ALGORITMO COOLEY TUKEY FFT

Integrantes:

-Sebastian Paz Ballón

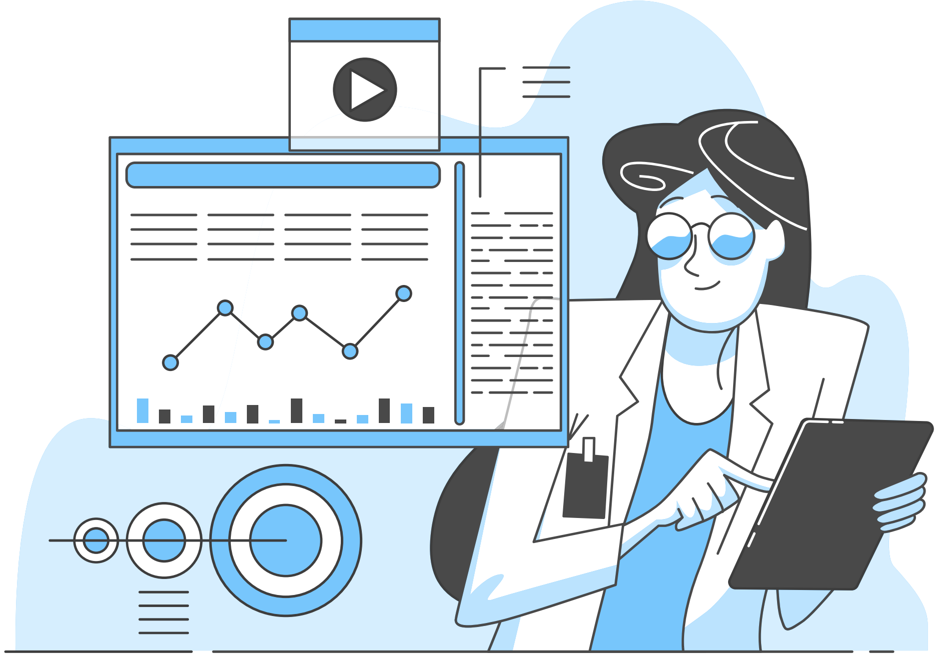
-Sebastian Ugarte Concha

-Sharon Valdivia Begazo

01 Introducción

02 Explicación del Algoritmo

03 Código



01

Introducción

Introducción

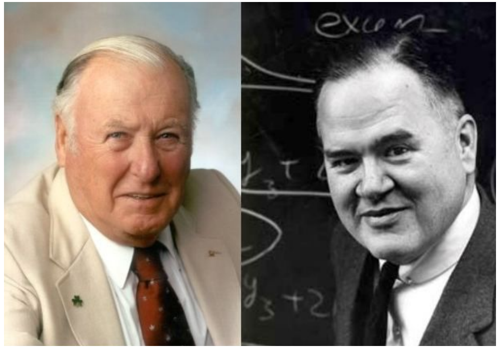
Gauss

1805

Divide y vencerás

DFT

. . .



J.W. Cooley J.W. Tukey

Abril - 1965

Transformada rápida de Fourier

Cooley Tukey FFT

. . .

Acuerdos SALT entre la Unión Soviética y Estados Unidos para limitar el nro de misiles nucleares

. . .

15km

Introducción

Este algoritmo nos permite reducir la complejidad de la DFT

$$O(n^2) \longrightarrow O(n \log_2(n))$$

DFT FFT

Disminuyendo el número de multiplicaciones y sumas complejas

02

Explicación del Algoritmo

Explicación del Algoritmo

Partimos de la Transformada Discreta de Fourier (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

$$W_N = e^{-j\frac{2\pi}{N}}$$

Explicación del Algoritmo

Donde se describe el cálculo de N ecuaciones. Por ejemplo si tomamos N = 4 la ecuación puede ser descrita como:

$$X(0) = x_0(0)W^0 + x_0(1)W^0 + x_0(2)W^0 + x_0(3)W^0$$
$$X(1) = x_0(0)W^0 + x_0(1)W^1 + x_0(2)W^2 + x_0(3)W^3$$
$$X(2) = x_0(0)W^0 + x_0(1)W^2 + x_0(2)W^4 + x_0(3)W^6$$
$$X(1) = x_0(0)W^0 + x_0(1)W^3 + x_0(2)W^6 + x_0(3)W^9$$

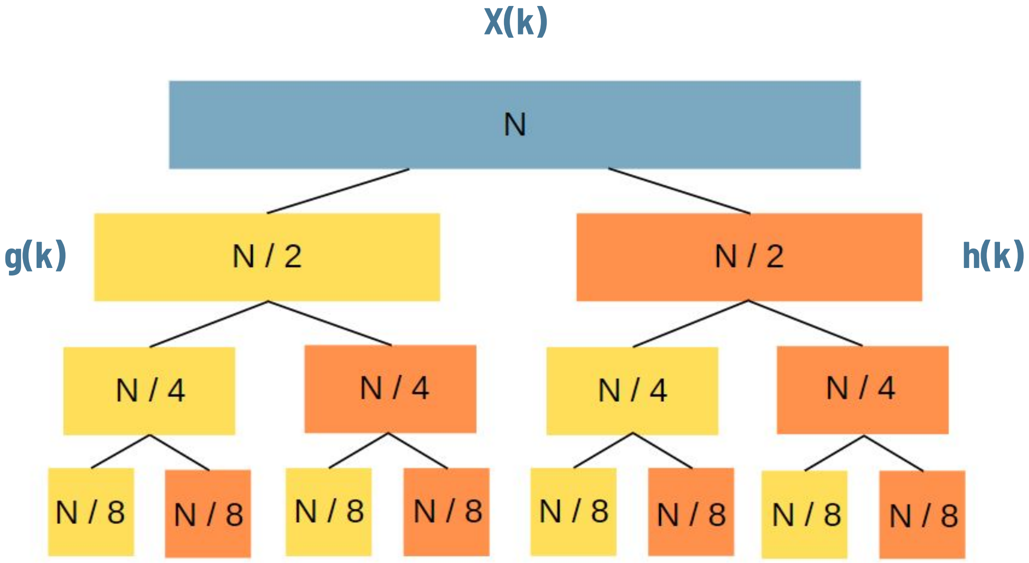
Se puede representar de forma matricial

Son necesarias N^2 multiplicaciones complejas y N(N-1) sumas complejas

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x_0(0) \\ x_0(1) \\ x_0(2) \\ x_0(3) \end{bmatrix}$$

Explicación del Algoritmo

El FFT está basado en "divide y vencerás" porque descompone la DFT original **(X(k))** en DFTs más pequeñas **(g(k) y h(k))**



Por ello **N** tiene que ser una potencia entera de 2

Explicación del Algoritmo

Partimos de la Transformada Discreta de Fourier:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad W_N = e^{-j\frac{2\pi}{N}}$$

Podemos reescribirla separando las muestras pares e impares de la siguiente forma:

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_N^{(2r+1)k}$$

x(0), x(2), x(4), x(2r) x(1), x(3), x(5), x(2r+1)

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)W_N^{(2r+1)k}$$

Se factorizan los exponentes señalados:

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_N^2)^{rk}$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_N^2)^{rk}$$

Tales expresiones equivalen a lo siguiente:

$$W_N = e^{-j\frac{2\pi}{N}} \rightarrow W_N^2 = \left(e^{-j\frac{2\pi}{N}}\right)^2$$

$$W_N^2 = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_N^2)^{rk}$$

$$W_N^2 = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$$

Por lo tanto, las podemos reemplazar en la expresión:

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk}$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk}$$

Tomamos un total de 8 muestras:

$$x(k) = [0,1,2,3,4,5,6,7]$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk}$$

$$x(k) = [0,1,2,3,4,5,6,7]$$

$$g(k) = [0,2,4,6]$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk}$$

$$x(k) = [0,1,2,3,4,5,6,7]$$

$$g(k) = [0,2,4,6]$$

$$h(k) = [1,3,5,7]$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk}$$

$$x(k) = [0,1,2,3,4,5,6,7]$$

$$g(k) = [0,2,4,6]$$

$$h(k) = [1,3,5,7]$$

Explicación del Algoritmo

$$x(k) = [0,1,2,3,4,5,6,7]$$

$$g(k) = [0,2,4,6]$$

$$\sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} = \sum_{n=0}^{N-1} g(n)(W_N)^{nk}$$

Transformada discreta de Fourier

$$\rightarrow \quad G(k) = \sum_{n=0}^{N-1} g(n)(W_N)^{nk}$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk}$$

$$x(k) = [0,1,2,3,4,5,6,7]$$

$$g(k) = [0,2,4,6]$$

$$h(k) = [1,3,5,7]$$

...

Explicación del Algoritmo

$$x(k) = [0,1,2,3,4,5,6,7]$$

$$h(k) = [1,3,5,7]$$

$$\sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk} = \sum_{n=0}^{N-1} h(n)(W_N)^{nk}$$

Transformada discreta de Fourier

$$\rightarrow \quad H(k) = \sum_{n=0}^{N-1} h(n)(W_N)^{nk}$$

Explicación del Algoritmo

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)(W_{N/2})^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)(W_{N/2})^{rk}$$

Entonces podemos reemplazar los términos que hallamos:

$$X(k) = G(k) + W_N^k H(k)$$

Explicación del Algoritmo

$$X(k) = G(k) + W_N^k H(k)$$

Como **G(k)** y **H(k)** tienen periodo **N/2** y **X(k)** tiene periodo **N**, entonces se tiene 2 periodos de cada una, así que la calcularemos en 2 mitades de la siguiente forma:

Donde k recorre de 0 a N/2

Primera mitad: $$X(k) = G(k) + W_N^k H(k)$$

Segunda mitad: $$X\left(k + \frac{N}{2}\right) = G\left(k + \frac{N}{2}\right) + W_N^{k+\frac{N}{2}} H(k + \frac{N}{2})$$

Explicación del Algoritmo

Segunda mitad:
$$X\left(k+\frac{N}{2}\right)=G\left(k+\frac{N}{2}\right)+W_N^{k+\frac{N}{2}}H\left(k+\frac{N}{2}\right)$$

Como G(k) y H(k) son periódicos y su periodo es igual a N/2, entonces **G(k) = G(k+N/2)** y **H(k)=H(k+N/2)**

Reemplazando esto nos queda:

$$X\left(k+\frac{N}{2}\right)=G(k)+W_N^{k+\frac{N}{2}}H(k)$$

Explicación del Algoritmo

$$X\left(k+\frac{N}{2}\right) = G(k) + W_N^{k+\frac{N}{2}} H(k)$$

Vemos que las expresiones señaladas equivalen a lo siguiente:

$$W_N = e^{-j\frac{2\pi}{N}} \rightarrow W_N^{k+\frac{N}{2}} = \left(e^{-j\frac{2\pi}{N}}\right)^{k+\frac{N}{2}}$$

$$W_N^{k+\frac{N}{2}} = e^{-j\frac{2\pi}{N}k} \cdot e^{-j\frac{2\pi N}{N2}}$$

$$W_N^{k+\frac{N}{2}} = W_N^k \cdot e^{-j\pi}$$

$$W_N^{k+\frac{N}{2}} = -W_N^k$$

Según la identidad de Euler:

$$e^{-j\pi} = -1$$

Explicación del Algoritmo

$$X\left(k+\frac{N}{2}\right)=G(k)+W_N^{k+\frac{N}{2}}H(k)$$

Y al reemplazar la equivalencia obtenida, tenemos:

$$X\left(k+\frac{N}{2}\right)=G(k)-W_N^{k}H(k)$$

Explicación del Algoritmo

Por último, tenemos que nuestro X(k) se puede definir en dos mitades:

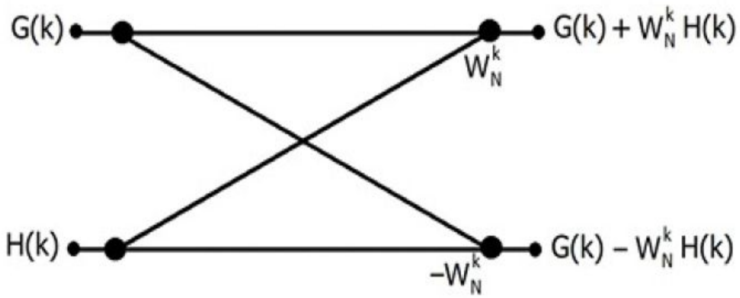
Primera mitad: $$X(k) = G(k) + W_N^k H(k)$$

Segunda mitad: $$X(k) = G(k) - W_N^k H(k)$$

...

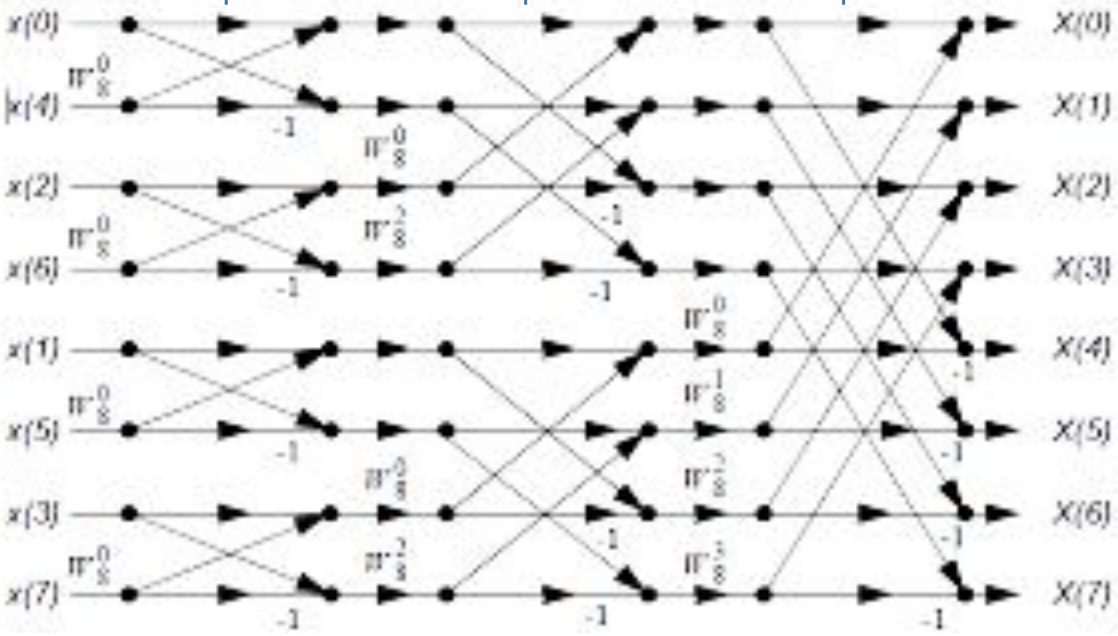
Explicación del Algoritmo

Podemos representar estas ecuaciones a través de un diagrama de flujo o también conocido como **Mariposa Básica de Fourier**



Explicación del Algoritmo

Etapa 1 Etapa 2 Etapa 3



x(0)
x(1)
x(2)
x(3)
x(4)
x(5)
x(6)
x(7)

Etapas = $\log_2 N = \log_2 8 = 3$

Explicación del Algoritmo

$x(n) = \{-1, 2, 0, 0, 4, 0, -3, 1\}$

$x(0) = 1$

$x(1) = -4.29 - j5.12$

$x(2) = 6 - j3$

$x(3) = -5.7 + j0.878$

$x(4) = -1$

$x(5) = -5.7 - j0.878$

$x(6) = 6 + j3$

$x(7) = -4.29 + j5.12$

03

Código

Explicación del Código

Para este código se creó una clase FFT con cuatro funciones, dos para la primera mitad del cálculo de fourier y otras dos para hallar la inversa, a continuación un ejemplo de una función para la primera parte

```
bool CFFT::Forward(complex *const Data, const unsigned int N)
{

  if (!Data || N < 1 || N & (N - 1))
    return false;

  Rearrange(Data, N);

  Perform(Data, N);

  return true;
}
```

Explicación del Código

La función Rearrange usa el concepto de “matemáticas espejo” para definir una nueva posición para cada elemento y cambia los elementos para obtener el nuevo flujo de datos

```
void CFFT::Rearrange(complex *const Data, const unsigned int N)
{
   unsigned int Target = 0;
   for (unsigned int Position = 0; Position < N; ++Position)
   {
      if (Target > Position)
      {
         const complex Temp(Data[Target]);
         Data[Target] = Data[Position];
         Data[Position] = Temp;
      }
      unsigned int Mask = N;
      while (Target & (Mask >>= 1))
         Target &= ~Mask;
      Target |= Mask;
   }
}
```

Explicación del Código

Ahora para hablar de la función principal, esta función será usada por ambas mitades por lo que en la primera línea se puede ver que el signo correspondiente a cada exponente se establece

```
void CFFT::Perform(complex *const Data, const unsigned int N,
   const bool Inverse )
{
   const double pi = Inverse ? 3.14159265358979323846 :
-3.14159265358979323846;
```

Las inicializaciones dentro del bucle externo son solo preparativos para el cálculo sucesivo de factores a través de la recurrencia trigonométrica. Y el trabajo realizado dentro del bucle interior, que realiza operaciones de mariposa.

```
for (unsigned int Step = 1; Step < N; Step <<= 1)
  {
    const unsigned int Jump = Step << 1;
    const double delta = pi / double(Step);
    const double Sine = sin(delta * .5);
    const complex Multiplier(-2. * Sine * Sine, sin(delta));
    complex Factor(1.);
    for (unsigned int Group = 0; Group < Step; ++Group)
    {
      for (unsigned int Pair = Group; Pair < N; Pair += Jump)
      {
        const unsigned int Match = Pair + Step;
        const complex Product(Factor * Data[Match]);
        Data[Match] = Data[Pair] - Product;
        Data[Pair] += Product;
      }
      Factor = Multiplier * Factor + Factor;
    }
  }
```

Explicación del Código

Y ahora le toca el turno a las funciones usadas a la hora de resolver la parte inversa del cálculo, donde se puede ver que en diferencia a Forward existe una función más

```
bool CFFT::Inverse(complex *const Data, const unsigned int N,const bool Scale /* = true */)
{
   if (!Data || N < 1 || N & (N - 1))
      return false;
   Rearrange(Data, N);
   Perform(Data, N, true);
   if (Scale)
      CFFT::Scale(Data, N);
   return true;
}
```

Explicación del Código

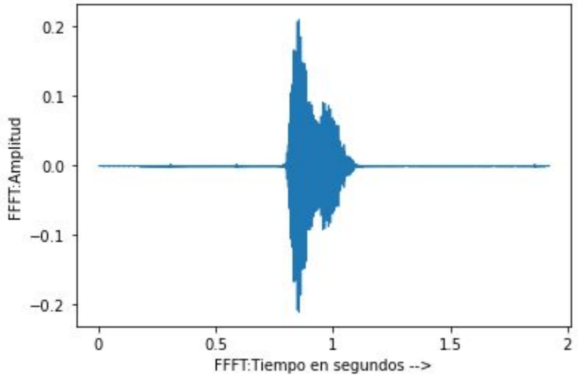
La función Scale se encarga de darle el correcto escalado condicional a los datos una vez ya han sido calculados, la escala se produce basándose en

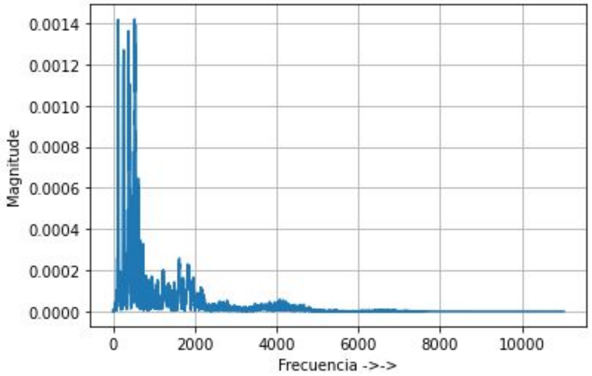
$$f_k = \frac{1}{N}\sum_{n=0}^{N-1} F_n e^{i\frac{2\pi}{N}kn}$$

```
void CFFT::Scale(complex *const Data, const unsigned int N)
{
   const double Factor = 1. / double(N);
   for (unsigned int Position = 0; Position < N; ++Position)
      Data[Position] *= Factor;
}
```

Explicación del Código

Una vez ya se tiene la clase se usa con la entrada de datos que provienen de nuestra muestra de audio, el audio es analizado en python que guarda los valores en un .txt que recibe el código en C++, donde este es procesado con la función FFT para obtener los nuevos datos que serán guardados en otro txt que será leído por nuestro código en python para hallar el gráfico.





```
int main(int argc, char* argv[])
{
    vector<double> data;
    string line;
    ifstream infile("test.txt");
    double numero;
    while (getline(infile, line)) {
        istringstream iss(line);

        if (iss >> numero) {
            data.push_back(numero);
        }
    }
    complex<double> Signal1[42336];
    for (int i = 0; i < data.size(); i++) {
        Signal1[i] = data[i];
    }
    ofstream fw("test1.txt");
    TFFT<double>::Forward(Signal1, 42336);
    std::cout << "re: ";
    for (unsigned int i = 0; i < 42336; ++i) {
        std::cout << Signal1[i].re() << " ";
        fw << Signal1[i].re()<<"\n";
    }
    return 0;
}
```

GRACIAS POR SU ATENCIÓN