2.1.3 La Transformada Rápida de Fourier

Esta es una herramienta que se utiliza para hacer análisis de señales entre otras aplicaciones, su dominio es el de la frecuencia y el de los números complejos, el 1805 en el que Gauss creo el algoritmo de Divide y Vencerás, este se empleó para realizar el cálculo de la Transformada de Fourier Discreta recursiva.

El 1965, J. W. Cooley y J. W. Tukey crearon un algoritmo más eficiente para realizar el cálculo, al cual llamaron Transformada Rápida de Fourier, para comprender dicho algoritmo es necesario definir en primer lugar la transformada discreta de Fourier DFT (siglas en inglés).

La DFT se define como un cambio de coordenadas en $\mathbb{C}^N$ (complejos) mediante $\Im \mathfrak{n}: \mathbb{C}^N \to \mathbb{C}^N$ en donde cada vector $f \in \mathbb{C}^N$ le hace corresponder el vector $\hat{f} = (\hat{f}(0), \hat{f}(1), \ldots . \hat{f}(N-1))$ en donde

$$\hat{f}(k) = \sum_{n=0}^{N-1} f(n) e^{\frac{-2i\pi}{N} kn} \qquad (1)$$

Para $0 \leq n \leq N-1$, su operación inversa la DFTI sería

$$f(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{f}(k) e^{\frac{2i\pi}{N} nk} \qquad (2)$$

Análisis de la DFT

El cálculo de la DFT empleando la ecuación (1) puede resultar costoso, de tal forma que el número de operaciones (multiplicaciones y sumas) a realizar para calcular el vector $(\hat{f}(0), \hat{f}(1), \ldots . \hat{f}(N-1))$ sería el siguiente

- $\hat{f}(0) = \sum_{n=0}^{N-1} f(n) e^{-0}$ con 0 multiplicaciones complejas y (N-1) sumas complejas
- $\hat{f}(1) = \sum_{n=0}^{N-1} f(n) e^{-\frac{2\pi i}{N} n}$ con (N-1) multiplicaciones complejas y (N-1) sumas complejas
- $\hat{f}(N-1) = \sum_{n=0}^{N-1} f(n) e^{-\frac{2\pi i}{N} n(N-1)}$ con (N-1) multiplicaciones complejas y (N-1) sumas complejas

Por lo tanto para calcular cada vector $(\hat{f}(0), \hat{f}(1), \ldots . \hat{f}(N-1))$ se requieren N(N-1) sumas complejas y (N-1)(N-1) multiplicaciones complejas, es decir $N(n-1) + (N-1)^2$ operaciones, si N=1000, entonces se tendrían que hacer más de un millón de operaciones de cada tipo. El coste de dicho cálculo se vuelve más oneroso si se trabajara con señales en dos o tres dimensiones. La complejidad será entonces $\Theta(N^2)$.

Los mencionados anteriormente, Cooley y Tukey crearon el algoritmo de la transformada rápida de Fourier (FFT, siglas en inglés), inspirado en muchas investigaciones de algebra para el procesamiento de señales e imágenes y también para minimizar el coste del cálculo de la DFT.

El código para programar la DFT requiere de una clase Complejo, una clase para realizar aritmética compleja, una clase que realice el cálculo mismo de la DFT y una que tenga el main. Ellas se muestran en el código 2.1.3. En la clase CalculadoraDFT se incluyen cuatro señales, un pulso cuadrado, un delta de dirac, un senoide y una rampa.

Código 2.1.3.1 Clases para el cálculo de la DFT de distintas señales.

```
/**
 *
 * @author sdelaot
 */
public class Complejo {
    private double real;
    private double imaginario;
    public Complejo() {
        this( 0.0, 0.0 );
    }
    public Complejo(double real, double imaginario) {
        this.real = real;
        this.imaginario = imaginario;
    }
    public void setComplejo(double real, double imaginario) {
        this.real = real;
        this.imaginario = imaginario;
    }
    public Complejo getComplejo() {
        return this;
    }
    public double getReal() {
        return real;
    }
    public void setReal(double real) {
        this.real = real;
    }
    public double getImaginario() {
        return imaginario;
    }
    public void setImaginario(double imaginario) {
        this.imaginario = imaginario;
    }
    public double calcularMagnitud() {
```

```
        return Math.sqrt(Math.pow(real, 2.0)+Math.pow(imaginario, 2.0));
    }
    public double calcularArgumento() {
        return Math.atan2(imaginario, real);
    }
    public Complejo conjugar() {
        return new Complejo( real, (-1)*imaginario );
    }
    @Override
    public String toString() {
        return "(" + real + ", i " + imaginario + ')';
    }
}
/**
 *
 * @author sdelaot
 */
public class AritmeticaCompleja {
    private Complejo z;
    public Complejo sumar( Complejo z1, Complejo z2 ) {
        z = new Complejo( z1.getReal()+z2.getReal(),
              z1.getImaginario()+z2.getImaginario() );
        return z;
    }
    public Complejo restar( Complejo z1, Complejo z2 ) {
        z = new Complejo( z1.getReal()-z2.getReal(),
                    z1.getImaginario()-z2.getImaginario() );
        return z;
    }
    public Complejo multiplicar( Complejo z1, Complejo z2 ) {
        z = new Complejo( z1.getReal()*z2.getReal()-
                    z1.getImaginario()*z2.getImaginario(),
                    z1.getReal()*z2.getImaginario()+
                    z1.getImaginario()*z2.getReal() );
        return z;
    }
    public Complejo dividir( Complejo z1, Complejo z2 ) {
        double potencia = Math.pow(z2.getReal(), 2.0) +
                    Math.pow(z2.getImaginario(), 2.0);
        double zReal = z1.getReal()*z2.getReal()+
                  z1.getImaginario()*z2.getImaginario();
        double zImaginario = z1.getImaginario()*z2.getReal()-
                     z1.getReal()*z2.getImaginario();
        z = new Complejo( zReal/potencia, zImaginario/potencia );
        return z;
    }
}
```

```
/**
 *
 * @author sdelaot
 */
public class TransformadaFourierDiscreta {
   private Complejo [] dft;
   private AritmeticaCompleja aritmetica;
   public TransformadaFourierDiscreta( int N ) {
      dft = new Complejo[N];
      aritmetica = new AritmeticaCompleja();
   }
   Complejo [] calcularTransformadaDirecta( Complejo [] f ) {
      int N = dft.length;
      if( N!=f.length ) {
         return null;
         }
      Complejo e;
      Complejo sumatoria = new Complejo();
      for( int k=0; k<N; k++ ) {
         sumatoria.setComplejo( 0.0, 0.0 );
         // DEBUG
         //System.out.println( sumatoria );
         for( int n=0; n<N; n++ ) {
            double x = (2.0*Math.PI*k*n)/N;
            e = new Complejo(Math.cos(x),-1.0*Math.sin(x));
            Complejo multiplica = aritmetica.multiplicar(f[n], e);
            sumatoria = aritmetica.sumar(sumatoria, multiplica);
            }
         // DEBUG
         //System.out.println( sumatoria );
         double real = sumatoria.getReal();
         double imag = sumatoria.getImaginario();
         if( Math.abs(real)<1.25e-10 ) {
            real = 0.0;
            }
         if( Math.abs(imag)<1.25e-10 ) {
            imag = 0.0;
            }
         dft[k] = new Complejo( real, imag );
         }
      return dft;
   }
}
/**
 *
 * @author sdelaot
 */
```

```
public class CalculadoraDFT {
   public static void main(String[] args) {
      int N = 16;
      Complejo [] f = { // Pulso cuadrado
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 )
         };
      Complejo [] fs = { // senoidal cos(nwT), w=2*PI*f
         new Complejo( Math.cos( 0.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 1.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 2.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 3.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 4.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 5.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 6.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 7.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 8.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos( 9.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos(10.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos(11.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos(12.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos(13.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos(14.0*2.0*Math.PI/N), 0.0 ),
         new Complejo( Math.cos(15.0*2.0*Math.PI/N), 0.0 )
         };

      Complejo [] fi = { // Inpulso delta
         new Complejo( 1.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
```

```
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 ),
    new Complejo( 0.0, 0.0 )
    };
Complejo [] fr = { // rampa
    new Complejo( 0.125, 0.0 ),
    new Complejo( 0.250, 0.0 ),
    new Complejo( 0.375, 0.0 ),
    new Complejo( 0.500, 0.0 ),
    new Complejo( 0.625, 0.0 ),
    new Complejo( 0.750, 0.0 ),
    new Complejo( 0.875, 0.0 ),
    new Complejo( 1.000, 0.0 ),
    new Complejo( 0.125, 0.0 ),
    new Complejo( 0.250, 0.0 ),
    new Complejo( 0.375, 0.0 ),
    new Complejo( 0.500, 0.0 ),
    new Complejo( 0.625, 0.0 ),
    new Complejo( 0.750, 0.0 ),
    new Complejo( 0.875, 0.0 ),
    new Complejo( 1.000, 0.0 )
    };
TransformadaFourierDiscreta transformada =
        new TransformadaFourierDiscreta( f.length );
System.out.println( "\nPulso cuadrado" );
for( Complejo z: f ) {
    System.out.print( " " + z );
    }
System.out.println( "\nDFT Pulso cuadrado" );
Complejo [] dft = transformada.calcularTransformadaDirecta(f);
for( Complejo z: dft ) {
    System.out.print( " " + z );
    }
System.out.println( "\nMagnitud Pulso cuadrado" );
for( Complejo z: dft ) {
    System.out.print( " " + z.calcularMagnitud() );
    }
System.out.println();
System.out.println( "\nDelta de dirac" );
for( Complejo z: fi ) {
```

```
            System.out.print( " " + z );
            }
        System.out.println( "\nDFT Delta de dirac" );
        dft = transformada.calcularTransformadaDirecta(fi);
        for( Complejo z: dft ) {
            System.out.print( " " + z );
            }
        System.out.println( "\nMagnitud delta dirac" );
        for( Complejo z: dft ) {
            System.out.print( " " + z.calcularMagnitud() );
            }
        System.out.println();
        System.out.println( "\nSenoidal" );
        for( Complejo z: fs ) {
            System.out.print( " " + z );
            }
        System.out.println( "\nDFT Senoidal" );
        dft = transformada.calcularTransformadaDirecta(fs);
        for( Complejo z: dft ) {
            System.out.print( " " + z );
            }
        System.out.println( "\nMagnitud Senoidal" );
        for( Complejo z: dft ) {
            System.out.print( " " + z.calcularMagnitud() );
            }
        System.out.println();
        System.out.println( "\nRampa" );
        for( Complejo z: fr ) {
            System.out.print( " " + z );
            }
        System.out.println( "\nDFT Rampa" );
        dft = transformada.calcularTransformadaDirecta(fr);
        for( Complejo z: dft ) {
            System.out.print( " " + z );
            }
        System.out.println( "\nMagnitud Rampa" );
        for( Complejo z: dft ) {
            System.out.print( " " + z.calcularMagnitud() );
            }
        System.out.println();
    }
}
```

Definición de la FFT

La Transformada Rápida de Fourier se define como el algoritmo para realizar el cálculo de la Transformada Discreta de Fourier y reducir la complejidad de $\Theta(N^2)$ a $\Theta(Nlog(N))$ en donde N es el tamaño de entrada del algoritmo.

Para realizar el cálculo de la DFT dado el vector $f = (f_0, f_1, \ldots\ldots, f_{N-1})$ de tamaño N, para calcular el vector $\hat{f} = (\hat{f}_0, \hat{f}_1, \ldots\ldots, \hat{f}_{N-1})$, donde

$$\hat{f}(k) = \sum_{n=0}^{N-1} f(n) e^{\frac{-2i\pi}{N}kn} \qquad (3)$$

Con k=0,….,N, basta con calcular el producto de la matriz $\bar{M}$ por el vector columna f, donde

$$\bar{M} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \overline{w_N} & \overline{w_N}^2 & \cdots & \overline{w_N}^{(N-1)} \\ 1 & \overline{w_N}^2 & \overline{w_N}^4 & \cdots & \overline{w_N}^{2(N-1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \overline{w_N}^{(N-1)} & \overline{w_N}^{2(N-1)} & \cdots & \overline{w_N}^{(N-1)(N-1)} \end{bmatrix}$$

Y $w_N = e^{\frac{2i\pi}{N}}$.

El algoritmo de la Transformada Rápida de Fourier explota la estructura matricial de $\bar{M}$ y el paradigma divide y vencerás, el algoritmo básico es:

1. Dividir el problema en varios subproblemas análogo de menor tamaño.
2. Resolver de forma recursiva cada uno de los subproblemas
3. Obtener la solución del problema mayor integrando las soluciones de cada subproblema.

Para atacar el algoritmo, se empleará el FFT de radio 2 (radix 2) recursivo para el cálculo de la DFT.

Algoritmos básicos

Existen dos algoritmos a emplear el cálculo de la FFT, uno es el de Decimación en Tiempo (DIT FFT), y el otro es el de Decimación en Frecuencia (DIF FFT)

Algoritmo de Decimación en Tiempo de Radio 2

Este algoritmo se basa en dividir la expresión (1) en dos sumas empleando términos pares e impares, como se muestra a continuación

$$\hat{f}(k) = \sum_{n=0}^{\frac{N}{2}-1} f(2n) \overline{w_N}^{k(2n)} + \sum_{n=0}^{\frac{N}{2}-1} f(2n+1) \overline{w_N}^{k(2n+1)} \qquad (4)$$

$$\hat{f}(k) = \sum_{n=0}^{\frac{N}{2}-1} f(2n)(\overline{w}_{\frac{N}{2}})^{kn} + \sum_{n=0}^{\frac{N}{2}-1} f(2n+1)(\overline{w_{\frac{N}{2}}})^{nk}\overline{w}_N{}^k \qquad (5)$$

Teniendo en cuenta que

$$\overline{w}_{\frac{N}{2}} = \left(e^{\frac{-2i\pi}{N}}\right)^2 = \overline{w}_N{}^2$$

Cada una de las sumas de las ecuaciones 4 y 5 pueden ser interpretadas como la FFT de un vector de tamaño $\frac{N}{2}$

Realizando algunos cambios $g(k) = f(2n)$ y $h(n) = f(2n+1)$, para la primera suma de la ecuación 4, se tiene el vector de Fourier $\left(g(0), g(1), \dots g(\frac{N}{2}-1)\right)$ el cual se denotará como $\hat{g}(k)$ para $k = 0,1,2,\dots,\frac{N}{2}-1$. En la segunda suma sin el factor $\overline{w_N}^k$ es la Transformada de Fourier Discreta de $\left(h(0), h(1), \dots, h\left(\frac{N}{2}\right)-1\right)$ que se denotará como $\hat{h}(k)$ para $k = 0,1,2,\dots,\frac{N}{2}-1$.

Con la notación anterior se tiene la siguiente ecuación

$$\hat{f}(k) = \hat{g}(k) + \overline{w_N}^k \hat{h}(k) \qquad (6)$$

Para $k = 0,1,2,\dots,\frac{N}{2}-1$, lo cual permite calcular los $\frac{N}{2}$ primeros términos; ahora para obtener los términos restantes se realizará lo siguiente:

$$\hat{f}\left(k+\frac{N}{2}\right) = \sum_{n=0}^{\frac{N}{2}-1} g(n)\overline{w}_{\frac{N}{2}}{}^{\left(k+\frac{N}{2}\right)n} + \overline{w}_{\frac{N}{2}}{}^{k+\frac{N}{2}} \sum_{n=0}^{\frac{N}{2}-1} h(n)\,\overline{w}_{\frac{N}{2}}{}^{\left(k+\frac{N}{2}\right)n}$$

$$\hat{f}\left(k+\frac{N}{2}\right) = \sum_{n=0}^{\frac{N}{2}-1} g(n)\overline{w}_{\frac{N}{2}}{}^{\left(k+\frac{N}{2}\right)n} - \overline{w}_N{}^k \sum_{n=0}^{\frac{N}{2}-1} h(n)\,\overline{w}_{\frac{N}{2}}{}^{kn}$$

$$\hat{f}\left(k+\frac{N}{2}\right) = \hat{g}(k) - \overline{w}_N{}^k \hat{h}(k) \qquad (7)$$

Para $k = 0,1,2,\dots,\frac{N}{2}-1$

Las ecuaciones 6 y 7 representan un gráfico denominado Operación Mariposa de Cooley-Tukey, cuya representación se muestra en la figura 2.1.3.1

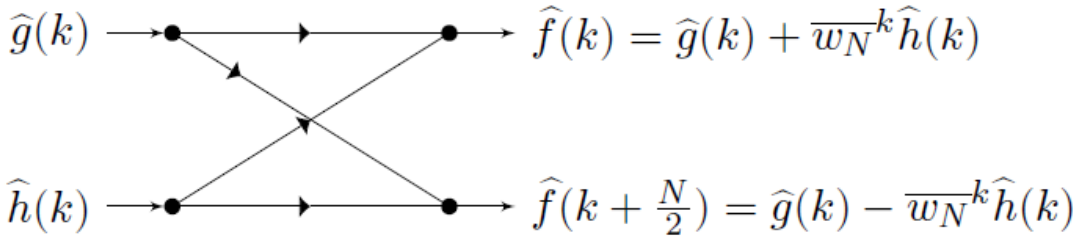

Figura 2.1.3.1 Operación Mariposa de Cooley-Tukey.

La operación mariposa implica que la partición del problema con un vector inicial de tamaño $n = 2^m$, se realizará en *m* etapas, las cuales aparecen al ir descomponiendo el cálculo inicial, en primer lugar con 2 transformadas de tamaño $n = 2^{m-1}$, luego cada subproblema dará lugar a otros 2 de tamaño $n = 2^{m-2}$, con ello se tendrá $4 = 2^2$ subproblemas de tamaño 2^{m-2} y así continuamos.

Implementación del algoritmo DIT FFT de radio 2

El pseudocódigo del cálculo se presenta a continuación.

```
begin
    paresEnGrupo := N / 2       //Comienza con N/2 mariposas en un grupo
    numeroDeGrupos := 1         //Igual factor de giro empleado en un grupo
    Distancia := N / 2
    while numeroDeGrupos < N do
        // Combina pares en cada grupo
        for K := 0 to numeroDeGrupos − 1 do
            jPrimero := 2 ∗ K ∗ paresEnGrupo
            jUltimo := jPrimero + paresEnGrupo − 1
            jGirar := K                 // Acceso consecutivo a w[m]
            W := w[jGirar]              // Asume w[m] = ω_N, m bit-inversos
            for J := jPrimero to jUltimo do
                Temp := W ∗ a[J + Distancia]
                a[J + Distancia] := a[J] − Temp
                a[J] := a[J] + Temp
            end for
        end for
        paresEnGrupo := paresEnGrupo / 2
```

numeroDeGrupos := *numeroDeGrupos* $*$ 2

Distancia:= *Distancia* / 2

end while

end

Algoritmo de Decimación en Frecuencia de Radio 2

Ahora comenzaremos con

$$\hat{f}(r) = \sum_{n=0}^{N-1} f(n)\overline{w_N}^{rn} = \sum_{n=0}^{\frac{N}{2}-1} f(n)\,\overline{w_N}^{rn} + \sum_{n=\frac{N}{2}}^{N-1} f(n)\overline{w_N}^{rn}$$

$$\hat{f}(r) = \sum_{n=0}^{\frac{N}{2}-1} f(n)\,\overline{w_N}^{rn} + \sum_{n=0}^{\frac{N}{2}-1} f\left(n+\frac{N}{2}\right)\overline{w_N}^{r\left(n+\frac{N}{2}\right)}$$

$$\hat{f}(r) = \sum_{n=0}^{\frac{N}{2}-1}\left(f(n) + f\left(n+\frac{N}{2}\right)\cdot \overline{w_N}^{r\frac{N}{2}}\right)\cdot \overline{w_N}^{rn} \qquad (8)$$

Para $r = 0,1,\ldots,N-1$. Si se toma $r = 2k$ en la ecuación 8.

$$\hat{f}(2k) = \sum_{n=0}^{\frac{N}{2}-1}\left(f(n) + f\left(n+\frac{N}{2}\right)\cdot \overline{w_N}^{kN}\right)\cdot \overline{w_N}^{2kn}$$

$$\hat{f}(2k) = \sum_{n=0}^{\frac{N}{2}-1}\left(f(n) + f\left(n+\frac{N}{2}\right)\right)\cdot \overline{w_N}^{2kn}$$

$$\hat{f}(2k) = \sum_{n=0}^{\frac{N}{2}}\left(f(n) + f\left(n+\frac{N}{2}\right)\right)\cdot \overline{w_{\frac{N}{2}}}^{kn}$$

$$\hat{f}(2k) = \sum_{n=0}^{\frac{N}{2}} g(n)\cdot \overline{w_{\frac{N}{2}}}^{kn} = \hat{g}(k) \qquad (9)$$

Para $k = 0,1,\dots,\frac{N}{2}$. Donde se ha definido $g(n) = f(n) + f\left(n+\frac{N}{2}\right)$ y para las igualdades $\overline{w_N}^{kN} = 1$ y $\overline{w_N}^{2kn} = \overline{w_{\frac{N}{2}}}^{2}$.

Ahora para la ecuación 8 aplicamos $r = 2k+1$ y por consecuencia

$$\hat{f}(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} \left(f(n) + f\left(n+\frac{N}{2}\right) \cdot \overline{w_N}^{(2k+1)\frac{N}{2}}\right) \cdot \overline{w_N}^{(2k+1)n}$$

$$\hat{f}(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} \left(f(n) + f\left(n+\frac{N}{2}\right)\right) \cdot \overline{w_N}^{n} \cdot \overline{w_{\frac{N}{2}}}^{kn}$$

$$\hat{f}(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} h(n)\overline{w_{\frac{N}{2}}}^{kn} = \hat{h}(k) \qquad (10)$$

Para $k = 0,1,\dots,\frac{N}{2}-1$. Donde se ha definido $h(n) = \left(f(n) - f\left(n-\frac{N}{2}\right)\right)\overline{w_N}^{n}$ y se ha tenido en cuenta la igualdad $\overline{w_N}^{(2k+1)\frac{N}{2}} = \left(\overline{W_N}^{N}\right)^{k} \cdot \overline{w_N}^{\frac{N}{2}} = 1 \cdot (-1) = -1$

De las ecuaciones 9 y 10 se obtiene la Operación Mariposa de Gentleman-Sande, cuya representación gráfica se muestra en la figura 2.1.3.2.

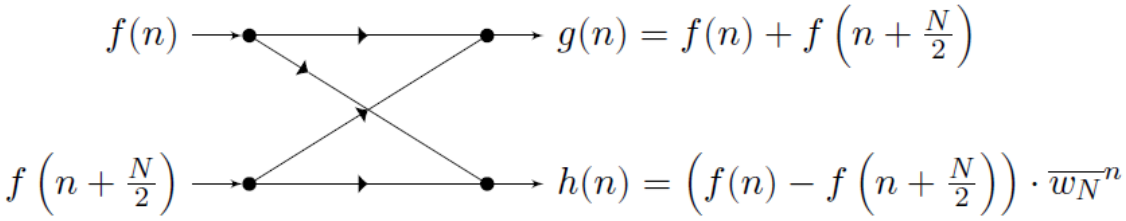

Figura 2.1.3.2 Operación Mariposa de Gentleman-Sande.

Implementación del algoritmo DIF FFT de radio 2

Dados los datos de entrada $f(0), f(1), \dots\dots, f(N-1)$ que están ordenados uno detrás del otro, es decir naturalmente si $f(i)$ y $f(i+1)$ están situados de forma consecutiva en un arreglo (o lista) para $o \leq i \leq N-2$

Ahora en pseudocódigo

Requiere: $w[n] = \overline{w_N}^{2}$ precalculados y almacenados. $N = 2^n$

numeroDeProblemas:=N

while tamanioProblema>1 **do**

```
tamanioMitad := tamanioProblema / 2
for k = 0 to k = numeroDeProblemas – 1 do
        jPrimero := k * tamanioDeProblema
        jUltimo := jPrimero + tamanioMitad – 1
        jMitad := 2 * (N - (N / numeroDeProblemas))
        for j = jPrimero to j = jUltimo do
                W := w[jMitad]
                temp := a[j]
                a[j] := temp + a[j+tamanioMitad]
                a[j+tamanioMitad] := W * (temp – a[j+tamanioMitad])
                jMitad := jMitad + 1
        end for
end for
numeroDeProblemas := 2 * numeroDeProblemas
tamanioProblema := tamanioMitad
end while
return a
```

Este algoritmo se refleja en la clase del código 3.1.3.2 en el método calcularTransformada(). Ahora con un ejemplo concreto para una secuencia de entrada con $N = 8 = 2^3$, esto se refleja en la figura 2.1.3.3

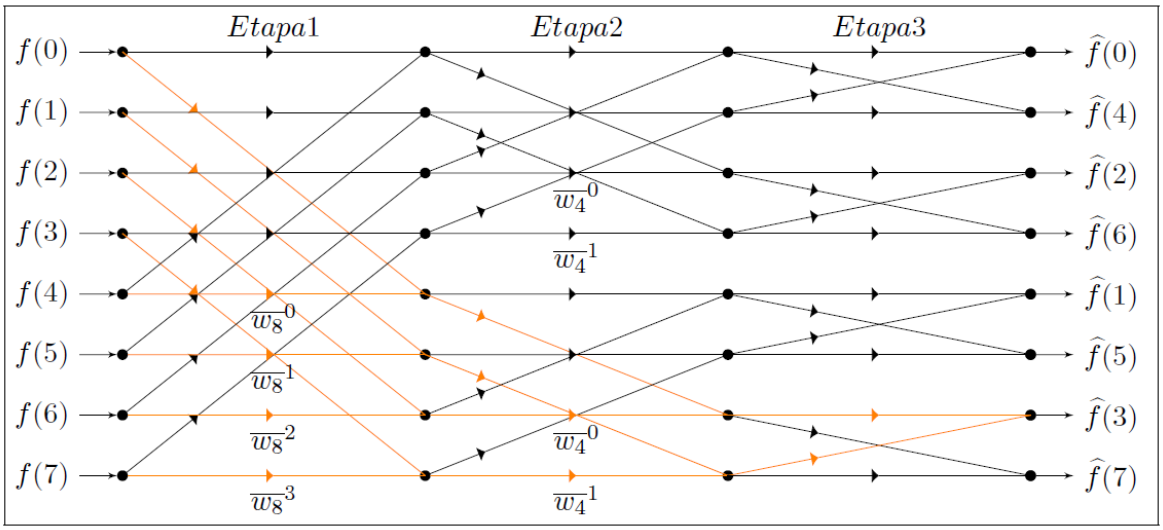

Figura 2.1.3.3 DIF FFT para un $N = 8 = 2^3$

Las raíces w[n] del algoritmo se calculan de la siguiente forma, dado un tamaño N que es potencia de dos (2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc.) y recordando que

$$\overline{w_N}^r = e^{-ir\theta} = \cos(r\theta) - isen(r\theta)$$

Con $\theta = \frac{2\pi}{N}$, donde cada $\overline{w_N}^r$ es una raíz compleja de la unidad, sabiéndolas simétricas e igualmente espaciadas en el círculo unitario. Por lo tanto si un complejo $a + ib$ es una raíz también lo serán $\pm a \pm ib$ y $\pm b \pm ia$.

Teniendo en cuenta esta propiedad, se requiere calcular los $\frac{N}{8} - 1$ valores, es decir, se tiene que calcular $\overline{w_N}^r$ para $r = 0, 1, \ldots, \frac{N}{8} - 1$ teniendo en cuenta que para $r = 0$ se tiene $\overline{w_N}^0 = 1$.

Para calcular $\overline{w_N}^r$ para $r = 0, 1, \ldots, \frac{N}{8} - 1$ habrá que calcular los valores de $\cos(r\theta)$ y $sen(r\theta)$ para $r = 0, 1, \ldots, \frac{N}{8} - 1$, empleando las siguientes funciones trigonométricas:

$$cos(a + b) = cos(a)cos(b) - sen(a)sen(b) \quad (11)$$

$$cos^2(a) + sen^2(a) = 1 \quad (12)$$

$$sen(a + b) = sen(a)cos(b) + sen(b)cos(a) \quad (13)$$

Como se dijo anteriormente $\theta = \frac{2\pi}{N}$, se puede calcular $cos\big((r+1)\theta\big)$ en términos de $\cos(r\theta)$ y $sen(r\theta)$ de la siguiente forma:

$$cos\big((r+1)\theta\big) = cos(r\theta + \theta) = cos(r\theta)cos(\theta) - sen(r\theta)sen(\theta)$$

Como se observa se ha empleado la igualdad de la ecuación 11. Para el algoritmo del cálculo de las raíces se hará lo siguiente $s \coloneqq sen(\theta)$ y empleando a 12 se tiene que $cos(0) = \sqrt{1 - s^2} \coloneqq c$ con *s* y *c* constantes:

$$cos\big((r+1)\theta\big) = cos(r\theta) \cdot c + sen(r\theta) \cdot s$$

Empleando las mismas constantes *s* y *c* se puede calcular $sen\big((r+1)\theta\big)$ de nuevo en términos de $cos(r\theta)$ y $sen(r\theta)$ como sigue

$$\begin{aligned} sen\big((r+1)\theta\big) &= sen(r\theta + \theta) = sen(r\theta)cos(\theta) + cos(r\theta)sen(\theta) \\ &= sen(r\theta) \cdot c + cos(r\theta) \cdot s \end{aligned}$$

Como se puede notar se ha empleado la ecuación 13. Bien para calcular $\overline{w_N}^r$ para $r = \frac{N}{8} - 1, \ldots, \frac{N}{4} - 1$ se empleará

$$cos\left(\frac{2\pi\left(\frac{N}{8}+r\right)}{N}\right) = -sen\left(\frac{2\pi\left(\frac{N}{8}+r\right)}{N}\right)$$

$$sen\left(\frac{2\pi\left(\frac{N}{8}+r\right)}{N}\right) = -cos\left(\frac{2\pi\left(\frac{N}{8}+r\right)}{N}\right)$$

Para $r = 0, 1, \ldots, \frac{N}{8} - 1$. Para calcular $\overline{w_N}^r$ para $r = \frac{N}{4} - 1, \ldots, N - 1$ se empleará

$$cos\left(\frac{2\pi\left(\frac{N}{4}+r\right)}{N}\right) = -sen\left(\frac{2\pi\left(\frac{N}{4}-r\right)}{N}\right)$$

$$sen\left(\frac{2\pi\left(\frac{N}{4}+r\right)}{N}\right) = -cos\left(\frac{2\pi\left(\frac{N}{4}-r\right)}{N}\right)$$

Para $r = 1, \ldots, \frac{N}{4} - 1$. Para finiquitar se calculan todas las raíces que se requieran, empleando las calculadas y repitiendo las necesarias para recorrer el algoritmo de la FFT de principio a fin. Este cálculo se muestra en la clase del código 3.1.3.2 en el método calcularRaices().

Código 3.1.3.2 Calculo de la FFT por Decimación en Frecuencia.

```
/**
 * Clase que realiza el claculo iterativo de la FFT por decimacion o diezmo de
 * frecuencia<br>
 *  Requiere: w[n]=(w_N )¯^2 precalculados y almacenados. N=2^n<br>
 *  numeroDeProblemas:=1<br>
 *  tamanioDeProblema:=N<br>
 *  while tamanioDeProblema>1 do<br>
 *     tamanioMitad := tamanioProblema / 2<br>
 *     for k = 0 to k = numeroDeProblemas – 1 do<br>
 *        jPrimero := k * tamanioDeProblema<br>
 *        jUltimo := jPrimero + tamanioMitad – 1<br>
 *        jGirar := 0<br>
 *        for j = jPrimero to j = jUltimo do<br>
 *           W := w[jGirar]<br>
 *           temp := a[j]<br>
 *           a[j] := temp + a[j+tamanioMitad]<br>
 *           a[j+tamanioMitad] := W * (temp – a[j+tamanioMitad])<br>
 *           jMitad := jGirar + numeroDeProblemas<br>
 *           end for<br>
 *        end for<br>
 *     numeroDeProblemas := 2 * numeroDeProblemas<br>
```

```
 *      tamanioProblema := tamanioMitad<br>
 *      end while<br>
 *  return a<br>
 * @author sdelaot
 */
public class TransformadaDIFFFT {
   /**
    * Para realizar la arirmetica compleja en los calculos de la FFT
    */
   AritmeticaCompleja aritmetica;
   /**
    * Constructor de la clase
    */
   public TransformadaDIFFFT() {
      aritmetica = new AritmeticaCompleja();
   }
   /**
    * Muestra los complejos funciona como DEBUG por si se quiere ver en algun
    * momento de la ejecucion resultados intermedios
    * @param zetas imprime esta variable
    */
   private void mostrarArreglo( Complejo [] zetas ) {
      for( int j=0; j<zetas.length; j++  ) {
         System.out.println( j + " " + zetas[j] );
         }
      System.out.println();
   }
   /**
    * Calcula la FFT por decimacion o diezmo en frecuencia
    * @param N tamanio de las muestras de entrada
    * @param a arreglo de tamanio N Complejos
    * @param w contiene las raices de la unidad tambien complejos
    * @return devuelve la transformada discreta de los valores que antes
    * contenia "a" (Complejos)
    */
   public Complejo [] calcularTransformada( int N, Complejo [] a,
         Complejo [] w ) {
      int numeroDeProblemas = 1;
      int tamanioProblema = N;
      int jGirar = 0;
      while( tamanioProblema>1 ) {
         int tamanioMitad = tamanioProblema / 2;
         for( int k=0; k<numeroDeProblemas; k++ ) {
            int jPrimero = k * tamanioProblema;
            int jUltimo = jPrimero + tamanioMitad - 1;
            // recorre las raices de la unidad sin dar saltos
            // las veces que sea necesario en el algoritmo
```

```
            jGirar = 0;//2 * ( N - ( N / numeroDeProblemas ) );
            for( int j=jPrimero; j<=jUltimo; j++ ) {
               Complejo W = new Complejo( w[jGirar] );
               Complejo temp = new Complejo( a[j] );
               // Se realiza la mariposa de la primera operacion
               a[j] = aritmetica.sumar( temp, a[j+tamanioMitad] );
               // segunda operacion
               a[j+tamanioMitad] =
                     aritmetica.multiplicar( W,
                           aritmetica.restar( temp,
                                 a[j+tamanioMitad] ) );
               // recorremos el arreglo de las raices
               jGirar = jGirar + numeroDeProblemas;
               }
            }
         numeroDeProblemas *= 2;// * numeroDeProblemas;
         tamanioProblema = tamanioMitad;
         }
      return invertirBits( a );
   }
   /**
    * La funcion calcula las raices de la unidad necesarias para aplicar en un
    * algoritmo de FFT, es decir, los w__{N}^{r} conjugados
    * @param N tamanio de la FFT
    * @return devuelve w que contiene las raices complejas ordenadas para
    * emplearlas cuando se requieran en el algoritmo FFT
    */
   public Complejo [] calcularRaices( int N ) {
      double theta = 2.0*Math.PI / (double)N;
      double [] wCos = new double[N];
      double [] wSen = new double[N];
      double s = Math.sin(theta);
      double c = 1.0 - 2.0 * Math.pow( Math.sin(theta/2.0), 2.0 );
      //double c = Math.sqrt( 1.0 - s*s );
      wCos[0] = 1.0;
      wSen[0] = 0.0;
      //System.out.println( "0 0 " + wCos[0] + " " + wSen[0] );
      for( int k=0; k<N/8; k++ ) {
         wCos[k+1] = c * wCos[k] - s*wSen[k];
         wSen[k+1] = s * wCos[k] + c*wSen[k];
         //System.out.println( (k+1) + " " + (k+1) + " " + wCos[k+1] + " " + wSen[k+1] );
         }
      //System.out.println();
      int L = N / 8;
      wCos[L] = Math.sqrt(2.0) / 2.0;
      wSen[L] = Math.sqrt(2.0) / 2.0;
      for( int k=1; k<=N/8; k++ ) {
```

```
        wCos[L+k] = wSen[L-k];
        wSen[L+k] =  wCos[L-k];
        //System.out.println( (k+L) + " " + (k+L) + " " + wCos[k+L] + " " + wSen[L+k]);
        }
    //System.out.println();
    L = N / 4;
    wCos[L] = 0.0;
    wSen[L] = 1.0;
    for( int k=1; k<N/4; k++ ) {
        wCos[L+k] = -wCos[L-k];
        wSen[L+k] =  wSen[L-k];
        //System.out.println( (k+L) + " " + (k+L) + " " + wCos[k+L] + " " + wSen[L+k]);
        }
    //System.out.println();
    Complejo [] w = new Complejo[N];
    //System.out.println( "PRIMER FOR" );
    for( int n=0; n<N/2; n++ ) {
        double real = wCos[n];
        double imag = wSen[n];
        if( imag!=0.0 ) {
            imag *= (-1.0);
            }
        w[n] = new Complejo( real, imag );
        //System.out.println( n + " " + w[n] );
        }
    //System.out.println( "SEGUNDO FOR" );
    for( int n=N/2; n<N; n++ ) {
        //System.out.println( wCos[n-N/2] + " " + wSen[n-N/2] );
        double real = wCos[n-N/2];
        double imag = wSen[n-N/2];
        if( real!=0.0 ) {
            real *= (-1.0);
            }
        //System.out.println( real + " " + imag );
        w[n] = new Complejo( real, imag );
        //System.out.println( n + " " + w[n] );
        }
    return w;
}
/**
 * Ordena los elementos de la FFT
 *
 * @param a arreglo complejo desordenado
 * @return devuelve el arreglo complejo ordenado
 */
private Complejo [] invertirBits( Complejo [] a ) {
    int N = a.length;
```

```
        Complejo [] fftTemp = new Complejo[N];
        for(int n=0; n<N; n++ ) {
           fftTemp[n] = new Complejo( a[n] );
           }
        int [] bitInverso = new int[N];
        int p = 1;
        for( int q=0; q<N; q++ ) {
           bitInverso[q] = q;
           }
        while( p<N ) {
           for( int q=0; q<p; q++ ) {
              bitInverso[q] = bitInverso[q]*2;
              bitInverso[q+p] = bitInverso[q]+1;
              }
           p = p*2;
           }
        for( int n=0; n<N ;n++ )
           a[bitInverso[n]]=fftTemp[n]; //fft_out FFT ordenada.
        return a;
     }
}
/**
 * Clase para probar la FFT DIF
 * @author sdelaot
 */
public class CalculadoraDIFFFT {
   public static void main(String[] args) {
      TransformadaDIFFFT transformada = new TransformadaDIFFFT();
      int N = 16;
      Complejo [] f = { // Pulso cuadrado
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 1.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 ),
         new Complejo( 0.0, 0.0 )
         };
```

```
Complejo [] fs = { // senoidal cos(nwT), w=2*PI*f
   new Complejo( Math.cos( 0.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 1.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 2.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 3.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 4.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 5.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 6.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 7.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 8.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos( 9.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos(10.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos(11.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos(12.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos(13.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos(14.0*2.0*Math.PI/N), 0.0 ),
   new Complejo( Math.cos(15.0*2.0*Math.PI/N), 0.0 )
   };

Complejo [] fi = { // Inpulso delta
   new Complejo( 1.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 ),
   new Complejo( 0.0, 0.0 )
   };
Complejo [] fr = { // rampa
   new Complejo( 0.125, 0.0 ),
   new Complejo( 0.250, 0.0 ),
   new Complejo( 0.375, 0.0 ),
   new Complejo( 0.500, 0.0 ),
   new Complejo( 0.625, 0.0 ),
   new Complejo( 0.750, 0.0 ),
   new Complejo( 0.875, 0.0 ),
   new Complejo( 1.000, 0.0 ),
   new Complejo( 0.125, 0.0 ),
```

```
    new Complejo( 0.250, 0.0 ),
    new Complejo( 0.375, 0.0 ),
    new Complejo( 0.500, 0.0 ),
    new Complejo( 0.625, 0.0 ),
    new Complejo( 0.750, 0.0 ),
    new Complejo( 0.875, 0.0 ),
    new Complejo( 1.000, 0.0 )
    };
System.out.println( "\nPulso cuadrado" );
for( Complejo z: f ) {
    System.out.print( " " + z );
    }
System.out.println( "\nFFT Pulso cuadrado" );
Complejo [] dft =
            transformada.calcularTransformada(N,f, transformada.calcularRaices(N));
for( Complejo z: dft ) {
    System.out.print( " " + z );
    }
System.out.println( "\nFFT Magnitud pulso" );
for( Complejo z: dft ) {
    System.out.print( " " + z.calcularMagnitud() );
    }
System.out.println();
System.out.println( "\nFuncion senoidal" );
for( Complejo z: fs ) {
    System.out.print( " " + z );
    }
System.out.println( "\nFFT Pulso cuadrado" );
dft = transformada.calcularTransformada(N,fs, transformada.calcularRaices(N));
for( Complejo z: dft ) {
    System.out.print( " " + z );
    }
System.out.println( "\nFFT Magnitud Senoidal" );
for( Complejo z: dft ) {
    System.out.print( " " + z.calcularMagnitud() );
    }
System.out.println();
System.out.println( "\nImpulso delta" );
for( Complejo z: fi ) {
    System.out.print( " " + z );
    }
System.out.println( "\nFFT Impulso delta" );
dft = transformada.calcularTransformada(N,fi, transformada.calcularRaices(N));
for( Complejo z: dft ) {
    System.out.print( " " + z );
    }
System.out.println( "\nFFT Magnitud delta" );
```

```
      for( Complejo z: dft ) {
         System.out.print( " " + z.calcularMagnitud() );
         }
      System.out.println();
      System.out.println( "\nRampa" );
      for( Complejo z: fr ) {
         System.out.print( " " + z );
         }
      System.out.println( "\nFFT Rampa" );
      dft = transformada.calcularTransformada(N,fr, transformada.calcularRaices(N));
      for( Complejo z: dft ) {
         System.out.print( " " + z );
         }
      System.out.println( "\nFFT Magnitud Rampa" );
      for( Complejo z: dft ) {
         System.out.print( " " + z.calcularMagnitud() );
         }
      System.out.println();
   }
}
```

TAREA

1. Copiar las clases en un proyecto que se llame fft de netbeans, ya sea en Java o en C++ y comparar los resultados que devuelven las clases, defina y escriba sus conclusiones.
2. Cree los componentes gráficos (GUI) para ver las señales de entrada y la salida de los cálculos de la DFT y de la DIFFFT.
3. Como habrá notado no hay código para el cálculo de la Transformada Rápida de Fourier para la Decimación en Tiempo DITFFT, créelo como clase e inclúyalo sus conclusiones en el punto 1 y adiciónelo al punto 2.
4. El algoritmo de Divide y vencerás necesita que los algoritmos sean recursivos y los algoritmos DIF FFT y DIT FFT son iterativos, así que les toca definirlos, analizarlos, programarlos y compararlos contra los iterativos definiendo conclusiones en su reporte.
5. **Este punto da sobre la calificación final de la Unidad de Aprendizaje dos puntos**. Lo que hay que hacer es grabar su voz en un archivo wav, mp3 o algún otro (no se admite texto con formato) medio minuto al menos, graficarla y obtener su Transformada de Fourier Rápida por ambos algoritmos DIF y DIT. Darle al usuario de calcularlo iterativamente o recursivamente. Reportar conclusiones.