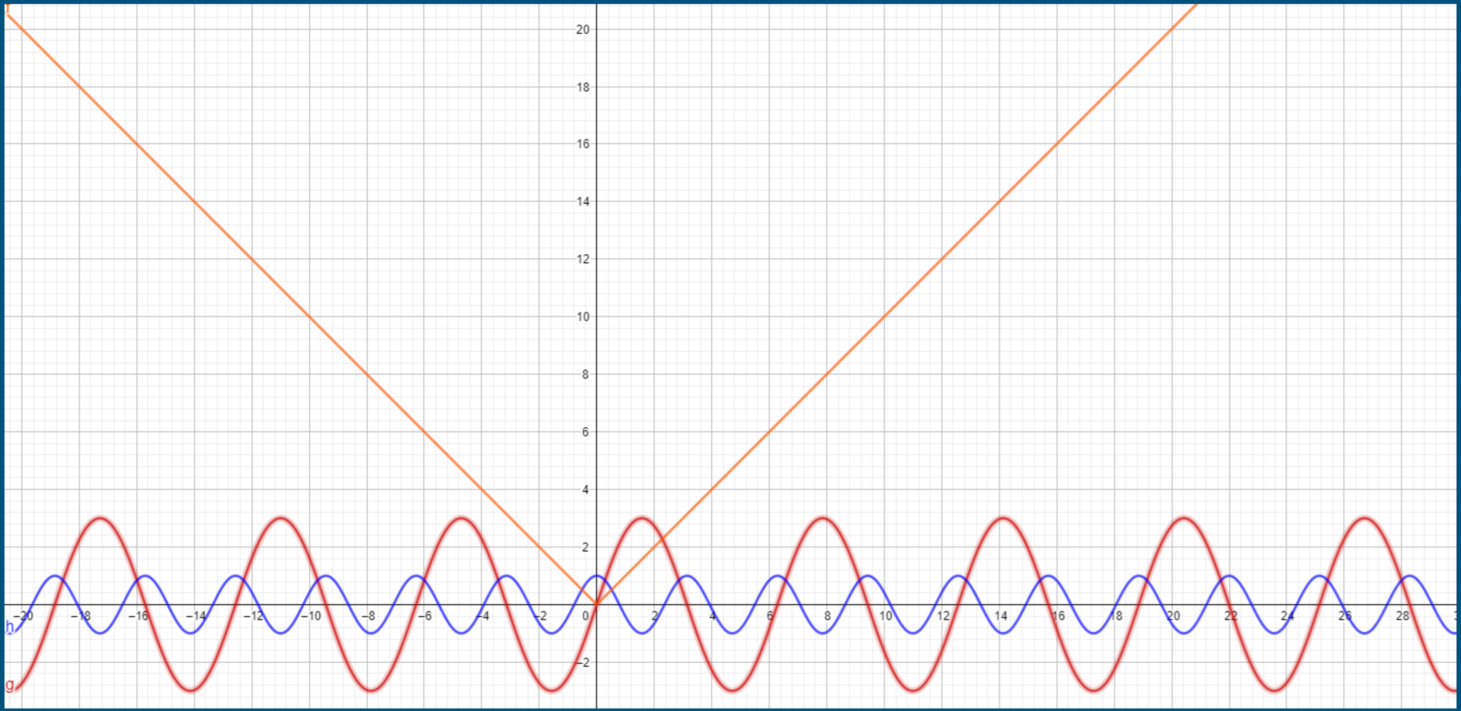
Transformada de Fourier



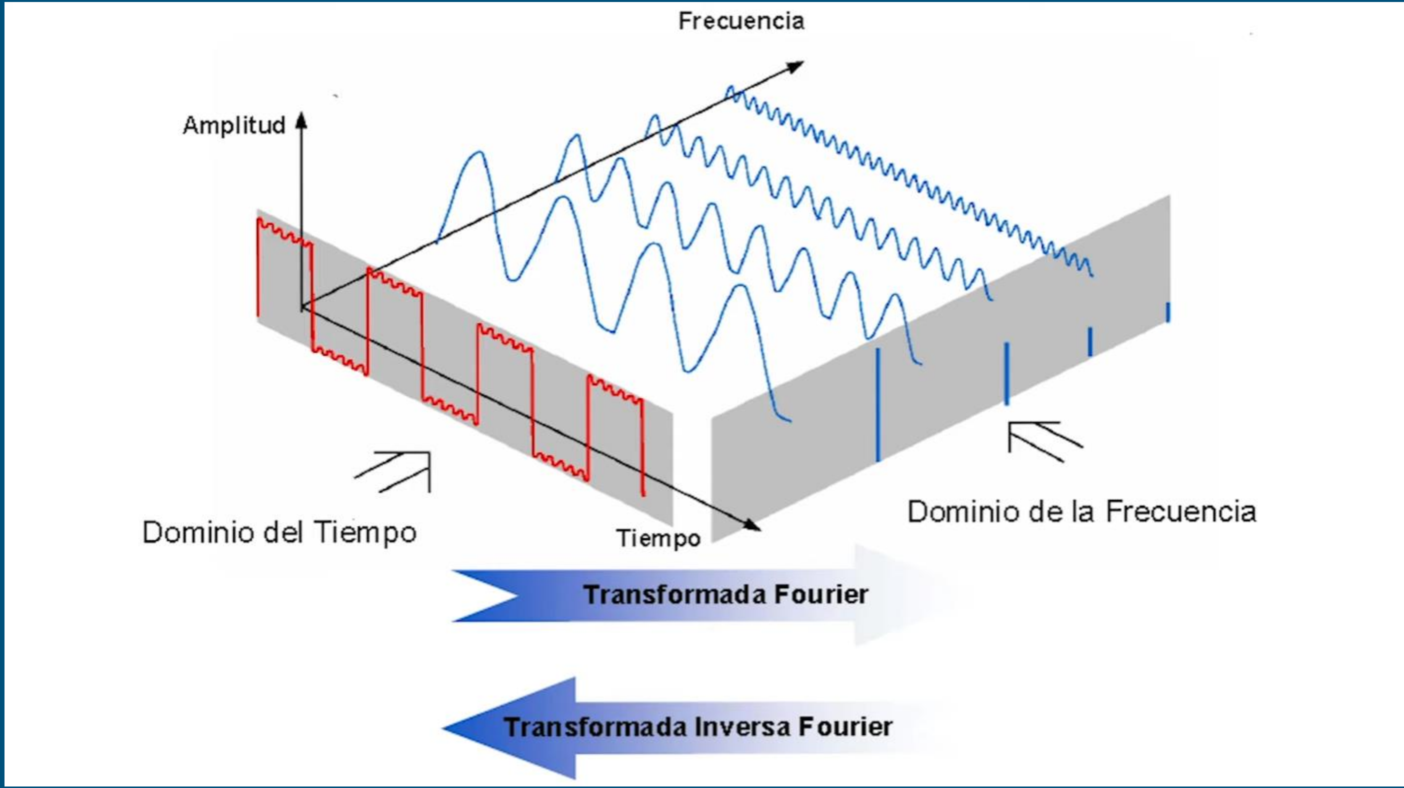
Transformada de Fourier



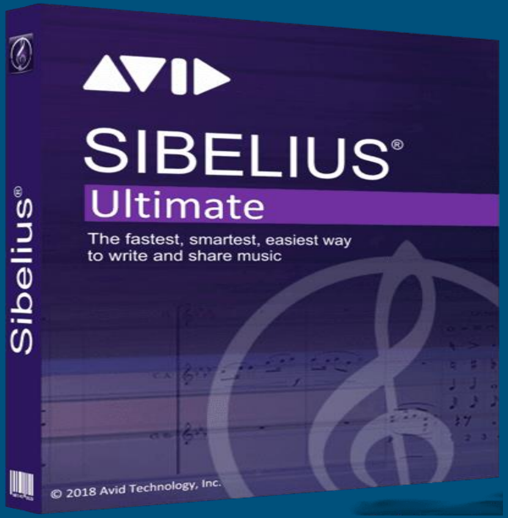
Transformada de Fourier



Transformada de Fourier

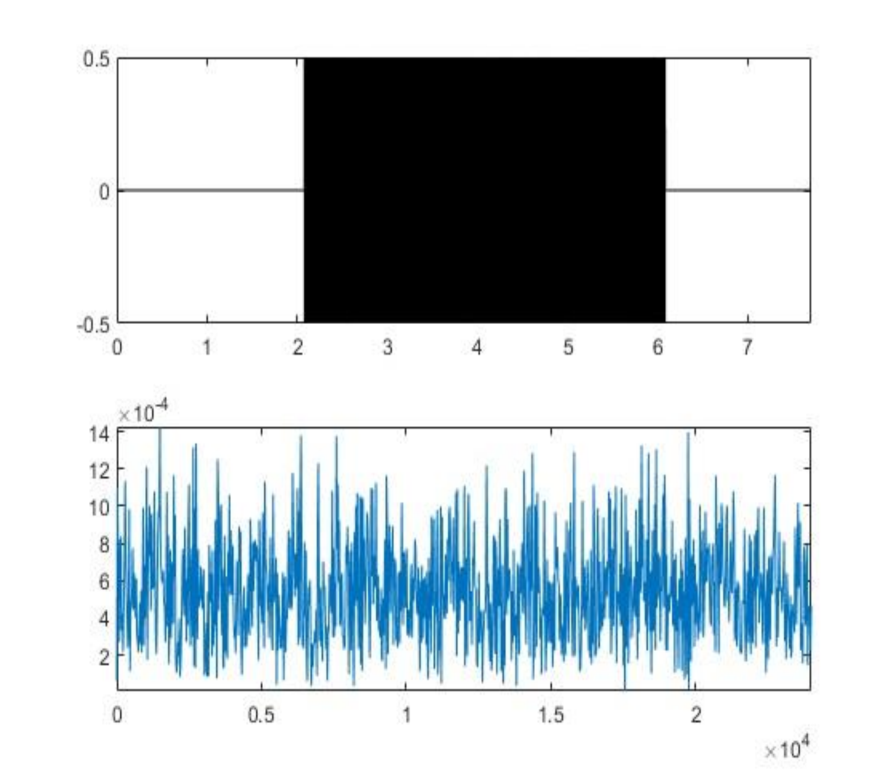


Fourier y su aplicación en tratamiento de audio

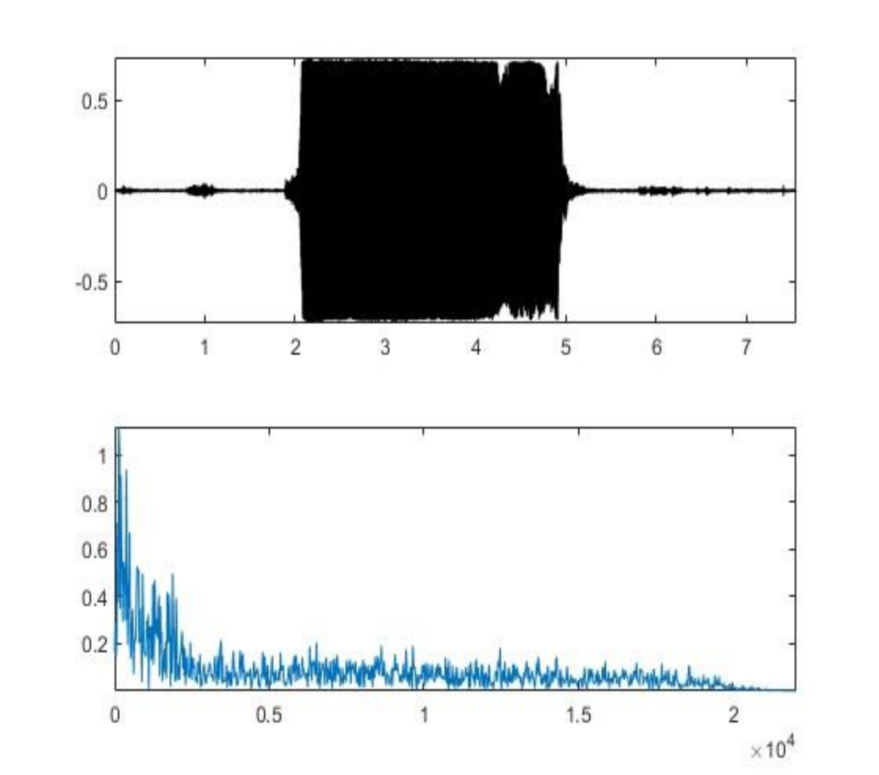


Sonido

BEEP



Flauta



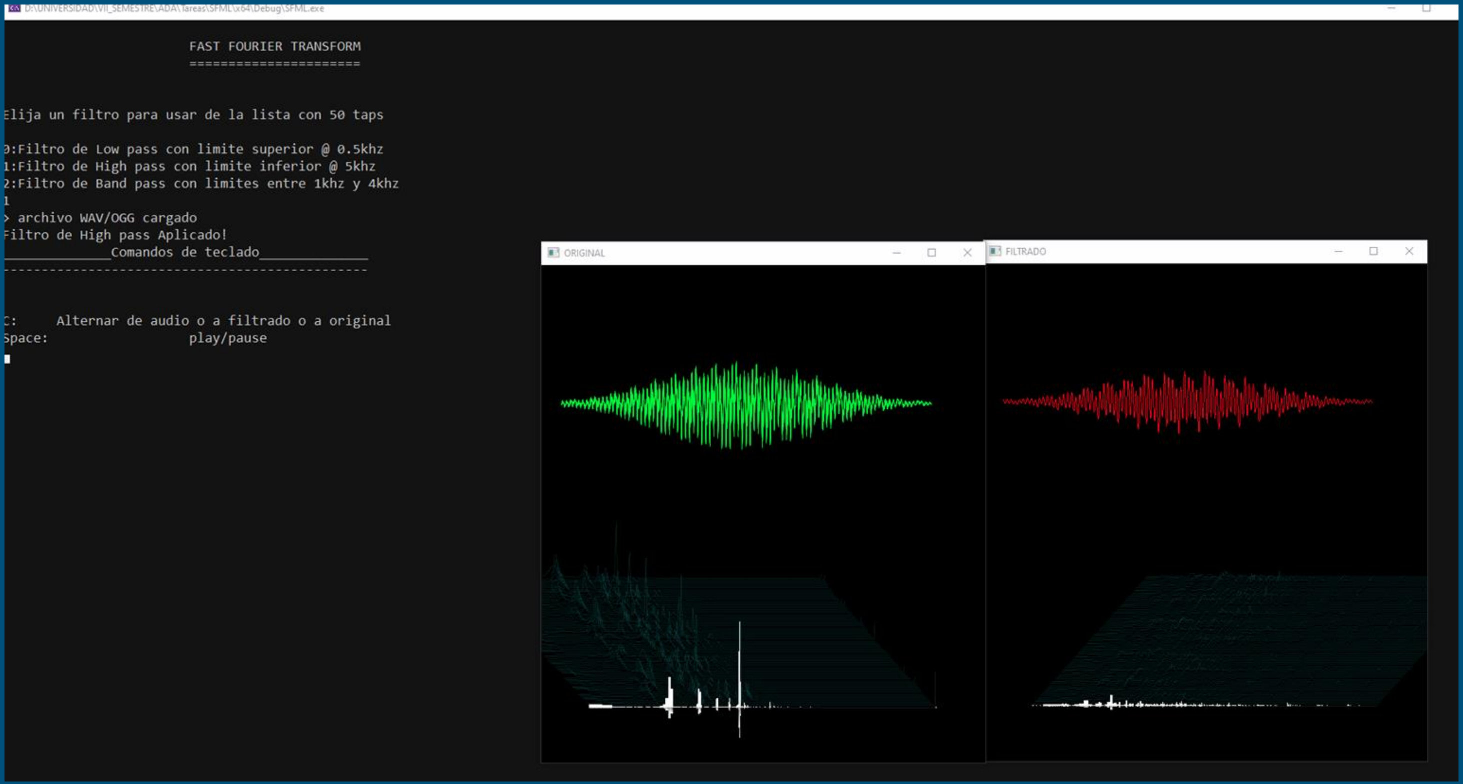
Almacenamiento

```
std::vector<sf::Int16> samples = ...;

buffer.loadFromSamples(&samples[0], samples.size(), 2, 44100);
```

Esta clase encapsula los datos de audio, que son básicamente una matriz de enteros con signo de 16 bits (llamados 'muestras de audio'). Una muestra es la amplitud de la señal de sonido en un momento dado y, por lo tanto, una matriz de muestras representa un sonido completo

Primero: Los Resultados



DATOS PREVIOS

```
typedef complex<double> Complex;    //almacen de numeros complejos (real, imaginario)
typedef valarray<Complex> CArray;   //se utiliza para realizar operaciones matemáticas en cada elemento de la matriz mas facil
```

```
VertexArray VA1;
VertexArray VA2;
VertexArray VA3;

VertexArray fSignalDeTiempo; //va1 señal de tiempo ya filtrado //array de vertices en la consola para dibujar
VertexArray fBarras; //va2- filter barras ya filtrado
VertexArray fCascada; //va3 - efecto cascada frequency ya filtrado
```

Filtros:

Los filtros están diseñados utilizando el "método de la serie de Fourier". Esto significa que los coeficientes de una aproximación de la serie de Fourier a la respuesta de frecuencia de un filtro ideal (LPF, HPF, BPF) se utilizan como derivaciones de filtro. Los filtros resultantes tienen alguna ondulación en la banda de paso por el fenómeno de Gibbs; los filtros son de fase lineal.

```
void Filter::designLPF()
{
    int n;
    double mm;

    for (n = 0; n < m_num_taps; n++) {
        mm = n - (m_num_taps - 1.0) / 2.0;
        if (mm == 0.0) m_taps[n] = m_lambda / M_PI;
        else m_taps[n] = sin(mm * m_lambda) / (mm * M_PI);
    }

    return;
}
```

```
void Filter::designHPF()
{
    int n;
    double mm;

    for (n = 0; n < m_num_taps; n++) {
        mm = n - (m_num_taps - 1.0) / 2.0;
        if (mm == 0.0) m_taps[n] = 1.0 - m_lambda / M_PI;
        else m_taps[n] = -sin(mm * m_lambda) / (mm * M_PI);
    }

    return;
}
```

```
void Filter::designBPF()
{
    int n;
    double mm;

    for (n = 0; n < m_num_taps; n++) {
        mm = n - (m_num_taps - 1.0) / 2.0;
        if (mm == 0.0) m_taps[n] = (m_phi - m_lambda) / M_PI;
        else m_taps[n] = (sin(mm * m_phi) - sin(mm * m_lambda)) / (mm * M_PI);
    }

    return;
}
```

CÓDIGO (MAIN)

```cpp
#include "FFT.h"

using namespace std;
using namespace sf;

int main()
{
    RenderWindow window(VideoMode(900, 900, 32), "ORIGINAL");

    RenderWindow window2(VideoMode(900, 900, 32), "FILTRADO");

    string path;
    int bufferSize;
    int numero;
    bool origen = false;

    cout << "\n\t\t\tFAST FOURIER TRANSFORM\n" <<
            "\t\t\t======================\n\n\n" <<

            "Elija un filtro para usar de la lista con 50 taps\n\n" <<
            "0:Filtro de Low pass con limite superior @ 0.5khz\n" <<
            "1:Filtro de High pass con limite inferior @ 5khz\n" <<
            "2:Filtro de Band pass con limites entre 1khz y 4khz\n";
    cin >> numero;

    ClaseFourier fft("Wavs/Gymnopédie No. 1.wav", 16384, numero);

    Event event; //para capturar pulsaciones de teclas, pulsaciones de ratón

    while (window.isOpen() && window2.isOpen())
    {
        while (window.pollEvent(event) || window2.pollEvent(event))
        {
            if (event.type == Event::Closed)
            {
                window.close();
            }
            //PAUSE
            if (event.type == sf::Event::KeyPressed)
            {
                if (event.key.code == sf::Keyboard::Space)
                {
                    fft.ppSound();
```


```cpp
            //PAUSE
            if (event.type == sf::Event::KeyPressed)
            {
                if (event.key.code == sf::Keyboard::Space)
                {
                    fft.ppSound();
```


```cpp
                }
                else if (sf::Keyboard::isKeyPressed(sf::Keyboard::C))
                {
                    fft.switchSource(origen);
                    origen = !origen;
                }
            }
        }

        fft.update();

        window.clear();
        window2.clear();


        fft.draw(window);
        fft.drawF(window2);

        window.display();
        window2.display();
    }

    return 0;
}
```

CONSTRUCTOR(clase FFT)

```cpp
ClaseFourier::ClaseFourier(string const& _path, int const& _bufferSize, int fc)
{
    FiltroEscogido = fc;
    PrecalculoDeFiltros();

    path = _path; //lugar donde esta la musica
    if (!buffer.loadFromFile(path)) //verifica si se puede leer la musica
    {
        cout << "No se puede cargar la Musica" << endl;
    }
    else
    {
        cout << "> archivo WAV/OGG cargado" << endl;
    }

    sound.setBuffer(buffer); //apuntamos al búfer que declaramos anteriormente para leer los datos del sonido


    sound.setVolume(0.0f);
    updateFilterSound();    //crear un búfer del sonido con la señal filtrada

    sound.play();
    fsound.play();          //toca la musica


    /*Listas para Graficar*/

    //dominio del tiempo
    VA1.setPrimitiveType(LineStrip);    //Lista de líneas conectadas, un punto usa el punto anterior para formar una línea
    fSignalDeTiempo.setPrimitiveType(LineStrip);

    //frequency domain
    VA2.setPrimitiveType(Lines);        // Lista individual de lineas
    fBarras.setPrimitiveType(LineStrip);

    //cascade background
    VA3.setPrimitiveType(LineStrip);    //lista de lineas conectadas
    fCascada.setPrimitiveType(LineStrip);
```

```cpp
bool FFT::PrecalculoDeFiltros()
{
    generalFilter.push_back(new Filter(LPF, 50, 44.1, 0.5));
    generalFilter.push_back(new Filter(HPF, 50, 44.1, 5.0));
    generalFilter.push_back(new Filter(BPF, 50, 44.1, 1.0, 4.0));

```

```cpp
void ClaseFourier::updateFilterSound()
{
    lfSamples = new Int16[buffer.getSampleCount()];
    double tempSample;
    switch (FiltroEscogido)
    {
    case 0:
        for (int i = 0; i < buffer.getSampleCount(); i++)
        {
            tempSample = (double)buffer.getSamples()[i];
            lfSamples[i] = lpfdataSample(tempSample);
        }
        cout << "Filtro de Low pass Aplicado!" << endl;
        break;
    case 1:
        for (int i = 0; i < buffer.getSampleCount(); i++)
        {
            tempSample = (double)buffer.getSamples()[i];
            lfSamples[i] = hpfdataSample(tempSample);
        }
        cout << "Filtro de High pass Aplicado!" << endl;
        break;
    case 2:
        for (int i = 0; i < buffer.getSampleCount(); i++)
        {
            tempSample = (double)buffer.getSamples()[i];
            lfSamples[i] = bpfdataSample(tempSample);
        }
        cout << "Filtro de Band pass Aplicado!" << endl;
        break;
    }

    fbuffer.loadFromSamples(lfSamples, buffer.getSampleCount(), buffer.getChannelCount(), buffer.getSampleRate())
    fsound.setBuffer(fbuffer);

}
```

¿Dónde Llamamos al Fast Fourier Transform?

```
    ClaseFourier fft("Wavs/Gymnopédie No. 1.wav", 16384, numero);

    Event event; //para capturar pulsaciones de teclas, pulsaciones de ratón

    while (window.isOpen() && window2.isOpen())
    {
        while (window.pollEvent(event) || window2.pollEvent(event))
        {
            if (event.type == Event::Closed)
            {
                window.close();
            }
            //PAUSE
            if (event.type == sf::Event::KeyPressed)
            {
                if (event.key.code == sf::Keyboard::Space)
                {
                    fft.ppSound();
                }
                else if (sf::Keyboard::isKeyPressed(sf::Keyboard::C))
                {
                    fft.switchSource(origen);
                    origen = !origen;
                }
            }
        }

        fft.update();

        window.clear();
        window2.clear();


        fft.draw(window);
        fft.drawF(window2);

        window.display();
        window2.display();
    }

    return 0;
}
```

Update

```cpp
void ClaseFourier::update()
{
    hammingWindow();

    VA2.clear();
    fBarras.clear();
    VA3.clear();
    fCascada.clear();


    //transformar el tiempo (muestra de musica) en el dominio de la frecuencia
    //bin de inicio y bin filtrado ( valores de datos de muestra, tamaño de búfer )
    bin = CArray(sample.data(), bufferSize);
    fbin = CArray(fsample.data(), bufferSize);

    fft(bin, fbin);

    float max = 100000000;
    lines(max);
    bars(max);
}
```

FAST FOURIER TRANSFORM

```cpp
/*
* FAST FOURIER TRANSFORM
* ----------------------
* calcula la DISCRETE FOURIER TRANSFORM en O(nlogn) aplicando divide y vencerás
*/
void ClaseFourier::fft(CArray& x, CArray& fx)
{
    const int N = x.size();
    if (N <= 1) return; //regresa al elemento anterior de la pila

    //Pares
    CArray Pares = x[slice(0, N / 2, 2)]; //cortamos (offset, número de elementos en el nuevo corte, pasos entre elementos de array)
    CArray fPares = fx[slice(0, N / 2, 2)];

    //Impares
    CArray  Impares = x[slice(1, N / 2, 2)];
    CArray fImpares = fx[slice(1, N / 2, 2)];

    fft(Pares, fPares);
    fft(Impares, fImpares);


    for (int k = 0; k < N / 2; k++)
    {
        Complex t = polar(1.0, -2 * PI * k / N) * Impares[k];
        //cout << t;;
        Complex ft = polar(1.0, -2 * PI * k / N) * fImpares[k];
        //cout << "\t" << ft << endl;

        x[k] = Pares[k] + t;
        fx[k] = fPares[k] + ft;

        x[k + N / 2] = Pares[k] - t;
        fx[k + N / 2] = fPares[k]; -ft;
    }
}
```