36

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

"ALGORITMOS RECIENTES PARA EL CALCULO DE LA TRANSFORMADA DE FOURIER DISCRETA"

Tesis que para obtener el Título de Matemático
Presenta

GERARDO VEGA HERNANDEZ

FALLA DE ORIGEN

CIUDAD UNIVERSITARIA, MEXICO, D.F.
1991

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

INDICE

INTRODUCCION

El objetivo del presente trabajo es el estudio de los diferentes algoritmos empleados para el cálculo de la transformada de *Fourier* discreta (TFD) así como la teoría matemática que los sustenta. Además, se propondrán aquí algunas mejoras a tales algoritmos e incluso se dará una implementación de ellos.

El trabajo se ha dividido en ocho capítulos y un apéndice. En el primer capítulo se dará la definición de lo que es una transformada de *Fourier* discreta y a continuación se hará una revisión del algoritmo clásico de *CooleyTuckey* conocido como FFT, para el cual se propondrá una versión recursiva.

En el capítulo dos se verá que bajo ciertas condiciones es posible realizar el cálculo de transformadas de orden grande en términos de transformadas de órdenes pequeños. Por tanto, se concluirá al final de este capítulo que los algoritmos modernos para el cálculo de la transformada estarán basados en algoritmos eficientes para el cálculo de transformadas de órdenes pequeños.

Los capítulos tres y cuatro, serán dedicados al estudio de la teoría matemática que permite el desarrollo de algoritmos eficientes para el cálculo de transformadas de orden pequeño.

En el capítulo cinco se presentará una de las partes más importantes de este trabajo, ya que se llevará a la práctica la teoría vista en los capítulos tres y cuatro, con el desarrollo de un algoritmo eficiente para el cálculo de la transformada de orden 13.

En el capítulo seis se resumirán los algoritmos eficientes, ya existentes, para el cálculo de transformadas de orden pequeño.

En el capítulo siete se verá de qué manera se emplean los algoritmos eficientes que calculan transformadas de orden pequeño, para el desarrollo de algoritmos eficientes en el cálculo de transformadas de órdenes grandes.

En el capítulo ocho se darán las conclusiones del presente trabajo y se comentarán las diferencias entre los algoritmos que aquí se exponen.

Finalmente, en el apéndice, se llevará a la práctica la teoría de los capítulos anteriores, presentando un programa en *Pascal*, basado en el algoritmo de anidamiento.

1. REVISION DEL ALGORITMO FFT CLASICO DE COOLEY TUCKEY

El objetivo de este primer capítulo es presentar la definición de una transformada de *Fourier* discreta (TFD). Además, se hará notar que el cálculo directo de una TFD, en general, es un problema que requiere de un esfuerzo computacional considerable, por esta razón el desarrollo de algoritmos eficientes para su cálculo es fundamental. Debido a esto se incluirá el algoritmo clásico de *Cooley Tuckey* , que es uno de los primeros que se emplearon para el cálculo eficiente de la TFD. Asimismo propondremos una versión recursiva de este algoritmo.

1.1. La Transformada de Fourier Discreta (TFD)

Usualmente cuando se tiene una función o señal periódica en el tiempo $F(t)$ de periodo P, resulta útil para su análisis, representar a esta función como la suma de senoidales de diferentes frecuencias. A tal representación de la función $F(t)$ se le conoce como representación de *Fourier* y está dada por

$$F(t) = \sum_{k=-\infty}^{\infty} C(k)\, \mathbf{e}^{\,i2\pi k t/P} \tag{1.1.1}$$

donde

$$C(k) = \frac{1}{P} \int_0^P F(t)\, \mathbf{e}^{\,-i2\pi k t/P}\, dt \tag{1.1.2}$$

A los valores $C(k)$ se les conocen como los coeficientes de *Fourier* de la función $F(t)$ y como se puede observar, $C(k)$ es el peso con que contribuye la senoidal de frecuencia k ($e^{i2\pi kt/P}$) para la representación de *Fourier* de $F(t)$.

Cuando $F(t)$ resulta ser una función elemental, el cálculo de $C(k)$ se reduce a resolver y evaluar una integral sencilla. Sin embargo, en la práctica es comun que $F(t)$ sea una función complicada o algo aún más comun es que sólo se cuente con algunos valores discretos de la función $F(t)$, los cuales podrían ser el resultado de una serie de mediciones a intervalos de tiempo regulares de un fenómeno particular. Cuando este es el caso lo que se hace es obtener un valor aproximado de $C(k)$.

Denotemos como $Z_0, Z_1, Z_2, \ldots, Z_n, \ldots, Z_{N-1}$ a los N valores tomados de $F(t)$, dentro del período P, a los tiempos $0, T, 2T, \ldots, nT, \ldots, (N-1)T$ respectivamente. Donde $T = P/N$ es el intervalo regular de tiempo empleado entre muestra y muestra.

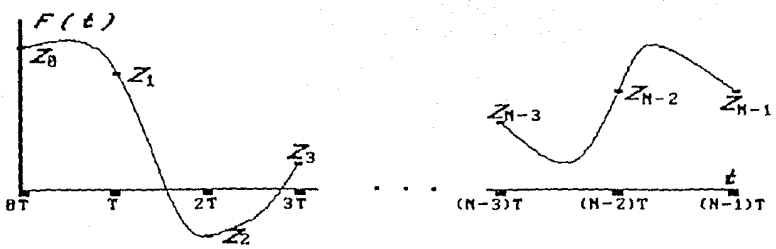

Figura 1. Obtención de los N valores Z_n a partir de $F(t)$ según la relación $Z_n = F(nT)$ con $n = 0,1,...,N-1$ y $T = P/N$.

De esta manera es posible obtener un valor aproximado de $C(k)$ si se sustituye en 1.1.2, la integral por una sumatoria, el periodo P por NT, t por nT y $F(t)$ por Z_n de lo cual se obtiene

$$C(k) \approx \hat{Z}_k = \frac{1}{NT} \sum_{n=0}^{N-1} Z_n \, \mathbf{e}^{-i2\pi k n T/NT} \, T \qquad (1.1.3$$

Una expresión más sencilla para $\hat{Z}_k$ se logra si se define a la N-ésima raíz de la unidad $e^{-i2\pi/N}$ como $\omega_N = \mathbf{e}^{-i2\pi/N}$ y de esta manera, $\hat{Z}_k$ estaría dada por

$$\hat{Z}_k = \frac{1}{N} \sum_{n=0}^{N-1} Z_n \, \omega_N^{kn} \qquad (1.1.4$$

A la expresión 1.1.4 se le conoce como la Transformada de *Fourier* Discreta (TFD) de orden N del vector $\mathbf{Z} = (Z_0, Z_1, ..., Z_{N-1})$. Algo que es importante hacer resaltar de la expresión 1.1.4 es que la sucesión de valores $\hat{Z}_k$ ($k = ...-2,-1,0,1,2,...$) es una sucesión N periódica, es decir $\hat{Z}_{k+N} = \hat{Z}_k$, esto se debe a que $\omega_N^{(k+N)n} = \omega_N^{kn}$, de ahí que será suficiente con efectuar el cálculo de $\hat{Z}_k$ para $k = 0,1,...,N-1$. A continuación se dan algunas otras propiedades importantes de las raíces de la unidad ω_N^l, $l = 0,1,...,N-1$.

$$1)\quad \omega_N^{(k+N)} = \omega_N^k \quad y\ por\ tanto \quad \omega_N^k = \omega_N^{k\,Mod\,N}$$
$$2)\quad \omega_{km}^k = \omega_m$$
$$3)\quad \omega_{2m}^{m+l} = -\omega_{2m}^l \qquad l = 0,1,...,m-1$$
$$4)\quad \sum_{n=0}^{N-1} \omega_N^{k(n-r)} = \begin{cases} 0 & si\ n \neq r \\ N & si\ n = r \end{cases} \qquad (1.1.5$$

Si se multiplica por ω_N^{-kr} ($r = ...,-2,-1,0,1,2,...$), a ambos lados de la expresión 1.1.4 y se efectúa la suma para $k = 0,1,...,N-1$

$$\sum_{k=0}^{N-1} \hat{Z}_k \omega_N^{-kr} = \sum_{k=0}^{N-1} \left(\frac{1}{N} \sum_{n=0}^{N-1} Z_n \omega_N^{kn}\right) \omega_N^{-kr} = \sum_{n=0}^{N-1} Z_n \left(\frac{1}{N} \sum_{k=0}^{N-1} \omega_N^{k(n-r)}\right) \quad (1.1$$

y por la propiedad 4 de 1.1.5 se tiene que

$$Z_n = \sum_{k=0}^{N-1} \hat{Z}_k \omega_N^{-kn} \quad (1.1$$

Como se puede observar esta última igualdad expresa la manera de obtener el cálculo aproximado de la Tranformada de *Fourier* inversa. Por eso a tal igualdad se le conoce como la Transformada Inversa de *Fourier* Discreta (TIFD). De esta forma se establece, que si dos sucesiones N periódicas Z_n y $\hat{Z}_k$ con $n,k = 0,1,...,N-1$, están relacionadas mediante 1.1.4 y 1.1.7, entonces estas sucesiones forman lo que se llama un par de tranformadas de *Fourier* discretas.

La siguiente definición de la TDF en términos matriciales será de gran utilidad en lo sucesivo para el desarrollo de algoritmos eficientes para su cálculo

Definición [6]: Dado el vector $\mathbf{Z} \in C^N$ definimos a la TFD de $\mathbf{Z}$ como el vector $\hat{\mathbf{Z}} \in C^N$ que se obtiene de

$$\hat{\mathbf{Z}} = \frac{1}{N} \mathbf{W}_N \mathbf{Z} \quad (1.1.$$

dode la matriz $\mathbf{W}_N$ queda definida por

$$\mathbf{W}_N = \left\{\omega_N^{kl}\right\} \quad k,l = 0,1,...,N-1 \quad (1.1.$$

con $\omega_N = \mathbf{e}^{-i2\pi/N}$

Algunas propiedades importantes de la matriz $\mathbf{W}_N$ son las siguientes

$$1)\ \mathbf{W}_N^T = \mathbf{W}_N$$
$$2)\ \bar{\mathbf{W}}_N = \mathbf{W}_N^{-1} \quad (1.1.$$

donde $\mathbf{W}_N^T$ y es la matriz transpuesta de $\mathbf{W}_N$ y $\bar{\mathbf{W}}_N$ es su matriz conjugada.

Por otro lado el procedimiento de la figura 3 permite la obtención del vector transformado $\hat{\mathbf{Z}}$, este procedimiento, como se puede apreciar, es la codificación directa en *Pascal* de la expresión 1.1.5, donde se esta considerando al arreglo w[l,m] como global.

```
Procedure Calcula_TFD(ze:arreglo; Var Zs:arreglo; N:integer);
  Var
    k, l, kl : Integer;
  Begin
    For k := 0 To N-1 Do Begin
      Zs[k,1] := 0;
      Zs[k,2] := 0;
      For l := 0 To N-1 Do Begin
        kl := (k*l) Mod N;                                  { ω_N^{kl} = ω_N^{kl Mod N} }
        Zs[k,1] := Zs[k,1] + Ze[i,1]*w[kl,1] - Ze[i,2]*w[kl,2];   { calculo de Ẑ }
        Zs[k,2] := Zs[k,2] + Ze[i,2]*w[kl,1] + Ze[i,1]*w[kl,2]
      End;
      Zs[k,1] := Zs[k,1]/N;                                 { Ẑ = (1/N) Ẑ }
      Zs[k,2] := Zs[k,2]/N;
    End
  End;
```

Figura 3. Procedimiento para el cálculo de la TFD.

Interpretación computacional: Procedimientos semejantes al expuesto en la figura 3, permiten el cálculo de la transformada de *Fourier* discreta. Sin embargo, un análisis detallado de éste revela que si se desea realizar una TFD de orden N, (es decir el vector de entrada Z_n consta de N valores; $n = 0,1,...,N-1$) entonces el número de multiplicaciones complejas que se tendrían que realizar sería N^2, de lo cual se concluye que el tiempo de procesamiento que requiere una computadora para el cálculo de una TFD de orden N es proporcional a N^2. Esto, como se puede inferir un serio obstáculo, pues aún con las altas velocidades de cálculo de los equipos de computo moderno, se requeriría de un tiempo excesivo para el cálculo de una TFD cuando el orden N es grande. Por esta razón se buscó el desarrollo de algoritmos alternativos para el cálculo de la TFD. Uno de los primeros desarrollos en este sentido fue el algoritmo llamado *Fast Fourier Trasform* (FFT) propuesto por *JohnW.Tuckey* y *JamesW.Cooley* en 1965 [5], tal algoritmo permite que el tiempo de procesamiento que requiere una computadora para el cálculo de una TFD de orden N sea proporcional a $(N/2)\,Log_2(N/4) + 1$, lo cual significa un ahorro considerable si se compara con el algoritmo original.

$$\mathbf{P}_{2m}\mathbf{W}_{2m} = \begin{pmatrix} \mathbf{W}_m & \mathbf{W}_m \\ \mathbf{W}_m\mathbf{D}_m & -\mathbf{W}_m\mathbf{D}_m \end{pmatrix} = \begin{pmatrix} \mathbf{W}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_m \end{pmatrix} \begin{pmatrix} \mathbf{I}_m & \mathbf{I}_m \\ \mathbf{D}_m & -\mathbf{D}_m \end{pmatrix} \tag{1.3.4}$$

donde $\mathbf{I}_m$ es la matriz identidad de orden m.

Una notación aún más sencilla se logra si se emplea el producto de *Kronecker* o tensorial de dos matrices $\mathbf{A} = \{\alpha_{ij}\}$ y $\mathbf{B} = \{\beta_{ij}\}$ definido como

$$\mathbf{A} \otimes \mathbf{B} = \{\alpha_{ij}\mathbf{B}\} = \begin{pmatrix} \alpha_{11}\mathbf{B} & \alpha_{12}\mathbf{B} & \dots & \alpha_{1m}\mathbf{B} \\ \alpha_{21}\mathbf{B} & \alpha_{22}\mathbf{B} & \dots & \alpha_{2m}\mathbf{B} \\ \cdot & \cdot & \cdot & \cdot \\ \alpha_{n1}\mathbf{B} & \alpha_{n2}\mathbf{B} & \dots & \alpha_{nm}\mathbf{B} \end{pmatrix} \tag{1.3.5}$$

Empleando este producto es posible reescribir a $\mathbf{P}_{2m}\mathbf{W}_{2m}$ como

$$\mathbf{P}_{2m}\mathbf{W}_{2m} = \left(\mathbf{I}_2 \otimes \mathbf{W}_m\right) \mathbf{H}_{2m} \left(\mathbf{W}_2 \otimes \mathbf{I}_m\right) \tag{1.3.6}$$

donde

$$\mathbf{H}_{2m} = \begin{pmatrix} \mathbf{I}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_m \end{pmatrix} \qquad y \qquad \mathbf{W}_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{1.3.7}$$

de esta manera el cálculo de una tranformada de orden $N = 2m$ puede efectuarse según

$$\hat{\mathbf{Z}} = \frac{1}{2m}\mathbf{P}_{2m}{}^{T}\left(\mathbf{I}_2 \otimes \mathbf{W}_m\right) \mathbf{H}_{2m} \left(\mathbf{W}_2 \otimes \mathbf{I}_m\right) \mathbf{Z} \tag{1.3.8}$$

en donde la matriz $\mathbf{P}_{2m}^{T}$ es tal que $\mathbf{P}_{2m}^{T}\,\mathbf{P}_{2m} = \mathbf{I}_{2m}$, ahora bien, el vector columna

$$\mathbf{H}_{2m} \left(\mathbf{W}_2 \otimes \mathbf{I}_m\right) \mathbf{Z} = \begin{pmatrix} \mathbf{I}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_m \end{pmatrix} \begin{pmatrix} \mathbf{I}_m & \mathbf{I}_m \\ \mathbf{I}_m & -\mathbf{I}_m \end{pmatrix} \mathbf{Z} \tag{1.3.9}$$

1.4. Versión recursiva del algoritmo FFT

Un algoritmo rápido (FFT) para el cálculo de la TFD se establece, si se aplica de manera recursiva la idea expuesta en la sección anterior, cuando el orden N de una TFD es una potencia de 2. Concretamente supóngase que el orden N de una TFD es de la forma 2^k (k = 1,2,...). Entonces el cálculo de $\hat{\mathbf{Z}} = \frac{1}{2^k}\mathbf{P}_2^k\left(\mathbf{I}_2 \otimes \mathbf{W}_2^{k-1}\right)\mathbf{H}_2^k\left(\mathbf{W}_2 \otimes \mathbf{I}_2^{k-1}\right)\mathbf{Z}$ puede llevarse a cabo en partes, efectuándose primeramente el cálculo de $\mathbf{H}_2^k\left(\mathbf{W}_2 \otimes \mathbf{I}_2^{k-1}\right)\mathbf{Z}$ el cual puede ser realizado, basándose en 1.3.10, mediante el siguiente algoritmo, con el que además se logra que el resultado quede en el mismo vector $\mathbf{Z}$.

Calculo de $\mathbf{H}_{2m}\left(\mathbf{W}_2 \otimes \mathbf{I}_m\right)\mathbf{Z}$

1.- $tem \leftarrow Z_{ind}$

$Z_{ind} \leftarrow tem + Z_{ind+m}$

$Z_{ind+m} \leftarrow tem - Z_{ind+m}$

2.- Para l = 1,2,...,$m-1$

$tem \leftarrow Z_{ind+l}$

$Z_{ind+l} \leftarrow tem + Z_{ind+m+l}$

$Z_{ind+m+l} \leftarrow \omega_2^{j.l}\,(tem - Z_{ind+m+l})$

donde $m = 2^{k-1}$, $ind = 0$ y $j = 1$. Con el algoritmo anterior se tiene apenas una parte de el cálculo, pues como se indica en 1.3.11 es necesario ahora calcular dos transformadas de orden 2^{k-1}; una transformada para el vector ($z_0, z_1, ..., z_{m-1}$) y la otra para el vector ($z_m, z_{m+1}, ..., z_{2m-1}$). Ahora bien, si $m > 1$ entonces estas transformadas pueden ser realizadas en base a la siguiente expresión

$$\frac{1}{2^k}\mathbf{P}_2^k\left(\mathbf{I}_2 \otimes \mathbf{P}_2^{k-1}\right)\left(\mathbf{I}_2 \otimes \mathbf{W}_2^{k-2}\right)\mathbf{H}_2^{k-1}\left(\mathbf{W}_2 \otimes \mathbf{I}_2^{k-2}\right)\begin{pmatrix} Z_{ind} \\ Z_{ind+1} \\ \cdot \\ Z_{ind+m-1} \end{pmatrix} \qquad ind = 0, m \tag{1.4.1}$$

donde $\mathbf{P}_2^{k-1}$ es la matriz de permutación que reaordena los renglones de la matriz $\mathbf{W}_2^{k-1}$ en pares e impares. En cuanto a la matriz $\mathbf{h}_2^{k-1}$ esta es de la forma

$$\mathbf{H}_2^{k-1} = \begin{pmatrix} \mathbf{I}_2^{k-2} & 0 \\ 0 & \mathbf{D}_2^{k-2} \end{pmatrix} \tag{1.4.2}$$

```
Procedure FFT(Var Z:arreglo; N, ind, j:integer);
  Var
    xr, xi  : Real;
    l, jl, m : Integer;
  Begin
    m := dim Shr 1;                          { m ← N/2 }
    xr := Z[ind,1];
    xi := Z[ind,2];
    Z[ind,1] := xr + Z[ind+m,1];
    Z[ind,2] := xi + Z[ind+m,2];
    Z[ind+m,1] := xr - Z[ind+m,1];           { multiplicación por ω^0 }
    Z[ind+m,2] := xi - Z[ind+m,2];
    jl := 0;
    For l := 1 To m-1 Do                     { calcula las sumas }
      Begin                                  { y multiplicaciones }
        xr := Z[ind+l,1];                    { restantes.       }
        xi := Z[ind+l,2];
        Z[ind+l,1] := xr + Z[ind+m+l,1];     { Z_ind+l + Z_ind+m+l }
        Z[ind+l,2] := xi + Z[ind+m+l,2];
        xr := xr - Z[ind+m+l,1];
        xi := xi - Z[ind+m+l,2];
        jl := jl + j;
        Z[ind+m+l,1] := xr*w[jl,1] - xi*w[jl,2];   { ω^jl ( Z_ind+l + Z_ind+m+l ) }
        Z[ind+m+l,2] := xi*w[jl,1] + xr*w[jl,2]
      End
    If m > 1 Then                            { ¿ es necesario calcular  }
      Begin                                  { otras dos transformadas ? }
        j := j Shl 1;                        { j ← 2j }
        FFT(Z, m, ind, j);                   { calcula dos TFD de orden m }
        FFT(Z, m, ind+m, j)
      End
  End;
```

Figura 4. Procedimiento para la FFT.

Es importante enfatizar que después de la ejecución del procedimiento anterior, es necesario reordenar las entradas del vector resultante lo cual es indicado en 1.4.1 por las matrices $\mathbf{P}_2^{T_r}$, $r = 0,1,...,k-1$. Este reordenamiento que ha de realizarse sobre el vector resultante se conoce como reversión binaria y será estudiado en la siguiente sección.

Interpretación computacional: El número de multiplicaciones necesarias para el cálculo de una TDF de orden N si se emplea el algoritmo FFT, puede ser determinado si se observa el paso 2 del algoritmo recursivo, el cual indica que para cada paso recursivo del algoritmo se efectúan $m-1$ multiplicaciones complejas, donde m es la mitad de la dimensión de la TFD en turno. Esto quiere decir que para llevar a cabo el cálculo de una TFD de orden $N = 2^k$ ($k = 1,2,...$), se efectuarán 2^{k-1} multiplicaciones complejas en el primer nivel de recursividad de el algoritmo, y dado que el paso 3 indica que se realice, si es necesario, dos TFD de orden 2^{k-1}, entonces $2(2^{k-2}-1)$ multiplicaciones complejas, serán las que se efectuen en el segundo nivel de recursividad. Y así sucesivamente por cada nivel de recursividad r ($r = 1,2,...,k-1$) se efectuarán $2^{r-1}(2^{k-r}-1)$ multiplicaciones complejas. Por tanto el número total de multiplicaciones complejas necesarias para el cálculo de una TFD de orden $N = 2^k$ ($k = 1,2,...$) estará dado por

$$\sum_{r=1}^{k-1} 2^{(r-1)}\,(2^{(k-r)}-1) = 2^{(k-1)}\,(k-2)+1 = \frac{N}{2}\,Log_2\frac{N}{4}+1 \qquad (1.4.$$

1.5. Reversión binaria

El algoritmo rápido FFT, desarrollado en la sección previa se fundamentó en la propiedad de la matriz $\mathbf{W}_{N=2m}$, la cual al ser multiplicada por la matriz de permutación $\mathbf{P}_{2m}$, que reordena sus renglones en pares e impares, da una matriz producto $\mathbf{P}_{2m}\mathbf{W}_{2m}$ que puede ser expresada en términos de la matriz $\mathbf{W}_m$ como se muestra a continuación

$$\mathbf{W}_{2m} = \mathbf{P}_{2m}{}^{T}\begin{pmatrix} \mathbf{W}_m & \mathbf{W}_m \\ \mathbf{W}_m\mathbf{D}_m & -\mathbf{W}_m\mathbf{D}_m \end{pmatrix} \qquad (1.5.1$$

Este reordenamiento que hace posible reducir casi a la mitad el número de multiplicaciones complejas necesarias para el cálculo de una de una TFD de orden $N = 2m$, también es el responsable de que sea necesario reordenar, en forma inversa al modo en que se hizo con los renglones de la matriz $\mathbf{W}_{2m}$, las entradas del vector $\mathbf{Z} = \mathbf{P}_{2m}\,\hat{\mathbf{Z}}$ para obtener el vector $\hat{\mathbf{Z}}$, es decir $\hat{\mathbf{Z}} = \mathbf{P}_{2m}^{T}\,\tilde{\mathbf{Z}}$.

$$\hat{\mathbf{Z}} = \frac{1}{2m}\mathbf{W}_{2m}\,\mathbf{Z} = \frac{1}{2m}\mathbf{P}_{2m}{}^{T}\mathbf{P}_{2m}\mathbf{W}_{2m}\,\mathbf{Z} \qquad (1.5.$$

El orden en que quedan las entradas del vector $\tilde{\mathbf{Z}} = \mathbf{P}_N \hat{\mathbf{Z}}$, con respecto al vector ordenado $\hat{\mathbf{Z}}$ es el siguiente

$$\tilde{Z}_i = \hat{Z}_{P_1(i)} \qquad i = 0,1,...,N-1 \tag{1.5.3}$$

donde $P_1 : \{0,1,...,N-1\} \rightarrow \{0,1,...,N-1\}$ es una función de permutación definida como

$$P_1(i) = \begin{cases} 2i & i = 0,1,2,...,N/2-1 \\ P_1(i-N/2)+1 & i = N/2,N/2+1,...,N-1 \end{cases} \tag{1.5.4}$$

Ahora bien y de acuerdo al principio de recursividad en el que se basó el algoritmo FFT, cuando el orden N de una TFD es de la forma 2^k ($k = 1,2,...$), la función $P_2 : \{0,1,...,N-1\} \rightarrow \{0,1,...,N-1\}$ definida como

$$P_2(i) = \begin{cases} 4i & i = 0,1,2,...,N/4-1 \\ P_2(i-N/4)+2 & i = N/4,N/4+1,...,N/2-1 \\ P_2(i-N/2)+1 & i = N/2,N/2+1,...,N-1 \end{cases} \tag{1.5.5}$$

permite relacionar en forma sencilla, según 1.5.6, el vector $\hat{\mathbf{Z}}$ con el vector $\tilde{\mathbf{Z}} = Diag(\mathbf{P}_2^{k-1})\mathbf{P}_2^k \hat{\mathbf{Z}}$ (donde $\mathbf{P}_2^k = \mathbf{P}_N$) que se obtiene como resultado de aplicar en forma recursiva dos veces el algoritmo FFT.

$$\tilde{Z}_i = \hat{Z}_{P_2(i)} \qquad i = 0,1,...,N-1 \tag{1.5.6}$$

De esta manera la función $P_{k-1} : \{0,1,...,N-1\} \rightarrow \{0,1,...,N-1\}$ dada por 1.5.7 relaciona, según 1.5.8, las entradas del vector $\hat{\mathbf{Z}}$ con el vector $\mathbf{Z} = Diag(\mathbf{P}_2^2)\cdots Diag(\mathbf{P}_2^{k-1})\mathbf{P}_2^k \hat{\mathbf{Z}}$ que se obtiene al finalizar el último ($k-1$) paso recursivo del algoritmo FFT.

$$P_{k-1}(i) = \begin{cases} 0 & i = 0 \\ P_{k-1}(i-1)+N/2 & i = 1 \\ P_{k-1}(i-2)+N/4 & i = 2,3 \\ \quad\vdots & \quad\vdots \\ P_{k-1}(i-N/2)+1 & i = N/2,N/2+1,...,N-1 \end{cases} \tag{1.5.7}$$

$$\tilde{Z}_i = \hat{Z}_{P_{k-1}(i)} \qquad i = 0,1,...,N-1 \tag{1.5.8}$$

Una manera alternativa de definir a la función P_{k-1} es la siguiente

$$\begin{aligned} P_{k-1}(0) &= 0 \\ P_{k-1}(1) &= 2^{k-1} \\ P_{k-1}(2) &= 0 + 2^{k-2} \\ P_{k-1}(3) &= 2^{k-1} + 2^{k-2} \\ P_{k-1}(4) &= 0 + 0 + 2^{k-3} \\ &\cdot \\ P_{k-1}(N-1) &= 2^{k-1} + 2^{k-2} + 2^{k-3} + \ldots + 1 \end{aligned} \tag{1.5.9}$$

Si es estudiada con cuidado la definición de P_{k-1} dada en 1.5.9, se hallará que por medio de esta función se obtiene el número que resulta de realizar una representación binaria revertida de su argumento i ($i = 0,1,...,N-1$). Es decir, supóngase que $N = 2^4 = 16$ y que se desea evaluar $P_3(5)$, como la representación en binario a 4 dígitos de 5 es 0101, entonces al ser éstos revertidos se obtiene el número binario 1010 que corresponde a el número 10 en decimal, por lo tanto se tiene que $P_3(5) = 10$.

Una característica final de la función P_{k-1} es que su función inversa P_{k-1}^{-1} es ella misma, es decir

$$P_{k-1}(P_{k-1}(i)) = i \qquad i = 0,1,...,N-1 \tag{1.5.10}$$

Esto se desprende del hecho de que si es revertido dos veces un número binario se obtiene como resultado el mismo número. Por lo tanto y a partir de 1.5.8 se obtiene que

$$\hat{Z}_i = \tilde{Z}_{P_{k-1}(i)} \qquad i = 0,1,...,N-1 \tag{1.5.11}$$

La ecuación anterior junto con 1.5.9 expresa una manera mediante la cual es posible obtener en forma ordenada de las entradas del vector $\hat{\mathbf{Z}}$. De esta forma un algoritmo que realice el reordenamiento las entradas del vector, que resulta de aplicar el algoritmo FFT, está basado a su vez en la función P_{k-1}, la cual puede ser fácilmente programada, como se muestra en el procedimiento de la figura 5, empleando sólo sumas y corrimientos de bits (funciones Shr y Shl) debido a la manera en que se definió en 1.5.9.

```
Procedure Reordena_e_Imprime_Z(Z : vector; N : Integer);
 Type
  TP = Array[0..dim_max] Of Integer;
 Var
  P : TP;
  i, j, k, fac, des : Integer;
 Begin
  k := Round(Ln(N)/Ln(2));                    { calcula k }
  P[0] := 0;                                  { P_{k-1}(0) = 0 }
  Writeln(Z[P[0],1]/N,Z[P[0],2]/N);           { imprime la primera entrada }
  For i := 1 To k Do Begin
   fac := N Shr i;                            { fac = N/2^i }
   des := N Shr (k-i+1);
   For j := 0 To des-1 Do Begin
    P[des+j] := P[j] + fac;                   { = P_{k-1}(j) + N/2^i }
    Writeln(Z[P[des+j],1]/N,Z[P[des+j],2]/N)  { impresión en orden }
   End
  End
 End;
```

Figura 5. Procedimiento para el reordenamiento e impresión de las entradas del vector $\hat{Z}$

2. ALGORITMO DE GOOD

En este capítulo se verá que cuando el orden de una transformada es el producto de primos relativos (o el producto de potencias de primos), entonces el cálculo de la transformada puede ser realizado en términos de pequeñas transformadas cuyos órdenes correspondan a los factores en que se descompuso el orden de la transformada original. Por tanto, el desarrollo de algoritmos eficientes para el cálculo de la transformada, estarán basados a su vez en el desarrollo de algoritmos eficientes para el cálculo de transformadas de órdenes pequeños.

El algoritmo que se presenta en este capítulo, está basado en la descomposición de la matriz $\mathbf{W}_N$ en términos de matrices pequeñas $\mathbf{W}_{N_i}$ cuyos órdenes N_i son los factores, primos relativos, de N. La descomposición de la matriz $\mathbf{W}_N$ se logra luego de realizar ciertos reordenamientos de sus renglones y columnas los cuales están basados en el teorema Chino del residuo (TCR) y una variante de éste conocido como representación entera ruritiana (RER). Estos teoremas, como se recordará (al menos el TCR), permiten determinar a un entero en términos de sus residuos módulo un conjunto de enteros que sean primos relativos entre sí.

2.1. Teorema Chino del residuo y representación entera ruritiana

Antes de enunciar el teorema Chino del residuo, es conveniente presentar un ejemplo sencillo relacionado con el teorema.

Supóngase que se están estudiando tres fenómenos naturales F_1, F_2 y F_3, los cuales se sabe son cíclicos con periodos de $P_1 = 3, P_2 = 4$ y $P_3 = 5$ años respectivamente, se sabe también cuanto tiempo ha transcurrido, para cada uno de los fenómenos, desde que se presentó cada uno por última vez, siendo estos tiempos los siguientes

Hace $T_1 = 1$ año que se presentó por última vez F_1
Hace $T_2 = 3$ años que se presentó por última vez F_2
Hace $T_3 = 0$ años que se presentó por última vez F_3

Pregunta: ¿Es posible, en base a estos datos, determinar el tiempo T que ha transcurrido desde la última vez en que ocurrieron estos tres fenómenos simultáneamente?. La respuesta es afirmativa y la manera en que se obtiene la solución es la siguiente:

Concretamente, lo que se está buscando es un número entero T que cumpla con lo siguiente

$$T_i = T \, Mod \, P_i \qquad i = 1,2,3$$

Dado que los números enteros P_1, P_2 y P_3 son primos relativos entre sí, entonces es claro que el número T que cumpla con la igualdad anterior deberá ser mayor o igual a cero, pero estrictamente menor a $P = P_1 \cdot P_2 \cdot P_3 = 60$, pues si fuera mayor o igual a P entonces aunque fuera cierto que hace un tiempo T ocurrieron simultáneamente los tres fenómenos, esta ocurrencia no sería la ultima.

Ahora bien, es fácil comprobar que los números $K_1 = (P/P_1)^2 = 400$, $K_2 = (P/P_2)^2 = 225$ y $K_3 = (P/P_3)^4 = 20736$ tienen la siguiente propiedad

$$K_i \; Mod \; P_j = \delta_{ij} \qquad ij = 1,2,3$$

donde δ es la delta de *Kronecker*. Por tanto el número

$$\tilde{T} = \left[K_1 \cdot T_1 + K_2 \cdot T_2 + K_3 \cdot T_3 \right]$$

es tal que

$$T_i = \tilde{T} \; Mod \; P_i \qquad i = 1,2,3$$

Pero como ya se dijo, se tiene que cumplir que $0 \leq a < P$, por lo que entonces

$$T = \tilde{T} \; Mod \; P$$

De esta manera, al emplear los valores de K_1, K_2, K_3, T_1, T_2 y T_3 se llega a que $\tilde{T} = 1075$ y por lo tanto, finalmente, se tiene que la última vez en que ocurrieron los tres fenómenos simultáneamente fue hace

$$T = 55 \qquad años$$

Teorema Chino del residuo para enteros (TCR): Sea $N = N_1 \cdot N_2 \cdot \ldots \cdot N_L$, con $(N_i, N_j) = 1, i \neq j$ y dados a_i, $0 \leq a_i < N_i$, entonces existe un entero único a con $0 \leq a < N$ tal que

$$a_i = a \; Mod \; N_i \qquad i = 1,2,\ldots,L \qquad (2.1.1$$

donde a está determinado por

$$a = \left[\sum_{i=1}^{L} a_i \left(\frac{N}{N_i} \right)^{\varphi(N_i)} \right] Mod\ N \qquad (2.1.2$$

y φ es la función fi de *Euler*, la cual asocia a cada entero positivo n el número de enteros positivos primos relativos con n y menores a n, es decir

$$\varphi(n) = \begin{cases} 1 & si\ n = 1 \\ \sum_{0 < l < n} 1 \quad con\ (l, n) = 1 & si\ n > 1 \end{cases} \qquad (2.1.3$$

La función φ de *Euler* tiene varias propiedades importantes, entre las cuales se han considerado como de mayor interés para este trabajo las siguientes [3]

1) $\sum_{i \in F_N} \varphi(i) = N$ donde $F_N = \{ i : i \mid N, i > 0 \}$ (Teorema de *Gauss* [3])

2) si $(a, N) = 1$ entonces $a^{\varphi(N)} Mod\ N = 1$ (Teorema de *Euler* [3])

3) Sea $N = N_1 \cdot N_2 \cdot \ldots \cdot N_L$ con $(N_i, N_j) = 1, i \neq j, i,j = 1,2,\ldots,L$, entonces

$$\left(\frac{N}{N_i} \right)^{\varphi(N_i)} Mod\ N_j = \delta_{ij} \qquad i = 1,2,\ldots,L$$

Representación entera ruritiana (RER): Sea $N = N_1 \cdot N_2 \cdot \ldots \cdot N_L$, con $(N_i, N_j) = 1, i \neq j$ y dados a_i, $0 \leq a_i < N_i$, entonces existe un entero único a con $0 \leq a < N$ tal que

$$a_i = \left[a \left(\frac{N}{N_i} \right)^{\varphi(N_i) - 1} \right] Mod\ N_i \qquad i = 1,2,\ldots,L \qquad (2.1.4$$

donde a queda determinado por

$$a = \left[\sum_{i=1}^{L} a_i \left(\frac{N}{N_i} \right) \right] Mod\ N \qquad (2.1.5$$

La demostración de estos resultados no se dará aquí [3]. Sin embargo, éstos pueden ser obtenidos fácilmente si se emplea la tercera propiedad de la función φ de *Euler*.

El TCR, al igual que la RER, permiten establecer una función biyectiva entre cada entero a, $0 \le a < N = N_1 \cdot N_2 \cdot \ldots \cdot N_L, (N_i, N_j) = 1$ y sus L residuos módulo cada uno de los factores de N, por ejemplo para el TCR se puede establecer la siguiente función $P_N : Z_N \rightarrow Z_{N_1} \times Z_{N_2} \times \cdots \times Z_{N_L}$

$$a <\!\!- - - - - - \mathbf{P_N} - - - - - -\!\!> a_1, a_2, \ldots, a_L \tag{2.1.7}$$

con

$$a_i = a \; Mod \; N_i \qquad i = 1,2,\ldots,L$$

$$a = \left[\sum_{i=1}^{L} a_i \left(\frac{N}{N_i} \right)^{\varphi(N_i)} \right] Mod \; N \tag{2.1.8}$$

En cuanto a la RER se establece la siguiente función biyectiva $Q_N : Z_N \rightarrow Z_{N_1} \times Z_{N_2} \times \cdots \times Z_{N_L}$

$$a <\!\!- - - - - - \mathbf{Q_N} - - - - - -\!\!> a_1, a_2, \ldots, a_L \tag{2.1.9}$$

con

$$a_i = \left[a \left(\frac{N}{N_i} \right)^{\varphi(N_i) - 1} \right] Mod \; N_i \qquad i = 1,2,\ldots,L$$

$$a = \left[\sum_{i=1}^{L} a_i \left(\frac{N}{N_i} \right) \right] Mod \; N \tag{2.1.10}$$

Estas representaciones de un entero a, $0 \le a < N$ a través de los L residuos módulo cada uno de los factores de N, pueden verse como transformaciones que asocian a cada valor unidimensional a, $0 \le a < N$, un vector L-dimensional $(a_1, a_2, \ldots, a_L)$. Por tanto, como se verá en la sección 2.3., a través de estas transformaciones es posible expresar una TFD unidimensional en términos de una TFD multidimensional. Pero antes de eso se dará la definición de una TFD bidimensional y multidimensional.

2.2. TFD bidimensional y multidimensional

La definición de una transformada bidimensional es una extensión directa de la definición dada en el capítulo anterior de la TFD unidimensional, sólo que ahora la transformada en lugar de operar sobre arreglos unidimensionales (vectores) actuará sobre arreglos bidimensionales (matrices).

Definición de TFD bidimensional: La transformada bidimensional de una matriz $\mathbf{Z} = \left\{ \mathbf{Z}_{l,n} \right\} \in C^{N_1} \times C^{N_2}$ es una matriz $\hat{\mathbf{Z}} = \left\{ \hat{\mathbf{Z}}_{k,j} \right\} \in C^{N_1} \times C^{N_2}$ dada por

$$\begin{aligned} \hat{\mathbf{Z}}_{k,j} &= \frac{1}{N} \sum_{l=0}^{N_1-1} \sum_{n=0}^{N_2-1} \omega_N^{k l N/N_1} \omega_N^{j n N/N_2} \mathbf{Z}_{k,j} \\ &= \frac{1}{N} \sum_{l=0}^{N_1-1} \omega_{N_1}^{k l} \sum_{n=0}^{N_2-1} \omega_{N_2}^{j n} \mathbf{Z}_{l n} \qquad k = 0,1,\ldots,N_1-1 \quad j = 0,1,\ldots,N_2-1 \end{aligned} \tag{2.2.1}$$

donde $N = N_1 \cdot N_2$ y $\omega_N = e^{-i 2\pi/N}$.

Es posible, de manera semejante a como se hizo en el capítulo anterior, dar una definición de la transformada bidimensional en términos matriciales. Sin embargo, esto no parece tan evidente a primera vista, pues hay que recordar que ahora tanto $\mathbf{Z}$ como $\hat{\mathbf{Z}}$ ya no son vectores sino matrices. Una manera de solucionar el problema es definiendo una función biyectiva que permita maperar matrices en vectores. Una función con estas características es la función lexicográfica $\wedge 2$ la cual opera sobre el espacio de matrices de orden $N_1 \times N_2$ con coeficientes complejos al espacio de los vectores de orden $N_1 \cdot N_2$ con coeficientes complejos ($\wedge 2 : C^{N_1} \times C^{N_2} \rightarrow C^{N_1 \cdot N_2}$) mediante la siguiente regla

$$\wedge 2 \left(\mathbf{A} = \left\{ \alpha_{k,j} \right\} \right) = \mathbf{B} = \left\{ \beta_l \right\} \tag{2.2.2}$$

con $k = 0,1,\ldots,N_1 - 1, j = 0,1,\ldots,N_2 - 1, l = 0,1,\ldots,N_1 \cdot N_2 - 1$ y donde el vector $\mathbf{B} = \left\{ \beta_l \right\}$ se determina a través de

$$\beta_{k \cdot N_2 + j} = \alpha_{k,j} \tag{2.2.3}$$

Por ejemplo, supóngase que $\mathbf{A}$ es una matriz de orden 3×4, entonces el vector $\mathbf{B}$ de orden 12 estará dado por la siguiente igualdad

$$\begin{bmatrix} \beta_0 & \beta_1 & \beta_2 & \beta_3 \\ \beta_4 & \beta_5 & \beta_6 & \beta_7 \\ \beta_8 & \beta_9 & \beta_{10} & \beta_{11} \end{bmatrix} = \begin{bmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} & \alpha_{0,3} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} \end{bmatrix} \qquad (2.2.4$$

Empleando la función lexicográfica Λ_2 se pueden construir vectores $\mathbf{X}$ y $\widetilde{\mathbf{X}}$ a partir de las matrices $\mathbf{Z}$ y $\hat{\mathbf{Z}}$ como sigue

$$\begin{aligned} \underset{\sim}{\mathbf{X}} &= \Lambda_2(\mathbf{Z}) & \mathbf{X}_{l \cdot N_2 + n} &= \mathbf{Z}_{l,n} \\ \mathbf{X} &= \Lambda_2(\hat{\mathbf{Z}}) & \tilde{\mathbf{X}}_{k \cdot N_2 + j} &= \hat{\mathbf{Z}}_{k,j} \end{aligned} \qquad (2.2.5$$

con $l, k = 0,1,...,N_1 - 1, n, j = 0,1,...,N_2 - 1$. De esta manera, al reescribir la ecuación 2.2.1 empleando éstos vectores se tiene que

$$\widetilde{\mathbf{X}}_{k \cdot N_2 + j} = \frac{1}{N} \sum_{l=0}^{N_1 - 1} \omega_{N_1}^{k,l} \sum_{n=0}^{N_2 - 1} \omega_{N_2}^{j,n} \mathbf{X}_{l \cdot N_2 + n} \qquad (2.2.6$$

Ahora bien, si se definen los sigientes bloques de vectores

$$\begin{aligned} \underset{\sim}{\mathbf{X}}^l &= \left(X_{l \cdot N_2}, X_{l \cdot N_2 + 1}, ..., X_{l \cdot N_2 + N_2 - 1} \right) & l &= 0,1,...,N_1 - 1 \\ \mathbf{X}^k &= \left(X_{k \cdot N_2}, X_{k \cdot N_2 + 1}, ..., X_{k \cdot N_2 + N_2 - 1} \right) & k &= 0,1,...,N_1 - 1 \end{aligned} \qquad (2.2.7$$

y dado que $\mathbf{W}_{N_2} = \left\{ \omega_{N_2}^{j,n} \right\}, j, n = 0,1,...,N_2 - 1$, entonces 2.2.6 puede ser escrita en términos de la siguiente suma vectorial

$$\widetilde{\mathbf{X}}^k = \frac{1}{N} \sum_{l=0}^{N_1 - 1} \omega_{N_1}^{k,l} \mathbf{W}_{N_2} \mathbf{X}^l \qquad k = 0,1,...,N_1 - 1 \qquad (2.2.8$$

la cual a su vez peude ser reescrita en los siguientes términos

$$\begin{pmatrix} \widetilde{X}^0 \\ \widetilde{X}^1 \\ . \\ \widetilde{X}^{N_1-1} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} \omega_{N_1}^0 W_{N_2} X^0 + \omega_{N_1}^0 W_{N_2} X^1 + \ldots + \omega_{N_1}^0 W_{N_2} X^{N_1-1} \\ \omega_{N_1}^0 W_{N_2} X^0 + \omega_{N_1}^1 W_{N_2} X^1 + \ldots + \omega_{N_1}^{N_1-1} W_{N_2} X^{N_1-1} \\ . \\ \omega_{N_1}^0 W_{N_2} X^0 + \omega_{N_1}^{N_1-1} W_{N_2} X^1 + \ldots + \omega_{N_1}^{(N_1-1)^2} W_{N_2} X^{N_1-1} \end{pmatrix} \qquad (2.2$$

y en términos del producto de una matriz y un vector por bloques

$$\begin{pmatrix} \widetilde{X}^0 \\ \widetilde{X}^1 \\ . \\ \widetilde{X}^{N_1-1} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} \omega_{N_1}^0 W_{N_2} & \omega_{N_1}^0 W_{N_2} & \ldots \omega_{N_1}^0 W_{N_2} \\ \omega_{N_1}^0 W_{N_2} & \omega_{N_1}^1 W_{N_2} & \ldots \omega_{N_1}^{N_1-1} W_{N_2} \\ . & . & . \\ \omega_{N_1}^0 W_{N_2} & \omega_{N_1}^{N_1-1} W_{N_2} & \ldots \omega_{N_1}^{(N_1-1)^2} W_{N_2} \end{pmatrix} \begin{pmatrix} X^0 \\ X^1 \\ . \\ X^{N_1-1} \end{pmatrix} \qquad (2.2.1$$

Por tanto, si se emplea el producto tensorial en la ecuación anterior se llega a la siguiente expresión matricial para el cálculo de la transformada bidimensional

$$\widetilde{\mathbf{X}} = \frac{1}{N} \left(\mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2} \right) \mathbf{X} \qquad (2.2.1$$

en donde se debe recordar que

$$\mathbf{Z}_{l,n} = \mathbf{X}_{l \cdot N_2 + n} \qquad \hat{\mathbf{Z}}_{k,j} = \widetilde{\mathbf{X}}_{k \cdot N_2 + j} \qquad (2.2.1$$

con $l, k = 0,1,\ldots,N_1 - 1, n, j = 0,1,\ldots,N_2 - 1$.

Es importante enfatizar que el vector $\widetilde{\mathbf{X}}$ en la ecuación anterior no debe interpretarse como la transformada unidimensional del vector $\mathbf{X}$, pues de lo contrario esto implicaría que $\mathbf{W}_{N_1 \cdot N_2} = \mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2}$ lo cual no es necesariamente cierto en general. No obstante, como se verá en la sección 2.3, si se emplea en lugar de la función lexicográfica $\Lambda_2 : C^{N_1} \times C^{N_2} \rightarrow C^{N_1 \cdot N_2}$ funciones biyectivas $P_N, Q_N : C^{N_1} \times C^{N_2} \rightarrow C^{N_1 \cdot N_2}$ construidas en base al TCR y a la RER, será posible expresar a la matriz $\mathbf{W}_{N_1 \cdot N_2}$ en términos del producto tensorial $\mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2}$ y por consiguiente se establecerá un puente, hayado por *Good* 1958, entre la transformada unidimensional y la transformada bidimensional, generalizandose posteriormente esta idea también para la transformada multidimensional.

Definición de TFD multidimensional: La transformada L dimensional de un arreglo $\mathbf{Z} \in C^{N_1} \times C^{N_2} \times \ldots \times C^{N_L}$ es un arreglo $\hat{\mathbf{Z}} \in C^{N_1} \times C^{N_2} \times \ldots \times C^{N_L}$ dado por

$$\hat{\mathbf{Z}}\left(k_1, k_2, \ldots, k_L\right) = \frac{1}{N} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_L=0}^{N_L-1} \mathbf{Z}\left(n_1, n_2, \ldots, n_L\right) \times \omega^{k_1 n_1 N / N_1} \omega^{k_2 n_2 N / N_2} \cdots \omega^{k_L n_L N / N_L} \tag{2.2.13}$$

donde $N = N_1 \cdot N_2 \cdot \ldots \cdot N_L$, $\omega = e^{-i 2\pi / N}$ y $n_i, k_i = 0,1,\ldots,N_i$, $i = 0,1,\ldots,L$.

De forma semejante a como se procedió en la transformada bidimensional, se definirá en este caso la función biyectiva $\wedge_L : C^{N_1} \times C^{N_2} \times \ldots \times C^{N_L} \rightarrow C^{N_1 \cdot N_2 \ldots N_L}$

$$\wedge_L \left(\mathbf{A} = \left\{ \alpha_{(n_1 n_2 \ldots n_L)} \right\} \right) = \mathbf{B} = \left\{ \beta_{\left(n_1 \cdot \frac{N}{N_1} + n_1 \cdot \frac{N}{N_1 \cdot N_2} + \ldots + n_L\right)} \right\} \tag{2.2.14}$$

con $n_i = 0,1,\ldots,N_i - 1$, $i = 0,1,\ldots,N-1$.

En base a esta función $\wedge_L$ se construyen los vectores $\mathbf{X}, \widetilde{\mathbf{X}} \in C^{N_1 \cdot N_2 \ldots N_L}$

$$\begin{aligned} \widetilde{\mathbf{X}} &= \wedge_L \left(\mathbf{Z} \right) & \widetilde{\mathbf{X}}_{\left(n_1 \cdot \frac{N}{N_1} + n_2 \cdot \frac{N}{N_1 \cdot N_2} + \ldots + n_L\right)} &= \mathbf{Z}_{(n_1 n_2 \ldots n_L)} \\ \widetilde{\mathbf{X}} &= \wedge_L \left(\hat{\mathbf{Z}} \right) & \widetilde{\mathbf{X}}_{\left(k_1 \cdot \frac{N}{N_1} + k_2 \cdot \frac{N}{N_1 \cdot N_2} + \ldots + k_L\right)} &= \hat{\mathbf{Z}}_{(k_1 k_2 \ldots k_L)} \end{aligned} \tag{2.2.15}$$

con $n_i, k_i = 0,1,\ldots,N_i - 1$, $i = 0,1,\ldots,N-1$.

Por tanto al proceder de una manera semejante a como se hizo para el caso de la transformada bidimensional, se llega a la siguiente representación matricial para la transformada L dimensional

$$\widetilde{\mathbf{X}} = \frac{1}{N} \mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2} \otimes \cdots \otimes \mathbf{W}_{N_L} \mathbf{X} \tag{2.2.16}$$

donde los vectores $\mathbf{X}$ y $\widetilde{\mathbf{X}}$ estan relacionados con los arreglos $\mathbf{Z}$ y $\hat{\mathbf{Z}}$ a través de la ecuación 2.2.15.

2.3. Lema de Good

Basados en el TCR y en la RER es posible construir matrices de permutación $\mathbf{P}_N$ y $\mathbf{Q}_N$ que reordenen los renglones y las columnas de la matriz $\mathbf{W}_N$ de tal manera que la matriz resultante sea el producto tensorial de matrices $\mathbf{W}_{N_i}$, siempre y cuando se tenga que $N = N_1 \cdot N_2 \cdot \ldots \cdot N_L$, $(N_i, N_j) = 1, i \neq j, ij = 1,2,\ldots,L$, es decir $\mathbf{P}_N \mathbf{W}_N \mathbf{Q}_N = \mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2} \otimes \cdots \otimes \mathbf{W}_{N_L}$

Por ejemplo, supóngase que $N = N_1 \cdot N_2$ con $(N_1, N_2) = 1$, entonces empleando el TCR, se construye la matriz de permutación $\mathbf{P}_N$ que permutará el renglón $R = N_1 I + J$, $I = 0,1,\ldots,N_2 - 1$, $J = 0,1,\ldots,N_1 - 1$ de la matriz $\mathbf{W}_N$ a la posición

$$< J \left(\frac{N}{N_1}\right)^{\varphi(N_1)} + I \left(\frac{N}{N_2}\right)^{\varphi(N_2)} >_N \qquad (2.3$$

en donde $< x >_N$ denota x *Mod* N. De manera más sencilla se tiene que la matriz $\mathbf{P}_N$ permutará los renglones R, $0 \leq R < N$, de la matriz $\mathbf{W}_N$ mediante la siguiente regla

$$R = N_1 I + J < \text{-----} \mathbf{P}_N \text{-----} > < J \left(\frac{N}{N_1}\right)^{\varphi(N_1)} + I \left(\frac{N}{N_2}\right)^{\varphi(N_2)} >_N \qquad (2.3.2$$

con $I = 0,1,\ldots,N_2 - 1$, $J = 0,1,\ldots,N_1 - 1$. De manera semejante, empleando la RER, la matriz $\mathbf{Q}_N$ permutará las columnas C, $0 \leq C < N$, de la matriz $\mathbf{W}_N$ como sigue

$$C = N_1 K + L < \text{-----} \mathbf{Q}_N \text{-----} > < L \left(\frac{N}{N_1}\right) + K \left(\frac{N}{N_2}\right) >_N \qquad (2.3.3$$

con $K = 0,1,\ldots,N_2 - 1$, $L = 0,1,\ldots,N_1 - 1$. Por tanto, siguiendo con el ejemplo se tiene que

$$\mathbf{P}_N \mathbf{W}_N \mathbf{Q}_N = \mathbf{P}_N \left\{ \omega_N^{RC} \right\} \mathbf{Q}_N$$

$$= \left\{ \omega_N^{< JN_2^{\varphi(N_1)} + IN_1^{\varphi(N_2)} >_N < LN_2 + KN_1 >_N} \right\} \qquad (2.3 -$$

y al realizar el producto $< \quad [J\cdot N_2{}^{\varphi(N_1)} + I\cdot N_1{}^{\varphi(N_2)}][L\cdot N_2 + K\cdot N_1]>_N$

$$= \left\{ \omega \overset{<}{N} JLN_2N_2^{\varphi(N_1)} + IKN_1N_1^{\varphi(N_2)} >_N \right\}$$

$$= \left\{ \omega \overset{<}{N} JLN_2N_2^{\varphi(N_1)} >_N \cdot \omega \overset{<}{N} IKN_1N_1^{\varphi(N_1)} + IKN_1N_1^{\varphi(N_2)} >_N \right\} \qquad (2.3.5$$

como $\omega_N^{N_2} = \omega_{N_1}$, $\omega_N^{N_1} = \omega_{N_2}$ y $< \left(\frac{N}{N_i}\right)^{\varphi(N_i)} >_N = \delta ij$ entonces

$$\mathbf{P}_N\mathbf{W}_N\mathbf{Q}_N = \left\{ \omega_{N_1}^{JN} \cdot \omega_{N_2}^{IK} \right\} = \left\{ \omega_{N_1}^{JN}\mathbf{W}_{N_2} \right\} \qquad (2.3.6$$

con $J,L = 0,1,...,N_1 - 1$, $I,K = 0,1,...,N_2 - 1$ y empleando el producto tensorial se llega a

$$\mathbf{P}_N\mathbf{W}_N\mathbf{Q}_N = \mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2} \qquad (2.3.7$$

2.4. Algoritmo de Good

El resultado del ejemplo de la sección anterior puede ser generalizado como sigue

Si $N = N_1\cdot N_2\cdot \ldots \cdot N_L$, con $(N_i, N_j) = 1$, $i \neq j$, $i,j = 1,2,...,L$ entonces existen matrices de permutación $\mathbf{P}_N$ y $\mathbf{Q}_N$ tales que

$$\mathbf{P}_N\mathbf{W}_N\mathbf{Q}_N = \mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2} \otimes \cdots \otimes \mathbf{W}_{N_L} \qquad (2.4.1$$

Eficiencia computacional: La ecuación anterior constituye el algoritmo de *Good*. Como se ve este algoritmo permite expresar una transformada unidimensional en términos de una transformada multidimensional. Por tanto, empleando este algoritmo el cálculo de una transformada de orden $N = N_1\cdot N_2\cdot \ldots \cdot N_L$, $(Ni, Nj) = 1$ puede ser realizado como sigue

$$\hat{\mathbf{Z}} = \frac{1}{N}\mathbf{P}_N^{-1}\left(\mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2} \otimes \cdots \otimes \mathbf{W}_{N_L}\right)\mathbf{Q}_N^{-1}\mathbf{Z} \qquad (2.4.2$$

es decir, el cálculo de esta transformada puede ser realizado en términos del cálculo de

3. TFD DE ORDEN N = P^k COMO UNA CONVOLUCION CICLICA

En este capítulo la atención se centrará en el problema del cálculo de $\hat{\mathbf{Z}} = \frac{1}{P^k} \mathbf{W}_{P^k} \mathbf{Z}$ con P un primo impar. La impresión que da a primera vista este problema, es que no es posible optimizar el cálculo de la transformada si su orden es primo o una potencia de un primo impar, pues es evidente que no son aplicables las técnicas empleadas para el desarrollo del algoritmo de *CooleyTuckey* en este caso. Sin embargo, basados en una idea original de *Rader* 1968, se verá que es posible realizar el cálculo de la transformada en términos de una convolución cíclica si su orden es un primo o una potencia de un primo mayor que dos. Ahora bien, podría cuestionarse el objeto de expresar a la transformada en términos de una convolución cíclica, debido a que aparentemente el cálculo de las convoluciones requiere de tanto o más esfuerzo computacional del que se requiere para realizar las transformadas. La razón de esto es que en la actualidad se cuenta con algoritmos muy eficientes para el cálculo de convoluciones cíclicas de ciertos órdenes, los cuales se han obtenido basados en nuevas herramientas matemáticas que serán estudiadas en detalle en el capítulo siguiente.

3.1. Convolución cíclica.

La convolución cíclica de orden N entre dos vectores complejos $\mathbf{g}$, $\mathbf{h}$ de dimensión N es otro vector $\mathbf{y}$ que se denota como $\mathbf{y} = \mathbf{g} * \mathbf{h}$ cuyas componentes son

$$y_i = \sum_{n=0}^{N-1} g_n \, h_{i-n} \qquad i = 0,1,\ldots,N-1 \tag{3.1}$$

donde los índices deberán interpretarse módulo N, es decir

$$h_{i-n} = h_{i-n+N} \tag{3.1}$$

con

$$\mathbf{y} = \left(y_0, y_1, \ldots, y_{N-1}\right)^t$$
$$\mathbf{g} = \left(g_0, g_1, \ldots, g_{N-1}\right)^t$$
$$\mathbf{h} = \left(h_0, h_1, \ldots, h_{N-1}\right)^t \tag{3.1}$$

3.2. Teorema de la convolución.

Si se definen a los vectores **Y**, **G** y **H** como

$$\begin{aligned}\mathbf{Y} &= \left(Y_0, Y_1, \ldots, Y_{N-1}\right)^t = \mathbf{W}_N \mathbf{y} \\ \mathbf{G} &= \left(G_0, G_1, \ldots, G_{N-1}\right)^t = \mathbf{W}_N \mathbf{g} \\ \mathbf{H} &= \left(H_0, H_1, \ldots, H_{N-1}\right)^t = \mathbf{W}_N \mathbf{h}\end{aligned} \tag{3.2.1}$$

donde

$$Y_k = \sum_{j=0}^{N-1} y_j \omega_N^{kj} \qquad k = 0,1,\ldots,N-1 \tag{3.2.2}$$

de manera análoga para G_k y H_k, entonces como $\mathbf{y} = \mathbf{g} \bullet \mathbf{h}$, se tiene que

$$Y_k = \sum_{j=0}^{N-1} \sum_{n=0}^{N-1} g_n h_{j-n} \omega_N^{kj} \tag{3.2.3}$$

Ahora bien, si se intercambia el orden de las sumatorias

$$Y_k = \sum_{n=0}^{N-1} g_n \sum_{j=0}^{N-1} h_{j-n} \omega_N^{kj} \tag{3.2.4}$$

y se define $l = j - n$

$$Y_k = \sum_{n=0}^{N-1} g_n \sum_{l=-n}^{N-1-n} h_l \omega_N^{k(l+n)} \tag{3.2.5}$$

pasando el término ω^{kn}, a la primera sumatoria y dado que el vector **h**, es N periódico se tiene

$$Y_k = \sum_{n=0}^{N-1} g_n \omega_N^{kn} \sum_{l=0}^{N-1} h_l \omega_N^{kl} = G_k H_k \tag{3.2.6}$$

El resultado anterior es conocido como el teorema de la convolución, que establece que la transformada de la convolución cíclica de dos vectores es igual al producto de sus transformadas, es decir si $y = g * h$ entonces

$$\mathbf{Y} = \mathbf{G} \circ \mathbf{H} \tag{3.2.7}$$

donde $\circ$ es el producto de *Hadamar*, con lo cual se quiere decir que el producto $\mathbf{G} \circ \mathbf{H}$ debe efectuarse elemento por elemento de los vectores $\mathbf{G}$ y $\mathbf{H}$, es decir

$$\mathbf{G} \circ \mathbf{H} = \left(G_0 H_0 , G_1 H_1 , ..., G_{N-1} H_{N-1} \right)^t \tag{3.2.8}$$

3.3. TFD de orden N = P vía una convolución cíclica.

El siguiente teorema algebraico [1] es de gran utilidad pues como se verá en base a este es posible tratar a una transformada de orden P, con P primo impar, en términos de una convolución cíclica de orden $P - 1$

Teorema [1]: sea $Z_P = \{ 0,1,...,P-1 \}$ el grupo de los enteros módulo P, con P primo impar, entonces el grupo $G(P) = \{1,2,...,P-1\}$ es un grupo cíclico multiplicativo módulo P y por consiguiente deberá existir un $\alpha \in G(P)$ generador del grupo, es decir

$$G(P) = \{ <\alpha^m>_P \}$$

$$G(P) = \{ <\alpha^{-m}>_P \} \qquad m = 0,1,...,P-2$$

$$1 = <\alpha^{P-1}>_P \tag{3.3.1}$$

Este resultado fue empleado por primera vez por *Rader* 1968 [4] para expresar una transformada de orden primo en términos de una convolución cíclica mediante la siguiente técnica

El cálculo de una transformada de orden P (P primo impar) se realiza con base en la siguiente ecuación

$$\hat{\mathbf{Z}}_k = \frac{1}{P}\sum_{j=0}^{P-1} \omega_P^{kj}\,\mathbf{Z}_j \qquad k = 0,1,...,P-1 \tag{3.3.}$$

lo cual claramente puede ser realizado en partes

$$\hat{\mathbf{Z}}_0 = \frac{1}{P}\sum_{j=0}^{P-1} \mathbf{Z}_j$$

$$\hat{\mathbf{Z}}_k = \frac{1}{P}\left(\mathbf{Z}_0 + \sum_{j=1}^{P-1} \omega_P^{kj}\,\mathbf{Z}_j\right) \qquad k = 1,2,...,P-1 \tag{3.3.}$$

Ahora bien, como $G(P)$ es un grupo ciclico multiplicativo módulo P, entonces existe $\alpha \in G(P)$ generador del grupo, y por tanto para cada $k, j \in G(P)$ existen $n, m \in \{0,1,...,P-2\}$ tales que

$$k = <\alpha^n>_P$$
$$j = <\alpha^{-m}>_P \tag{3.3.}$$

De esta manera, el cálculo de $\hat{\mathbf{Z}}_k$ con $k = 1,2,...,P-1$ puede ser realizado según

$$\hat{\mathbf{Z}}_{\alpha^n} = \frac{1}{P}\left(\mathbf{Z}_0 + \sum_{m=0}^{P-2} \omega_P^{\alpha^n\alpha^{-m}}\,\mathbf{Z}_{\alpha^{-m}}\right)$$

$$= \frac{1}{P}\left(\mathbf{Z}_0 + \sum_{m=0}^{P-2} \omega_P^{\alpha^{n-m}}\,\mathbf{Z}_{\alpha^{-m}}\right) \qquad n = 0,1,...,P-2 \tag{3.3}$$

donde los términos α^n, α^{-m} y α^{n-m} deberán ser tomados módulo P. Como $<\alpha^{P-1}>_P = 1$, entonces se tiene que

$$\omega_P^{<\alpha^{n-m}>_P} = \omega_P^{<\alpha^{n-m+P-1}>_P} \tag{3.3}$$

y por tanto la sumatoria

$$\overline{Z}_{\alpha^n} = \sum_{m=0}^{P-2} \omega_P^{\alpha^{n-m}} Z_{\alpha^{-m}} \qquad l = 0,1,...,P-2 \tag{3.3.7}$$

corresponde a una convolución cíclica de orden $P-1$, entre los vectores

$$\widetilde{\omega} = \left(\omega_P^{\alpha^0}, \omega_P^{\alpha^1}, \omega_P^{\alpha^2}, ..., \omega_P^{\alpha^{P-2}}\right)^t$$

$$\widetilde{Z} = \left(Z_{\alpha^0}, Z_{\alpha^{-1}}, Z_{\alpha^{-2}}, ..., Z_{\alpha^{-P+2}}\right)^t \tag{3.3.8}$$

donde nuevamente los términos α^{algo} deberán ser tomados módulo P.

De esta forma hemos visto que es posible realizar el cálculo de una TFD de orden $P > 2$ con P primo, en términos de una convolución cíclica de orden $P-1$.

Para ejemplificar lo anterior se expresará el cálculo de una transformada de orden 13 en términos de una convolución cíclica de orden 12. En este caso el grupo cíclico $G(P)$ es $G(P=13) = \{1,2,...,12\}$ un generador α de $G(13)$ es $\alpha = 2$ y los vectores $\widetilde{\omega}$ y $\widetilde{Z}$ son

$$\widetilde{\omega} = \left(\omega_{13}^1, \omega_{13}^2, \omega_{13}^4, \omega_{13}^8, \omega_{13}^3, \omega_{13}^6, \omega_{13}^{12}, \omega_{13}^{11}, \omega_{13}^9, \omega_{13}^5, \omega_{13}^{10}, \omega_{13}^7, \right)^t$$

$$\widetilde{Z} = \left(Z_1, Z_7, Z_{10}, Z_5, Z_9, Z_{11}, Z_{12}, Z_6, Z_3, Z_8, Z_4, Z_2\right)^t \tag{3.3.9}$$

Entonces el cálculo de la transformada de orden 13 puede ser realizado a través de la siguiente convolución cíclica

$$\overline{Z} = \widetilde{\omega} * \widetilde{Z} \tag{3.3.10}$$

donde

$$\overline{Z} = \left(Z_1, Z_2, Z_4, Z_8, Z_3, Z_6, Z_{12}, Z_{11}, Z_9, Z_5, Z_{10}, Z_7\right)^t \tag{3.3.11}$$

y por lo que entonces

$$\hat{\mathbf{Z}}_0 = \frac{1}{13}\sum_{j=0}^{12} \mathbf{Z}_j$$

$$\hat{\mathbf{Z}}_k = \frac{1}{13}\left(\mathbf{Z}_o + \mathbf{Z}_k\right) \qquad k = 1,2,...,P-1 \tag{3.3.12}$$

3.4. TFD de orden $N = P^k$ vía una convolución cíclica

El resultado de la sección anterior, puede ser generalizado en el caso en que el orden N de la transformada sea una potencia de P, es decir $N = P^k$ con P un primo impar. Sólo que ahora los elementos de $G(P^k)$ serán todos aquellos enteros menores a P^k que sean primos relativos con P, es decir $G(P^k) = \{n : 0 \le n < P^k,\ (n, P) = 1\}$

Teorema [1]: sea $Z_P^k = \{0,1,...,P^k - 1\}$ el grupo de los enteros módulo P^k, con P primo impar, entonces el grupo $G(P^k) = \{n : 0 \le n < P^k,\ (n, P) = 1\}$ es un grupo cíclico multiplicativo módulo P^k y por consiguiente deberá existir un $\alpha \in G(P^k)$ generador del grupo, es decir

$$G(P^k) = \{<\alpha^m>_{P^k}\}$$

$$G(P) = \{<\alpha^{-m}>_{P^k}\} \qquad m = 0,1,...,P^{k-1}(P-2)-1$$

$$1 = <\alpha^{P^{k-1}(P-1)}>_{P^k} \tag{3.4.1}$$

De manera semejante a 3.3.8 y en base a $G(P^k)$ y α se determina una convolución cíclica de orden $P^{k-1}(P-1)$ entre los vectores

$$\widetilde{\omega} = \left(\omega_P^{\alpha_k^0}, \omega_P^{\alpha_k^1}, \omega_P^{\alpha_k^2}, ..., \omega_P^{\alpha_k^{P^k(P-1)-1}}\right)^t$$

$$\widetilde{\mathbf{Z}} = \left(\mathbf{Z}_{\alpha^0}, \mathbf{Z}_{\alpha^{-1}}, \mathbf{Z}_{\alpha^{-2}}, ..., \mathbf{Z}_{\alpha^{-P^k(P-1)+1}}\right)^t \tag{3.4}$$

Por tanto, parte del cálculo de la transformada de orden P^k, se realiza en términos de la convolución cíclica de la forma 3.3.10 y el cálculo restante, que involucra a los renglones y columnas $0,P,2P,...,(P^{k-1}-1)P$ de la matriz $\mathbf{W}_{P^k}$, que no han sido tomados en cuenta, queda en términos de dos TFD's de orden P^{k-1}, las cuales a su vez, pueden ser obtenidas en términos de convoluciones cíclicas. Es decir, para calcular la transformada de orden P^k se realiza primero la convolución cíclica entre los vectores $\tilde{\omega}$ y $\tilde{\mathbf{Z}}$ definidos en 3.4.2 lo cual da lugar al vector $\mathbf{Z}$.

$$\overline{\mathbf{Z}} = \tilde{\omega} * \tilde{\mathbf{Z}} \tag{3.4}$$

posteriormente, es necesario calcular directamente los elementos $\hat{Z}_{Pn}$ con $n \in A = \{0,1,2,...,P^{k-1}-1\}$ del vector transformado $\hat{\mathbf{Z}}$, que corresponden a los renglones de la matriz $\mathbf{W}_{P^k}$ que no se han incluido en la convolución cíclica 3.4.3. Para tales elementos es necesario realizar el siguiente cálculo

$$\begin{aligned}\hat{\mathbf{Z}}_{Pn} &= \frac{1}{P^k}\sum_{j=0}^{P^k-1}\omega_{P^k}^{Pnj}\mathbf{Z}_j \\ &= \frac{1}{P^k}\sum_{j=0}^{P^k-1}\omega_{P^{k-1}}^{nj}\mathbf{Z}_j \qquad n \in A\end{aligned} \tag{3.4}$$

Ahora bien, como los índices j corren para los valores $0,1,...,P^k-1$ entonces para cada una de estas j existen enteros $r \in A$ y $c \in B = \{0,1,...,P-1\}$ tales que $j = r + cP^{k-1}$ y por tanto

$$\begin{aligned}\hat{\mathbf{Z}}_{Pn} &= \frac{1}{P^k}\sum_{r\in A}\sum_{c\in B}\omega_{P^{k-1}}^{n(r+cP^{k-1})}\mathbf{Z}_{(r+cP^{k-1})} \\ &= \frac{1}{P^k}\sum_{r\in A}\omega_{P^{k-1}}^{nr}\sum_{c\in B}\mathbf{Z}_{(r+cP^{k-1})} \qquad n \in A\end{aligned} \tag{3.4}$$

pero como $\mathbf{W}_{P^{k-1}} = \left\{\omega_{P^{k-1}}^{lr}\right\}$, $n, r \in A$, entonces la igualdad anterior puede ser tratada en términos de una transformada de orden P^{k-1} la cual, como ya se dijo, puede a su vez ser obtenida en términos de una convolución cíclica.

En cuanto a las columnas de la matriz $\mathbf{W}_{P^k}$ que fueron omitidas en la convolución cíclica 3.4.3, es necesario realizar el siguiente cálculo

$$\mathbf{Y}_k = \sum_{j \in A} \omega_{p^k}^{k p_j} \mathbf{Z}_{p_j} = \sum_{j \in A} \omega_{p^{k-1}}^{k j} \mathbf{Z}_{p_j} \qquad k = 0,1,...,P^k - 1 \tag{3.4.6}$$

la cual puede ser tratada en términos de una segunda transformada de orden P^{k-1} e igual que la anterior esta puede ser cálculada en base a una convolución cíclica.

Finalmente, para los elementos $\hat{Z}_n, n \in \{0,1,...,P^k - 1\} - A$, del vector transformado $\hat{\mathbf{Z}}$, que no han sido obtenidas, se realiza el siguiente cálculo

$$\mathbf{Z}_n = \frac{1}{P^k}\left(\mathbf{Y}_n + \mathbf{Z}_n\right) \qquad n \in \{0,1,...,P^k - 1\} - A \tag{3.4.7}$$

Para ejemplificar lo anterior, considerese el cálculo de una transformada de orden $N = 3^2 = 9$. En este caso se tiene que $G(P^k = 9) = \{1,2,4,5,7,8\}$ y un generador α de $G(9)$ es $\alpha = 2$ y por tanto los vectores $\tilde{\omega}$ y $\tilde{\mathbf{Z}}$ son

$$\tilde{\omega} = \left(\omega_9^1, \omega_9^2, \omega_9^4, \omega_9^5, \omega_9^7, \omega_9^8\right)^t$$

$$\tilde{\mathbf{Z}} = \left(\mathbf{Z}_1, \mathbf{Z}_5, \mathbf{Z}_7, \mathbf{Z}_8, \mathbf{Z}_4, \mathbf{Z}_2\right)^t \tag{3.4.8}$$

con los cuales se determina el vector $\mathbf{Z}$ en base a la siguiente convolución cíclica de orden 6

$$\begin{pmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_2 \\ \mathbf{Z}_4 \\ \mathbf{Z}_8 \\ \mathbf{Z}_7 \\ \mathbf{Z}_5 \end{pmatrix} = \begin{pmatrix} \omega^1 \omega^5 \omega^7 \omega^6 \omega^4 \omega^2 \\ \omega^2 \omega^1 \omega^5 \omega^7 \omega^8 \omega^4 \\ \omega^4 \omega^2 \omega^1 \omega^5 \omega^7 \omega^8 \\ \omega^8 \omega^4 \omega^2 \omega^1 \omega^5 \omega^7 \\ \omega^7 \omega^8 \omega^4 \omega^2 \omega^1 \omega^5 \\ \omega^5 \omega^7 \omega^8 \omega^4 \omega^2 \omega^1 \end{pmatrix} \begin{pmatrix} \mathbf{Z}_1 \\ \mathbf{Z}_5 \\ \mathbf{Z}_7 \\ \mathbf{Z}_8 \\ \mathbf{Z}_4 \\ \mathbf{Z}_2 \end{pmatrix} \tag{3.4.9}$$

A continuación se procede a calcular aparte los elementos $\hat{Z}_0$, $\hat{Z}_3$ y $\hat{Z}_6$ del vector transformado $\hat{\mathbf{Z}}$

$$\begin{pmatrix}\hat{Z}_0\\ \hat{Z}_3\\ \hat{Z}_6\end{pmatrix} = \frac{1}{9}\begin{pmatrix}1&1&1&1&1&1&1&1&1\\ 1&\omega^3&\omega^6&1&\omega^3&\omega^6&1&\omega^3&\omega^6\\ 1&\omega^6&\omega^3&1&\omega^6&\omega^3&1&\omega^6&\omega^3\end{pmatrix}\begin{pmatrix}Z_0\\ Z_1\\ Z_2\\ Z_3\\ Z_4\\ Z_5\\ Z_6\\ Z_7\\ Z_8\end{pmatrix} \qquad (3.4.10$$

en donde se ha utilizado ω en lugar de ω_9 para simplificar la notación. Dado que $\omega_9^3 = \omega_3$, entonces el cálculo anterior puede ser expresado en términos de una TFD de orden 3

$$\begin{pmatrix}\hat{Z}_0\\ \hat{Z}_3\\ \hat{Z}_6\end{pmatrix} = \frac{1}{9}\begin{pmatrix}1&1&1\\ 1&\omega_3^1&\omega_3^2\\ 1&\omega_3^2&\omega_3^1\end{pmatrix}\begin{pmatrix}Z_0+Z_3+Z_6\\ Z_1+Z_4+Z_7\\ Z_2+Z_5+Z_8\end{pmatrix} \qquad (3.4.11$$

Ahora bien, para las columnas 0, 3 y 6 de la matriz W_9, que no han sido consideradas, se realiza el siguiente cálculo

$$\begin{pmatrix}Y_0\\ Y_1\\ Y_2\\ Y_3\\ Y_4\\ Y_5\\ Y_6\\ Y_7\\ Y_8\end{pmatrix} = \begin{pmatrix}1&1&1\\ 1&\omega^3&\omega^6\\ 1&\omega^6&\omega^3\\ 1&1&1\\ 1&\omega^3&\omega^6\\ 1&\omega^6&\omega^3\\ 1&1&1\\ 1&\omega^3&\omega^6\\ 1&\omega^6&\omega^3\end{pmatrix}\begin{pmatrix}Z_0\\ Z_3\\ Z_6\end{pmatrix} \qquad (3.4.1$$

lo cual puede ser expresado en términos de una segunda transformada de orden 3

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} Y_3 \\ Y_4 \\ Y_5 \end{pmatrix} = \begin{pmatrix} Y_6 \\ Y_7 \\ Y_8 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & \omega_3^1 & \omega_3^2 \\ 1 & \omega_3^2 & \omega_3^1 \end{pmatrix} \begin{pmatrix} Z_0 \\ Z_3 \\ Z_6 \end{pmatrix} \tag{3.4.13}$$

Por tanto el cálculo de la transformada de orden 9 se concluye con

$$\begin{pmatrix} \hat{Z}_1 \\ \hat{Z}_2 \\ \hat{Z}_4 \\ \hat{Z}_5 \\ \hat{Z}_7 \\ \hat{Z}_8 \end{pmatrix} = \frac{1}{9} \begin{pmatrix} Y_1 + \bar{Z}_1 \\ Y_2 + \bar{Z}_2 \\ Y_1 + \bar{Z}_4 \\ Y_2 + \bar{Z}_5 \\ Y_1 + \bar{Z}_7 \\ Y_2 + \bar{Z}_8 \end{pmatrix} \tag{3.4.14}$$

Ya que es posible realizar una transformada de orden $N = P^k$, P primo impar, en términos de convoluciones cíclicas, lo que interesa ahora es encontrar una manera de realizar las convoluciones cíclicas económicamente, lo cual será el tema del capítulo siguiente.

Es importante aclarar que existen algoritmos eficientes para el cálculo de transformadas de órdenes $2^1, 2^2, 2^3$ y 2^4, pero el desarrollo de tales algoritmos no se basa en las ideas expuestas en este capítulo, sino más bien en la simetría que presentan los elementos de la matriz $\mathbf{W}_{P^k}$, cuando P es 2. Para mayores detalles de estos algoritmos consultar el capítulo seis.

4. CALCULO ECONOMICO DE CONVOLUCION CICLICA

En este capítulo se estudiará la teoría matemática que permitirá desarrollar algoritmos eficientes para el cálculo de las convoluciones cíclicas. Para ello se mostrará primero, en la sección uno de este capítulo, que el problema de el cálculo de la convolución cíclica puede atacarse en términos de un producto polinomial. Por esta razón la siguiente sección será dedicada a la teoría de polinomios. Las dos últimas secciones del capítulo serán dedicadas en buena parte a mostrar en base a ejemplos como se emplea la teoría de las secciones uno y dos para desarrollar algoritmos óptimos para el cálculo de la transformada.

4.1. Cálculo de una convolución cíclica como el producto de dos polinomios.

En el capítulo anterior se mostró que el teorema de la convolución permite relacionar a la transformada de la convolución cíclica de orden N

$$\mathbf{y} = \left(y_0, y_1, \ldots, y_{N-1}\right)^t = \mathbf{g} * \mathbf{h} \tag{4.1.}$$

de los vectores **g** y **h**

$$\mathbf{g} = \left(g_0, g_1, \ldots, g_{N-1}\right)^t$$
$$\mathbf{h} = \left(h_0, h_1, \ldots, h_{N-1}\right)^t \quad y \quad h_{i-n} = h_{i-n+N} \tag{4.1.}$$

con la transformada individual de cada uno de estos vectores, mediante la siguiente expresión

$$\mathbf{Y} = \mathbf{G} \circ \mathbf{H} \tag{4.1.}$$

donde $\circ$ es el producto de *Hadamar* y **Y, G, H** son las transformaciones de *Fourier* de los vectores **y, g** y **h** definidas como

$$\begin{aligned}\mathbf{Y} &= \left(Y_0, Y_1, \ldots, Y_{N-1}\right)^t = \mathbf{W}_N \mathbf{y} \\ \mathbf{G} &= \left(G_0, G_1, \ldots, G_{N-1}\right)^t = \mathbf{W}_N \mathbf{g} \\ \mathbf{H} &= \left(H_0, H_1, \ldots, H_{N-1}\right)^t = \mathbf{W}_N \mathbf{h}\end{aligned} \tag{4.1.4}$$

o de manera equivalente, los elementos de los vectores $\mathbf{Y}$, $\mathbf{G}$ y $\mathbf{H}$ se definen como

$$Y_k = \sum_{j=0}^{N-1} y_j \omega_N^{kj} \qquad k = 0,1,\ldots,N-1 \tag{4.1.5}$$

de igual modo para G_k y H_k. Ahora bien, es posible reescribir $\mathbf{Y}_k$, $\mathbf{G}_k$ y $\mathbf{H}_k$ en términos de los valores de ciertos polinomios en z cuando $\mathbf{z} = \omega_N^k$, como por ejemplo

$$Y(z) = \sum_{j=0}^{N-1} y_j z^j \quad \textit{en donde} \quad Y(z)\Big|_{z=\omega^k} = Y_k \tag{4.1.6}$$

análogo para $G(z)$ y $H(z)$. De esta manera el producto $\mathbf{G}_k \mathbf{H}_k$, puede verse como el producto de los polinomios

$$\tilde{Y}(z) = G(z)\, H(z) \tag{4.1.7}$$

y como los polinomios $G(z)$ y $H(z)$ son polinomios de grado $N-1$, entonces el polinomio $\tilde{\mathbf{Y}}(z)$ es de la forma

$$\tilde{Y}(z) = \tilde{y}_0 + \tilde{y}_1 z + \tilde{y}_2 z^2 + \ldots + \tilde{y}_{2N-1} z^{2N-1} \tag{4.1.8}$$

dado que $z^N = 1$, $z^{N+n} = z^n$ para $z = \omega_N^k$ y como $\mathbf{Y}(z)$ es un polinomio de grado $N-1$, entonces

$$Y(z) = \tilde{y}_0 + \tilde{y}_N + \left[\tilde{y}_1 + \tilde{y}_{N-1}\right] z + \ldots + \left[\tilde{y}_{N-2} + \tilde{y}_{2N-2}\right] z^{N-2} + \tilde{y}_{N-1} z^{N-1} \tag{4.1.9}$$

pero el término derecho de la igualdad anterior resulta ser

$$\tilde{Y}(z)\ Mod\ (z^N - 1) \tag{4.1.10}$$

por tanto

$$Y(z) = G(z)H(z)\ Mod\ (z^N - 1) \tag{4.1.11}$$

donde ahora los coeficientes y_i, $i = 0,1,...,N-1$, del polinomio $\mathbf{Y}(z)$ resultan ser los elementos del vector $\mathbf{y}$, que se obtiene al realizar la convolución cíclica $\mathbf{g} \bullet \mathbf{h}$.

De esta manera, es posible analizar el problema de la convolucíón en términos del producto de dos polinomios. La ventaja de ver problema de esta manera, radica en que ahora es posible emplear la teoría de polinomios para tratar de reducir el cálculo necesario con el fin de obtener los coeficientes del polinomio $G(z)H(z)\ Mod\ (z^N - 1)$.

4.2. Teorema Chino del Residuo para Polinomios

Una manera de obtener los coeficientes del producto polinomial $G(z)H(z)\ Mod\ (z^N - 1)$ es mediante la interpolación de *Lagrange* de $G(z)H(z)$. Otra alternativa es mediante el Teorema Chino del Residuo para Polinomios (TCRP), que como se verá, más adelante, permite descomponer el producto anterior en productos más sencillos de calcular. Antes de esto se darán algunas definiciones y resultados relacionados con la teoría de polinomios

Anillo de polinomios: Sea F un campo, el conjunto $F[z]$ de todos los polinomios con coeficientes en F, constituye un anillo de polinomios sobre el campo F con las operaciones de suma y multiplicación usuales entre polinomios.

Un polinomio $P(z)$ sobre el campo F es una expresión matemática de la forma

$$P(z) = p_0 + p_1 z + \ldots + p_N z^N = \sum_{j=0}^{N} p_j z^j \tag{4.2.1}$$

donde el símbolo z es una indeterminante y los coeficientes $p_0, p_1, ..., p_N$ son elementos del campo F. El polinomio nulo $P(z) \equiv 0$ es aquel cuyos coeficientes p_j son todos cero.

1. $\widetilde{P}(z) <- - - P(z)$
   $\widetilde{Q}(z) <- - - Q(z)$
2. Mientras $\widetilde{Q}(z) \neq 0$ efectúe
   2.1. $R(z) <- - - \widetilde{P}(z)\ Mod\ \widetilde{Q}(z)$
   2.2. $\widetilde{P}(z) <- - - \widetilde{Q}(z)$
   2.3. $\widetilde{Q}(z) <- - - R(z)$
3. El máximo comun divisor de $P(z)$ y $Q(z)$ es: $\alpha \widetilde{P}(z)$

donde α es el elemento del campo F talque el polinomio $\alpha \widetilde{P}(z)$ sea mónico.

Una relación para los polinomios primos relativos: Sean $Q_1(z)$ y $Q_2(z)$ dos polinomios primos relativos sobre F, entonces existen polinomios $S_1(z)$ y $S_2(z)$ tales que

$$S_i(z)\ Mod\ Q_j(z) = \delta_{ij} \qquad i,j = 1,2 \tag{4.2.4}$$

tales polinomios $S_1(z)$ y $S_2(z)$, pueden ser obtenidos empleando el algoritmo de *Euclides* para polinomios.

Por ejemplo

$$\begin{aligned} Q_1(z) &= z^2 + z + 1 \\ Q_2(z) &= z - 1 \\ \left(Q_1(z), Q_2(z)\right) &= 1 \end{aligned} \tag{4.2.5}$$

Empleando el algoritmo de *Euclides* se llega a

$$3 = Q_1(z) - \left(z + 2\right) Q_2(z) = Q_1(z) - C(z) Q_2(z) \tag{4.2.6}$$

de lo cual se desprende que

$$\begin{aligned} C(z) Q_2(z)\ Mod\ Q_1(z) &= -3 \\ Q_1(z)\ Mod\ Q_2(z) &= 3 \end{aligned} \tag{4.2.7}$$

y los polinomios $S_i(z)$ satisfacen

$$S_i(z)\,Q_j(z) = \delta_{ij} \qquad i,j = 1,2,...,k \tag{4.2.1}$$

La demostración de este teorema es muy sencilla y se hará en tres pasos

Primer paso; la solución es única, supóngase que existe una segunda solución $Y_0(z)$. Entonces $Y(z)\ Mod\ Q_i(z) = Y_0(z)\ Mod\ Q_i(z)$ así que $Q_i(z) \mid [\,Y(z) - Y_0(z)\,]$, para toda $i = 1,2,...,k$, dado que los polinomios $Q_i(z)$ son primos relativos, se tiene que $Q(z) \mid [\,Y(z) - Y_0(z)\,]$, esto implica que $Y(z) = Y_0(z)\ Mod\ Q(z)$ y $Grad\,(Y_0(z)) > Grad\,(Y(z))$, lo cual contradice la aseveración de que la solución tiene grado menor que N. Por tanto la solución es única.

Segundo paso; la construcción de los polinomios $S_i(z)$, sea $\widetilde{Q}_i(z) = Q(z) / Q_i(z)$, esto implica que $Q_i(z)$ y $\widetilde{Q}_i(z)$ son primos relativos, por tanto existe un polinomio $S_i(z)$ tal que $S_i(z)\ Mod\ Q_i(z) = 1$ y $S_i(z)\ Mod\ \widetilde{Q}_i(z) = 0$. Pero si $S_i(z)\ Mod\ \widetilde{Q}_i(z) = 0$ implica que $\widetilde{Q}_i(z) \mid S_i(z)$, es decir $S_i(z)$ es de la forma $S_i(z) = C_i(z) \prod_{j \neq i} Q_j(z)$ para algún polinomio $C_i(z)$, de lo cual se desprende que $S_i(z)\ Mod\ Q_j(z) = 0$ con $i \neq j$.

Tercer paso; $[\sum_{i=1}^{k} Y_i(z) S_i(z)]\ Mod\ Q(z)$ es el polinomio $Y(z)$ que se busca, dado que $S_i(z)\ Mod\ Q_i(z) = \delta_{ij}$ entonces, claramente $[\sum_{i=1}^{k} Y_i(z) S_i(z)]\ Mod\ Q_i(z) = Y_i(z)$.

Polinomios ciclotómicos [2]: La ecuación $z^N = 1$ tiene como N soluciones sobre el campo de los complejos

$$\omega^m = e^{-i2\pi m/N} \qquad m = 0,1,...,N-1 \tag{4.2.14}$$

donde ω^m es la m-ésima raíz de la unidad. Por tanto;

$$z^N - 1 = \prod_{m=0}^{N-1} (z - \omega^m) \tag{4.2.1}$$

Por ejemplo,

$$z^4 - 1 = (z-1)(z+1)(z-i)(z+i) \tag{4.2.1}$$

La mayoría de los factores $(z - \omega^m), m = 0,1,..,N-1$, de $z^N - 1$, son polinomios con coeficientes en el campo de los complejos. Sin embargo, siempre es posible encontrar una factorización del polinomio $z^N - 1$ sobre el campo de los enteros. Esto se hace de la siguiente manera

$$z^N - 1 = \prod_{l \in F_N} Q_l(z) \qquad F_N = \{ l : l \mid N, l > 0 \} \tag{4.2.17}$$

En donde los polinomios $Q_l(z)$ son polinomios con coeficientes en los enteros, denominados polinomios ciclotómicos, los cuales se obtienen a su vez como

$$Q_l(z) = \prod_{k_l \in E_l} \left(z - \omega^{k_l} \right) \qquad l \in F_N \tag{4.2.18}$$

donde los conjuntos E_l están dados por

$$\begin{aligned} E_1 &= \{ 0 \} \\ E_l &= \left\{ k_l : k_l = r \left(\frac{N}{l} \right), 0 < r < l \text{ y } (r, l) = 1 \right\} \qquad l \in F_N \end{aligned} \tag{4.2.19}$$

Por ejemplo, para $z^{12} - 1$ se tiene

$$\begin{aligned} &F_{12} = \{ 1,2,3,4,6,12 \} \\ &Q_1(z) = z - 1 \quad Q_2(z) = z + 1 \quad Q_3(z) = z^2 + z + 1 \\ &Q_4(z) = z^2 + 1 \quad Q_6(z) = z^2 - z + 1 \quad Q_{12}(z) = z^4 - z^2 + 1 \\ &z^{12} - 1 = (z-1)(z+1)\left(z^2+z+1\right)\left(z^2+1\right)\left(z^2-z+1\right)\left(z^4-z^2+1\right) \end{aligned} \tag{4.2.20}$$

Los polinomios ciclotómicos además tienen las siguientes propiedades [2]:

1) Los coeficientes de $Q_l(z)$ son todos enteros.
2) $(Q_l(z), Q_m(z)) = 1, l \neq m$, sobre el campo de los números racionales.
3) *Grad* $(Q_l(z)) = \varphi(l)$.

Solución de $G(z)H(z)\ Mod\ (z^N - 1)$ a través del TCRP: Como ya se vió $z^N - 1$, puede factorizarse como el producto de polinomios ciclotómicos $Q_i(z)$ sobre el campo de los racionales

$$z^N - 1 = \prod_{i \in F_N} Q_i(z) \tag{4.2.2}$$

Ahora bien, por el TCRP, el polinomio $Y(z) = G(z)H(z)\ Mod\ (z^N - 1)$ puede ser hallado por

$$Y(z) = \left[\sum_{i \in F_N} Y_i(z)\, S_i(z) \right] Mod \left(z^N - 1 \right) \tag{4.2.2}$$

donde

$$\begin{aligned} Y_i(z) &= G_i(z)\, H_i(z)\ Mod\ Q_i(z) \\ G_i(z) &= G(z)\ Mod\ Q_i(z) \\ H(z) &= H_i(z)\ Mod\ Q_i(z) \end{aligned} \tag{4.2.2}$$

y

$$S_i(z)\ Mod\ Q_j(z) = \delta_{ij} \qquad i, j \in F_N \tag{4.2.2}$$

Observacion : A primera vista, pareciera que obtener los coeficientes del polinomio $Y(z)$, con ayuda del TCRP requiere de más cálculos que si se obtuviera directamente del producto $G(z)H(z)\ Mod\ (z^N - 1)$. Sin embargo, los polinomios $S_i(z)$ no dependen en lo absoluto de los polinomios $G(z)$ y $H(z)$, por esta razón pueden ser precalculados. Por otro lado, para la obtención de los polinomios $G_i(z)$ y $H_i(z)$ sólo se requieren sumas y además si el polinomio $H(z)$ es fijado con anterioridad, entonces al igual que los polinomios $S_i(z)$, los polinomios $H_i(z)$ pueden ser precalculados. Ahora bien, la obtención de los polinomios $Y_i(z)$ depende de los productos $G_i(z)H_i(z)$, lo que es muy semejante a lo que pasa con el polinomio $Y(z)$, que se encuentra en términos del producto $G(z)H(z)$. Sin embargo, dado que el grado de los polinomios $G_i(z)$ y $H_i(z)$ es $Grad\ (Q_i(z)) - 1$, que por lo general es mucho menor al grado de los polinomios $G(z)$ y $H(z)$, entonces de aquí que la obtención de los polinomios $Y_i(z)$ sea realizada de una manera más sencilla. Finalmente, una vez que se tienen los polinomios $Y_i(z)$ y $S_i(z)$ los coeficientes deseados del polinomio $Y(z)$, estarán en términos de combinaciones lineales de los coeficientes de los polinomios $Y_i(z)$.

4.3. Factorización S C T

El objetivo de esta sección será mostrar que mediante el empleo de la teoría descrita en el capítulo anterior y en las dos últimas secciones de éste, es posible desarrollar algoritmos eficientes para el cálculo de las convoluciones cíclicas de orden pequeño, estos algoritmos, como ya se comentó en el capitulo tres, se emplearán a su vez para el desarrollo de algoritmos eficientes para el cálculo de transformadas de orden pequeño.

También se verá en esta sección que es posible hallar una representación matricial de la forma

$$\hat{\mathbf{Z}} = \frac{1}{N}\mathbf{SCTZ} \tag{4.3.1}$$

para cada uno de los algoritmos eficientes que calculan transformadas pequeñas de orden $N = P^k$ (P, primo impar), donde $\mathbf{T}$ es una matriz de orden $M \times N$, cuyos elementos son enteros muy pequeños, por lo general ceros, unos o menos unos, la cual indica una serie de sumas (sumas de entrada) que involucran a los elementos del vector $\mathbf{Z}$. La matriz $\mathbf{C}$ es una matriz diagonal de orden $M \times M$, cuyos elementos son puramente reales o imaginarias, por medio de esta matriz se indican todas las multiplicaciones que se han de realizar para el cálculo de la transformada pequeña. Por otro lado, la matriz $\mathbf{S}$ de orden $N \times M$ es muy semejante a la matriz $\mathbf{T}$, donde se indican también una serie de sumas (sumas de salida). El objetivo de tal representación matricial, es que esta puede ser manipulada con poderosas herramientas matriciales que permitirán llegar al los algoritmos de anidamiento y factor primo que serán estudiados en el capítulo siete.

Cálculo economico de una convolución ciclica de orden 4 : Para mostrar la técnica que se emplea para expresar una TFD en términos matriciales de la forma 4.3.1, considérese como ejemplo la transformada de orden 5. Dado que en este caso, el orden N de la transformada es de la forma $N = P = 5$, entonces, como ya se vió, su cálculo puede llevarse a cabo en términos de una convolución cíclica de orden 4

$$y_i = \sum_{n=0}^{3} g_n h_{i-n} \qquad i = 0,1,2,3 \tag{4.3.2}$$

la cual puede obtenerse en base al siguiente producto polinomial

$$Y(z) = G(z)H(z)\ Mod\ \left(z^4 - 1\right) \tag{4.3.3}$$

con

$$Y(z) = y_0 + y_1 z + y_2 z^2 + y_3 z^3$$
$$G(z) = g_0 + g_1 z + g_2 z^2 + g_3 z^3$$
$$H(z) = h_0 + h_1 z + h_2 z^2 + h_3 z^3 \qquad (4.3.4$$

Ahora bien, el polinomio $(z^4 - 1)$ es expresado como el producto de polinomios ciclotómicos

$$\left(z^4 - 1\right) = \left(z - 1\right)\left(z + 1\right)\left(z^2 + 1\right) = Q_1(z)\, Q_2(z)\, Q_4(z) \qquad (4.3.5$$

Por tanto, siguiendo la idea del TCRP es necesario calcular

$$\begin{aligned} G_1(z) &= G(z)\ Mod\left(z - 1\right) = g_0^1 \\ &= g_0 + g_1 + g_2 + g_3 \\ G_2(z) &= G(z)\ Mod\left(z + 1\right) = g_0^2 \\ &= g_0 - g_1 + g_2 - g_3 \\ G_4(z) &= G(z)\ Mod\left(z^2 + 1\right) = g_0^4 + g_1^4 z \\ &= g_0 - g_2 + \left(g_1 - g_3\right) z \end{aligned} \qquad (4.3.6$$

análogamente para $H_1(Z)$, $H_2(Z)$ y $H_4(Z)$. Para la obtención de los polinomios $Y_i(Z)$ se realizan los siguientes productos

$$Y_1(z) = G_1(z) H_1(z)\ Mod\left(z - 1\right) = y_0^1 = g_0^1 h_0^1$$

$$Y_2(z) = G_2(z) H_2(z)\ Mod\left(z + 1\right) = y_0^2 = g_0^2 h_0^2$$

$$\begin{aligned} Y_4(z) &= G_4(z) H_4(z)\ Mod\left(z^2 + 1\right) = y_0^4 + y_1^4 z \\ &= g_0^4 h_0^4 - g_1^4 h_1^4 + \left(g_0^4 h_1^4 + g_1^4 h_0^4\right) z \end{aligned} \qquad (4.3.$$

De esta manera, la obtención de los polinomios $Y_1(Z)$ y $Y_2(Z)$ requiere de una multiplicación cada uno

$$m_1 = g_0^1 h_0^1 = y_0^1$$
$$m_2 = g_0^2 h_0^2 = y_0^2 \qquad (4.3.$$

y en términos de los productos m_i

$$Y(z) = \frac{1}{4}\Big[\big(m_1 + m_2 + 2\big(m_3 - m_4\big)\big) + \big(m_1 - m_2 + 2\big(m_3 + m_5\big)\big) \big(m_1 + m_2 - 2\big(m_3 - m_4\big)\big) + \big(m_1 - m_2 - 2\big(m_3 + m_5\big)\big) \Big] \tag{4.3.14}$$

Por lo que entonces

$$\begin{pmatrix} y0 \\ y1 \\ y2 \\ y3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1- & 1 & 0 \\ 1- & 1 & 1 & 0 & 1 \\ 1 & 1- & 1 & 1 & 0 \\ 1- & 1- & 1 & 0- & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{4}m1 \\ \frac{1}{4}m2 \\ \frac{1}{2}m3 \\ \frac{1}{2}m4 \\ \frac{1}{2}m3 \end{pmatrix} \tag{4.3.15}$$

Combinando la expresión anterior con la encontrada para el vector **m** se tiene la siguiente expresión matricial

$$\begin{pmatrix} y0 \\ y1 \\ y2 \\ y3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1- & 1 & 0 \\ 1- & 1 & 1 & 0 & 1 \\ 1 & 1- & 1 & 1 & 0 \\ 1- & 1- & 1 & 0- & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{4}h_0^1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4}h_0^2 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2}h_0^4 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(h_0^4+h_1^4) & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(h_1^4-h_0^4) \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1- & 1 & 1- & 1 \\ 1 & 1- & 1- & 1 \\ 0 & 1 & 0- & 1 \\ 1 & 0- & 1 & 0 \end{pmatrix} \begin{pmatrix} g0 \\ g1 \\ g2 \\ g3 \end{pmatrix} \tag{4.3.16}$$

la cual, como se observa, es parecida a 4.3.1

Cálculo eficiente de una transformada de orden 5 : Ahora que ya se cuenta con el algoritmo óptimo para el cálculo de la convolución cíclica de orden 4, se empleará éste para desarrollar un algoritmo óptimo para la transformada de orden 5

El cálculo directo de la transformada de orden 5 se realiza como sigue

$$\begin{pmatrix} \hat{Z}_0 \\ \hat{Z}_1 \\ \hat{Z}_2 \\ \hat{Z}_3 \\ \hat{Z}_4 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 \\ \omega^0 & \omega^2 & \omega^4 & \omega^1 & \omega^3 \\ \omega^0 & \omega^3 & \omega^1 & \omega^4 & \omega^2 \\ \omega^0 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix} \begin{pmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \end{pmatrix} \qquad (4.3.17$$

Expresado este producto matricial en términos de una convolución cíclica de orden 4, empleando como generador de $G(5)$ a $\alpha = 2$, se tiene que

$$\begin{pmatrix} \tilde{Z}_0 \\ \tilde{Z}_2 \\ \tilde{Z}_4 \\ \tilde{Z}_3 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} \omega^1 & \omega^3 & \omega^4 & \omega^2 \\ \omega^2 & \omega^1 & \omega^3 & \omega^4 \\ \omega^4 & \omega^2 & \omega^1 & \omega^3 \\ \omega^3 & \omega^4 & \omega^2 & \omega^1 \end{pmatrix} \begin{pmatrix} Z_1 \\ Z_3 \\ Z_4 \\ Z_2 \end{pmatrix} \qquad (4.3.18$$

por lo que entonces

$$\hat{Z}_0 = \frac{1}{5}\left(Z_0 + Z_1 + Z_2 + Z_3 + Z_4\right)$$
$$\hat{Z}_k = \frac{1}{5}\left(Z_0 + \tilde{Z}_k\right) \qquad k = 1,2,3,4 \qquad (4.3.1$$

Ahora bien, la convolución cíclica 4.3.18 puede calcularse en base a 4.3.16 si se definen los vectores **y**, **g** y **h** de la siguiente manera

$$\mathbf{y} = \left(y0, y1, y2, y3\right)^t = \left(\tilde{Z}_1, \tilde{Z}_2, \tilde{Z}_4, \tilde{Z}_3\right)^t$$
$$\mathbf{g} = \left(g0, g1, g2, g3\right)^t = \left(Z_1, Z_3, Z_4, Z_2\right)^t$$
$$\mathbf{h} = \left(h0, h1, h2, h3\right)^t = \left(\omega^1, \omega^2, \omega^4, \omega^2\right)^t \qquad (4.3.2$$

y por tanto la diagonal de la matriz **C** estará dada por los siguientes valores

$$t1 = z1 + z4 \quad t2 = z3 + z2$$

$$t3 = z1 - z4 \quad t4 = z3 - z2$$

$$t5 = t1 + t2$$

$$m0 = 1\left(z0 + t5\right)$$

$$m1 = \left(c1 - 1\right) t5$$

$$m2 = c2\left(t1 - t2\right) \quad m3 = c3\left(t3 + t4\right)$$

$$m4 = c4\, t4 \quad m5 = c5\, t3$$

$$s1 = m0 + m1 \quad s2 = s1 + m2 \quad s3 = m3 - m4$$

$$s4 = s1 - m2 \quad s5 = m3 + m5$$

$$\hat{Z}0 = m0 \quad \hat{Z}1 = s2 + s3 \quad \hat{Z}2 = s4 + m5$$

$$\hat{Z}3 = s4 - s5 \quad \hat{Z}4 = s2 - s3 \qquad (4.3.24)$$

En donde el factor 1/5 ha sido omitido por simplicidad. Como puede observarse, este algoritmo permite calcular una transformada de orden 5 a un costo de 17 adiciones complejas y tan sólo 5 multiplicaciones semicomplejas (es decir 10 multiplicaciones reales), pues los elementos de la matriz **C** son puramente reales o imaginarias lo cual se debe a que los elementos se obtienen como suma o diferencia de conjugados.

B) Como se habrá notado, la elección de las multiplicaciones $m3$, $m4$ y $m5$ en el desarrollo anterior, ha sido un tanto rebuscada, de hecho el polinomio

$$Y4(z) = g0^4 h0^4 - g1^4 h1^4 + \left(g0^4 h1^4 + g1^4 h0^4\right) z \qquad (4.3.25)$$

puede verse como el resultado de la multiplicacíon de dos numeros complejos cuando la variable z es substitiuda por i, es decir

$$\left(g0^4 + g1^4 i\right)\left(h0^4 + h1^4 i\right) = g0^4 h0^4 - g1^4 h1^4 + \left(g0^4 h1^4 + g1^4 h0^4\right) i \qquad (4.3.26)$$

Por tanto, con las elecciones escogidas anteriormente para $m3$, $m4$ y $m5$, es posible realizar una multiplicación compleja a un costo de 3 multiplicaciones y 5 adiciones reales en lugar de las 4 multiplicaciones y 2 adiciones que normalmente suelen realizarse. Otra elección alternativa para $m3$, $m4$ y $m5$ podría ser

$$m_3 = h_0^4 g_0^4$$
$$m_4 = h_1^4 g_1^4$$
$$m_5 = \left(h_0^4 + h_1^4\right)\left(g_0^4 + g_1^4\right) \qquad (4.3.27$$

la cual, al igual que la eleccíon original, sólo requiere de 3 multiplicaciones y 5 adiciones para determinar los coeficientes y_0^4 y y_1^4. Sin embargo, dado que el vector **h** es fijado, entonces, con la elección inicial, las adiciones que involucran a m_4 y m_5 pueden ser precalculadas, ahorrándose así una adición con respecto a la anterior elección. Es claro entonces, por todo esto, que la elección de las multiplicaciones m_i es un tanto arbitraria debido a que uno busca la opción más eficiente. No obstante en la siguiente sección de este capítulo se verá cómo encontrar el número mínimo de multiplicaciones necesarias para determinar a los polinomios $Y_i(Z)$ y de qué manera construir tales polinomios con este mínimo de multiplicaciones.

4.4. Algoritmo de Toom Cook y teorema de Winograd

En algún momento a lo largo del desarrollo que se lleva a cabo para encontrar la representación matricial **S C T** para el cálculo de una transformada pequeña, es necesario determinar los coeficientes de los polinomios $Y_i(z)$, los cuales se hallan en términos de los productos $G_i(z)H_i(z)$. Dado que los polinomios $G_i(z)$ y $H_i(z)$ se obtienen a partir de los polinomios ciclotómicos $Q_i(z)$, $i \in F_N$

$$G_i(z) = G(z)\ Mod\ Q_i(z)$$
$$H_i(z) = H(z)\ Mod\ Q_i(z) \qquad i \in F_N \qquad (4.4.1$$

y como el grado de tales polinomios es

$$Grad\left(Q_i(z)\right) = \varphi(i) = \varphi_i \qquad i \in F_N \qquad (4.4.2$$

entonces es claro que

$$Grad\left(G_i(z)\right) = Grad\left(H_i(z)\right) = \varphi_i - 1 \qquad (4.4.3$$

y por tanto

$$Grad\left(G_i(z)\,H_i(z)\right) = 2\left(\varphi_i - 1\right) \qquad (4.4.4$$

De esta manera el polinomio $Y_i(z) = G_i(z)H_i(z) Mod\ Q_i(z)$ puede ser hallado empleando la fórmula de interpolación de *Lagrange*, evaluando el polinomio $G_i(z)H_i(z)$ en $2\varphi i - 1$ puntos $\{\alpha_j\}$ diferentes, es decir

$$m_j^i = G_i(\alpha_j)\, H_i(\alpha_j) \qquad \alpha_j \neq \alpha_k \qquad j, k = 1,2,....,2\varphi_i - 1$$

$$\widetilde{Y}_i(z) = \sum_{j=1}^{2\varphi_i - 1} m_j^i l_j(z)$$

$$Y_i(z) = \widetilde{Y}_i(z)\ Mod\ Q_i(z) \tag{4.4.5}$$

en donde es claro, se eligen para α_j valores lo más sencillo posible, quizá enteros muy pequeños. Los productos $m_j^i = G_i(\alpha_j)H_i(\alpha_j)$ pueden ser expresados matricialmente si se emplea el producto de *Hadamard*

$$\mathbf{m}^i = \mathbf{T}^i\mathbf{h}^i \circ \mathbf{T}^i\mathbf{g}^i \tag{4.4.6}$$

con

$$\mathbf{m}^i = \left(m_1^i, m_2^i, ..., m_{2\varphi_i - 1}^i\right)^t$$
$$\mathbf{h}^i = \left(h_0^i, h_1^i, ..., h_{\varphi_i - 1}^i\right)^t$$
$$\mathbf{g}^i = \left(g_0^i, g_1^i, ..., g_{\varphi_i - 1}^i\right)^t \tag{4.4.7}$$

y

$$\mathbf{T}^i = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{\varphi_i - 1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{\varphi_i - 1} \\ . & . & . & \cdots & . \\ 1 & \alpha_{2\varphi_i - 1} & \alpha_{2\varphi_i - 1}^2 & \cdots & \alpha_{2\varphi_i - 1}^{\varphi_i - 1} \end{pmatrix} \tag{4.4.8}$$

Como el vector $\mathbf{h}$ es fijado, entonces el vector $\mathbf{T}^i\,\mathbf{h}^i$ puede ser precalculado y expresado en términos de la matriz diagonal

$$\mathbf{C}^i = \mathbf{I}_{(2\varphi_i - 1)} \mathbf{T}^i \mathbf{h}^i \tag{4.4}$$

con lo que se tiene que

$$\mathbf{m}^i = \mathbf{C}^i \mathbf{T}^i \mathbf{g}^i \tag{4.4.1}$$

Ahora bien, como

$$Y_i(z) = \left[\sum_{j=1}^{2\varphi_i - 1} m_j^i l_j(z) \right] Mod\, Q_i(z) \tag{4.4.1}$$

entonces los coeficientes y_j^i, $i \in F_N$, $j = 0,1,...,\varphi_i - 1$, de los polinomios $Y_i(z)$ se encontrarán en términos de combinaciones lineales de los elementos de los vectores m^i, es decir, para cada vector m^i hay una matriz $\mathbf{S}^i$ de orden $\varphi_i \times 2\varphi_i - 1$ tal que

$$\mathbf{y}^i = \mathbf{S}^i \mathbf{m}^i$$
$$\mathbf{y}^i = \mathbf{S}^i \mathbf{C}^i \mathbf{T}^i \mathbf{g}^i \tag{4.4.1}$$

con

$$\mathbf{y}^i = \left(y_1^i, y_2^i, ..., y_{\varphi_i - 1}^i \right)^t \tag{4.4.1}$$

En donde los elementos de las matrices $\mathbf{S}^i$ y $\mathbf{T}^i$ serán números racionales si números racionales son escogidos para los valores de $\alpha_j, j = 1,2,...,2\varphi_i - 1$.

Por ejemplo, para el desarrollo del algoritmo óptimo para el cálculo de una convolución cíclica de orden 4, en la sección anterior, fue necesario realizar el siguiente producto

$$Y_4(z) = G_4(z) H_4(z)\, Mod \left(z^2 + 1 \right) \tag{4.4.}$$

donde

$$Y_4(z) = y_0^4 + y_1^4 z$$
$$G_4(z) = g_0^4 + g_1^4 z$$
$$H_4(z) = h_0^4 + h_1^4 z \qquad (4.4.15$$

Ahora bien, como $Grad(G_4(z)H_4(z)) = 2$, entonces será necesario interpolar a $G_4(z)H_4(z)$ en 3 puntos distintos, los cuales pudieran ser $\alpha 1 = -1, \alpha 2 = 0$ y $\alpha 3 = 1$. Con esta elección para los puntos de interpolación se tendría que el vector de multiplicaciones $\mathbf{m}^4$, dado a través de 4.4.10, sería

$$\mathbf{m}^4 = \begin{pmatrix} m_1^4 \\ m_2^4 \\ m_3^4 \end{pmatrix} = \begin{pmatrix} h_0^4 - h_1^4 & 0 & 0 \\ 0 & h_0^4 & 0 \\ 0 & 0 & h_0^4 + h_1^4 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} g_0^4 \\ g_1^4 \end{pmatrix} \qquad (4.4.16$$

y al emplear estas multiplicaciones en la fórmula general de *Lagrange* 4.4.11

$$G_4(z)H_4(z) = m_2^4 \frac{z(z+1)}{(1)(1+1)} + m_1^4 \frac{(z+1)(z-1)}{(1)(-1)} + m_0^4 \frac{z(z-1)}{(-1)(-1-1)}$$

$$= \left(\frac{1}{2}(m_2^4 + m_0^4) - m_1^4\right) z^2 + \frac{1}{2}(m_2^4 - m_0^4) z^2 + m_1^4 \qquad (4.4.17$$

Por tanto el polinomio $Y_4(z)$ estará dado en términos de los productos $\mathbf{m}_i^4$ como sigue

$$Y_4(z) = G_4(z)H_4(z) \; Mod\left(z^2 + 1\right)$$
$$= \frac{1}{2}(m_2^4 - m_0^4) z^2 + 2 m_1^4 - \frac{1}{2}(m_2^4 + m_0^4) = y_1^4 z + y_0^4 \qquad (4.4.18$$

y en términos matriciales

$$\mathbf{y}^4 = \begin{pmatrix} y_0^4 \\ y_1^4 \end{pmatrix} = \begin{pmatrix} -1/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 \end{pmatrix} \begin{pmatrix} m_1^4 \\ m_2^4 \\ m_3^4 \end{pmatrix}$$

$$= \begin{pmatrix} -1/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 \end{pmatrix} \begin{pmatrix} h_0^4 - h_1^4 & 0 & 0 \\ 0 & h_0^4 & 0 \\ 0 & 0 & h_0^4 + h_1^4 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} g_0^4 \\ g_1^4 \end{pmatrix} \qquad (4.4.1$$

Ahora bien, si el polinomio $H_4(z)$ es fijo y $G_4(z)$ es variable, entonces es conveniente precalcular las constantes

Teorema de Winograd: El número mínimo de multiplicaciones requeridas para calcular

$$Y(z) = G(z)\,H(z)\,Mod\,(z^N - 1) \tag{4.4.24}$$

es

$$2N - k \tag{4.4.25}$$

donde k es el número de de factores irreducibles $Q_i(z)$ sobre el campo de los racionales del polinomio $(z^N - 1)$

Es importante hacer énfacis sobre el papel que juega el campo de los racionales en el teorema anterior, pues al tomar como campo a los racionales se está asegurando que los coeficientes de los polinomios $Q_i(z)$ sean todos racionales (de hecho son todos enteros), evitándose de esta manera el trato con los coeficientes sobre el campo de los complejos, cuya aritmética requiere de un mayor esfuerzo computacional que la aritmética racional.

Teniendo en cuenta el algoritmo de *Toom Cook*, la demostración de este teorema es muy sencilla. Como ya se vió, el cálculo de los polinomios $Y_i(z)$, empleando la fórmula general de *Lagrange*, requiere de $2\varphi_i - 1$ multiplicaciones para cada polinomio y dado que existen $k = \#\,F_N$ polinomios $Y_i(z)$, entonces el número total de multiplicaciones necesarias para obtener $G(z)H(z)Mod\,(z^N - 1)$ es

$$T = \sum_{i \in F_N} 2\varphi_i - 1$$

$$= 2\sum_{i \in F_N} \varphi_i - k \tag{4.4.26}$$

Ahora bien, una propiedad de la función φ de *Euler* es

$$\sum_{i \in F_N} \varphi(i) = N \tag{4.4.27}$$

por tanto

$$T = 2N - k \tag{4.4.2}$$

Por ejemplo, para desarrollar un algoritmo óptimo para la transformada de orden 13, es necesario calcular $G(z)H(z)Mod\,(z^{12}-1)$. Por lo que el teorema anterior predice que se requieren de $2(12)-6=18$ multiplicaciones para su cálculo. Sin embargo, como se verá empleando este mismo ejemplo, el alcanzar este mínimo de multiplicaciones en ocasiones repercute de manera negativa en la matriz **S** cuyos elementos pudieran quedar constituidas por enteros no tan pequeños como se quisiera y en consecuencia se complicaría el cálculo para las sumas de salida.

Para el desarrollo de un algoritmo óptimo para el cálculo de una transformada de orden 13 se requiere calcular, como anteriormente se expuso, el polinomio $Y(z)=G(z)H(z)Mod\,(z^{12}-1)$. En esta ocasión el polinomio $(z^{12}-1)$ es factorizado en términos de los polinomios ciclotómicos

$$\begin{aligned} z^{12}-1 &= (z-1)(z+1)\left(z^2+z+1\right)\left(z^2+1\right)\left(z^2-z+1\right)\left(z^4-z^2+1\right) \\ &= Q_1(z)\,Q_2(z)\,Q_3(z)\,Q_4(z)\,Q_6(z)\,Q_{12}(z) \end{aligned} \tag{4.4.29}$$

A partir de estos polinomios se obtienen los polinomios $G_i(z)=G(z)Mod\,Q_i(z)$ y $H_i(z)=H(z)Mod\,Q_i(z)$, $i\in F_{12}=\{1,2,3,4,6,12\}$ (donde se empleará una notación semejante a la empleada en la sección anterior)

$$\begin{aligned} G_1(z) &= g_0^1 \\ G_2(z) &= g_0^2 \\ G_3(z) &= g_0^3+g_1^3 z \\ G_4(z) &= g_0^4+g_1^4 z \\ G_6(z) &= g_0^6+g_1^6 z \\ G_{12}(z) &= g_0^{12}+g_1^{12} z+g_2^{12} z^2+g_3^{12} z\,3 \end{aligned} \tag{4.4.30}$$

de igual modo se procede para los polinomios $H_i(z)$. Ahora se procede a calcular los polinomios $Y_i(z)=G_i(z)H_i(z)Mod\,Q_i(z)$

$$Y_1(z)=y_0^1=g_0^1 h_0^1$$

$$Y_2(z)=y_0^2=g_0^2 h_0^2$$

$$Y_3(z)=y_0^3+y_1^3 z=g_0^3 h_0^3-g_1^3 h_1^3+\left(g_0^3 h_1^3+g_1^3 h_0^3-g_1^3 h_1^3\right)z$$

$$Y_4(z)=y_0^4+y_1^4 z=g_0^4 h_0^4-g_1^4 h_1^4+\left(g_0^4 h_1^4+g_1^4 h_0^4\right)z$$

$$Y_6(z)=y_0^6+y_1^6 z=g_0^6 h_0^6-g_1^6 h_1^6+\left(g_0^6 h_1^6+g_1^6 h_0^6+g_1^6 h_1^6\right)z$$

$$720\,H_{12}(z)\,G_{12}(z) = \left(m_{12} - 6\,m_{13} + 15\,m_{14} - 20\,m_{15} + 15\,m_{16} - 6\,m_{17} - m_{18}\right) z^6 +$$

$$\left(-3\,m_{12} + 12\,m_{13} - 15\,m_{14} + 0\,m_{15} + 15\,m_{16} - 12\,m_{17} + 3\,m_{18}\right) z^5 +$$

$$\left(-5\,m_{12} + 60\,m_{13} - 195\,m_{14} + 280\,m_{15} - 195\,m_{16} + 60\,m_{17} - 5\,m_{18}\right) z^4 +$$

$$\left(15\,m_{12} - 120\,m_{13} + 195\,m_{14} + 0\,m_{15} - 195\,m_{16} + 120\,m_{17} - 5\,m_{18}\right) z^3 +$$

$$\left(4\,m_{12} - 54\,m_{13} + 540\,m_{14} - 980\,m_{15} + 540\,m_{16} - 54\,m_{17} + 4\,m_{18}\right) z^2 +$$

$$\left(-12\,m_{12} - 108\,m_{13} - 540\,m_{14} + 0\,m_{15} + 540\,m_{16} - 108\,m_{17} + 12\,m_{18}\right) z + 720 \qquad (4.4.34$$

con

$$m_j = G_{12}(\alpha_j)\,H_{12}(\alpha_j) \qquad \alpha_j \in \{-3,-2,-1,0,1,2,3\} \qquad j = 12,13,\ldots,18 \qquad (4.4.35$$

Por tanto, al interpolar $G_{12}(z)H_{12}(z)$ se logra el mínimo de multiplicaciones, pero como se puede apreciar, los coeficientes de las multiplicaciones m_j, aunque enteros, son muy grandes lo cual repercutiría negativamente en la matriz de sumas de salida **S**. Una alternativa para la solución del problema anterior es emplear algunas multiplicaciones más de las que indica el algoritmo de *Toom Cook*, por ejemplo

$$m_{12} = g_0^{12} h_0^{12} \qquad m_{13} = g_1^{12} h_1^{12} \qquad m_{14} = g_2^{12} h_2^{12} \qquad m_{15} = g_3^{12} h_3^{12}$$

$$m_{16} = \left(g_1^{12} + g_3^{12}\right)\left(h_1^{12} + h_3^{12}\right) \qquad m_{17} = \left(g_2^{12} + g_3^{12}\right)\left(h_2^{12} + h_3^{12}\right) \qquad m_{18} = \left(g_0^{12} + g_1^{12}\right)\left(h_0^{12} + h_1^{12}\right)$$

$$m_{19} = \left(g_0^{12} + g_2^{12}\right)\left(h_0^{12} + h_2^{12}\right) \qquad m_{20} = \left(g_0^{12} + g_3^{12}\right)\left(h_0^{12} + h_3^{12}\right) \qquad m_{21} = \left(g_1^{12} + g_2^{12}\right)\left(h_1^{12} + h_2^{12}\right) \qquad (4.4.3$$

con las cuales, ahora el polinomio $Y_{12}(z)$ quedará determinado como sigue

$$Y_{12}(z) = \left(m_{12} + m_{13} - m_{14} - m_{16}\right) +$$

$$\left(m_{14} + m_{18} - m_{12} - m_{13} + m_{15} - m_{17}\right) z +$$

$$\left(m_{16} - m_{12} - m_{15} + m_{19}\right) z^2 +$$

$$\left(m_{17} - m_{12} - m_{13} + m_{20} + m_{21} - 2m_{14} - 2m_{15}\right) z^3 +$$

$$= y_0^{12} + y_1^{12} z + y_2^{12} z^2 + y_3^{12} z^3 \qquad (4.4.3$$

De esta forma, se han empleado 3 multiplicaciones más de lo establecido por el algoritmo de *Toom Cook*, pero como se observa los coeficientes del polinomio $Y_{12}(z)$ se encuentran expresados en términos de las multiplicaciones de una manera mucho más sencilla de lo que se tenía anteriormente.

Por todo lo anterior, es pues importante aclarar que el algoritmo de *Toom Cook*, da una manera de llegar a la factorización **S C T** empleando un mínimo de multiplicaciones. Sin embargo, en la práctica uno trata de alcanzar este mínimo de la manera más óptima, desde el punto de vista computacional, consiguiéndose esto, muchas veces, sin necesidad de interpolar.

el cálculo de la transformada se realiza en términos de

$$yi = \sum_{n=0}^{11} gn\, hi - n$$

$$\hat{\mathbf{Z}}_0 = \frac{1}{13} \sum_{j=0}^{12} \mathbf{Z}_j$$

$$\hat{\mathbf{Z}}_1 = \frac{1}{13} \left(\mathbf{Z}_0 + \mathbf{y}_0 \right)$$

$$\hat{\mathbf{Z}}_2 = \frac{1}{13} \left(\mathbf{Z}_0 + \mathbf{y}_1 \right)$$

$$\hat{\mathbf{Z}}_3 = \frac{1}{13} \left(\mathbf{Z}_0 + \mathbf{y}_4 \right)$$

$$\hat{\mathbf{Z}}_{12} = \frac{1}{13} \left(\mathbf{Z}_0 + \mathbf{y}_6 \right) \qquad (5.4$$

Como ya se sabe, los valores yi de la convolución cíclica serán los coeficientes del polinomio

$$Y(z) = G(z)H(z)\ Mod\ (z^{12} - 1) \qquad (5.5$$

con

$$Y(z) = \sum_{j=0}^{11} yj\, z^j$$

$$G(z) = \sum_{j=0}^{11} gj\, z^j$$

$$H(z) = \sum_{j=0}^{11} hj\, z^j \qquad (5.6$$

El polinomio $(z^{12} - 1)$ es factorizado en términos de los polinomios ciclotómicos

$$z^{12} - 1 = (z - 1)(z + 1)(z^2 + z + 1)(z^2 + 1)(z^2 - z + 1)(z^4 - z^2 + 1)$$
$$= Q_1(z)\, Q_2(z)\, Q_3(z)\, Q_4(z)\, Q_6(z)\, Q_{12}(z) \qquad (5.$$

Ahora bien, siguiendo la idea del TCRP es necesario calcular para cada $i \in F_{12} = \{1,2,3,4,6,12\}$ los polinomios $G_i(z) = G(z) \; Mod \; Q_i(z)$ y $H_i(z) = H(z) \; Mod \; Q_i(z)$

$$G_1(z) = G(z) \; Mod \; (z-1) = g_0^1$$
$$= g0 + g1 + g2 + g3 + g4 + g5 + g6 + g7 + g8 + g9 + g10 + g11$$

$$G_2(z) = G(z) \; Mod \; (z+1) = g_0^2$$
$$= g0 - g1 + g2 - g3 + g4 - g5 + g6 - g7 + g8 - g9 + g10 - g11$$

$$G_3(z) = G(z) \; Mod \; (z^2 + z + 1) = g_0^3 + g_1^3 z$$
$$= (g0 - g2 + g3 - g5 + g6 - g8 + g9 - g11) +$$
$$(g1 - g2 + g4 - g5 + g7 - g8 + g10 - g11)\, z$$

$$G_4(z) = G(z) \; Mod \; (z^2 + 1) = g_0^4 + g_1^4 z$$
$$= (g0 - g2 + g4 - g6 + g8 - g10) + (g1 - g3 + g5 - g7 + g9 - g11)\, z$$

$$G_6(z) = G(z) \; Mod \; (z^2 - z + 1) = g_0^6 + g_1^6 z$$
$$= (g0 - g2 - g3 + g5 + g6 - g8 - g9 + g11) +$$
$$(g1 + g2 - g4 - g5 + g7 + g8 - g10 - g11)\, z$$

$$G_{12}(z) = G(z) \; Mod \; (z^4 - z^2 + 1) = g_0^{12} + g_1^{12} z + g_2^{12} z^2 + g_3^{12} z^3$$
$$= (g0 - g4 - g6 + g10) + (g1 - g5 - g7 + g11)\, z +$$
$$(g2 + g4 - g8 - g10)\, z^2 + (g3 + g5 - g9 - g11)\, z^3 \qquad (5.8$$

análogamente para los polinomios $H_i(z)$, $i \in F_{12} = \{1,2,3,4,6,12\}$. Las ecuaciones que involucran a los coeficientes de los polinomios $G_i(z)$ son las que determinarán las sumas de entrada, es decir determinarán a la matriz **T**.

Ahora se procede a calcular los polinomios $Y_i(z) = G_i(z) H_i(z) \; Mod \; Q_i(z)$, $i \in F_{12}$, lo cual dará lugar a la matriz **C** de multiplicaciones

$$Y_1(z) = y_0^1 = g_0^1 h_0^1$$

$$Y_2(z) = y_0^2 = g_0^2 h_0^2$$

$$Y_3(z) = y_0^3 + y_1^3 z = g_0^3 h_0^3 - g_1^3 h_1^3 + \left(g_0^3 h_1^3 + g_1^3 h_0^3 - g_1^3 h_1^3\right) z$$

$$Y_4(z) = y_0^4 + y_1^4 z = g_0^4 h_0^4 - g_1^4 h_1^4 + \left(g_0^4 h_1^4 + g_1^4 h_0^4\right) z$$

$$Y_6(z) = y_0^6 + y_1^6 z = g_0^6 h_0^6 - g_1^6 h_1^6 + \left(g_0^6 h_1^6 + g_1^6 h_0^6 + g_1^6 h_1^6\right) z$$

$$\begin{aligned} Y_{12}(z) &= y_0^{12} + y_1^{12} z + y_2^{12} z^2 + y_3^{12} z^3 \\ &= \left(g_0^{12} h_0^{12} - g_1^{12} h_3^{12} - g_2^{12} h_2^{12} - g_3^{12} h_1^{12} - g_3^{12} h_3^{12}\right) + \\ &\quad \left(g_0^{12} h_1^{12} + g_1^{12} h_0^{12} - g_2^{12} h_3^{12} - g_3^{12} h_2^{12}\right) z + \\ &\quad \left(g_0^{12} h_2^{12} + g_1^{12} h_1^{12} + g_2^{12} h_0^{12} + g_1^{12} h_3^{12} + g_2^{12} h_2^{12} + g_3^{12} h_1^{12}\right) z^2 + \\ &\quad \left(g_0^{12} h_3^{12} + g_1^{12} h_2^{12} + g_2^{12} h_1^{12} + g_3^{12} h_0^{12} + g_2^{12} h_3^{12} + g_3^{12} h_2^{12}\right) z^3 \end{aligned} \tag{5.9}$$

Nótese que los polinomios $Y_1(z)$ y $Y_2(z)$ requieren de una multiplicación cada uno

$$\begin{aligned} m_1 &= g_0^1 h_0^1 = y_0^1 \\ m_2 &= g_0^2 h_0^2 = y_0^2 \end{aligned} \tag{5.10}$$

mientras que los polinomios $Y_3(z)$, $Y_4(z)$ y $Y_6(z)$ requieren tres multiplicaciones cada uno

$$\begin{aligned} &m_3 = g_0^3 h_0^3 \qquad m_4 = g_1^3 h_1^3 \qquad m_5 = \left(g_1^3 - g_0^3\right)\left(h_0^3 - h_1^3\right) \\ &y_0^3 = m_3 - m_4 \qquad y_1^3 = m_5 - m_3 \\ \\ &m_6 = g_0^4 h_0^4 \qquad m_7 = g_1^4 h_1^4 \qquad m_8 = \left(g_0^4 + g_1^4\right)\left(h_0^4 + h_1^4\right) \\ &y_0^4 = m_6 - m_7 \qquad y_1^4 = m_8 - m_6 - m_7 \\ \\ &m_9 = g_0^6 h_0^6 \qquad m_{10} = g_1^6 h_1^6 \qquad m_{11} = \left(g_0^6 + g_1^6\right)\left(h_0^6 + h_1^6\right) \\ &y_0^6 = m_9 - m_{10} \qquad y_1^6 = m_{11} - m_9 \end{aligned} \tag{5.11}$$

y de manera matricial

$$\begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{15} \\ m_{16} \\ m_{17} \\ m_{18} \\ m_{19} \\ m_{20} \\ m_{21} \end{pmatrix} = \mathbf{C} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 & 0 & 1 & -1 \\ -1 & 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & -1 & 1 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & -1 & -1 & -1 & -1 \\ 1 & 1 & 0 & 0 & -1 & -1 & -1 & -1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & 0 & -1 & 1 & -1 \\ 0 & 1 & 1 & 0 & 1 & -1 & 0 & -1 & -1 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \\ g_8 \\ g_9 \\ g_{10} \\ g_{11} \end{pmatrix} \qquad (5.14$$

donde los coeficientes c_i, $i = 1,2,...,21$ son los coeficientes precalculados de la matríz diagonal **C**, los cuales se obtienen a partir de la definición del vector **h** dada en 5.3 y de las ecuaciones 5.10-5.12, obteniéndose para en este caso

$$c_1 = (Cos\mu + Cos2\,\mu + Cos3\,\mu + Cos4\,\mu + Cos5\,\mu + Cos6\,\mu)/6$$
$$c_2 = (Cos\mu - Cos2\,\mu + Cos3\,\mu + Cos4\,\mu - Cos5\,\mu - Cos6\,\mu)/6$$
$$c_3 = (Cos\mu - Cos4\,\mu + Cos5\,\mu - Cos6\,\mu)/6$$
$$c_4 = (Cos2\,\mu + Cos3\,\mu - Cos4\,\mu - Cos6\,\mu)/6$$
$$c_5 = (Cos\mu - Cos2\,\mu - Cos3\,\mu + Cos5\,\mu)/6$$
$$c_6 = i\,(-Sen\mu - Sen3\,\mu + Sen4\,\mu)/6$$
$$c_7 = i\,(-Sen2\,\mu - Sen5\,\mu - Sen6\,\mu)/6$$
$$c_8 = i\,(-Sen\mu - Sen2\,\mu - Sen3\,\mu + Sen4\,\mu - Sen5\,\mu - Sen6\,\mu)/6$$
$$c_9 = (Cos\mu - Cos4\,\mu - Cos5\,\mu + Cos6\,\mu)/6$$
$$c_{10} = (Cos2\,\mu - Cos3\,\mu + Cos4\,\mu - Cos6\,\mu)/6$$
$$c_{11} = (Cos\mu + Cos2\,\mu - Cos3\,\mu - Cos5\,\mu)/6$$
$$c_{12} = i\,(-Sen\mu + Sen3\,\mu)/6$$
$$c_{13} = i\,(-Sen2\,\mu + Sen6\,\mu)/6$$

$$
\begin{aligned}
c_{14} &= i\,(-Sen3\,\mu - Sen4\,\mu)/6 \\
c_{15} &= (Sen5\,\mu - Sen6\,\mu)/6 \\
c_{16} &= i\,(-Sen2\,\mu + Sen5\,\mu)/6 \\
c_{17} &= i\,(-Sen3\,\mu - Sen4\,\mu + Sen5\,\mu - Sen6\,\mu)/6 \\
c_{18} &= i\,(-Sen\mu - Sen2\,\mu + Sen3\,\mu + Sen6\,\mu)/6 \\
c_{19} &= i\,(-Sen\mu - Sen4\,\mu)/6 \\
c_{20} &= i\,(-Sen\mu + Sen3\,\mu + Sen5\,\mu - Sen6\,\mu)/6 \\
c_{21} &= i\,(-Sen2\,\mu - Sen3\,\mu - Sen4\,\mu + Sen6\,\mu)/6
\end{aligned}
\tag{5.15}
$$

con $\mu = 2\pi/13$. Por tanto los coeficientes de los polinomios $Y_i(z)$ estarán dados a través del siguiente producto matricial

$$
\begin{pmatrix} y_0^1 \\ y_0^2 \\ y_0^3 \\ y_1^3 \\ y_0^4 \\ y_1^4 \\ y_0^6 \\ y_1^6 \\ y_0^{12} \\ y_1^{12} \\ y_2^{12} \\ y_3^{12} \end{pmatrix}
=
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -2 & -2 & 0 & 1 & 0 & 0 & 1 & 1
\end{pmatrix}
\begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \\ m_6 \\ m_7 \\ m_8 \\ m_9 \\ m_{10} \\ m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{15} \\ m_{16} \\ m_{17} \\ m_{18} \\ m_{19} \\ m_{20} \\ m_{21} \end{pmatrix}
\tag{5.16}
$$

Ahora bien, una vez que se cuenta con los coeficientes de los polinomios $Y_i(z)$, $i \in F_{12}$, la determinación del polinomio $Y(z)$ se realiza según

$$
\begin{aligned}
Y(z) = \big[\, & Y_1(z)\,S_1(z) + Y_2(z)\,S_2(z) + Y_3(z)\,S_3(z) + Y_4(z)\,S_4(z) + \\
& Y_6(z)\,S_6(z) + Y_{12}(z)\,S_{12}(z) \,\big]\; Mod\,(z^{12} - 1)
\end{aligned}
\tag{5.17}
$$

en donde los polinomios $S_i(z)$, $i \in F_{12}$ son tales que $S_i(z)\ Mod\ Q_j(z) = \delta_{ij}$, $i,j \in F_{12}$. Estos polinomios se encuentran a partir del algoritmo de *Euclides* para polinomios y para este caso se tiene que

$$
\begin{aligned}
S_1(z) &= (1 - z + z^2 - z^3 + z^4 - z^5 + z^6 - z^7 + z^8 - z^9 + z^{10} - z^{11})/12 \\
S_2(z) &= (1 + z + z^2 + z^3 + z^4 + z^5 + z^6 + z^7 + z^8 + z^9 + z^{10} + z^{11})/12 \\
S_3(z) &= (2 - z - z^2 + 2z^3 - z^4 - z^5 + 2z^6 - z^7 - z^8 + 2z^9 - z^{10} - z^{11})/12 \\
S_4(z) &= (2 - 2z^2 + 2z^4 - 2z^6 + 2z^8 - 2z^{10})/12 \\
S_6(z) &= (2 + z - z^2 - 2z^3 - z^4 + z^5 + 2z^6 + z^7 - z^8 - 2z^9 - z^{10} + z^{11})/12 \\
S_{12}(z) &= (4 + 2z^2 - 2z^4 - 4z^6 - 2z^8 + 2z^{10})/12
\end{aligned}
\tag{5.18}
$$

Por lo que entonces, al resolver la ecuación 5.17 en términos de los polinomios $Y_i(z)$ y $S_i(z)$ y depués de pasar la constante 1/12, que aparece en los polinomios $S_i(z)$, a la matriz **C** (por esta razón, en 5.15 las entradas c_j de la matriz **C** aparece el factor 1/6), se tiene que los coeficientes del polinomio $Y(z)$ estarán dados en base al siguiente producto matricial

$$
\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \end{pmatrix}
=
\begin{pmatrix}
1 & 1 & 2 & -1 & 2 & 0 & 2 & 1 & 4 & 0 & 2 & 0 \\
1 & -1 & -1 & 2 & 0 & 2 & 1 & 2 & 0 & 4 & 0 & 2 \\
1 & 1 & -1 & -1 & -2 & 0 & -1 & 1 & 2 & 0 & 4 & 0 \\
1 & -1 & 2 & -1 & 0 & -2 & -2 & -1 & 0 & 2 & 0 & 4 \\
1 & 1 & -1 & 2 & 2 & 0 & -1 & -2 & -2 & 0 & 2 & 0 \\
1 & -1 & -1 & -1 & 0 & 2 & 1 & -1 & 0 & -2 & 0 & 2 \\
1 & 1 & 2 & -1 & -2 & 0 & 2 & 1 & -4 & 0 & -2 & 0 \\
1 & -1 & -1 & 2 & 0 & -2 & 1 & 2 & 0 & -4 & 0 & -2 \\
1 & 1 & -1 & -1 & 2 & 0 & -1 & 1 & -2 & 0 & -4 & 0 \\
1 & -1 & 2 & -1 & 0 & 2 & -2 & -1 & 0 & -2 & 0 & -4 \\
1 & 1 & -1 & 2 & -2 & 0 & -1 & -2 & 2 & 0 & -2 & 0 \\
1 & -1 & -1 & -1 & 0 & -2 & 1 & -1 & 0 & 2 & 0 & -2
\end{pmatrix}
\begin{pmatrix} y_0^1 \\ y_0^2 \\ y_0^3 \\ y_1^3 \\ y_0^4 \\ y_1^4 \\ y_0^6 \\ y_1^6 \\ y_0^{12} \\ y_1^{12} \\ y_2^{12} \\ y_3^{12} \end{pmatrix}
\tag{5.19}
$$

Finalmente, la ecuación anterior junto con la ecuación 5.4 determinarán las entradas del vector transformado $\hat{\mathbf{Z}}$.

6. ALGORITMOS OPTIMOS PARA EL CALCULO DE TFD's PEQUEÑAS

En este capítulo se presentarán los algoritmos óptimos ya existentes para el cálculo de transformadas pequeñas [3]. El algoritmo que se presenta en este capítulo para el cálculo de la transformada de orden 13 fue uno de los resultados obtenidos en este trabajo.

Algoritmo para la transformada de orden 2; $\mu = 2\pi/2$

$$m_0 = 1\,(z_0 + z_1) \quad m_1 = 1\,(z_0 - z_1)$$

$$\hat{Z}_0 = m_0 \quad \hat{Z}_1 = m_1$$

Algoritmo para la transformada de orden 3; $\mu = 2\pi/3$

$$t_1 = z_1 + z_2$$

$$m_0 = 1\,(z_0 + t_1) \quad m_1 = (Cos\mu - 1)\,t_1 \quad m_2 = i\,Sen\mu\,(z_2 - z_1)$$

$$s_1 = m_0 + m_1$$

$$\hat{Z}_0 = m_0 \quad \hat{Z}_1 = s_1 + m_2 \quad \hat{Z}_2 = s_1 - m_2$$

Algoritmo para la transformada de orden 4; $\mu = 2\pi/4$

$$t_1 = z_0 + z_2 \quad t_2 = z_1 + z_3$$

$$m_0 = 1\,(t_1 + t_2) \quad m_1 = 1\,(t_1 - t_2) \quad m_2 = 1\,(z_0 - z_2) \quad m_3 = i\,(z_3 - z_1)$$

$$\hat{Z}_0 = m_0 \quad \hat{Z}_1 = m_2 + m_3 \quad \hat{Z}_2 = m_1 \quad \hat{Z}_3 = m_2 - m_3$$

Algoritmo para la transformada de orden 5; $\mu = 2\pi/5$

$$t_1 = z_1 + z_4 \quad t_2 = z_3 + z_2 \quad t_3 = z_1 - z_4 \quad t_4 = z_3 - z_2 \quad t_5 = t_1 + t_2$$

$$m_0 = 1\,(z_0 + t_5) \quad m_1 = \left[\frac{1}{2}\,(Cos\mu + Cos2\mu) - 1\right] t_5$$

$$m_2 = \frac{1}{2}\,(Cos\mu - Cos2\mu)\,(t_1 - t_2) \quad m_3 = -\,i\,Sen\mu\,(t_3 + t_4)$$

$$m4 = -i\left(Sen\mu + Sen2\mu\right) t4 \quad m5 = i\left(Sen\mu - Sen2\mu\right) t3$$

$$s1 = m0 + m1 \quad s2 = s1 + m2 \quad s3 = m3 - m4 \quad s4 = s1 - m2 \quad s5 = m3 + m5$$

$$\hat{Z}_0 = m0 \quad \hat{Z}_1 = s2 + s3 \quad \hat{Z}_2 = s4 + m5 \quad \hat{Z}_3 = s4 - s5 \quad \hat{Z}_4 = s2 - s3$$

Algoritmo para la transformada de orden 7; $\mu = 2\pi/7$

$$t1 = z1 + z6 \quad t2 = z2 + z5 \quad t3 = z3 + z4 \quad t4 = t1 + t2 + t4$$

$$t5 = z1 - z6 \quad t6 = z2 - z5 \quad t7 = z4 - z3$$

$$m0 = 1\left(z0 + t4\right)$$

$$m1 = \left[\frac{1}{3}\left(Cos\mu + Cos2\mu + Cos3\mu\right) - 1\right] t4 \quad m2 = \frac{1}{3}\left(2Cos\mu - Cos2\mu - Cos3\mu\right)\left(t1 - t3\right)$$

$$m3 = \frac{1}{3}\left(Cos\mu - 2Cos2\mu + Cos3\mu\right)\left(t3 - t2\right) \quad m4 = \frac{1}{3}\left(Cos\mu + Cos2\mu - 2Cos3\mu\right)\left(t2 - t1\right)$$

$$m5 = -i\frac{1}{3}\left(Sen\mu + Sen2\mu - Sen3\mu\right)\left(t5 + t6 + t7\right) \quad m6 = i\frac{1}{3}\left(2Sen\mu - Sen2\mu - Sen3\mu\right)\left(t7 - t5\right)$$

$$m7 = i\frac{1}{3}\left(Sen\mu - 2Sen2\mu - Sen3\mu\right)\left(t6 - t7\right) \quad m8 = i\frac{1}{3}\left(Sen\mu + Sen2\mu + 2Sen3\mu\right)\left(t5 - t6\right)$$

$$s1 = m0 + m1 \quad s2 = s1 + m2 + m3 \quad s3 = s1 - m2 - m4 \quad s4 = s1 - m3 + m4$$

$$s5 = m5 + m6 + m7 \quad s6 = m5 - m6 - m8 \quad s7 = m5 - m7 + m8$$

$$\hat{Z}_0 = m0 \quad \hat{Z}_1 = s2 + s5 \quad \hat{Z}_2 = s3 + s6 \quad \hat{Z}_3 = s4 - s7$$

$$\hat{Z}_4 = s4 + s7 \quad \hat{Z}_5 = s3 - s6 \quad \hat{Z}_6 = s2 - s5$$

Algoritmo para la transformada de orden 8; $\mu = 2\pi/8$

$$t1 = z0 + z4 \quad t2 = z2 + z6 \quad t3 = z1 + z5 \quad t4 = z1 - z5$$

$$t5 = z3 + z7 \quad t6 = z3 - z7 \quad t7 = t1 + t2 \quad t8 = t3 + t5$$

$$m0 = 1\left(t7 + t8\right) \quad m1 = 1\left(t7 - t8\right) \quad m2 = 1\left(t1 - t2\right) \quad m3 = 1\left(z0 - z4\right)$$

$$m4 = Cos\mu\ (t4 - t6) \quad m5 = i\ (t5 - t3) \quad m6 = i\ (z6 - z2) \quad m7 = -\ i\ Sen\mu\ (t4 + t6)$$

$$s1 = m3 + m4 \quad s2 = m3 - m4 \quad s3 = m6 + m7 \quad s4 = m6 - m7$$

$$\hat{Z}_0 = m0 \quad \hat{Z}_1 = s1 + s3 \quad \hat{Z}_2 = m2 + m4 \quad \hat{Z}_3 = s2 - s4$$

$$\hat{Z}_4 = m1 \quad \hat{Z}_5 = s2 + s4 \quad \hat{Z}_6 = m2 - m5 \quad \hat{Z}_7 = s1 - s3$$

Algoritmo para la transformada de orden 9; $\mu = 2\pi/9$

$$t1 = z1 + z8 \quad t2 = z2 + z7 \quad t3 = z3 + z6 \quad t4 = z4 + z5 \quad t5 = t1 + t2 + t4$$

$$t6 = z1 - z8 \quad t7 = z7 - z2 \quad t8 = z3 - z6 \quad t9 = z4 - z5 \quad t10 = t6 + t7 + t9$$

$$m0 = 1\ (z0 + t3 + t5) \quad m1 = \frac{3}{2} t3 \quad m2 = -\frac{1}{2} t5$$

$$m3 = \frac{1}{3}\ (2Cos\mu - Cos2\mu - Cos4\mu)\ (t1 - t2) \quad m4 = \frac{1}{3}\ (Cos\mu + Cos2\mu - 2Cos4\mu)\ (t2 - t4)$$

$$m5 = \frac{1}{3}\ (Cos\mu - 2Cos2\mu + Cos4\mu)\ (t4 - t1) \quad m6 = -\ i\ Sen3\mu\ t10 \quad m7 = -\ i\ Sen3\mu\ t8$$

$$m8 = i\ Sen\mu\ (t7 - t6) \quad m9 = i\ Sen4\mu\ (t7 - t9) \quad m10 = i\ Sen2\mu\ (t6 - t9)$$

$$s1 = m0 + m2 + m2 \quad s2 = s1 - m1 \quad s3 = s1 + m2 \quad s4 = s2 + m3 + m4 \quad s5 = s2 - m4 + m5$$

$$s6 = s2 - m3 - m5 \quad s7 = m7 + m8 + m9 \quad s8 = m7 - m9 + m10 \quad s9 = m7 - m8 - m10$$

$$\hat{Z}_0 = m0 \quad \hat{Z}_1 = s4 + s7 \quad \hat{Z}_2 = s5 - s8 \quad \hat{Z}_3 = s3 + m6 \quad \hat{Z}_4 = s6 + s9$$

$$\hat{Z}_5 = s6 - s9 \quad \hat{Z}_6 = s3 - m6 \quad \hat{Z}_7 = s5 + s8 \quad \hat{Z}_8 = s4 - s7$$

Algoritmo para la transformada de orden 13; $\mu = 2\pi/13$

$$t1 = z1 + z9 + z3 \quad t2 = z7 + z11 + z8 \quad t3 = z10 + z12 + z4 \quad t4 = z5 + z6 + z2$$

$$t5 = z1 + z12 \quad t6 = z7 + z6 \quad t7 = z10 + z3 \quad t8 = z5 + z8$$

$$t9 = z9 + z4 \quad t10 = z11 + z2 \quad t11 = z1 - z12 \quad t12 = z7 - z6$$

$$t13 = z10 - z3 \quad t14 = z5 - z8 \quad t15 = z9 - z4 \quad t16 = z11 - z2$$

$$t17 = t1 + t2 + t3 + t4 \quad t18 = t7 + t10 \quad t19 = t5 + t8 - t18 \quad t20 = t6 + t9 - t18$$

$$t21 = t1 - t3 \quad t22 = t2 - t4 \quad t23 = t10 - t7 \quad t24 = t5 - t8 + t23$$

$$t25 = t6 - t9 - t23 \quad t26 = t11 - t15 \quad t27 = t12 - t16 \quad t28 = t13 + t15 \quad t29 = t14 + t16$$

$$m1 = \left[\frac{1}{6}\left(Cos\mu + Cos2\,\mu + Cos3\,\mu + Cos4\,\mu + Cos5\,\mu + Cos6\,\mu\right) - 1\right] t17$$

$$m2 = \frac{1}{6}\left(Cos\mu - Cos2\,\mu + Cos3\,\mu + Cos4\,\mu - Cos5\,\mu - Cos6\,\mu\right)\left(t1 + t2 - t3 - t4\right)$$

$$m3 = \frac{1}{6}\left(Cos\mu - Cos4\,\mu + Cos5\,\mu - Cos6\,\mu\right) t19$$

$$m4 = \frac{1}{6}\left(Cos2\,\mu + Cos3\,\mu - Cos4\,\mu - Cos6\,\mu\right) t20$$

$$m5 = \frac{1}{6}\left(Cos\mu - Cos2\,\mu - Cos3\,\mu + Cos5\,\mu\right)\left(t20 - t19\right)$$

$$m6 = i\frac{1}{6}\left(-Sen\mu - Sen3\,\mu + Sen4\,\mu\right) t21$$

$$m7 = i\frac{1}{6}\left(-Sen2\,\mu - Sen5\,\mu - Sen6\,\mu\right) t22$$

$$m8 = i\frac{1}{6}\left(-Sen\mu - Sen2\,\mu - Sen3\,\mu + Sen4\,\mu - Sen5\,\mu - Sen6\,\mu\right)\left(t21 + t22\right)$$

$$m9 = \frac{1}{6}\left(Cos\mu - Cos4\,\mu - Cos5\,\mu + Cos6\,\mu\right) t24$$

$$m10 = \frac{1}{6}\left(Cos2\,\mu - Cos3\,\mu + Cos4\,\mu - Cos6\,\mu\right) t25$$

$$m11 = \frac{1}{6}\left(Cos\mu + Cos2\,\mu - Cos3\,\mu - Cos5\,\mu\right)\left(t21 + t22\right)$$

$$m12 = i\frac{1}{6}(- Sen\mu + Sen3\mu)\ t26$$

$$m13 = i\frac{1}{6}(- Sen2\mu + Sen6\mu)\ t27$$

$$m14 = i\frac{1}{6}(- Sen3\mu - Sen4\mu)\ t28$$

$$m15 = \frac{1}{6}\left(Sen5\mu - Sen6\mu\right)\ t29$$

$$m16 = i\frac{1}{6}(- Sen2\mu + Sen5\mu)\ \left(t27 + t29\right)$$

$$m17 = i\frac{1}{6}(- Sen3\mu - Sen4\mu + Sen5\mu - Sen6\mu)\ \left(t28 + t29\right)$$

$$m18 = i\frac{1}{6}(- Sen\mu - Sen2\mu + Sen3\mu + Sen6\mu)\ \left(t26 + t27\right)$$

$$m19 = i\frac{1}{6}(- Sen\mu - Sen4\mu)\ \left(t26 + t28\right)$$

$$m20 = i\frac{1}{6}(- Sen\mu + Sen3\mu + Sen5\mu - Sen6\mu)\ \left(t26 + t29\right)$$

$$m21 = yi\frac{1}{6}(- Sen2\mu - Sen3\mu - Sen4\mu + Sen6\mu)\ \left(t27 + t28\right)$$

$$s1 = m0 + m1 \quad s2 = s1 + m2 \quad s3 = s1 - m2 \quad s4 = m3 - m4$$

$$s5 = s4 + s4 \quad s6 = s2 + s5 \quad s7 = s2 - s4 \quad s8 = s3 + s5$$

$$s9 = s3 - s4 \quad s10 = m10 + m3 \quad s11 = s5 + s10 \quad s12 = s6 - s10$$

$$s13 = s7 - s10 \quad s14 = s8 - s10 \quad s15 = s9 - s10 \quad s16 = s9 + s11$$

$$s17 = s7 + s11 \quad s18 = m9 - m10 \quad s19 = m11 - m9 \quad s20 = s18 + s18$$

$$s21 = s19 + s19 \quad s22 = s19 + s20 \quad s23 = s18 + s21 \quad s24 = s19 - s18$$

$$s25 = m6 - m7 \quad s25 = s25 + s25 \quad s26 = m8 - m6 - m7 \quad s26 = s26 + s26$$

$$s27 = m12 + m13 - m14 \quad s28 = m15 - m17 \quad s29 = m14 + m15$$

$$s30 = s27 - m16 \quad s30 = s30 + s30 \quad s31 = m18 - s27 + s28 \quad s31 = s31 + s31$$

$$s32 = m16 + m19 - m12 - m15 \quad s32 = s32 + s32$$

$$s33 = m17 + m20 + m21 - s27 - s29 - s29 - m14 \quad s33 = s33 + s33$$

$$s34 = s30 + s30 \quad s35 = s31 + s31 \quad s36 = s32 + s32 \quad s37 = s33 - s33$$

$$\hat{Z}_0 = m0$$

$$\hat{Z}_1 = s12+s25+s22+s34+s32 \quad \hat{Z}_2 = s16+s26+s23+s35+s33 \quad \hat{Z}_3 = s17+s25-s23+s30+s32$$

$$\hat{Z}_4 = s13-s25+s24+s30+s36 \quad \hat{Z}_5 = s14+s26-s22-s31-s37 \quad \hat{Z}_6 = s15+s26-s24-s31+s33$$

$$\hat{Z}_7 = s15-s26-s24+s31-s33 \quad \hat{Z}_8 = s14-s26-s22+s31+s37 \quad \hat{Z}_9 = s13+s25+s24-s30-s36$$

$$\hat{Z}_{10} = s17-s25-s23+s30-s32 \quad \hat{Z}_{11} = s16-s26+s23-s35-s33 \quad \hat{Z}_{12} = s12-s25+s22-s34+s32$$

Nota: Se hace la aclaración de que es posible optimizar aún más las sumas de salida para este algoritmo, lo cual no fue hecho por razones de tiempo.

Algoritmo para la transformada de orden 16; $\mu = 2\pi/16$

$$t1 = z0 + z8 \quad t2 = z4 + z2 \quad t3 = z2 + z10 \quad t4 = z2 - z10 \quad t5 = z6 + z14$$

$$t6 = z6 - z14 \quad t7 = z1 + z9 \quad t8 = z1 - z9 \quad t9 = z3 + z11 \quad t10 = z3 - z11$$

$$t11 = z5 + z13 \quad t12 = z5 - z13 \quad t13 = z7 + z15 \quad t14 = z7 - z15$$

$$t15 = t1 + t2 \quad t16 = t3 + t5 \quad t17 = t15 + t16 \quad t18 = t7 + t11$$

$$t19 = t7 - t11 \quad t20 = t9 + t13 \quad t21 = t9 - t13 \quad t22 = t18 + t20$$

$$t23 = t8 + t14 \quad t24 = t8 - t14 \quad t25 = t10 + t12 \quad t26 = t10 - t12$$

$$m0 = 1\left(t17 + t22\right) \quad m1 = 1\left(t17 - t22\right) \quad m2 = 1\left(t15 - t16\right) \quad m3 = 1\left(t1 - t2\right)$$

$$m4 = 1\left(z0 - z8\right) \quad m5 = Cos2\mu\left(t19 - t21\right) \quad m6 = Cos2\mu\left(t4 - t6\right)$$

$$m7 = Cos3\mu\left(t24 + t26\right) \quad m8 = \left(Cos\mu + Cos3\mu\right)t24 \quad m9 = \left(Cos3\mu - Cos\mu\right)t26$$

$$m10 = i\left(t20 - t18\right) \quad m11 = i\left(t5 - t3\right) \quad m12 = i\left(z12 - z4\right)$$

$$m13 = -i\,Sen2\mu\left(t19 + t21\right) \quad m14 = -i\,Sen2\mu\left(t4 + t6\right) \quad m15 = -i\,Sen3\mu\left(t23 + t25\right)$$

$$m16 = i\left(Sen3\mu - Sen\mu\right)t23 \quad m17 = -i\left(Sen\mu + Sen3\mu\right)t25$$

$$s1 = m3 + m5 \quad s2 = m3 - m5 \quad s3 = m11 + m13 \quad s4 = m13 - m11 \quad s5 = m4 + m6$$

$$s6 = m4 - m6 \quad s7 = m8 - m7 \quad s8 = m9 - m7 \quad s9 = m5 + m7 \quad s10 = m5 - m7$$

$$s11 = m6 + m8 \quad s12 = m6 - m8 \quad s13 = m12 + m14 \quad s14 = m12 - m14 \quad s15 = m15 + m16$$

$$s16 = m15 - m17 \quad s17 = s13 + s15 \quad s18 = s13 - s15 \quad s19 = s14 + s16 \quad s20 = s14 - s16$$

$$\hat{Z}_0 = m0 \quad \hat{Z}_1 = s9 + s17 \quad \hat{Z}_2 = s1 + s3 \quad \hat{Z}_3 = s12 - s20$$

$$\hat{Z}_4 = m2 + m10 \quad \hat{Z}_5 = s11 + s19 \quad \hat{Z}_6 = s2 + s4 \quad \hat{Z}_7 = s10 - s18$$

$$\hat{Z}_8 = m1 \quad \hat{Z}_9 = s10 + s18 \quad \hat{Z}_{10} = s2 - s4 \quad \hat{Z}_{11} = s11 - s19$$

$$\hat{Z}_{12} = m2 - m10 \quad \hat{Z}_{13} = s12 + s20 \quad \hat{Z}_{14} = s1 - s3 \quad \hat{Z}_{15} = s9 - s17$$

7. ALGORITMOS DE ANIDAMIENTO Y FACTOR PRIMO

Tres son las ideas fundamentales que dan lugar a los nuevos algoritmos rápidos para el cálculo de la TFD:

A) La descomposición de una transformada unidimensional en términos de una transformada multidimensional (algoritmo de *Good*).

B) La conversión de una TFD a una convolución cíclica.

C) La implementación de convoluciones cíclicas de órdenes pequeños con un mínimo de multiplicacones.

Retomando a la idea del algoritmo de *Good* (ver capítulo dos), dada una transformada cuyo orden N puede ser descompuesto en términos de más de dos factores primos relativos, es decir $N = N_1 \cdot N_2 \cdot \ldots \cdot N_L$, $(N_i, N_j) = 1, i \neq j$ entonces es posible hallar matrices de permutación $\mathbf{P}_N$ y $\mathbf{Q}_N$ tales que

$$\mathbf{P}_N \mathbf{W}_N \mathbf{Q}_N = \mathbf{W}_{N_1} \otimes \mathbf{W}_{N_2} \otimes \cdots \otimes \mathbf{W}_{N_L} \qquad (7.1)$$

Ahora bien, en los capítulos tres y cuatro se ha visto que si cada factor $N_i, i = 1,2,\ldots,L$, es de la forma $N_i = P^k$ con P primo mayor que 2, entonces cada matriz $\mathbf{W}_{N_i}$ puede ser reescrita en términos de la factorización **S C T**, es decir para cada matriz $\mathbf{W}_{N_i}$ existen matrices $\mathbf{S}_{N_i}$, $\mathbf{C}_{N_i}$ y $\mathbf{T}_{N_i}$ tales que

$$\mathbf{W}_{N_i} = \left(\mathbf{S}_{N_i} \mathbf{C}_{N_i} \mathbf{T}_{N_i} \right) \qquad (7.2)$$

Combinando esta última igualdad con el algoritmo de *Good* se tiene que

$$\mathbf{P}_N \mathbf{W}_N \mathbf{Q}_N = \left(\mathbf{S}_{N_1} \mathbf{C}_{N_1} \mathbf{T}_{N_1} \right) \otimes \left(\mathbf{S}_{N_2} \mathbf{C}_{N_2} \mathbf{T}_{N_2} \right) \otimes \cdots \otimes \left(\mathbf{S}_{N_L} \mathbf{C}_{N_L} \mathbf{T}_{N_L} \right) \qquad (7.3)$$

La ecuación anterior constituye el **algoritmo de anidamiento o de Good.** Como puede apreciarse el cálculo de la transformada, mediante este algoritmo, requiere que para cada factor, en el producto tensorial, se realicen sumas de entrada, productos y sumas de salida. Sin embargo, es posible concentrar todas las sumas de entrada, productos y sumas de salida si se toma en cuenta la siguiente propiedad del producto tensorial

$$\left(\mathbf{AB}\right)_{\otimes}\left(\mathbf{CD}\right) = \left(\mathbf{A}_{\otimes}\mathbf{C}\right)\left(\mathbf{B}_{\otimes}\mathbf{D}\right) \tag{7.4}$$

donde **A**,**B**,**C** y **D** son matrices de órdenes $M_1 x N_1$, $N_1 x N_2$, $M_3 x N_3$ y $N_3 x N_4$, respectivamente. Combinando esta propiedad con el algoritmo de anidamiento se llega a la siguiente ecuación

$$\mathbf{P}_N \mathbf{W}_N \mathbf{Q}_N = \left(\mathbf{S}_{N_1} \otimes \mathbf{S}_{N_2} \otimes \cdots \otimes \mathbf{S}_{N_L}\right)\left(\mathbf{C}_{N_1} \otimes \mathbf{C}_{N_2} \otimes \cdots \otimes \mathbf{C}_{N_L}\right)\left(\mathbf{T}_{N_1} \otimes \mathbf{T}_{N_2} \otimes \cdots \otimes \mathbf{T}_{N_L}\right) \tag{7.5}$$

la cual constituye el **algoritmo del factor primo o de Winograd.**

La diferencia fundamental entre el algoritmo del anidamiento y el del factor primo es que en este último todas las multiplicaciones se encuentran concentradas en la matriz diagonal

$$\left(\mathbf{C}_{N_1} \otimes \mathbf{C}_{N_2} \otimes \cdots \otimes \mathbf{C}_{N_L}\right) \tag{7.6}$$

y por tanto esta matriz puede ser precalculada, consiguiéndose con esto una mayor rapidez para el cálculo de la transformada. Sin embargo, existen dos desventajas de este algoritmo con respecto al de anidamiento, pues por una parte es necesario almacenar la diagonal de la matriz $\left(\mathbf{C}_{N_1} \otimes \mathbf{C}_{N_2} \otimes \cdots \otimes \mathbf{C}_{N_L}\right)$ lo cual representará un gasto extra de memoria y por otra parte, las multiplicaciones innecesarias en el algoritmo de anidamiento (multiplicaciones por 1 y por *i*, ver capítulo seis) deberán ser consideradas como necesarias en la matriz precalculada $\left(\mathbf{C}_{N_1} \otimes \mathbf{C}_{N_2} \otimes \cdots \otimes \mathbf{C}_{N_L}\right)$ y por consiguiente la diagonal de esta matriz puede llegar a ser muy grande.

En el apéndice de este trabajo se encuentra un programa en *Pascal*, el cual es una implementación del algoritmo de anidamiento. Este programa, aunque no es tan rápido como lo sería una implementación del algoritmo del factor primo, sí es aproximadamente un 30 % más rápido que un programa basado en el algoritmo de *CooleyTuckey* y por otro lado no requiere de tanta memoria para su ejecución, además a este programa se le ha incluído una rutina para el cálculo de la transformada de orden 13, lo cual permite calcular transformadas cuyos órdenes son de la forma

$$N = N_1 \cdot N_2 \cdot N_3 \cdot N_4 \cdot N_5 \tag{7.7}$$

con

$$N_1 \in \{1,2,4,8,16\}$$

$$N_2 \in \{1,3,9\}$$

$$N_3 \in \{1,5\}$$

$$N_4 \in \{1,7\}$$

$$N_5 \in \{1,13\} \tag{7.8}$$

De esta manera se cuenta con una mayor variedad en el orden de las transformadas, desde una transformada de orden 1, hasta una transformada de orden $16 \cdot 9 \cdot 5 \cdot 7 \cdot 13 = 65520$, exactamente 120 tipos diferentes de órdenes de transformadas, lo cual representa una mayor variedad de órdenes de transformadas que lo que se consigue con el algoritmo de *CooleyTuckey*.

8. CONCLUSIONES

Por lo general el desarrollo de un algoritmo más eficiente que otro, desde un punto de vista de rapidez, repercute de manera directa en una mayor complejidad del algoritmo y por consiguiente en un programa con un mayor código. Por tanto, al comparar el programa basado el algoritmo de *Cooley Tuckey* del capítulo uno y el programa basado en el algoritmo de anidamiento del apéndice, no deberá sorprender sus enormes diferencias en cuanto a complejidad y tamaño, pues como ya se ha visto la herramienta matemática empleada para los nuevos algoritmos de anidamiento y factor primo, es mucho mayor a la empleada en el algoritmo de *Cooley Tuckey*.

Ahora bien, se resumirá en tres puntos principales las ventajas de los algoritmos de anidamiento y factor primo sobre el algoritmo clásico de *Cooley Tuckey*

-Mayor rapidez
-Mayor variedad en el orden de las transformadas
-Mayor precisión

8.1. Mayor rapidez

Con respecto a la ganancia en rapidez de estos nuevos algoritmos sobre el algoritmo clásico de *Cooley Tuckey*, se han realizado corridas en una *PS* – 60 del programa basado en el algoritmo de anidamiento y el programa basado en el algoritmo de *Cooley Tuckey* y se ha observado que el primero es aproximadamente un 30 % más rápido que el segundo (ver tabla 8.1). Además se estima que una vez implementado el algoritmo del factor primo su tiempo de ejecución será aproximadamente un 40 % más rápido que en el caso del programa basado en el algoritmo de *Cooley Tuckey*.

Orden de la TFD	*Cooley Tuckey*	Algoritmo de anidamiento
256	0.66	
260		0.49
504		0.77
512	1.54	
1024	3.40	
1040		2.25
2048	7.47	
2184		5.43

Tabla 8.1 comparativa de los tiempos (en segundos) de ejecución para los algoritmos de Cooley Tuckey y anidamiento.

8.2. Mayor variedad en el orden de las transformadas

En cuanto a la variedad del orden de las transformadas puede decirse que tomando en cuenta el algoritmo óptimo para el cálculo de la transformada de orden 13, desarrollado en el presente trabajo, los nuevos algoritmos permiten calcular hasta 120 diferentes tipos de órdenes de transformadas, desde una transformada de orden 1 hasta una transformada de orden $16 \cdot 9 \cdot 5 \cdot 7 \cdot 13 = 65520$ (aproximadamente 2^{16}), pasando por todos aquellos órdenes N de transformadas que se obtienen como

$$N = N_1 \cdot N_2 \cdot N_3 \cdot N_4 \cdot N_5 \qquad (8.2.1)$$

con

$$N_1 \in \{1,2,4,8,16\}$$

$$N_2 \in \{1,3,9\}$$

$$N_3 \in \{1,5\}$$

$$N_4 \in \{1,7\}$$

$$N_5 \in \{1,13\} \qquad (8.2.2)$$

En cambio, ya que el algoritmo de *Cooley Tuckey* requiere que el orden de la transformada sea una potencia de 2 entonces, aunque no haya límite para el orden de la transformada, el espaciamiento entre el orden de una transformada y el siguiente aumenta a medida que se va eligiendo un orden mayor.

8.3. Mayor precisión

La mayor precisión de los nuevos algoritmos de anidamiento y factor primo sobre el algoritmo clásico de *Cooley Tuckey*, se debe a que estos nuevos algoritmos requieren menos operaciones de punto flotante, con lo cual se logra una menor pérdida de precisión, al calcular las transformadas, pues aunque los nuevos algoritmos requieren aproximadamente el mismo número de adiciones que el algoritmo de *Cooley Tuckey*, no sucede así con las multiplicaciones, en donde para transformadas de órdenes grandes se logra un ahorro de más de la mitad de las multiplicaciones.

Por ejemplo, para calcular una transformada de orden 512 empleando el algoritmo de *Cooley Tuckey* se requieren de

$$7172 = 4\left(\frac{512}{2} Log_2 \frac{512}{4} + 1\right) \quad \textit{multiplicaciones reales} \qquad (8.3.1)$$

Por otro lado, para calcular una transformada de orden 520 = 8·5·13 empleando el algoritmo de anidamiento es necesario realizar

$$\begin{aligned} 65 &= 5\cdot 13 \quad \textit{transformadas de orden } 8 \\ 104 &= 8\cdot 13 \quad \textit{transformadas de orden } 5 \\ 40 &= 8\cdot 5 \quad \textit{transformadas de orden } 13 \end{aligned} \tag{8.3.2}$$

y dado que una transformada de orden 8 requiere de 4 multiplicaciones reales (es decir, 2 semicomplejas, si se recuerda que las entradas de la matriz C son puramente reales o imaginarias), una de orden 5 requiere de 10 multiplicaciones reales y una de orden 13 requiere de 42 multiplicaciones reales, entonces el número total de multiplicaciones necesarias para calcular una transformada de orden 520 es

$$2980 = 65\cdot 4 + 104\cdot 10 + 40\cdot 42 \quad \textit{multiplicaciones reales} \tag{8.3.3}$$

lo cual implica un ahorro de aproximadamente un 60 % de las multiplicaciones sobre el algoritmo de *Cooley Tuckey*.

Como segundo ejemplo, considérese la función $F(t) = 5Sen\,(2t) + 2Cos\,(t) + 4 + i\left(7Cos\,(5t) + 2\right)$, para la cual se han calculado TFD's (tabla 8.2) de órdenes 16, 16 y 13 empleando los algoritmos de *Cooley Tuckey* y de anidamiento. Como puede apreciarse en la siguiente tabla, con el algoritmo de anidamiento se logra una mayor precisión.

	Cooley Tuckey DFT orden 16		**Anidamiento** DFT orden 16		**Anidamiento** DFT orden 13	
$\hat{Z}_k$	$Re(\hat{Z}_k)$	$Im(\hat{Z}_k)$	$Re(\hat{Z}_k)$	$Im(\hat{Z}_k)$	$Re(\hat{Z}_k)$	$Im(\hat{Z}_k)$
0	4.00000000000	2.00000000000	4.0000000000	2.0000000000	4.0000000000	2.0000000000
1	1.00000000000	-0.00000000000	1.0000000000	-0.0000000000	1.0000000000	0.0000000000
2	-0.00000000000	-2.50000000000	-0.0000000000	-2.5000000000	-0.0000000000	-2.5000000000
3	-0.00000000000	-0.00000000001	-0.0000000000	-0.0000000000	-0.0000000000	0.0000000000
4	0.00000000000	-0.00000000000	0.0000000000	-0.0000000000	-0.0000000000	-0.0000000000
5	-0.00000000000	3.50000000000	-0.0000000000	3.5000000000	-0.0000000000	3.5000000000
6	0.00000000000	-0.00000000001	0.0000000000	-0.0000000000	-0.0000000000	-0.0000000000
7	0.00000000000	-0.00000000000	0.0000000000	-0.0000000000	0.0000000000	-0.0000000000
8	0.00000000000	0.00000000001	0.0000000000	0.0000000000	0.0000000000	3.5000000000
9	-0.00000000001	-0.00000000001	-0.0000000000	-0.0000000000	0.0000000000	-0.0000000000
10	-0.00000000000	0.00000000000	0.0000000000	0.0000000000	0.0000000000	0.0000000000
11	0.00000000000	3.50000000000	0.0000000000	3.5000000000	0.0000000000	2.5000000000
12	0.00000000000	0.00000000000	0.0000000000	0.0000000000	1.0000000000	0.0000000000
13	-0.00000000000	-0.00000000001	0.0000000000	-0.0000000000		
14	-0.00000000000	2.50000000000	-0.0000000000	2.5000000000		
15	0.99999999999	-0.00000000001	1.0000000000	-0.0000000000		

Tabla 8.2 TFD's de la función $F(t) = 5Sen(2t) + 2Cos(t) + 4 + i\left(7Cos(5t) + 2\right)$ de órdenes 16, 16 y 13 empleando los algoritmos de Cooley Tuckey y anidamiento, donde $Z_j = F(Tj)$ con $T = 2\pi/N$, $j = 0,1,...,N-1$ y N = orden de la TFD.

BIBLIOGRAFIA

[1] **John D. Lipson** "Elements of algebra and algebraic computing" (1981).

[2] **Richard E. Blahut** "Fast algorithms for digital signal processing" (1984).

[3] **Douglas F. Elliott & K. Ramamohan Rao** "Fast transforms" (1982).

[4] **McClellan and Rader** "Number theory in digital signal processing" (1979).

[5] **E. Oran Brigham** "The fast Fourier transform" (1974).

[6] **Pablo Barrera S. & Javier Aurrecoechea H.** "Una versión matricial para los algoritmos rápidos en el cálculo de la Transformada de Fourier Discreta" (1984).

```
(*************************************************************************)
(*                                                                       *)
(*    IMPLEMENTACION DEL ALGORITMO DE GOOD PARA EL CALCULO DE LA TFD     *)
(*                                                                       *)
(*       Este programa obtiene la transformada de Fourier discreta       *)
(*    (TFD) de una señal cuyo orden sea de la forma:                     *)
(*                                                                       *)
(*                          N = N1*N2*N3*N4*N5                           *)
(*    donde:                                                             *)
(*       N1 = {1,2,4,8,16}                                               *)
(*       N2 = {1,3,9}                                                    *)
(*       N3 = {1,5}                                                      *)
(*       N4 = {1,7}                                                      *)
(*       N5 = {1,13}                                                     *)
(*                                                                       *)
(*       Esta versión corre adecuadamente en una computadora PC-XT       *)
(*    siempre y cuando el orden de la TFD no sea mayor a 5040. sin       *)
(*    embargo es posible obtener TFD's de orden mayor en equipos         *)
(*    de cómputo con mayor capacidad, simplemente modificando el         *)
(*    valor de la costante maxotf.                                       *)
(*                                                                       *)
(*       Programo:                                                       *)
(*                                                                       *)
(*         Mat. Gerardo Vega H.          Dpto. de Innovación y          *)
(*                                       Desarrollo en Sistemas,        *)
(*                                       DGSCA, UNAM.                   *)
(*                                                                       *)
(*************************************************************************)
Program ArteFFT;

  uses
    Dos;

  Const
  maxotfd = 5039;                    { maximo orden de la TFD = 16*9*7*5 }
  { Multiplicaciones para una TFD de orden 3 }
  c1f3   = -1.500000000000000000;
  c2f3   =  0.866025403784438646 77;

  { Multiplicaciones para una TFD de orden 5 }
  c1f5   = -1.250000000000000000;
  c2f5   =  0.559016994374947424 10;
  c3f5   = -0.951056516295153572 11;
  c4f5   = -1.538841768587626701 29;
  c5f5   =  0.363271264002680442 94;
```

```
  { Multiplicaciones para una TFD de orden 16 }
  c1f16  =  0.707106781186547524400;
  c2f16  =  0.707106781186547524400;
  c3f16  =  0.382683432365089771730;
  c4f16  =  1.306562965487637652786;
  c5f16  = -0.541196100146196984400;
  c6f16  = -0.707106781186547524400;
  c7f16  = -0.707106781186547524400;
  c8f16  = -0.923879532511286756130;
  c9f16  =  0.541196100146196984400;
  c10f16 = -1.306562964876376527850;

Type
  vector  = Array[0..maxotfd,1..2] Of Real;
  indices = Array[0..maxotfd]      Of Integer;

Var
  x : vector;                                  { señal de entrada y de salida. }
  t1r, t2r, t3r, t4r, t5r, t6r, t7r, t8r, t9r, t10r, { variables para las }
  t11r,t12r,t13r,t14r,t15r,t16r,t17r,t18r,t19r,t20r, { sumas de entrada.  }
  t21r,t22r,t23r,t24r,t25r,t26r,t27r,t28r,t29r,
  t1i, t2i, t3i, t4i, t5i, t6i, t7i, t8i, t9i, t10i,
  t11i,t12i,t13i,t14i,t15i,t16i,t17i,t18i,t19i,t20i,
  t21i,t22i,t23i,t24i,t25i,t26i,t27i,t28i,t29i,
  m0r,m1r, m2r, m3r, m4r, m5r, mór, m7r, m8r, m9r,   { variables para las }
  m10r,m11r,m12r,m13r,m14r,m15r,m16r,m17r,m18r,m19r, { multiplicaciones.  }
  m20r,m21r,
  m0i,m1i, m2i, m3i, m4i, m5i, m6i, m7i, m8i, m9i,
  m10i,m11i,m12i,m13i,m14i,m15i,m16i,m17i,m18i,m19i,
  m20i,m21i,
  s1r, s2r, s3r, s4r, s5r, s6r, s7r, s8r, s9r, s10r, { variables para las }
  s11r,s12r,s13r,s14r,s15r,s16r,s17r,s18r,s19r,s20r, { sumas de salida.   }
  s21r,s22r,s23r,s24r,s25r,s26r,s27r,s28r,s29r,s30r,
  s31r,s32r,s33r,s34r,s35r,s36r,s37r,
  s1i, s2i, s3i, s4i, s5i, s6i, s7i, s8i, s9i, s10i,
  s11i,s12i,s13i,s14i,s15i,s16i,s17i,s18i,s19i,s20i,
  s21i,s22i,s23i,s24i,s25i,s26i,s27i,s28i,s29i,s30i,
  s31i,s32i,s33i,s34i,s35i,s36i,s37i                : Real;
  i0, i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, { índices para los   }
  i12,i13,i14,i15,                                  { reordenamientos.   }
  N, N1, N2, N3, N4, N5,                            { orden de la TFD y  }
  NM1, NM2, NM3, NM4, NM5,                          { sus factores.      }
  desp, otfd, i, j, j0                              : Integer;
  pot                                               : LongInt;
  h1, m1, s1, hund1, h2, m2, s2, hund2              : Word;
```

```
(*****************************************************************************)
(*                                                                           *)
(*      Procedimiento que genera y reordena la señal x de orden  N.          *)
(*      El reordenamiento se realiza según una variante del teorema          *)
(*      chino del residuo.                                                   *)
(*                                                                           *)
(*****************************************************************************)
Procedure Gen_Vector(Var x:vector; N:Integer);

  Var
    index              : indices;        { arreglo para el reordenamiento }
                                         { del vector x                   }
    t, coef            : Real;
    i1,i2,i3,i4,i5,j,k : Integer;

  Begin
    j := 0;
    k := 0;
    For i5 := 1 To N5 Do                 { obtención del arreglo index    }
      Begin                              { según una variente de el       }
        For i4 := 1 To N4 Do             { teorema chino del residuo.     }
          Begin
            For i3 := 1 To N3 Do
              Begin
                For i2 := 1 To N2 Do
                  Begin
                    For i1 := 1 To N1 Do
                      Begin
                        If (k >= N) Then k := k Mod N;
                        index[k] := j;
                        j := j + 1;
                        k := k + NM1     { NM1 = N / N1                   }
                      End;
                    k := k + NM2         { NM2 = N / N2                   }
                  End;
                k := k + NM3             { NM3 = N / N3                   }
              End;
            k := k + NM4                 { NM4 = N / N4                   }
          End;
        k := k + NM5                     { NM5 = N / N5                   }
      End;
    coef := 2*Pi/N;
    For j := 0 To N - 1 Do               { genera y reordena la señal de  }
      Begin                              { entrada.                       }
        t := coef*j;
        x[index[j],1] := 5*Sin(2*t)+2*Cos(t)+4;
        x[index[j],2] := 7*Cos(5*t)+2
      End
  End;
```

```
(*****************************************************************************)
(*                                                                           *)
(*          Procedimiento que realiza N / 2 TFD's de orden 2                 *)
(*                                                                           *)
(*****************************************************************************)
Procedure Transformada_de_orden_2(Var x:vector; Var desp:Integer);
  Begin
    i0 := 0;
    For i := 1 To N Div 2 Do            { realiza N / 2 TFD's de orden 2   }
      Begin
        i1 := i0 + 1;
        t1r := x[i0,1];
        t1i := x[i0,2];
        x[i0,1] := t1r + x[i1,1];
        x[i0,2] := t1i + x[i1,2];
        x[i1,1] := t1r - x[i1,1];
        x[i1,2] := t1i - x[i1,2];
        i0 := i0 + 2
      End;
    desp := 2                           { desplasamiento para sig. TFD }
  End;

(*****************************************************************************)
(*                                                                           *)
(*          Procedimiento que realiza N / 3 TFD's de orden 3                 *)
(*                                                                           *)
(*****************************************************************************)
Procedure Transformada_de_orden_3(Var x:vector; Var desp,NM:Integer);
  Begin
    otfd := 3*desp;
    j0 := 0;
    For i := 1 To N Div otfd Do     { realiza N / 3 TFD's de orden 3       }
      Begin
        i0 := j0;
        For j := 1 To desp Do       { realiza desp TFD's de orden 3        }
          Begin
            i1 := i0 + desp;
            i2 := i1 + desp;
            t1r := x[i1,1] + x[i2,1];               { sumas de entrada }
            t1i := x[i1,2] + x[i2,2];
            m0r := t1r + x[i0,1];                   { multiplicaciones }
            m0i := t1i + x[i0,2];
            m1r := c1f3*t1r;
            m1i := c1f3*t1i;
            m2r := c2f3*(x[i1,2] - x[i2,2]);
            m2i := c2f3*(x[i2,1] - x[i1,1]);
            s1r := m0r + m1r;                       { sumas de salida  }
            s1i := m0i + m1i;
            x[i0,1] := m0r;
            x[i0,2] := m0i;
            x[i1,1] := s1r + m2r;
            x[i1,2] := s1i + m2i;
            x[i2,1] := s1r - m2r;
            x[i2,2] := s1i - m2i;
            i0 := i0 + 1
          End;
```

```
      j0 := j0 + otfd                    { sig. grupo de TFD's de Ord. 3 }
    End;
  desp := otfd;                          { desplasamiento para sig. TFD  }
  pot  := NM;
  NM   := (pot*pot) Mod N                { NM = ((N/N2)^Ø(N2)) Mod N     }
End;

(*************************************************************************)
(*                                                                       *)
(*          Procedimiento que realiza N / 4 TFD's de orden 4             *)
(*                                                                       *)
(*************************************************************************)
Procedure Transformada_de_orden_4(Var x:vector; Var desp,NM:Integer);
  Begin
    i0 := 0;
    For i := 1 To N Div 4 Do          { realiza N / 4 TFD's de orden 4   }
      Begin
        i1 := i0 + 1;
        i2 := i1 + 1;
        i3 := i2 + 1;
        t1r := x[i0,1] + x[i2,1];                    { sumas de entrada }
        t1i := x[i0,2] + x[i2,2];
        t2r := x[i1,1] + x[i3,1];
        t2i := x[i1,2] + x[i3,2];
        m2r := x[i0,1] - x[i2,1];                    { multiplicaciones }
        m2i := x[i0,2] - x[i2,2];
        m3r := x[i1,2] - x[i3,2];
        m3i := x[i3,1] - x[i1,1];
        x[i0,1] := t1r + t2r;                        { sumas de salida  }
        x[i0,2] := t1i + t2i;
        x[i1,1] := m2r + m3r;
        x[i1,2] := m2i + m3i;
        x[i2,1] := t1r - t2r;
        x[i2,2] := t1i - t2i;
        x[i3,1] := m2r - m3r;
        x[i3,2] := m2i - m3i;
        i0 := i0 + 4
      End;
    desp := 4;                          { desplasamiento para sig. TFD }
    pot  := NM;
    NM   := (pot*pot) Mod N             { NM = ((N/N1)^Ø(N1)) Mod N     }
  End;

(*************************************************************************)
(*                                                                       *)
(*          Procedimiento que realiza N / 5 TFD's de orden 5             *)
(*                                                                       *)
(*************************************************************************)
Procedure Transformada_de_orden_5(Var x:vector; Var desp,NM:Integer);
  Begin
    otfd := 5*desp;
    j0 := 0;
    For i := 1 To N Div otfd Do     { realiza N / 5 TFD's de orden 5      }
      Begin
        i0 := j0;
```

```
      For j := 1 To desp Do       { realiza desp TFD's de orden 5          }
        Begin
          i1 := i0 + desp;
          i2 := i1 + desp;
          i3 := i2 + desp;
          i4 := i3 + desp;
          t1r := x[i1,1] + x[i4,1];                  { sumas de entrada }
          t1i := x[i1,2] + x[i4,2];
          t2r := x[i2,1] + x[i3,1];
          t2i := x[i2,2] + x[i3,2];
          t3r := x[i1,1] - x[i4,1];
          t3i := x[i1,2] - x[i4,2];
          t4r := x[i3,1] - x[i2,1];
          t4i := x[i3,2] - x[i2,2];
          t5r := t1r + t2r;
          t5i := t1i + t2i;
          m0r := x[i0,1] + t5r;                      { multiplicaciones }
          m0i := x[i0,2] + t5i;
          m1r := c1f5*t5r;
          m1i := c1f5*t5i;
          m2r := c2f5*(t1r - t2r);
          m2i := c2f5*(t1i - t2i);
          m3r := -c3f5*(t3i + t4i);
          m3i :=  c3f5*(t3r + t4r);
          m4r := -c4f5*t4i;
          m4i :=  c4f5*t4r;
          m5r := -c5f5*t3i;
          m5i :=  c5f5*t3r;
          s1r := m0r + m1r;                          { sumas de salida  }
          s1i := m0i + m1i;
          s2r := s1r + m2r;
          s2i := s1i + m2i;
          s3r := m3r - m4r;
          s3i := m3i - m4i;
          s4r := s1r - m2r;
          s4i := s1i - m2i;
          s5r := m3r + m5r;
          s5i := m3i + m5i;
          x[i0,1] := m0r;
          x[i0,2] := m0i;
          x[i1,1] := s2r + s3r;
          x[i1,2] := s2i + s3i;
          x[i2,1] := s4r + s5r;
          x[i2,2] := s4i + s5i;
          x[i3,1] := s4r - s5r;
          x[i3,2] := s4i - s5i;
          x[i4,1] := s2r - s3r;
          x[i4,2] := s2i - s3i;
          i0 := i0 + 1
        End;
      j0 := j0 + otfd                    { sig. grupo de TFD's de Ord. 5 }
    End;
  desp := otfd;                          { desplasamiento para sig. TFD  }
  pot  := NM;
  pot  := (pot*pot) Mod N;
  NM   := (pot*pot) Mod N                { NM = ((N/N3)^0(N3)) Mod N      }
End;
```

```
            s1r := m0r + m1r;
            s1i := m0i + m1i;
            s2r := s1r + m2r + m3r;                    { sumas de salida  }
            s2i := s1i + m2i + m3i;
            s3r := s1r - m2r - m4r;
            s3i := s1i - m2i - m4i;
            s4r := s1r - m3r + m4r;
            s4i := s1i - m3i + m4i;
            s5r := m5r + m6r + m7r;
            s5i := m5i + m6i + m7i;
            s6r := m5r - m6r - m8r;
            s6i := m5i - m6i - m8i;
            s7r := m5r - m7r + m8r;
            s7i := m5i - m7i + m8i;
            x[i0,1] := m0r;
            x[i0,2] := m0i;
            x[i1,1] := s2r + s5r;
            x[i1,2] := s2i + s5i;
            x[i2,1] := s3r + s6r;
            x[i2,2] := s3i + s6i;
            x[i3,1] := s4r - s7r;
            x[i3,2] := s4i - s7i;
            x[i4,1] := s4r + s7r;
            x[i4,2] := s4i + s7i;
            x[i5,1] := s3r - s6r;
            x[i5,2] := s3i - s6i;
            x[i6,1] := s2r - s5r;
            x[i6,2] := s2i - s5i;
            i0 := i0 + 1
          End;
        j0 := j0 + otfd                    { sig. grupo de TFD's de Ord. 7 }
      End;
    desp := otfd;                          { desplasamiento para sig. TFD  }
    pot  := NM;
    pot  := (pot*pot) Mod N;
    pot  := (pot*NM)  Mod N;
    NM   := (pot*pot) Mod N                { NM = ((N/N4)^0(N4)) Mod N      }
  End;

(*****************************************************************************)
(*                                                                           *)
(*          Procedimiento que realiza N / 8 TFD's de orden 8                 *)
(*                                                                           *)
(*****************************************************************************)
Procedure Transformada_de_orden_8(Var x:vector; Var desp,NM:Integer);
  Begin
    i0 := 0;
    For i := 1 To N Div 8 Do           { realiza N / 8 TFD's de orden 8    }
      Begin
        i1 := i0 + 1;
        i2 := i1 + 1;
        i3 := i2 + 1;
        i4 := i3 + 1;
        i5 := i4 + 1;
        i6 := i5 + 1;
        i7 := i6 + 1;
```

```
    t1r := x[i0,1] + x[i4,1];                    { sumas de entrada }
    t1i := x[i0,2] + x[i4,2];
    t2r := x[i2,1] + x[i6,1];
    t2i := x[i2,2] + x[i6,2];
    t3r := x[i1,1] + x[i5,1];
    t3i := x[i1,2] + x[i5,2];
    t4r := x[i1,1] - x[i5,1];
    t4i := x[i1,2] - x[i5,2];
    t5r := x[i3,1] + x[i7,1];
    t5i := x[i3,2] + x[i7,2];
    t6r := x[i3,1] - x[i7,1];
    t6i := x[i3,2] - x[i7,2];
    t7r := t1r + t2r;
    t7i := t1i + t2i;
    t8r := t3r + t5r;
    t8i := t3i + t5i;
    m2r := t1r - t2r;
    m2i := t1i - t2i;
    m3r := x[i0,1] - x[i4,1];
    m3i := x[i0,2] - x[i4,2];
    m4r := c1f8*(t4r - t6r);                     { multiplicaciones }
    m4i := c1f8*(t4i - t6i);
    m5r := t3i - t5i;
    m5i := t5r - t3r;
    m6r := x[i2,2] - x[i6,2];
    m6i := x[i6,1] - x[i2,1];
    m7r := -c2f8*(t4i + t6i);
    m7i :=  c2f8*(t4r + t6r);
    s1r := m3r + m4r;                            { sumas de salida  }
    s1i := m3i + m4i;
    s2r := m3r - m4r;
    s2i := m3i - m4i;
    s3r := m6r + m7r;
    s3i := m6i + m7i;
    s4r := m6r - m7r;
    s4i := m6i - m7i;
    x[i0,1] := t7r + t8r;
    x[i0,2] := t7i + t8i;
    x[i1,1] := s1r + s3r;
    x[i1,2] := s1i + s3i;
    x[i2,1] := m2r + m5r;
    x[i2,2] := m2i + m5i;
    x[i3,1] := s2r - s4r;
    x[i3,2] := s2i - s4i;
    x[i4,1] := t7r - t8r;
    x[i4,2] := t7i - t8i;
    x[i5,1] := s2r + s4r;
    x[i5,2] := s2i + s4i;
    x[i6,1] := m2r - m5r;
    x[i6,2] := m2i - m5i;
    x[i7,1] := s1r - s3r;
    x[i7,2] := s1i - s3i;
    i0 := i0 + 8
  End;
```

```
    desp := 8;
    pot  := NM;
    pot  := (pot*pot) Mod N;
    NM   := (pot*pot) Mod N                    { NM = ((N/N1)^Ø(N1)) Mod N    }
  End;

(*****************************************************************************)
(*                                                                           *)
(*          Procedimiento que realiza N / 9 TFD's de orden 9                 *)
(*                                                                           *)
(*****************************************************************************)
Procedure Transformada_de_orden_9(Var x:vector; Var desp,NM:Integer);
  Begin
    otfd := 9*desp;
    j0 := 0;
    For i := 1 To N Div otfd Do     { realiza N / 9 TFD's de orden 9       }
      Begin
        i0 := j0;
        For j := 1 To desp Do       { realiza desp TFD's de orden 9        }
          Begin
            i1 := i0 + desp;
            i2 := i1 + desp;
            i3 := i2 + desp;
            i4 := i3 + desp;
            i5 := i4 + desp;
            i6 := i5 + desp;
            i7 := i6 + desp;
            i8 := i7 + desp;
            t1r  := x[i1,1] + x[i8,1];                  { sumas de entrada }
            t1i  := x[i1,2] + x[i8,2];
            t2r  := x[i2,1] + x[i7,1];
            t2i  := x[i2,2] + x[i7,2];
            t3r  := x[i3,1] + x[i6,1];
            t3i  := x[i3,2] + x[i6,2];
            t4r  := x[i4,1] + x[i5,1];
            t4i  := x[i4,2] + x[i5,2];
            t5r  := t1r + t2r + t4r;
            t5i  := t1i + t2i + t4i;
            t6r  := x[i1,1] - x[i8,1];
            t6i  := x[i1,2] - x[i8,2];
            t7r  := x[i7,1] - x[i2,1];
            t7i  := x[i7,2] - x[i2,2];
            t8r  := x[i3,1] - x[i6,1];
            t8i  := x[i3,2] - x[i6,2];
            t9r  := x[i4,1] - x[i5,1];
            t9i  := x[i4,2] - x[i5,2];
            t10r := t6r + t7r + t9r;
            t10i := t6i + t7i + t9i;
            m0r  := x[i0,1] + t3r + t5r;
            m0i  := x[i0,2] + t3i + t5i;
            m1r  := c1f9*t3r;                           { multiplicaciones }
            m1i  := c1f9*t3i;
            m2r  := c2f9*t5r;
            m2i  := c2f9*t5i;
            m3r  := c3f9*(t1r - t2r);
            m3i  := c3f9*(t1i - t2i);
            m4r  := c4f9*(t2r - t4r);
            m4i  := c4f9*(t2i - t4i);
```

```
(*****************************************************************************)
(*                                                                           *)
(*         Procedimiento que realiza N / 13 TFD's de orden 13                *)
(*                                                                           *)
(*****************************************************************************)
Procedure Transformada_de_orden_13(Var x:vector; Var desp,NM:Integer);
  Begin
    otfd := 13*desp;
    j0 := 0;
    For i := 1 To N Div otfd Do     { realiza N / 13 TFD's de orden 13     }
      Begin
        i0 := j0;
        For j := 1 To desp Do       { realiza desp TFD's de orden 13       }
          Begin
            i1  := i0  + desp;
            i2  := i1  + desp;
            i3  := i2  + desp;
            i4  := i3  + desp;
            i5  := i4  + desp;
            i6  := i5  + desp;
            i7  := i6  + desp;
            i8  := i7  + desp;
            i9  := i8  + desp;
            i10 := i9  + desp;
            i11 := i10 + desp;
            i12 := i11 + desp;
            t1r  := x[i1,1] + x[i9,1] + x[i3,1];       { sumas de entrada }
            t1i  := x[i1,2] + x[i9,2] + x[i3,2];
            t2r  := x[i7,1] + x[i11,1] + x[i8,1];
            t2i  := x[i7,2] + x[i11,2] + x[i8,2];
            t3r  := x[i10,1] + x[i12,1] + x[i4,1];
            t3i  := x[i10,2] + x[i12,2] + x[i4,2];
            t4r  := x[i5,1] + x[i6,1] + x[i2,1];
            t4i  := x[i5,2] + x[i6,2] + x[i2,2];
            t5r  := x[i1,1] + x[i12,1];
            t5i  := x[i1,2] + x[i12,2];
            t6r  := x[i7,1] + x[i6,1];
            t6i  := x[i7,2] + x[i6,2];
            t7r  := x[i10,1] + x[i3,1];
            t7i  := x[i10,2] + x[i3,2];
            t8r  := x[i5,1] + x[i8,1];
            t8i  := x[i5,2] + x[i8,2];
            t9r  := x[i9,1] + x[i4,1];
            t9i  := x[i9,2] + x[i4,2];
            t10r := x[i11,1] + x[i2,1];
            t10i := x[i11,2] + x[i2,2];
            t11r := x[i1,1] - x[i12,1];
            t11i := x[i1,2] - x[i12,2];
            t12r := x[i7,1] - x[i6,1];
            t12i := x[i7,2] - x[i6,2];
            t13r := x[i10,1] - x[i3,1];
            t13i := x[i10,2] - x[i3,2];
            t14r := x[i5,1] - x[i8,1];
            t14i := x[i5,2] - x[i8,2];
            t15r := x[i9,1] - x[i4,1];
            t15i := x[i9,2] - x[i4,2];
```

```
t16r := x[i11,1] - x[i2,1];
t16i := x[i11,2] - x[i2,2];
t17r := t1r + t2r + t3r + t4r;
t17i := t1i + t2i + t3i + t4i;
t18r := t7r + t10r;
t18i := t7i + t10i;
t19r := t5r + t8r - t18r;
t19i := t5i + t8i - t18i;
t20r := t6r + t9r - t18r;
t20i := t6i + t9i - t18i;
t21r := t1r - t3r;
t21i := t1i - t3i;
t22r := t2r - t4r;
t22i := t2i - t4i;
t23r := t10r - t7r;
t23i := t10i - t7i;
t24r := t5r - t8r + t23r;
t24i := t5i - t8i + t23i;
t25r := t6r - t9r - t23r;
t25i := t6i - t9i - t23i;
t26r := t11r - t15r;
t26i := t11i - t15i;
t27r := t12r - t16r;
t27i := t12i - t16i;
t28r := t13r + t15r;
t28i := t13i + t15i;
t29r := t14r + t16r;
t29i := t14i + t16i;
m0r  := x[i0,1] + t17r;
m0i  := x[i0,2] + t17i;
m1r  := c1f13*t17r;                          { multiplicaciones }
m1i  := c1f13*t17i;
m2r  := c2f13*(t1r+t3r-t2r-t4r);
m2i  := c2f13*(t1i+t3i-t2i-t4i);
m3r  := c3f13*t19r;
m3i  := c3f13*t19i;
m4r  := c4f13*t20r;
m4i  := c4f13*t20i;
m5r  := c5f13*(t20r-t19r);
m5i  := c5f13*(t20i-t19i);
m6r  := -c6f13*t21i;
m6i  :=  c6f13*t21r;
m7r  := -c7f13*t22i;
m7i  :=  c7f13*t22r;
m8r  := -c8f13*(t21i + t22i);
m8i  :=  c8f13*(t21r + t22r);
m9r  := c9f13*t24r;
m9i  := c9f13*t24i;
m10r := c10f13*t25r;
m10i := c10f13*t25i;
m11r := c11f13*(t24r + t25r);
m11i := c11f13*(t24i + t25i);
m12r := -c12f13*t26i;
m12i :=  c12f13*t26r;
m13r := -c13f13*t27i;
m13i :=  c13f13*t27r;
```

```
s21r := s19r + s19r;
s21i := s19i + s19i;
s22r := s19r + s20r;
s22i := s19i + s20i;
s23r := s18r + s21r;
s23i := s18i + s21i;
s24r := s19r - s18r;
s24i := s19i - s18i;
s25r := m6r - m7r;
s25i := m6i - m7i;
s25r := s25r + s25r;
s25i := s25i + s25i;
s26r := m8r - m6r - m7r;
s26i := m8i - m6i - m7i;
s26r := s26r + s26r;
s26i := s26i + s26i;
s27r := m12r + m13r - m14r;
s27i := m12i + m13i - m14i;
s28r := m15r - m17r;
s28i := m15i - m17i;
s29r := m14r + m15r;
s29i := m14i + m15i;
s30r := s27r - m16r;
s30i := s27i - m16i;
s31r := -s27r + s28r + m18r;
s31i := -s27i + s28i + m18i;
s32r := -m12r - m15r + m16r + m19r;
s32i := -m12i - m15i + m16i + m19i;
s33r := -s27r - s29r - s29r - m14r + m17r + m20r + m21r;
s33i := -s27i - s29i - s29i - m14i + m17i + m20i + m21i;
s30r := s30r + s30r;
s30i := s30i + s30i;
s31r := s31r + s31r;
s31i := s31i + s31i;
s32r := s32r + s32r;
s32i := s32i + s32i;
s33r := s33r + s33r;
s33i := s33i + s33i;
s34r := s30r + s30r;
s34i := s30i + s30i;
s35r := s31r + s31r;
s35i := s31i + s31i;
s36r := s32r + s32r;
s36i := s32i + s32i;
s37r := s33r + s33r;
s37i := s33i + s33i;
x[i0,1]  := m0r;
x[i0,2]  := m0i;
x[i1,1]  := s12r + s25r + s22r + s34r + s32r;
x[i1,2]  := s12i + s25i + s22i + s34i + s32i;
x[i2,1]  := s16r + s26r + s23r + s35r + s33r;
x[i2,2]  := s16i + s26i + s23i + s35i + s33i;
x[i4,1]  := s13r - s25r + s24r + s30r + s36r;
x[i4,2]  := s13i - s25i + s24i + s30i + s36i;
x[i8,1]  := s14r - s26r - s22r + s31r + s37r;
x[i8,2]  := s14i - s26i - s22i + s31i + s37i;
```

```
t1r  := x[i0,1] + x[i8,1];                    ( sumas de entrada )
t1i  := x[i0,2] + x[i8,2];
t2r  := x[i4,1] + x[i12,1];
t2i  := x[i4,2] + x[i12,2];
t3r  := x[i2,1] + x[i10,1];
t3i  := x[i2,2] + x[i10,2];
t4r  := x[i2,1] - x[i10,1];
t4i  := x[i2,2] - x[i10,2];
t5r  := x[i6,1] + x[i14,1];
t5i  := x[i6,2] + x[i14,2];
t6r  := x[i6,1] - x[i14,1];
t6i  := x[i6,2] - x[i14,2];
t7r  := x[i1,1] + x[i9,1];
t7i  := x[i1,2] + x[i9,2];
t8r  := x[i1,1] - x[i9,1];
t8i  := x[i1,2] - x[i9,2];
t9r  := x[i3,1] + x[i11,1];
t9i  := x[i3,2] + x[i11,2];
t10r := x[i3,1] - x[i11,1];
t10i := x[i3,2] - x[i11,2];
t11r := x[i5,1] + x[i13,1];
t11i := x[i5,2] + x[i13,2];
t12r := x[i5,1] - x[i13,1];
t12i := x[i5,2] - x[i13,2];
t13r := x[i7,1] + x[i15,1];
t13i := x[i7,2] + x[i15,2];
t14r := x[i7,1] - x[i15,1];
t14i := x[i7,2] - x[i15,2];
t15r := t1r + t2r;
t15i := t1i + t2i;
t16r := t3r + t5r;
t16i := t3i + t5i;
t17r := t15r + t16r;
t17i := t15i + t16i;
t18r := t7r + t11r;
t18i := t7i + t11i;
t19r := t7r - t11r;
t19i := t7i - t11i;
t20r := t9r + t13r;
t20i := t9i + t13i;
t21r := t9r - t13r;
t21i := t9i - t13i;
t22r := t18r + t20r;
t22i := t18i + t20i;
t23r := t8r + t14r;
t23i := t8i + t14i;
t24r := t8r - t14r;
t24i := t8i - t14i;
t25r := t10r + t12r;
t25i := t10i + t12i;
t26r := t12r - t10r;
t26i := t12i - t10i;
```

```
            x[i3,1]  := s17r + s25r - s23r - s30r + s32r;
            x[i3,2]  := s17i + s25i - s23i - s30i + s32i;
            x[i6,1]  := s15r + s26r - s24r - s31r + s33r;
            x[i6,2]  := s15i + s26i - s24i - s31i + s33i;
            x[i12,1] := s12r - s25r + s22r - s34r - s32r;
            x[i12,2] := s12i - s25i + s22i - s34i - s32i;
            x[i11,1] := s16r - s26r + s23r - s35r - s33r;
            x[i11,2] := s16i - s26i + s23i - s35i - s33i;
            x[i9,1]  := s13r + s25r + s24r - s30r - s36r;
            x[i9,2]  := s13i + s25i + s24i - s30i - s36i;
            x[i5,1]  := s14r + s26r - s22r - s31r - s37r;
            x[i5,2]  := s14i + s26i - s22i - s31i - s37i;
            x[i10,1] := s17r - s25r - s23r + s30r - s32r;
            x[i10,2] := s17i - s25i - s23i + s30i - s32i;
            x[i7,1]  := s15r - s26r - s24r + s31r - s33r;
            x[i7,2]  := s15i - s26i - s24i + s31i - s33i;
            i0 := i0 + 1
          End;
        j0 := j0 + otfd                  { sig. grupo de TFD's de Ord. 13 }
      End;
    desp := otfd;                        { desplasamiento para sig. TFD   }
    pot  := NM;
    pot  := (pot*pot) Mod N;
    pot  := (pot*NM)  Mod N;
    pot  := (pot*pot) Mod N;
    NM   := (pot*pot) Mod N              { NM = ((N/N5)!Ø(N5)) Mod N      }
  End;

(*****************************************************************************)
(*                                                                           *)
(*         Procedimiento que realiza N / 16 TFD's de orden 16                *)
(*                                                                           *)
(*****************************************************************************)
Procedure Transformada_de_orden_16(Var x:vector; Var desp,NM:Integer);
  Begin
    i0 := 0;
    For i := 1 To N Div 16 Do        { realiza N / 16 TFD's de orden 16   }
      Begin
        i1  := i0  + 1;
        i2  := i1  + 1;
        i3  := i2  + 1;
        i4  := i3  + 1;
        i5  := i4  + 1;
        i6  := i5  + 1;
        i7  := i6  + 1;
        i8  := i7  + 1;
        i9  := i8  + 1;
        i10 := i9  + 1;
        i11 := i10 + 1;
        i12 := i11 + 1;
        i13 := i12 + 1;
        i14 := i13 + 1;
        i15 := i14 + 1;
```

```
m2r  := t15r - t16r;                          { multiplicaciones }
m2i  := t15i - t16i;
m3r  := t1r - t2r;
m3i  := t1i - t2i;
m4r  := x[i0,1] - x[i8,1];
m4i  := x[i0,2] - x[i8,2];
m5r  := c1f16*(t19r - t21r);
m5i  := c1f16*(t19i - t21i);
m6r  := c2f16*(t4r - t6r);
m6i  := c2f16*(t4i - t6i);
m7r  := c3f16*(t24r + t26r);
m7i  := c3f16*(t24i + t26i);
m8r  := c4f16*t24r;
m8i  := c4f16*t24i;
m9r  := c5f16*t26r;
m9i  := c5f16*t26i;
m10r := t18i - t20i;
m10i := t20r - t18r;
m11r := t3i - t5i;
m11i := t5r - t3r;
m12r := x[i4,2] - x[i12,2];
m12i := x[i12,1] - x[i4,1];
m13r := -c6f16*(t19i + t21i);
m13i :=  c6f16*(t19r + t21r);
m14r := -c7f16*(t4i + t6i);
m14i :=  c7f16*(t4r + t6r);
m15r := -c8f16*(t23i + t25i);
m15i :=  c8f16*(t23r + t25r);
m16r := -c9f16*t23i;
m16i :=  c9f16*t23r;
m17r := -c10f16*t25i;
m17i :=  c10f16*t25r;
s1r  := m3r + m5r;                            { sumas de salida }
s1i  := m3i + m5i;
s2r  := m3r - m5r;
s2i  := m3i - m5i;
s3r  := m11r + m13r;
s3i  := m11i + m13i;
s4r  := m13r - m11r;
s4i  := m13i - m11i;
s5r  := m4r + m6r;
s5i  := m4i + m6i;
s6r  := m4r - m6r;
s6i  := m4i - m6i;
s7r  := m8r - m7r;
s7i  := m8i - m7i;
s8r  := m9r - m7r;
s8i  := m9i - m7i;
s9r  := s5r + s7r;
s9i  := s5i + s7i;
s10r := s5r - s7r;
s10i := s5i - s7i;
s11r := s6r + s8r;
s11i := s6i + s8i;
s12r := s6r - s8r;
s12i := s6i - s8i;
```

```
      s13r := m12r + m14r;
      s13i := m12i + m14i;
      s14r := m12r - m14r;
      s14i := m12i - m14i;
      s15r := m15r + m16r;
      s15i := m15i + m16i;
      s16r := m15r - m17r;
      s16i := m15i - m17i;
      s17r := s13r + s15r;
      s17i := s13i + s15i;
      s18r := s13r - s15r;
      s18i := s13i - s15i;
      s19r := s14r + s16r;
      s19i := s14i + s16i;
      s20r := s14r - s16r;
      s20i := s14i - s16i;
      x[i0,1]  := t17r + t22r;
      x[i0,2]  := t17i + t22i;
      x[i1,1]  := s9r + s17r;
      x[i1,2]  := s9i + s17i;
      x[i2,1]  := s1r + s3r;
      x[i2,2]  := s1i + s3i;
      x[i3,1]  := s12r - s20r;
      x[i3,2]  := s12i - s20i;
      x[i4,1]  := m2r + m10r;
      x[i4,2]  := m2i + m10i;
      x[i5,1]  := s11r + s19r;
      x[i5,2]  := s11i + s19i;
      x[i6,1]  := s2r + s4r;
      x[i6,2]  := s2i + s4i;
      x[i7,1]  := s10r - s18r;
      x[i7,2]  := s10i - s18i;
      x[i8,1]  := t17r - t22r;
      x[i8,2]  := t17i - t22i;
      x[i9,1]  := s10r + s18r;
      x[i9,2]  := s10i + s18i;
      x[i10,1] := s2r - s4r;
      x[i10,2] := s2i - s4i;
      x[i11,1] := s11r - s19r;
      x[i11,2] := s11i - s19i;
      x[i12,1] := m2r - m10r;
      x[i12,2] := m2i - m10i;
      x[i13,1] := s12r + s20r;
      x[i13,2] := s12i + s20i;
      x[i14,1] := s1r - s3r;
      x[i14,2] := s1i - s3i;
      x[i15,1] := s9r - s17r;
      x[i15,2] := s9i - s17i;
      i0 := i0 + 16
    End;
  desp := 16;                          { desplasamiento para sig. TFD }
  pot  := NM;
  pot  := (pot*pot) Mod N;
  pot  := (pot*pot) Mod N;
  NM   := (pot*pot) Mod N              { NM = ((N/N1)^Ø(N1)) Mod N    }
End;
```

```
(*****************************************************************************)
(*                                                                           *)
(*     Procedimiento que realiza el despliegue y el ordenamiento de          *)
(*     salida del vector  x. Este ordenamiento se realiza según  el          *)
(*     teorema chino del residuo.                                            *)
(*                                                                           *)
(*****************************************************************************)
Procedure Desp_Vector(Var x:vector; N:Integer);

  Var
    index              : indices;
    i1,i2,i3,i4,i5,j,k : LongInt;

  Begin
    j := 0;                                  { obtención del arreglo index }
    k := 0;                                  { según el teorema chino del  }
    For i5 := 1 To N5 Do                     { residuo                     }
      Begin
        For i4 := 1 To N4 Do
          Begin
            For i3 := 1 To N3 Do
              Begin
                For i2 := 1 To N2 Do
                  Begin
                    For i1 := 1 To N1 Do
                      Begin
                        If (k >= N) Then k := k Mod N;
                        index[k] := j;
                        j := j + 1;
                        k := k + NM1          { NM1 = ((N/N1)^Ø(N1)) Mod N }
                      End;
                    k := k + NM2              { NM2 = ((N/N2)^Ø(N2)) Mod N }
                  End;
                k := k + NM3                  { NM3 = ((N/N3)^Ø(N3)) Mod N }
              End;
            k := k + NM4                      { NM4 = ((N/N4)^Ø(N4)) Mod N }
          End;
        k := k + NM5                          { NM5 = ((N/N5)^Ø(N5)) Mod N }
      End;
    For i := 0 To N - 1 Do
      Writeln(x[index[i],1]/N:20:10, x[index[i],2]/N:20:10)
  End;
```

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

```
(*****************************************************************************)
(*                                                                           *)
(*                    Comienza programa principal                            *)
(*                                                                           *)
(*****************************************************************************)
Begin
  Writeln('Introduzca los factores N1, N2, N3, N4 y N5 que conforman el');
  Writeln('          orden de la transformada de Fourier:');
  Writeln(' N1=(1,2,4,8,16)  N2=(1,3,9)  N3=(1,5)  N4=(1,7) N5=(1,13)');
  Readln(N1, N2, N3, N4, N5);          { lectura de los factores de N       }
  If Not (N1 In [1,2,4,8,16]) Or       { ¿ estan correctos los factores ?   }
     Not (N2 In [1,3,9])      Or
     Not (N3 In [1,5])        Or
     Not (N4 In [1,7])        Or
     Not (N5 In [1,13]) Then
     Begin
       Writeln('Error en los factores, ejecución finalizada');
       Halt
     End;
  N := N1*N2*N3*N4*N5;                 { N orden de la TFD.                 }
  NM1 := N Div N1;                     { calculo de N/N1, N/N2, N/N3,       }
  NM2 := N Div N2;                     { N/N4 y N/N5 que serán empleados    }
  NM3 := N Div N3;                     { en el reordenamiento de la señal.  }
  NM4 := N Div N4;
  NM5 := N Div N5;
  Gen_Vector(x, N);                    { genera y reordena el vector x.     }
  desp := 1;                           { desplasamiento para la prim. TFD.  }
  If N1 <> 1 Then
  Case N1 Of
     2  : Transformada_de_orden_2 (x, desp);
     4  : Transformada_de_orden_4 (x, desp, NM1);
     8  : Transformada_de_orden_8 (x, desp, NM1);
     16 : Transformada_de_orden_16(x, desp, NM1)
  End;
  If N2 <> 1 Then
  Case N2 Of
     3  : Transformada_de_orden_3(x, desp, NM2);
     9  : Transformada_de_orden_9(x, desp, NM2)
  End;
  If N3 = 5  Then Transformada_de_orden_5 (x, desp, NM3);
  If N4 = 7  Then Transformada_de_orden_7 (x, desp, NM4);
  If N5 = 13 Then Transformada_de_orden_13(x, desp, NM5);
  Desp_Vector(x, N)                    { ordenamiento de salida y despliegue }
End.                                   { de vector x.                       }
```