El algoritmo del factor truncado en base cuatro*

** IVAN CASTRO CHADID
FABIO MOLINA FOCAZZIO
YOLIMA UMAÑA HERNANDEZ
DIONICIO VILLALBA ALDANA
ALVARO DUQUE HOYOS, S.J.

"El Teorema de Fourier no es solamente uno de los resultados más hermosos del análisis moderno, sino que puede decirse además que proporciona un instrumento indispensable en el tratamiento de casi todas las cuestiones de la física moderna, por recónditas que sean".

Lord Kelvin
[Sir William Thompson (1824-1907)]

RESUMEN

Se presenta un algoritmo que permite calcular la Transformada discreta de Fourier para $N = 4^m$ datos, utilizando tan solo $(m-1)4^m$ multiplicaciones complejas, siendo este valor inferior al del número de operaciones que requiere el conocido algoritmo de Cooley-Tukey ($m4^m$). Se realizan además comparaciones entre el tiempo de ejecución de la Transformada Discreta de Fourier, empleando los algoritmos de Cooley-Tukey y el del factor truncado, para algunos valores, observándose que a medida que el número de datos aumenta, disminuye a menos de la mitad el tiempo de ejecución del nuevo algoritmo. Se presenta además el diagrama de flujo y el programa en lenguaje BASIC.

* Trabajo presentado para el Coloquio Distrital de Matemáticas y Estadística. U. Distrital Francisco José de Caldas. Nov. 1989.

** Seminario de Investigación en Transformada Rápida de Fourier. Departamento de Matemáticas. Facultad de Ciencias. Pontificia Universidad Javeriana.

INTRODUCCION

Uno de los tópicos de mayor vigencia en el campo de la matemática aplicada es el relativo al estudio de los métodos rápidos para calcular la transformada discreta de Fourier, ya que ésta es protagonista de primer orden cuando se trata de realizar cálculos relativos a problemas de la más variada y mayor actualidad; baste señalar por ejemplo los siguientes:

a) En 1953 gracias a las técnicas de difracción de rayos X y a la transformada de Fourier, se descubrió la forma de la doble hélice del ADN.

b) Recientemente se ha logrado conocer la estructura de otras biomoléculas y de algunos virus.

c) La NASA la está empleando para mejorar la calidad y detalle de las imágenes de los objetos celestes tomadas en el espacio.

d) Es básica en física de plasmas, en física de semiconductores, en acústica de micro-ondas, en sismografía, oceanografía, en cartografía por medio del radar y en confección de imágenes en medicina.

e) Entre las muchas aplicaciones a la química tenemos el espectrómetro de transformación de Fourier utilizado en análisis químico.

f) Gracias a los trabajos de Ronald L. Bracewell (U. Standord 1956) se logró, utilizando la transformada de Fourier, una importante vía para la reconstrucción de imágenes conocida hoy con el nombre de Reconstrucción Tomográfica; algunos años después el mismo Bracewell creó el denominado "algoritmo modificado de proyección retrográfica" empleando ampliamente hoy en día en la tomografía por rayos X computarizada o **Escáner** TAC.

g) Gracias al empleo de la transformada de Fourier se logró la primera antena con resolución más fina que el ojo humano, la cual ha sido fundamental para la confección de los mapas solares.

En 1965 James W. Cooley (Centro de Investigaciones Thomas J. Watson de la IBM) y John W Tukey (Laboratorios Bell) desarrollaron un programa para calcular con un menor número de operaciones la transformada discreta de Fourier. Este programa fue denominado **Fast Fourier Transform** F.F.T. (Transformada rápida de Fourier) [6]. En 1971 A. Schönhage y V. Strassen demostraron matemáticamente que la complejidad en la multiplicación de dos números puede ser sólo ligeramente superior a la de la adición; estos hechos llevaron a corroborar en forma práctica un principio fundamental de la informática teórica y es que muchos de los algoritmos que nos son familiares, como por ejemplo, la regla de multiplicación que se enseña a los niños en los cursos de aritmética,

distan de ser los óptimos. Tampoco es cierto que si se trabajan los números en base 10 se tenga la mayor eficiencia operativa, una prueba de ello lo dan las más de 100.000.000 de cifras significativas del desarrollo decimal de π que se obtuvieron en 1987, al emplear la F.F.T. en un algoritmo debido a Jonathan M. Borwein y Peter B. Borwein. Desde la perspectiva de los primeros años de la década de 1970, generar tal cantidad de cifras hubiera sido una utopía ya que los factores fundamentales para realizar estos cálculos eran la fórmula y la velocidad de la máquina y poco intervenían la forma de operar y la base numérica sobre la cual estaban trabajando.

Ejemplos como el anterior, justificaron la necesidad de refinar cada vez más los procedimientos para calcular la transformada discreta de Fourier, empleando bases numéricas apropiadas que permitieran hacer no sólo menos operaciones sino que también, disminuyeran su complejidad.

El propósito de este artículo es difundir un algoritmo que, como lo veremos, es mucho más eficiente que el debido a Cooley-Tukey.

Sabemos que la Transformada Discreta de Fourier es un operador que transforma señales digitales periódicas con dominio temporal, en señales digitales periódicas con dominio frecuencial, de modo tal que el período de ambas señales, la original y la transformada, se mantiene invariable.

Si N es el período de la señales digitales con dominio temporal

$$x[0], x[1], \ldots, x[N-1]$$

debemos calcular las correspondientes señales digitales con dominio frecuencial

$$X[0], X[1], \ldots, X[N-1]$$

sabiendo que $\forall_n = 0, \ldots, N-1$

$$X[n] = \sum_{K=0}^{N-1} x[K]\, w^{nK}$$

con $w = e^{-\frac{2\pi i}{N}}$.

A diferencia del algoritmo de Cooley-Tukey, en éste vamos a suponer que $N = 4^m$. Sean

$$H = \{ \sum_{j=0}^{m-1} n_j 4^j \mid n_j = 0,1,2,3 \} \quad y \quad T = \{0,1,\ldots,N-1\}$$

Es claro que:

a) H tiene N elementos.

b) Si $h \in H$, entonces existen $n_0, n_1, \ldots, n_{m-1}$ enteros positivos menores que cuatro tales que

$$h = \sum_{j=0}^{m-1} n_j 4^j$$

como $n_i < 3 \forall j$, entonces

$$h \leq 3 \sum_{j=0}^{m-1} 4^j = 4^m - 1.$$

De lo anterior se desprende que $h \in T$. Por lo tanto $H \subseteq T$.

Como T también tiene N elementos, entonces $H = T$.

Por otra parte, el algoritmo de la división nos permite concluir que si

$$\sum_{j=0}^{m-1} n_j = \sum_{j=0}^{m-1} r_j 4^j, \quad \text{entonces, } n_j = r_j \ \forall j = 0, \ldots, m-1.$$

Lo anterior nos conduce a identificar los conjuntos.

$$\{x[n] \mid n = 0, 1, \ldots, N-1\} \quad y$$

$$\{x[n_{m-1}, n_{m-2}, \ldots, n_1, n_0] \quad n_j = 0,1,2,3 \quad \forall j = 0, \ldots, m-1\}$$

De donde, la suma (1) puede escribirse en la forma

$$X[n_{m-1}, n_{m-2}, \ldots, n_0] = \sum_{k_0=0}^{3} \sum_{k_1=0}^{3} \cdots \sum_{k=0}^{3} x[k_{m-1}, \ldots, k_1, k_0] \, w^{nk}$$

siendo

$$n = n_0 + n_1 4 + n_2 4^2 + n_3 4^3 + \ldots + n_{m-1} 4^{m-1} \quad y$$

$$K = K_{m-1} 4^{m-1} + K_{m-2} 4^{m-2} + K_{m-3} 4^{m-3} + K_{m-4} 4^{m-4} + \ldots + K_1 4 + K_0$$

En forma similar a como sucede con otros algoritmos, la clave de éste radica en la manera como se multiplican n y K, la cual se indica a continuación:

$$nK = 4^{m-1} n_0 K_{m-1} + 4^{m-2} n_0 K_{m-2} +$$

$$+ 4^{m-1} n_1 K_{m-2} + 4^{m-3} (n_0 + n_1 4) K_{m-3} +$$

$$+ 4^{m-1} n_2 K_{m-3} + 4^{m-4} (n_0 + n_1 4 + n_2 4^2) K_{m-4} +$$

$$+ 4^{m-1} n_{j-1} K_{m-j} + 4^{m-(j+1)} (n_0 + n_1 4 + \ldots + n_{j-1} 4^{j-1}) K_{m-(j+1)} +$$

$$\vdots$$

$$+ 4^{m-1} n_{m-2} K_1 + (n_0 + n_1 4 + \ldots + n_{m-2} 4^{m-2}) K_0 +$$

$$+ 4^{m-1} n_{m-1} K_0 + R$$

en donde R es un múltiplo de N.

Como $w^R = 1$ y $w^{4^{m-1}} = -i$, podemos afirmar que este algoritmo se reduce a resolver, para cada $n = 0, \ldots, N-1$, el siguiente sistema, en donde $x_0 = x$:

$$x_1[n_0, k_{m-2}, \ldots, k_0] = \sum_{k_{m-1}=0}^{3} x_0[k_{m-1}, k_{m-2}, \ldots, k_0] \, (-1)^{n_0 k_{m-1}} \, w^{4^{m-2} n_0 k_{m-2}}$$

$$x_2[n_0,n_1,k_{m-3},\ldots,k_0] = \Big(\sum_{k_{m-2}=0}^{3} x_1[n_0,k_{m-2},\ldots\ k_0]\,(-1)^{n_1 k_{m-2}}\Big)\,w^{4^{m-3}(n_0+n_1 4)k_{m-3}}$$

$$x_j[n_0,\ldots,n_{j-1},k_{m-(j+1)},\ldots,k_0] = \Big(\sum_{k_{m-j}=0}^{3} x_{j-1}[n_0,\ldots,n_{j-2},k_{m-j},\ldots,k_0]\,(-1)^{n_{j-1}k_{m-j}}\Big)\,w^{4^{m-(j+1)}\left(\sum_{i=0}^{j-1} n_i 4^i\right)k_{m-(j+1)}}$$

$$x_{m-1}[n_0,\ldots,n_{m-2},k_0] = \Big(\sum_{k_1=0}^{3} x_{m-2}[n_0,\ldots,n_{m-3},k_1,k_0]\,(-1)^{n_{m-2}k_1}\Big)\,w^{\left(\sum_{i=0}^{m-2} n_i 4^i\right)k_0}$$

$$x_m[n_0,\ldots,n_{m-2},n_{m-1}] = \sum_{k_0=0}^{3} x_{m-1}[n_0,\ldots,n_{m-2},k_0]\,(-1)^{n_{m-1}k_0}$$

$$x_m[n_{m-1},\ldots,n_1,n_0] = x_m[n_0,n_1,\ldots,n_{m-1}]$$

Examinemos la ecuación J-ésima.

Si llamamos $K = \sum_{h=0}^{m-j-1} K_h\, 4^h$, $L = n_{j-1}$, $T = \sum_{h=0}^{j-2} n_{j-2-h}\, 4^h$

podemos escribir esta ecuación de la siguiente manera:

$$x_j[T\,4^{m-j+1} + L\,4^{m-j} + K] = \Big(x_{j-1}[T\,4^{m-j+1} + K] + (-1)^L\, x_{j-1}[T\,4^{m-j+1} + 2.4^{m-j} + K]\Big) +$$

$$+^L \Big(x_{j-1}[T\,4^{m-j+1} + 3.4^{m-j} + K] + (-1)^L\, x_{j-1}[T\,4^{m-j+1} + 4^{m-j} + K])\Big)\,w^{S1}$$

siendo* $S = -\frac{\pi}{8}\,\mathrm{INT}(K/4^{m-j-1})\,(L + 4^{-j+1}\,\mathrm{INVC}(T))$.

Las variaciones posibles de los parámetros son:

$$K = 0,\ldots, 4^{m-j} - 1,\ T = 0,1,\ldots, 4^{j-1} - 1 \text{ y } L = 0,1,2,3.$$

Si organizamos adecuadamente esta ecuación, podemos darnos cuenta que para K,L y T fijos, vamos a realizar sólo una multiplicación compleja por cada J = 1 ,..., m-1, de donde el número total de multiplicaciones complejas que debemos efectuar es:

$$\boxed{(m-1)4^m}$$

Si comparamos este número con m 4^m que es el obtenido al emplear el algoritmo clásico de Cooley-Tukey para la Transformada rápida de Fourier, podemos observar que nos ahorramos 4^m multiplicaciones complejas, lo cual justifica la utilización de este procedimiento denominado **Algoritmo del Factor Truncado en Base Cuatro.**

COMPARACION ENTRE LOS TIEMPOS DE OPERACION DE LOS ALGORITMOS DE COOLEY-TUKEY Y DEL FACTOR TRUNCADO PARA ALGUNOS VALORES

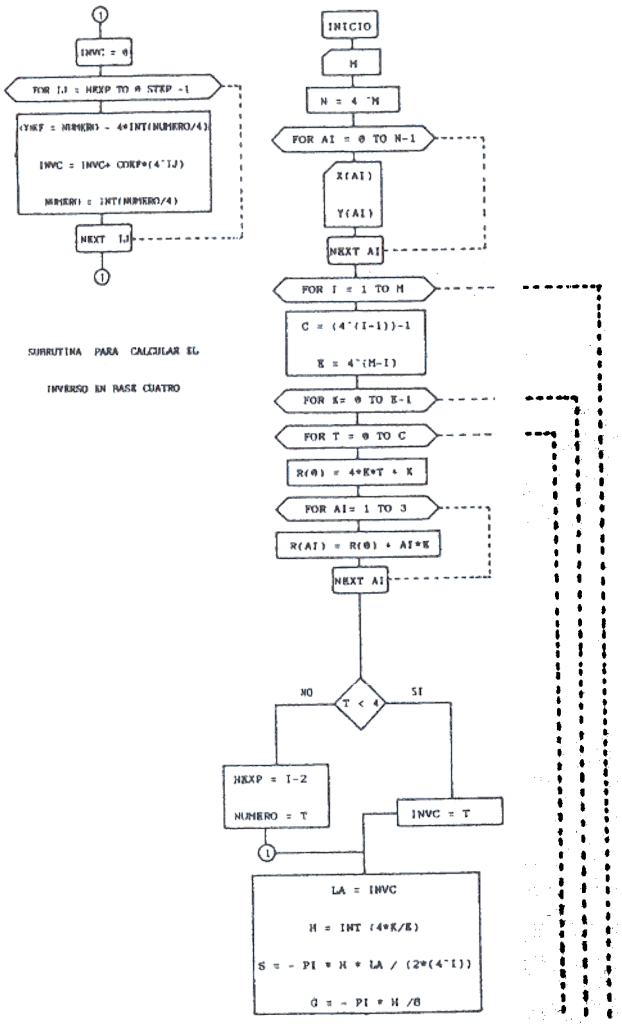
Al comparar el tiempo de operación entre estos dos algoritmos para los valores: $N = 4^m$, m = 1,2,3,4,5,6

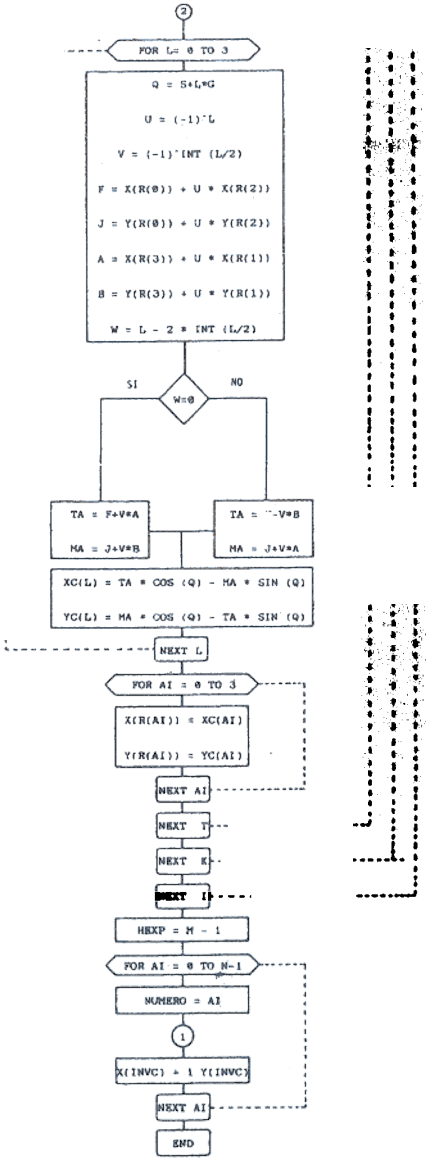
x[n] = (n + 1) +i(n+1) obtuvimos los siguientes resultados:

* Estamos notando la "parte de R", como INT(R) y el "inverso en base cuatro de R" como INV(R), en donde si

$$R = \sum_{\ell=0}^{t} a_\ell 4^\ell, \qquad \mathrm{INV}(R) = \sum_{\ell=0}^{t} a_\ell 4^{t-\ell}$$

DIAGRAMA DE FLUJO
PARA EL ALGORITMO DEL FACTOR
TRUNCADO EN BASE CUATRO





Número de Datos		Tiempo empleado en sgs.	
$N = 4^m$	No.	Col-Tuk	Fact. trunc.
4^1	4	1.54	0.28
4^2	16	3.24	1.32
4^3	64	15.15	8.07
4^4	256	86.95	43.33
4^5	1024	489	223.66
4^6	4096	2612.43	1094.44

Hay que tener en cuenta que en ambos casos el tiempo de operación no incluyó el ordenamiento final para la impresión.

Utilizamos un computador SAMSUNG SPC-6500 con coprocesador matemático 80287 y compilador GW-BASIC Versión 3.22. Creemos conveniente informar que también ensayamos con TURBO-BASIC Versión 1.0, la cual por ser tan rápida nos hacía perder precisión en la diferencia entre uno y otro algoritmo para los primeros datos; por tal razón la descartamos.

Las gráfica adjuntas nos ayudan a observar con mayor claridad las diferencias de tiempo entre uno y otro algoritmo.

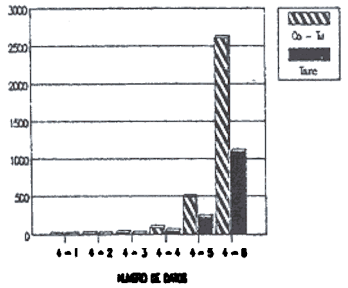
CONCLUSIONES

Existen algoritmos que permiten calcular la Transformada Discreta de Fourier con mayor rapidez y precisión que la F.F.T.; el más conocido es el denominado Transformada Rápida de Hartley, el cual sirve para calcular la Transformada Discreta de Hartley ya que las partes correspondientes a los valores pares e impares que toma dicha transformada son precisamente las partes real y menos imaginaria de la Transformada Discreta de Fourier [2].

El problema de calcular la Transformada Discreta de Fourier en la forma más rápida posible no está resuelto en su totalidad, como sí sucede en otras áreas de la matemática; (es el caso de la teoría de valuaciones euclidianas, en donde es posible construir con absoluta certeza la más rápida para cada dominio euclidiano en particular [5]), pero lo que hasta ahora se perfila en el horizonte es que este problema no debe prescindir de factores tales como: la forma en que realizamos las operaciones, la base sobre la cual trabajemos, la fórmula que utilicemos y por supuesto la velocidad de la máquina.

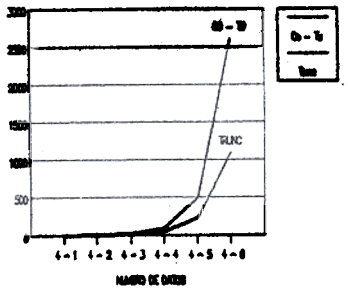
COMPARACION

C.T. – TRUNC



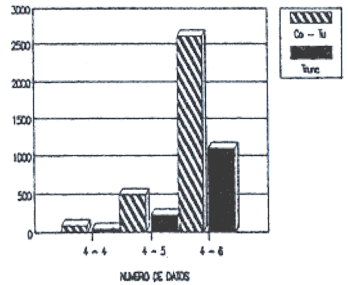
COMPARACION

C.T. – TRUNC



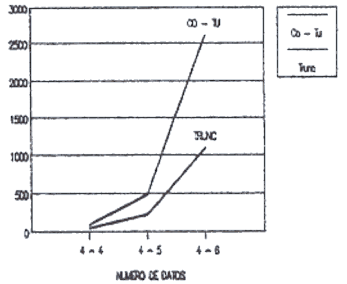
COMPARACION

C.T. – TRUNC



COMPARACION

C.T. – TRUNC



PROGRAMA EN LENGUAJE BASIC PARA CALCULAR

EL ALGORITMO DEL FACTOR TRUNCADO EN BASE 4

```
70  PI=3.14151168
80  INPUT "N=4^M   M=";M
90  N=4^M:DIM X(N),Y(N),H(3),HC(4),YC(4)
100 FOR NI=0 TO N-1
110    PRINT "COMPLEJO No. ";NI+1
120    INPUT X(NI)
130    INPUT Y(NI)
140 NEXT NI
150 FOR I=1 TO M
160    C=(4^(I-1))-1
170    B=4^(M-I)
180    FOR E=0 TO B-1
190       FOR T=0 TO -
200          K(0)=E+4*T+E
210          FOR AI=1 TO 3
220             K(AI)=K(0)+AI*E
230          NEXT AI
240          IF T<4 THEN GOTO 290
250          HKEP= .2
260          NUMERO=T
270          GOSUB 570
280          GOTO 300
290          INVC=T
300          LA=INVC
310          E=INT(4*E/B)
320          D=-PI*E*LA/(24*(4^I))
330          G=PI*E/B
340          FOR L=0 TO
350             A=E*L*D
360             D=(-1)^L
370             W=(-1)^INT(L/2.
380             F=X(K(0))+W*X(K(2))
390             J=Y(K(0))+W*Y(K(2))
400             A=X(K(1))+W*X(K(3))
410             B=Y(K(1))+W*Y(K(3))
420             W=L-2*INT(L/2)
430             IF W=0 THEN GOSUB 490
440             TA=F-VAR
450             HA=J-VAA
460             GOTO 490
470             TA=F+VAA
480             HA=J+VAR
490             XC(L)=TA*COS(G)-HA*SIN(G)
500             YC(L)=HA*COS(G)+TA*SIN(G)
510          NEXT L
520          FOR AI=0 TO 3
530             X(K(AI))=XC(AI)
540             Y(K(AI))=YC(AI)
550          NEXT AI
560       NEXT T
570    NEXT E
580 NEXT I
590 NKEP=N-1
600 FOR AI=0 TO N-1
610    NUMERO=AI
620    GOSUB 660
630    LPRINT "X (";AI;" ) = ";X(INVC);" + i ( ";Y(INVC);" )"
640 NEXT AI
650 END
660 REM
670 REM /* SUBRUTINA PARA CALCULAR EL INVERSO EN BASE CUATRO */
680 REM
690 INVC=0
700 FOR JJ=NKEP TO 1 STEP
710    COCF=NUMERO 4*(INT(NUMERO/4
720    INVC=INVC+COCF*(4^JJ)
730    NUMERO=INT(NUMERO/4)
740 NEXT JJ
750 RETURN
```


PROGRAMA EN LENGUAJE BASIC PARA CALCULAR LA TRANSFORMADA

RAPIDA DE FOURIER MEDIANTE EL ALGORITMO DE COOLEY-TUKEY

```
70 PI=3.14151168
80 DIM X(2,4000)
90 INPUT "N = 2 ^ NU /*/ NU = ";NU
100 N=2^NU
110 FOR I=0 TO N-1
120       PRINT "COMPLEJO No. ";I+1
130       INPUT X(1,I)
140       INPUT X(2,I)
150 NEXT I
160 L=1:I=0
170 N2=N/2
180 NU1=NU-1
190 IF L>NU THEN 440
200      I=1
210      H=INT(I/2^NU1)
220      I=H
230      GOSUB 640
240      P=IBR
250      A=COS(2*PI*P/N)
260      B=-SIN(2*PI*P/N)
270      T1=A*X(1,K+N2)-B*X(2,K+N2)
280      T2=A*X(2,K+N2)+B*X(1,K+N2)
290      X(1,K+N2)=X(1,I)-T1
300      X(2,K+N2)=X(2,I)-T2
310      X(1,I)=X(1,I)+T1
320      X(2,I)=X(2,I)+T2
330      I=I+1
340      IF I=N2 THEN 370
350           I=I+1
360           GOTO 210
370           K=K+N2
380           IF K<N-1 THEN 200
390      L=L+1
400      N2=N2/2
410      NU1=NU1-1
420      I=0
430 GOTO 190
440 I=I
450 GOSUB 640
460      B=IBR
470      IF B<= I THEN 540
480           T3=X(1,I)
490           T4=X(2,I)
500           X(1,I)=X(1,B)
510           X(2,I)=X(2,B)
520           X(1,B)=T3
530           X(2,B)=T4
540      IF I=N-1 THEN 570
550 I=I+1
560 GOTO 440
570 FOR J=0 TO N-1
580      LPRINT " X ( ";J+1;" ) = ";X(1,J);" + i (";X(2,J);" )"
590 NEXT J
600 END
610 REM
620 REM SUBRUTINA PARA CALCULAR EL INVERSO EN BASE DOS


650 II=1
660 J2=INT(I/2)
670 B=I-2*J2
680 IBR=IBR+B*(2^(NU-II))
690 I=J2
700 II=II+1
710 IF II>NU THEN RETURN
720 GOTO 660
```

EJEMPLOS ILUSTRATIVOS

Para estos ejemplos hemos escogido como datos de entrada los siguientes valores:

DATO No.	VALOR
1.	1 + i 1
2	2 + i 2
3	3 + i 3
4	4 + i 4
5	5 + i 5
6	6 + i 6
7	7 + i 7
8	8 + i 8
9	9 + i 9
10	1 + i 1
11	2 + i 2
12	3 + i 3
13	4 + i 4
14	5 + i 5
15	6 + i 6
16	7 + i 7
17	8 + i 8
18	9 + i 9
19	1 + i 1
20	2 + i 2
21	3 + i 3
22	4 + i 4
23	5 + i 5
24	6 + i 6
25	7 + i 7
26	8 + i 8
27	9 + i 9
28	1 + i 1
29	2 + i 2
30	3 + i 3
31	4 + i 4
32	5 + i 5
33	6 + i 6
34	7 + i 7
35	8 + i 8
36	9 + i 9
37	1 + i 1
38	2 + i 2
39	3 + i 3
40	4 + i 4
41	5 + i 5
42	6 + i 6
43	7 + i 7
44	8 + i 8
45	9 + i 9
46	1 + i 1
47	2 + i 2
48	3 + i 3
49	4 + i 4
50	5 + i 5
51	6 + i 6
52	7 + i 7
53	8 + i 8
54	9 + i 9
55	1 + i 1
56	2 + i 2
57	3 + i 3
58	4 + i 4
59	5 + i 5
60	6 + i 6
61	7 + i 7
62	8 + i 8
63	9 + i 9
64	1 + i 1

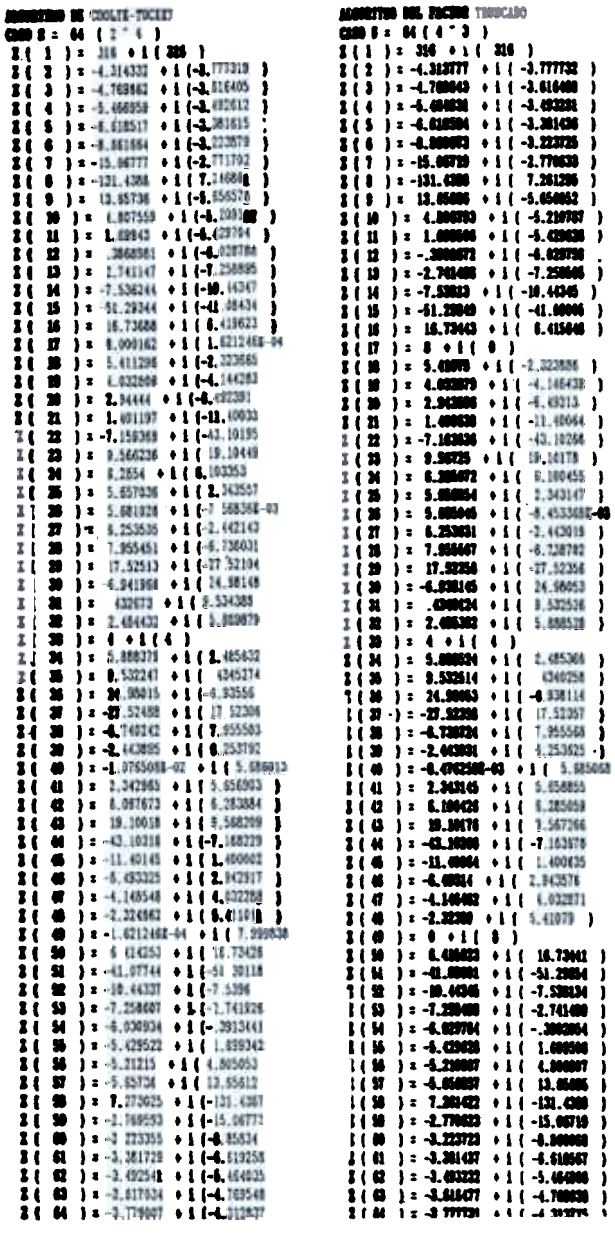
RESULTADOS

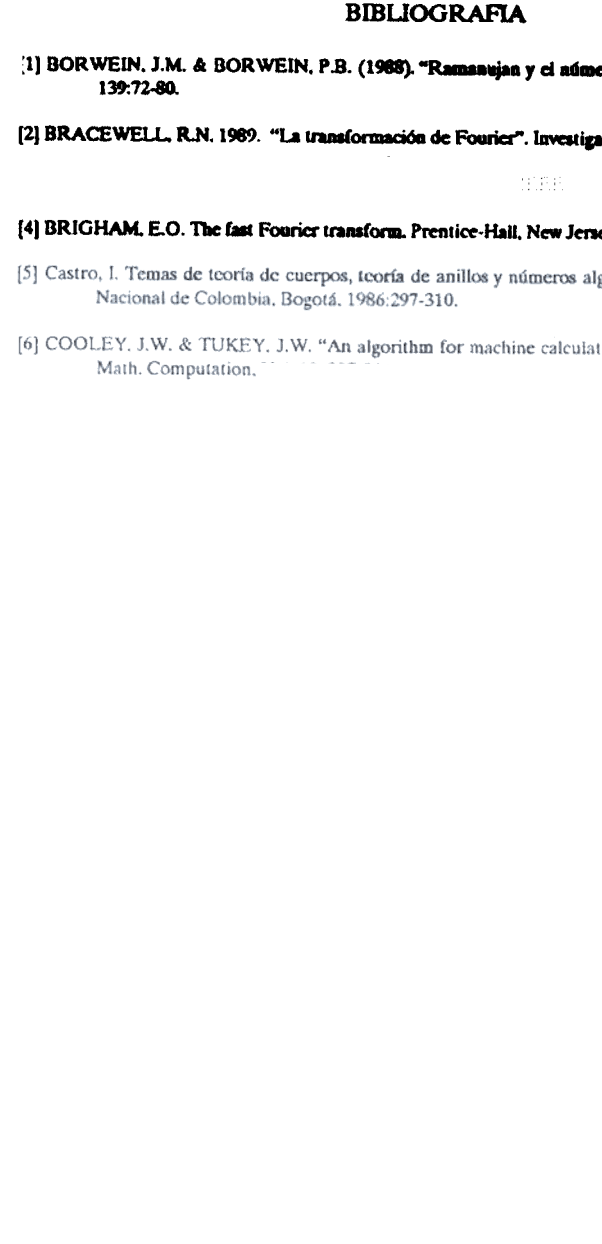
```
ALGORITMO DE COOLEY-TUCKEY
CASO N =  4 ( 2 ^ 2 )
 X(  1 ) =  10  + i (  10 )
 X(  2 ) = -4.000001  + i (-8.106232E-05  )
 X(  3 ) = -2  + i (-2 )
 X(  4 ) =  8.106232E-05  + i (-3.999919 )

ALGORITMO DEL FACTOR TRUNCADO
CASO N =  4 ( 4 ^ 1 )
X ( 1 ) =  10  + i (  10 )
X ( 2 ) = -4  + i (  0 )
X ( 3 ) = -2  + i ( -2 )
X ( 4 ) =  0  + i ( -4 )

ALGORITMO DE COOLEY-TUCKEY
CASO N =  16 ( 2 ^ 4 )
 X(  1 ) =  73  + i (  73 )
 X(  2 ) = -2.972365  + i (-13.02723  )
 X(  3 ) = -18.31472  + i (  20.31316  )
 X(  4 ) = -6.503204  + i (-9.456473  )
 X(  5 ) = -7.000324  + i (  8.999676  )
 X(  6 ) = -7.331784  + i (-8.668076  )
 X(  7 ) = -2.314072  + i (  4.312867  )
 X(  8 ) = -7.800982  + i (-8.198693  )
 X(  9 ) =  1  + i (  1 )
 X(  10  ) = -8.198956  + i (-7.80106  )
 X(  11  ) =  4.314071  + i (-2.313605  )
 X(  12  ) = -8.668278  + i (-7.331749  )
 X(  13  ) =  9.000324  + i (-6.999676  )
 X(  14  ) = -9.496734  + i (-6.503475  )
 X(  15  ) =  20.31472  + i (-18.31224  )
 X(  16  ) = -13.0277  + i (-2.973247  )

ALGORITMO DEL FACTOR TRUNCADO
CASO N =  16 ( 4 ^ 2 )
X ( 1 ) =  73  + i (  73 )
X ( 2 ) = -2.972659  + i ( -13.02734  )
X ( 3 ) = -18.31372  + i (  20.31371  )
X ( 4 ) = -6.503393  + i (  -9.496606  )
X ( 5 ) = -7  + i (  9  )
X ( 6 ) = -7.331822  + i (  -8.668179  )
X ( 7 ) = -2.313706  + i (  4.313707  )
X ( 8 ) = -7.801088  + i (  -8.198912  )
X ( 9 ) =  1  + i (  1 )
X ( 10  ) = -8.198913  + i (  -7.801086  )
X ( 11  ) =  4.313712  + i (  -2.31371  )
X ( 12  ) = -8.668179  + i (  -7.331821  )
X ( 13  ) =  9  + i (  -7  )
X ( 14  ) = -9.496806  + i (  -6.503395  )
X ( 15  ) =  20.31371  + i (  -18.3137  )
X ( 16  ) = -13.02734  + i (  -2.972663  )
```





BIBLIOGRAFIA

[1] BORWEIN, J.M. & BORWEIN, P.B. (1988). "Ramanujan y el número pi". Investigación y Ciencia. 139:72-80.

[2] BRACEWELL, R.N. 1989. "La transformación de Fourier". Investigación y Ciencia. 155:56-64.

[4] BRIGHAM, E.O. The fast Fourier transform. Prentice-Hall, New Jersey, 1974:184-197.

[5] Castro, I. Temas de teoría de cuerpos, teoría de anillos y números algebraicos. Tomo I. Universidad Nacional de Colombia, Bogotá. 1986:297-310.

[6] COOLEY, J.W. & TUKEY, J.W. "An algorithm for machine calculation of complex Fourier series' Math. Computation.