11

Métodos rápidos para el cálculo de la TDF

11.1. Introducción

La transformada discreta de Fourier (TDF) tiene un papel muy importante en el diseño, análisis y realización de sistemas y algoritmos de tratamiento de señales en tiempo discreto[1]. Las propiedades básicas de la transformada de Fourier y de la TDF, presentadas en los Capítulos 3 y 4 respectivamente, hacen que analizar y diseñar sistemas en el dominio transformado sea conveniente y práctico. Este hecho, junto con la existencia de algoritmos eficientes para el cálculo explícito de la TDF, la convierten en un componente importante de muchas aplicaciones prácticas de los sistemas en tiempo discreto.

Como se comentó en el Capítulo 4, la TDF $X[k]$ de N puntos son las muestras de la transformada de Fourier $X(e^{j\omega})$ en N frecuencias equiespaciadas $\omega_k = 2\pi k/N$ es decir, en N puntos de la circunferencia unidad del plano complejo. En este capítulo se presentan varios métodos muy eficientes para calcular la TDF. Estos algoritmos se denominan colectivamente *transformada rápida de Fourier* (FFT, "Fast Fourier Transform"), y los dos más conocidos se estudian en las Secciones 11.3 y 11.4. Estos algoritmos son aplicables cuando el orden N de la TDF es potencia entera de 2, $N = 2^{\alpha}$. Si éste no es el caso, existen algoritmos un poco más complicados, algunos de los cuales se presentan en la Sección 11.6, tales como el método de las raíces variadas o el de factores primos.

Hay muchas formas de medir la complejidad y la eficiencia de un determinado algoritmo o realización, y el resultado final depende tanto de la tecnología disponible como de la aplicación. La medida de la complejidad computacional que se utilizará es el número de multiplicaciones y sumas aritméticas. Esta medida es simple de aplicar, y el número de multiplicaciones y sumas está directamente relacionado con la velocidad de cómputo cuando los algoritmos se ejecutan en computadores digitales de propósito general o en microprocesadores dedicados. Sin embargo, en realizaciones VLSI existen otras medidas que pueden ser más apropiadas, por ejemplo, el área del chip y los requisitos de potencia, que pueden no estar relacionadas directamente con el número de operaciones aritméticas.

[1] Este capítulo está basado mayormente en el Capítulo 9 "Computation of the Discrete Fourier Transform", de la 2da eidición del libro de Oppenheim, junto con algunos agregados de la 1ra. edición y de los libros de Ingle y Proakis y de Porat.

En términos de multiplicaciones y sumas, los algoritmos FFT puede ser varios órdenes de magnitud más eficientes que otros algoritmos alternativos, a tal punto que en muchos casos el procedimiento más conveniente para realizar una convolución es calcular la transformada de las sucesiones que se van a convolucionar, multiplicar las transformadas y calcular luego la transformada inversa del producto de dichas transformadas. Esta forma de implementación se denomina *convolución de alta velocidad*. Los detalles de esta técnica se presentaron en el Capítulo 4 (convolución por bloques: métodos *overlap-add* y *overlap-save*). En la Sección 11.7 se compara el tiempo de cálculo y el número de operaciones necesarios para calcular una TDF de N puntos utilizando el comando `fft` de MATLAB. Este comando elige el tipo de algoritmo a utilizar según el valor de N, por lo que su registro en función del largo de la sucesión permite inferir detalles de la implementación. También se comparan los tiempos de cálculo que insume una convolución tradicional, y una de alta velocidad, y su variación en función del largo de las sucesiones a convolucionar.

En aparente contradicción, existe una serie de algoritmos para calcular un conjunto más general de muestras de la TDF, que resultan de reescribirla como una convolución. Son útiles cuando se requieren sólo *algunos* valores de la TDF en un del intervalo de frecuencias comprendido en $0 \leq \omega < 2\pi$, en lugar de *todos* los N valores que se obtienen con la FFT. Como ejemplo se presentan los siguientes algoritmos:

- de Goertzel (Sección 11.8.1);
- transformada zoom (Sección 11.8.2);
- transformada chirp (Sección 11.8.3).

La convolución requerida por estos algoritmos se puede efectuar aprovechando los procedimientos eficientes descritos más arriba.

En algunas implementaciones, las multiplicaciones requieren mucho más tiempo de cómputo que las sumas. Para este tipo de sistemas el algoritmo de Winograd (Sección 11.8.5), basado en avanzados conceptos de teoría de números, como el Teorema Chino del Resto, permite obtener la mayor eficiencia llevando el número de multiplicaciones al mínimo posible.

Antes de presentar en detalle cada uno de estos algoritmos, se reseña brevemente la historia del desarrollo de los métodos rápidos para el cálculo de la TDF.

11.2. Cómputo eficiente de la TDF

En el Capítulo 4 se definió la TDF de una sucesión finita de longitud N como

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk}, \quad k = 0,\, 1,\, \ldots,\, N-1. \tag{11.1}$$

La expresión de la TDF inversa es

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}nk}, \quad n = 0,\, 1,\, \ldots,\, N-1. \tag{11.2}$$

En (11.1) y (11.2), tanto $x[n]$ como $X[k]$ pueden ser complejos. Como las expresiones del miembro derecho de esas ecuaciones sólo difieren en el signo del exponente de $e^{j\frac{2\pi}{N}nk}$ y en el factor de escala $1/N$, los procedimientos computacionales necesarios para calcular (11.1) son enteramente similares a los necesarios para evaluar (11.2).

El cómputo de cada valor $X[k]$ de la TDF evaluando directamente la ecuación (11.1) requiere N multiplicaciones complejas y $(N-1)$ sumas complejas[2]. Para obtener los N valores se necesitan en total N^2 multiplicaciones complejas y $N(N-1)$ sumas complejas. Como casi ningún procesador efectúa operaciones con números complejos, conviene expresar (11.1) de manera de destacar las operaciones con números reales,

$$X[k] = \sum_{n=0}^{N-1} \Big[\Big(\mathrm{Re}\{x[n]\}\,\mathrm{Re}\{e^{-j\frac{2\pi}{N}nk}\} - \mathrm{Im}\{x[n]\}\,\mathrm{Im}\{e^{-j\frac{2\pi}{N}nk}\} \Big) + j\,\Big(\mathrm{Re}\{x[n]\}\,\mathrm{Im}\{e^{-j\frac{2\pi}{N}nk}\} + \mathrm{Im}\{x[n]\}\,\mathrm{Re}\{e^{-j\frac{2\pi}{N}nk}\} \Big) \Big], \quad (11.3)$$

con $k = 0, 1, \ldots, N-1$, que demuestra que cada multiplicación compleja requiere cuatro multiplicaciones reales y dos sumas reales, y cada suma compleja requiere dos sumas reales. Por tanto, para cada valor de k, el cálculo directo de $X[k]$ necesita $4N$ multiplicaciones reales y $(4N-2)$ sumas reales[3]. Como $X[k]$ se debe calcular para N valores diferentes de k, el cálculo directo de la TDF de una sucesión $x[n]$ requiere $4N^2$ multiplicaciones reales y $N(4N-2)$ sumas reales. Además de las multiplicaciones y las sumas que indica (11.3), para el cálculo de la TDF utilizando una computadora digital de propósito general o hardware específico se necesita disponer de memoria suficiente para almacenar y acceder a los N valores complejos de la sucesión de entrada $x[n]$ y los N^2 valores de los coeficientes complejos $e^{-j\frac{2\pi}{N}nk}$. Como el número de cálculos y, en consecuencia, el tiempo de cómputo es aproximadamente proporcional a N^2, el número de operaciones aritméticas necesarias para calcular la TDF por el método directo es muy elevado para valores grandes de N. Por este motivo interesa estudiar procedimientos de cómputo que reduzcan el número de multiplicaciones y sumas.

La mayoría de los procedimientos para mejorar la eficiencia en los cálculos de la TDF explotan las propiedades de periodicidad y de simetría de $e^{-j\frac{2\pi}{N}nk}$. Concretamente,

1. $e^{-j\frac{2\pi}{N}k(N-n)} = e^{j\frac{2\pi}{N}nk} = (e^{-j\frac{2\pi}{N}nk})^*$ (simetría compleja conjugada);
2. $e^{-j\frac{2\pi}{N}kn} = e^{-j\frac{2\pi}{N}k(n+N)} = e^{-j\frac{2\pi}{N}(k+N)n}$ (periodicidad en n y en k).

Por ejemplo, utilizando la primera propiedad, es decir, la simetría de las funciones seno y coseno implícitas en la exponencial compleja, se pueden agrupar los términos correspondientes a n y $(N-n)$ en (11.3)

$$\mathrm{Re}\{x[n]\}\,\mathrm{Re}\{e^{-j\frac{2\pi}{N}nk}\} + \mathrm{Re}\,\{x[N-n]\}\,\mathrm{Re}\{e^{-j\frac{2\pi}{N}(N-n)k}\} = (\mathrm{Re}\{x[n]\} + \mathrm{Re}\{x[N-n]\})\,\mathrm{Re}\{e^{-j\frac{2\pi}{N}nk}\},$$

[2] Para mayor generalidad, conviene pensar que $x[n]$ puede ser compleja.

[3] En este apunte, el valor del número de operaciones es sólo aproximado, ya que, por ejemplo, la multiplicación por 1 no requiere realmente una multiplicación. No obstante, la estimación de la complejidad computacional que se obtiene incluyendo esas multiplicaciones es lo suficientemente exacta como para permitir la comparación entre diferentes clases de algoritmos.

$$- \operatorname{Im}\{x[n]\} \operatorname{Im}\{e^{-j\frac{2\pi}{N}nk}\} - \operatorname{Im}\{x[N-n]\} \operatorname{Im}\{e^{-j\frac{2\pi}{N}[N-n]k}\}$$
$$= -\left(\operatorname{Im}\{x[n]\} - \operatorname{Im}\{x[N-n]\}\right) \operatorname{Im}\{e^{-j\frac{2\pi}{N}nk}\}.$$

Los otros términos de (11.3) se pueden agrupar de manera similar. De esta forma, el número de multiplicaciones se puede reducir aproximadamente a la mitad. Además, se puede obviar la multiplicación para ciertos valores del producto kn donde las funciones seno y coseno toman los valores 1 o 0. Sin embargo, las simplificaciones de este tipo sólo cambian la constante de proporcionalidad entre el número de operaciones y N^2. La segunda propiedad, la periodicidad de la sucesión compleja $e^{-j\frac{2\pi}{N}kn}$, se puede explotar para conseguir una reducción más significativa del número de operaciones.

Los algoritmos que aprovechan la periodicidad y la simetría de la sucesión $e^{-j\frac{2\pi}{N}kn}$ se conocen desde mucho antes de la era de las computadoras digitales de alta velocidad. En ese momento, cualquier esquema que redujera los cálculos (manuales!) incluso en un factor de 2 era bienvenido. Heideman *et al.* (1984) ha rastreado los orígenes de la FFT hasta Gauss en 1805. Runge (1905) y después Danielson y Lanczos (1942) describieron algoritmos cuya carga computacional era aproximadamente proporcional a $N \log N$ en lugar de a N^2. Sin embargo, la diferencia no era de mucha importancia cuando los cálculos se realizaban a mano, para valores pequeños de N. La posibilidad de un ahorro computacional importante se ignoró de forma general hasta 1965, cuando Cooley y Tukey (1965) publicaron un algoritmo para al cálculo de la transformada discreta de Fourier aplicable cuando N es un número compuesto, es decir, producto de dos o más enteros. La publicación de su artículo fue el punto de partida de una gran actividad para la aplicación de la TDF al tratamiento de señales y produjo el descubrimiento de varios algoritmos computacionales altamente eficientes. Estos algoritmos se conocen colectivamente como *transformada rápida de Fourier*, o FFT[4].

Los algoritmos de FFT se basan en descomponer el cálculo de la TDF de una sucesión de longitud N en varias TDF más pequeñas. La forma en que se aplica este principio conduce a una variedad de algoritmos semejantes respecto a la mejora de la velocidad de cómputo. En este Capítulo la presentación no es exhaustiva, sino que se ilustran los principios comunes, considerando en detalle sólo los esquemas más populares. Para el caso en que N es potencia entera de 2 se consideran dos tipos de algoritmos de FFT. El primero, denominado de *decimación en tiempo* (DET), se presenta en la Sección 11.3, y se basa en descomponer la sucesión $x[n]$ (que generalmente se considera una sucesión temporal) en subsecuencias más pequeñas. En el segundo tipo de algoritmos, presentado en la Sección 11.4, la sucesión de los coeficientes $X[k]$ de la TDF se descompone en subsecuencias más cortas; de ahí su nombre de *decimación en frecuencia* (DEF). El número de operaciones necesario en ambos casos es proporcional a $N \log_2 N$

En la Sección 11.6 se estudian algunas extensiones más generales, en que N no necesariamente es potencia de 2, pero sí altamente compuesto.

En las secciones que siguen se presentan otros algoritmos para calcular la TDF con distinto grado de eficiencia, pero siempre con menos multiplicaciones y sumas que la evaluación directa de (11.3). Por ejemplo, en la Sección 11.8 se estudian algoritmos que se basan en formular el cálculo de la TDF como una convolución. En la Sección 11.8.1 se presenta el algoritmo de Goertzel (Goertzel, 1958), cuyo costo computacional sigue siendo proporcional a N^2 pero con una constante de proporcionalidad menor que la del método directo.

[4]En Cooley *et al.* (1967) se incluye un resumen histórico de los resultados relacionados con la FFT.

Una de sus principales ventajas es que permite calcular el valor de la transformada de Fourier $X(e^{j\omega})$ de una sucesión de longitud finita para *cualquier* valor de ω, y no necesariamente para $\omega = \omega_k = 2\pi k/N$; es decir, no está limitado al cálculo de la TDF. En las Secciones 11.8.2 y 11.8.3 se estudian algoritmos que permiten calcular muestras de $X(e^{j\omega})$ en un rango acotado de valores de frecuencia, y no necesariamente en todo el intervalo $[0,\, 2\pi)$.

11.3. Algoritmos de decimación en tiempo

El cálculo de la TDF se puede hacer mucho más eficiente computando mayor cantidad de TDF de menor longitud (Duhamel y Vetterli, 1990). Este proceso explota tanto la simetría como la periodicidad de $e^{-j\frac{2\pi}{N}kn}$. Los algoritmos en los que la señal temporal $x[n]$ se descompone sucesivamente en sucesiones más pequeñas se denominan algoritmos de *decimación en el tiempo* (DET).

Para ilustrar el principio de este algoritmo es conveniente considerar el caso especial de que N es una potencia entera de 2, es decir, $N = 2^\alpha$. Como N es un número entero par, se puede considerar el cálculo de $X[k]$ separando $x[n]$ en dos sucesiones de $(N/2)$ puntos[5] formadas por los puntos pares de $x[n]$ y los puntos impares de $x[n]$. Si $X[k]$ está dada por

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk}, \quad k = 0,\, 1,\, \ldots,\, N-1, \tag{11.4}$$

dividiendo $x[n]$ en las muestras pares e impares se obtiene

$$X[k] = \sum_{n \text{ par}} x[n]e^{-j\frac{2\pi}{N}nk} + \sum_{n \text{ impar}} x[n]e^{-j\frac{2\pi}{N}nk} \tag{11.5}$$

y sustituyendo las variables $n = 2r$ para n par y $n = 2r+1$ para n impar,

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r]e^{-j\frac{2\pi}{N}k2r} + \sum_{r=0}^{(N/2)-1} x[2r+1]e^{-j\frac{2\pi}{N}k(2r+1)} \\ &= \sum_{r=0}^{(N/2)-1} x[2r]e^{-j2\frac{2\pi}{N}kr} + e^{-j\frac{2\pi}{N}k} \sum_{r=0}^{(N/2)-1} x[2r+1]e^{-j2\frac{2\pi}{N}kr}. \end{aligned} \tag{11.6}$$

Teniendo en cuenta que $e^{-j2\frac{2\pi}{N}kr} = e^{-j\frac{2\pi}{N/2}kr}$, la ecuación (11.6) puede escribirse como

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r]e^{-j\frac{2\pi}{N/2}kr} + e^{-j\frac{2\pi}{N}k} \sum_{r=0}^{(N/2)-1} x[2r+1]e^{-j\frac{2\pi}{N/2}kr} \\ &= G[k] + e^{-j\frac{2\pi}{N}k}H[k], \quad k = 0,\, 1,\, \ldots,\, N-1. \end{aligned} \tag{11.7}$$

En cada una de las sumas de (11.7) se reconoce una TDF de $N/2$ puntos. La primera suma es la TDF de $(N/2)$ puntos de las muestras pares de la sucesión original, y la segunda suma

[5] Las palabras *muestra* y *punto* se usan indistintamente para significar "valor de la sucesión". Además, una sucesión de longitud N se dirá que es una "sucesión de N puntos", y la TDF de una sucesión de longitud N se denominará TDF de N puntos, o también una TDF de *orden N*.

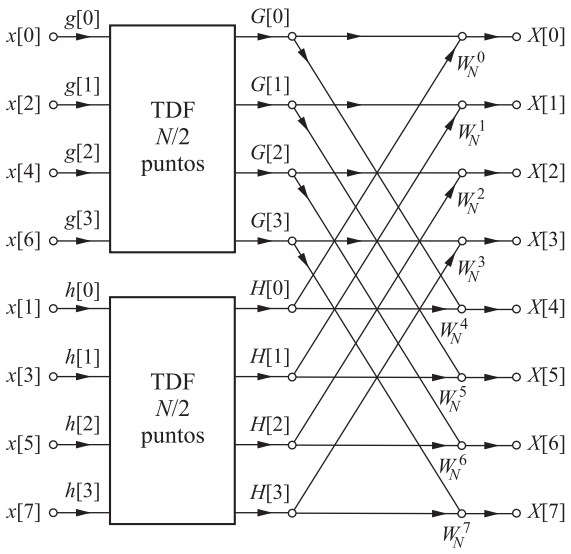


Fig. 11.1. DET: descomposición de una TDF de N puntos en dos TDF de $N/2$ puntos ($N = 8$).

es la TDF de $(N/2)$ puntos de las muestras impares. Aunque el índice k toma los valores $k = 0, 1, \ldots, N-1$, cada una de las sumas se calcula sólo para los valores de k entre 0 y $N/2-1$, ya que tanto $G[k]$ como $H[k]$ son periódicas de período $N/2$. Después de calcular las dos TDF, se combinan como indica (11.7) para obtener la TDF de N puntos $X[k]$. La Fig. 11.1 muestra este cálculo para $N = 8$. En esta figura las ramas que entran a un nodo se suman para producir la variable del nodo. Cuando no se indica coeficiente, se supone que el arco tiene una ganancia unidad; en otros arcos, la ganancia es una potencia entera de $e^{-j\frac{2\pi}{N}}$. Este coeficiente aparece tan a menudo en estas operaciones que es frecuente indicarlo como

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

de manera que (11.7) se escribe

$$X[k] = G[k] + W_N^k H[k].$$

En la Fig. 11.1 muestra el cálculo de una TDF de $N = 8$ puntos a partir de dos TDF de 4 puntos. $G[k]$ indica la TDF de 4 puntos de las muestras pares y $H[k]$ la TDF de 4 puntos de las muestras impares. $X[0]$ se obtiene multiplicando $H[0]$ por $e^{j\frac{2\pi}{N}0} = W_N^0 = 1$ y sumando el resultado a $G[0]$. Para obtener $X[1]$ se multiplica $H[1]$ por $e^{j\frac{2\pi}{N}1} = W_N$ y se suma el resultado a $G[1]$, etc. La ecuación (11.7) indica que para calcular $X[4]$ hay que multiplicar $H[4]$ por $e^{j\frac{2\pi}{N}4} = W_N^4$ y sumar el resultado a $G[4]$. Sin embargo, como $G[k]$ y $H[k]$ son periódicas en k con período 4, $G[4] = G[0]$ y $H[4] = H[0]$. Por tanto, $X[4]$ se obtiene multiplicando $H[0]$ por $e^{j\frac{2\pi}{N}4} = W_N^4 = 1$ y sumando el resultado a $G[0]$. Como se observa en la Fig. 11.1, los valores de $X[5]$, $X[6]$, y $X[7]$ se obtienen de forma similar.

Es interesante comparar el número de multiplicaciones y sumas necesarias para calcular la TDF en la forma sugerida por (11.7) con el que demanda el cálculo directo de la TDF con la ecuación (11.4). En este último caso, donde no se explota la simetría del factor $e^{j\frac{2\pi}{N}kn} = W_N^{kn}$, se necesitan aproximadamente N^2 multiplicaciones y sumas complejas[6].

[6]Por simplicidad se supondrá que N es grande, por lo que $(N-1)$ es aproximadamente igual a N.

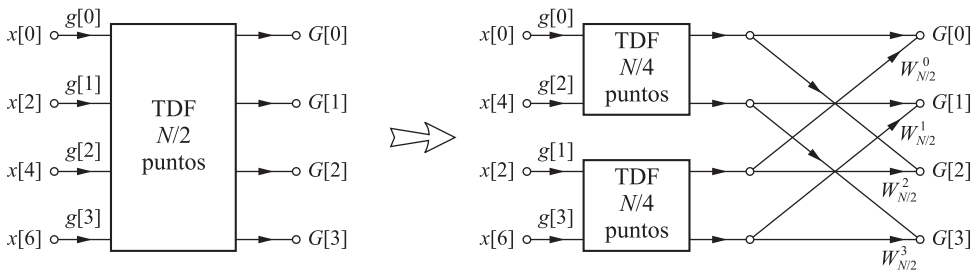


Fig. 11.2. DET: descomposición de una TDF de $N/2$ puntos en dos TDF de $N/4$ puntos ($N=8$).

Si cada una de las transformadas de $N/2$ puntos de (11.7) se efectúa por cálculo directo, se deben efectuar $2(N/2)^2$ multiplicaciones complejas y aproximadamente $2(N/2)^2$ sumas complejas. Además, para combinar las dos TDF se requieren N productos complejos para multiplicar la segunda TDF de $N/2$ puntos por $e^{j\frac{2\pi}{N}k} = W_N^k$ y N sumas complejas para sumar este resultado con la primera TDF de $N/2$ puntos. Por tanto, el cálculo de (11.7) para todos los valores de k requiere un máximo de $N + 2(N/2)^2$ o $N + N^2/2$ multiplicaciones y sumas complejas. Es fácil verificar que para $N > 2$, $N + N^2/2$ será menor que N^2.

En la ecuación (11.7) la TDF original de N puntos se calcula a partir de dos TDF de $(N/2)$ puntos. Si $N/2$ es par, lo que ocurre cuando N es potencia de 2, se puede volver a aplicar la idea de la ecuación (11.7) para calcular cada TDF de $(N/2)$ puntos en función de dos TDF de $(N/4)$ puntos. Por tanto, $G[k]$ en (11.7) se puede escribir como

$$G[k] = \sum_{r=0}^{(N/2)-1} g[r]e^{-j\frac{2\pi}{N/2}kr} = \sum_{\ell=0}^{(N/4)-1} g[2\ell]e^{-j\frac{2\pi}{N/2}k2\ell} + \sum_{\ell=0}^{(N/4)-1} g[2\ell+1]e^{-j\frac{2\pi}{N/2}k(2\ell+1)}$$

o bien

$$G[k] = \underbrace{\sum_{\ell=0}^{(N/4)-1} g[2\ell]e^{-j\frac{2\pi}{N/4}k\ell}}_{\substack{\text{TDF de } N/4 \text{ puntos de las} \\ \text{muestras } \textbf{pares} \text{ de } g[n]}} + e^{-j\frac{2\pi}{N/2}k} \underbrace{\sum_{\ell=0}^{(N/4)-1} g[2\ell+1]e^{-j\frac{2\pi}{N/4}k\ell}}_{\substack{\text{TDF de } N/4 \text{ puntos de las} \\ \text{muestras } \textbf{impares} \text{ de } g[n]}}. \tag{11.8}$$

De manera análoga, $H[k]$ se puede expresar como

$$H[k] = \underbrace{\sum_{\ell=0}^{(N/4)-1} h[2\ell]e^{-j\frac{2\pi}{N/4}k\ell}}_{\substack{\text{TDF de } N/4 \text{ puntos de las} \\ \text{muestras } \textbf{pares} \text{ de } h[n]}} + e^{-j\frac{2\pi}{N/2}k} \underbrace{\sum_{\ell=0}^{(N/4)-1} h[2\ell+1]e^{-j\frac{2\pi}{N/4}k\ell}}_{\substack{\text{TDF de } N/4 \text{ puntos de las} \\ \text{muestras } \textbf{impares} \text{ de } h[n]}}. \tag{11.9}$$

En consecuencia, la TDF de $(N/2)$ puntos $G[k]$ se puede obtener combinando dos TDF de $(N/4)$ puntos: una TDF de $N/4$ puntos de la subsucesión de $g[n]$ formada por las muestras pares $g[2\ell]$ y otra TDF de $N/4$ puntos de la subsucesión de $g[n]$ formada por las muestras impares $g[2\ell+1]$. Análogamente, la TDF de $(N/2)$ puntos $H[k]$ se puede obtener combinando dos TDF de $(N/4)$ puntos de las subsucesiones de $h[n]$ formadas por las muestras pares $h[2\ell]$ e impares $h[2\ell+1]$. Por tanto, las TDF de 4 puntos de la Fig. 11.1 calculadas con (11.8) y (11.9) se representan como indica la Fig. 11.2.

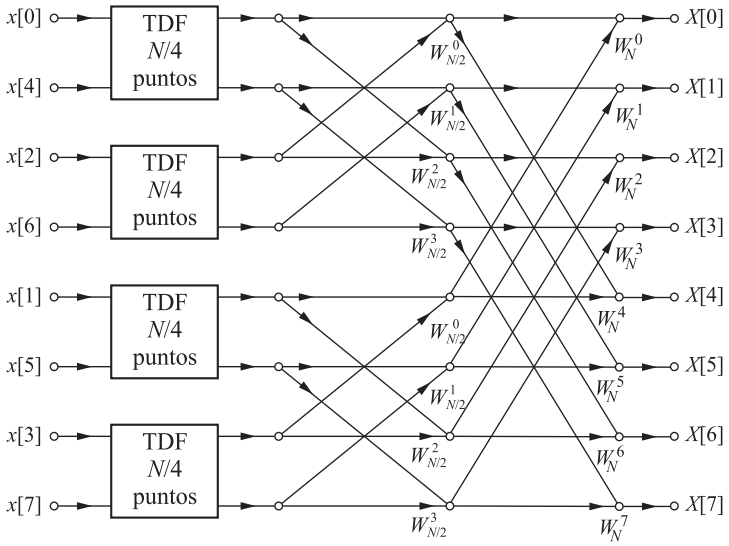


Fig. 11.3. DET: sustitución de la estructura de la Fig. 11.2 en la Fig. 11.1.

La sucesión $g[n]$ esta formada por las muestras pares de $x[n]$, y $h[n]$ por las muestras impares de $x[n]$, es decir

$$g[n] = x[2n] \quad \Longrightarrow \quad \begin{cases} g[2\ell] = x[4\ell] & \text{(muestras pares de } g[n]) \\ g[2\ell+1] = x[4\ell+2] & \text{(muestras impares de } g[n]) \end{cases}$$

$$h[n] = x[2n+1] \quad \Longrightarrow \quad \begin{cases} h[2\ell] = x[4\ell+1] & \text{(muestras pares de } h[n]) \\ h[2\ell+1] = x[4\ell+3] & \text{(muestras impares de } h[n]) \end{cases}$$

de manera que

$$\begin{array}{rcccc} g[0] & = & x[0], & g[2] = x[4], & g[4] = x[8], ... \\ g[1] & = & x[2], & g[3] = x[6], & g[5] = x[10], ... \\ h[0] & = & x[1], & h[2] = x[5], & h[4] = x[9], ... \\ h[1] & = & x[3], & h[3] = x[7], & h[5] = x[11], ... \end{array}$$

como se aprecia en la Fig. 11.2. Insertando este esquema en el diagrama de la Fig. 11.1 se obtiene la Fig. 11.3, donde se han expresado los coeficientes en función de potencias de $e^{-j\frac{2\pi}{N}} = W_N$ en vez de en potencias de $e^{-j\frac{2\pi}{N/2}} = W_{N/2}$ utilizando el hecho de que $e^{-j\frac{2\pi}{N/2}} = (e^{-j\frac{2\pi}{N}})^2$, es decir $(W_{N/2})^2 = W_N$.

Para la TDF de 8 puntos utilizada como ejemplo, el cálculo se ha reducido a calcular varias TDF de 2 puntos. Estas son muy sencillas: si $y[n]$ es una sucesión de dos puntos, la TDF $Y[k]$ está dada por

$$Y[k] = \sum_{n=0}^{N-1} y[n] e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^{1} y[n] e^{-j\frac{2\pi}{2}kn} = y[0] + (-1)^k y[1],$$

de modo que

$$\begin{aligned} Y[0] &= y[0] + y[1], \\ Y[1] &= y[0] - y[1]. \end{aligned}$$

Por lo tanto, la TDF de 2 puntos sólo necesita una suma y una resta; la Fig. 11.4 muestra la TDF de 2 puntos de la sucesión formada por $x[0]$ y $x[4]$. Si se inserta este diagrama en el esquema de la Fig. 11.3 se obtiene la Fig. 11.5.

En el caso de un N genérico, pero todavía con N potencia de 2, las transformadas de $(N/4)$ puntos de (11.8) y (11.9) se pueden seguir descomponiendo en dos transformadas de $(N/8)$ puntos cada una. Esta descomposición se puede continuar hasta llegar a transformadas de 2 puntos, siendo necesarias $\alpha = \log_2 N$ etapas de cómputo. Como en la descomposición original de una transformada de N puntos en dos transformadas de $(N/2)$ puntos, se requerían $N + 2(N/2)^2$ multiplicaciones y sumas, cuando las transformadas de $(N/2)$ puntos se descomponen en transformadas de $(N/4)$ puntos, el factor $(N/2)^2$ se debe sustituir por $N/2+2(N/4)^2$ por lo que el número total de operaciones es $N+N+4(N/4)^2$ multiplicaciones y sumas complejas. Si $N = 2^\alpha$, este proceso se puede repetir un máximo de $\alpha = \log_2 N$ veces, por lo que tras realizar la descomposición tantas veces como sea posible, el número de multiplicaciones y sumas es igual a $N\alpha = N \log_2 N$.

El número de cálculos necesarios para computar la TDF de la Fig. 11.5 se puede reducir aún más explotando la simetría y la periodicidad de los coeficientes $W_N^r = e^{-j\frac{2\pi}{N}r}$. En el esquema de la Fig. 11.5 para pasar de una etapa a la siguiente se debe efectuar el cálculo básico que muestra la Fig. 11.6(a), que por la forma de representación se denomina *mariposa*, donde los coeficientes son siempre potencias de $W_N = e^{j\frac{2\pi}{N}}$ y los exponentes difieren en $N/2$. Como

$$W_N^{N/2} e^{-j\frac{2\pi}{N}\frac{N}{2}} = e^{-j\pi} = -1,$$

el factor $W_N^{(r+N/2)} = e^{-j\frac{2\pi}{N}(r+\frac{N}{2})}$ se puede escribir como

$$W_N^{(r+N/2)} = e^{-j\frac{2\pi}{N}(r+\frac{N}{2})} = e^{-j\frac{2\pi}{N}\frac{N}{2}} e^{j\frac{2\pi}{N}r} = -e^{j\frac{2\pi}{N}r} = -W_N^r.$$

Teniendo en cuenta este resultado el cálculo de la mariposa de la Fig. 11.6(a) se puede simplificar como se muestra en la Fig. 11.6(b), que requiere sólo una multiplicación compleja en vez de dos. Sustituyendo las mariposas de la Fig. 11.6(a) por el diagrama de la Fig. 11.6(b), la Fig. 11.5 se convierte en el esquema de la Fig. 11.7, en donde el número de multiplicaciones complejas se reduce en un factor de 2.

El diagrama de la Fig. 11.7 muestra explícitamente las operaciones necesarias para el cálculo de una TDF de $N = 8$ puntos. En cada etapa se realizan $N/2$ multiplicaciones

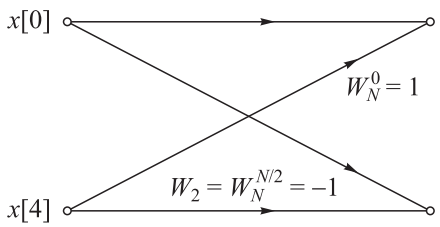


Fig. 11.4. DET: Esquema de cálculo de una TDF de 2 puntos.

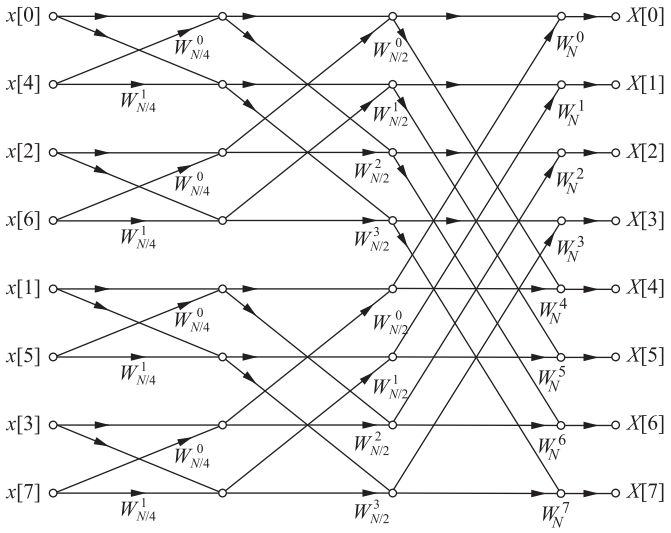


Fig. 11.5. DET: esquema completo de una TDF de 8 puntos.

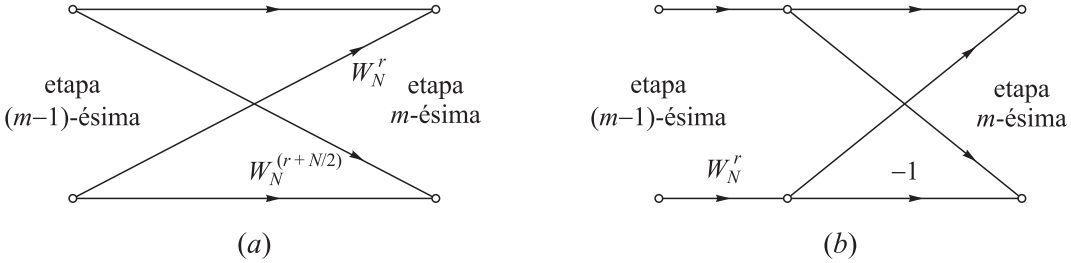


Fig. 11.6. Una de las *mariposas* de la Fig.11.5 (a). Mariposa con sólo un producto complejo (b).

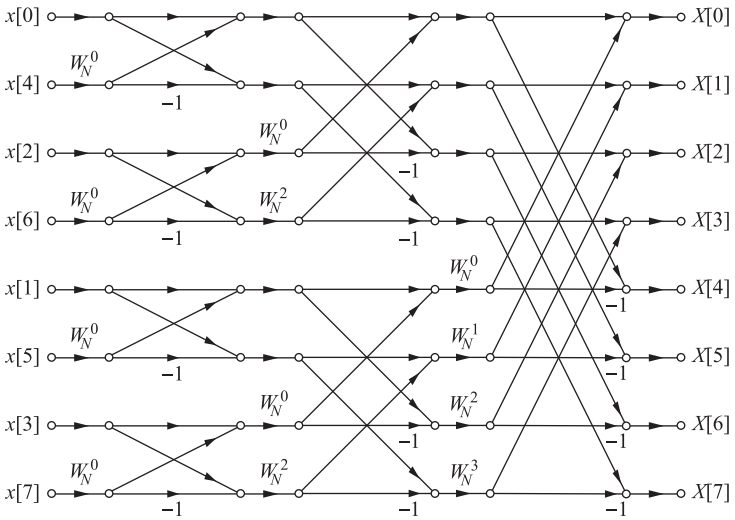


Fig. 11.7. DET: TDF de 8 puntos utilizando la mariposa de la Fig.11.6(b).

complejas (arcos con ganancia $W_N^r = e^{-j\frac{2\pi}{N}r}$) y N sumas complejas. Como hay $\log_2 N$ etapas, resulta un total de $(N/2)\log_2 N$ multiplicaciones y $N\log_2 N$ sumas, mucho menor que el número de operaciones necesarias para la implementación directa, aproximadamente igual a N^2. Por ejemplo, si $N = 2^{10} = 1\,024$, entonces $N^2 = 2^{20} = 1\,048\,576$, y $(N/2)\log_2 N = 5\,120$, una reducción de más de dos órdenes de magnitud: en el caso hipotético de un procesador que realizase 10 operaciones por segundo, el cálculo directo demandaría aproximadamente 1 día y 5 horas, mientras que el cálculo efectuado con el proceso de decimación en tiempo sólo insumiría ¡8 minutos con 32 segundos!

11.3.1. Cómputo en el lugar

La Fig. 11.7 representa una manera de describir un algoritmo para el cálculo de la TDF. Las características esenciales del esquema son los arcos que conectan los nodos y las ganancias de dichos arcos. La disposición de los nodos es indistinta, pues distintos arreglos representan el mismo cálculo si se mantienen las conexiones entre los nodos y sus ganancias respectivas. La forma particular del esquema de la Fig. 11.7 resulta de desarrollar el algoritmo separando la sucesión $x[n]$ original en muestras pares e impares y subdividir de la misma manera cada nueva sucesión en subsucesiones de menor longitud. Estos esquemas no sólo describen un procedimiento eficiente para calcular la TDF sino que también sugieren una forma de almacenar los datos originales y los resultados de los cálculos intermedios en arreglos o vectores.

Por ejemplo, en cada etapa del cálculo de la Fig. 11.7 se toman N números complejos y se transforman en otros N números complejos calculando mariposas como la de la Fig. 11.6(b). El proceso se repite $\alpha = \log_2 N$ veces hasta obtener la TDF. El cálculo sugerido por la Fig. 11.7 se puede efectuar utilizando dos arreglos complejos (lugares de memoria o registros de almacenamiento), uno de ellos para el conjunto de resultados y el otro para los datos que se están utilizando en el cálculo. Por ejemplo, al calcular el primer vector de la Fig. 11.7 uno de los arreglos puede almacenar los datos de entrada y segundo contener los resultados de la primera etapa. Aunque no es imprescindible, el vector de números complejos se puede ordenar en la misma forma en que se muestran en la dicha figura (de arriba a abajo). Si $X_m[\ell]$ con $\ell = 0, 1, ..., N-1$, es la sucesión de números complejos resultantes de la *m*-ésima etapa de cálculo, con $m = 1, 2, ..., \alpha$, entonces $X_{m-1}[\ell]$ es el vector de entrada y $X_m[\ell]$ el vector de salida de la *m*-ésima etapa de los cálculos. Por conveniencia, el conjunto de muestras de entrada se notará como $X_0[\ell]$. Por tanto, para $N = 8$, como los elementos del arreglo de entrada contendrán las muestras de la entrada ordenadas de la manera siguiente

$$\begin{aligned} X_0[0] &= x[0], \\ X_0[1] &= x[4], \\ X_0[2] &= x[2], \\ X_0[3] &= x[6], \\ X_0[4] &= x[1], \\ X_0[5] &= x[5], \\ X_0[6] &= x[3], \\ X_0[7] &= x[7], \end{aligned} \tag{11.10}$$

Utilizando esta notación, la entrada y la salida de la mariposa de la Fig. 11.6(b) se etiquetan como se indica en la Fig. 11.8, es decir que la mariposa que convierte los datos de la

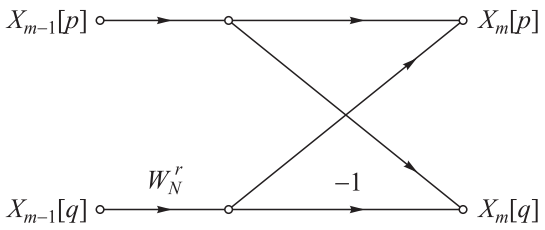


Fig. 11.8. Esquema de las ecuaciones (11.11).

etapa $(m-1)$ en los de la etapa m–ésima representa la ecuación a diferencias

$$\begin{aligned} X_m[p] &= X_{m-1}[p] + W_N^r X_{m-1}[q], \\ X_m[q] &= X_{m-1}[p] - W_N^r X_{m-1}[q]. \end{aligned} \tag{11.11}$$

donde $W_N^r = e^{-j\frac{2\pi}{N}r}$. Los índices p, q, y r en (11.11) varían de etapa en etapa como se deduce fácilmente de la Fig. 11.7 y de las ecuaciones (11.5), (11.7), (11.8), etc. De las Figs. 11.7 y 11.8 es evidente que para calcular los elementos de las posiciones p y q del vector m-ésimo solo se necesitan los números complejos de las posiciones p y q del $(m-1)$-ésimo arreglo. Por tanto, si $X_m[p]$ y $X_m[q]$ se almacenan en los mismos registros que $X_{m-1}[p]$ y $X_{m-1}[q]$ respectivamente, para realizar el cálculo completo sólo se necesitan N lugares de memoria (complejos). Esta forma de organizar los cálculos se denomina *cómputo en el lugar*. La interpretación gráfica del cómputo en el lugar en la Fig. 11.7 (o en la Fig. 11.5) es que nodos que están en la misma línea horizontal corresponden a la misma posición de memoria, y que el pasaje de un vector a otro está dado por el cálculo de una mariposa en la que los nodos de la entrada y de la salida son adyacentes en sentido horizontal.

Para poder "calcular en el lugar" la sucesión de entrada se debe almacenar (o al menos se debe poder acceder a ella) en orden no secuencial, como se observa en la Fig. 11.7. Este ordenamiento se denomina *orden de inversión de bits*. El porqué de este nombre se hace evidente al escribir el índice del arreglo (11.10) en formato binario:

$$\begin{aligned} X_0[0] &= x[000], \\ X_0[1] &= x[100], \\ X_0[2] &= x[010], \\ X_0[3] &= x[110], \\ X_0[4] &= x[001], \\ X_0[5] &= x[101], \\ X_0[6] &= x[011], \\ X_0[7] &= x[111], \end{aligned} \tag{11.12}$$

Si (n_2, n_1, n_0) es la representación binaria del índice de la sucesión $x[n]$, el valor del elemento $x[n_2, n_1, n_0]$ se almacena en el lugar $X_0[n_0, n_1, n_2]$ del arreglo de memoria. En otras palabras, para determinar la posición de $x[n_2, n_1, n_0]$ en el arreglo de memoria se debe invertir el orden de los bits del índice.

El orden que deben respetar los datos de entrada para poder efectuar un cómputo en el lugar es consecuencia directa de la forma de subdividir la sucesión original para calcular la TDF. Para comprender este proceso, en la Fig. 11.9(a) se muestra una sucesión de datos

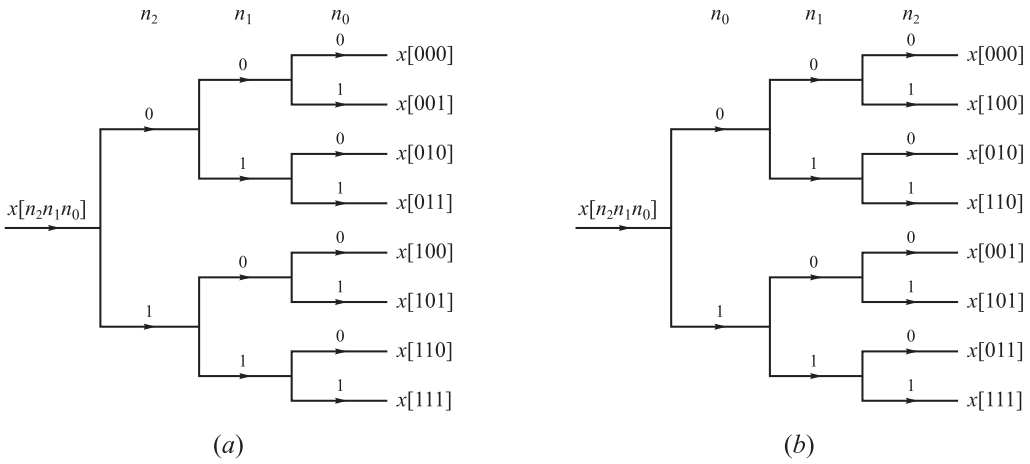


Fig. 11.9. (a) Orden natural. (b) Orden de inversión de bits.

ordenada naturalmente, explicitando sucesivamente los bits que representan el índice de entrada. Si el bit más significativo del índice de los datos es cero, $x[n]$ pertenece a la mitad superior del arreglo ordenado; en caso contrario pertenece a la mitad inferior. Seguidamente, las subsucesiones de las mitades superior e inferior se pueden ordenar examinando el segundo bit más significativo, y así sucesivamente.

Para calcular la TDF por decimación en tiempo, la sucesión $x[n]$ se separa primero en las muestras pares, que se colocan en la mitad superior de la figura y en las muestras impares, que se colocan en la mitad inferior. Esa separación de los datos se puede realizar examinando el bit menos significativo del índice (n_0). Si el bit menos significativo es 0, el valor de la sucesión corresponde a una muestra par y por tanto debe aparecer en la parte superior del vector $X_0[\ell]$ y si el bit menos significativo es 1, el valor de la sucesión corresponde a una muestra impar y por consiguiente debe aparecer en la parte inferior del vector. A continuación, las subsucesiones de índice par e impar se ordenan a su vez en sus elementos pares e impares, lo que se puede hacer examinando el segundo bit menos significativo del índice. Si el segundo bit menos significativo es 0 (1), el valor es un término de índice par (impar) de dicha subsecuencia. El proceso se repite hasta obtener N subsucesiones de longitud 1. La Fig. 11.9(b) muestra la forma de efectuar el ordenamiento.

Los diagramas de las Figs.11.9(a) y 11.9(b) son idénticos, excepto que para el ordenamiento natural se examinan los bits del índice de *izquierda a derecha*, mientras que en el orden por inversión de bits, necesario para el cálculo en el lugar de la Fig. 11.5 o de la Fig. 11.7 los bits se examinan de *derecha a izquierda*, es decir, con los bits ordenados de manera inversa. Por tanto, la necesidad del orden de inversión de bits de la sucesión $x[n]$ es consecuencia de la forma en la que se descomponen los cálculos de la TDF en sucesivas TDF de menor menores (separando las muestras pares e impares de cada subsucesión), como se indica en las Figs. 11.5 y 11.7.

11.3.2. Formas alternativas

Aunque es razonable ordenar los resultados de cada etapa de cálculo en la forma sugerida por la Fig. 11.7, no es obligatorio hacerlo de esta forma. Independientemente de la dispo-

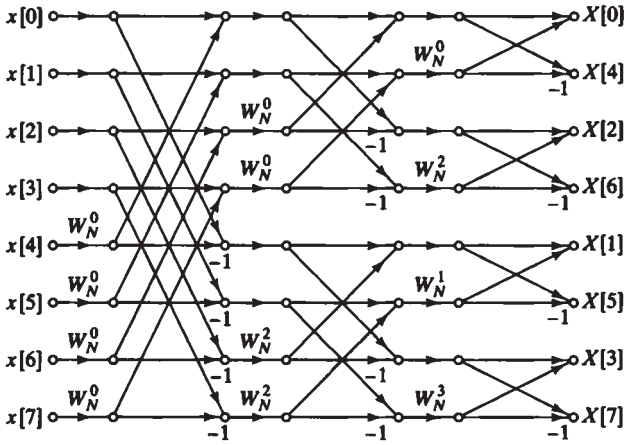


Fig. 11.10. DET: Arreglo de la Fig.11.7 con entrada en orden normal y salida en orden de inversión de bits.

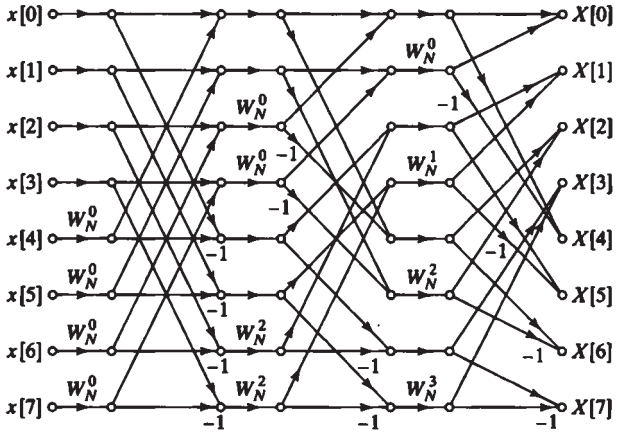


Fig. 11.11. DET: reordenamiento de la Fig.11.7 con entrada y salida en orden normal.

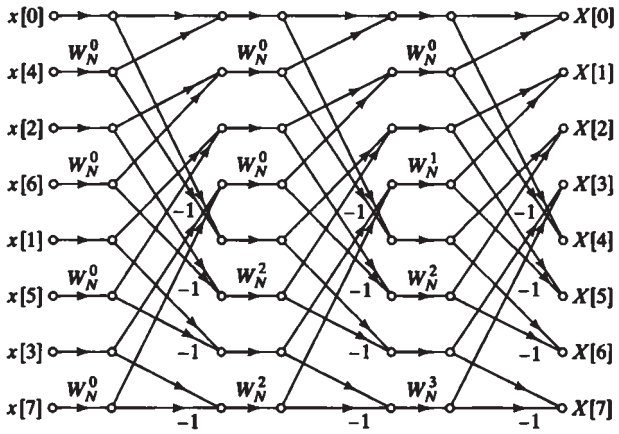


Fig. 11.12. DET: reordenamiento de la Fig.11.7 que preserva la simetría de cada etapa.

sición de los nodos de la Fig. 11.7, el resultado siempre será un cálculo válido de la TDF de $x[n]$ siempre que las ganancias de los arcos no se modifiquen. Sólo cambia el orden en que se almacenan y se accede a los datos. Si los nodos se asocian con la indización de un arreglo (complejo) de almacenamiento sólo se obtendrá un esquema de cálculo "en el lugar" si los nodos de entrada y de salida de cada mariposa son adyacentes horizontalmente. Si no es así, se necesitarán dos arreglos de memoria. La Fig. 11.7 es un ejemplo de disposición del primer tipo. En la Fig. 11.10 se muestra otra disposición, en donde la sucesión $x[n]$ de entrada está en orden normal y la sucesión $X[k]$ en orden de inversión de bits. La Fig. 11.10 se puede obtener a partir de la Fig. 11.7 intercambiando todos los nodos horizontalmente adyacentes a $x[1]$ con todos los nodos adyacentes horizontalmente a $x[4]$; todos los nodos horizontalmente adyacentes a $x[6]$ se intercambian con los adyacentes horizontalmente a $x[3]$, etc. Los nodos horizontalmente adyacentes a $x[0]$, $x[2]$, $x[5]$ y $x[7]$ no se modifican. El esquema de la Fig. 11.10 representa la forma del algoritmo de decimación en el tiempo propuesta originalmente por Cooley y Tukey (1965).

La única diferencia entre las Figs. 11.7 y 11.10 es el orden de los nodos; las ganancias de los arcos (potencias de $W_n = e^{j\frac{2\pi}{N}}$) no cambian. Aunque desplazando los nodos se puede obtener una amplia variedad de esquemas de cálculo, la mayoría de ellas no tiene mucho sentido desde un punto de vista computacional. Por ejemplo, si los nodos se ordenan de forma que la entrada y la salida aparecen en orden normal, como se muestra en la Fig. 11.11, se pierde la facultad realizar el cómputo en el lugar debido a que la estructura en mariposa no se mantiene tras la primera etapa, de manera que para realizar los cálculos indicados por esta figura se necesitarían dos arreglos complejos de longitud N.

Para realizar los cálculos de las Figs. 11.7, 11.10 y 11.11 es necesario acceder a los elementos de los arreglos intermedios en orden no secuencial. Por tanto, para conseguir mayor velocidad, los números complejos se deben almacenar en memoria de acceso aleatorio (RAM). Por ejemplo, en el cálculo de la primera etapa de la Fig. 11.7, las entradas de cada mariposa están en nodos adyacentes y se consideran almacenadas en posiciones de almacenamiento contiguas. En el cálculo de la segunda etapa las entradas a una mariposa están separadas por 2 lugares de memoria. En el cálculo del tercer vector a partir del segundo, las entradas a una mariposa están separadas por 4 posiciones. Si $N > 8$, la separación entre las mariposas de entrada es 8 para la cuarta etapa, 16 para la quinta etapa, etc. La separación en la última etapa (α) es $N/2$. En la Fig. 11.10, en cambio, el cálculo del primer vector a partir de los datos de entrada necesita datos separados por 4 muestras. Para calcular el segundo vector a partir del primero los datos quedan separados por 2 elementos, y finalmente, para calcular el último vector se utilizan datos adyacentes. En síntesis, en los esquemas de las Figs. 11.7 o 11.10 aunque se necesita un mínimo de lugares de memoria, los datos no se acceden secuencialmente, lo que hace deseable que estos datos estén almacenados en memorias de acceso aleatorio (RAM). El diagrama de la Fig. 11.11 no tiene ninguna ventaja aparente, ya que los datos se acceden de forma no secuencial, el cómputo no es en el lugar, el esquema de indización es considerablemente más complicado que en cualquiera de los dos casos anteriores, y se necesitan lugares de almacenamiento adicionales.

La Fig. 11.12 muestra una configuración distinta de la Fig. 11.7 que es particularmente útil cuando no se dispone de memoria de acceso aleatorio, y representa al algoritmo de decimación en el tiempo propuesto por Singleton (1969); en DSP Commitee (1979) hay un programa que utiliza este esquema de cómputo. En este diagrama la entrada está en orden de inversión de bits y la salida está en orden normal, pero la característica importante

es que se preserva la geometría de cada etapa; sólo las ganancias de los arcos cambian de etapa en etapa. De esta forma es posible acceder a los datos de manera secuencial, lo que era importante en la década del '60 cuando la memoria RAM era escasa y costosa, y el almacenamiento se hacía en unidades de cinta o disco. Si se definen cuatro archivos diferentes en disco, y la primera mitad de la sucesión de entrada se almacena (en orden de inversión de bits) en el archivo 1 y la segunda mitad en el archivo 2, la sucesión se puede leer secuencialmente de los archivos 1 y 2 y los resultados se pueden escribir secuencialmente en los archivos 3 y 4. La primera mitad del nuevo arreglo se escribe en el archivo 3 y la segunda mitad en el archivo 4. En la siguiente etapa de cómputo los archivos 3 y 4 corresponden a la entrada y la salida se escribe en los archivos 1 y 2. Este proceso se repite para cada una de las α etapas. Aunque hoy en día no se usa memoria en cinta magnética, el esquema conserva su interés para el caso en que sea necesario calcular la TDF de sucesiones extremadamente largas (por ejemplo, en MATLAB el cálculo de sucesiones de más de 4×10^6 puntos es extremadamente lenta) o bien en el caso de sucesiones de menor longitud cuyas TDF quieran calcularse muy rápidamente disponiendo los datos en la memoria caché del procesador en lugar de utilizar la memoria RAM del sistema.

11.4. Algoritmos de decimación en frecuencia

Los algoritmos de FFT por decimación en tiempo se basan en calcular la TDF dividiendo la sucesión temporal $x[n]$ en sucesiones de menor longitud. Una forma dual se obtiene si se divide la sucesión frecuencial $X[k]$ de la TDF en sucesiones más pequeñas; estos algoritmos se denominan de *decimación en frecuencia* (DEF).

Se supone nuevamente que N es potencia de 2, y se distinguen las muestras pares e impares de la transformada $X[k]$. Como

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk}, \quad k = 0,\, 1,\, \ldots,\, N-1, \tag{11.13}$$

las muestras pares son

$$X[2r] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}n(2r)}, \quad r = 0,\, 1,\, \ldots,\, (N/2)-1, \tag{11.14}$$

que puede expresarse como

$$X[2r] = \sum_{n=0}^{(N/2)-1} x[n] e^{-j\frac{2\pi}{N}n(2r)} + \sum_{n=(N/2)}^{N-1} x[n] e^{-j\frac{2\pi}{N}n(2r)}. \tag{11.15}$$

Sustituyendo variables en la segunda suma de (11.15), se obtiene

$$X[2r] = \sum_{n=0}^{(N/2)-1} x[n] e^{-j\frac{2\pi}{N}n(2r)} + \sum_{n=0}^{(N/2)-1} x[n+(N/2)] e^{-j\frac{2\pi}{N}[n+(N/2)](2r)}. \tag{11.16}$$

Finalmente, debido a la periodicidad de $e^{-j\frac{2\pi}{N}n(2r)} = W_N^{n(2r)}$,

$$W_N^{[n+(N/2)](2r)} = e^{-j\frac{2\pi}{N}[n+(N/2)](2r)} = e^{-j\frac{2\pi}{N}n(2r)} e^{-j\frac{2\pi}{N}Nr} = e^{-j\frac{2\pi}{N}n(2r)} = W_N^{n(2r)}, \tag{11.17}$$

y como $e^{-j\frac{2\pi}{N}n(2r)} = W_N^{n(2r)} = e^{-j\frac{2\pi}{N/2}nr} = W_{N/2}^{nr}$, la ecuación (11.16) se puede expresar como

$$X[2r] = \sum_{n=0}^{(N/2)-1} \left(x[n] + x[n+(N/2)]\right) e^{-j\frac{2\pi}{N/2}nr}, \quad r = 0,\, 1,\, \ldots,\, (N/2)-1. \tag{11.18}$$

La ecuación (11.18) es la TDF de $N/2$ puntos de la sucesión de $N/2$ puntos obtenida sumando la primera y segunda mitad de la sucesión $x[n]$ de entrada. La suma de las dos mitades de la entrada produce solapamiento temporal, ya que al calcular sólo las muestras pares en frecuencia se submuestrea la transformada de Fourier de $x[n]$.

Resta calcular las muestras impares de la TDF, dados por

$$X[2r+1] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}n(2r+1)}, \quad r = 0,\, 1,\, \ldots,\, (N/2)-1. \tag{11.19}$$

Como antes, (11.19) se puede escribir como

$$X[2r+1] = \sum_{n=0}^{(N/2)-1} x[n] e^{-j\frac{2\pi}{N}n(2r+1)} + \sum_{n=(N/2)}^{N-1} x[n] e^{-j\frac{2\pi}{N}n(2r+1)}. \tag{11.20}$$

La segunda suma de la ecuación (11.20) es

$$\begin{aligned} \sum_{n=(N/2)}^{N-1} x[n] e^{-j\frac{2\pi}{N}n(2r+1)} &= \sum_{n=0}^{(N/2)-1} x[n+(N/2)] e^{-j\frac{2\pi}{N}[n+(N/2)](2r+1)} \\ &= e^{-j\frac{2\pi}{N}(N/2)(2r+1)} \sum_{n=0}^{(N/2)-1} x[n+(N/2)] e^{-j\frac{2\pi}{N}n(2r+1)} \\ &= -\sum_{n=0}^{(N/2)-1} x[n+(N/2)] e^{-j\frac{2\pi}{N}n(2r+1)}, \end{aligned} \tag{11.21}$$

donde se ha tenido en cuenta que $e^{-j\frac{2\pi}{N}(N/2)(2r)} = W_N^{(N/2)(2r)} = 1$, y que $e^{-j\frac{2\pi}{N}(N/2)} = W_N^{N/2} = -1$. Sustituyendo (11.21) en (11.20) y combinando las dos sumas, se obtiene

$$X[2r+1] = \sum_{n=0}^{(N/2)-1} \left(x[n] - x[n+(N/2)]\right) e^{-j\frac{2\pi}{N}n(2r+1)}, \tag{11.22}$$

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

$$X[2r+1] = \sum_{n=0}^{(N/2)-1} \left\{\left(x[n] - x[n+(N/2)]\right) e^{-j\frac{2\pi}{N}n}\right\} e^{-j\frac{2\pi}{N/2}nr}, \quad r = 0,\, 1,\, \ldots,\, (N/2)-1. \tag{11.23}$$

La ecuación (11.23) es la TDF de $(N/2)$ puntos de la sucesión que se obtiene restando a la primera mitad de la sucesión de entrada la segunda mitad, y multiplicando la sucesión resultante por $e^{-j\frac{2\pi}{N}n} = W_N^n$. Por tanto, de acuerdo con (11.18) y (11.23), con

$g[n] = x[n]+x[n+(N/2)]$ y $h[n] = x[n]-x[n+(N/2)]$, la TDF se puede calcular formando primero las sucesiones $g[n]$ y $h[n]$, calculando luego $h[n]e^{-\frac{2\pi}{N}n}$ y calculando finalmente las TDF de $(N/2)$ puntos de esas dos sucesiones, con lo que se obtienen respectivamente las muestras pares e impares de la TDF. La Fig. 11.13 ilustra el procedimiento sugerido por las ecuaciones (11.18) y (11.23) para una TDF de 8 puntos.

Como N es potencia de 2, $N/2$ es par, de manera que las TDF de $(N/2)$ puntos se pueden calcular separando las muestras pares e impares de la salida. Análogamente al desarrollo que conduce a (11.18) y (11.23), se deben combinar la primera y la segunda mitad de los muestras de entrada de cada una de las dos TDF de $(N/2)$ puntos, y calcular TDF de $(N/4)$ puntos. La Fig. 11.14 muestra el esquema que resulta al realizar este paso en el ejemplo de $N = 8$ puntos. En este caso el problema se ha reducido al cálculo de TDF de 2 puntos, que se realizan sumando y restando las muestras de entrada, como se muestra en la Fig. 11.15. La sustitución de este bloque en el esquema de la Fig. 11.14 resulta en la Fig. 11.16, que muestra el algoritmo completo para calcular una TDF de 8 puntos.

Generalizando para $N = 2^\alpha$, se deduce que el cálculo de la Fig. 11.16 necesita $(N/2)\log_2 N$ multiplicaciones complejas y $N \log_2 N$ sumas complejas, de modo que el número total de operaciones en el algoritmo de decimación en tiempo es el mismo que en el algoritmo de decimación en frecuencia.

11.4.1. Cómputo en el lugar

El diagrama de cálculo de la TDF por decimación en frecuencia de la Fig. 11.16 exhibe similitudes y diferencias con los esquemas de decimación en tiempo (por ejemplo, la Fig. 11.10). El esquema de la Fig. 11.16 representa el mismo cálculo independientemente de la manera en que se dibuje el grafo, siempre y cuando se conecten los mismos nodos con las ganancias adecuadas. Si los nodos verticales sucesivos se asocian a lugares de memoria, la Fig. 11.16 requiere la sucesión de entrada en orden natural y proporciona la TDF en orden de inversión de bits. El cálculo básico tiene nuevamente la estructura de una mariposa (Fig. 11.17), pero ésta es diferente a la de los algoritmos de decimación en tiempo que se muestra en la Fig. 11.6(b). Debido a la naturaleza del cálculo de la mariposa, el grafo de flujo de la Fig. 11.16 se puede interpretar como un cómputo en el lugar.

11.4.2. Formas alternativas

Transponiendo los esquemas de decimación en tiempo que se presentaron en las Sección 11.3.2 se pueden obtener varias formas alternativas para el algoritmo de decimación en frecuencia. Si $X_m[\ell]$, donde $\ell = 0, 1, \ldots, N-1$ y $m = 0, 1, \ldots, \alpha$, es la sucesión de números complejos resultante del cálculo de la m-ésima etapa, las operaciones indicadas por la mariposa básica de la Fig. 11.17 tiene la forma

$$\begin{aligned} X_m[p] &= X_{m-1}[p] + X_{m-1}[q], \\ X_m[q] &= (X_{m-1}[p] - X_{m-1}[q])\, e^{-j\frac{2\pi}{N}r}. \end{aligned} \tag{11.24}$$

Comparando las Figs. 11.8 y 11.17 o las ecuaciones (11.11) y (11.24) se observa que los cálculos de la mariposa son diferentes para las dos clases de algoritmos. Sin embargo, existe una semejanza entre los esquemas de las Figs. 11.8 y 11.17 y los de las Figs. 11.7 y

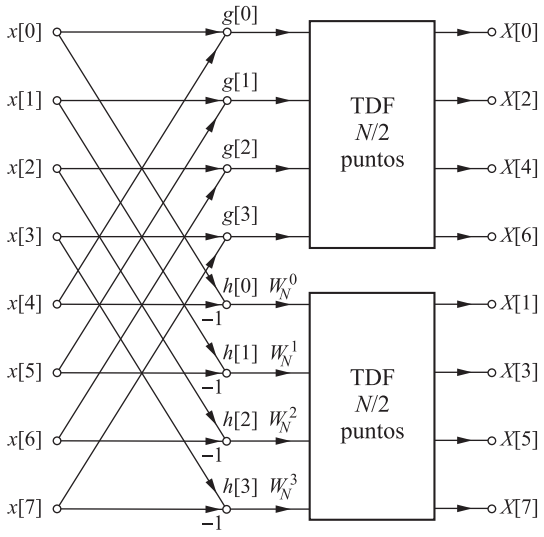


Fig. 11.13. DEF: cálculo de una TDF de N puntos con dos TDF de $(N/2)$ puntos $(N = 8)$.

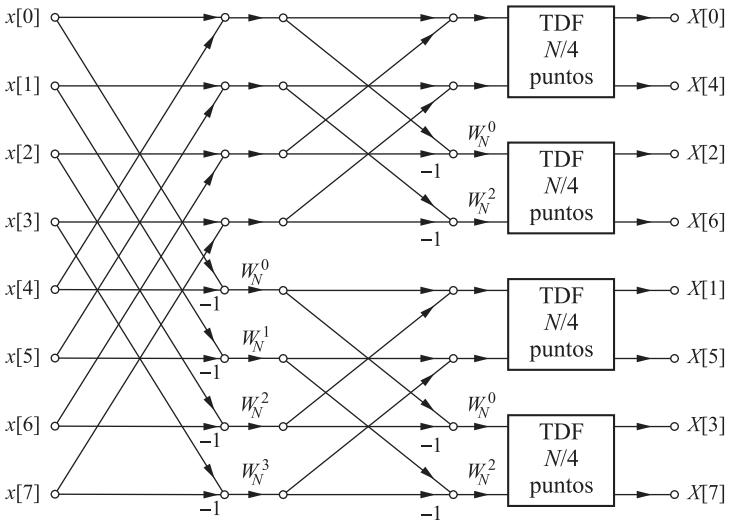


Fig. 11.14. DEF: descomposición de una TDF de 8 puntos en cuatro TDF de 2 puntos.

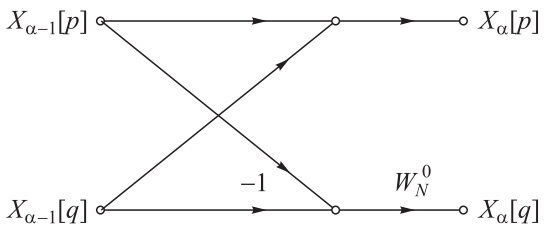


Fig. 11.15. DEF: cálculo *mariposa* de la última etapa.

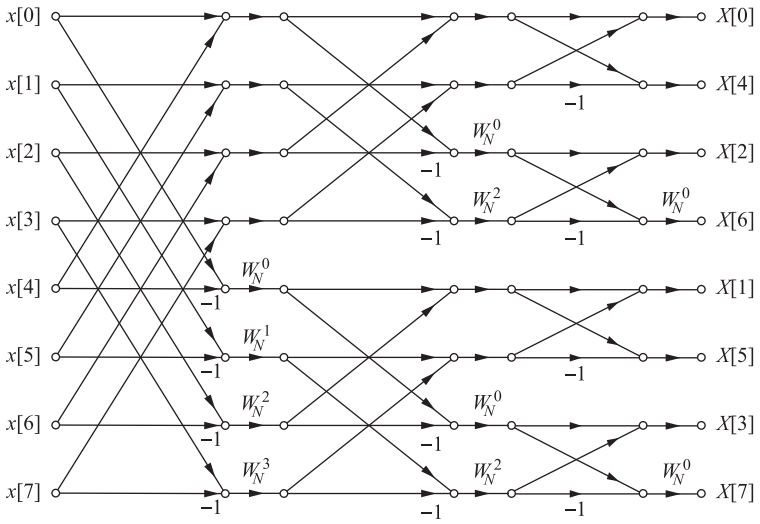


Fig. 11.16. DEF: esquema de una TDF de 8 puntos.

11.16. Concretamente, la Fig. 11.16 se puede obtener a partir de la Fig. 11.7 y la Fig. 11.17 a partir de la Fig. 11.8 invirtiendo la dirección del flujo de las señales e intercambiando la entrada y la salida: la Fig. 11.16 es el diagrama transpuesto de la Fig. 11.7 y la Fig. 11.17 es el transpuesto del esquema de la Fig. 11.8. La aplicación del teorema de transposición no es directa en este caso, ya que los diagramas tienen más de un nodo de entrada y de salida; sin embargo, se verifica que las características de entrada-salida de los esquemas de las Figs. 11.7 y 11.16 son las mismas, lo que puede demostrarse notando que las ecuaciones de la mariposa (11.24) se pueden resolver hacia atrás, comenzando con el vector de salida. De manera general, para cada algoritmo de FFT por decimación en tiempo existe un algoritmo de FFT por decimación en frecuencia que se obtiene a intercambiando la entrada con la salida e invirtiendo la dirección de todas las flechas de los diagramas de flujo.

Este resultado implica que todos los esquemas de la Sección 11.3 se pueden reformular como algoritmos de decimación en frecuencia. Aplicando el procedimiento de transposición a la Fig. 11.10 se llega a la Fig. 11.18. En este grafo de flujo, la salida está en orden normal y la entrada está en orden de inversión de bits. Alternativamente, el grafo de flujo transpuesto del que se presenta en la Fig. 11.11 es el de la Fig. 11.19, donde tanto la entrada como la salida están en orden normal. Como en el caso de la Fig. 11.11, este grafo

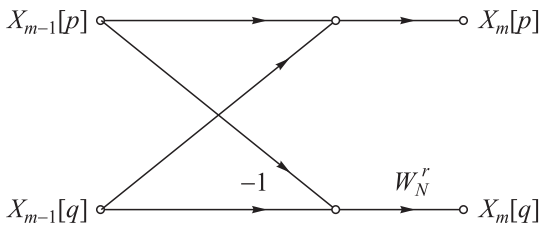


Fig. 11.17. DEF: Cálculo en mariposa típico de la Fig. 11.16.

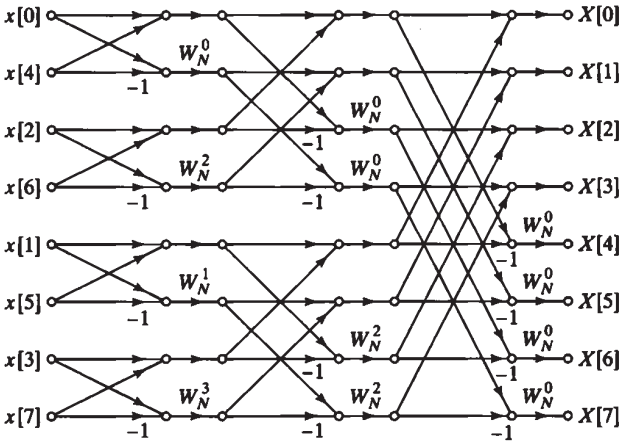


Fig. 11.18. DEF: reordenamiento de la Figura 11.16, con entrada en orden normal (transpuesto de la Fig. 11.10).

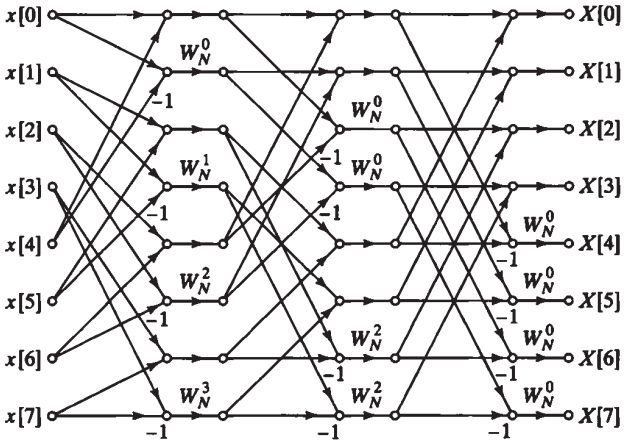


Fig. 11.19. DEF: Reordenamiento de la Fig. 11.16 con la entrada y la salida en orden normal (transpuesto de la Fig. 11.11).

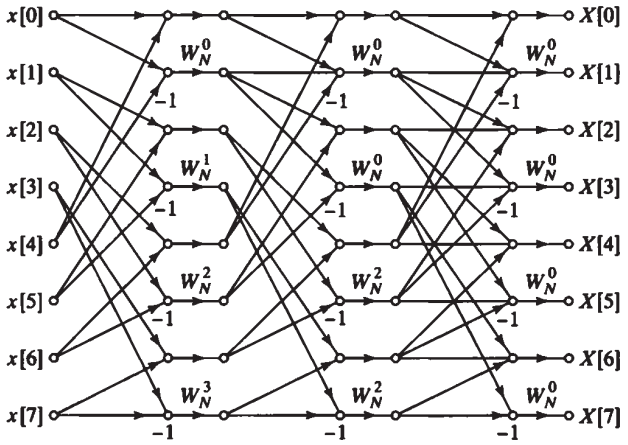


Fig. 11.20. DEF: Reordenamiento de la Fig. 11.16 que preserva la geometría de cada etapa (transpuesto de la Fig. 11.12).

de flujo no corresponde a un cómputo en el lugar.

La Fig. 11.20 muestra el grafo transpuesto al de la Fig. 11.12. Todas las etapas de la Fig. 11.20 tienen la misma geometría, propiedad que es deseable para al cálculo de transformadas grandes utilizando almacenamiento secuencial de datos, como ya se comentó.

11.5. Consideraciones prácticas

En las Secciones 11.3 y 11.4 se han presentado dos alternativas para calcular eficientemente la TDF cuando N es potencia entera de 2. En la exposición se ha hecho hincapié en las representaciones esquemáticas en lugar de escribir detalladamente las ecuaciones, y se han ilustrado diagramas para valores concretos de N (en particular, para $N = 8$). Sin embargo, a partir de un esquema como el de la Fig. 11.7 se puede generalizar un algoritmo de cómputo válido para cualquier valor $N = 2^\alpha$.

Si bien los esquemas de las secciones previas capturan la esencia de los algoritmos de FFT, al momento de implementar un algoritmo determinado deben tenerse en cuenta algunos detalles, que se comentarán brevemente en esta sección. Concretamente, en la Sección 11.5.1 se presentan aspectos relacionados con el acceso y almacenamiento de los *datos* en los arreglos intermedios, mientras que en la Sección 11.5.2 se comentan dos alternativas para el tratamiento de los *coeficientes* $W_N^{kn} = e^{-j\frac{2\pi}{N}kn}$: el cómputo recursivo o el almacenamiento en tablas. Si bien la discusión se basa en algoritmos en los que N es potencia de 2, la ideas se pueden extender al caso general; para simplificar la presentación se analizará principalmente el algoritmo de decimación en tiempo de la Fig. 11.7.

11.5.1. Indización

En el algoritmo de la Fig. 11.7, para que el cómputo se pueda realizar en el lugar es necesario que la entrada esté en orden de inversión de bits. El resultado (la TDF) queda ordenado naturalmente. Generalmente, las sucesiones no se originan con orden de inversión de bits, de forma que el primer paso en la realización de la Fig. 11.7 es reordenar la sucesión $x[n]$. Como muestra esta figura y las ecuaciones (11.10) y (11.11), la ordenación por inversión de bits se puede realizar en el lugar, ya que los elementos se intercambian de a pares: una muestra en un índice $[n_2n_1n_0]$ dado se intercambia con la muestra especificada por el índice con los bits al revés $[n_0n_1n_2]$. Esta tarea se puede realizar por software usando dos contadores para los índices, uno en orden normal y el otro en orden de inversión de bits: basta con intercambiar los datos apuntados por los dos contadores. Una vez que la entrada está en orden de inversión de bits, se puede proceder con la primera etapa de los cálculos. En este caso, las entradas a las mariposas son elementos adyacentes del vector $X_0[\,\cdot\,]$ En la segunda etapa, las entradas a las mariposas están separadas por 2 muestras, y en la *m*-ésima etapa las entradas a las mariposas quedan separadas por 2^{m-1} muestras. Los coeficientes de la *m*-ésima etapa son potencias de $e^{-j\frac{2\pi}{N}(N/2^m)} = W_N^{N/2^m}$, ordenadas naturalmente si el cálculo de las mariposas empieza por la parte superior de la Fig. 11.7. Estas consideraciones definen el acceso a los datos de una etapa determinada, que lógicamente depende del esquema que se implemente. Por ejemplo, en la etapa *m*-ésima de la Fig. 11.10, la separación de las mariposas es $2^{\alpha-m}$ y en este caso los coeficientes deben ordenarse por inversión de bits. En este caso la entrada $x[n]$ debe estar ordenada natural-

mente, pero la salida está en orden de inversión de bits, por lo que en general es necesario reordenar la salida utilizando un par de contadores como se comentó más arriba.

Cada uno de los esquemas de las Secciones 11.3 y 11.4 tiene sus propios problemas de indización característicos, y la elección de un algoritmo en particular depende de diversos factores. Los algoritmos que utilizan cómputo en el lugar son eficientes en el uso de memoria, pero presentan dos desventajas

- necesitan memoria de acceso aleatorio en lugar de secuencial;
- alguna de las sucesiones de entrada o de salida están en orden de inversión de bits.

El orden de acceso a los coeficientes será normal o en inversión de bits según el algoritmo sea de decimación en tiempo o de decimación en frecuencia y de si las entradas o las salidas están en orden de inversión de bits. Si no se usa memoria de acceso aleatorio (RAM), existen algunos algoritmos de FFT que utilizan memoria de acceso secuencial, pero en este caso las entradas o las salidas debe estar en orden de inversión de bits. Aunque esos algoritmos se pueden reestructurar de manera que la entrada, la salida y los coeficientes estén en orden normal, la indización resulta muy complicada, y se dobla la cantidad de memoria necesaria. Por tanto, el empleo de esos algoritmos no parece ventajoso.

Los algoritmos de FFT que se utilizan más frecuentemente son los que admiten cómputo en el lugar, como los de las Figs. 11.7, 11.10, 11.16 y 11.18. Si una sucesión se va a transformar sólo una vez, el ordenamiento de inversión de bits se puede realizar tanto sobre la entrada como sobre la salida. Sin embargo, algunos algoritmos necesitan transformar una sucesión, operar con el resultado transformado y calcular la TDF inversa. Por ejemplo, en la implementación de filtros FIR mediante convolución por bloques usando la TDF, se multiplica la TDF $X[k]$ de una sección de la sucesión de entrada por la TDF $H[k]$ de la respuesta impulsiva del filtro, y se calcula después la transformada inversa del resultado para obtener un segmento de la salida del filtro. También, al calcular la función de autocorrelación o la correlación cruzada mediante la TDF se transforma una sucesión, se multiplican las TDF y se obtiene la transformada inversa del producto resultante. Cuando, como en los casos mencionados, se conectan "en cascada" dos transformadas es posible evitar la inversión de bits escogiendo adecuadamente los algoritmos de FFT utilizados. Por ejemplo, al realizar un filtro digital FIR mediante la TDF se puede escoger un algoritmo para la transformada directa que utilice los datos en orden normal y proporcione la TDF en orden de inversión de bits. Se podrían utilizar el esquema de la Fig. 11.10 (decimación en tiempo) o de la Fig. 11.16 (decimación en frecuencia). La diferencia es que la estructura basada en decimación en el tiempo necesita los coeficientes en orden de inversión de bits mientras que la estructura basada en decimación en frecuencia los requiere ordenados normalmente.

Al utilizar cualquiera de los dos algoritmos, la transformada se obtiene en inversión de bits. Por tanto, al utilizar la TDF para realizar convolución por bloques, si la TDF de la respuesta impulsiva del filtro se almacena en orden de inversión de bits, basta multiplicar punto a punto las TDF en el orden en que se han almacenado. Para calcular la TDF inversa, se puede escoger un esquema que requiera los datos de la entrada en orden de inversión de bits y proporcione a la salida los datos en forma normal. Para ello se puede utilizar el diagrama de la Fig. 11.7 (decimación en tiempo) o el de la Fig. 11.18 (decimación en frecuencia). Nuevamente, la diferencia es que el esquema de la Fig. 11.7 utiliza los coeficientes en orden normal mientras que el de la Fig. 11.18 los utiliza en orden de inversión

de bits. Para evitar reordenar los coeficientes, si la transformación directa se efectúa con el algoritmo de decimación en tiempo la transformación inversa debe realizarse con el algoritmo de decimación en frecuencia, pues ambos requieren los coeficientes en orden de inversión de bits. De manera similar, si la transformación directa se calcula con un algoritmo de decimación en frecuencia es conveniente emplear con un algoritmo de decimación en tiempo para la transformación inversa pues ambos utilizan los coeficientes en orden normal.

11.5.2. Coeficientes

Según el algoritmo utilizado, será necesario contar con los coeficientes $W_N^r = e^{-j\frac{2\pi}{N}r}$ en orden normal o en orden inversión de bits. En cada caso, se pueden almacenar una tabla de tamaño suficiente, o bien se pueden calcular a medida que se necesiten. La primera alternativa es más veloz, pero necesita memoria adicional. De los esquemas de cálculo se puede observar que los valores de $W_N^r = e^{-j\frac{2\pi}{N}r}$ que invervienen en las operaciones son los correspondientes a $r = 0, 1, \ldots, (N/2)-1$. Por tanto la cantidad de memoria requerida por la tabla es de $N/2$ lugares (complejos); este número se puede reducir utilizando simetría, a costa de una mayor complejidad de acceso a los valores deseados. Para los algoritmos en los que los coeficientes se acceden en orden de inversión de bits basta almacenar la tabla en ese orden.

Si es necesario economizar memoria, los coeficientes se pueden calcular a medida que se necesiten. Esta alternativa insume más tiempo, y es menos eficiente que la búsqueda en una tabla. Como en una determinada etapa los coeficientes necesarios son potencias de un número complejo de la forma $W_N^q = (e^{-j\frac{2\pi}{N}})^q$ donde q depende del algoritmo y de la etapa, suele ser preferible calcularlos utilizando una fórmula recursiva. Por tanto, si se necesitan en orden normal, para obtener el ℓ-ésimo coeficiente a partir del coeficiente $(\ell-1)$ se puede utilizar la fórmula recursiva

$$e^{-j\frac{2\pi}{N}q\ell} = e^{-j\frac{2\pi}{N}q} e^{-j\frac{2\pi}{N}q(\ell-1)}. \tag{11.25}$$

Si $x[\ell] = \mathrm{Re}\{e^{-j\frac{2\pi}{N}q\ell}\}$, $y[\ell] = \mathrm{Im}\{e^{-j\frac{2\pi}{N}q\ell}\}$, la ecuación anterior puede escribirse como

$$\begin{aligned} x[\ell] &= \cos\left(\tfrac{2\pi}{N}q\right) x[\ell-1] - \mathrm{sen}\left(\tfrac{2\pi}{N}q\right) y[\ell-1], \\ y[\ell] &= \mathrm{sen}\left(\tfrac{2\pi}{N}q\right) x[\ell-1] + \cos\left(\tfrac{2\pi}{N}q\right) y[\ell-1], \end{aligned}$$

que se conoce como *oscilador en cuadratura* (Fig. 11.21). Cuando se utiliza aritmética de precisión finita, la iteración sucesiva de esta ecuación a diferencias puede potenciar los errores, por lo que generalmente es necesario reinicializar la sucesión en puntos preestablecidos (por ejemplo, $W_N^{N/4} = (e^{j\frac{2\pi}{N}})^{N/4} = j$) de manera de mantener acotado el error.

La solución propuesta por (11.25) no es conveniente para los algoritmos que requieren los coeficientes en orden de inversión de bits.

11.6. Algoritmos para valores de N más generales

Cuando N es potencia de 2 la TDF se puede calcular con algoritmos que tienen una estructura simple. Cuando se desea evaluar la TDF de manera eficiente para otros valores de

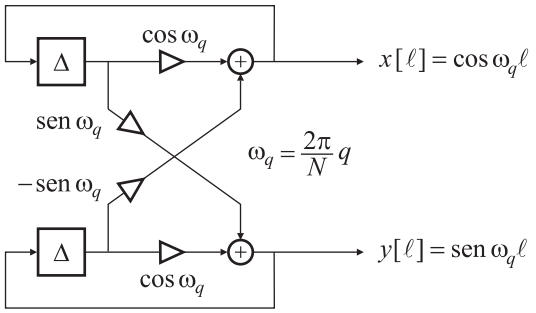


Fig. 11.21. Generación de los coeficientes W_N^q con un oscilador en cuadratura.

N, se pueden aplicar los mismos principios utilizados para los algoritmos de decimación en tiempo y decimación en frecuencia si N es un entero compuesto, es decir, el producto de dos o más factores enteros. Estas estrategias se conocen colectivamente como métodos de "dividir y combinar". Por ejemplo, si $N = LM$ es posible expresar la TDF de N puntos como la suma de L TDF de M puntos o bien como la suma de M TDF de L puntos; si N tiene muchos factores el proceso puede repetirse para cada uno de ellos, dando origen a los algoritmos de *raíces variadas*. En general, la indización es más complicada que en los algoritmos de decimación en tiempo o en frecuencia (con N potencia de 2).

Algunos de estos algoritmos son generalizaciones de los de decimación en tiempo o en frecuencia, y se los denomina tipo "Cooley-Tukey". Si los factores de N son relativamente primos, los algoritmos de *factores primos* permiten reducir aún más el número de operaciones a costa de complicar el esquema de indización. A continuación se desarrollan las principales ideas de estos algoritmos.

11.6.1. Método "dividir y combinar"

El desarrollo de métodos eficientes para calcular la TDF se basa en encontrar una forma de dividir un problema de orden N en problemas de menor dimensión (Burrus, 1977, 1988). En esta idea se basa el método de decimación en tiempo (o el de decimación en frecuencia), que en el caso en que la longitud N de la sucesión sea una potencia de 2, $N = 2^\alpha$, dividen repetidamente la sucesión temporal en dos subsucesiones de longitud mitad.

Si N no es potencia de 2, pero puede factorizarse como $N = LM$ se puede desarrollar un algoritmo más general, que se basa en el hecho que, para N grande,

$$L^2 + M^2 \ll N^2.$$

La idea es entonces dividir la sucesión de entrada en L sucesiones más cortas de largo M, y calcular L TDFs de largo M, y después combinarlas en una TDF de mayor orden a partir de M TDFs de largo L.

La sucesión temporal $x[n]$ puede organizarse como un arreglo unidimensional (donde el índice n toma los valores desde 0 hasta $N-1$), o bien, aprovechando el hecho que N puede factorizarse como el producto LM, como un arreglo *bidimensional* definiendo un par de índices ℓ y m, donde $0 \leq \ell \leq L-1$, y $0 \leq m \leq M-1$. Esta operación de convertir

un único índice n en un par de índices ℓ, m, de manera biunívoca (es decir, que dado n se pueden determinar ℓ, m y viceversa) se denomina *mapeo*. Por ejemplo, si se elige el mapeo

$$n = \ell M + m, \tag{11.26}$$

la sucesión $x[n]$ (que se notará ahora como $x[\ell, m]$) puede pensarse como una matriz, donde ℓ indica el número de fila, y m el número de columna, tal como se muestra a continuación:

	$m{=}0$	$m{=}1$	$\cdots$	$m{=}M{-}1$
$\ell{=}0$	$x[0]$	$x[1]$	$\cdots$	$x[M-1]$
$\ell{=}1$	$x[M]$	$x[M+1]$	$\cdots$	$x[2M-1]$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\ell{=}L{-}1$	$x[(L-1)\,M]$	$x[(L-1)\,M+1]$	$\cdots$	$x[LM-1]$

Esta manera de ordenar el arreglo bidimensional se conoce como *arreglo por filas*, ya que cada fila está formada por M elementos consecutivos del arreglo temporal $x[n]$. Si, en cambio, se define el mapeo

$$n = \ell + mL, \tag{11.27}$$

se obtiene otro ordenamiento, donde ahora cada *columna* del arreglo bidimensional $x[\ell, m]$ está formada por L elementos consecutivos de $x[n]$:

	$m{=}0$	$m{=}1$	$\cdots$	$m{=}M{-}1$
$\ell{=}0$	$x[0]$	$x[L]$	$\cdots$	$x[(M-1)\,L]$
$\ell{=}1$	$x[1]$	$x[L+1]$	$\cdots$	$x[(M-1)\,L+1]$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
$\ell{=}L{-}1$	$x[(L-1)]$	$x[2L-1]$	$\cdots$	$x[LM-1]$

Una distribución similar puede adoptarse para almacenar los valores de la TDF $X[k]$, mapeando el índice k a un par de índices p, q, donde $0 \leq p \leq L-1$, y $0 \leq q \leq M-1$. Tal como se vio para el caso de $x[n]$, si se elige la relación

$$k = pM + q, \tag{11.28}$$

las muestras de la TDF $X[p, q]$ estarán ordenadas "por filas": los M primeros valores de $X[k]$ estarán almacenados en la primera fila de la matriz, los segundos M valores de $X[k]$ en la segunda fila, etc. En cambio, si se elige indizar k según

$$k = p + qL, \tag{11.29}$$

cada conjunto de L valores consecutivos de $X[k]$ estarán almacenados en las columnas del arreglo bidimensional $X[p, q]$.

Para esbozar la idea del algoritmo "dividir y combinar" es conveniente suponer que la sucesión temporal $x[n]$ y el vector de muestras $X[k]$ de la TDF se almacenan como arreglos bidimensionales $x[l, m]$ y $X[p, q]$, respectivamente. La TDF puede expresarse como una

doble suma sobre cada uno de los índices del arreglo bidimensional, multiplicado por los factores de fase $e^{-j\frac{2\pi}{N}mq}$, $e^{-j\frac{2\pi}{N}p\ell}$, etc. Por ejemplo, si se adopta un mapeo tipo *columna* para $x[n]$ y tipo *fila* para $X[k]$, los índices n y k se representarán como

$$\begin{aligned} n &= \ell + mL, \qquad 0 \leq \ell \leq L-1, \quad 0 \leq m \leq M-1, \\ k &= pM + q, \qquad 0 \leq p \leq L-1, \quad 0 \leq q \leq M-1. \end{aligned}$$

Reemplazando en la definición de la TDF,

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \tag{11.30}$$

se tiene que

$$X[k] = \sum_{m=0}^{M-1} \sum_{\ell=0}^{L-1} x[\ell, m] e^{-j\frac{2\pi}{N}k(\ell+mL)},$$

o bien, utilizando la notación bidimensional para $X[k]$,

$$X[p, q] = \sum_{m=0}^{M-1} \sum_{\ell=0}^{L-1} x[\ell, m] e^{-j\frac{2\pi}{N}(pM+q)(\ell+mL)}. \tag{11.31}$$

Teniendo en cuenta que

$$e^{-j\frac{2\pi}{N}(pM+q)(\ell+mL)} = e^{-j\frac{2\pi}{N}pM\ell} e^{-j\frac{2\pi}{N}pmML} e^{-j\frac{2\pi}{N}q\ell} e^{-j\frac{2\pi}{N}qmL}, \tag{11.32}$$

y que

$$\begin{aligned} e^{-j\frac{2\pi}{N}pM\ell} &= e^{-j\frac{2\pi}{N/M}p\ell} = e^{-j\frac{2\pi}{L}p\ell}, \\ e^{-j\frac{2\pi}{N}pmML} &= e^{-j2\pi pm} = 1, \\ e^{-j\frac{2\pi}{N}qmL} &= e^{-j\frac{2\pi}{N/L}qm} = e^{-j\frac{2\pi}{M}qm}, \end{aligned}$$

se puede reescribir (11.31) como

$$\begin{aligned} X[p, q] &= \sum_{\ell=0}^{L-1} \sum_{m=0}^{M-1} x[\ell, m] e^{-j\frac{2\pi}{M}qm} e^{-j\frac{2\pi}{N}q\ell} e^{-j\frac{2\pi}{L}p\ell} \\ &= \sum_{\ell=0}^{L-1} \left[e^{-j\frac{2\pi}{N}q\ell} \left(\sum_{m=0}^{M-1} x[\ell, m] e^{-j\frac{2\pi}{M}qm} \right) \right] e^{-j\frac{2\pi}{L}p\ell}. \end{aligned} \tag{11.33}$$

Analizando (11.33) se observa que

$$X[p, q] = \underbrace{\sum_{\ell=0}^{L-1} \left[\underbrace{e^{-j\frac{2\pi}{N}\ell q}}_{\text{fact. rotación}} \underbrace{\left(\sum_{m=0}^{M-1} x[\ell, m] e^{-j\frac{2\pi}{M}mq} \right)}_{F[\ell,q]\text{: TDF de } M \text{ puntos}} \right] e^{-j\frac{2\pi}{L}\ell p}}_{\text{TDF de } L \text{ puntos}}. \tag{11.34}$$

La expresión (11.34) involucra el cálculo de TDF de largo M y de largo L, y así el cómputo de la TDF se puede efectuar en cinco pasos:

Paso 1: Se ordenan los datos en M columnas de L filas cada una (cada una de las M columnas tiene L datos consecutivos).

Paso 2: Para cada una de las *filas* $\ell = 0, \ldots, L-1$ se calculan las L TDFde M puntos

$$F[\ell, q] = \sum_{m=0}^{M-1} x[\ell, m] e^{-j\frac{2\pi}{M}mq}, \quad 0 \leq q \leq M-1.$$

Paso 3: Se calcula un nuevo arreglo bidimensional $G[\ell, q]$ multiplicando la matriz $F[\ell, q]$ del paso previo por los factores de fase $e^{-j\frac{2\pi}{N}ql}$ ("twiddle factors", factores de rotación)

$$G[\ell, q] = e^{-j\frac{2\pi}{N}\ell q} F[\ell, q] = e^{-j\frac{2\pi}{N}\ell q} \left(\sum_{m=0}^{M-1} x[\ell, m] e^{-j\frac{2\pi}{M}mq} \right),$$

donde $0 \leq \ell \leq L-1$, y $0 \leq q \leq M-1$.

Paso 4: Finalmente, para cada *columna* $q = 0, 1, \ldots M-1$ de la matriz $G[\ell, q]$ se calculan las M TDFde L puntos

$$X[p, q] = \sum_{\ell=0}^{L-1} G[\ell, q]\, e^{-j\frac{2\pi}{L}\ell p}.$$

Paso 5: La TDF $X[k]$ se obtiene leyendo *fila por fila* la matriz $X[p, q]$.

Este procedimiento de cálculo es más complicado que el cálculo directo de la TDF por definición (11.30). Sin embargo, la complejidad *computacional* de este algoritmo, que mide el número de sumas y productos necesarios para obtener el resultado, es menor que la complejidad computacional de la implementación directa. En efecto: el primer paso requiere LM^2 multiplicaciones complejas, y $LM\,(M-1)$ sumas complejas. El paso 2 necesita LM multiplicaciones complejas, y por último, el tercer paso involucra ML^2 productos y $ML\,(L-1)$ sumas complejos. En consecuencia, la complejidad computacional es

$N\,(M+L+1)$	multiplicaciones complejas
$N\,(M+L-2)$	sumas complejas

donde $N = LM$. De esta forma el número de multiplicaciones se ha reducido de N^2 a $N\,(M+L+1)$, y el número de sumas de $N\,(N-1)$ a $N\,(M+L-2)$.

Para fijar ideas, considérese una sucesión de longitud $N = 1000$, de modo que para calcular la TDF de dicha secuencia se adopta $L = 2$, $M = 500$. El cálculo de la TDF utilizando el método directo requiere 10^6 $(\approx N^2)$ multiplicaciones, mientras que con el método "dividir y combinar" sólo son necesarias 503000 operaciones: aproximadamente la mitad. El número de sumas también se reduce al 50 %, aproximadamente. Si en cambio se eligen $L = 20$, $M = 50$, el número de operaciones desciende a 71000. Esto revela que los factores L y M deben elegirse con cuidado.

EJEMPLO **11.1. Cálculo de la TDF de $N = 15$ puntos**

Como $N = 5 \times 3 = 15$, se elige $L = 5$, $M = 3$.

Paso 1: La sucesión temporal $x[n]$ se representa como un arreglo bidimensional compuesto por tres columnas de 5 filas cada una:

	Columna 1:	Columna 2:	Columna 3:
Fila 1:	$x[0,0] = x[0]$	$x[0,1] = x[5]$	$x[0,2] = x[10]$
Fila 2:	$x[1,0] = x[1]$	$x[1,1] = x[6]$	$x[1,2] = x[11]$
Fila 3:	$x[2,0] = x[2]$	$x[2,1] = x[7]$	$x[2,2] = x[12]$
Fila 4:	$x[3,0] = x[3]$	$x[3,1] = x[8]$	$x[3,2] = x[12]$
Fila 5:	$x[4,0] = x[4]$	$x[4,1] = x[9]$	$x[4,2] = x[14]$

Paso2: Se calculan las TDFs de 3 puntos para cada una de las 5 filas, que resulta en el siguiente arreglo de 5 filas por 3 columnas

$$\begin{array}{lll} F[0,0] & F[0,1] & F[0,2] \\ F[1,0] & F[1,1] & F[1,2] \\ F[2,0] & F[2,1] & F[2,2] \\ F[3,0] & F[3,1] & F[3,2] \\ F[4,0] & F[4,1] & F[4,2] \end{array}$$

Paso 3: Se multiplican cada uno de los términos $F[\ell, q]$ por los factores $e^{-j\frac{2\pi}{N}\ell q} = e^{-j\frac{2\pi}{15}\ell q}$, con $0 \leq \ell \leq 4$ y $0 \leq q \leq 2$, obteniéndose el arreglo de dimensión 5×3

Columna 1	Columna 2	Columna 3
$G[0,0]$	$G[0,1]$	$G[0,2]$
$G[1,0]$	$G[1,1]$	$G[1,2]$
$G[2,0]$	$G[2,1]$	$G[2,2]$
$G[3,0]$	$G[3,1]$	$G[3,2]$
$G[4,0]$	$G[4,1]$	$G[4,2]$

Paso 4: Se calculan 3 TDFs de 5 puntos para cada una de las columnas de la matriz G resultando los valores de $X[k]$ ordenados de la siguiente manera:

$$\begin{array}{lll} X[0,0] = X[0] & X[0,1] = X[1] & X[0,2] = X[2] \\ X[1,0] = X[3] & X[1,1] = X[4] & X[1,2] = X[5] \\ X[2,0] = X[6] & X[2,1] = X[7] & X[2,2] = X[8] \\ X[3,0] = X[9] & X[3,1] = X[10] & X[3,2] = X[11] \\ X[4,0] = X[12] & X[4,1] = X[13] & X[4,2] = X[14] \end{array}$$

Paso 5: La TDF $X[k]$ de la señal original se obtiene leyendo fila por fila la matriz $X[p,q]$. La Fig. 11.22 ilustra los pasos necesarios para el cálculo. □

Es interesante estudiar el orden de las sucesiones de datos $x[n]$ y la TDF $X[k]$ cuando se los considera como arreglos unidimensionales. Si la sucesión $x[n]$ y la TDF $X[k]$ se leen por filas, se obtienen las siguientes sucesiones:

arreglo de entrada

$$x[0]\ x[5]\ x[10]\ x[1]\ x[6]\ x[11]\ x[2]\ x[7]\ x[12]\ x[3]\ x[8]\ x[13]\ x[4]\ x[9]\ x[14]$$

arreglo de salida

$$X[0]\ X[1]\ X[2]\ X[3]\ X[4]\ X[5]\ X[6]\ X[7]\ X[8]\ X[9]\ X[10]\ X[11]\ X[12]\ X[13]\ X[14]$$

La sucesión de entrada queda permutada, mientras que la salida conserva el orden natural. En este caso, la permutación de la sucesión de entrada se debe a la forma en que construyó el arreglo bidimensional, y al orden en que se calcularon las TDFs.

En resumen, el algoritmo que se acaba de describir involucra los siguientes cálculos:

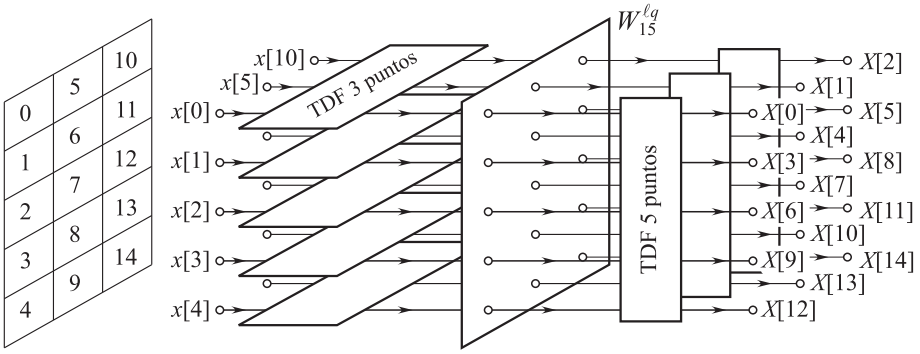


Fig. 11.22. Cálculo de una TDF de 15 puntos usando TDFs de 3 y 5 puntos.

11.6.1.1. Algoritmo 1

1. Almacenar la sucesión temporal $x[n]$ en M columnas de L filas cada una.
2. Calcular la TDF de M puntos para cada una de las L filas.
3. Multiplicar el arreglo resultante por los factores $e^{-j\frac{2\pi}{N}\ell q}$.
4. Calcular la TDF de L puntos de cada una de las M columnas.
5. Leer el arreglo resultante fila por fila.

Un algoritmo dual con la misma estructura, puede obtenerse si la señal de entrada se almacena por filas, y el resultado final se lee columna por columna. Eligiendo $n = \ell M + m$, $k = p + qL$, se tiene que

$$\begin{aligned} X[p,q] &= \sum_{m=0}^{M-1}\sum_{\ell=0}^{L-1} x[\ell,m]e^{-j\frac{2\pi}{L}p\ell}e^{-j\frac{2\pi}{N}pm}e^{-j\frac{2\pi}{M}qm} \\ &= \sum_{m=0}^{M-1}\left[e^{-j\frac{2\pi}{N}pm}\left(\sum_{\ell=0}^{L-1} x[l,m]e^{-j\frac{2\pi}{L}\ell p}\right)\right]e^{-j\frac{2\pi}{M}mq}, \end{aligned}$$

que puede verse como

$$X[p,q] = \underbrace{\sum_{m=0}^{M-1}\left[\underbrace{e^{-j\frac{2\pi}{N}pm}}_{\text{fact. rotación}}\underbrace{\left(\sum_{\ell=0}^{L-1} x[\ell,m]e^{-j\frac{2\pi}{L}\ell p}\right)}_{F[p,m]\text{: TDF de } L \text{ puntos}}\right]e^{-j\frac{2\pi}{M}mq}}_{\text{TDF de } M \text{ puntos}}, \tag{11.35}$$

a partir de la cual puede escribirse un segundo algoritmo:

11.6.1.2. Algoritmo 2

1. Almacenar la sucesión temporal $x[n]$ en L filas de M columnas cada una.
2. Calcular M TDFs de L puntos para cada una de las columnas.
3. Multiplicar el arreglo resultante por los factores $e^{-j\frac{2\pi}{N}pm}$.
4. Calcular la TDF de M puntos de cada una de las L filas.
5. Leer el arreglo resultante columna por columna.

Cada uno de los algoritmos descriptos tiene la misma complejidad computacional; la única diferencia entre ambos es la manera de ordenar los datos y la forma de calcular las TDF de menor dimensión.

Los algoritmos basados en mapear los índices de la TDF según las ecuaciones (11.26) o (11.27) para $x[n]$ y (11.28) o (11.29) para $X[k]$ involucran el empleo de los factores de rotación ("twiddle factors") como un paso intermedio en el cálculo de las TDF de tamaño menor. El artículo original de Cooley y Tukey (1965) presentaba una técnica muy similar a la desarrollada aquí, de modo que todos los algoritmos de esta clase se conocen, en general, como algoritmos de Cooley-Tukey.

Si N es un número altamente compuesto, es decir que N puede factorizarse como un producto de números primos de la forma

$$N = r_1 r_2 \cdots r_v,$$

entonces la descomposición descrita puede repetirse v veces. Esto conduce a TDFs de menor dimensión, y por ende, a un algoritmo más eficiente, que se denomina *de raíces variadas*.

La primera segmentación de la sucesión $x[n]$ en un arreglo bidimensional de M columnas de L elementos cada una resulta en TDFs de largo L y M. La descomposición de los datos en subsucesiones más cortas implica segmentar cada fila o columna en arreglos bidimensionales que resultan en TDFs de menor longitud. El procedimiento se agota cuando N queda factorizado en sus factores primos.

En el caso particular que N sea una potencia entera de un número r, es decir $N = r^v$, se pueden obtener implementaciones altamente eficientes, que se conocen como *algoritmos de raíz r*. Si $r = 2$, y si se eligen $M = N/2$ columnas de $L = 2$ filas (almacenamiento por filas), la aplicación del Algoritmo 1 parte la sucesión $x[n]$ original en dos sucesiones $f_1[n]$ y $f_2[n]$ (las filas de la matriz) que corresponden a las muestras pares e impares de $x[n]$:

$$\left.\begin{array}{l} f_1[n] = x[2n], \\ f_2[n] = x[2n+1], \end{array}\right\} \quad n = 0, 1, \ldots, N/2 - 1.$$

Esta partición coincide con el primer paso del procedimiento de *decimación en tiempo* descrito en la Sección 11.3. La descomposición sucesiva de cada una de las L filas repitiendo el Algoritmo 1 requiere en total $v = \log_2 N$ pasos, y permite reproducir el esquema de cálculo que se muestra en la Fig. 11.7. El método de *decimación en frecuencia* de la Sección 11.4 se obtiene al aplicar el Algoritmo 2 con $M = 2$ columnas de $L = N/2$ filas cada una (almacenamiento por columnas). La descomposición sucesiva de cada TDF en TDF

de menor dimensión conduce a un mapeo multidimensional de los índices de la TDF, por ejemplo

$$\begin{aligned} n &= 2^{v-1}n_1 + \cdots + 2n_{v-1} + n_v, \\ k &= k_1 + 2k_2 + \cdots + 2^{v-1}k_v. \end{aligned} \tag{11.36}$$

Como los índices n_i y k_i son 0 o 1, la ecuación (11.36) revela que si la entrada tiene orden natural (índice n), la salida (índice k) queda permutada (*bit reversal*), y viceversa.

Para comparar la eficacia de los algoritmos de cálculo de la TDF es conveniente contar con una ecuación que reporte el número total de multiplicaciones $\mu(N)$ necesarias para el cálculo de la TDF de N puntos. Para el caso de $N = 2^v$, es sencillo determinar el número total de multiplicaciones. La evaluación de cada una de las transformaciones de fila necesita $L\mu(M)$ productos, los factores de rotación agregan $LM = N$ multiplicaciones, y el conjunto final de transformaciones por columna requiere $M\mu(L)$ productos. En definitiva, el número total es

$$\mu(N) = L\mu(M) + LM + M\mu(L) = N\left(\frac{\mu(M)}{M} + \frac{\mu(L)}{L} + 1\right). \tag{11.37}$$

Si N tiene v factores, $N = N_1 N_2 \cdots N_v$, aplicando reiteradamente la ecuación (11.37) se puede mostrar que el número total de multiplicaciones necesario para calcular la TDF por descomposición sucesiva en transformaciones de dimensión menor es

$$\mu(N) = N\left[\sum_{i=1}^{v} \frac{\mu(N_i)}{N_i} + (v-1)\right]. \tag{11.38}$$

Esta ecuación muestra la forma en la que el mapeo de índices puede ayudar a reducir el volumen de cálculo, aunque hay muchas variantes. Si las transformaciones de menor número de puntos se calculan por el método directo, cada una de ellas necesita N_i^2 multiplicaciones. Con esta hipótesis, Cooley y Tukey mostraron que el menor número de operaciones se logra si $N = 3^v$. Sin embargo, algunas transformaciones de bajo orden tampoco necesitan N_i^2 operaciones: por ejemplo, las transformadas de $N = 2$ no requieren *ningún* producto. La ecuación (11.38) da el resultado correcto para el caso $N = 2^v$ si se recuerda que las transformaciones de $N_i = 2$ no necesitan ninguna multiplicación ($\mu(N_i) = 0$), y que en el producto en la última etapa del algoritmo de decimación en frecuencia interviene el factor $W_N^0 = e^{j\frac{2\pi}{N}0} = 1$. Entonces, el número total de productos es $\mu(N) = N(v-1)$, todos los cuales se deben a los factores de rotación (*twiddle factors*).

De la discusión precedente, se concluye que una manera de reducir los cálculos es diseñando algoritmos eficientes para TDF de N_i puntos (con N_i pequeño) que requieran menos de N_i^2 operaciones. Un ejemplo interesante es la TDF de $N_i = 4$ puntos, ya que en este caso los factores de rotación son $+1$, j, -1, $-j$, y no implican multiplicación alguna (el "multiplicar" por j significa intercambiar la parte real con la parte imaginaria, cambiando el signo de esta última). La Fig. 11.23 muestra el gráfico ("mariposa") que permite calcular la TDF de $N = 4$. En el caso en que $N = 4^v$, el volumen de cálculo es menor si se utilizan v etapas de TDF de orden 4 en lugar de usar $2v$ etapas de orden 2, pues en el primer caso el número de etapas es la mitad que en el segundo, reduciendo también a la mitad el número de multiplicaciones por los factores de rotación.

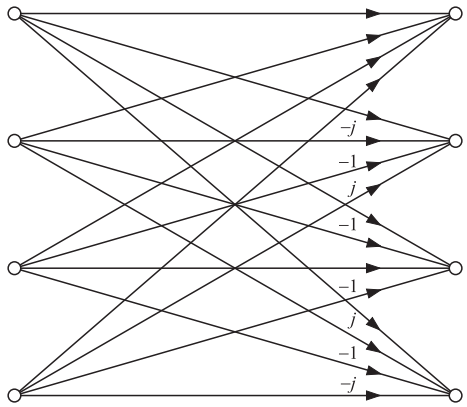


Fig. 11.23. "Mariposa" para una TDF de $N = 4$.

11.6.2. Algoritmos de factores primos

De las secciones anteriores se desprende que desde el punto de vista del cálculo es más conveniente efectuar varias TDF de menor tamaño que una única de mayor orden. Ciertas longitudes son ventajosas pues no necesitan multiplicaciones; también es beneficioso reducir el número de multiplicaciones por los factores de rotación. En algunos casos es posible eliminar completamente la multiplicación por estos factores. Para ello es necesario definir los mapeos de los índices de los arreglos de la forma

$$n = ((\ell A + mB))_N, \quad \begin{cases} 0 \leq \ell \leq L-1, \\ 0 \leq m \leq M-1, \end{cases}$$

$$k = ((pC + qD))_N, \quad \begin{cases} 0 \leq p \leq L-1, \\ 0 \leq q \leq M-1, \end{cases}$$

donde $((\cdot))_N$ indica la operación "módulo N". Si se elige $A = 1$, $B = L$, $C = M$ y $D = 1$ se tienen los mapeos del Algoritmo 1, y con $A = M$, $B = 1$, $C = 1$ y $D = L$, los mapeos del Algoritmo 2. En ambos casos, los mapeos nunca exceden el valor N, y por lo tanto la operación módulo es innecesaria. Estos mapas producen la descomposición de Cooley y Tukey para cualquier valor de L y M. Sin embargo, si L y M son relativamente primos (es decir que no tienen factores en común), una elección diferente de las constantes A, B, C, D permite eliminar completamente los factores de rotación. Específicamente, se deben elegir A, B, C, D de manera que todos los valores de n y k entre 0 y $N-1$ queden representados una única vez, y de forma tal que los factores de rotación (los términos $e^{-j\frac{2\pi}{N}q\ell}$) desaparezcan, es decir

$$e^{-j\frac{2\pi}{N}(A\ell+Bm)}e^{-j\frac{2\pi}{N}(Cp+Dq)} = e^{-j\frac{2\pi}{N}\ell p}e^{-j\frac{2\pi}{N}mq}$$

[c.f. (11.32)], es decir

$$((AC))_N = M, \qquad ((BD))_N = L, \qquad ((AD))_N = ((BC))_N = 0.$$

El cálculo de un conjunto de coeficientes que satisfaga estos requerimientos se basa en el *teorema chino del resto*. La justificación utiliza conceptos muy avanzados de teoría de

números, y no serán tratados aquí. Un conjunto de coeficientes que satisface la condición de unicidad y elimina los factores de rotación es (Burrus 1977, 1988)

$$\begin{aligned} A &= M \qquad \text{y} \qquad B = L, \\ C &= M((M^{-1}))_L \qquad \text{y} \qquad D = M((M^{-1}))_L, \end{aligned}$$

donde $((M^{-1}))_L$ indica la inversa multiplicativa de M reducida módulo L: por ejemplo, si $L = 4$ y $M = 3$, $((3^{-1}))_4 = 3$ pues $((3 \cdot 3))_4 = 1$. Con esta elección (que no es única) la TDF puede expresarse como

$$X[pM((M^{-1}))_L + qM((M^{-1}))_L] = \sum_{m=0}^{M-1} \left[\sum_{\ell=0}^{L-1} x[\ell M + mL] e^{-j\frac{2\pi}{L}\ell p} \right] e^{-j\frac{2\pi}{N}mq}, \qquad (11.39)$$

donde $0 \leq p \leq L-1$ y $0 \leq q \leq M-1$. De esta forma, se puede expresar la TDF unidimensional como una TDF bidimensional sin la intervención de los factores de rotación. Como es necesario que L y M sean primos entre sí, los algoritmos de esta clase de denominan *algoritmos de factores primos* (Rader, 1968). Al eliminarse los factores de rotación, el cálculo de las transformadas por fila o por columna puede hacerse en cualquier orden [es equivalente a intercambiar el orden de las sumatorias de la ecuación (11.39)], y el número total de multiplicaciones está dado por (11.38) eliminado el término $(v-1)$. De modo que, si se utiliza el mismo algoritmo para calcular las TDF de longitud N_i, la ecuación (11.39) necesita $N(v-1)$ multiplicaciones menos que la ecuación (11.34) o (11.35).

La eliminación de los factores de rotación que permite el algoritmo de factores primos se logra a expensas de una mayor complejidad de indización y de programación del algoritmo. Al contrario que lo que ocurre con los algoritmos de Cooley y Tukey de raíz 2 o 4, que utilizan una única "mariposa" (sin multiplicaciones) y tienen una estructura altamente anidada, el algoritmo de factores primos necesita una "mariposa" diferente para cada factor, y se programa más fácilmente tomando los factores uno a uno. El valor de este algoritmo se incrementa si se dispone de algoritmos altamente eficientes para el cálculo de las TDF de pequeño tamaño necesarias para cada una de los factores relativamente primos.

11.7. El comando `fft` de MATLAB

MATLAB provee una función o comando `fft` que permite calcular la TDF de un vector `x`. Invocado con la sentencia `X = fft(x,N)` calcula la TDF de N puntos de `x`, agregando ceros en caso de ser necesario. Si se omite el argumento N, el tamaño de la TDF es el largo del vector `x`. Si x es una matriz, entonces `fft(x,N)` calcula la TDF de N puntos de cada una de las columnas de `x`.

Esta función está codificada en lenguaje máquina, y no utilizando comandos MATLAB (no existe un archivo "fft.m") de modo que se ejecuta muy velozmente. La implementación se basa en una librería FFTW (http://www.fftw.org; Frigo y Johnson, 1998). Para calcular una TDF de orden N, la librería descompone el problema utilizando el algoritmo de Cooley-Tukey si N es compuesto (es decir, $N = N_1 N_2$) que calcula primero N_1 transformadas de largo N_2, y luego N_2 transformadas de N_1 puntos. La descomposición se aplica recursivamente hasta que el problema se puede resolver utilizando algunas transformadas

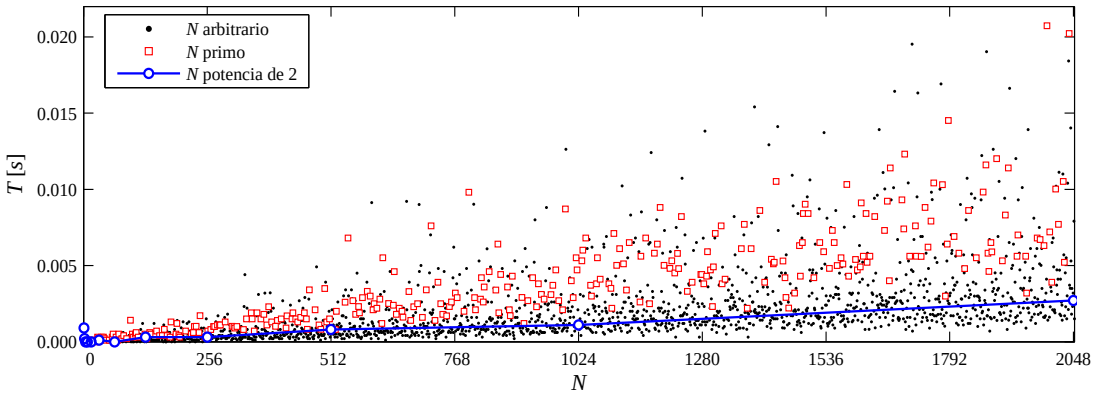


Fig. 11.24. Tiempo de cálculo de FFTs de distintas longitudes en una Pentium III de 750 MHz. Los valores de tiempo correspondientes a N primos se indican con cuadrados; los correspondientes a N potencia de 2 con círculos.

de orden predeterminado codificadas en lenguaje máquina. Estas transformadas, a su vez, usan varias combinaciones de algoritmos, entre ellos una variación del de Cooley y Tukey, un algoritmo de factores primos, y un algoritmo de raíces variadas (*mixed-radix*). La factorización se elige heurísticamente. Si N es un número primo, la librería FFTW utiliza el algoritmo de Rader (1968) para descomponer el problema de N puntos en tres problemas de $(N-1)$ puntos, los que se resuelven utilizando la descomposición de Cooley y Tukey descrita más arriba. En el caso en que $x[n]$ es real, el número de operaciones es aproximadamente la mitad que si $x[n]$ es complejo, para casi cualquier N. Sin embargo, si N tiene factores primos grandes no hay diferencia significativa de velocidad. El tiempo de ejecución del comando `fft` depende del número de puntos de la transformada. Es más veloz cuando N es potencia de 2. La velocidad es comparable cuando N tiene factores primos pequeños, y es considerablemente más lenta para longitudes que son primas, o que tienen factores primos grandes. La transformada inversa se calcula usando la función `ifft`, que tiene las mismas características que `fft`.

El comportamiento del algoritmo puede analizarse calculando la TDF de señales de distinta longitud, y evaluando el tiempo y/o el número de operaciones que demanda calcular cada una de ellas. El análisis de los tiempos de ejecución y del número de operaciones revela la naturaleza "dividir y combinar" de la implementación.

Para determinar los tiempos de ejecución, MATLAB provee dos funciones. Una de ellas es el comando `clock`, que da una lectura instantánea del reloj interno, y la otra es la función `etime(t1,t2)` que calcula el lapso de tiempo transcurrido entre dos marcas temporales `t1` y `t2`. Para evaluar el tiempo de cálculo, se generan vectores aleatorios de longitud 1 hasta 2048, se calculan sus FFTs, y se registra el tiempo de cálculo en un arreglo. Finalmente, se grafica el tiempo de cálculo en función de N.

Un cálculo similar puede llevarse a cabo para el número de operaciones en punto flotante ("flops") que demanda el cómputo de la FFT[7].

El gráfico de los tiempos de ejecución se muestra en la Fig. 11.24, y el del número de operaciones en la Fig. 11.25. Ambas gráficas son muy informativas, pues los puntos no

[7]La función `flops` no está disponible en MATLAB versión 6 y superior.

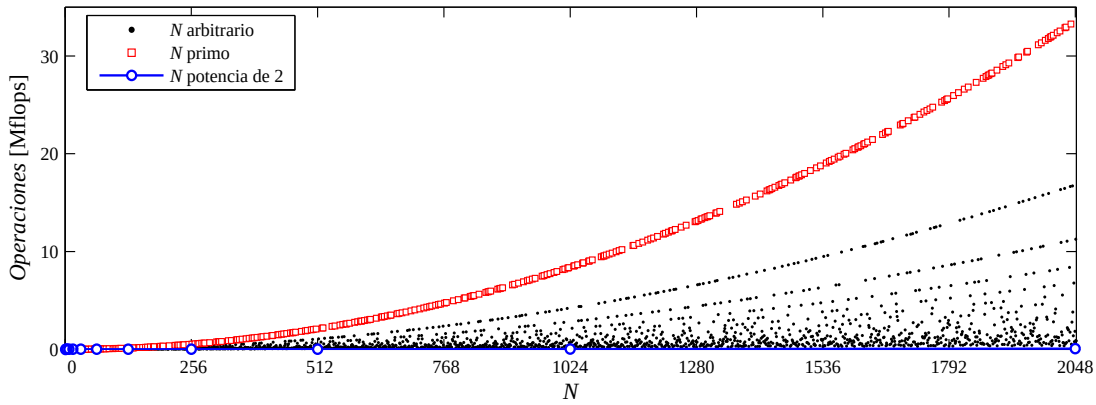


Fig. 11.25. Número de operaciones necesarias para calcular la FFT de N puntos. El número de operaciones correspondientes a N primos se indican con cuadrados; los correspondientes a N potencia de 2 con círculos.

revelan una única función, sino que parecen agruparse en bandas mas o menos definidas. En ambos casos, el conjunto superior de puntos muestra una dependencia proporcional a N^2, es decir que los valores de N que arrojan estos resultados deben ser números primos, para los cuales `fft` calcula la TDF por el método directo (esto es más evidente en la Fig. 11.25). Hay varios grupos que corresponden a bandas proporcionales a $N^2/2$, $N^2/3$, $N^2/4$, etc., dependencias que corresponden a casos para los cuales N tiene pocas descomposiciones (no es "muy divisible"). Finalmente, la última banda (casi indistinguible del eje de las abscisas) corresponde a una dependencia del orden de $N \log_2 N$, para los valores de N que son potencia de 2. Como se dijo anteriormente, para estos valores de N se utiliza la implementación altamente eficiente de raíz 2, mientras que para el resto de los casos se calcula la FFT de raíces variadas. Estas figuras muestran que la estrategia "dividir y combinar" es muy eficiente cuando N es altamente compuesto. Por ejemplo, para $N = 2048$, el tiempo de ejecución es de 0.565 milisegundos, para $N = 2047 = 23 \times 89$ es de 3.015 milisegundos, para $N = 2038 = 2 \times 1019$ es de 7.685 milisegundos, y para $N = 2039$ (primo) es de 15.295 milisegundos. A su vez, el número de operaciones en cada caso es de 67k, 1.84 M, 16.6 M y 33.3 M, respectivamente. La relación entre el número de operaciones necesarias para calcular la FFT de 2039 muestras es casi 497 veces mayor que las necesarias para calcular la FFT de 2048 puntos!

11.7.0.1. La FFTW

Desde la versión 7 en adelante, la rutina de cálculo de la FFT en Matlab ha sido reemplazada por el paquete FFTW (http://www.fftw.org), una librería gratuita independiente del hardware cuyo desempeño es competitivo aún con programas optimizados para una arquitectura en particular. La FFTW tiene dos etapas:

- **Planificación**, donde se adaptan los algoritmos al hardware en el que se corre el algoritmo. En esta etapa el planificador mide el tiempo de ejecución de diferentes estrategias, y selecciona la más veloz, aplicando una serie de reglas para descomponer el problema en subproblemas más simples del mismo tipo. Los problemas "suficientemente simples" se resuelven por código optimizado, denominados `codelets`,

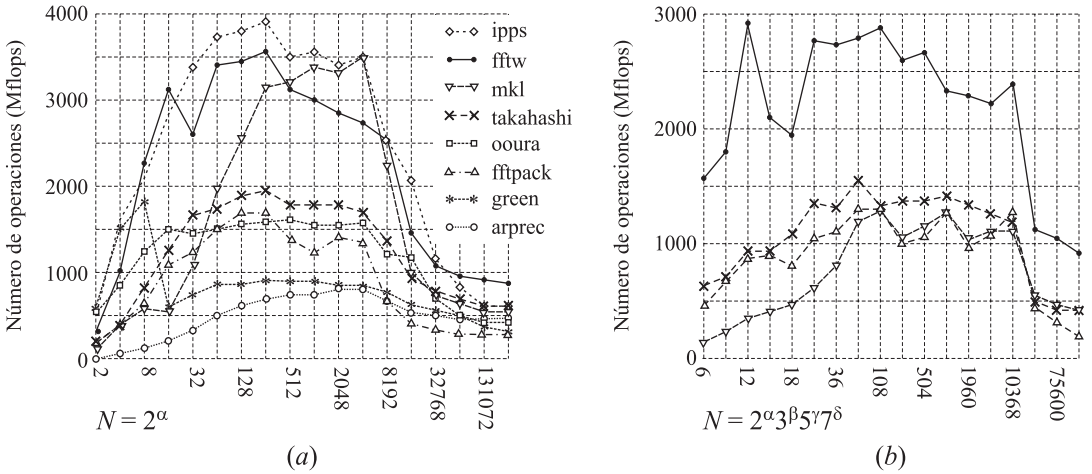


Fig. 11.26. Comparación de TDF complejas unidimensionales de datos en doble precisión para longitud de los datos $N = 2^\alpha$ (a) y $N = 2^\alpha 3^\beta 5^\gamma 7^\delta$ (b) en una Pentium IV de 2.8-GHz.

generados por el compilador especializado `genfft`. Debido a las diferentes pruebas que ejecuta, esta etapa suele demandar una gran cantidad de tiempo.

- **Ejecución**, en la cual la FFTW realiza la tarea útil para el usuario.

Para calcular una TDF, el usuario invoca al planificador, especificando el problema a resolver. El problema es una estructura de datos que describe el aspecto de los datos de entrada –tamaño de los arreglos y disposición de la memoria– pero no contiene los datos propiamente dichos. El planificador devuelve una *estrategia*, una estructura de datos ejecutable que acepta los datos de entrada y calcula la TDF. De ahí en más el usuario puede reutilizar la estrategia tantas veces como desee.

La FFTW es rápida y flexible:

- Está escrita en C portable, y se puede ejecutar en muchas arquitecturas y sistemas operativos.
- Calcula la TDF en un tiempo $\mathcal{O}(N \log N)$ para *cualquier* longitud N (la mayoría de las implementaciones de la TDF demandan este tiempo sólo para datos de determinada longitud, y el tiempo suele crecer a $\mathcal{O}(N^2)$ si N es primo).
- No impone restricciones sobre la dimensión de las transformadas, mientras que la amplia mayoría de las implementaciones se especializa para el caso unidimensional y más raramente para transformaciones bi- y tri- dimensionales.
- Soporta el cálculo simultáneo de múltiples TDF; la mayoría de las implementaciones están diseñadas para el cálculo de una única TDF de datos contiguos.

La Fig. 11.26 muestra el desempeño de la FFTW respecto a otros algoritmos, medido en "Mflops" que en este caso se definen como $5N \log_2 N/t$ para una TDF de largo N, y t es el tiempo en microsegundos (sin incluir el tiempo que requiere la inicialización). En la Fig. 11.26(a) se grafica el desempeño para el caso en que el tamaño N de la TDF es

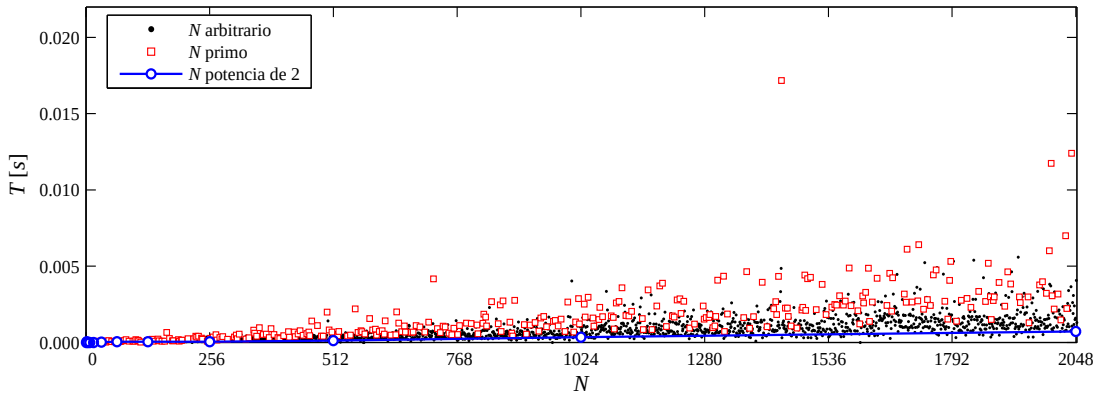


Fig. 11.27. Tiempo de cálculo de FFTW de distintas longitudes en una Pentium III de 750 MHz. Los valores de tiempo correspondientes a N primos se indican con cuadrados; los correspondientes a N potencia de 2 con círculos.

potencia de 2, mientras que en la Fig. 11.26(b) se ilustran los resultados en caso que N es un número compuesto, $N = 2^\alpha 3^\beta 5^\gamma 7^\delta$ en una computadora Pentium IV de 2.8-GHz. Los diferentes algoritmos estudiados son:

sigla	algoritmo
`ipps`	Intel Integrated Performance Primitives (Signal Processing), v. 3.0.
`mkl`	Intel Math Kernel Library v. 6.1, interface DFTI.
`takahashi`	librería FFTE v. 3.2 en FORTRAN de D. Takahashi.
`ooura`	librería FORTRAN y C de T. Ooura (2001).
`fftpack`	librería FORTRAN de P. N. Swarztrauber (1982).
`green`	código gratuito en C de J. Green (1998).
`arprec`	implementación en 4 etapas en C++ (2002) de D. Bailey (1990).

La sigla `simd` indica las instrucciones únicas que permiten procesar datos múltiples (típicamente vectores de 2 a 4 elementos) en los procesadores Pentium MMX; solamente las librerías FFTW, `ipps` y `mkl` (de Intel) y `takahashi` han sido específicamente diseñadas para explotar estas instrucciones especiales.

El desempeño de todas las rutinas decae a medida que aumenta N, el tamaño de la TDF, reflejando el tamaño de la memoria cache de la máquina. El desempeño también es pobre para valores pequeños de N, debido a la carga de llamar una rutina para efectuar poco trabajo. La librería FFTW es la única que explota las instrucciones `simd` para tamaños que no sean potencia de 2, lo que le da una clara ventaja. La librería `ipps` está limitada para datos contiguos y cálculo en el lugar, mientras que las librerías FFTW y `mkl` soportan datos dispersos.

La Fig. 11.27 muestra los tiempos de cómputo para FFTW de diferentes longitudes en una computadora Pentium III de 750 MHz; los resultados obtenidos no incluyen el tiempo de inicialización (etapa de planificación), que para algunas longitudes particulares puede alcanzar a varios segundos. Al comparar esta figura con la Fig. 11.24 de los tiempos de

cómputo de la FFT de la versión 5 de Matlab se nota que la FFTW es ligeramente más veloz, y una relativa independencia del tiempo de cálculo respecto a la naturaleza de la longitud de los datos; en particular, el conjunto de valores parece agruparse más hacia la cota impuesta por las longitudes que son potencia de 2. La comparación entre los tiempos de ejecución de FFT y de FFTW se resumen en la tabla siguiente.

N	FFT	FFTW	Observaciones
2038	7,685	3,646	$N = 2 \times 1019$
2039	15,295	12,402	N es primo
2047	3,015	1,584	$N = 23 \times 89$
2048	0,565	0,720	$N = 2^{11}$

11.7.1. Cálculo de la convolución usando la FFT

En MATLAB, la función `conv` que calcula la convolución entre dos vectores está basada en la función `filter`, la cual está codificada en lenguaje máquina y por lo tanto es muy eficiente, al menos para valores pequeños de N. Para valores de N grandes es posible acelerar el cálculo de la convolución usando el algoritmo de la FFT. Esta aproximación al cálculo de la convolución utiliza la convolución circular para computar la convolución lineal. El algoritmo resultante se denomina frecuentemente *algoritmo de convolución rápida*. Además, si se elige que N sea potencia de 2, y la FFT se implementa usando un algoritmo de raíz 2, el algoritmo se denomina *convolución de alta velocidad*. Si $x_1[n]$ y $x_2[n]$ son las sucesiones a convolucionar, de longitud N_1 y N_2, respectivamente, para el cálculo de la convolución de alta velocidad se elige N como

$$N = 2^{\lceil \log_2(N_1+N_2-1) \rceil}$$

donde $\lceil \cdot \rceil$ es la función *techo*; $\lceil x \rceil$ es el menor entero mayor que x. La convolución lineal $x_1[n] * x_2[n]$ se puede implementar calculando un par de FFT de dimensión N, una TDF inversa (IFFT) de longitud N, y N multiplicaciones complejas:

$$x_1[n] * x_2[n] = IFFT\left[FFT\left(x_1[n]\right) \times FFT\left(x_2[n]\right)\right].$$

Para valores de N grandes, esta operación es mucho más veloz que el cálculo directo de la convolución. El valor de N para el cual conviene una u otra implementación depende tanto del software como de la plataforma de hardware. Para comparar la velocidad de cálculo de ambos algoritmos implementados en MATLAB se generan dos sucesiones aleatorias de L puntos $x_1[n]$ y $x_2[n]$. La sucesión $x_1[n]$ tiene distribución uniforme en el intervalo $[0, 1]$, y $x_2[n]$ tiene distribución normal de media nula y varianza unitaria. Se determinan los tiempos de ejecución promedio para sucesiones de longitud L comprendidas entre 0 y 1000 muestras, donde el promedio se calcula sobre 1000 realizaciones de las sucesiones aleatorias. Este promedio es necesario no sólo para compensar por la distinta cantidad de ceros que puedan tener las sucesiones, sino también para compensar los errores de medición de tiempo debido a la naturaleza del sistema operativo (el tiempo que demanda cada rutina puede variar según la carga del sistema). El archivo MATLAB para efectuar esta comparación se lista a continuación.

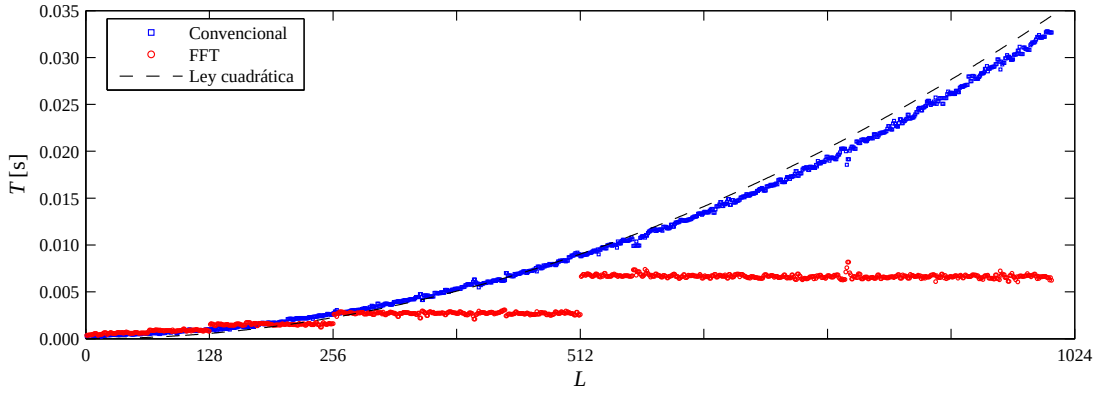


Fig. 11.28. Comparación de los tiempos de cálculo de la convolución y la convolución de alta velocidad para sucesiones de distinta longitud.

```
% Comparación de los tiempos de calculo de la
% convolución normal vs. la convolución rápida
%
Largo = 1000;
Prome = 1000;
conv_time = zeros(1,Largo); fft_time = zeros(1,Largo);
for L = 1:Largo
  tc = 0; tf=0;
  N = 2*L-1; nu = ceil(log10(N)/log10(2)); N = 2^nu;
  for I = 1:Prome
    h = randn(1,L);
    x = rand(1,L);
    t0 = clock; y1 = conv(h,x); t1=etime(clock,t0);
    tc = tc+t1;
    t0 = clock; y2 = ifft(fft(h,N).*fft(x,N)); t2=etime(clock,t0);
    tf = tf+t2;
  end
  conv_time(L) = tc/Prome;
  fft_time(L) = tf/Prome;
end
%
% Grafica los resultados
n = 1:Largo;
plot(n, conv_time, 'bs', n, fft_time, 'ro');
xlabel('{\itL}');
ylabel('{\itT} [{\its}]');
legend('Convencional','FFT',2)
title('Tiempos de cálculo de la convolución');
```

La Fig. 11.28 muestra los tiempos de ejecución de ambos tipos convoluciones, para sucesiones de longitudes comprendidas entre 1 y 1000. Estos tiempos de cálculo dependen fuertemente de la computadora donde se ejecute el programa MATLAB; las curvas de la Fig. 11.28 se obtuvieron en una Pentium III de 750 MHz. Esta figura permite apreciar que para valores pequeños de L la convolución lineal es más rápida. El punto de quiebre se ubica alrededor de las 150 muestras. Para longitudes mayores, los tiempos de ejecución de la convolución lineal crecen cuadráticamente, mientras que el tiempo de cálculo de la convolución de alta velocidad varía de manera mas o menos lineal. Como la convolución de alta velocidad se calcula para $N = 2^v$, el tiempo de cálculo queda constante para un rango de valores de L: calcular la convolución de sucesiones de largo $L = 513$ demanda aproximadamente el mismo tiempo convolucionar dos sucesiones de largo 1023, pues ambas usan una TDF de orden $N = 2048$.

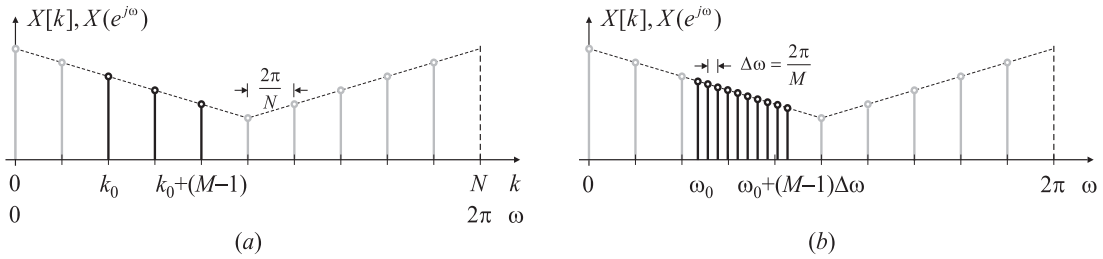


Fig. 11.29. TDF de bandas de frecuencias: (a) transformada zoom; (b) transformada chirp.

11.8. Cálculo de la TDF por convolución

Debido a la gran eficiencia de la FFT es frecuente calcular la convolución computando la TDF inversa del producto de las TDF de cada sucesión a convolucionar, usando para cada cálculo de la TDF (directa e inversa) el algoritmo de la FFT, como se comentó en la Sección anterior. Sin embargo, muchas veces es preferible reformular el cálculo de la TDF como una convolución, en particular cuando no se necesita conocer la TDF en toda la banda de frecuencias entre 0 y 2π, sino en una banda más limitada.

Si basta con conocer algunas *pocas muestras* $X[k]$ del espectro $X\left(e^{j\omega}\right)$ se puede aplicar el algoritmo de Goertzel, donde el cálculo de la TDF se formula como una operación de filtrado recursivo.

Cuando se desea calcular un *conjunto* de M muestras del espectro $X\left(e^{j\omega}\right)$ de una sucesión finita $x[n]$ de longitud N, con $M < N$ en las frecuencias $\omega_m = \omega_0 + m\Delta\omega$, $0 \leq m \leq M-1$, se dispone de dos algoritmos, según sea la naturaleza de $\Delta\omega$ y de ω_0:

- Si, como muestra la Fig. 11.29(a) , los ω_m coinciden con un conjunto de muestras de frecuencia de la TDF,

$$\omega_m = \frac{2\pi}{N}k_0 + \frac{2\pi}{N}m, \quad 0 \leq m \leq M-1,$$

  es decir, $\omega_0 = 2\pi k_0/N$, $\Delta\omega = 2\pi/N$, $\omega_S = 2\pi(k_0 + M - 1)/N$ se puede aplicar el método conocido como *transformada zoom*.

- Si ω_0 y $\Delta\omega$ son arbitrarios, como ilustra la Fig. 11.29(b) , es más conveniente calcular la *transformada chirp*.

Finalmente, el algoritmo de Winograd explota las ideas anteriores para calcular eficientemente las convoluciones, disminuyendo el número de productos a costa de incrementar el número de sumas.

A continuación se estudian cada uno de los algoritmos mencionados.

11.8.1. Algoritmo de Goertzel

El algoritmo de Goertzel (Goertzel, 1958) aprovecha la periodicidad de la sucesión $e^{-j\frac{2\pi}{N}kn}$ para reducir cálculos. Es evidente que

$$e^{j\frac{2\pi}{N}kn}\Big|_{n=N} = e^{j\frac{2\pi}{N}kN} = 1.$$

Premultiplicando el miembro derecho de (11.30) por $e^{j\frac{2\pi}{N}kN}$, la ecuación no varía. Entonces,

$$X[k] = e^{j\frac{2\pi}{N}kN}\sum_{m=0}^{N-1} x[m]e^{-j\frac{2\pi}{N}km} = \sum_{m=0}^{N-1} x[m]e^{j\frac{2\pi}{N}k(N-m)}. \tag{11.40}$$

Si se define la sucesión

$$h_k[n] = (e^{j\frac{2\pi}{N}k})^n u[n],$$

la expresión (11.40) es similar a una convolución entre $x[n]$ y $h_k[n]$

$$y_k[n] = x[n] * h_k[n] = \sum_{m=0}^{n} x[m]h_k[n-m] = \sum_{m=0}^{n} x[m]e^{j\frac{2\pi}{N}k(n-m)}, \tag{11.41}$$

donde los límites de la sumatoria resultan de suponer que tanto $h_k[n]$ como $x[n]$ son causales (es decir, $h_k[n] = x[n] = 0$ para $n < 0$). Comparando (11.40) y (11.41) se tiene que

$$X[k] = \left. y_k[n]\right|_{n=N}.$$

Al calcular $\left. y_k[n]\right|_{n=N}$, el último de los elementos de la sumatoria (11.41) es $x[N]e^{j\frac{2\pi}{N}k(N-N)}$. Como $x[n]$ es de longitud finita con $0 \leq n \leq N-1$, $x[N]$ no está definido. Para solucionar este problema y conseguir que (11.41) coincida con (11.40) cuando $n = N$ basta con definir $x[N] = 0$. En otras palabras, es necesario agregar una muestra nula a la "cola" de la sucesión $x[n]$.

La ecuación (11.41) se puede interpretar como la salida de un sistema con respuesta impulsiva $e^{j\frac{2\pi}{N}kn}u[n]$ ante una excitación con una entrada de longitud finita $x[n]$. En particular, $X[k]$ es el valor de la muestra N-ésima de la salida, $y_k[N]$. El resto de las muestras de la salida del sistema no intersan para el cálculo de la TDF $X[k]$.

La Fig. 11.30 muestra el diagrama bloque de un sistema con respuesta impulsiva $h_k[n] = (e^{j\frac{2\pi}{N}k})^n u[n]$, suponiendo condiciones iniciales de reposo. La ecuación a diferencias de este sistema es

$$y[n] = e^{j\frac{2\pi}{N}k}y[n-1] + x[n], \quad y[-1] = 0, \tag{11.42}$$

y su función transferencia

$$H_k(e^{j\omega}) = \frac{1}{1 - e^{j\frac{2\pi}{N}k}e^{-j\omega}}. \tag{11.43}$$

Ya que la sucesión de entrada $x[n]$ y los coeficientes $e^{j\frac{2\pi}{N}kn}$ pueden ser ambos complejos, el cálculo de cada nuevo valor de $y_k[n]$ requiere 4 multiplicaciones y 4 sumas reales. Para calcular $y_k[N] = X[k]$ se deben calcular los N valores previos $y_k[0] = x[0]$, $y_k[1]$, $\ldots$, $y_k[N-1]$, por lo que el sistema de la Fig. 11.30 representa un "algoritmo de cálculo" que requiere $4N$ multiplicaciones reales y $4N$ sumas reales para calcular $X[k]$ en cada valor deseado de k[8]. Por lo tanto el procedimiento es ligeramente menos eficiente que el método directo, aunque evita el cálculo o almacenamiento de los $(N+1)$ coeficientes $e^{j\frac{2\pi}{N}kn}$, $0 \leq n \leq N$, pues estas cantidades se calculan mediante la recursión (11.42) que se observa en la Fig. 11.30.

El número de productos puede reducirse por un factor de 2. Multiplicando el numerador y el denominador de la función transferencia $H_k(e^{j\omega})$ de la ecuación (11.43) por el factor

[8] Como $y_k[0] = x[0]$, en realidad hace falta calcular N valores, desde $y_k[1]$ hasta $y_k[N]$.

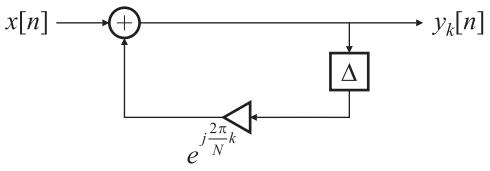


Fig. 11.30. Cálculo recursivo de $X[k]$ con un sistema recursivo de primer orden.

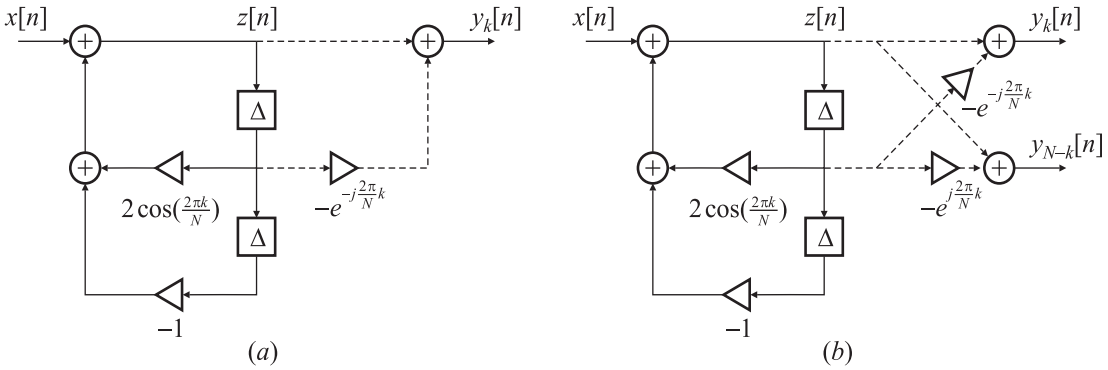


Fig. 11.31. (a) Sistema de segundo orden para el cálculo recursivo de $X[k]$. (b) Modificación para calcular simultáneamente $X[N-k]$. Las líneas de trazo indican las operaciones que se ejecutan sólo cuando $n = N$.

$(1 - e^{-j\frac{2\pi}{N}k}e^{-j\omega})$ se obtiene

$$\begin{aligned} H_k(e^{j\omega}) &= \frac{1}{(1 - e^{j\frac{2\pi}{N}k}e^{-j\omega})} \times \frac{1 - e^{-j\frac{2\pi}{N}k}e^{-j\omega}}{(1 - e^{-j\frac{2\pi}{N}k}e^{-j\omega})} \\ &= \frac{1 - e^{-j\frac{2\pi}{N}k}e^{-j\omega}}{1 - 2\cos(\frac{2\pi k}{N})e^{-j\omega} + e^{-j2\omega}} \end{aligned} \tag{11.44}$$

El sistema (11.44) puede verse como la conexión en cascada de dos sistemas, $H_k\left(e^{j\omega}\right) = H_1\left(e^{j\omega}\right) H_2\left(e^{j\omega}\right)$, donde

$$\begin{aligned} H_1\left(e^{j\omega}\right) &= \frac{Z\left(e^{j\omega}\right)}{X\left(e^{j\omega}\right)} = \frac{1}{1 - 2\cos(2\pi k/N)e^{-j\omega} + e^{-j2\omega}}, \\ H_2\left(e^{j\omega}\right) &= \frac{Y_k\left(e^{j\omega}\right)}{Z\left(e^{j\omega}\right)} = 1 - e^{-j\frac{2\pi}{N}k}e^{-j\omega}, \end{aligned}$$

que se puede implementar con el sistema acoplado de ecuaciones a diferencias

$$z[n] = 2\cos\left(\tfrac{2\pi}{N}k\right) z[n-1] - z[n-2] + x[n], \tag{11.45}$$

$$y[n] = z[n] - e^{-j\frac{2\pi}{N}k} z[n-1], \tag{11.46}$$

cuyo diagrama en bloque se muestra en la Fig. 11.31(a).

Si la entrada $x[n]$ es compleja, la ecuación (11.45), que implementa el denominador de (11.44) muestra que sólo son necesarias dos multiplicaciones reales por muestra ya que no es necesario hacer la multiplicación por el factor ± 1, y el coeficiente $[-2\cos(2\pi k/N)]$

es real . Con respecto a las sumas, como sucede con el sistema de primer orden (11.43), si la entrada es compleja se necesitan 4 sumas reales para implementar el denominador de (11.44) mediante la ecuación a diferencias (11.45). La multiplicación por el coeficiente $e^{-j\frac{2\pi}{N}k}$ necesaria para implementar el cero de la ecuación transferencia (11.44) mediante la ecuación a diferencias (11.46) no necesita efectuarse para cada muestra de la salida, sino una única vez, al computar $y_k[N]$. Mientras que (11.45) debe calcularse para cada una de las $(N+1)$ muestras de la salida, $n = 0, \ldots, N$, la ecuación (11.46) sólo necesita calcularse en la última iteración, cuando $n = N$, es decir,

$$y_k[N] = z[N] - e^{-j\frac{2\pi}{N}k} z[N-1].$$

El número total de operaciones es de $2N$ multiplicaciones y $4(N+1)$ sumas reales para la implementación de la ecuación a diferencias (11.45), y 4 multiplicaciones y 4 sumas reales para calcular (11.46) en el último paso; aproximadamente la mitad de multiplicaciones reales que se necesitan en la implementación del método directo. En este esquema apenas más eficiente cuenta al menos con la ventaja que sólo se deben calcular y almacenar *dos* coeficientes $2\cos(2\pi k/N)$ y $e^{-j\frac{2\pi}{N}k}$, en lugar de los $N+1$ coeficientes $e^{j\frac{2\pi}{N}kn}$ que aparecen en (11.40). Estos coeficientes se calculan implícitamente por la recursión de la ecuación a diferencias (11.45) que se muestra en Fig. 11.31(a).

Con muy pocas operaciones más también es posible calcular simultáneamente $X[N-k]$. En este caso, la respuesta impulsiva del sistema de primer orden (11.44) es

$$h_{N-k}[n] = e^{j\frac{2\pi}{N}(N-k)} u[n] = e^{-j\frac{2\pi}{N}k} u[n]$$

con respuesta en frecuencia dada por

$$H_{N-k}(e^{j\omega}) = \frac{1}{1 - e^{-j\frac{2\pi}{N}k} e^{-j\omega}}.$$

Esta función puede llevarse a un sistema de segundo orden con polos reales multiplicando numerador y denominador por el factor $(1 - e^{j\frac{2\pi}{N}k} e^{-j\omega})$, lo que resulta en

$$H_{N-k}(e^{j\omega}) = \frac{1 - e^{j\frac{2\pi}{N}k} e^{-j\omega}}{1 - 2\cos(2\pi k/N) e^{-j\omega} + e^{-j2\omega}} \tag{11.47}$$

cuyo denominador es el mismo que el de la función transferencia (11.44) necesaria para calcular $X[k]$: la única diferencia está en el numerador, donde aparece un factor que es el conjugado del coeficiente de (11.47), como se muestra en la Fig. 11.31(b) . Como este coeficiente interviene únicamente en la última iteración, las $2N$ multiplicaciones y las $4N$ sumas necesarias para calcular la ecuación a diferencias (11.45) pueden aprovecharse para calcular *dos* valores de la TDF, $X[k]$ y $X[N-k]$.

Para el cálculo de las N muestras de la TDF usando el algoritmo de Goertzel el número de productos reales es proporcional a N^2, y el número de sumas reales es $2N^2$. Aunque es más eficiente que el cálculo directo de la TDF, el volumen de cálculo sigue siendo proporcional a N^2.

Tanto con el método directo como con el algoritmo de Goertzel no es necesario evaluar $X[k]$ para los N valores de k. En realidad, se puede evaluar $X[k]$ para M valores de k, donde cada uno de los M valores deseados se calcula por el sistema recursivo de la

Fig. 11.31(a) con los coeficientes apropiados. En este caso el volumen de cálculo es proporcional a NM. En consecuencia, el método directo y el algoritmo de Goertzel son atractivos cuando se desea conocer unas pocas muestras del espectro de $X\left(e^{j\omega}\right)$, es decir cuando M es pequeño. Sin embargo, teniendo en cuenta que se dispone de algoritmos eficientes que permiten calcular la TDF de orden N utilizando sólo $N \log_2 N$ operaciones cuando N es potencia de 2, tanto el método directo como el algoritmo de Goertzel pueden ser los algoritmos de cómputo más eficientes si el número de puntos M donde se desea conocer la TDF satisface

$$M < \log_2 N.$$

Una ventaja del algoritmo de Goertzel sobre los restantes es que no sólo sirve para calcular $X[k]$, sino que también es útil para computar el espectro $X(e^{j\omega})$ de una sucesión $x[n]$ de longitud finita, para $\omega = \omega_0$, donde ω_0 no tiene por qué coincidir con alguna de las N frecuencias $2\pi k/N$ de la TDF; basta con reemplazar los coeficientes $2\cos(2\pi k/N)$ y $e^{\pm j\frac{2\pi}{N}k}$ por $2\cos(\omega_0)$ y $e^{\pm j\omega_0}$, respectivamente.

11.8.1.1. Código MATLAB para implementar el algoritmo de Goertzel

```
function X = gfft(x,k)
%X = GFFT(x,k)
%    Esta funcion implementa el Algoritmo de Goertzel (de segundo orden)
%    El orden N de la FFT esta dado por la longitud de la sucesion "x". La
%    variable "k" indica el indice donde se calcula la TDF (0 <= k <= N-1)
%
N = length(x);
x(N+1) = 0;
D = [1 -2*cos(2*pi*k/N) 1];
y = filter(1,D,x);
X = y(end)-exp(-sqrt(-1)*2*pi*k/N)*y(end-1);
```

11.8.2. Transformada zoom

La transformada zoom (Porat, 1997) es similar a la transformada chirp que se estudia en la Sección 11.8.3. Se utiliza cuando la resolución frecuencial es suficiente (no necesita ser incrementada), y sólo se necesita conocer $X[k]$ para un conjunto limitado de muestras, $k_0 \leq k \leq k_0 + (M-1)$. En general, la muestra inicial k_0 y el número de muestras M donde se desea computar $X[k]$ son arbitrarios, pero es natural fijar $0 \leq k_0, k_0 + (M-1) \leq (N-1)$[9].

Es conveniente suponer que la longitud N de la sucesión $x[n]$ puede factorizarse como $N = ML$, donde M es el número de muestras donde se desea calcular la TDF, y L es un entero. Eventualmente, puede hacerse un "padding" de ceros para llevar la sucesión original al largo requerido. En este sentido, el método es una particularización de la técnica de "dividir y combinar" presentada en la Sección 11.6.1.

La TDF de una señal $x[n]$ está dada por

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \quad 0 \leq k \leq N-1.$$

[9] También podría pensarse como una transformada chirp, descripta en la Sección 11.8.3, donde $\omega_0 = 2\pi k_0/N$, y $\Delta\omega = 2\pi/N$. Sin embargo, el punto de vista enfatizado por esta transformada permite el cálculo más eficiente que si se aplica la transformada chirp.

Ya que $N = ML$, se puede indizar la sucesión $x[n]$ con dos índices ℓ y m, eligiendo por ejemplo

$$n = \ell + mL, \quad 0 \leq \ell \leq L-1,\, 0 \leq m \leq M-1,$$

aunque en este caso no se enfatiza la organización por columnas, como en la Sección 11.6.1. Entonces,

$$X[k] = \sum_{\ell=0}^{L-1} \sum_{m=0}^{M-1} x[\ell + mL] e^{-j\frac{2\pi}{N}k(\ell+mL)}, \quad 0 \leq k \leq N-1.$$

Teniendo en cuenta que $e^{-j\frac{2\pi}{N}k(\ell+mL)} = e^{-j\frac{2\pi}{N}k\ell} e^{-j\frac{2\pi}{N}kmL} = e^{-j\frac{2\pi}{N}k\ell} e^{-j\frac{2\pi}{M}km}$, resulta

$$\begin{aligned} X[k] &= \sum_{\ell=0}^{L-1} \sum_{m=0}^{M-1} x[\ell + mL] e^{-j\frac{2\pi}{N}k\ell} e^{-j\frac{2\pi}{M}km} \\ &= \sum_{\ell=0}^{L-1} \underbrace{\left(\sum_{m=0}^{M-1} x[\ell + mL] e^{-j\frac{2\pi}{M}km} \right)}_{X_\ell[k]} e^{-j\frac{2\pi}{N}k\ell}, \quad 0 \leq k \leq N-1. \end{aligned} \tag{11.48}$$

El término entre paréntesis representa la TDF de M puntos de la sucesión $x_\ell[n] = x[\ell + nL]$, $0 \leq n \leq M-1$, $0 \leq \ell \leq L-1$. Cada una de estas L sucesiones $x_\ell[n]$ representa una sucesión de longitud M formada por una muestra cada L de la sucesión original $x[n]$, desfasadas entre sí una muestra de $x[n]$; se dice que $x_\ell[n]$ es una *decimación por* L de $x[n]$. Si $X_\ell[k]$ es la TDF de orden M de cada una de estas sucesiones,

$$X_\ell[k] = \sum_{m=0}^{M-1} x[\ell + mL] e^{-j\frac{2\pi}{M}km}, \quad 0 \leq k \leq M-1, \tag{11.49}$$

donde el índice k en (11.49) está limitado al intervalo $0 \leq k \leq M-1$, mientras que en (11.48) está comprendido entre 0 y $N-1$. No obstante, la periodicidad inherente de la exponencial compleja hace que pueda sustituirse (11.49) en (11.48) o bien, para evitar problemas con los índices, utilizar $X_\ell[((k))_M]$, y entonces,

$$X[k] = \sum_{\ell=0}^{L-1} X_\ell[((k))_M] e^{-j\frac{2\pi}{N}k\ell}, \quad 0 \leq k \leq N-1.$$

Esta sumatoria no puede calcularse con un algoritmo tipo FFT, pues $X_\ell[((k))_M]$ depende de k; debe notarse la diferencia con (11.49) donde el argumento es función de ℓ y m pero no de k.

En resumen, la TDF $X[k]$ en el rango $k_0 \leq k \leq k_0 + (M-1)$ puede calcularse de la siguiente manera:

1. Se calculan las L TDF de orden M de las sucesiones decimadas $x_\ell[n] = x[\ell + nL]$, $0 \leq n \leq M-1$, $0 \leq \ell \leq L-1$, de acuerdo con (11.49).

2. Para cada k en el rango deseado se calcula

$$X[k] = \sum_{\ell=0}^{L-1} X_\ell[((k))_M] e^{-j\frac{2\pi}{N}k\ell}, \quad k_0 \leq k \leq k_0 + (M-1). \tag{11.50}$$

11.8.2.1. Número de operaciones

Para un cálculo aproximado del número de operaciones necesarias y suponiendo que $M = 2^v$, en el paso 1 se requieren L veces $M \log_2 M$ operaciones si la TDF se calcula por un algoritmo de raíz 2. Por otra parte, en el paso 2 son necesarias L operaciones para cada k deseado, lo que hace un total de $ML = N$ multiplicaciones complejas. El número total de operaciones que necesita este método es

$$\mathcal{N}_{\text{zoom}} = N + LM \log_2 M. \tag{11.51}$$

11.8.2.2. Código MATLAB para el cómputo de la transformada zoom

```
function X = zoomfft(x,k0,M,N)
%X = zoomfft(x,k0,M,N)
%    Esta funcion calcula la FFT-zoom, ie la TF X[k] de N puntos de x[n]
%    solo para las M muestras k0,k0+1,...k0+(M-1);
if rem(N,M) == 0
     x = reshape(x,1,length(x));  % x como vector fila
     L = ceil(N/M);
     x = [x zeros(1,N-length(x))]; % agrega ceros si hace falta
     k = [0:M-1];
     X = reshape(x,L,M);          % M columnas de L elementos consecutivos
     X = fft(X,M,2);              % la TDF de las filas
     X = sum(exp(-2*pi/N*1i*[0:L-1]'*(k+k0)).*X(:,1+rem(k0+k,M)),1);
else
     error('K debe ser divisor de N')
     return
end
```

11.8.3. Transformada chirp

El cálculo de la TDF por medio de la FFT de orden N permite una resolución frecuencial de $2\pi/N$, donde en general N es la longitud de la sucesión de entrada. Si para alguna aplicación particular esta resolución no es suficiente, se puede aumentar la longitud de la sucesión agregándole N_C ceros (efectuado un "padding" de ceros), y efectuando una TDF de largo $(N + N_C)$, llevando la resolución a $2\pi/(N + N_C)$. Aunque esta es una solución válida, si el aumento de resolución deseado es significativo se puede incrementar demasiado el número de operaciones necesarias. Además, frecuentemente no se desea una gran resolución en toda la gama de frecuencias, sino en un determinado ancho de banda. Por ejemplo, si se desea determinar la frecuencia ω_0 de una sinusoide $x[n]$ con buena precisión a partir de un conjunto de N muestras, el cálculo de la TDF de N puntos permitirá una resolución frecuencial de $\pm\pi/N$. Si se pudiese computar $X\left(e^{j\omega}\right)$ con alta resolución en un intervalo de ancho $2\pi/N$ alrededor del valor máximo de la TDF podría mejorarse la precisión de la estimación de ω_0. Un método denominado *transformada chirp* (Rabiner, 1969; Bluestein, 1970) permite efectuar este cálculo.

Originariamente, el algoritmo se desarrolló para detectar las frecuencias de los formantes en el procesamiento de voz (Rabiner, 2004). El espectro de la voz se calculaba utilizando filtrado homomórfico (una técnica de filtrado no lineal), y la frecuencia de los formantes relevantes se estimaban detectando los picos del espectro. Aunque la técnica funcionaba bien en general, fallaba cuando las frecuencias de los formantes estaban muy próximas, o su ancho de banda era muy amplio.

La transformada chirp también se basa en representar la TDF como una suma convolución. No es óptima en la minimización de ningún tipo de complejidad computacional,

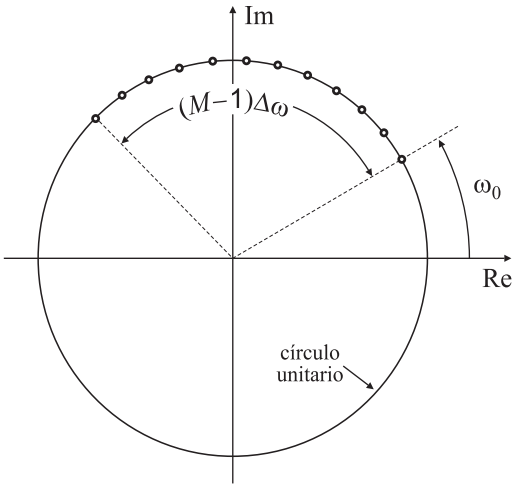


Fig. 11.32. Muestras frecuenciales de la transformada chirp.

pero ha sido útil en gran variedad de aplicaciones, particularmente en aquellos casos en que se puede implementar de manera eficiente una convolución con una respuesta impulsiva predeterminada. El algoritmo de la transformada chirp es más flexible que la FFT, pues se puede utilizar para calcular cualquier conjunto de muestras equiespaciadas de la transformada de Fourier.

Para derivar el algoritmo de la transformada chirp se supone que $x[n]$ es una sucesión de N puntos y con transformada de Fourier $X(e^{j\omega})$. La evaluación de M muestras equiespaciadas de esta transformada en las frecuencias

$$\omega_k = \omega_0 + k\Delta\omega, \quad k = 0, 1, \ldots M-1, \tag{11.52}$$

donde ω_0 y $\Delta\omega$ son arbitrarias (Fig. 11.32) es

$$X\left(e^{j\omega}\right)\Big|_{\omega=\omega_k} = X\left(e^{j\omega_k}\right) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_k n}. \tag{11.53}$$

Si $\omega_0 = 0$, $M = N$ y $\Delta\omega = 2\pi/N$, el cálculo coincide con el de la TDF. Si ω_0 y $\Delta\omega/2\pi \notin Q$, los puntos ω_k no podrían calcularse directamente con la TDF, pero sí a partir de ella usando la fórmula de interpolación derivada en el Ejercicio 7 de los Problemas del Capítulo 4.

Reemplazando (11.52) en (11.53),

$$X\left(e^{j\omega_k}\right) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_0 n}e^{-j\,\Delta\omega\,kn} \tag{11.54}$$

y notando que

$$nk = \frac{1}{2}\left[n^2 + k^2 - (k-n)^2\right],$$

se puede expresar (11.54) como

$$X\left(e^{j\omega_k}\right) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_0 n}e^{-j\,\Delta\omega\frac{n^2}{2}}e^{-j\,\Delta\omega\,\frac{k^2}{2}}e^{j\,\Delta\omega\,\frac{(k-n)^2}{2}}.$$

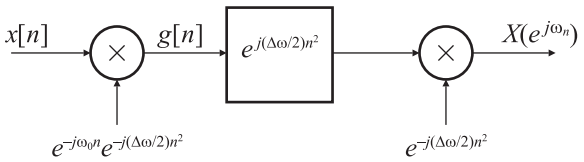


Fig. 11.33. Diagrama bloque del algoritmo de la transformada chirp.

Introduciendo la sucesión auxiliar $g[n] = x[n]e^{-j\omega_0 n}e^{-j\,\Delta\omega\frac{n^2}{2}}$ se puede escribir

$$X\left(e^{j\omega_k}\right) = e^{-j\,\Delta\omega\frac{k^2}{2}}\sum_{n=0}^{N-1} g[n]e^{j\,\Delta\omega\,\frac{(k-n)^2}{2}}, \quad k = 0, 1, \ldots M-1. \tag{11.55}$$

Para poner en evidencia que (11.55) se puede escribir como la salida de un sistema lineal e invariante en el tiempo es conveniente intercambiar los índices k y n,

$$X\left(e^{j\omega_n}\right) = e^{-j\,\Delta\omega\frac{n^2}{2}}\sum_{k=0}^{N-1} g[k]e^{-j\,\Delta\omega\,\frac{-(n-k)^2}{2}}, \quad n = 0, 1, \ldots M-1. \tag{11.56}$$

La ecuación (11.56) revela que $X\left(e^{j\omega_n}\right)$ corresponde a la convolución de la sucesión $g[n]$ con la sucesión $e^{j\,\Delta\omega\frac{n^2}{2}}$, seguida por una multiplicación con la sucesión $e^{-j\,\Delta\omega\frac{n^2}{2}}$. La sucesión de salida, indexada en la variable independiente n es la sucesión de las muestras espectrales $X\left(e^{j\omega_n}\right)$. La sucesión $e^{j\,\Delta\omega\frac{n^2}{2}}$ es una exponencial compleja cuya frecuencia $(n\,\Delta\omega)$ varía linealmente con n; estas señales se denominan *chirp* en los sistemas de radar[10], y de aquí el nombre de la transformada. En el procesamiento de señales de sonar se utiliza un sistema similar al de la Fig. 11.33 para compresión de pulsos (Skolnik, 1986).

Para evaluar las muestras de la transformada de Fourier especificadas en (11.56) se necesita calcular la salida del sistema de la Fig. 11.33 sólo sobre un intervalo finito. En la Fig. 11.34 se muestran las sucesiones $g[n]$, $e^{j\,\Delta\omega\frac{n^2}{2}}$ y $g[n] * e^{j\,\Delta\omega\frac{n^2}{2}}$. Como $g[n]$ tiene duración finita, en el cálculo de la convolución para el intervalo $n = 0$, 1, ..., $M-1$ se utiliza sólo un conjunto finito de muestras de $e^{j\,\Delta\omega\frac{n^2}{2}}$. Específicamente, sólo interesa la porción entre $n = -(N-1)$ hasta $n = M-1$. Definiendo

$$h[n] = \begin{cases} e^{j\,\Delta\omega\frac{n^2}{2}}, & -(N-1) \le n \le M-1, \\ 0, & \text{en caso contrario,} \end{cases} \tag{11.57}$$

como se muestra en la Fig. 11.35, es sencillo verificar que

$$g[n] * e^{j\,\Delta\omega\frac{n^2}{2}} = g[n] * h[n], \quad n = 0, 1, \ldots M-1.$$

En consecuencia, la sucesión $e^{j\,\Delta\omega\frac{n^2}{2}}$ de longitud infinita en el sistema de la Fig. 11.33 se puede reemplazar por la sucesión de longitud finita $h[n]$ de la Fig. 11.35. El cálculo se

[10] Una señal $x(t) = e^{j\frac{1}{2}\alpha t^2}$, con $\alpha > 0$, denominada *chirp*, es una señal compleja cuyo módulo es constante, y cuya frecuencia crece linealmente en el tiempo. El ancho de banda de una señal *chirp* de duración T_0 es aproximadamente αT_0. En aplicaciones de radar y sonar la duración de la señal determina la resolución de la medida de la velocidad del blanco; la señal *chirp* permite el control simultáneo de la resolución de velocidad (con T_0), y la resolución de rango (con el parámetro α).

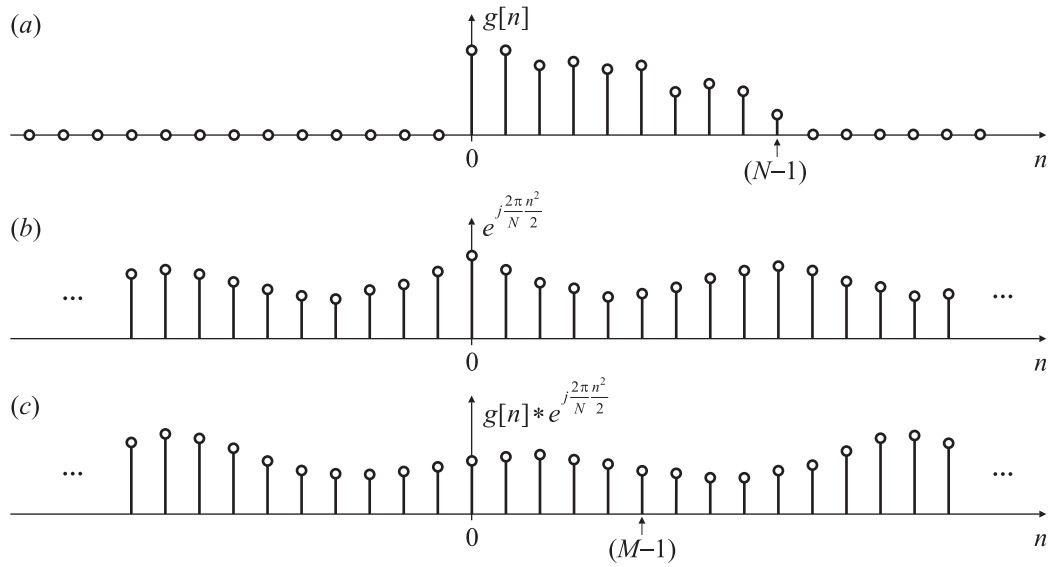


Fig. 11.34. Representación de las sucesiones del algoritmo de la transformada chirp (las sucesiones que intervienen son complejas). (a) $g[n] = x[n]e^{-j\omega_0 n}e^{-j\frac{2\pi}{N}\frac{n^2}{2}}$; (b) $e^{-j\frac{2\pi}{N}\frac{n^2}{2}}$; (c) $g[n] * e^{-j\frac{2\pi}{N}\frac{n^2}{2}}$.

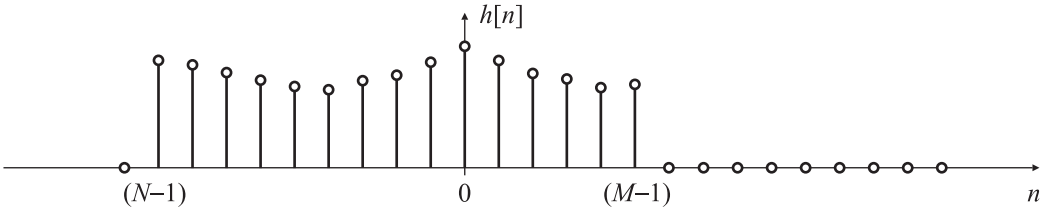


Fig. 11.35. Representación de la respuesta impulsiva del filtro FIR $h[n]$ (en realidad, la respuesta es compleja).

puede representar como el sistema de la Fig. 11.36(a), donde $h[n]$ está dado por (11.57), y las muestras en frecuencia son

$$X\left(e^{j\omega_n}\right) = y[n], \quad n = 0, 1, \ldots M-1. \tag{11.58}$$

Tanto el algoritmo de la transformada chirp como el algoritmo de Goertzel calculan la TDF como la salida de un filtro; sin embargo en el primero las M muestras de la salida corresponden a muestras del espectro $X\left(e^{j\omega_n}\right)$, mientras que en el segundo sólo la muestra N-ésima de la salida corresponde a la muestra k-ésima del espectro.

La evaluación de las muestras frecuenciales como sugiere la Fig. 11.36(a) tiene un número de ventajas potenciales. En general, no se necesita que $N = M$ como en los algoritmos de FFT, y no se necesita que N o M sean números altamente compuestos; y hasta pueden ser números primos. Además, el parámetro ω_0 es arbitrario. Esta flexibilidad no impide un cálculo eficiente, pues la convolución de la Fig. 11.36(a) puede implementarse a partir de la convolución circular calculada con la TDF. Como se discutió entonces, el tamaño de la TDF debe ser superior o igual a $(M + N - 1)$ para que la convolución circular coincida con $g[n] * h[n]$ para $n = 0, 1, \ldots M-1$. Respetando esta condición, el orden de la TDF

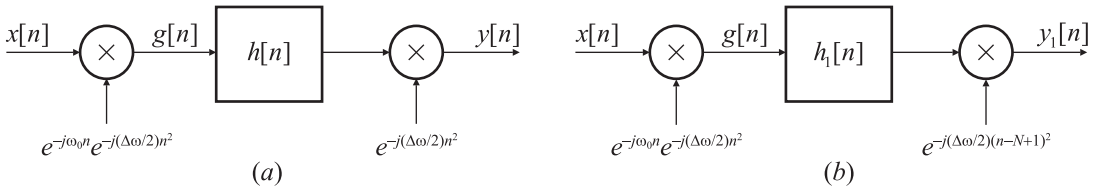


Fig. 11.36. Diagrama bloque del filtro FIR para el cálculo de la transformada chirp: (a) filtro no causal; (b)filtro causal.

es arbitrario, y puede elegirse potencia de 2 para hacer el cómputo todavía más eficiente. También se puede calcular la TDF utilizando el algoritmo de Winograd que se basa en una convolución.

En el sistema de la Fig. 11.36(a) la respuesta impulsiva $h[n]$ es no causal, y no puede utilizarse para aplicaciones de tiempo real. Sin embargo, como la respuesta es de duración finita se puede causalizar fácilmente retardando $h[n]$ por $(N-1)$ muestras para obtener la respuesta impulsiva (causal)

$$h_1[n] = \begin{cases} e^{j\,\Delta\omega\frac{(n-N+1)^2}{2}}, & 0 \le n \le M+N-2, \\ 0, & \text{en caso contrario.} \end{cases} \tag{11.59}$$

Como tanto la salida como la señal chirp que la multiplica quedan retardadas $(N-1)$ muestras, las muestras de las transformadas de Fourier coinciden con las últimas M muestras de la salida $y[n]$,

$$X\left(e^{j\omega_n}\right) = y_1[n+N-1], \quad n = 0, 1, \ldots M-1.$$

La modificación del sistema de la Fig. 11.36(a) para hacerlo causal resulta en el sistema de la Fig. 11.36(b). Una ventaja de este sistema es que involucra la convolución de la señal de entrada (modulada por la chirp) con una respuesta impulsiva causal y predeterminada. Ciertas tecnologías, como los dispositivos acoplados por carga (CCD) o por ondas acústicas de superficie (SAW) son particularmente útiles para este tipo de implementación, donde la respuesta impulsiva se especifica en el momento de fabricación de acuerdo a un patrón geométrico de electrodos. Una aproximación similar fue desarrollada por Hewes (1979) para implementar el algoritmo de la transformada chirp con CCDs.

11.8.3.1. Código Matlab para el cálculo de la transformada chirp

```
function X = chirpfft(x,w0,Dw,M);
% X = chirpfft(x,w0,Dw,M).
%   Esta función calcula la FFT-chirp en un intervalo de frecuencias.
%   x:  vector de entrada
%   w0: frecuencia inicial (en radianes)
%   Dw: incremento frecuencial (en radianes)
%   M:  número de puntos donde se calcula la tranformada chirp
N = length(x);
n = 0:N-1;                                  % índice de las muestras
x = reshape(x,1,N);                         % x como vector fila
g = x.*exp(-j*(Dw/2*n+w0).*n);              % modulación del vector x
h = exp(j*Dw/2*([0:M+N-2]-N+1).^2);         % la rta impulsiva del filtro FIR
g = [g, zeros(1,M-1)];                      % padding de ceros de g
X = filter(h,1,g);                          % filtrado de g por h
X = X(N:N+M-1).*exp(-j*Dw/2*(0:M-1).^2); % modulación de la salida
```

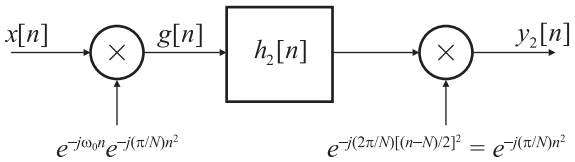


Fig. 11.37. Algoritmo de la transformada chirp para obtener las muestras de la TDF.

11.8.3.2. Cálculo de la TDF con la Transformada chirp

La transformada chirp se simplifica cuando las muestras en frecuencia ω_n coinciden con las de la TDF. En este caso, $\omega_0 = 0$, $e^{-j\,\Delta\omega} = e^{-j\frac{2\pi}{N}}$, y entonces $\omega_n = 2\pi n/N$. En este caso es conveniente modificar el sistema de la Fig. 11.36(b). Agregando un retardo adicional a la respuesta impulsiva del filtro, y considerando que N es par, $(e^{-j\frac{2\pi}{N}})^N = 1$, de manera que

$$e^{-j\frac{2\pi}{N}\frac{(n-N)^2}{2}} = e^{-j\frac{2\pi}{N}\frac{n^2}{2}}.$$

El sistema resultante se muestra en la Fig.11.37, donde

$$h_2[n] = \begin{cases} e^{j\frac{2\pi}{N}\frac{n^2}{2}}, & 1 \le n \le M+N-1, \\ 0, & \text{en caso contrario.} \end{cases}$$

En este caso, la señal chirp que modula la entrada $x[n]$ y la que modula la salida del filtro FIR son idénticas, y

$$X\left(e^{j\omega}\right)\big|_{\omega=\frac{2\pi}{N}n} = y_2[n+N], \quad n = 0, 1, \ldots, M-1.$$

EJEMPLO **11.2. Parámetros de la transformada chirp**

La sucesión $x[n]$ es no nula en el intervalo $n = 0, \ldots, 25$, y se desean calcular 16 muestras de $X\left(e^{j\omega}\right)$ en las frecuencias

$$\omega_k = \frac{2\pi}{27} + \frac{2\pi}{1024}k,$$

para $k = 0, \ldots 15$. Se pueden calcular las muestras frecuenciales deseadas efectuando la convolución con una respuesta impulsiva causal usando el sistema de la Fig. 11.36(b) seleccionando adecuadamente los parámetros. Se elige el número de muestras deseadas $M = 16$, el largo de la sucesión $N = 26$. La frecuencia de la muestra inicial es $\omega_0 = 2\pi/27$, mientras que el intervalo entre muestras es $\Delta\omega = 2\pi/1024$. Con esta elección de parámetros la respuesta impulsiva causal está dada por la ecuación (11.59),

$$h_1[n] = \begin{cases} e^{j\frac{2\pi}{1024}\frac{(n-25)^2}{2}}, & 0 \le n \le 40, \\ 0, & \text{en caso contrario.} \end{cases}$$

Para esta respuesta impulsiva causal, las muestras en frecuencia deseadas coinciden con la salida $y_1[n]$ del sistema a partir de la muestra 25, es decir

$$y_1[n+25] = X\left(e^{j\omega}\right)\big|_{\omega=\frac{2\pi}{27}+\frac{2\pi}{1024}n}, \quad n = 0, \ldots, 15. \qquad \square$$

Para el caso en que $\Delta\omega = 2\pi/N$, con N cuadrado perfecto, Bluestein (1970) propuso un algoritmo similar al de la transformada chirp que usa un filtro recursivo en lugar de FIR. Rabiner y colaboradores (1969) generalizaron el algoritmo de la transformada chirp para obtener muestras equiespaciadas de la transformada z sobre un contorno espiralado en el plano complejo. Este algoritmo más general se denomina Transformada "chirp $\mathcal{Z}$".

11.8.3.3. Número de operaciones

El número total de operaciones necesarias para el cálculo de la transformada chirp está dado por la cantidad de operaciones que demande el cálculo de la convolución, además de las N multiplicaciones de la sucesión de entrada $x[n]$ por $e^{-j\omega_0}e^{-j\frac{\Delta\omega}{2}n^2}$, y los M productos de las muestras de la salida por $e^{-j\frac{\Delta\omega}{2}(n-N+1)^2}$.

La convolución lineal entre la sucesión $g[n]$, de longitud N, y el filtro $h_1[n]$ dado por (11.59), cuya longitud es $(N-1)+(M-1)+1 = N+M-1$ muestras es una sucesión de longitud $N+(N+M-1)-1 = 2N+M-2$. De esta convolución sólo interesan las M últimas muestras, de manera que si se la calcula efectuando la sumatoria indicada en (11.56) son necesarias M veces N productos. En consecuencia, el número total de operaciones es

$$\mathcal{N}_{\text{chirp}} = N + M + NM. \tag{11.60}$$

Si la convolución entre $g[n]$ y $h_1[n]$ se computa por los métodos rápidos basados en la TDF, para que la convolución circular coincida con la convolución lineal el orden de la TDF debe ser igual a la longitud de la sucesión resultado, es decir, $2N+M-2$. Este número puede o no ser mayor que el orden N_{TDF} de la TDF que sería necesario para tener la resolución deseada, que al menos es $N_{TDF} = 2\pi/\Delta\omega$. Sin embargo, como no interesa calcular *todas* las muestras de la convolución, sino sólo los últimos M valores, pueden utilizarse convoluciones circulares de dimensión menor. Recordando que una convolución lineal de orden L es análoga a la suma de réplicas de la convolución lineal demoradas rL muestras, $r \in \mathbb{Z}$, el orden de la TDF necesaria resulta de evitar el solapamiento de M muestras. Una posibilidad es la descrita en el siguiente Algoritmo

Algoritmo:

1. Calcular un número $L = 2^v$, tal que $L > N+M-1$.
2. Formar la sucesión $\tilde{g}[n]$ agregando $L-N$ ceros a la sucesión $g[n]$.
3. Formar la sucesión $\hat{h}_1[n]$ agregando $L-M$ ceros a la sucesión $h_1[n]$ dada por (11.59)
4. Calcular $\tilde{h}[n] = \hat{h}_1[((n-(L-M)))_L]$: esta sucesión es la concatenación de $h[n]$, $0 \le n \le N-1$ con $h[n]$, $-(L-K) \le n \le -1$, donde $h[n]$ está dada por (11.57).
5. Computar la TDF inversa de orden L de $G[k]H[k]$, donde $G[k]$ y $H[k]$ son las TDF de orden L de $\tilde{g}[n]$ y $\tilde{h}[n]$.
6. Las M muestras de la transformada chirp son las primeras M muestras de la sucesión resultante.

Si las TDF se calculan con los algoritmos de decimación en tiempo y/o decimación en frecuencia, el número de operaciones necesarias para calcular la convolución es aproximadamente $(N+M)\log_2(N+M)$. Finalmente, el cálculo de la transformada insume

$$\mathcal{N}_{\text{chirp}} = (N+M)[1+\log_2(N+M)]. \tag{11.61}$$

El siguiente algoritmo implementa el cómputo de la transformada chirp usando la convolución circular.

11.8.3.4. Código Matlab para calcular la transformada chirp por convolución circular

```
function X = chirpfftcc(x,w0,Dw,M);
% X = chirpfftcc(x,w0,Dw,M).
%   Esta función calcula la FFT-chirp en un intervalo de frecuencias usando
%   convolución circular.
%   x:  vector de entrada
%   w0: frecuencia inicial (en radianes)
%   Dw: incremento frecuencial (en radianes)
%   M:  número de puntos donde se calcula la tranformada chirp
N = length(x);
n = 0:N-1;                                  % índice de las muestras
x = reshape(x,1,N);                         % x como vector fila
g = x.*exp(-j*(Dw/2*n+w0).*n);              % modulación del vector x
L = 2^nextpow2(N+M-1);                      % largo potencia de 2 para mayor eficiencia
h = exp(j*Dw/2*([0:M-1 -L+M:-1]).^2);       % la rta impulsiva del filtro FIR
g = [g, zeros(1,L-N)];                      % padding de ceros de g
X = ifft(fft(g).*fft(h));                   % filtrado de g por h (convolución circular)
X = X(1:M).*exp(-j*Dw/2*(0:M-1).^2);        % modulación de la salida
```

11.8.4. Comparación de la eficiencia de las transformadas zoom y chirp

El número total de operaciones para calcular las N muestras en frecuencia aplicando una FFT de orden $N = LM$ es aproximadamente

$$\mathcal{N}_{\text{FFT}} = N \log_2 N = LM \left(\log_2 M + \log_2 L\right).$$

La diferencia entre el número de operaciones de la transformada zoom y la FFT es

$$\Delta\mathcal{N} = \mathcal{N}_{\text{FFT}} - \mathcal{N}_{\text{zoom}} = N(\log_2 L - 1) = N(\log_2 N/M - 1).$$

El ahorro en el número de operaciones es tanto más importante cuanto menor sea el número de puntos M donde se desea calcular la TDF frente a la longitud N de la sucesión $x[n]$, o a la resolución $2\pi/N$ deseada. Basta con que M sea cuatro veces menor que N para que la transformada zoom sea más conveniente que la TDF calculada con la FFT de raíz 2.

Si la TDF en las muestras deseadas se calcula con la transformada chirp eligiendo $\omega_0 = 0$ y $\Delta\omega = 2\pi/N$, el número de operaciones es

$$\mathcal{N}_{\text{chirp}} = (N + M)[1 + \log_2(N + M)] \tag{11.62}$$

y la diferencia en el número de operaciones entre las transformadas chirp y zoom es (si $N \gg 1$),

$$\Delta\mathcal{N} = \mathcal{N}_{\text{chirp}} - \mathcal{N}_{\text{zoom}} \simeq M[1 + \log_2(N + M)] + N \log_2 \frac{N + M}{M}.$$

En la Fig. 11.38 se grafica el número de operaciones de la transformadas chirp y zoom relativas al número de operaciones de la FFT en función de la relación N/M. Aunque estas relaciones son función del orden N de la TDF, el gráfico no varía significativamente. La comparación no es favorable para la transformada chirp, pero debe tenerse presente que el número de operaciones calculado en (11.62) está basado en una aplicación no habitual de la transformada. Para el cálculo de la TDF en algunos pocos puntos $k_0 \leq k \leq k_0 + (M-1)$, la transformada zoom es bastante más eficiente que la FFT o que la transformada chirp.

EJEMPLO **11.3. Comparación de las transformadas zoom y chirp**

La señal

$$x[n] = \sum_{m=0}^{4} A_m \cos(\omega_0 \pi n), \quad 0 \leq n \leq 799, \tag{11.63}$$

está compuesta por cinco tonos cuyas frecuencias y amplitudes relativas son

$$\begin{array}{ll} \omega_0 = 0{,}1330, & A_0 = 1{,}00, \\ \omega_1 = 0{,}3750, & A_1 = 0{,}02, \\ \omega_2 = 0{,}6095, & A_2 = 1{,}00, \\ \omega_3 = 0{,}6250, & A_3 = 0{,}10, \\ \omega_4 = 0{,}9000, & A_4 = 0{,}10. \end{array}$$

La Fig. 11.39(a) muestra la el módulo en dB de la TDF de $N = 800$ puntos de $x[n]$, graficada para $0 \leq k \leq 399$. Los valores de $X[k]$ se han interpolado linealmente para facilitar la observación del espectro. En la figura se grafican dos ejes de abscisas, uno correspondiente al número de muestra k y otro a la frecuencia ω relativa a π. El cálculo de la transformada insume $\mathcal{N}_{\mathsf{FFT}} = 134136$ operaciones de punto flotante ("flops").

Si se desea estudiar sólo el entorno de las frecuencias ω_2 y ω_3, se puede calcular una transformada zoom. En este caso, se elige $k_0 = 239$, $M_{\mathsf{zoom}} = 16$. La gráfica del módulo de $X[k]$ en dB se muestra en la Fig. 11.39(b) . Este resultado es idéntico a magnificar la Fig. 11.39(a) en el rango de muestras (o frecuencias) de interés. Sin embargo, el número de operaciones necesarias para el cálculo de la transformada zoom es de $\mathcal{N}_{\mathsf{zoom}} = 31662$ flops, un poco menos que la cuarta parte del número de operaciones necesarias para calcular la FFT de 800 puntos.

Finalmente, en la Fig. 11.39(c) se muestra el resultado de calcular la transformada chirp, donde los parámetros ω_0, $\Delta\omega$ y M_{chirp} se eligien de manera de cubrir el mismo rango de frecuencias que con la transformada zoom, pero con una resolución 4 veces mayor. Resulta entonces

$$\omega_0 = \frac{2\pi}{N} k_0 = 2\pi \frac{239}{800}, \quad \Delta\omega = \frac{1}{4}\frac{2\pi}{N}, \quad M_{\mathsf{chirp}} = 4M_{\mathsf{zoom}} - 3 = 61.$$

La mejora de la resolución espectral se logra a costa de un mayor número de operaciones: $\mathcal{N}_{\mathsf{chirp}} = 384361$ flops, es decir casi 2.9 veces mayor que $\mathcal{N}_{\mathsf{FFT}}$. Sin embargo, si se calcula la TDF de $4N$ puntos, de manera de tener la misma resolución que con la transformada chirp, resulta $\mathcal{N}_{\mathsf{FFT}} = 638983$ flops $\approx 1{,}7\mathcal{N}_{\mathsf{chirp}}$. Finalmente, si la transformada chirp se calcula en los mismos puntos que la transformada zoom el número de operaciones casi no varía ($\mathcal{N}_{\mathsf{chirp}} = 383821$ flops) pese a que el número de puntos se redujo en cuatro veces. Esto revela que el número de cálculos auxiliares no es despreciable frente a la cantidad de operaciones que dependen del número de puntos donde se calcula la transformada. En consecuencia, debido a su mayor complejidad computacional, el empleo de esta transformada debe limitarse a aquellos casos en que se desee calcular muestras del espectro $X(e^{j\omega})$ que sean complicadas de calcular con la FFT o la transformada zoom. □

11.8.5. Algoritmo de Winograd

El algoritmo propuesto por Winograd (1978) expresa la TDF en función de convoluciones o multiplicaciones de polinomios. Se basa en descomponer la TDF en múltiples TDF de menor longitud, donde el orden de cada una de ellas es relativamente primo. Rader (1968) propuso un algoritmo eficiente para convertir la TDF en una convolución cuando el número de muestras es primo, pero su aplicación necesitó del desarrollo de métodos eficientes para el calculo de convoluciones periódicas. Winograd obtuvo una nueva aproximación

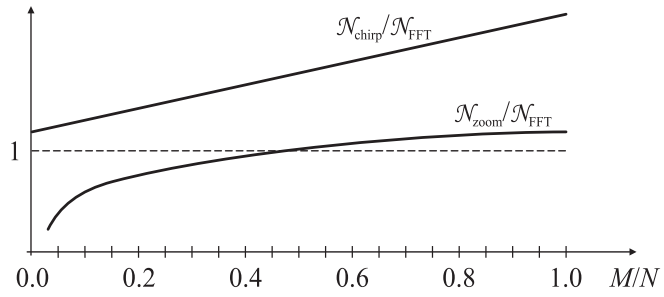


Fig. 11.38. Número de operaciones de la transformada chirp y la transformada zoom relativos al número de operaciones de la FFT.

para el cálculo de la TDF al combinar estos resultados junto con algoritmos altamente eficientes para el cálculo de convoluciones cíclicas. Estas técnicas se basan en conceptos relativamente avanzados de la teoría de números, tal como el teorema chino del resto, y no serán exploradas aquí; en McClellan y Rader (1979), Blahut (1985) y Burrus (1988) pueden encontrarse excelentes discusiones de los detalles del algoritmo de Winograd.

Con el algoritmo de Winograd el número de multiplicaciones necesarias para una TDF de N puntos se reduce de $N \log_2 N$ a N, a costa de incrementar notablemente el número de sumas. Es apropiado para usar en procesadores en que los productos son mucho más lentos que las sumas. Sin embargo, en los procesadores modernos en que la multiplicación y la suma se ejecutan en la misma cantidad de ciclos de reloj, los algoritmos de Cooley-Tukey o de factores primos son más convenientes. Otras dificultades del algoritmo de Winograd son la indización complicada, la imposibilidad de realizar el cálculo "en el lugar" (es necesario contar con memoria adicional), y la existencia de diferencias estructurales notorias para distintos valores de N.

En consecuencia, aunque el algoritmo de Winograd es extremadamente importante para determinar qué tan eficiente puede ser el cómputo de la TDF según el número de multiplicaciones, en la determinación de la velocidad o eficiencia de una implementación en hardware o software del cálculo de la TDF suelen participar otros factores de deben analizarse cuidadosamente en cada caso.

11.9. Breve reseña histórica del desarrollo de la FFT

La primera aparición de un método eficiente para el cálculo de la transformada de Fourier (TDF), como muchos otros algoritmos, se debe a Gauss (Heideman *et al.*, 1985). Como se mostró en el Capítulo 2, Gauss estaba interesado en ciertos cálculos astronómicos –un área recurrente para la aplicación de la FFT– para obtener la interpolación de las órbitas de algunos asteroides a partir de un conjunto de observaciones equiespaciadas. La perspectiva de un tedioso y laborioso cálculo manual fue sin duda un buen aliciente para desarrollar un algoritmo de cálculo más eficiente. Un número menor de operaciones disminuye la oportunidad de cometer errores, y también es numéricamente más estable. Gauss notó que una serie de Fourier de $N = N_1 N_2$ observaciones se podría descomponer como N_2 TDF de largo N_1 que se combinaban como N_1 TDF de largo N_2. El algoritmo de Gauss, desarrollado alrededor de 1805, pasó desapercibido porque fue publicado en latín,

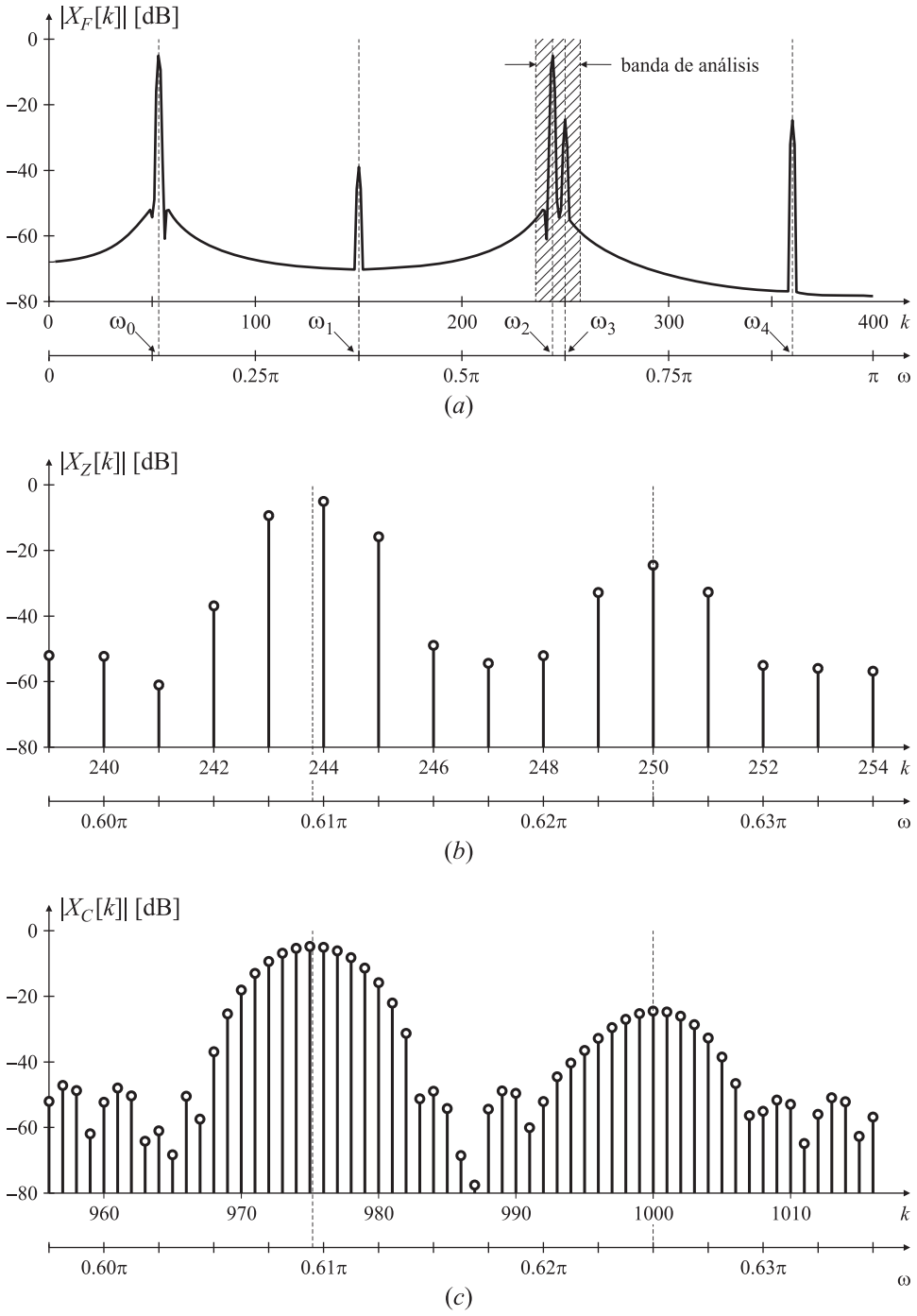


Fig. 11.39. TDF de la señal $x[n]$ de la ecuación (11.63). TDF de 1024 puntos (a) . Transformada zoom (b). Transformada chirp (c).

después de su muerte, en la recopilación de todas sus obras.

John W. Tukey concibió la idea básica del algoritmo para el cálculo rápido de la transformada de Fourier durante una reunión del Comité Científico Asesor del presidente Kennedy. Entre otros temas, el Comité estudiaba el desarrollo de técnicas para detección de pruebas

nucleares en la Unión Soviética, ya que la ratificación de un tratado para la eliminación de las mismas dependía de la obtención de un método que permitiese detectarlas sin necesidad de visitar las plantas nucleares rusas. Una idea era analizar los registros sismológicos obtenidos mediante sismógrafos ubicados en el lecho marino. La cantidad de sensores y la longitud de los datos demandaban algoritmos que permitieran calcular las TDF rápidamente. Otras aplicaciones potenciales vinculadas a la seguridad de Estados Unidos incluían la detección acústica a gran distancia de submarinos nucleares usando sonares.

Richard Garwin, de IBM, era otro de los asistentes a la reunión; cuando Tukey le mostró su idea inmediatamente vislumbró el amplio rango de aplicaciones. Decidido a implementarla, se dirigió al centro de cómputos de IBM Research en Yorktown Heights, Nueva York, donde James W. Cooley era un miembro relativamente reciente del plantel. Según algunas versiones (Brigham, 1974), Cooley fue el encargado de programar el algoritmo porque, según sus propias palabras, "no tenía nada importante que hacer". Según otras versiones (Burk *et al.*, 2004), Garwin tuvo que insistir para que Cooley dejase de lado otros proyectos en los que estaba trabajando, y se dedicase a programar ese algoritmo. Debido al secreto impuesto por el Comité, Garwin hizo creer a Cooley que el programa se utilizaría para calcular la periodicidad de las orientaciones del spin de los átomos en un cristal tridimensional de He3 (Cooley, 1992). En cualquier caso, Cooley lo implementó muy rápidamente, con la esperanza de sacarse ese proyecto de encima y poder seguir con su trabajo. Sin embargo, los pedidos de copias del programa y de instrucciones para ejecutarlo se empezaron a acumular. En algún momento se vislumbró la posibilidad de patentar el algoritmo, pero finalmente se decidió que fuese de dominio público. En 1965 apareció en *Mathematics of Computation* el ahora famoso "An algorithm for the machine calculation of complex Fourier series" (Fig. 11.40) por Cooley y Tukey; fue publicado rápidamente (en menos de ocho meses), y el artículo junto con el proselitismo de Garwin hicieron que el algoritmo se difundiera muy rápidamente. Sin la tenacidad de Garwin, es posible que la FFT siguiese siendo desconocida aún hoy en día. Aunque según reza la introducción al algoritmo de la FFT en *Numerical Recipes* –un libro clásico sobre algoritmos computacionales– "si se acelera cualquier algoritmo no trivial por un factor de un millón todo el mundo buscará la forma de encontrarle aplicaciones útiles".

Este trabajo fue un hito en la literatura sobre transformadas de Fourier, y se le ha atribuido (Oppenheim y Schafer, 1975; Rabiner y Rader, 1972) el tremendo incremento en las aplicaciones y el desarrollo del procesamiento digital de señales que comenzó alrededor de 1970:

> "A paper by Cooley and Tukey described a recipe for computing Fourier coefficients of a time series that used many fewer machine operations than did the previous procedure... What lies over the horizon in digital signal processing is anyone's guess, but I think it will surprise us all." (Bogert, 1967)

El algoritmo es apropiado para el cálculo de TDF de longitud $N = N_1 N_2$, y es del tipo que derivó Gauss 150 años antes. En su artículo, los autores daban un ejemplo en el cual $N = 2^\alpha$, derivando lo que hoy en día se conoce como algoritmo de decimación en tiempo de raíz 2; de allí data la errónea creencia que la FFT de Cooley y Tukey correspondía solamente a este caso.

Luego que Cooley y Tukey publicaran sus descubrimientos, comenzaron a aparecer reportes de otros investigadores que habían utilizado técnicas similares (Cooley *et al.*, 1967). P. Rudnick (1966) de la Oceanic Institution, en La Jolla, California, reportó que había estado

An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

Consider the problem of calculating the complex Fourier series

$$X(j) = \sum_{k=0}^{N-1} A(k) \cdot W^{jk}, \qquad j = 0, 1, \cdots, N-1, \tag{1}$$

where the given Fourier coefficients $A(k)$ are complex and W is the principal Nth root of unity,

$$W = e^{2\pi i/N}. \tag{2}$$

A straightforward calculation using (1) would require N^2 operations where "operation" means, as it will throughout this note, a complex multiplication followed by a complex addition.

The algorithm described here iterates on the array of given complex Fourier amplitudes and yields the result in less than $2N \log_2 N$ operations without requiring more data storage than is required for the given array A. To derive the algorithm, suppose N is a composite, i.e., $N = r_1 \cdot r_2$. Then let the indices in (1) be expressed

$$\begin{aligned} j &= j_1 r_1 + j_0, \qquad j_0 = 0, 1, \cdots, r_1 - 1, \qquad j_1 = 0, 1, \cdots, r_2 - 1, \\ k &= k_1 r_2 + k_0, \qquad k_0 = 0, 1, \cdots, r_2 - 1, \qquad k_1 = 0, 1, \cdots, r_1 - 1. \end{aligned} \tag{3}$$

Then, one can write

$$X(j_1, j_0) = \sum_{k_0} \sum_{k_1} A(k_1, k_0) \cdot W^{jk_1 r_2} W^{jk_0}. \tag{4}$$

Received August 17, 1964. Research in part at Princeton University under the sponsorship of the Army Research Office (Durham). The authors wish to thank Richard Garwin for his essential role in communication and encouragement.

Fig. 11.40. Portada del artículo de Cooley y Tukey "An Algorithm for the Machine Computation of Complex Fourier Series," *Mathematics of Computations*, **19**, pp. 297-301, 1965.

usando una técnica similar, que había adaptado de un trabajo publicado en 1942 por Danielson y Lanczos. Estos autores mostraban cómo reducir una transformada de $2N$ puntos en dos transformadas de N puntos, utilizando solamente N operaciones adicionales. Hoy en día el tamaño y el tiempo de cálculo de su problema causan estupor:

> "Adopting these improvements the approximate times for Fourier analysis are 10 minutes for 8 coefficients, 25 minutes for 16 coefficients, 60 minutes for 32 coefficients, and 140 minutes for 64 coefficents".

A su vez, este artículo citaba un par de trabajos de Runge (1903 y 1905); estos artículos, junto con las notas de curso de Runge y Konig (1964) describen esencialmente las ventajas computacionales del algoritmo de la FFT tal como se lo conoce hoy en día. El algoritmo de 1903 se concentraba en el caso en que N era potencia de 2, y el artículo de 1905 extendía la idea para potencias de 3. Fue utilizado ampliamente alrededor de 1940, de manera que fue bastante bien conocido. De todos modos, desapareció después de la Segunda Guerra Mundial.

L. H. Thomas, un estadístico del IBM Watson Laboratory también había desarrollado una técnica eficiente para el cálculo de la TDF (Thomas, 1963), que había encontrado en un libro de Stumpf (1939). Thomas generalizó los resultados de Stumpf y derivó una técnica muy similar a la de la FFT. También Good, en 1960, extendió una idea de Yates (1937) para calcular la interacción de 2^α experimentos, y obtuvo un procedimiento para calcular las transformadas de Fourier de N puntos que es esencialmente equivalente al de Thomas.

Respecto a los desarrollos posteriores, Yavne (1968) presentó un algoritmo –en un artículo poco difundido– que requiere el menor número conocido de multiplicaciones y de sumas para FFTs de longitud 2^α, tanto para datos reales como complejos. Esta marca se mantuvo hasta hace muy poco tiempo, como se comenta más adelante, al menos para algoritmos prácticos. El mismo número de operaciones fue obtenido posteriormente por otros algoritmos más sencillos; debido al estilo críptico de Yavne pocos investigadores pudieron aplicar sus ideas en la época de su publicación.

En 1976 Rader y Brenner presentaron un algoritmo en el cual todos los factores eran reales o imaginarios puros (es decir, no necesitaba de los factores de rotación o *twiddle factors*), lo que permite disminuir sensiblemente el número de operaciones, a costa de aumentar el número de sumas y de una mayor sensitividad al ruido de redondeo.

Tomó casi 15 años la aparición de nuevos algoritmos para FFTs de largo 2^α que requirieran tan pocas operaciones como el de Yavne. En 1984 cuatro trabajos se presentaron casi simultáneamente (Duhamel y Hollmann, 1984; Martens, 1984; Stasinski, 1984; Vetterli y Nussbaumer, 1984) que presentaban las bases de los métodos de "raíces partidas", los que se basan en utilizar un algoritmo de raíz 2 para la parte par de la señal y uno de base 4 para la parte impar.

El trabajo de Winograd en teoría de complejidad (1977), particularmente sobre el número de multiplicaciones requeridas para calcular el producto o la convolución de polinomios, permitió aplicar los resultados previos de Good (1960) y de Rader (1968), quien mostró cómo calcular una FFT de largo N primo en una convolución circular de largo $N-1$. Si bien estos resultados fueron considerados como meras curiosidades al momento de su publicación, la combinación de los mismos, primero por Winograd (1976) y luego por Kolba y Parks (1977) despertó un gran interés en este tipo del algoritmos, que se conocen colectivamente como Winograd Fourier Transforms Algorithm (WFTA), y que requieren el

menor número de multiplicaciones para TDFs de longitud moderada. La desventaja es que este procedimiento, lo mismo que uno muy similar conocido como algoritmo de factores primos, se basan en conceptos matemáticos muy avanzados, ocasionando cierta demora y esfuerzo para trasladar estos resultados teóricos a algoritmos computacionales eficientes.

La búsqueda de mejores prestaciones sigue en la actualidad. El último algoritmo conocido fue presentado en 2006 (Johnson y Frigo, 2006): es una variación del algoritmo de raíces partidas y según sus autores utiliza un 6 % menos de operaciones que el algoritmo de Yavne.

Revisando esta breve historia resulta llamativo que los algoritmos anteriores al de Cooley y Tukey hayan desaparecido o permaneciesen ocultos, mientras que éste tuvo enorme repercusión. Una explicación posible es que el interés en los aspectos teóricos del procesamiento digital de señales fue creciendo junto con las mejoras tecnológicas en la industria de los semiconductores, ya que la disponibilidad de mayor potencia de cómputo a costo razonable permite desarrollar muchas aplicaciones novedosas.

A su vez, el avance de la tecnología ha influido en el diseño del algoritmo: mientras que en los años 60 y principios del 70 el interés se centraba en disminuir el número de multiplicaciones, el progreso reveló que también son de importancia fundamental el número de sumas y de accesos a memoria (desde el punto de vista del software) y los costos de comunicación (desde el punto de vista del hardware). Por ejemplo, algoritmos modernos como FFTW (Frigo y Johnson, 2005) no implementan el método de Winograd porque a pesar de utilizar el menor número de multiplicaciones, el arreglo de los datos y otras particularidades de implementación lo hacen relativamente ineficiente con los procesadores actuales. Algunos procesadores modernos incluyen un módulo optimizado para el cálculo por hardware de la DTF, como se muestra en la Fig. 11.41.

11.10. Referencias bibliográficas

Bailey, D. H. “FFTs in external or hierarchical memory”, *J. Supercomput.*, **4**, 1, pp. 2335, 1990.

Blahut, R. E., *Fast Algorithms for Digital Signal Processing*, Adisson-Wesley Publishing Company, Reading, MA, 1985.

Bluestein, L. I., “A Linear Filtering Approach to the Computation of Discrete Fourier Transform,” *IEEE Trans. Audio Electroacoustics*, AU-18, pp. 451-455, 1970.

Bogert, B. P. *IEEE Trans. Audio Electronics*, AU-15, **2**, p. 43, 1967.

Brigham, E. O. *The fast Fourier Transform*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, EE.UU., 1974.

Burk, P., L. Polansky, D. Repetto, M. Roberts y D. Rockmore, *Music and Computers - A Theoretical and historical Approach*, Key College Publishing, Emeryville, California, EE. UU., 2004.

Burrus, C. S., “Index Mappings for Multidimensional Formulation of the DFT and Convolution,” *IEEE Trans. on Acoustics, Speech and Signal Processing*, ASSP-25, pp. 239-242, 1977.

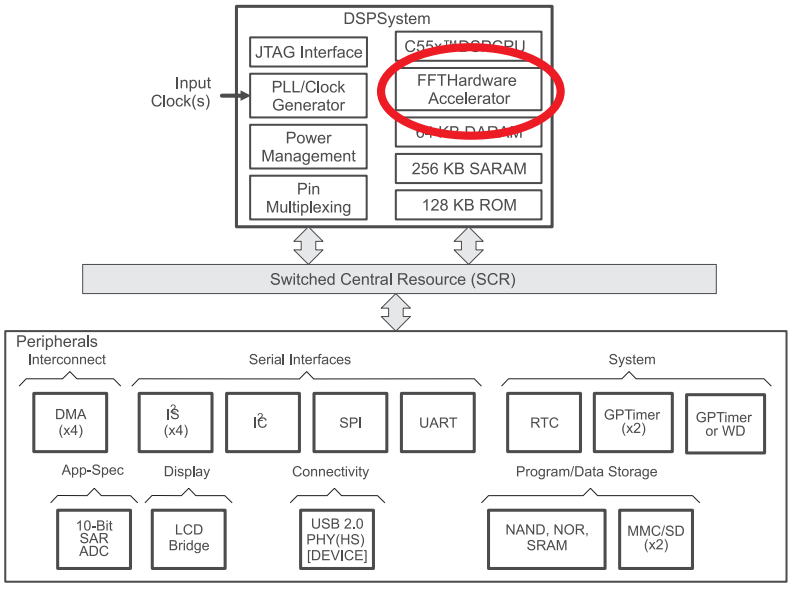
1 Key Message: Low-Power Device for Portable Audio/Medical Devices

The TMS320VC5505 device is a DSP that is optimized for applications that require low power and need a rich peripheral set. This device is the industry's best combination of standby (under 0.34 mW @ 1.05 V at 25°C) and active power (under 0.3 mW/MHz @ 1.05 V at 25°C), enabling longer battery life and making it the ideal choice for a wide range of portable device applications.

1.1 Key Features:

- Low-Power TMS320C55x™ Fixed-Point Digital Signal Processor
  - Software-Compatible With C55x™ Devices
  - 60-, 100-MHz Clock Rate
- 320K Bytes On-Chip RAM
- 128K Bytes On-Chip ROM
- 1.05-V Core (60 MHz), 1.8-/2.5-/2.8-/3.3-V I/Os
- 1.3-V Core (100 MHz), 1.8-/2.5-/2.8-/3.3-V I/Os
- Power Management/Power Savings
  - Real-Time Clock (RTC) With Crystal Input, Separate Clock Domain and Power Supply
  - Dynamic Voltage & Frequency Scaling
- 196-Terminal, Pb-Free Plastic BGA (ZCH)
- Applications:
  - Wireless Audio Devices (e.g., Headsets, Microphones, Speakerphones, etc.)
  - Echo Cancellation Headphones
  - Portable Medical Devices
  - Voice Applications
  - Industrial Controls
  - Fingerprint Biometrics
  - Software Defined Radio

1.2 TMS320VC5505 Functional Block Diagram



Note: Not all peripherals are available at the same time due to pin multiplexing.

Fig. 11.41. Información preliminar (junio 2009) del DSP TMS320VC5505 que incluye un acelerador por hardware para el cálculo de TDF de 1024 puntos para datos reales o complejos de 16 bits.

Burrus, C. S., "Efficient Fourier Transform and Convolution Algorithms," in *Advanced Topics in Signal Processing*, J. S. Lim y A. V. Oppenheim, Eds., Prentice Hall, Englewood Cliffs, NJ, 1988.

Cooley, C. W., P. S. Lewis y P. D. Welch, "Historical Notes on the Fast Fourier Transform," *IEEE Trans. Audio Electroacoustics*, AU-15, pp. 76-79, 1967.

Cooley, C. W. y J. W. Tukey, "An Algorithm for the Machine Computation of Complex Fourier Series," *Mathematics of Computations*, **19**, pp. 297-301, 1965.

Cooley, J. W., R. L. Garwin, C. M. Rader, B. P. Bogert, y T. C. Stockham "The 1968 Arden House Workshop on fast Fourier transform processing," *IEEE Trans. on Audio and Electroacoustics*, AV-17, 2, pp. 66-76, 1969.

Cooley, J. "How the FFT gained acceptance", *IEEE Signal Processing Magazine*, **9**, 1, pp. 10-13, 1992.

Danielson, G. C., y C. Lanczos, "Some Improvements in Practical Fourier Analysis and their Application to X-Ray Scattering from Liquids," *J. Franklin Inst.,* **233**, pp. 365-380 y 435-452, 1942.

DSP Committee, IEEE ASSP Eds., *Programs for Digital Signal Processing*, IEEE Press, New York, NY, 1979.

Duhamel, P. y H. Hollmann, "Split-radix FFT algorithm," *Electron. Lett.*, **20,** 1, 14-16, 1984.

Duhamel, P. y M. Vetterli, "Fast Fourier Transforms: A Tutorial Review and a State of the Art," *Signal Processing*, **19**, pp. 259-299, 1990.

Frigo, M. y S. G. Johnson, "FFTW: An Adaptive Software Architecture for the FFT," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, **3**, pp. 1381-1384, 1998.

Frigo, M. y S. G. Johnson, "The design and implementation of FFTW3," *IEEE Proceedings*, **93**, 2, pp. 216-231, 2005.

Ganapathiraju, A, J. Hamaker, A. Skjellum y J. Picone, "Contemporary View of FFT Algorithms," *Proceedings of the IASTED International Conference on Signal and Image Processing*, Las Vegas, Nevada, EE.UU., pp. 130-133, October 1998.

Goertzel, G., "An Algorithm for the Evaluation of Finite Trigonometric Series," *American Math. Monthly*, **65**, pp. 34-35, Jan. 1958.

Good, I. J. "The Interaction Algorithm and Practical Fourier Analysis," *J. Royal Statistical Society, Ser. B.*, **20**, 361-372, 1960.

Heideman, M. T., D. H. Johnson y C. S. Burrus, "Gauss and the History of the Fast Fourier Transform," *IEEE ASSP Magazine*, **1**, 4, pp. 14-21, 1984.

Hewes, C. R., R. W. Broderson y D. D. Buss, "Applications of CCD and Switched Capacitor Filter Technology," *Proc. IEEE*, **67**, 10, pp. 1403-1415, 1979.

Ingle, V. K. y J. G. Proakis, *Digital Signal Processing using MATLAB*, Bookware Companion Series, Brooks/Cole, 2000.

Johnson, S. G. y M. Frigo, "A modified split-radix FFT with fewer arithmetic operations," *IEEE Trans. Signal Processing*, in press, 2006.

Kolba, D.P. y T. W. Parks, "A prime factor algorithm using high-speed convolution," *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-25, pp. 281-294, 1977.

Martens, J. B., "Recursive cyclotomic factorization-A new algorithm for calculating the discrete Fourier transform", *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-32, 4, pp. 750-761, 1984.

McClellan, J. H., y C. M. Rader, *Number Theory in Digital Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, EE. UU., 1979.

Oppenheim, A. V.y R. W. Schafer *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, EE. UU., 1975.

Oppenheim A. V., y R. W. Schafer, *Discrete-Time Signal Processing*, 1ra. Edición, Prentice Hall, Englewood Cliffs, New Jersey, EE. UU., 1988.

Porat, B., *A Course on Digital Signal Processing*, John Wiley & Sons, 1997.

Press, W. H., B. P. Flannery, S. A. Teukolsky, y W. T. Vettering *Numerical recipes in C: The art of scientific computing*, Cambridge University Press, New York, EE. UU.,1988.

Rabiner, L. R., R. W. Schafer y C. M. Rader, "The Chirp *Z*-Transform Algorithm," *IEEE Trans. Audio Electroacoustics*, AU-17, pp. 86–92, 1969.

Rabiner, L. R. y C. M. Rader, Ed., *Digital Signal Processing,* IEEE Press, New York, EE. UU., 1972.

Rabiner, R. L. "The Chirp z-transform algorithm–A lesson in serendipity," *IEEE Signal Processing Magazine*, **21**, 2, pp. 118-119, 2004.

Rader, C. M., "Discrete Fourier Transforms when the Number of Data Samples is Prime," *Proceedings of the IEEE*, **56**, pp. 1107-1108, 1968.

Rader, C. M. y Brenner, N .M. "A new principle for fast Fourier transformation," *IEEE Trans. Acoust. Speech Signal Process.*, ASSP-24, pp. 264-265, 1976.

Rudnick, P. "Notes on the Calculation of Fourier Series," *Mathematics of Computation*, **20**, pp. 429-430, 1966.

Runge, C., "Über die Zerlegung Empirisch Gegebener Periodischer Functionen in Sinuswellen," *Z. Math Physik*, **48**, pp. 443-456, 1903; **53**, pp. 117-123, 1905.

Runge C., y H. König, "Die Grundlehren der Mathematischen Wissenschaften," *Vorlesungen uber Numerischen Rechnen*, **11**. J. Springer, Berlin, Alemania, 1924.

Singleton, R. C., "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," *IEEE Trans. Audio Electroacoustics*, AU-17, pp. 93-103, 1969.

Skolnik, M. I., *Introduction to Radar Systems*, McGraw Hill, segunda edición, New York, NY, 1986.

Stumpff, K., *Tafeln und Aufgabenfür Harmonischen Analyse und Perrodogram-mechnung,* J. Springer, Berlin, Alemania, 1939.

Swarztrauber, P. N. "Vectorizing the FFTs," in *Parallel Computations*, G. Rodrigue, Ed., Academic Press, New York, EE.UU., pp. 5183, 1982.

Takahashi, D. "A blocking algorithm for FFT on cache-based processors," in *Lecture Notes in Computer Science, High-Performance Computing and Networking*. Springer-Verlag, Heidelberg, Alemania, **2110**, pp. 551-554, 2001.

Thomas, L. H. "Using a Computer to Solve Problems in Physics," *Applications of Digital Computers,* Guin, Boston, Massachussetts, EE. UU., 1963.

Vetterli, M. y H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Process.*, **6**, 4, pp. 267-278, 1984.

Winograd, S. "On computing the discrete Fourier transform," *Proc. Nat. Acad. Sci. USA*, **73**, pp. 1005-1006, 1976.

Winograd, S. "On Computing the Discrete Fourier Transform," *Mathematics of Computations*, **32**, 141, pp. 175-199, 1978.

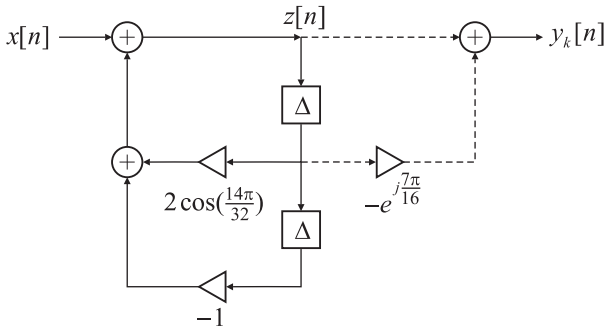
Yates, F. "The Design and Analysis of Factorial Experiments," *Commonwealth Agriculture Bureaux,* Farnam Royal, Burks, Inglaterra, 1937.

11.11. Ejercicios

Ejercicio 1. La señal $x[n]$ satisface $|x[n]| \leq 1$, para $0 \leq n \leq N-1$. Muestre que el máximo valor posible de $|X[k]|$ es N. Encuentre todas las sucesiones $x[n]$ para las cuales $|X[k]| = N$ para algún k.

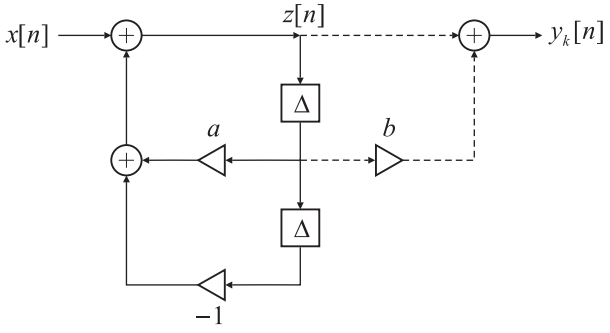
(I) **Ejercicio 2.** Se dispone de un programa para calcular la TDF, es decir, que dada una señal $x[n]$ de entrada, el programa entrega como salida la TDF $X[k]$ de $x[n]$. Muestre cómo puede utilizarse *el mismo programa* para calcular la transformada inversa: utilizando como entrada $X[k]$, o una sucesión relacionada simplemente con $X[k]$, el programa debe dar como salida la sucesión temporal $x[n]$ o una sucesión relacionada de manera simple con ella.

Ejercicio 3. La entrada al sistema de la figura es una sucesión $x[n]$ de largo $N = 32$, definida para $0 \leq n \leq 31$. La salida $y[n]$ en $n = 32$ es igual a $X\left(e^{j\omega}\right)$ evaluada a una frecuencia ω_k específica. Determine ω_k de acuerdo a los coeficientes del filtro de la figura.



Ejercicio 4. Una sucesión $x[n]$ real de largo $N = 32$ se invierte temporalmente y se demora, obteniéndose la sucesión $x_1[n] = x[32-n]$. Si esta sucesión se usa como entrada al filtro del Ejercicio 3, encuentre una expresión para $y[32]$ en función de la transformada de Fourier $X\left(e^{j\omega}\right)$ de la sucesión $x[n]$ original.

Ejercicio 5. Para el diagrama bloque de la figura, elija los valores de a y b de modo que $y[8] = X\left(e^{j6\pi/8}\right)$ si la entrada $x[n]$ es una sucesión de $N = 8$ puntos.



Ejercicio 6. Se desea calcular la segunda muestra ($X[1]$) de la TDF $X[k]$ de la sucesión $x[n] = \{0, 1, 1\}$ utilizando el algoritmo de Goertzel. Para ello,

1. Derive una expresión que permita el cálculo de una muestra de la TDF a partir de la convolución entre $x[n]$ y una sucesión $h[n]$ apropiada.
2. Dibuje el diagrama bloque de un sistema que efectúe la convolución del inciso previo (es el diagrama bloque del algoritmo de Goertzel de primer orden)
3. Calcule $X[1]$ a partir de la ecuación a diferencias sugerida por el diagrama bloque del inciso anterior.

Ⓒ **Ejercicio 7. Implementación del algoritmo de Goertzel**

El algoritmo de Goertzel requiere aproximadamente N^2 operaciones para calcular una TDF de orden N, aproximadamente la mitad de las multiplicaciones requeridas por los métodos que implementan la fórmula de la TDF. Si a partir de la sucesión de datos $x[n]$ se forma el polinomio

$$X(z) = \sum_{n=0}^{N-1} x[n] z^n, \tag{11.64}$$

el k-ésimo valor de la TDF de $x[n]$ se puede hallar evaluando $X(z)$ en $z = e^{-j\frac{2\pi}{N}k}$. Un método eficiente para evaluar el polinomio es el método de Horner, también conocido como "evaluación anidada". El método de Horner se basa en agrupar las operaciones de una manera especial; por ejemplo, para una sucesión de 5 muestras de longitud,

$$X(z) = (((x[4]z + x[3])\, z + x[2])\, z + x[1])\, z + x[0]. \tag{11.65}$$

Esta secuencia de operaciones puede ser escrita de manera recursiva como:

$$y[m] = z\, y[m-1] + x[N-m], \tag{11.66}$$

con condición inicial $y[0] = 0$. El polinomio a evaluar es la solución de la ecuación a diferencias en $m = N$:

$$X(z) = y[N]. \tag{11.67}$$

El valor de la TDF $X[k]$ coincide con $y[n]$ cuando $z = e^{-j\frac{2\pi}{N}k}$ y $n = N$; es decir, $X[k]$ puede pensarse como la salida de un filtro IIR de primer orden con un polo complejo en $z = e^{-j\frac{2\pi}{N}k}$, tomando como entrada la secuencia de datos ordenada al revés (el último dato al comienzo de la secuencia). El valor de la TDF es la salida del filtro *después de N iteraciones*.

El algoritmo de Goertzel de segundo orden permite reducir el número de operaciones requeridas convirtiendo el filtro de primer orden en uno de segundo orden de manera de eliminar la multiplicación compleja en (11.66), multiplicando el numerador y denominador de la función transferencia del filtro de primer orden por $z - e^{j\frac{2\pi}{N}k}$. Para reducir el número de multiplicaciones es importante *no* implementar el numerador sino recién en la última iteración.

Nota: En este ejercicio se utilizará el comando `polyval`, el comando `filter`, y un programa o m-file escrito para implementar el ecuación a diferencias del algoritmo de Goertzel.

1. **El método de Horner.** Verifique que las ecuaciones (11.66) y (11.67) son implementaciones del método de evaluación de Horner (11.65). Escriba un programa MATLAB para calcular la TDF de una secuencia usando el comando `polyval` para evaluar (11.64) en $z = e^{-j\frac{2\pi}{N}k}$. Después que haya verificado que el programa funciona correctamente, coloque el programa dentro de un lazo para evaluar todos los N valores de la TDF. Escriba una versión de este programa que no utilice un lazo, sino que llame a la función `polyval` con un argumento vectorial para hacer todos los cálculos simultáneamente. Mida los flops de ambas versiones para diferentes valores de N. Compare los resultados con los del Ejercicio 26.

2. **Usando el comando filter.** Escriba un programa que calcule la TDF en un valor k usando el comando `filter`. Después que verifique que el programa funciona correctamente, coloque el programa dentro de un lazo para evaluar los N valores de la TDF. Compare los tiempos de ejecución y los flops para distintos valores de N con los resultados obtenidos para el método de evaluación directa, y con los resultados del inciso anterior. Explique las diferencias.

3. **Usando una ecuación a diferencias.** Para comprender mejor la implementación del algoritmo de Goertzel, escriba un programa que implemente la ecuación a diferencias (11.66), en lugar de usar el comando filter. Los valores de la TDF deben ser los mismos que los obtenidos en la implementación usando `filter` en el inciso anterior, o los métodos directos usados en el Ejercicio 26. Después de verificar que el programa funciona correctamente, colóquelo dentro de un lazo (o escríbalo de manera que acepte un argumento vectorial) para evaluar todos los N valores de la TDF. Compare los flops requeridos y los tiempos de ejecución necesarios con los obtenidos en los incisos (a) y (b).

4. **Método de Goertzel de segundo orden.** El método de Goertzel de primer orden puede convertirse en un método de segundo orden que sólo utiliza multiplicaciones reales, disminuyendo el número de operaciones requeridas. Escriba una m-file que implemente una realización de Goertzel de segundo orden, y evalúe el tiempo de cálculo y el número de operaciones necesarias (aproximadamente el mismo número de sumas, y la mitad de las multiplicaciones).

5. **Datos reales y complejos.** En general, cuando las sucesiones a estudiar son reales, es ineficiente utilizar un programa general de cálculo de TDF que puede trabajar sobre señales de entrada complejas. Evalúe el número de operaciones requeridas para calcular la TDF de datos reales y de datos complejos para los algoritmos de Goertzel de primer y segundo orden.

(I) **Ejercicio 8.** Se tienen dos sucesiones $x[n]$ e $y[n]$, cada una de largo 2^r. Calcule el número de multiplicaciones reales que se necesitan para calcular la convolución lineal de las dos sucesiones,

1. Por convolución directa

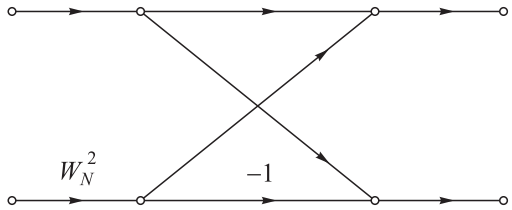
2. Utilizando FFT de raíz 2 de la manera más eficiente

Ejercicio 9. La señal $x[n]$ tiene 1021 muestras no nulas. Se desea estimar su transformada de Fourier $X\left(e^{j\omega}\right)$ utilizando la TDF $X[k]$. Un programa para calcular la TDF de 1021 puntos de $x[n]$ demora 100 segundos en calcular $X[k]$. Si se forma la sucesión $x_1[n]$ agregando tres ceros a la sucesión $x[n]$, el mismo programa demora 1 segundo para calcular $X_1[k]$, la TDF de 1024 puntos de $x_1[n]$. Note que, agregando tres ceros a la sucesión original, pudo obtener un mayor número de muestras de $X\left(e^{j\omega}\right)$ en mucho menos tiempo. ¿Cómo explica esta aparente paradoja?

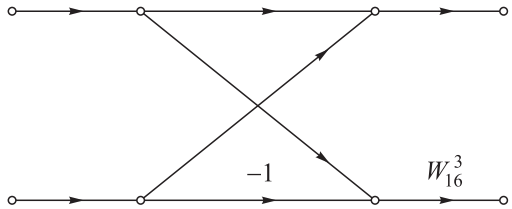
Ejercicio 10. Construya el diagrama de flujo de señales para un algoritmo de FFT de 16 puntos (de raíz 2), utilizando la técnica de decimación en tiempo. Indique los valores de todos los multiplicadores como potencias de $W_{16} = e^{-j\frac{2\pi}{26}}$. Indique en los nodos de entrada y salida los índices de las sucesiones $x[n]$ y $X[k]$. Calcule el número de sumas reales y productos reales necesarios para implementar el algoritmo.

Ejercicio 11. La "mariposa" de la figura es parte de un diagrama de flujo de señales de una implementación de la FFT. ¿Cuál de estas proposiciones es correcta?

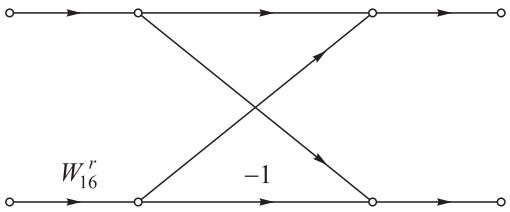
1. La mariposa es parte de un algoritmo de decimación en tiempo.
2. La mariposa es parte de un algoritmo de decimación en frecuencia.
3. No se puede asegurar de qué tipo de algoritmo forma parte la mariposa.



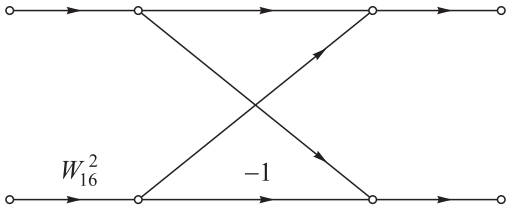
Ejercicio 12. La mariposa de la figura forma parte de un algoritmo de FFT de decimación en frecuencia, con $N = 16$, donde la sucesión de entrada $x[n]$ está ordenada normalmente. Sabiendo que una FFT de $N = 16$ necesita de 4 etapas, indizadas desde $m = 1$ hasta $m = 4$, determine y justifique cuál de las cuatro etapas utiliza mariposas de esta forma.



Ejercicio 13. La mariposa de la figura forma parte de un algoritmo de FFT de decimación en tiempo, con $N = 16$. Si cada una de las etapas del grafo de señales se indizan como $m = 1, \ldots, 4$, ¿cuáles son los posibles valores de r para cada etapa?



Ejercicio 14. La mariposa de la figura forma parte de un algoritmo de FFT de decimación en tiempo, con $N = 16$. Si cada una de las etapas del grafo de señales se indizan como $m = 1, \ldots, 4$, ¿cuál de las cuatro etapas utiliza mariposas de esta forma?



Ejercicio 15. Para calcular la TDF de una sucesión $x[n]$ de largo $N = 2^v$ se cuenta con dos programas. El programa "A" calcula la TDF aplicando directamente la definición, demorando N^2 segundos. El programa "B" efectúa el cálculo utilizando una FFT por decimación en tiempo, y demora $10N \log_2 N$ segundos. ¿Cuál es el menor largo N de la sucesión $x[n]$ tal que el programa "B" demora menos que el programa "A"?

Ejercicio 16. Ordene las muestras de una sucesión $x[n]$ de largo $N = 16$ para aplicar el algoritmo de cálculo de la TDF por decimación en tiempo.

(I) **Ejercicio 17.** Se dispone de un programa para calcular la TDF de una sucesión de cualquier longitud N que sea potencia de 2, es decir $N = 2^v$. Se desea calcular la transformada de Fourier $X\left(e^{j\omega}\right)$ de una sucesión $x[n]$ (que se anula para $n < 0$ y $n > 626$) en los valores de frecuencia $\omega_k = \frac{2\pi}{627} + \frac{2\pi}{256}k$, $k = 0, \ldots 255$. Muestre cómo generar una nueva sucesión $y[n]$ a partir de $x[n]$ tal que, aplicando el programa disponible a $y[n]$, se puedan obtener las muestras en frecuencia deseadas, con v tan pequeño como sea posible.

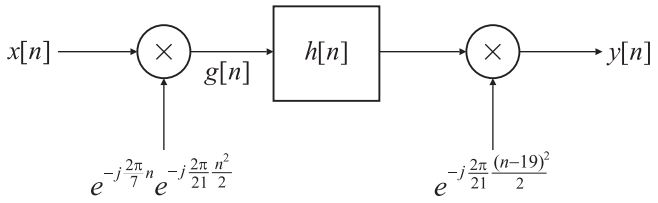
Ejercicio 18. Una sucesión compleja $x[n]$ de longitud finita tiene una TDF de N puntos $X[k]$, y satisface la condición de simetría $x[n] = -x[((n + N/2))_N]$ si N es par.

1. Muestre que $X[k] = 0$ para $k = 0, 2, 4, \ldots N - 2$.
2. Especifique cómo calcular las muestras impares de $X[k]$, $k = 1, 3, 5, \ldots N - 1$, utilizando solamente una TDF de $N/2$ puntos y una pequeña cantidad de cálculo adicional.

Ejercicio 19. Una señal $x[n]$ de longitud finita es no nula en el intervalo $0 \leq n \leq 19$. Esta señal es la entrada al sistema de la figura, para el cual

$$h[n] = \begin{cases} e^{j\frac{2\pi}{21}\frac{(n-19)^2}{2}}, & n = 0, 1, \ldots, 28, \\ 0, & \text{en caso contrario.} \end{cases} \qquad \text{y} \quad W = e^{-j\frac{2\pi}{21}}.$$

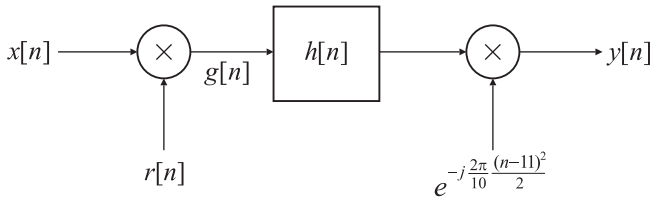
La salida $y[n]$ del sistema en el intervalo $n = 19, \ldots, 28$ se puede expresar en función de $X\left(e^{j\omega}\right)$ para valores apropiados de ω. Escriba una expresión para $y[n]$ en este intervalo en función de $X\left(e^{j\omega}\right)$.



Ejercicio 20. El sistema que se muestra en la figura tiene respuesta impulsiva

$$h[n] = \begin{cases} e^{j\frac{2\pi}{10}\frac{(n-11)^2}{2}}, & n = 0, 1, \ldots, 15, \\ 0, & \text{en caso contrario.} \end{cases}$$

Determine los valores de la sucesión $r[n]$ para que $y[n+11] = X\left(e^{j\omega_n}\right)$ con $\omega_n = (2\pi/19) + (2\pi/10)n$, $n = 0, \ldots 4$.



Ejercicio 21. La TDF de N puntos de una sucesión $x[n] = e^{-j\frac{\pi}{N}n^2}$ es $X[k] = \sqrt{N}e^{-j\frac{\pi}{4}}e^{j\frac{\pi}{N}k^2}$. Determine la TDF de $2N$ puntos de la sucesión de longitud $2N$ $y[n] = e^{-j\frac{\pi}{N}n^2}$. En todos los casos, suponga que N es par.

Ejercicio 22. Se dispone de un programa que calcula la FFT de una sucesión compleja. Si se desea calcular la TDF de una sucesión real, se puede especificar que la parte imaginaria de la sucesión es nula, y utilizar el programa directamente. Sin embargo, las propiedades de simetría de la TDF de una sucesión real pueden aprovecharse ventajosamente para disminuir el volumen de cálculo. En todos los casos se supone que N es par.

1. Sea $x[n]$ una sucesión real de longitud N, y sea $X[k]$ su TDF de N puntos, con parte real e imaginaria $X_R[k]$ y $X_I[k]$, respectivamente:

   $$X[k] = X_R[k] + jX_I[k].$$

   Muestre que si $x[n]$ es real, $X_R[k] = X_R[N-k]$ y $X_I[k] = -X_I[N-k]$, para $0 \leq k \leq N-1$.

2. Considere ahora dos sucesiones reales $x_1[n]$ y $x_2[n]$, con TDF $X_1[k]$ y $X_2[k]$, respectivamente. Construya la sucesión $y[n] = x_1[n] + jx_2[n]$, cuya TDF es $Y[k] = Y_R[k] + jY_I[k]$. Sean $Y_{OR}[k]$, $Y_{ER}[k]$, $Y_{OI}[k]$, $Y_{EI}[k]$ la parte impar de la parte real, la parte par de la parte real, la parte impar de la parte imaginaria, y la parte par de la parte imaginaria de $Y[k]$, respectivamente, donde $Y_E[k] = \frac{1}{2}\left(Y[((k))_N] + Y^*[((N-k))_N]\right)$, $Y_O[k] = \frac{1}{2}\left(Y[((k))_N] - Y^*[((N-k))_N]\right)$, o bien, para evitar el cálculo de los índices módulo N,

$$\begin{aligned} Y_{OR}[k] &= \tfrac{1}{2}\left(Y_R[k] - Y_R[N-k]\right), & Y_{ER}[k] &= \tfrac{1}{2}\left(Y_R[k] + Y_R[N-k]\right), \\ Y_{OI}[k] &= \tfrac{1}{2}\left(Y_I[k] - Y_I[N-k]\right), & Y_{EI}[k] &= \tfrac{1}{2}\left(Y_I[k] + Y_I[N-k]\right), \end{aligned}$$

para $1 \leq k \leq N-1$, y además $Y_{OR}[0] = Y_{OI}[0] = 0$, $Y_{ER}[0] = Y_R[0]$, $Y_{EI}[0] = Y_I[0]$. Determine expresiones para $X_1[k]$ y $X_2[k]$ en función de $Y_{OR}[k]$, $Y_{ER}[k]$, $Y_{OI}[k]$, $Y_{EI}[k]$.

3. Si $N = 2^v$ y se cuenta con un programa para el cálculo de la TDF utilizando un método de raíz 2, determine el número de multiplicaciones y sumas reales que se requieren para calcular $X_1[k]$ y $X_2[k]$:
   a) usando el programa un par de veces (con la parte imaginaria igualada a cero) para calcular las TDF de N puntos $X_1[k]$ y $X_2[k]$ separadamente;
   b) usando el esquema sugerido en el inciso 2, que necesita computar sólo una TDF de largo N.

4. Sea $x[n]$ una única sucesión real de largo N, donde N es potencia de 2, y sean $x_1[n]$ y $x_2[n]$ dos sucesiones reales de longitud $N/2$ formadas por las muestras pares e impares de $x[n]$, es decir, $x_1[n] = x[2n]$ y $x_2[n] = x[2n+1]$, con $n = 0, \ldots, N/2-1$. Calcule $X[k]$ en función de las TDF de $N/2$ puntos $X_1[k]$ y $X_2[k]$.

5. Utilizando los resultados de los incisos (2), (3), (4), describa un procedimiento para calcular la TDF de una sucesión real $x[n]$ utilizando solamente una TDF de $N/2$ puntos. Determine el número de multiplicaciones y sumas reales requeridas por este procedimiento, y compare con los necesarios para calcular $X[k]$ utilizando una única TDF de N puntos con la parte imaginaria de $x[n]$ igualada a cero.

Ejercicio 23. Sean $x[n]$ y $h[n]$ dos sucesiones reales de longitud finita, tales que $x[n] = 0$ para n fuera del intervalo $0 \leq n \leq L-1$, y $h[n] = 0$ fuera del intervalo $0 \leq n \leq P-1$. Se desea calcular la convolución lineal $y[n] = x[n] * h[n]$

1. ¿Cuál es la longitud de la sucesión $y[n]$?

2. Calcule el número de multiplicaciones reales que se necesitan para calcular la suma de convolución para todas las muestras no nulas de $y[n]$. La siguiente identidad puede ser de utilidad: $\sum_{k=1}^{N} k = N(N+1)/2$.

3. Describa un procedimiento que utilice la TDF para calcular todas las muestras no nulas de $y[n]$. Determine el menor tamaño de las TDF y de la TDF inversa en función de L y P.

4. Si $L = P = N/2$, donde $N = 2^v$ es el tamaño de la FFT de raíz 2, utilice los resultados del inciso (3) para calcular el número de multiplicaciones reales necesarias para computar los valores no nulos de $y[n]$. En base a los resultados del inciso (2), determine el valor mínimo de N para el cual el método de la TDF requiere menor cantidad de operaciones que la evaluación directa de la suma convolución.

Ejercicio 24. Los métodos overlap-add y overlap-save permiten implementar la convolución lineal de dos sucesiones $x[n]$ y $h[n]$ seccionando la entrada $x[n]$ en segmentos de longitud finita, y utilizando la TDF para calcular la convolución circular sobre estos segmentos y $h[n]$. Si las TDF se calculan utilizando algoritmos de FFT, estos métodos necesitan menor cantidad de operaciones por muestra de salida que la evaluación directa de la suma convolución.

1. Suponga que la sucesión de entrada $x[n]$ es compleja y de duración infinita, y que la respuesta impulsiva $h[n]$ es compleja y de longitud P, de forma tal que $h[n] \neq 0$ para $0 \leq n \leq P-1$. Suponga que la salida se calcula mediante el método overlap-save, con TDF de largo $L = 2^v$ que se calculan utilizando un algoritmo de FFT de raíz 2. Determine el número de operaciones complejas necesarias por muestra de salida en función de v y P.

2. Utilizando los resultados del inciso (1) grafique el número de operaciones por muestra de salida en función de v para $v \leq 20$, y $P = 500$. ¿Para qué valor de v el número de operaciones es mínimo? Compare los resultados obtenidos con el número de multiplicaciones complejas por muestra de salida que se requieren al evaluar directamente la suma convolución.

3. Muestre que para FFT de gran número de puntos el número de multiplicaciones complejas por muestra de salida es aproximadamente v. Esto indica que, más allá de una determinada longitud N de la TDF, el método overlap-save es *menos* eficiente que el método directo. Si $P = 500$, ¿para qué valor de v el método directo será más eficiente?

4. Si la longitud N de la FFT es el doble de la longitud de la respuesta impulsiva ($N = 2P$) y $P = 2^v$, aplique los resultados del inciso (1) para determinar el menor valor de P para el cual el método de overlap-save (usando la FFT) necesita menor cantidad de multiplicaciones complejas que la implementación directa de la convolución.

(M) **Ejercicio 25.** Sea $x[n] = \cos \pi n/99$, $0 \leq n \leq N-1$ una sucesión de N puntos. Calcule $X[k]$, la TDF de $x[n]$ eligiendo N como una potencia de 4, es decir, $N = 4^r$, y determine los tiempos de ejecución en MATLAB para $r = 5, 6, \ldots, 10$. Verifique que estos tiempos son proporcionales a $N \log_4 N$.

Ayuda: para medir los tiempos de ejecución son útiles las funciones `clock` y `etime`. El programa puede estructurarse de esta forma:

```
% t0 guarda la información del instante actual
t0 = clock;
% aqui se ejecuta la parte que se desea cronometrar
X = fft(x);
% t1 registra el intervalo de tiempo necesario para computar la fft.
t1 = etime(clock,t0);
```

Ⓒ **Ejercicio 26. Cálculo de la TDF por definición (*puede llevar mucho tiempo!*).**

La TDF $X[k]$ de una sucesión de longitud finita $x[n]$ se calcula como

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \quad k = 0, 1, \ldots N-1, \tag{11.68}$$

donde $W_N^{kn} = e^{-j\frac{2\pi}{N}kn} = \cos(2\pi kn/N) + j\,\mathrm{sen}(2\pi kn/N)$, $n, k = 0$, 1, ..., $N-1$, y se supone que la sucesión de entrada $x[n]$ es compleja. La operación indicada por la ecuación (11.68) puede implementarse en MATLAB de diferentes maneras.

1. Utilizando dos ciclos `for` anidados dentro de los cuales se ejecutan operaciones escalares consistentes en una multiplicación y una suma complejas:

   ```
   i = sqrt(-1);
   for k = 0:N-1;
     aux = 0;
     for n = 0:N-1;
       X(k+1) = aux + x(n+1)*exp(-i*2*pi*k*n/N);
     end;
   end;
   ```

   El argumento de `X` es `k+1`, y el de `x` es `n+1` para evitar problemas de indizado.

2. Usando un único ciclo `for` que calcula $X[k]$ para cada k efectuando el producto escalar de un vector de datos formando por N muestras de $x[n]$, y un vector formado por $W_N^{nk} = e^{-j\frac{2\pi}{N}nk}$, para $n = 0, 1, ..., N-1$, y k fijo. En este fragmento de código tanto `n` como `exp()` son vectores fila de longitud N.

   ```
   i = sqrt(-1);
   n = 0:N-1;
   for k = 0:N-1;
     WNk = exp(-i*2*pi*k*n/N);
     X(k+1) = (exp(-i*2*pi*k*n/N))*x';
   end;
   ```

3. Como un producto matricial donde la matriz $\mathbf{W} = \{W_N^{nk}\}_{n,k}$ puede calcularse "a mano", como en el fragmento de programa que sigue a continuación o bien utilizando los comandos `dftmtx` o `fft(eye(N))`:

   ```
   i = sqrt(-1);
   n = 0:N-1;
   k = 0:N-1;
   W = exp((-i*2*pi/N)*n*k'); % o es exp((-i*2*pi/N)*k*n') ???;
   X = W*x';
   end;
   ```

El propósito de este ejercicio es evaluar el tiempo de cálculo y al número de operaciones de punto flotante requeridas ("flops") de las distintas implementaciones de la TDF usando sucesiones $x[n]$ de distinta longitud N, con $2 \leq N \leq 2100$; Ya que sólo se desea determinar la influencia del *largo* de la sucesión en el tiempo de ejecución y número de operaciones de la TDF, $x[n]$ será una sucesión compleja de números aleatorios: `x = rand(1,N) + j*rand(1,N);`. El análisis del tiempo de ejecución y del número de operaciones puede efectuarse en MATLAB de la siguiente manera:

```
... (pasos previos de programa)
t0 = clock;
f0 = flops;
... (instrucciones a cronometrar y medir)
f1 = flops-f0;
t1 = etime(clock,t0)
```

Para la medición de tiempos, también pueden utilizarse los comandos `tic` y `toc`.

En MATLAB el direccionamiento de todos los vectores y arreglos comienza en 1, mientras que las fórmulas para la TDF comienzan en cero; esto se ha tenido en cuenta en los fragmentos de programa indicados más arriba. Debido a la forma en que MATLAB cuenta las operaciones en punto flotante, puede haber algunas diferencias en el número de "flops" utilizados si la TDF se implementa usando `exp()` o si se utiliza la forma de Euler, `cos()+j*sin()`. El comando `flops` cuenta *todas* las operaciones aritméticas: cálculo de las exponenciales, aritmética de los índices, y aritmética de los datos. (El comando flops no está disponible desde la versión 6 en adelante de MATLAB.)

1. **Un programa con dos ciclos.** Escriba una función en MATLAB que calcule el tiempo de cálculo y el número de flops ejecutados para calcular una TDF de longitud N usando dos ciclos `for` anidados, con el lazo interno sumando sobre n y el lazo externo indizando sobre k: `[t,f] = tf_fft2c(N)`. La función deberá generar internamente el vector aleatorio $x[n]$ de longitud N; cronometre y mida los flops *únicamente de la porción de la función que calcula la TDF*.

2. **Programa con un único ciclo.** Escriba una segunda función donde la TDF se calcule con un único ciclo que efectúe un producto entre vectores para cada valor de k: `[t,f] = tf_fft1c(N)`.

3. **Programa sin lazos.** Escriba una tercera función `[t,f] = tf_fftM(N)` que calcule el tiempo de cálculo y el número de flops ejecutados para calcular una TDF de longitud N utilizando una multiplicación matricial.

4. **Programa usando la función FFT de** MATLAB. Finalmente escriba una función `[t,f] = tf_fft(N)` que evalúe el tiempo de cálculo y el número de operaciones realizadas por la función `fft()` incluida en MATLAB.

5. **Tiempo y flops en función de la longitud.** Escriba un programa que invoque cada una de las funciones de los incisos (1)–(4), y que registre el tiempo de cálculo y el número de operaciones que insume cada una de ellas en sendos vectores, para diferentes valores de N desde $N = 2$ hasta $N = 600$. Como Windows no es un sistema preemptivo (es decir, el sistema operativo no otorga un tiempo fijo a cada tarea) el tiempo de cada rutina puede variar según la carga del sistema operativo. Por ello ejecute un promedio de 10 a 20 corridas de cada rutina antes de registrar el tiempo que demanda cada una. Grafique estos vectores en función del largo N de la TDF, y compare la eficiencia de cada una en términos de velocidad de cálculo y del número de operaciones. Compare los resultados con la dependencia tipo N^2 que predice la teoría.

Los resultados obtenidos deberían ser similares a los que se muestran en las Figs. 11.42 y 11.43. Los cálculos se ejecutaron en una PC tipo Pentium II de 233 MHz. En general, no hay grandes diferencias entre los tiempos requeridos por la implementación directa o la matricial, mientras que las implementaciones que usan lazos son mucho mas lentas.

Desde el punto de vista del número de operaciones, se aprecia que es mucho menor para la implementación interna o la matricial. También se observa que la implementación interna es en general mucho más eficiente que la matricial, excepto para el caso en que N es primo, en cuyo caso el número de operaciones es prácticamente el mismo (Fig. 11.43).

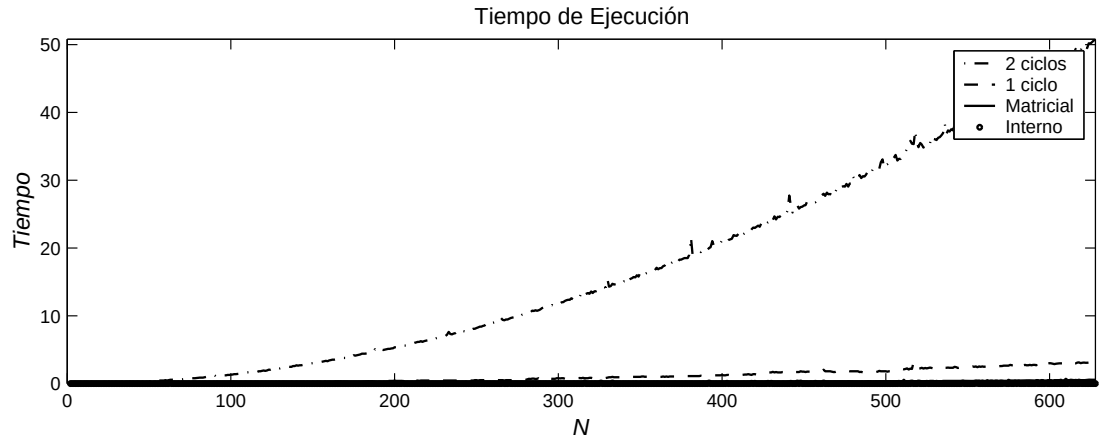


Fig. 11.42. Tiempo de ejecución de FFTs de diferente longitud N, según los distintos métodos.

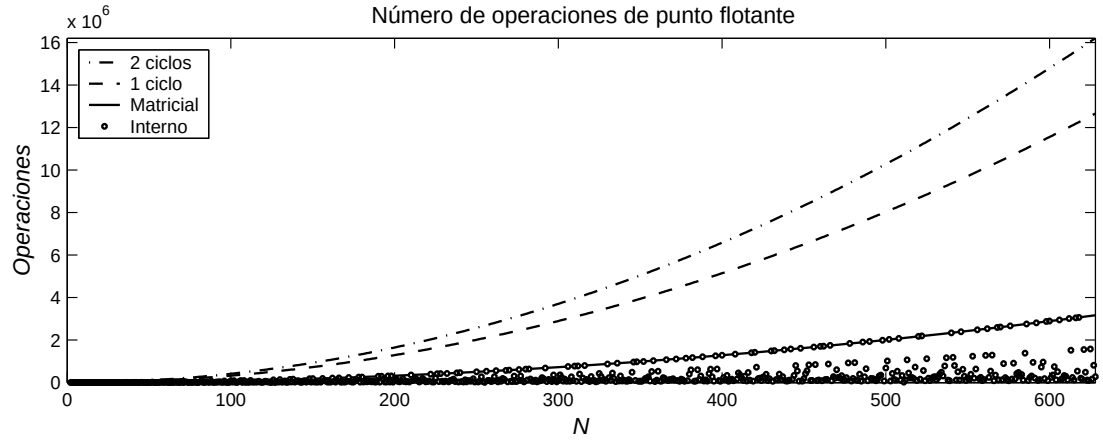


Fig. 11.43. Número de operaciones de punto flotante requeridas para el cálculo de la FFT para distintos métodos de implementación.

(C) Ejercicio 27. Decodificación de tonos DTMF

1. Escriba una función en Matlab que calcule la TDF utilizando el algoritmo de Goertzel, `Xk = gfft(x,k)`. La función debe recibir como argumentos la señal $x[n]$, y el número de muestra k donde desea calcularse la transformada $X[k]$. El valor de N (el orden de la TDF) de debe obtener a partir del largo de la sucesión de entrada, `n = length(x)`. Para implementar la ecuación recursiva del filtro puede utilizar el comando `filter`. Elija el tipo de filtro (de primer o segundo orden) que prefiera; si implementa la versión de segundo orden, optimice el número de operaciones evaluando el numerador de la función transferencia sólo cuando sea necesario.

2. Genere una sucesión cualquiera, por ejemplo, `x = rand(1,N)`, y compruebe su programa contrastando el resultado obtenido con el de calcular directamente `fft(x,N)`. Tenga en cuenta que la realización de la sucesión aleatoria cambia cada vez que se invoca el comando `rand()`, de modo que en lugar de anidar las funciones, es conveniente asignar su resultado a una variable (como se detalló arriba) y utilizar esta

variable como argumento de la función `fft` o `gfft`. En general, `fft(rand(1,N))` seguramente dará un resultado distinto de `gfft(rand(1,N)),k)`.

3. Escriba una función `y = gentono(n)` que genere los tonos apropiados de acuerdo al valor del número n ingresado. Tenga en cuenta la frecuencia de muestreo de 8000 Hz, y que la duración mínima del tono debe ser al menos de 40 ms. (Vea el Inciso 5.)

4. Usando la función `gfft` diseñada en el inciso 1, construya un decodificador para DTMF, `n = dec_dtmf(y)`. El decodificador debe ser capaz de distinguir las muestras más destacadas del módulo del espectro $|X[k]|^2$ del tono $y[n]$, y a partir de ellas devolver un número n correspondiente al dígito "discado". La función `dec_dtmf()` debe graficar el tono $y[n]$ y el módulo del espectro $|X[k]|^2$.

5. (optativo) A continuación se da un listado de una posible función `gentono`, que incluye la posibilidad de contaminar con ruido las señales tonales. Ensaye la robustez de su detector ante la presencia de señales ruidosas. ¿Cuál es la máxima relación señal a ruido que permite decodificar los tonos sin error?

```
function y = gentono(n,dT,R,Ts,Sonido);
%  Y = GENTONO(N,dT,[Ar,Fr],Ts,S)
%    Esta funcion genera los tonos DTMF correspondientes al numero N elegido,
%    de acuerdo con la siguiente tabla:
%
%      Numero  TonoFila  TonoColumna   N
%        1       697       1209      1
%        2       697       1336      2
%        3       697       1477      3
%        4       770       1209      4
%        5       770       1336      5
%        6       770       1477      6
%        7       852       1209      7
%        8       852       1336      8
%        9       852       1477      9
%        0       941       1209      0
%        *       941       1336      10
%       %       941       1477     11
%
%    dT indica la duración del tono (por omisión es 40 ms) y
%    Ar: amplitud del ruido superpuesto (por omisión es 0 [cero]);
%    Fr: amplitud del ruido de fase (aleatorio para cada tono, por omisión
es 0)
%    La amplitud de cada tono es 1.
%
%    Ts es el tiempo de silencio despues de cada tono (por omisión es 40
ms)
%    S ~0 hace que suene el tono correspondiente a cada numero (por omisión
es 0)
if nargin<5; Sonido = 0; end;
if nargin<4; Ts = 0.04; end;
if nargin<3; R = [0 0]; end;
if nargin<2; dT = 0.04; end;
Ar = R(1);
Fr = R(2);
R = [697 770 852 941]';
C = [1209 1336 1477]';
Fs = 8000;% frecuencia de muestreo de la linea
fiR = Fr*rand(4,1)*2*pi;% ruido de los tonos de las filas.
fiC = Fr*rand(3,1)*2*pi;% ruido de los tonos de las columnas.
t = [0:1/Fs:dT];% vector de tiempos
ts = [dT:1/Fs:dT+Ts];% vector de tiempos despues del tono
tR = sin(2*pi*R*t+fiR*ones(size(t)))+Ar*(ones(size(R))*rand(size(t))-0.5);
tC = sin(2*pi*C*t+fiC*ones(size(t)))+Ar*(ones(size(C))*rand(size(t))-0.5);
switch n
  case 1
    y = tR(1,:)+tC(1,:);
  case 2
    y = tR(1,:)+tC(2,:);
  case 3;
    y = tR(1,:)+tC(3,:);
  case 4;
    y = tR(2,:)+tC(1,:);
  case 5;
```

```
    y = tR(2,:)+tC(2,:);
  case 6;
    y = tR(2,:)+tC(3,:);
  case 7;
    y = tR(3,:)+tC(1,:);
  case 8;
    y = tR(3,:)+tC(2,:);
  case 9;
    y = tR(3,:)+tC(3,:);
  case 0;
    y = tR(4,:)+tC(2,:);
  case 10;% el "*"
    y = tR(4,:)+tC(1,:);
  case 11;% el "#"
    y = tR(4,:)+tC(3,:);
  otherwise
    error('el N ingresado no corresponde al rango 0-11');
end;
y = [y zeros(size(ts))];
if Sonido~=0;
  sound(y,Fs);
end;
```

Apéndice: Señalización DTMF

Las compañías telefónicas poseen un sistema masivo de cables de cobre trenzados que llevan señales de voz y de control. Antiguamente, el discado, el inicio y el fin de la comunicación, etc., procesos conocidos como *señalización*, se basaban en interrumpir el circuito eléctrico. Por ejemplo, la marcación con disco abría el circuito un número de veces igual al dígito discado, donde cada interrupción duraba entre 40 ms y 60 ms. Este cambio en la corriente de la línea era detectado por relés, que efectuaban la conmutación de los distintos circuitos hasta lograr la comunicación. Los relés permanecían energizados con la corriente fluyendo por el circuito del teléfono, hasta que se cortaba la comunicación y el relé se desenergizaba (Fig. 11.44).

Sin embargo, la conexión y desconexión abrupta genera transitorios, "glitches" y perturbaciones en la línea que pueden perturbar el resto del equipamiento. Además, como la señalización va en modo común, es difícil asegurar el buen comportamiento de redes que cubran grandes distancias. A medida que las compañías telefónicas se modernizaron, los relés fueron reemplazados por transistores y circuitos integrados, abaratando costos y reduciendo el tamaño de los equipos. A principios de los 70 se propuso señalizar utilizando tonos audibles, evitando así la señalización en modo común, y todos sus inconvenientes. Los cables telefónicos (pares) soportan el ancho de banda necesario para una comunicación vocal inteligible, que es de 300 Hz a 3400 Hz, aproximadamente, de modo que la banda de frecuencia disponible para los tonos de señalización queda limitada a este rango.

La señalización podría haberse diseñado a partir de un único tono. Sin embargo, la presencia de señales vocales pueden generar interferencias con el tono de control. El empleo de más de un tono, relacionados entre sí de manera no armónica, elimina la posibilidad que la voz pueda generar accidentalmente una combinación válida. Combinando solamente siete tonos, cuatro de "baja frecuencia" y tres de "alta frecuencia" se pueden generar 12 combinaciones válidas. El sistema de discado telefónico en uso hoy en día utiliza 10 de esas combinaciones de tonos para generar los dígitos 0–9, y además los códigos accesorios "asterisco (⁎)" y "numeral (#)". En realidad, el sistema prevé el empleo de un cuarto tono de alta frecuencia, lo que permite generar las 16 combinaciones que se ven en la Fig. 11.45. La cuarta columna (símbolos A–D) se utiliza para fines especiales, dentro de la central. Los teléfonos normales utilizan el teclado comercial y los teléfonos o aparatos especiales utilizan además las teclas A, B, C y D, que junto con el teclado convencional constituyen el teclado extendido. Esta técnica de discado se conoce como DTMF (*dual-tone, multi-frequency*).

Al pulsar alguna tecla, se ordena al circuito generador de señalización DTMF que sume

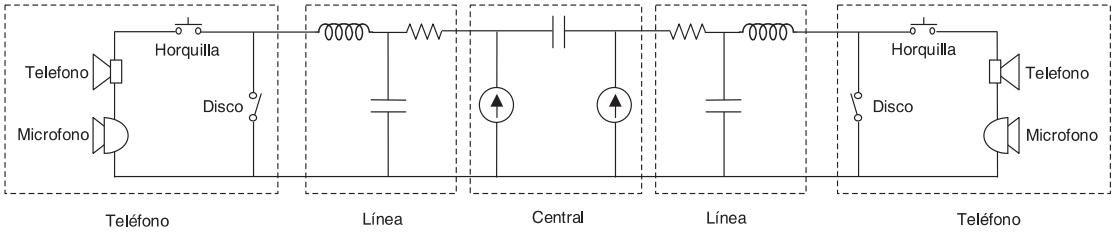


Fig. 11.44. Esquema simplificado de una línea telefónica.

dos señales sinusoidales de las frecuencias indicadas por la matriz y las envíe por la línea telefónica: cuando se pulsa la tecla 4, se envía la señal que es la suma de dos sinusoidales (una de frecuencia 770 Hz. y la otra de 1209 Hz). La central telefónica decodificará esta señal como el dígito 4 y obrará en consecuencia.

El sistema de discado funciona aún con la presencia simultánea de la voz humana. Los armónicos de los tonos de baja frecuencia (las frecuencias correspondientes a las filas de la tabla) no se superponen a los tonos altos (las frecuencias correspondientes a las columnas de la tabla), por lo que no existe la posibilidad de confusión debido a la generación de armónicas por las no linealidades de la red. La mayoría de la energía de la voz se debe a la vibración de las cuerdas vocales, y cae dentro de la zona asignada a los tonos de "baja frecuencia". Un análisis espectral de la voz mostraría un espectro discreto con múltiples armónicos, y no un espectro uniforme, lo que hace casi imposible que un único timbre de voz pueda generar una señal DTMF válida. Además, los tonos DTMF fueron diseñados de forma que no sean armónicos de frecuencias muy usadas como las de 50 o 60 Hz. Si la generación y decodificación de los tonos es suficientemente precisa, la señalización DTMF supera a la de pulsos pues es más rápida, permite seleccionar un mayor número de dígitos (16 en lugar de 10), presenta mayor inmunidad al ruido, no utiliza señales de modo común, etc. Además, los tonos resultan melodiosos al oído y permiten recordar un número cualquiera por la manera en que suena (al menos, para aquellos con oído musical).

La generación electrónica de estos tonos es sencilla: la parte complicada es la decodificación. La recepción y decodificación de estos tonos con la tecnología de los años 70 era difícil. Se utilizaban circuitos resonantes L-C, pero a las frecuencias utilizadas, las inductancias y capacitores eran grandes y voluminosos. Además, requerían costosos ajustes manuales. No obstante, éste no era un gran problema porque estos circuitos se ubican en la central telefónica, y no en el equipo del abonado.

A medida que la tecnología progresó, los circuitos sintonizados L-C dieron paso a filtros pasabandas analógicos construidos con circuitos integrados, mucho más pequeños, pero costosos por la necesidad de ajustes y la deriva térmica. Después llegaron los filtros de capacitores conmutados, construidos sobre una pequeña oblea de silicio. Como funcionan bajo el control de un único oscilador a cristal eliminan el problema de la deriva y el ajuste. Con el avance de la técnica de integración, los filtros de capacitores conmutados

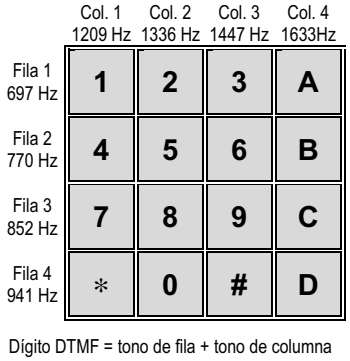


Fig. 11.45. Tonos multifrecuentes asociados a cada dígito.

se combinaron con otros circuitos dentro de la misma oblea. Esto incluye el excitador del circuito del reloj, los circuitos de temporización, la detección de cruce por cero,"latches" y "buffers" para el decodificador, junto con la interfaz analógica para la conexión a la línea. Las únicas partes externas son el cristal del reloj y algunos pocos componentes pasivos para adaptarlos a alguna aplicación específica. Hoy en día, un único integrado puede procesar y decodificar las 16 combinaciones de DTMF.

El diseño del circuito decodificador era una tarea formidable hasta hace 20 años, cuando aparecieron en el mercado los circuitos integrados. Al momento de su introducción eran caros, entre U$S20.00 y U$S50.00 cada uno; actualmente son muy económicos. Hoy en día existe una gran variedad de generadores y decodificadores DTMF; también se dispone de microcontroladores que incluyen el generador y el decodificador de DTMF con capacidad de control por programa. Algunos integrados detectores/decodificadores son SSI303/3/4, de Silicon Systems, CS20x, de Crystal Semiconductor, MSM6843 de Oki, MC145436 de Motorola, M957-0x de Teltone, UM9203/4 y UM92870A/B/C de UMC, MV8870 de Mitel, TC35301 de Toshiba, CD22202/3/4 de Harris, etc. En general, todos estos integrados necesitan solamente un cristal de 3.58 MHz (que se usa en los receptores de televisión NTSC, y por lo tanto son muy comunes y económicos), y la salida son 4 bits + 1 strobe, o decodificación código 2 de 8.

11.11.0.1. Algunos datos técnicos.

Para que un par de tonos sea detectado como válido, se deben satisfacer las siguientes condiciones:

- En la señal puede estar presente sólo un tono de cada grupo (uno de frecuencias altas y otro de frecuencias bajas);
- La diferencia temporal entre el comienzo de cada uno de los tonos del par debe ser menor a 5 ms;
- La duración de la señal asociada a cada dígito debe ser de al menos 40 ms;
- La variación de la frecuencia fundamental de cada tono debe ser menor al $\pm 2\,\%$ ($\pm 1{,}5\,\%$ en Argentina);
- La diferencia de niveles entre tonos del mismo grupo (conocida en la jerga como "*twist*") debe ser menor que 6 dB, y además la señal de tono alto debe ser 3 a 4 dB más elevada que la de tono bajo.

Todas estas características hacen que la detección accidental de un tono DTMF sea altamente improbable.

11.11.0.2. Decodificación de los tonos DTMF

Aunque existe una gran variedad de circuitos para generar y decodificar tonos DTMF, actualmente se prefiere una implementación discreta, programada en un DSP, ya que es más estable, precisa, versátil, etc. que la solución analógica.

Tabla 11.1. Índices más próximos a las frecuencias de los tonos DTMF para las TDF de $N = 201$ y $N = 205$ puntos.

Índices para TDF de *N* = 201 puntos				Índices para TDF de *N* = 205 puntos			
Tono básico en Hz	Valor de *k* exacto	Valor entero más próximo	Error absoluto en *k*	Segundo armónico en Hz	Valor de *k* exacto	Valor entero más próximo	Error absoluto en *k*
697	17.861	18	0.139	1394	35.024	35	0.024
770	19.731	20	0.269	1450	38.692	39	0.308
852	21.833	22	0.167	1704	42.813	43	0.187
941	24.113	24	0.113	1882	47.285	47	0.285
1209	30.981	31	0.019	2418	60.752	61	0.248
1336	34.235	34	0.235	2672	67.134	67	0.134
1477	37.848	38	0.152	2954	74.219	74	0.219
1633	41.846	42	0.154	3266	82.058	82	0.058

La generación discreta de una señal DTMF involucra la suma de dos sucesiones sinusoidales de longitud finita; cada una de estas sucesiones puede generarse recorriendo periódicamente una tabla (método de *look-up*) o calculando una expansión polinomial. La detección de los tonos puede hacerse calculando la TDF de la señal DTMF, y midiendo la energía presente en cada una de la 8 frecuencias de interés. Como la duración mínima de la señal es de 40 ms, con una frecuencia de muestreo de 8 kHz cada señal tiene una longitud mínima de $L = 0{,}04 \times 8000 = 320$ muestras, a partir de las cuales se debe decodificar cada uno de los dígitos. El número de puntos N de la TDF es menor que L, y se elige de modo de minimizar la diferencia entre la ubicación real del tono y el valor entero más próximo del índice k de la TDF.

El decodificador DTMF calcula las muestras de la TDF más cercanas en frecuencia a cada uno de los ocho tonos fundamentales, y también calcula las muestras correspondientes a las segundas armónicas de estos tonos. Esto permite diferenciar entre la voz humana y las sinusoides puras generadas por la señal DTMF, ya que el espectro de la voz humana contiene componentes de todas las frecuencias mientras que los segundos armónicos de los tonos DTMF generados por el teléfono son de nivel despreciable. Usualmente el método de cálculo de la TDF es una versión modificada del algoritmo de Goertzel, donde en lugar de calcular el valor de $X[k]$ se calcula la energía $|X[k]|^2$.

El largo N determina el espaciado frecuencial y el tiempo de cálculo de cada muestra de la TDF. Un N grande hace que el espaciado sea menor, aumentando la resolución en el dominio frecuencial, e incrementando el tiempo de cálculo. La frecuencia f_k en Hz correspondiente a la muestra k-ésima de la TDF es

$$f_k = \frac{k}{N} F_s, \quad k = 0, \ldots, N-1, \tag{11.69}$$

donde F_s es la frecuencia de muestreo en Hz (típicamente, $F_s = 8$ kHz). Si la señal de entrada al detector contiene una sinusoide de frecuencia f_i que no coincide exactamente con alguna de las f_k determinadas por la ecuación (11.69), el fenómeno de fuga espectral

causa que el valor de $X[k]$ no sea cero para valores de k alejados del k más próximo a $f_i N/F_s$. Para minimizar este efecto se elege N de modo que los tonos del DTMF queden ubicados lo más próximo posible a alguna de las muestras de la TDF. En la práctica se ha encontrado que el N óptimo para detectar las tonos fundamentales del DTMF es $N = 201$, y es $N = 205$ para la detección de los segundos armónicos, siempre suponiendo una frecuencia de muestreo $F_s = 8$ kHz. En la Tabla 11.1 se indican los valores de los índices de la TDFs de 201 y 205 puntos más próximos a las frecuencias de los tonos.