K O N I N K L IJ K N E D E R L A N D S
M E T E O R O L O G I S C H I N S T I T U U T

D e B i l t

WETENSCHAPPELIJK RAPPORT
W.R. 74-1

E.H.J. Vermaas

Fast Fourier transformatie

De Bilt, 1974

Publikatienummer: K.N.M.I. W.R. 74-1 (M.B.W.)

Inhoud

Summary.

This report deals with the fast Fourier transform (FFT), an efficient technique for evaluating the finite discrete Fourier transform (DFT). Principal advantages of the FFT-algorithm are a reduction of the number of computations and a better accuracy, especially for long records.

After an enumeration of definitions and theorems related to the DFT, an explanation of the FFT-algorithm is given. The DFT transforms complex functions. Usually one is interested in transforming real functions. For this purpose we give two algorithms, which make use of the DFT for complex functions.

This report is also a manual for the use of the mentioned algorithms translated in ALGOL-60, which are available at the KNMI Computing Division. The procedures orginate from the Collected Algorithms from C.A.C.M., exoept the procedures REALTWIN and COEFTWIN.

The relation between the continuous Fourier transform and the DFT is treated and an example is given of the use of the DFT for the evaluation of a Fourier integral.

As examples of algorithms which make use of FFT-procedures, we describe two ALGOL-60 procedures, named COVSPEK and FOUSPEK. Both procedures compute auto- and crossspectra of stationary time series. COVSPEK first compute the auto- and crosscovariancefunctions with FFT-techniques. After that we get the spectra by a DFT of the covariancefunctions. FOUSPEK compute the spectra by averaging the periodograms of overlapping sections of the time series. The computational time required for the procedure COVSPEK is about two times the time required for the procedure FOUSPEK for all record lenghts. This contradicts the statement of Edge and Liu (4) that for long records the FOUSPEK-method is the only feasible approach.

1. Inleiding.

De "fast _ourier transformatie (FFT)" is een efficiente methode voor het berekenen van de eindige discrete Fourier transformatie. Deze methode werd in 1965 herontdekt door Cooley en Tukey (1), na eerder te zijn gevonden door Runge en König (3) in 1924. Het berekenen van de discrete Fourier transformatie van N complexe getallen met behulp van de definiërende formule kost N^2 bewerkingen (1 bewerking = 1 complexe vermenigvuldiging + 1 complexe optelling). Met behulp van het FFT-algorithme wordt dit aantal teruggebracht tot $N(p_1 + p_2 + ... + p_k)$ waarbij $N = p_1 \times p_2 \times .... \times p_k$ ($p_i \in N$). We zien dus direkt dat het FFT-algorithme geen zin heeft wanneer N een priemgetal is. Hoe groter het aantal faktoren in de ontbinding van N is, hoe groter de versnellingsfaktor zal zijn. Een voorbeeld: voor $N=486=3^5 \times 2$ worden de aantallen voor beide methodes resp. 236196 en 8262, de versnellingsfaktor 28.

Het voornaamste doel van dit verslag is de toekomstige gebruiker van de op het K.N.M.I. aanwezige FFT-procedures in Algol-60 wegwijs te maken met het gebruik en de mogelijkheden ervan. Om dit doel te bereiken worden eerst een aantal eigenschappen van de eindige discrete Fourier transformatie behandeld. Daarna wordt de werking van het FFT-algorithme toegelicht, terwijl ook de doorwerking van de afrondfouten die de computer maakt bekeken zal worden. De reeds eerder vermelde Algol-60 procedures zijn afkomstig uit de Collected Algorithms from C.A.C.M. De maker is R.C. Singleton (14). De aanroepen van deze procedures zullen behandeld worden.

Vaak zal men niet geinteresseerd zijn in de Fourier transformatie van een aantal complexe getallen, maar van een aantal reële getallen. De methode om dan het imaginaire gedeelte nul te maken kost overbodige rekentijd en geheugenruimte. Er zullen daarom enkele algorithmes gegeven worden om reële Fourier transformaties efficient te berekenen.

Als voorbeelden van algorithmes die van FFT-procedures gebruik maken worden de procedures COVSPEK en FOUSPEK besproken. Deze procedures berekenen beide spektra van zwak stationaire tijdreeksen. COVSPEK doet dat door eerst de covariantiefunctie te berekenen (op een handige manier m.b.v. de FFT), waarna het spektrum berekend wordt door een Fourier transformatie van die covariantiefunctie. FOUSPEK echter berekent het spektrum door direkt een Fourier transformatie uit te voeren op overlappende segmenten van de tijdreeks.

Het blijkt dat COVSPEK slechts ongeveer twee maal zo lang rekent als FOUSPEK. Hieruit blijkt dat de COVSPEK-methode ook voor lange tijdreeksen efficient is uit te voeren, dit in tegenstelling tot wat bijvoorbeeld Edge en Liu (4) in hun artikel stellen.

Door de behandeling van deze twee procedures geeft dit verslag ook een tussentijds overzicht van de vorderingen die worden gemaakt bij het ontwerpen van rekenprogramma's voor spektraalanalyse ten behoeve van de onderafdeling Fysische Meteorologie.

---

2. De eindige discrete fourier transformatie.

We beschouwen functies van de gehele getallen modulo N ($\mathbb{Z}$ mod N) naar de complexe getallen ($\mathbb{C}$). Deze functies noteren we met een kleine latijnse letter uit het begin van het alfabet, behoudens enkele bijzondere functies. Met $\mathbb{R}$ geven we de verzameling der reëele getallen aan.

Nu eerst enkele definities:

(2.1) e_N $\qquad e_N(n) = e^{2\pi in/N} \qquad (i^2 = -1)$

De transformatie die nu gedefinieerd wordt, heet in de literatuur meestal de eindige inverse discrete Fourier transformatie.

(2.2) $\hat{a}$ $\qquad \hat{a}(\hat{n}) = \sum_{n=0}^{N-1} a(n)e_N(n\hat{n}) \qquad n,\hat{n} \in \mathbb{Z} \text{ mod } N$

$\hat{a}$ zullen we ook noteren als $IDFT_N(a)$.

De $IDFT_N$ is dus een afbeelding van de verzameling van bovengenoemde functies ($\mathbb{Z}$ mod N$\rightarrow \mathbb{C}$) op zichzelf.

(2.3) $\bar{a}$ $\qquad \bar{a}(n) = \overline{a(n)}$ (complex geconjugeerde)

(2.4) $\tilde{a}$ $\qquad \tilde{a}(n) = \overline{a(-n)}$ (involutie)

N.B. aangezien $n \in \mathbb{Z}$ mod N geldt: $a(-n) = a(N-n)$

(2.5) a_k $\qquad a_k(n) = a(n-k)$

(2.6) a.b $\qquad (a.b)\ (n) = a(n).b(n)$

We schrijven ook wel ab i.p.v. a.b

(2.7) $a * b$ $\qquad (a * b)\ (n) = \sum_{i=0}^{N-1} a(n-i)b(i)$ (convolutie)

(2.8) $\langle a,b \rangle$ $\qquad \langle a,b \rangle = \sum_{n=0}^{N-1} a(n).b(n)$

(2.9) $||a||_E$ $\qquad ||a||_E = (\langle a,a \rangle)^{\frac{1}{2}}$ (euclidische norm)

(2.10) δ_k $\qquad \delta_k(n) = 1$ indien $n = k$

$\qquad\qquad\qquad = 0$ indien $n \neq k$

(2.11) Δa $\qquad (\Delta a)\ (n) = a(n)-a(n-1)$

Bij het ontwerpen van rekenprocedures die van de $IDFT_N$ gebruik maken, moet men goed op de hoogte zijn van de diverse eigenschappen van de $IDFT_N$. Hiertoe volgen nu een aantal stellingen die gemakkelijk af te leiden zijn.

(2.12) De $IDFT_N$ is lineair, d.w.z.:

$$\widehat{a+b} = \hat{a} + \hat{b}$$

$$\widehat{\alpha a} = \alpha\, \hat{a} \qquad (\alpha \in \mathbb{C})$$

(2.13) $$\hat{\tilde{a}} = \bar{\hat{a}} \qquad \hat{\bar{a}} = \tilde{\hat{a}} \qquad \hat{\tilde{\bar{a}}} = \tilde{\bar{\hat{a}}}$$

een gevolg van de tweede betrekking van (2.13) is bijvoorbeeld dat als a reëel ($a(n) \in \mathbb{R}$), dus $a = \bar{a}$, de getransformeerde van a invariant is onder de involutie, dus $\hat{a} = \tilde{\hat{a}}$.

(2.14) $$a(n) = \frac{1}{N} \sum_{\hat{n}=0}^{N-1} \hat{a}(\hat{n})\, e_N(-n\hat{n})$$

Hiermee is ook de inverse van de transformatie (2.2) bepaald. In de literatuur wordt deze meestal de eindige discrete Fourier transformatie genoemd. We noteren de getransformeerde van a volgens (2.14) als $DFT_N(a)$ of $\check{a}$.

(2.15) $$DFT_N(a) = \frac{1}{N}\, \overline{IDFT_N(\bar{a})}$$

Hieruit zien we dat met behulp van een rekenprocedure voor de $IDFT_N$ ook de DFT_N berekend kan worden.

(2.16) $$\check{\check{a}} = N\, \tilde{a} \qquad \check{\tilde{a}} = \frac{1}{N}\, \tilde{\check{a}}$$

(2.17) $a \ast b = b \ast a$ de convolutie is commutatief

(2.18) $$\widetilde{a \ast b} = \tilde{a} \ast \tilde{b} \qquad \overline{a \ast b} = \bar{a} \ast \bar{b}$$

(2.19) $\hat{\delta}_k(j) = e_N(kj)$

(2.20) $a_k = a * \delta_k$

(2.21) $\hat{a}_k(j) = \hat{a}(j).e_N(kj)$

(2.22) $a(0) = \sum_{n=0}^{N-1} \hat{a}(n) \qquad \hat{a}(0) = \frac{1}{N} \sum_{n=0}^{N-1} a(n)$

(2.23) $\widehat{a * b} = \hat{a} \,.\, \hat{b} \qquad \widehat{a.b} = \frac{1}{N} (\hat{a} * \hat{b})$

Dit is een belangrijke stelling die een weg opent naar een efficientere berekening van convoluties, covariantiefuncties etc.

(2.24) $\langle a,b \rangle = \frac{1}{N} \langle \hat{a}, \hat{b} \rangle$

(2.25) $\sum_{n=0}^{N-1} |a(n)|^2 = \frac{1}{N} \sum_{n=0}^{N-1} |\hat{a}(n)|^2$ (Parseval)

(2.26) $\widehat{\Delta a}(j) = \hat{a}(j).(1-e_N(j))$

Notatie: soms zullen we waar mogelijk de argumenten weglaten. Stelling (2.26) wordt dan b.v. $\widehat{\Delta a} = \hat{a}(1-e_N)$

(2.27) $e_N(kN) = 1 \qquad (k \in \mathbb{Z}) \qquad e_N(p+q) = e_N(p).e_N(q)$

$e_{pq}(kp) = e_q(k) \qquad (k \in \mathbb{Z}) \qquad e_2(n) = (-1)^n$

We kunnen ook een meerdimensionale discrete Fourier transformatie definieren, b.v. voor dimensie = 2:

(2.28) $\hat{a}(\hat{k},\hat{l}) = \sum_{k=0}^{P-1} \sum_{l=0}^{Q-1} a(k,l)\, e_p(k\hat{k})\, e_Q(l\hat{l}) \qquad \begin{matrix} k,\hat{k} \in \mathbb{Z} \bmod P \\ l,\hat{l} \in \mathbb{Z} \bmod Q \end{matrix}$

Voor deze transformatie zijn vele stellingen analoog aan die voor het één-dimensionale geval:

Definieer b.v. $\tilde{a}(k,l) = \overline{a(-k,-l)}$ en $\bar{a}(k,l) = \overline{a(k,l)}$ dan geldt stelling (2.13) ook.

---

3. De fast Fourier transformatie (FFT).

We zullen een stelling afleiden die een weg opent naar een efficiente berekening van de $IDFT_N$.

Laat $N = p \times q$, dan kunnen we $n, \hat{n} \in \mathbb{Z} \bmod N$ als volgt schrijven:

(3.1) $$n = t + sq$$

(3.2) $$\hat{n} = \hat{s} + \hat{t}p \qquad s, \hat{s} \in \mathbb{Z} \bmod p \qquad t, \hat{t} \in \mathbb{Z} \bmod q$$

Definitie (2.2) van de $IDFT_N$:

$$\hat{a}(\hat{n}) = \sum_{n=0}^{N-1} a(n) e_N(n\hat{n})$$

Met gebruikmaking van (3.1) en (3.2) kunnen we dit ook schrijven als:

$$\hat{a}(\hat{s}+\hat{t}p) = \sum_{t=0}^{q-1} \sum_{s=0}^{p-1} a(t+sq)\, e_N((t+sq)(\hat{s}+\hat{t}p))$$

$$= \sum_{t=0}^{q-1} e_N(t(\hat{s}+\hat{t}p)) \sum_{s=0}^{p-1} a(t+sq)\, e_N(s\hat{s}q)$$

(3.3) $$= \sum_{t=0}^{q-1} e_N(t(\hat{s}+\hat{t}p)) \sum_{s=0}^{p-1} a(t+sq)\, e_p(s\hat{s})$$

definieer $a_t^q(s) = a(t+sq)$, dan:

(3.4) $$\hat{a}(\hat{s}+\hat{t}p) = \sum_{t=0}^{q-1} \left[e_N(t\hat{s}).IDFT_p(a_t^q)(\hat{s})\right].e_q(t\hat{t})$$

We zien dus dat de $IDFT_N$ uitgevoerd kan worden door:

I. Het uitvoeren van de $IDFT_p$ op de functies a_t^q voor $t=0,1,\ldots,q-1$

II. Het vermenigvuldigen van de uitkomsten van I. met de faktor $e_N(t\hat{s})$

III. Het uitvoeren van de $IDFT_q$ op het omhaakte gedeelte in 3.4, dat als functie van t beschouwd wordt, voor $\hat{s}=0,1,\ldots,p-1$

Het uitvoeren van II. behoeft geen extra berekeningen te vergen indien men bij het uitvoeren van de $IDFT_q$ in III. de faktoren $e_q(t\hat{t})$ vervangt door $e_N(t(\hat{s}+\hat{t}p))$.

Wanneer men één complexe vermenigvuldiging en één complexe optelling één bewerking noemt, dan kost de berekening van de $IDFT_N$ m.b.v. de definiërende formule(2.2) N^2 bewerkingen. Met bovenstaande methode

wordt dit aantal teruggebracht tot qp^2(voor I.) + pq^2(voor III.) = $N(p+q)$ bewerkingen.

Laat N te ontbinden zijn als $p_1\ p_2 \cdot \ldots \cdot p_k (p_i \in \mathbb{N})$, dan is het door herhaald toepassen van de stelling mogelijk het aantal bewerkingen terug te brengen tot $N(p_1 p_2 \cdot \ldots \cdot p_k)$.

We zullen dit herhaald toepassen demonstreren aan een eenvoudig geval, nl. N=8. Er blijken twee manieren mogelijk te zijn: één waarbij q iedere keer 2 is en één waarbij p iedere keer 2 is. De eerste methode wordt in de literatuur "decimation in time" genoemd, de tweede "decimation in frequence". Zie bijv. Cochran et.al.(6).

decimation in time

voor q=2 wordt (3.3):

$$\hat{a}(\hat{s}+\hat{t}p) = \sum_{t=0}^{1} e_N(t(\hat{s}+\hat{t}p)) \sum_{s=0}^{p-1} a(t+2s)e_p(s\hat{s}) \qquad \begin{matrix} s,\hat{s} \in \mathbb{Z} \bmod p \\ t,\hat{t} \in \mathbb{Z} \bmod 2 \end{matrix}$$

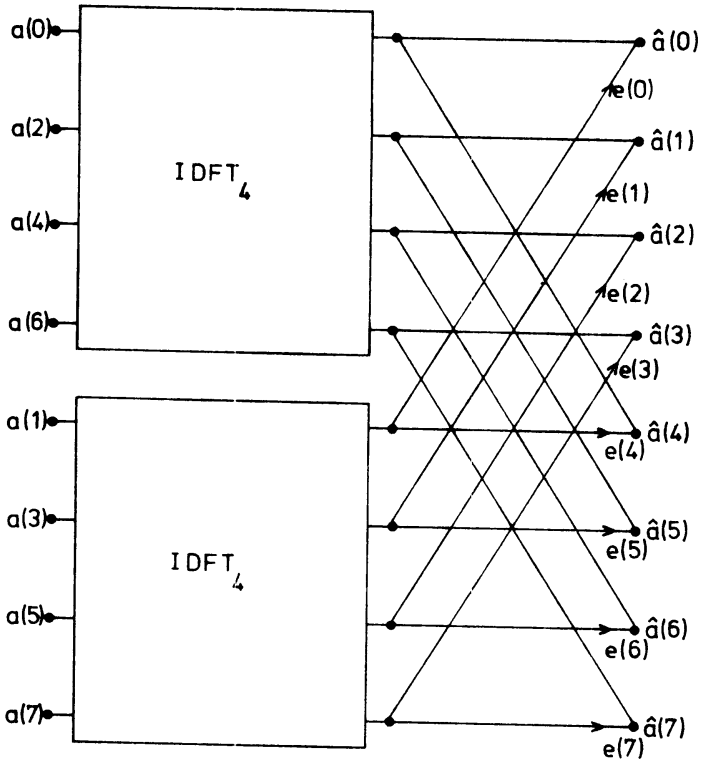
$$= \sum_{s=0}^{p-1} a(2s)e_p(s\hat{s}) + \Big[\sum_{s=0}^{p-1} a(2s+1)e_p(s\hat{s})\Big].e_N(\hat{s}+\hat{t}p) \tag{3.5}$$

$$= \hat{a}_0^2(\hat{s}) + \hat{a}_1^2(\hat{s}).e_N(\hat{s}+\hat{t}p) \tag{3.6}$$

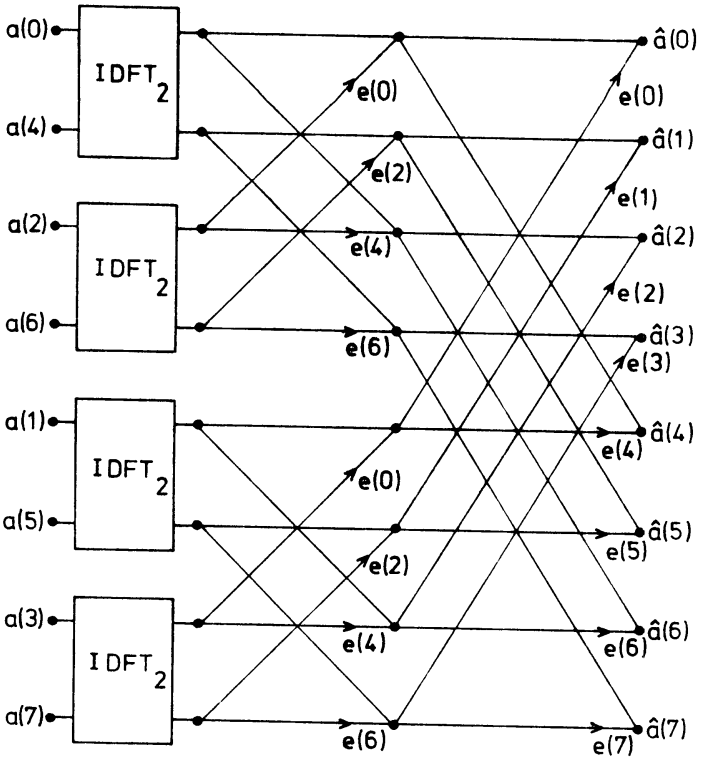
In fig.1 wordt (3.6) toegepast met p=4. Op elke $IDFT_4$ in fig.1 kan nu (3.6) worden toegepast. Dit levert fig.2 op. Uitschrijven van elke $IDFT_2$ levert fig.3. We zien dat de inputdata (links) in een andere volgorde staat dan de normale. Laat $(i_2,i_1,i_0) = 4i_2 + 2i_1 + i_0$ de binaire voorstelling van een getal), dan staat $a((i_2,i_1,i_0))$ op plaats (i_0,i_1,i_2). We zullen dit de omgekeerd binaire volgorde noemen. Wanneer we onze invoergegevens in de computer op omgekeerd binaire volgorde in een array zetten, heeft dit als voordeel, dat de berekening ter plaatse in het array gedaan kunnen worden, d.w.z. tussenresultaten worden over de inputdata heengeschreven, het eindresultaat over de tussenresultaten. We kunnen dus werken met een minimum aan hulpgeheugen.

We kunnen het schema van fig.3 ook zodanig omwerken dat de inputdata in normale volgorde staan maar de output in omgekeerd binaire volgorde (zie fig.4). Dat schema heeft echter als nadeel dat de faktoren waarmee vermenigvuldigd moet worden niet meer in de natuurlijke volgorde staan.

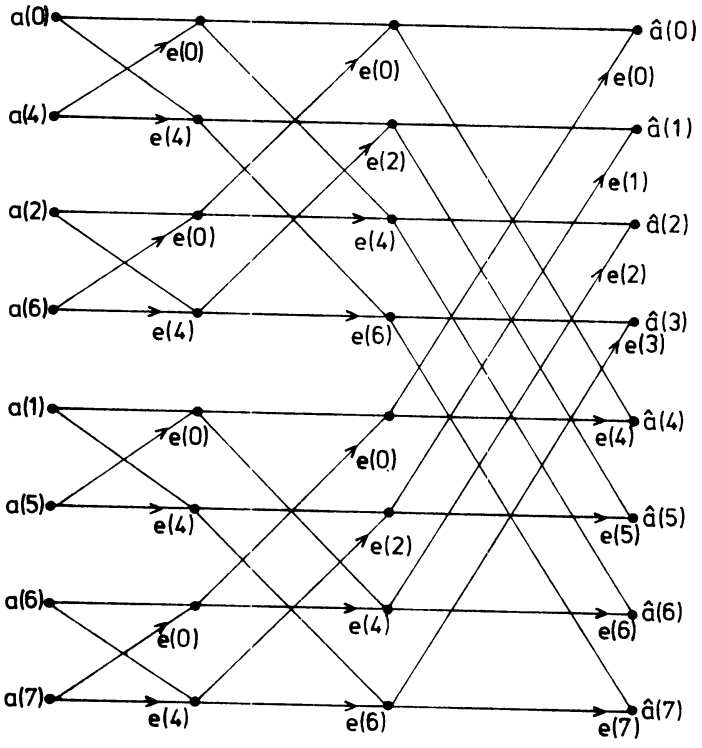
In het algemene geval dat $N=p_1 p_2 \cdot \ldots \cdot p_k$ kan men ook "ter plaatse" berekenen. Indien b.v. de input in normale volgorde staat, dan staat de output als volgt:



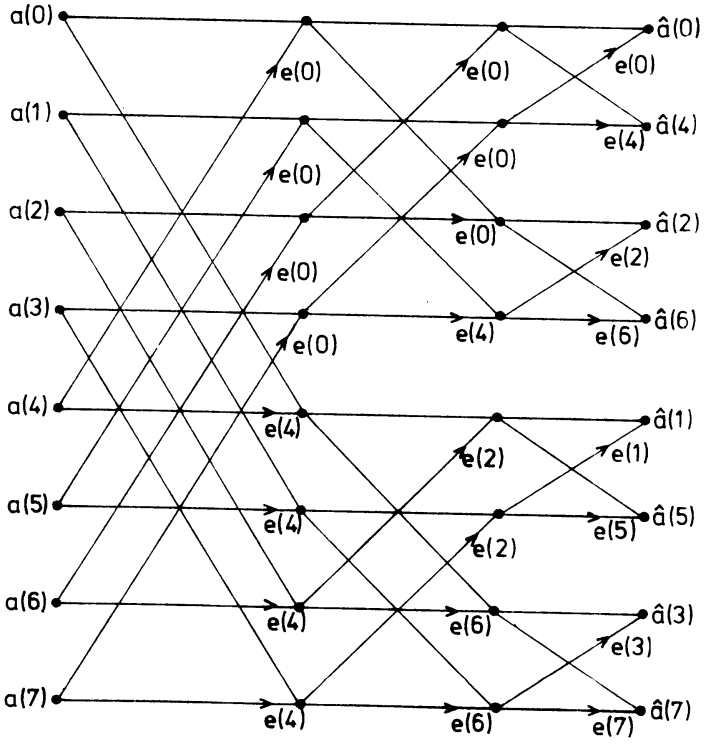
figuur 1



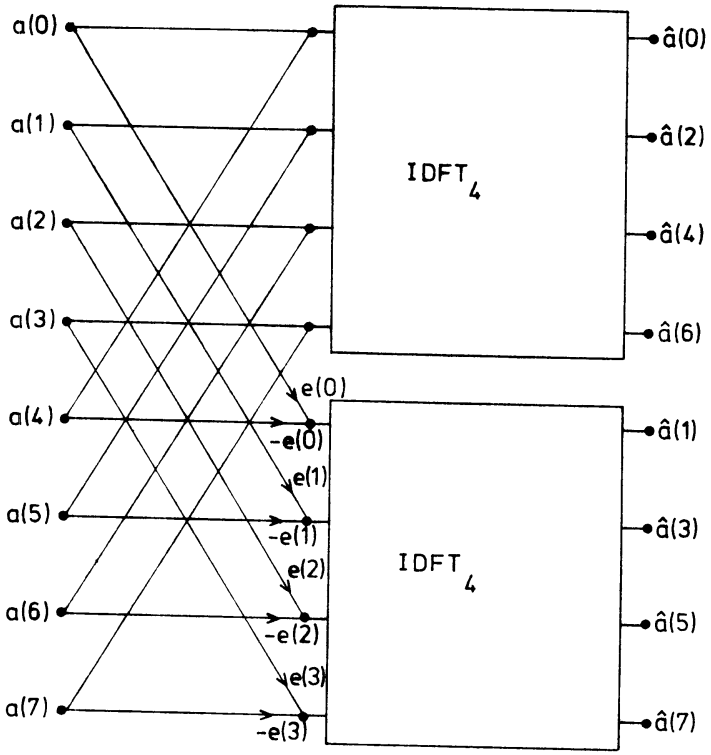
figuur 2



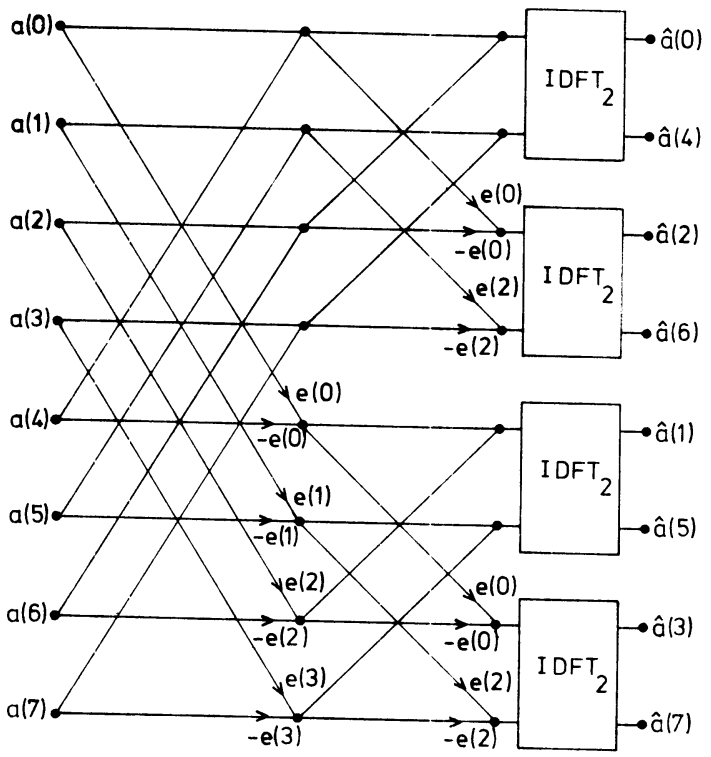
figuur 3



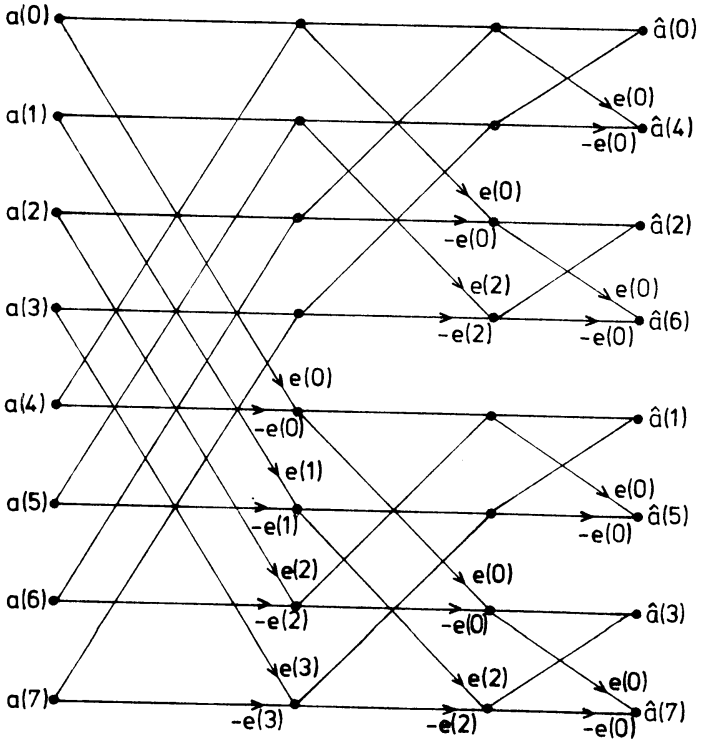
figuur 4



figuur 5

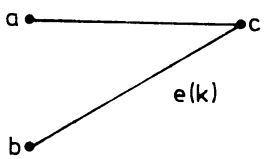


figuur 6



figuur 7

In de figuren 1 t/m7 betekent:



$c = a + b.e_8(k)$

Laat: $\lceil s_1,s_2,\ldots,s_k \rceil = s_1(p_2p_3..p_k)+s_2(p_3p_4..p_k)+..+s_k$

en $\lfloor s_k,..,s_2,s_1 \rfloor = s_k(p_1p_2..p_{k-1})+s_{k-1}(p_1p_2..p_{k-2})+..+s_1$

Dan staat $\hat{a}(\lceil u_1,u_2,..,u_k \rceil)$ op plaats $\lfloor u_k,u_{k-1},..,u_1 \rfloor$.
Zie hiervoor Bergland (12) en Gentleman en Sande (5).

decimation in frequence

voor p=2 wordt (3.3):

$$\hat{a}(\hat{s}+2\hat{t}) = \sum_{t=0}^{q-1} e_N(t(\hat{s}+2\hat{t})) \sum_{s=0}^{1} a(t+sq)e_2(s\hat{s}) \qquad \begin{matrix} s,\hat{s} \in \mathbb{Z} \bmod 2 \\ t,\hat{t} \in \mathbb{Z} \bmod q \end{matrix}$$

(3.7) indien $\hat{s}=0$: $\hat{a}(2\hat{t}) = \sum_{t=0}^{q-1} \{a(t)+a(t+q)\}e_q(t\hat{t})$

(3.8) indien $\hat{s}=1$: $\hat{a}(2\hat{t}+1) = \sum_{t=0}^{q-1} \{a(t)-e_N(t)a(t+q)\}e_q(t\hat{t})$

In fig.5 worden (3.7) en (3.8) toegepast met q=4. Op elke $IDFT_4$ in fig.5 kunnen nu (3.7) en (3.8) toegepast worden met q=2. Dit levert fig.6 op. Uitschrijven van elke $IDFT_2$ levert fig.7 op.

De in hoofdstuk 5 te behandelen procedures FFT, FFT2, FFT4, FFT8 zijn gebaseerd op een schema soortgelijk aan fig.7 (de inputdata in normale volgorde), REVFFT2, REVFFT4, REVFFT8 zijn gebaseerd op schema's soortgelijk aan fig.3 (de inputdata in omgekeerd binaire orde).

---

4. Doorwerking van afrondfouten in de FFT

Doordat een rekenmachine met een eindig aantal bits per getal werkt zullen afrondfouten optreden, die een onnauwkeurigheid in de berekende Fourier getransformeerde introduceerde.

Voor het geval dat de rekenmachine met floating-point getallen met een mantisse van b bits werkt en na berekeningen afrondt (dus niet afkapt), hebben Gentleman en Sande (5) het volgende afgeleid:

voor berekening met behulp van de definiërende formule

$$||\hat{a}_B - \hat{a}||_E < 1.06 \sqrt{N}\ (2N)^{3/2}\ 2^{-b}\ ||\hat{a}||_E$$

voor berekening met behulp van het FFT-algorithme

$$||\hat{a}_B - \hat{a}||_E < 1.06 \sqrt{N}\ \sum_{i=1}^{k} (2p_i)^{3/2}\ 2^{-b}\ ||\hat{a}||_E$$

waarbij $\hat{a}_B$ de berekende getransformeerde van a is en $N = p_1 p_2 p_3 \cdots p_k$.

Op de EL-X8 is b=40. Voor N=1024 geeft dit als bovengrens voor de relatieve nauwkeurigheid van $||\hat{a}||_E$ respectievelijk $3{,}6 \times 10^{-6}$ en $2{,}6 \times 10^{-9}$. We zien dus dat het FFT-algorithme naast het sneller rekenen, ook nog nauwkeuriger werkt.

Voor uitgebreidere foutenbeschouwingen zie Welch (8) en Kaneko en Liu (9).

---

5. FFT-procedures uit de Collected Algorithms from C.A.C.M.

De algorithmes 338, 339 en 345 van R.C. Singleton bevatten verscheidene FFT-procedures. Procedure FFT (uit alg.339) is bruikbaar voor elke N, procedures FFT2, REVFFT2 (uit alg.338), FFT4, FFT8, REVFFT4 en REVFFT8 (uit alg.345) in combinatie met REORDER of REVERSEBINARY zijn bruikbaar voor $N = 2^m$. Alle procedures voeren een complexe Fourier transformatie uit volgens definitie (2.2). FFT2, FFT4 en FFT8 zijn onderling uitwisselbaar, evenzo voor REVFFT2, REVFFT4 en REVFFT8.

Laat $N = N_1 \times N_2 \times N_3$

definieer: $a_q^p(i) = a(p \times N_2 \times N_3 + i \times N_3 + q) \quad i = 0,1,2,\ldots,N_2-1$

waarbij $p \in \mathbb{Z} \bmod N_1$

$q \in \mathbb{Z} \bmod N_3$

voor aanroep: Re(a(n)) in A[n]

Im(a(n)) in B[n]

A,B gedeclareerd als [0:N-1]

aanroep luidt: FFT(A,B,N,N_2,$N_2 \times N_3$);

na uitvoering: Re($\widehat{a_q^p}$(i)) in A[$p \times N_2 \times N_3 + i \times N_3 + q$]

Im($\widehat{a_q^p}$(i)) in B[$p \times N_2 \times N_3 + i \times N_3 + q$]

indien $N_2 = 2^m$ kan de aanroep ook vervangen worden door:

```
FFT4(A,B,N,m,N2×N3);
REORDER(A,B,N,m,N2×N3,false);
```

of:

```
REORDER(A,B,N,m,N2×N3,false);
REVFFT4(A,B,N,m,N3);
```

indien $N_1=N_3=1$ en $N_2=N$, dan wordt het bovenstaande:

voor aanroep: Re(a(n)) in A[n]

Im(a(n)) in B[n]

A,B gedeclareerd als [0:N-1]

aanroep luidt: FFT(A,B,N,N,N)

na uitvoering: Re($\hat{a}$(n)) in A[n]

Im($\hat{a}$(n)) in B[n]

indien $N = 2^m$ kan de aanroep ook vervangen worden door:

```
FFT4(A,B,N,m,N);
REVERSEBINARY(A,B,m);
```

of:

```
REVERSEBINARY(A,B,m).
REVFFT4(A,B,N,m,1);
```

Bovenstaande aanroepen zijn bedoeld voor de transformatie van complexe functies. Vaak zal men echter geïnteresseerd zijn in de Fourier-transformatie van reële functies. Eenvoudigweg het imaginaire deel nul maken (array B met nullen vullen) is minder efficiënt. Het volgende hoofdstuk handelt hierover.

Hieronder volgt een tabel met enkele gemeten rekentijden op de EL-X8. Klasfou is een procedure die de IDFT berekent m.b.v. de definiërende formule (2.2)

N	REVERSEBINARY	FFT4	FFT	REVFFT4	Klasfou	
64	0.05	0.23	0.37	0.25	5.9	
128	0.10	0.61	0.87	0.65	23.8	
256	0.22	1.24	1.86	1.32	94.5	
512	0.43	3.02	4.28	3.26	380	
1024	0.87	6.14	9.04	6.62	-	tijden in seconden

De rekentijd van FFT4 + REVERSEBINARY is ongeveer $\frac{2mN}{2900}$ sec.

De rekentijd van FFT is ongeveer $\frac{N(p_1+p_2+\ldots+p_k)}{2200}$ sec.

waarbij $N = p_1 p_2 \cdot \ldots \cdot p_k$ (p_i priem)

De FFT-procedures zijn ook bruikbaar voor meerdimensionale Fourier transformaties. Aanroep in het geval van dimensie=2:
(zie ook definitie 2.28)

voor aanroep: Re(a(k,l)) in A[kxQ+l]
Im(a(k,l)) in B[kxQ+l]

aanroep luidt:
```
FFT(A,B,PxQ,P,PxQ);
FFT(A,B,PxQ,Q,Q);
```

na uitvoering: Re(â(k,l)) in A[kxQ+l]
Im(â(k,l)) in B[kxQ+l]

---

6. De berekening van sinus- en cosinuscoëfficiënten m.b.v. de FFT

Laat $f(n) \in \mathbb{Z}$ voor alle $n \in \mathbb{Z}$ mod N.

Definieer M zodanig dat $2M = N$ indien N is even

en $2M+1 = N$ indien N is oneven.

We kunnen nu sinus- en cosinuscoëfficiënten a_k en b_k definiëren:

$$(6.1)\quad a_k = \frac{1}{N} \sum_{n=0}^{N-1} f(n) \cos(2\pi\frac{nk}{N})$$

$$(6.2)\quad b_k = \frac{1}{N} \sum_{n=0}^{N-1} f(n) \sin(2\pi\frac{nk}{N}) \qquad k = 0,1,2,.....,M$$

In het geval dat N is even zijn $b_0=0$ en $b_M=0$. Indien N is oneven dan alleen $b_0=0$.

Het verband tussen de f(n) en de a_k en b_k is:

$$(6.3)\quad \text{indien N even:}\quad f(n) = \frac{a_0}{2} + \sum_{k=1}^{M-1} a_k\cos(2\pi\frac{nk}{N}) + b_k\sin(2\pi\frac{nk}{N}) + \frac{a_M}{2}(-1)^n$$

$$(6.4)\quad \text{indien N oneven:}\quad f(n) = \frac{a_0}{2} + \sum_{k=1}^{M-1} a_k\cos(2\pi\frac{nk}{N}) + b_k\sin(2\pi\frac{nk}{N})$$

Het verband tussen de a_k en b_k en de $\hat{f}$ en $\check{f}$ is:

$$(6.5)\quad a_k = \frac{2}{N}\,\mathrm{Re}(\hat{f}(k)) = 2\,\mathrm{Re}(\check{f}(k))$$

$$(6.6)\quad b_k = \frac{2}{N}\,\mathrm{Im}(\hat{f}(k)) = -2\,\mathrm{Im}(\check{f}(k)) \qquad k = 0.1.2.....,M$$

Hierdoor is de $IDFT_N$ bruikbaar om deze sinus- en cosinuscoëfficiënten te berekenen. Er zijn doordat f reëel is, mogelijkheden om $\hat{f}$ nog efficiënter te berekenen. In algorithme 1 worden twee reële transformaties uitgevoerd met behulp van één $IDFT_N$. In algorithme 2 wordt één reëele transformatie uitgevoerd met behulp van één $IDFT_M$. Voorwaarde voor algorithme 2 is dat N is even.

algorithme 1

f_1, f_2 reëel, dus $\bar{f}_1=f_1$, $\widetilde{\hat{f}}_1=\hat{f}_1$ evenzo voor f_2.

definieer $g = f_1 + i.f_2$, dan kunnen we op g de $IDFT_N$ toepassen ($i^2=-1$)

(6.7) $\hat{g} = \widehat{f_1 + i.f_2} = \hat{f}_1 + i.\hat{f}_2$

(6.8) $\tilde{\hat{g}} = \hat{\tilde{g}} = \widehat{f_1 + i.f_2} = \widehat{f_1 - i.f_2} = \hat{f}_1 - i.\hat{f}_2$

uit (6.7) en (6.8) volgt:

(6.9) $\hat{f}_1 = \frac{1}{2}(\tilde{\hat{g}} + \hat{g})$ $\qquad f_2 = \frac{i}{2}(\tilde{\hat{g}} - \hat{g})$

Dit algorithme vertaald in Algol-60 is:

procedure REALTWIN(F1,F2,n): integer n; array F1,F2;

voor aanroep: $f_1(n)$ in F1[n]
$f_2(n)$ in F2[n] n= 0,1,2,...,N-1
F1, F2 gedeclareerd als [0:N-1]

aanroep luidt: FFT(F1,F2,N,N,N);
REALTWIN(F1,F2,N);

na uitvoering: a_k in F1[k] voor k= 0,1...,M
b_k in F1[N-k] k= 0,1,...,M-1(,M)
waarbij a_k en b_k de sinus- en cosinuscoëfficiënten van f_1.
Evenzo voor f_2 en F2.

Dit algorithme kan ook omgekeerd worden, dus uit twee stellen Fouriercoëfficiënten twee reële functies bepalen. Dit gaat met
procedure COEFTWIN(F1,F2,n); integer n; array F1,F2;

voor aanroep: a_k in F1[k] voor k= 0,1,...,M
b_k in F1[N-k] k= 0,1,...,M-1(,M)
evenzo voor F2

aanroep luidt: COEFTWIN(F1,F2,N);
FFT(F1,F2,N,N,N);

na uitvoering: $f_1(n)$ in F1[n]
$f_2(n)$ in F2[n] n= 0,1,...,N-1

algorithme 2

f reëel, dus $\bar{f} = f$ en $\widetilde{\hat{f}} = \hat{f}$

Laat verder N = 2M

Definieer: $f_1(n) = f(2n)$

$f_2(n) = f(2n+1)$ $\quad n = 0,1,\ldots,M-1$

Definieer $g = f_1 + i.f_2$, dan kunnen we op g de $IDFT_M$ toepassen. Met behulp van algorithme 1 vinden we $\hat{f}_1$ en $\hat{f}_2$.

Volgens stelling (3.5) is dan $\hat{f}(n) = \hat{f}_1(n) + e_N(n).\hat{f}_2(n)$ met $n = 0,1,\ldots,N-1$, waarbij de n bij f en e_N geïnterpreteerd moet worden mod N, maar bij f_1 en f_2 als mod M.

In Algol-60 wordt dit algorithme gerealiseerd door:

```
procedure REALTRAN(A,B,n,evaluate);
integer n; boolean evaluate; array A,B;
```

voor aanroep: f(2n) in A[n]
f(2n+1) in B[n]
arrays A en B gedeclareerd als [0:M]

de aanroep luidt:
```
FFT(A,B,M,M,M);
REALTRAN(A,B,M,false);
```

na uitvoering: $a_k \times N$ in A[k]
$b_k \times N$ in B[k] $\quad k = 0,1,\ldots,M \quad N = 2M$

Dit algorithme kan ook omgekeerd worden, dus uit een stel Fourier-coëfficiënten de reële functie bepalen.

voor aanroep: $+a_k$ in A[k]
$+b_k$ in B[k] $\quad k = 0,1,\ldots,M$

aanroep luidt:
```
REALTRAN (A,B,M,true);
for i:=0,i+1 while i<M do B[i]:= -B[i];
FFT(A,B,M,M,M);
```

na uitvoering: $+2 \times f(2n)$ in A[n]
$-2 \times f(2n+1)$ in B[n]

---

7. Het verband tussen de continue en de discrete Fourier-transformatie

We kunnen drie Fourier-transformaties definiëren:

(i) de continue Fourier-transformatie

$$x^F(f) = \int_{-\infty}^{\infty} x(t)\, e^{2\pi i f t}\, dt \qquad f,t \in \mathbb{R} \tag{7.1}$$

$$= \int_{-\infty}^{\infty} x(t)\, \cos(2\pi f t)dt + i \cdot \int_{-\infty}^{\infty} x(t)\, \sin(2\pi f t)\, dt \tag{7.2}$$

(ii) de oneindige discrete Fourier-transformatie

$$x^F(f) = \sum_{t=-\infty}^{+\infty} x(t)\, e^{2\pi i f t} \qquad f \in [-\tfrac{1}{2},+\tfrac{1}{2}],\quad t \in \mathbb{Z} \tag{7.3}$$

(iii) de eindige discrete Fourier-transformatie

$$x^F(f) = \sum_{t=0}^{N-1} x(t)\, e^{2\pi i f t/N} \qquad f,t \in \mathbb{Z} \bmod N \tag{7.4}$$

De invèrses van deze transformaties zijn:

(7.5) van (i): $$x(t) = \int_{-\infty}^{\infty} x^F(f)\, e^{-2\pi i f t}\, df$$

(7.6) van (ii): $$x(t) = \int_{-\frac{1}{2}}^{\frac{1}{2}} x^F(f)\, e^{-2\pi i f t}\, df$$

(7.7) van (iii): $$x(t) = \frac{1}{N} \sum_{f=0}^{N-1} x^F(f)\, e^{-2\pi i f t/N}$$

We zullen in dit hoofdstuk de Fourier-getransformeerden van x (t) aan geven met $x^F(f)$. Welke transformatie bedoeld wordt blijkt uit het definitiegebied van f en t.

Er wordt verder niet ingegaan op de vraag onder welke condities bovenstaande getransformeerden bestaan en onderstaande stellingen gelden. Indien bv. x(t) periodiek, dan is het duidelijk dat de integraal van (7.1) niet bestaat voor bepaalde f. Periodieke signalen pakt men aan met sinus- en cosinuscoëfficiënten die analoog gedefinieerd worden als de Fouriercoëfficiënten in hoofdstuk 6 (Een werkwijze die neerkomt op de inverse transformatie van (ii)).

Beschouw x(t) en laat: $x^F(f) = a(f)$, $f,t \in \mathbb{R}$

Definieer: $x_1(j) = \Delta t.x(j.\Delta t)$ $\quad j \in \mathbb{Z}$

$$x_2(j) = \Delta t. \sum_{k=-\infty}^{\infty} x(j.\Delta t+kT) \quad j \in \mathbb{Z} \bmod \frac{T}{\Delta t}$$

$$a_1(f) = \sum_{k=-\infty}^{\infty} a(fF+kF) \qquad f \in [-\tfrac{1}{2},+\tfrac{1}{2}]$$

$$a_2(j) = \sum_{k=-\infty}^{\infty} a(j.\Delta f+kF) \qquad j \in \mathbb{Z} \bmod \frac{F}{\Delta f}$$

Nu is indien $F = \frac{1}{\Delta t}$: $\quad x_1^F(f) = a_1(f) \qquad f \in [-\tfrac{1}{2},+\tfrac{1}{2}]$

en indien $T = \frac{1}{\Delta f}$ en $F = \frac{1}{\Delta t}$: $x_2^F(j) = a_2(j) \qquad j \in \mathbb{Z} \bmod \frac{1}{\Delta t \Delta f}$

Voorbeeld van een toepassing:

Laat $x(t) = 10\, e^{3t} \sin(2\pi t)$ voor $t \in [0,+\infty]$

$= 0$ voor $t \in [-\infty,0]$

We nemen $\Delta t = \frac{1}{32}$ en $T = 3$.

Nu is: $x_2(j) = \frac{1}{32} \sum_{k=-\infty}^{\infty} x(j/32 + 3k) \approx \frac{1}{32}\, x(j/32)$

(de laatste gelijkheid aangezien x(t) zeer snel naar nul gaat)

Als we nu op $x_1(j) = \frac{1}{32}\,(j/32)$ $(j=0,1,..,95)$ transformatie (iii) toepassen, dan krijgen we dus een benadering voor a_2. Aangezien voor grote f de Fourier-getransformeerde snel naar nul gaat zal $a_2(j) \approx a(j.\Delta f) = a(j/3)$

Opmerking: Van vele meetreeksen, bv. een eindig seismisch signaal is het bekend dat de Fourier-getransformeerde voor grote f nul wordt vanwege responsietijden van instrumenten.

Wil men waarden van a(f) ook op meer dan 96 punten berekenen, dan kan dat door nullen toe te voegen aan de 96 x(t)-waarden en daarop een Fourier-transformatie volgens (iii) met grotere N uit te voeren.

Een stuk van een programma waarbij de 96 x(t)-waarden met een getallenband worden ingevoerd luidt:

```
for i:=0,i+1 while i<48 do
begin  A[i]:=READ; B[i]:=READ
end;
for i:=48,i+1 while i<200 do A[i]:=B[i]:=0;
FFT(A,B,200,200,200); REALTRAN(A,B,200,false);
scale:=1/64;
for i:=0,i+1 while i<200 do
begin  A[i]:= A[i]xscale; B[i]:= B[i]xscale
end;
```

Vermenigvuldigen met 1/64 was nodig vanwege t = 1/32 en een schaalfaktor ½ wegens het gebruik van REALTRAN.
Nu staat in A[i]de berekende waarde van Re(a(ix0.08))
en in B[i]de berekende waarde van Im(a(ix0.08))

Voor f = 0 (0.08) 2.40 hebben we de met het bovenstaande programma berekende waarden vergeleken met analytisch berekende waarden.

F	RE(A(F)) ANALYT.	RE(A(F)) BEREKEND	IM(A(F)) ANALYT.	IM(A(F)) BEREKEND
.00	+1.296	+1.291	+.0000	-.0000
.08	+1.298	+1.293	+.0812	+.0810
.16	+1.303	+1.298	+.1655	+.1655
.24	+1.310	+1.305	+.2565	+.2566
.32	+1.317	+1.312	+.3575	+.3575
.40	+1.321	+1.316	+.4726	+.4724
.48	+1.317	+1.312	+.6053	+.6053
.56	+1.297	+1.292	+.7585	+.7587
.64	+1.248	+1.243	+.9324	+.9323
.72	+1.157	+1.152	+1.1209	+1.1207
.80	+1.007	+1.002	+1.3083	+1.3084
.88	+.792	+.786	+1.4667	+1.4668
.96	+.522	+.517	+1.5617	+1.5616
1.04	+.231	+.226	+1.5685	+1.5683
1.12	-.037	-.042	+1.4872	+1.4873
1.20	-.248	-.254	+1.3429	+1.3430
1.28	-.393	-.398	+1.1702	+1.1700
1.36	-.477	-.482	+.9971	+.9970
1.44	-.516	-.522	+.8398	+.8399
1.52	-.525	-.531	+.7046	+.7047
1.60	-.516	-.521	+.5918	+.5917
1.68	-.496	-.501	+.4991	+.4990
1.76	-.471	-.476	+.4232	+.4233
1.84	-.444	-.449	+.3612	+.3612
1.92	-.416	-.421	+.3103	+.3102
2.00	-.389	-.394	+.2682	+.2682
2.08	-.364	-.369	+.2334	+.2334
2.16	-.340	-.346	+.2043	+.2043
2.24	-.319	-.324	+.1798	+.1797
2.32	-.298	-.303	+.1591	+.1590
2.40	-.280	-.285	+.1414	+.1414
2.48	-.263	-.268	+.1263	+.1263
2.56	-.247	-.252	+.1133	+.1133
2.64	-.232	-.238	+.1020	+.1020
2.72	-.219	-.224	+.0922	+.0922
2.80	-.207	-.212	+.0837	+.0837
2.88	-.196	-.201	+.0761	+.0761
2.96	-.185	-.190	+.0695	+.0694
3.04	-.176	-.181	+.0636	+.0636
3.12	-.167	-.172	+.0584	+.0584
3.20	-.158	-.164	+.0537	+.0537

8. De procedures COVSPEK en FOUSPEK.

De procedures COVSPEK en FOUSPEK zijn rekenprocedures om tot schattingen te komen van het variantiespectrum van een reëelwaardig zwak stationair stochastisch proces. Uitgebreide theorie hierover vindt men in Harris (19), Grenander en Rosenblatt (20) en Doob(21). We zullen in het kort enkele begrippen uit de spectraalanalyse geven, zonder op de problemen in te gaan tot het verkrijgen van consistente schatters (de procedures geven alleen een oplossing voor reken- en efficiëntie-problemen!).

Laat $\{ \underline{a}_0(t) , \underline{a}_1(t) , \ldots\ldots , \underline{a}_{nf-1}(t)\}$ een stochastisch vektor-proces zijn van dimensie nf. Een onderstreept symbool stelt een complexe stochastische variabele voor. We zullen $t \in \mathbb{Z}$ beschouwen. Het continue geval loopt geheel analoog, zie Doob (21).

We definiëren:

$$C^{ij}(t_1,t_2) = E\ (\ \underline{a}_i(t_1)\ \overline{\underline{a}_j(t_2)}\ ) \tag{8.1}$$

waarbij E de verwachtingsoperator is.

Een proces heet zwak stationair indien:

(i) $E\ \underline{a}_i(t) = c_i$ (constante)

(ii) C^{ij} is een functie van t_2-t_1 , dus $C^{ij}(t_1+k,t_2+k) = C^{ij}(t_1,t_2)$

We zullen voortaan in het geval van zwakke stationariteit schrijven $C^{ij}(lag)$ met $lag=t_2-t_1$.

Zonder de algemeenheid van de theorie tekort te doen kunnen we stellen dat $c_i=0$ $(i=0,1,\ldots,nf-1)$.

C^{ij} wordt autocovariantiefunctie genoemd indien i=j en kruiscovariantie-functie indien i≠j.

Het spectrum van een zwak stationair proces wordt als volgt gedefinieerd:

$$S^{ij}(f) = \sum_{lag=-\infty}^{+\infty} C^{ij}(lag)\ e^{+2\pi i f \times lag} \qquad f \in [-\tfrac{1}{2},+\tfrac{1}{2}] \tag{8.2}$$

S^{ij} noemen we autospectrum indien i=j en kruisspectrum indien i≠j. Aangezien $C^{ij}(lag) = C^{ij}(-lag)$ oftewel $\overline{C^{ii}} = C^{ii}$ is $S^{ii} = S^{ii}$, dus het autospectrum is reëel. Kruisspectra zijn echter complex, ook als het proces reëelwaardig is. $Re(S^{ij})$ noemen we het kwadratuurspectrum en $Im(S^{ij})$ het cospectrum.

Indien we een realisatie van het vektorproces hebben voor t=0,1,2,.. ...,T-1 kunnen we C^{ij} als volgt schatten:

$$(8.3) \qquad C^{ij}(lag) = \frac{1}{T-lag} \sum_{t=0}^{T-lag-1} a_i(t)\ \overline{a_j(t+lag)} \quad (lag \geqslant 0)$$

In de formule (8.3) wordt $\frac{1}{T-lag}$ ook wel vervangen door $\frac{1}{T}$. Dit heeft als nadeel dat de schatter niet meer zuiver is, doch heeft ook voordelen, zodat vele auteurs hieraan de voorkeur geven. Zie hiervoor Parzen (23).

Door C^{ij} te schatten voor lag$\leqslant$M en C^{ij} nul te stellen voor lag>M kunnen we S^{ij} schatten met $IDFT_{2M}(C^{ij})$ voor $f = 0, \pm\frac{1}{2M}, \pm\frac{2}{2M}, \ldots, \pm\frac{M}{2M}$.

Deze methode van schatten (in de literatuur de Blackman-Tukey-methode genoemd) wordt door de procedure COVSPEK uitgevoerd.

Vaak wordt de geschatte covariantiefunctie nog vermenigvuldigd met een gewichtsfunctie (voor redenen hiervoor zie Blackman en Tukey (16) en Parzen (23)). Hieronder volgen de definities van de in de procedure COVSPEK ter beschikking staande gewichtsfuncties.

laat $u = \frac{lag}{M}$

(8.4.0) RECTAN
$$h_0(u) = 1 \quad |u| \leqslant 1$$
$$= 0 \quad |u| > 1$$

(8.4.1) HANNING
$$h_1(u) = \tfrac{1}{2}(1+\cos\pi u) \quad |u| \leqslant 1$$
$$= 0 \quad |u| > 1$$

(8.4.2) BARTLETT
$$h_2(u) = 1 - |u| \quad |u| \leqslant 1$$
$$= 0 \quad |u| > 1$$

(8.4.3) PARZEN
$$h_3(u) = 1 - 6u^2 + 6|u|^3 \quad |u| < \tfrac{1}{2}$$
$$= 2(1-|u|)^3 \quad \tfrac{1}{2} \leqslant |u| \leqslant 1$$
$$= 0 \quad |u| > 1$$

Het berekenen van de geschatte C^{ij} kost nogal wat rekentijd. Met behulp van het FFT-algorithme kan dit sneller (mits M en T groot genoeg). Voor het rekenwerk definiëren we $a_i(t) = 0$ voor $t \geqslant T$
Laat lim = entier ((T-1)/M), dan:

$$C^{ij}(lag) = \frac{1}{T} \sum_{k=0}^{lim} \sum_{t=kM}^{(k+1)M-1} a_i(t)\ \overline{a_j(t+lag)}$$

definieer:

(8.5) $a_{ik}(n) = a_i(n + kM) \qquad n = 0,1,2,\ldots,2M-1$

(8.6) $a^0_{ik}(n) = a_i(n + kM) \quad$ voor $n = 0,1,2,\ldots,M-1$

$= 0 \quad$ voor $n = M,M+1,M+2,\ldots,2M-1$

dan: $C^{ij}(lag) = \frac{1}{T} \lim_{k=0}\Sigma \sum_{n=0}^{2M-1} a^0_{ik}(n)\ \overline{a_{jk}(n+lag)} \qquad lag = 0,1,2,\ldots,M$

$= \frac{1}{T} \lim_{k=0}\Sigma\ (a^0_{ik} * \tilde{a}_{jk})\ (lag)$

$= \frac{1}{T} \lim_{k=0}\Sigma\ DFT_{2M}(\widehat{a^0_{ik} * \tilde{a}_{jk}})(lag)$

$= \frac{1}{T} \lim_{k=0}\Sigma\ DFT_{2M}(\hat{a}^0_{ik} \cdot \overline{\hat{a}}_{jk})(lag)$

(8.7) $= \frac{1}{T}\ DFT_{2M}\left(\lim_{k=0}\Sigma\ (\hat{a}^0_{ik} \cdot \overline{\hat{a}}_{jk})\right)(lag) \qquad lag = 0,1,2,\ldots,M$

verder is $a_{jk} = a^0_{jk} + a^0_{j(k+1)} * \delta_M$

(8.8) dus $\hat{a}_{jk} = \hat{a}^0_{jk} + \hat{a}^0_{j(k+1)} \cdot e_2 \qquad$ N.B. $e_2(n) = (-1)^n$

We kunnen dus door alle $\hat{a}^0_{jk}$ te berekenen, daarna wat vermenigvuldigingen en sommaties en een Fourier-transformatie terug de covariantiefunctie berekenen.

De procedure FOUSPEK werkt volgens een door Welch (15) ontwikkelde schattingstechniek. Hieronder volgt een korte beschrijving.

definieer: $a_{jk}(n) = a_j(n + k \times stap) \times wd(n) \qquad n = 0,1,2,\ldots,2M-1$

a_{jk} stelt dus een datasegment van een lengte 2M voor dat gewogen is met een "data-window".

Laat lim = entier ((T-2M)/stap),

een schatting voor $S^{ij}(n/2M)$ is dan:

(8.9) $\left[\lim_{k=0}\Sigma\ \hat{a}_{ik}(n).\overline{\hat{a}}_{jk}(n)\right] / \left[\sum_{l=0}^{2M-1} wd^2(l) \times (lim+1)\right] \qquad n = 0,\pm1,\pm2,\ldots,\pm M$

De in de procedure FOUSPEK ter beschikking staande data-windows zijn als volgt gedefinieerd:

laat $u = \frac{M-\frac{1}{2}-n}{M}$

(8.10.0) RECTAN $wd_0(n) = 1 \quad |u| \leq 1$
$= 0 \quad |u| > 1$

(8.10.1) HANNING-achtig $wd_1(n) = 1 - u^2 \quad |u| \leq 1$
$= 0 \quad |u| > 1$

(8.10.3) PARZEN-achtig $wd_3(n) = 1 - |u| \quad |u| \leq 1$
$= 0 \quad |u| > 1$

De ervaringen met data-windows (8.10.1) en (8.10.3) zijn niet zo best. Een betere aanpak lijkt het gebruik van (8.10.0) met smoothing achteraf van het spectrum met coëfficiënten $\frac{1}{4}, \frac{1}{2}, \frac{1}{4}$(Hanning). Hiervoor kan men gebruik maken van de procedure SMOOTH.

De procedure COVSPEK en FOUSPEK verwerken gegevens die op een achtergrondgeheugen staan, in het geval EL-X8 van het KNMI is dit de drum.Dit heeft als voordeel dat er geen beperkingen zijn voor de lengte T van de tijdreeks, doch alleen voor de keuze van M. COVSPEK heeft een geheugenruimte nodig van ong. (2+nf)xnfxlagx4 + M + 250 (EL-X8)-woorden. Voor FOUSPEK is dit ong. (2+nf)xnfxlagx2 + 2M + 250 . Dit wil zeggen dat op de EL-X8 de maximale keuze voor M bij toepassing van COVSPEK is 2048×2^{-nf} en bij toepassing FOUSPEK 4096×2^{-nf} .
Verdere beperkingen zijn:
M moet een macht van 2 zijn.
nf ε {1,2,3,4}

Voor aanroep moeten de gegevens dus eerst op de drum gezet worden, hetgeen het beste in een binnenblok kan gebeuren, zodat de hiervoor te gebruiken geheugenruimte bij aanroep van FOUSPEK of COVSPEK weer ter beschikking kan zijn. In dat binnenblok declareren we een array FL[0:nf-1,0:T-1], waarna $a_i(t)$ in FL [i,t] komt. Dit array wordt met de volgende statements op de drum gezet:

```
outarray(drum,adres,FL); hold(FL);
```

waarbij adres als integer is gedeclareerd en de adresplaats bevat vanaf waar FL wordt weggezet op de drum.

De aanroep van COVSPEK luidt:

COVSPEK(nf,M,adres,weg,T,window);

waarbij weg als integer is gedeclareerd en de adresplaats bevat vanaf waar de procedure de resultaten op drum moet wegzetten, window is ook een integer en geeft aan dat de gewichtsfunctie (8.4.window) gebruikt moet worden.

De aanroep van FOUSPEK luidt:

FOUSPEK(nf,M,adres,weg,T,window,stap);

waarbij window aangeeft dat data-window (8.10.window) gebruikt moet worden.

De resultaten zijn op te halen door de aanroepen:

GETCOV(C,weg,nf,M); (alleen in geval van gebruik COVSPEK)

GETSPEK(S,weg,nf,M);

waarbij S,C gedeclareerd zijn als array [0:(M+1)×nf×nf - 1] .

Na uitvoering staat:

$C^{ij}(lag)$ in $C[lag \times nf \times nf + i \times nf + j]$ $\quad lag = 0,1,2,\ldots,M$

$Re(S^{ij}(k/2M))$ in $S[k \times nf \times nf + i \times nf + j]$ $\quad i \leq j$, $k = 0,1,2,\ldots,M$

$Im(S^{ij}(k/2M))$ in $S[k \times nf \times nf + j \times nf + i]$ $\quad i < j$, $k = 0,1,2,\ldots,M$

Het ophalen van de resultaten en de eventuele presentatie ervan kan ook weer gebeuren in een binnenblok, zodat de procedure COVSPEK en FOUSPEK zoveel mogelijk geheugenruimte tot hun beschikking hebben.

In de tabel hieronder vermelden we enkele op de EL-X8 gemeten rekentijden van procedure COVSPEK voor T = 1024. (Rekentijden van beide procedures zijn evenredig met T). De rekentijden voor FOUSPEK (met stap = M) zijn ong. 0,4× de rekentijden van COVSPEK.

M \ nf	1	2	3	4
32	12	24	39	56
64	13	26	43	64
128	16	35	60	91
256	21	48	85	xx
512	34	83	xx	xx

rekentijden in seconden.

De aanroep van SMOOTH kan gebeuren na FOUSPEK met window=0 en GETSPEK, waarna de aanroep luidt:

```
SMOOTH(S,M,nf,0.25,0.5,0.25);
```

Bij gebruik van procedure COVSPEK moeten tevens worden gedeclareerd de procedures FFT4, REVFFT4, UNSCRAMBEL, VUL, OPSLAG, WINDOW. Bij gebruik FOUSPEK is dit de procedure REVFFT4. Deze procedures worden door COVSPEK resp. FOUSPEK zelf aangeroepen.

Hieronder volgt nog een tabel met de benodigde geheugenruimtes in EL-X8-woorden voor de procedures zelf.

FFT4	576	REVFFT4	516
REVFFT4	516	FOUSPEK	1520
UNSCRAMBEL	274	SMOOTH	161
VUL	232		———
OPSLAG	942	totaal FOUSPEK-pakket	2197
WINDOW	256		
COVSPEK	1193	FFT	2919
GETCOV	38	REALTWIN	129
	———	COEFTWIN	149
totaal COVSPEK-pakket	4048	REALTRAN	216

Voorbeelden van gebruik procedures COVSPEK en FOUSPEK

1. Golfspectra

Van de afdeling Oceanografisch Onderzoek (OO) werd een reeks golfmetingen bewerkt met de procedures. Deze reeks was door dé afdeling OO reeds zelf bewerkt volgens de FOUSPEK-methode (met stap=2M, window=0, T=3328, hanning achteraf). Resultaten met de procedure FOUSPEK kwamen overeen met de resultaten van afdeling OO.
Met de procedure COVSPEK is de reeks ook bewerkt. De resultaten hiervan gebruiken we nu ter illustratie van het gebruik van gewichtsfuncties. Fig. 8,9,10,11 geven de resultaten na gebruikmaking van resp. RECTAN, HANNING, BARTLETT en PARZEN. Verbindingslijnen in de grafieken zijn alleen getrokken om de interpretatie te vergemakkelijken en hebben dus geen betekenis.

2. Spectra van kunstmatig gegenereerde tijdreeksen.

Met behulp van een methode beschreven in Levert en Van Galen (24) is een autoregressieve tijdreeks gegenereerd volgens de vergelijking:

$$a(t) = 0.75a(t-1) - 0.50a(t-2) + \varepsilon_t$$

waarbij ε_t een stochastische variabele is met normale verdeling, zodat

$$\mathbb{E}\,\varepsilon_t = 0 \text{ en } \mathbb{E}\,\varepsilon_{t_1}.\,\varepsilon_{t_2} = 0 \quad \text{indien } t_1 \neq t_2$$
$$= 0.5625 \quad \text{indien } t_1 = t_2$$

Voor dit proces kan men het theoretische spectrum berekenen. De variantie van het proces zelf is 1. Van dit proces hebben we meerdere realisaties gegenereerd van lengte 2500. In de grafieken van fig. 12 t/m 17 zien we telkens het met de procedure COVSPEK (M=64, window=3) berekende spectrum (aangeduid met $*$) en het theoretische spectrum (aangeduid met o).
Fig. 18 geeft een realisatie van dit autoregressieve proces.

3. Windspektra.

Met de procedure COVSPEK is het autospektrum berekent (met T=1194 en window=1, dus hanning) van de horizontale windcomponent langs de gemiddelde windrichting bij een gemiddelde windsnelheid van 13 m/sec.
De voor deze berekening benodigde meetreeks is afkomstig van een trivaanmeting van de onderafdeling Fysische Meteorologie. De berekende waarden zijn op een log-log grafiek uitgezet. De cijfers geven het aantal schattingen op die plaats aan. De berekening is uitgevoerd met meerdere waarden van M. In fig. 19 is M=32 en in fig. 20 is M=128. We zien dat bij groter wordende M het oplossend vermogen toeneemt, maar dat tegelijkertijd de variantie van de schattingen ook groter wordt en daarmee de betrouwbaarheid van de schattingen afneemt. In de figuren zijn lijnen getrokken met richtingscoefficient -5/3 wegens het theoretisch te verwachten gedrag van de spectrale dichtheid als $c.f^{-5/3}$ aan de kant van de hoge frequenties.

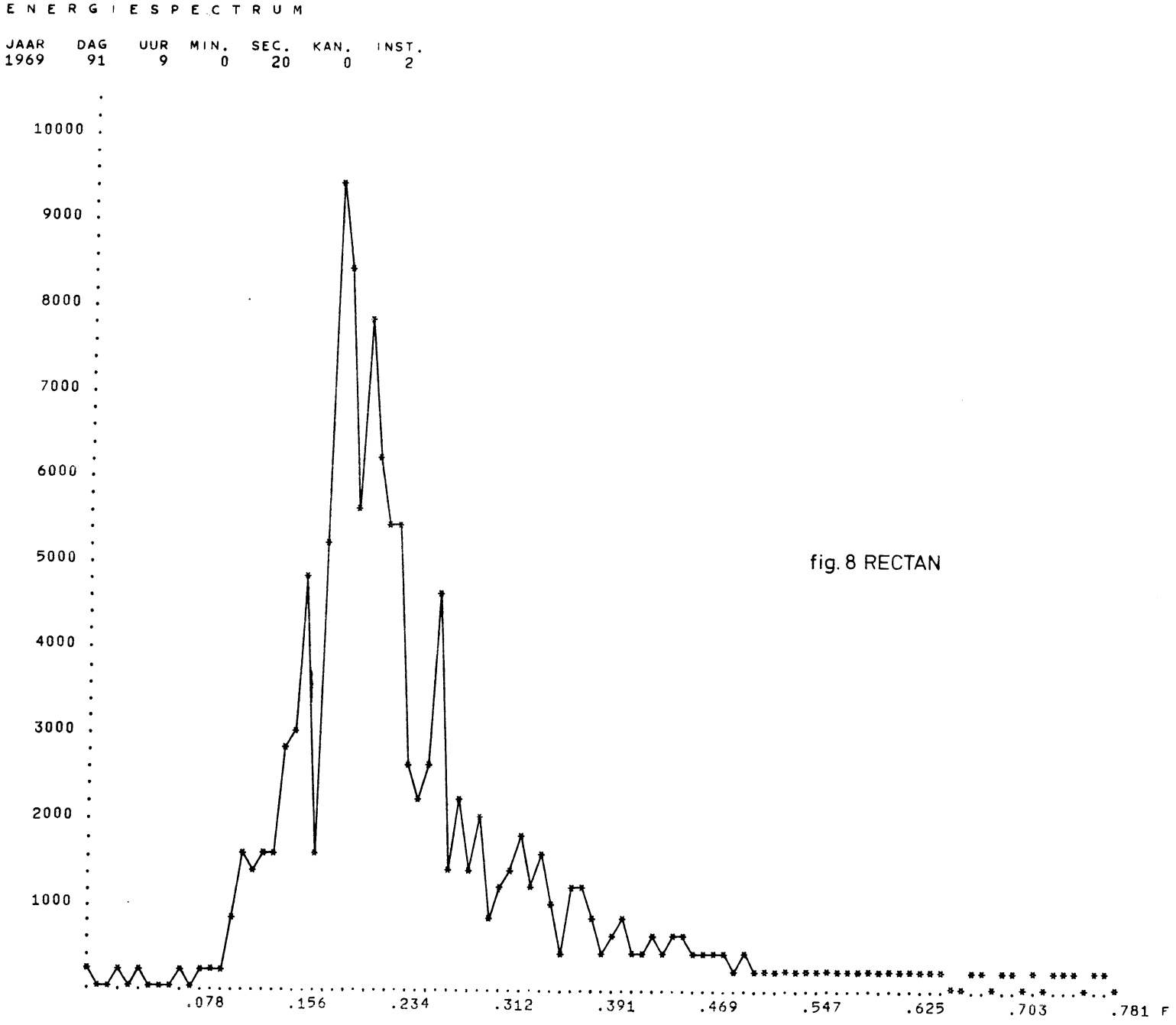

ENERGIESPECTRUM
JAAR DAG UUR MIN. SEC. KAN. INST.
1969 91 9 0 20 0 2
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
.078
.156
.234
.312
.391
.469
.547
.625
.703
.781 F

fig. 8 RECTAN

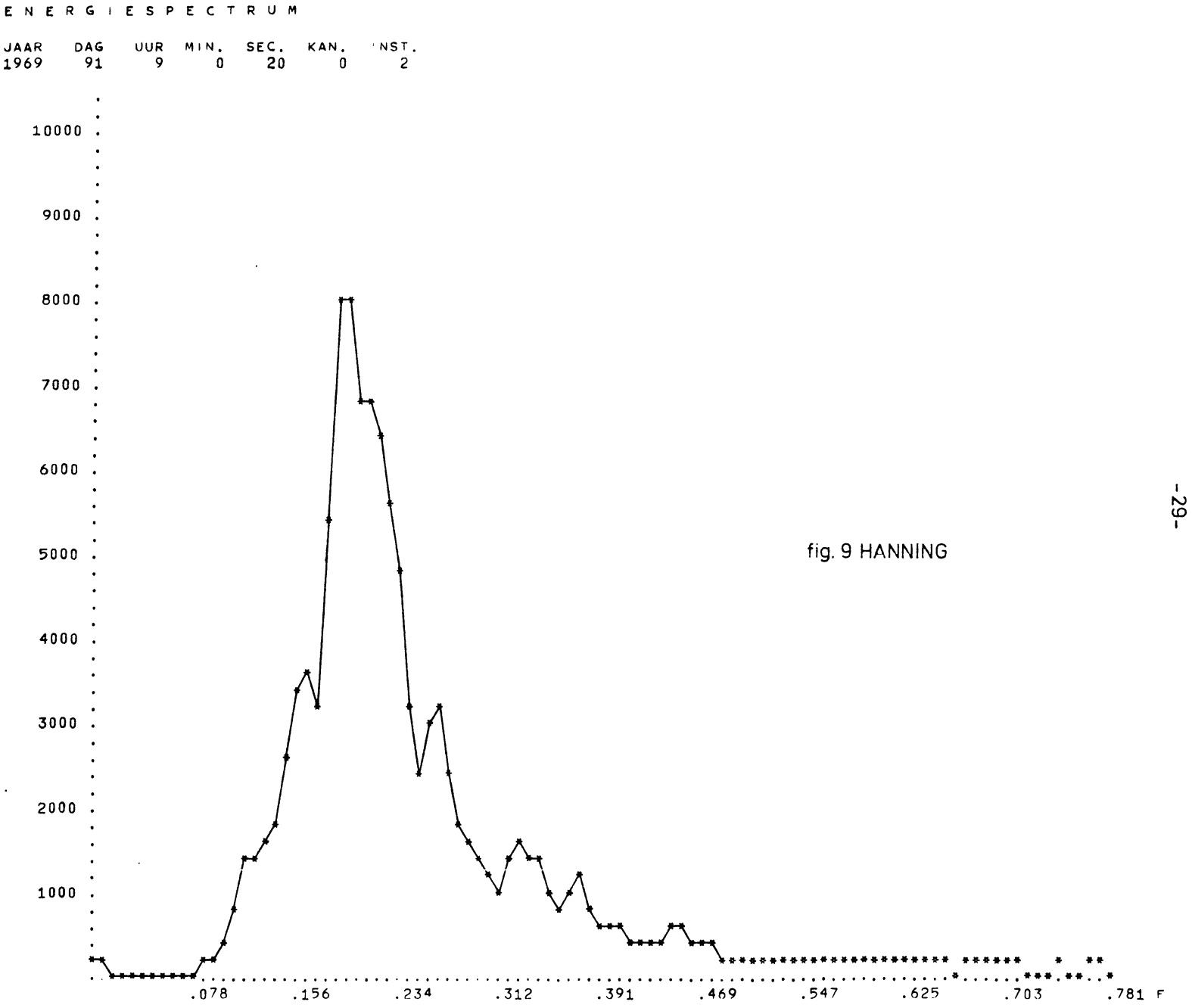


fig. 9 HANNING

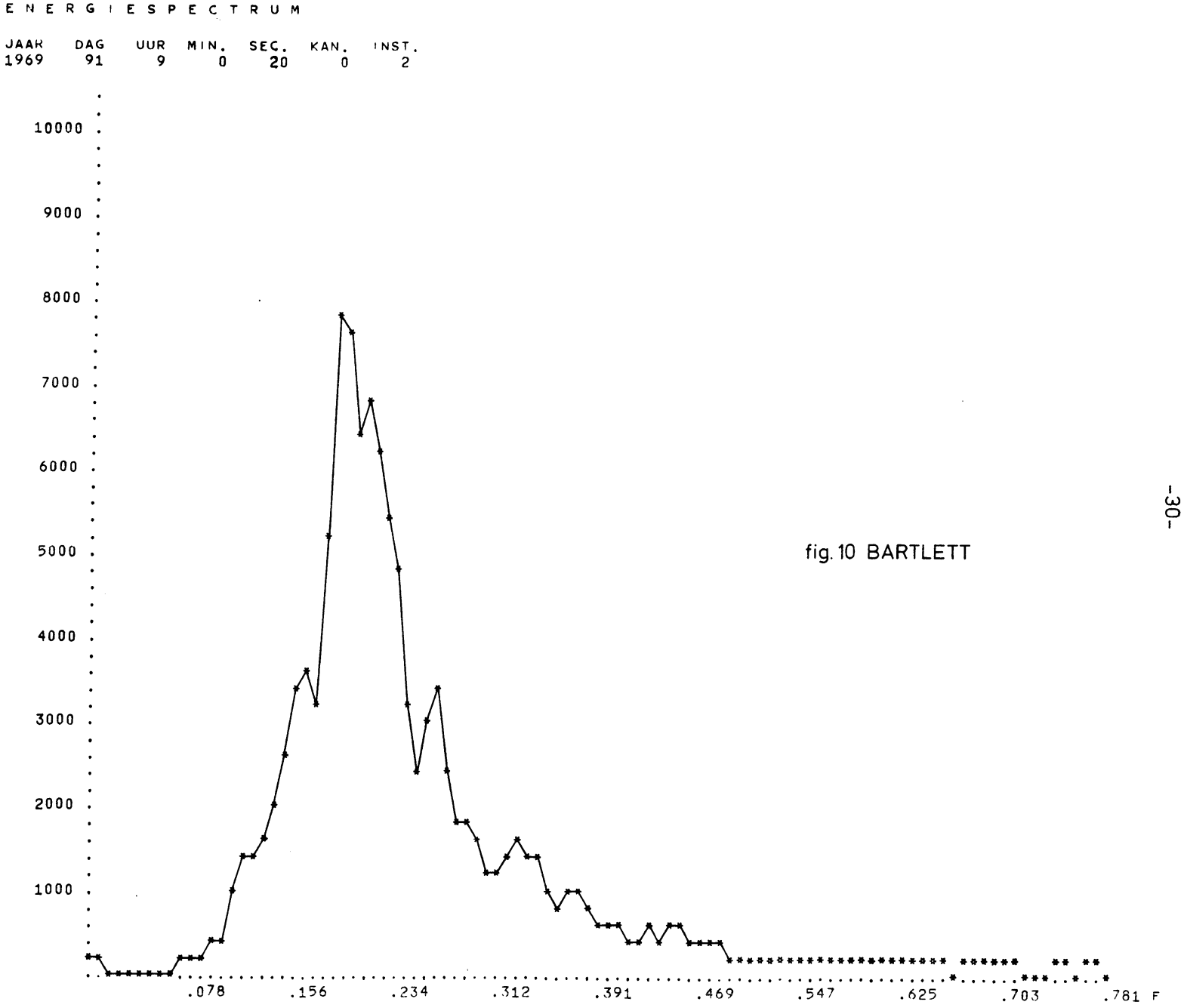


fig. 10 BARTLETT

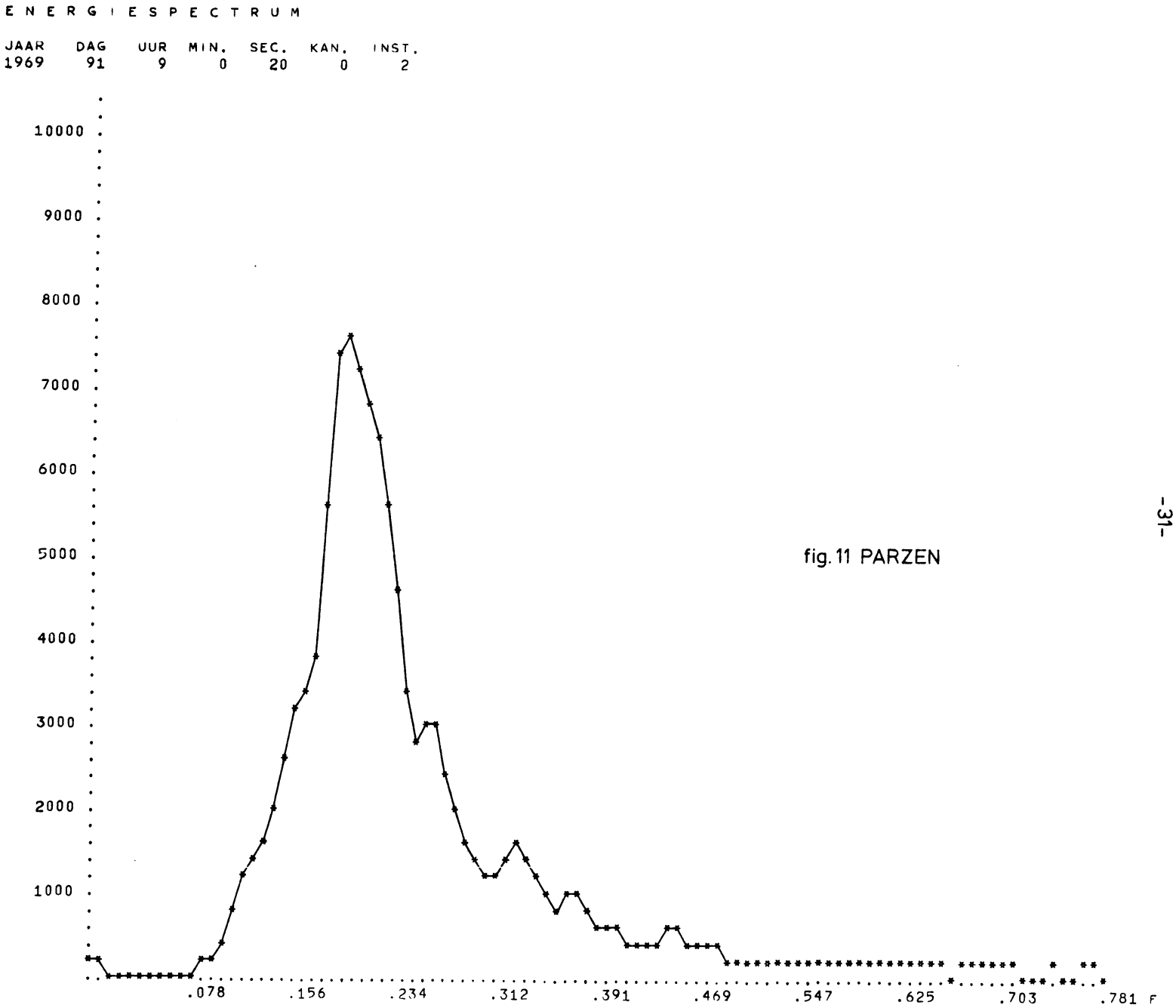


fig. 11 PARZEN

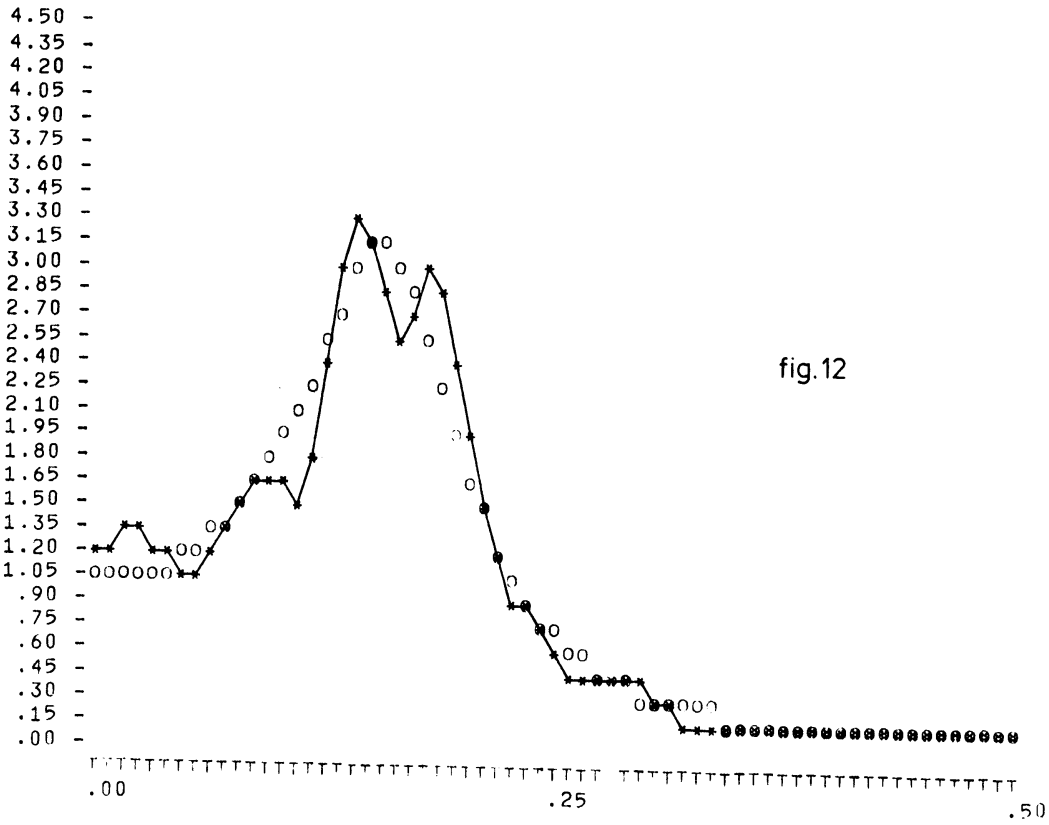

4.50 -
4.35 -
4.20 -
4.05 -
3.90 -
3.75 -
3.60 -
3.45 -
3.30 -
3.15 -
3.00 -
2.85 -
2.70 -
2.55 -
2.40 -
2.25 -
2.10 -
1.95 -
1.80 -
1.65 -
1.50 -
1.35 -
1.20 -
1.05 -
.90 -
.75 -
.60 -
.45 -
.30 -
.15 -
.00 -
.00
.25
.50

fig.12

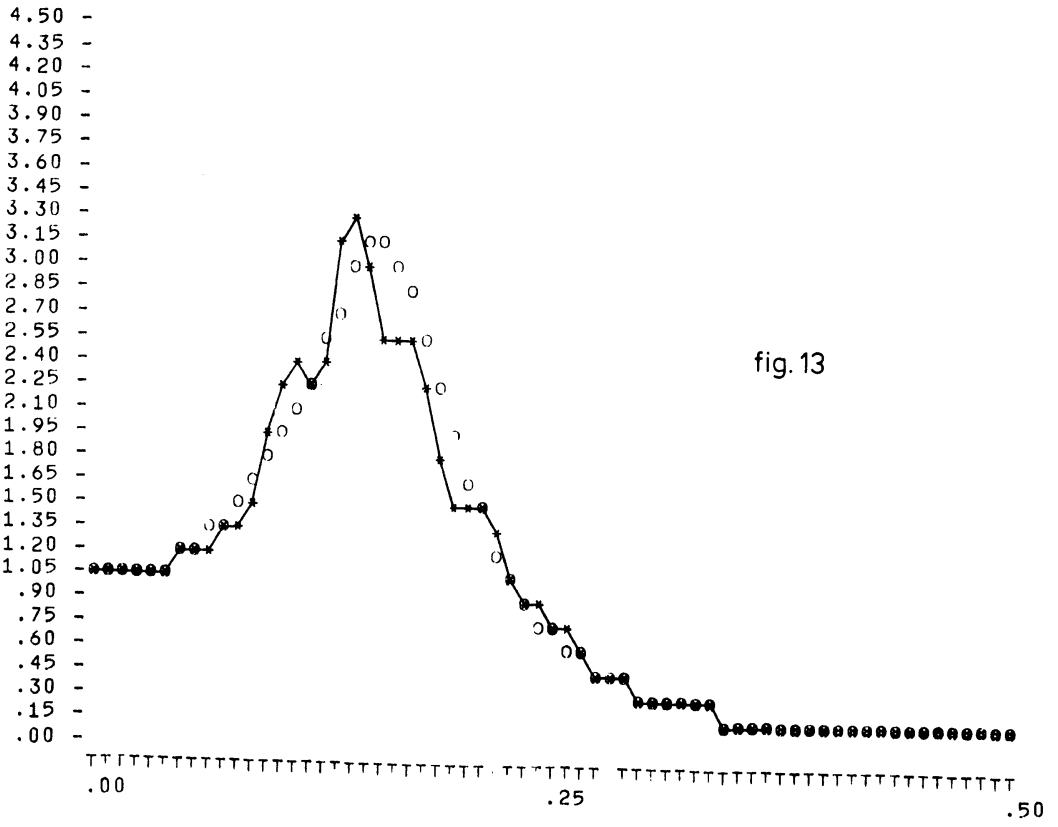

4.50 -
4.35 -
4.20 -
4.05 -
3.90 -
3.75 -
3.60 -
3.45 -
3.30 -
3.15 -
3.00 -
2.85 -
2.70 -
2.55 -
2.40 -
2.25 -
2.10 -
1.95 -
1.80 -
1.65 -
1.50 -
1.35 -
1.20 -
1.05 -
.90 -
.75 -
.60 -
.45 -
.30 -
.15 -
.00 -
.00
.25
.50

fig. 13

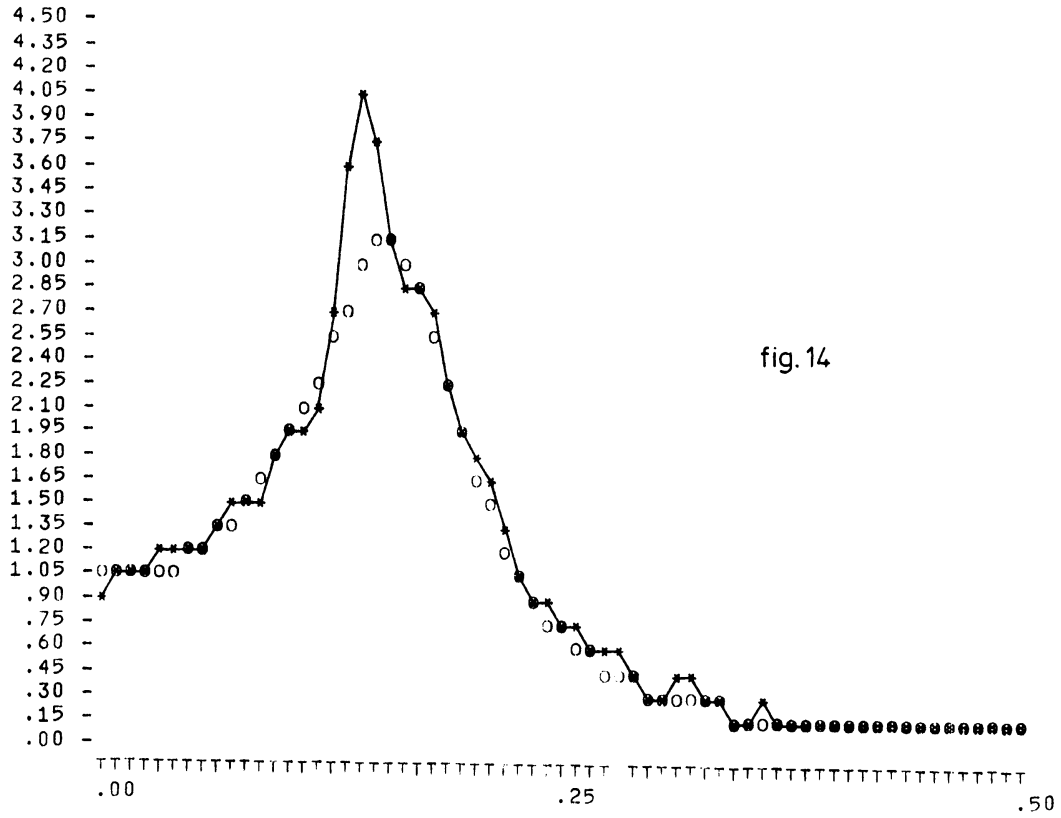

4.50
4.35
4.20
4.05
3.90
3.75
3.60
3.45
3.30
3.15
3.00
2.85
2.70
2.55
2.40
2.25
2.10
1.95
1.80
1.65
1.50
1.35
1.20
1.05
.90
.75
.60
.45
.30
.15
.00
.00
.25
.50

fig. 14

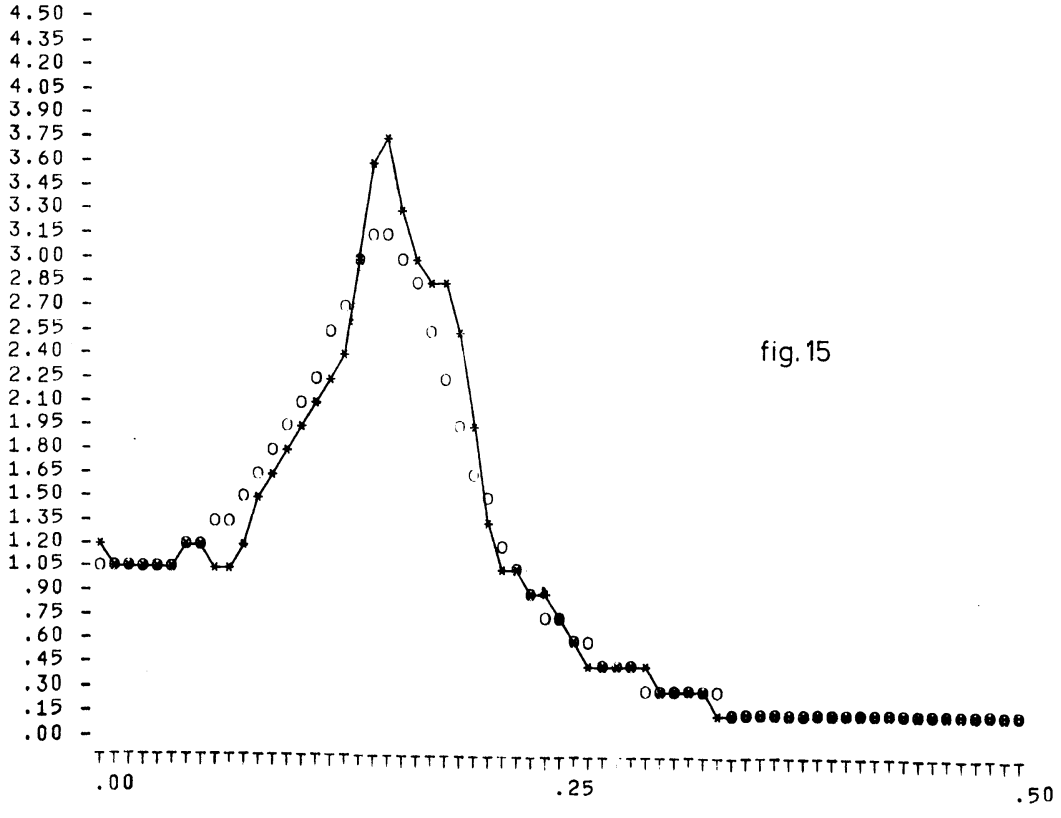

4.50
4.35
4.20
4.05
3.90
3.75
3.60
3.45
3.30
3.15
3.00
2.85
2.70
2.55
2.40
2.25
2.10
1.95
1.80
1.65
1.50
1.35
1.20
1.05
.90
.75
.60
.45
.30
.15
.00
.00
.25
.50

fig. 15

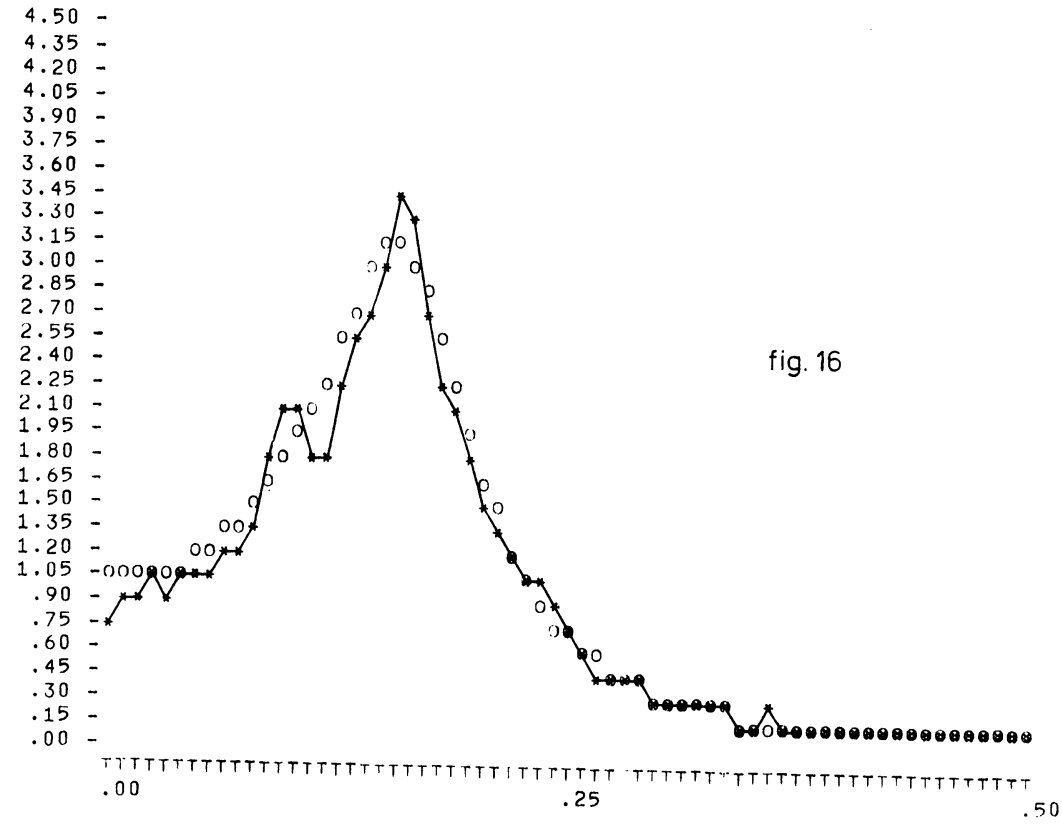

4.50 -
4.35 -
4.20 -
4.05 -
3.90 -
3.75 -
3.60 -
3.45 -
3.30 -
3.15 -
3.00 -
2.85 -
2.70 -
2.55 -
2.40 -
2.25 -
2.10 -
1.95 -
1.80 -
1.65 -
1.50 -
1.35 -
1.20 -
1.05 -
.90 -
.75 -
.60 -
.45 -
.30 -
.15 -
.00 -
.00
.25
.50

fig. 16

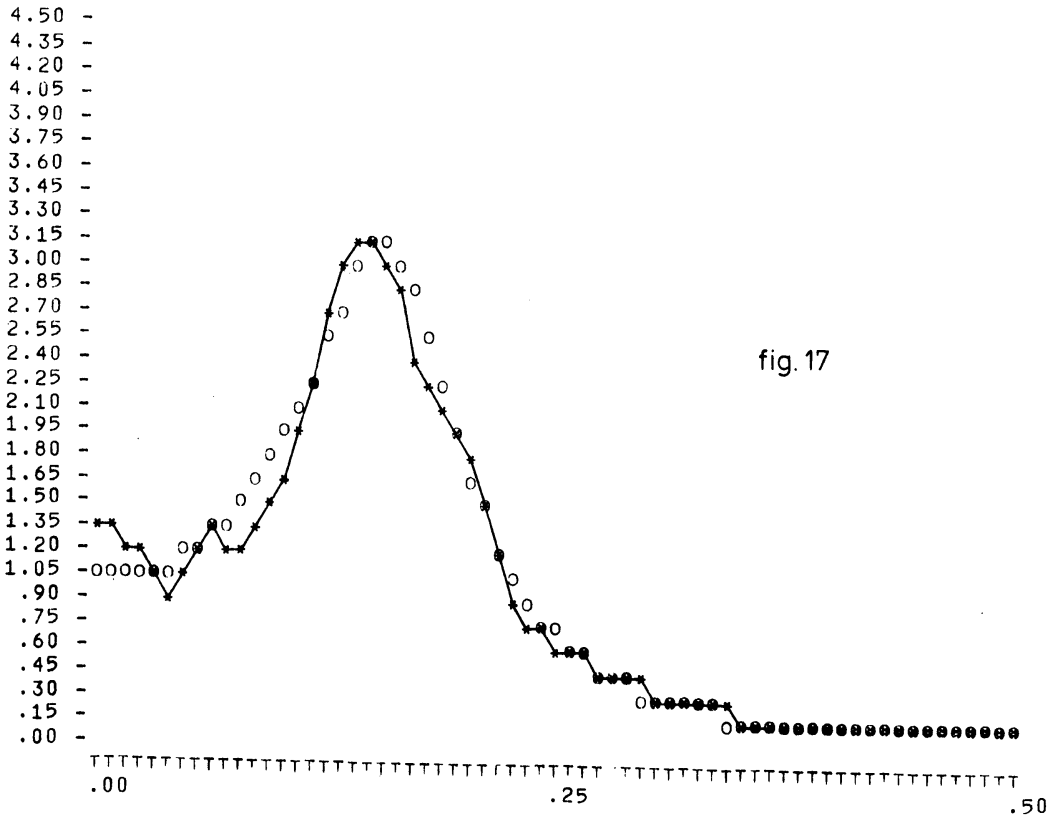

4.50 -
4.35 -
4.20 -
4.05 -
3.90 -
3.75 -
3.60 -
3.45 -
3.30 -
3.15 -
3.00 -
2.85 -
2.70 -
2.55 -
2.40 -
2.25 -
2.10 -
1.95 -
1.80 -
1.65 -
1.50 -
1.35 -
1.20 -
1.05 -
.90 -
.75 -
.60 -
.45 -
.30 -
.15 -
.00 -
.00
.25
.50

fig. 17

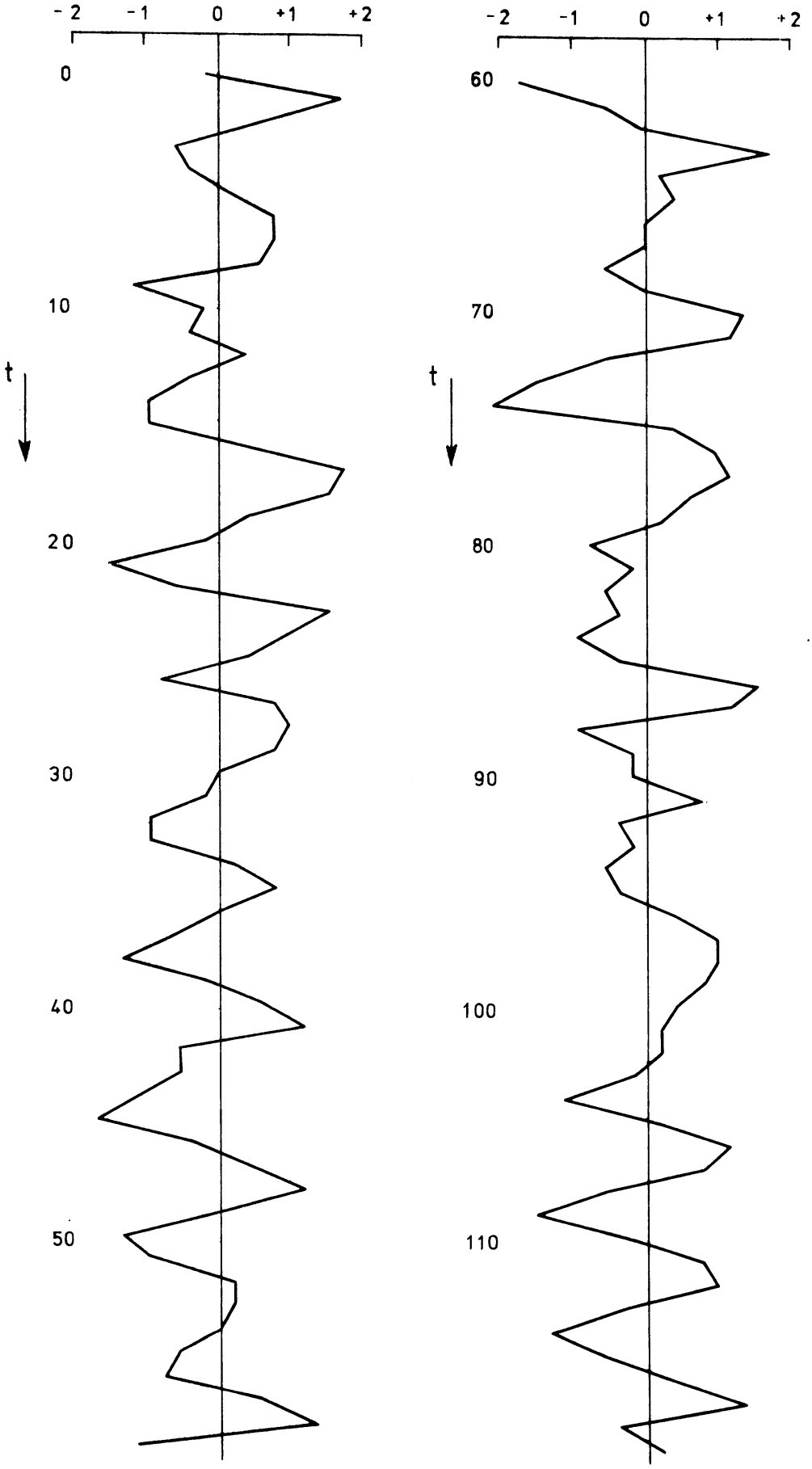


fig. 18

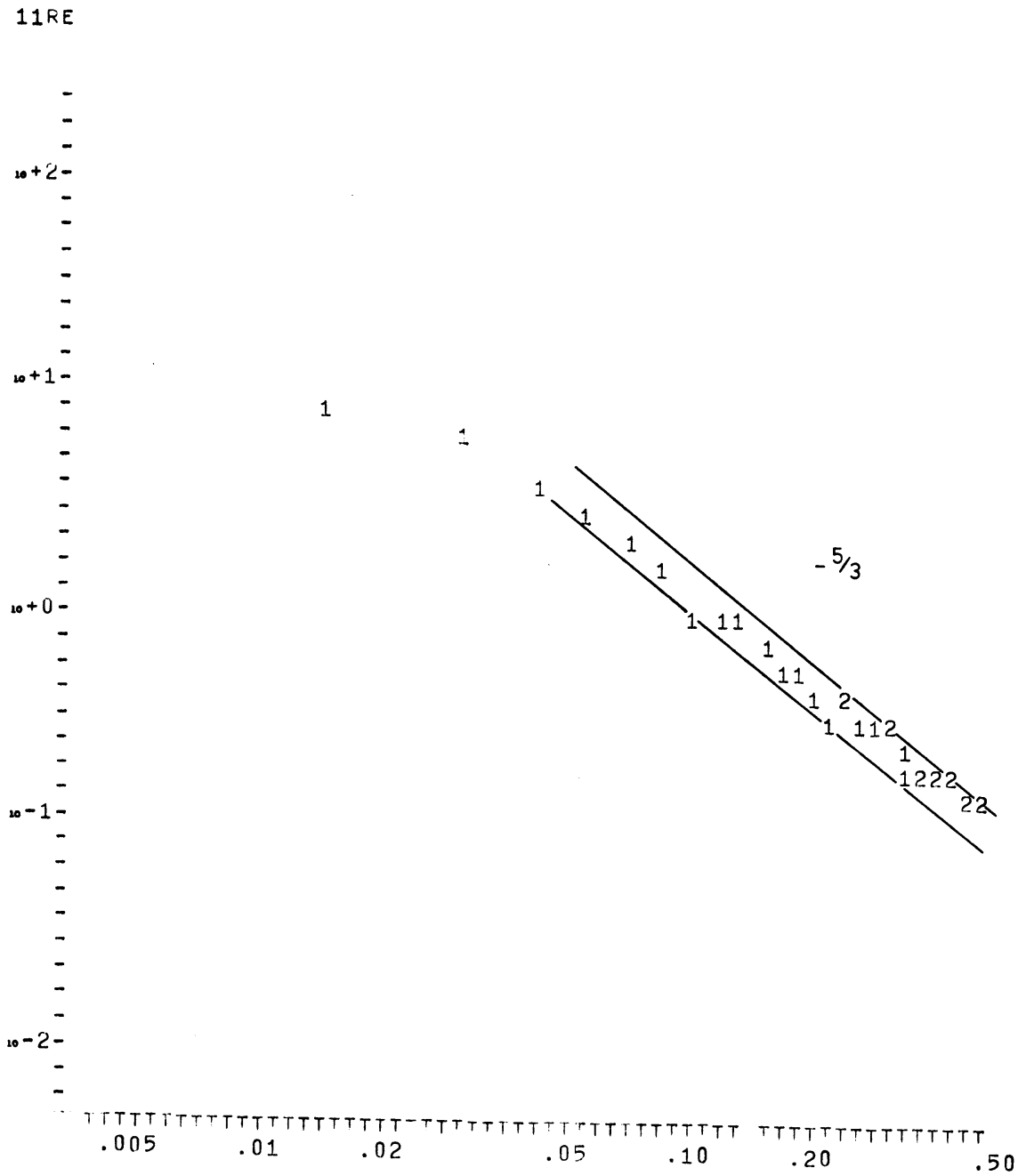


fig.19

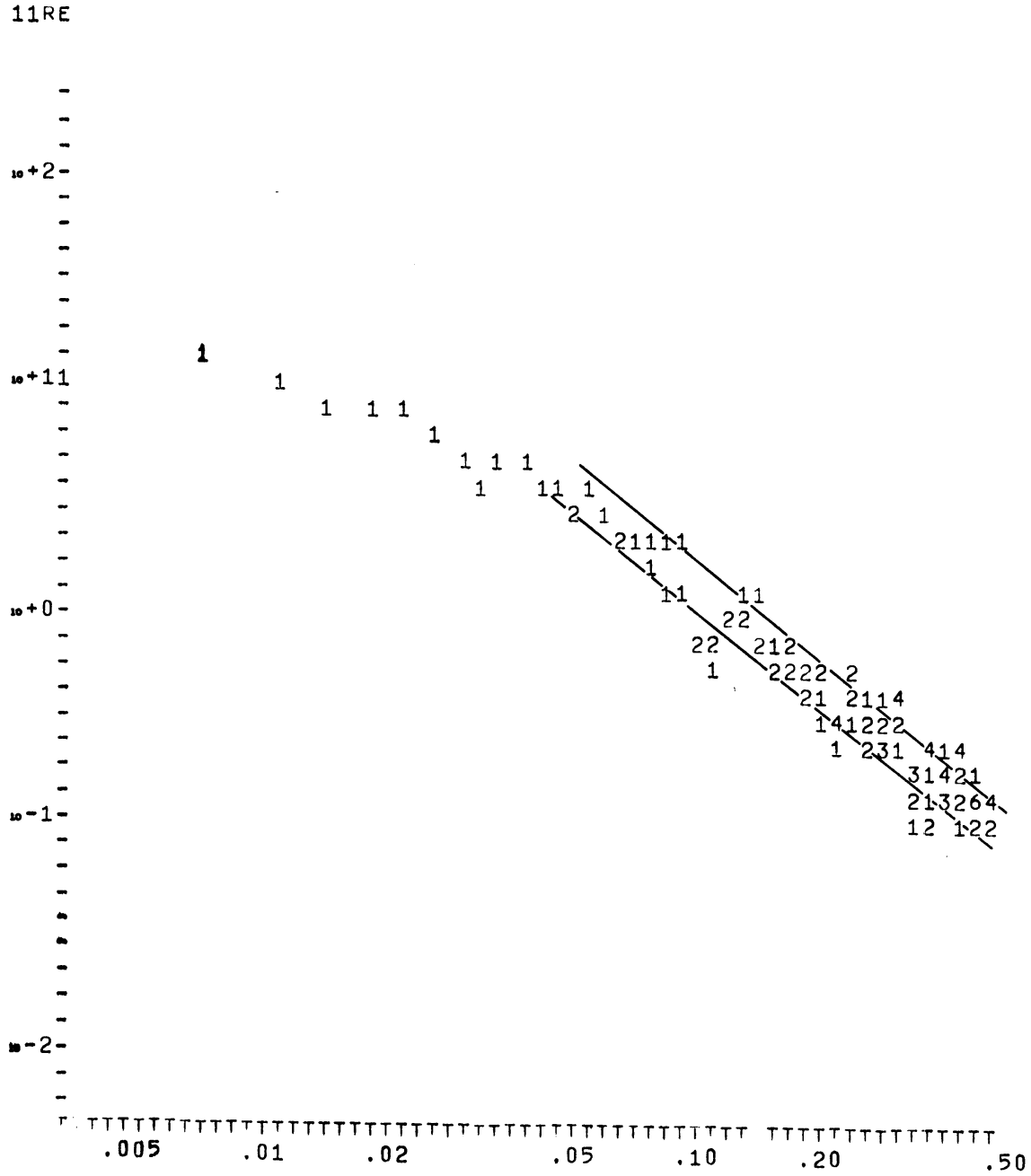


fig. 20

9. Literatuur

(1) Cooley, J.W., and Tukey, J.W.
An algorithm for the machine calculation of complex Fourier series.
Math. of Computation, vol. 19 (1965), pp. 297-301.

(2) Cooley, J.W., et al.
Historical notes on the fast Fourier transform.
IEEE Trans. on Audio- and Electroacoustics., vol. AU-15 (1967), no.2, pp. 76-79.

(3) Runge, C., and König, H.
Die Grundlehren der mathematischen Wissenschaften.
In: Vorlesungen über numerisches Rechnen, vol. 11. Berlin 1924.

(4) Edge, B.L., and Liu, P.C.
Comparing power spectra computed by Blackman-Tukey and fast Fourier transform.
Water Resources Res., vol. 6 (1970), no. 6, pp. 1601-1610.

(5) Gentleman, W.M., and Sande, G.
Fast Fourier transform - for fun and profit.
AFIPS Proc., vol. 19 (1966).

(6) Cochran, W.T., et al.
What is the fast Fourier transform.
IEEE Trans. on Audio and Electroacoustics. vol. AU-15 (1967), no. 2, pp. 45-55.

(7) Cooley, J.W., Lewis, P.A.W., and Welch, P.D.
The fast Fourier transform algorithm. Programming considerations in the calculation of sine, cosine and Laplace transform.
Journ. Sound. Vib., vol. 12 (1970), no.3, pp. 315-337.

(8) Welch, P.D.
A fixed-point fast Fourier transform error analysis.
IEEE Trans. on Audio and Electroacoustics, vol. AU-17 (1969), pp. 151-157.

(9) Kaneko, T., and Liu, B.
Accumulation of round-off error in fast Fourier transforms.
Journ. of the A.C.M., vol. 17 (1970), no. 4, pp. 637-654.

(10) Maas, W.A.K.
Fourier-reeksen en spectra.
Syllabus ERCU-107 (1971).

(11) Bergland, G.D.
A fast Fourier transform algorithms using base 8 iterations.
Math. of Computation, vol. 22 (1968), pp. 275-279.

(12) Bergland, G.D.
The fast Fourier transform recursive equation for arbitrary length records.
Math. of Computation, vol. 21 (1967), pp. 236-238.

(13) Singleton, R.C.
On computing the fast Fourier transform.
Comm. of the A.C.M., vol. 10 (1967), no. 10, pp. 647-654.

(14) Singleton, R.C.
Algorithms 338,339,345.
Collected Algorithms from C.A.C.M.

(15) Welch, P.D.
The use of fast Fourier transform for the estimation of power spectra.
A method based on time averaging over short modified periodograms.
IEEE Trans. on Audio and Electroacoustics, vol. AU-15 (1967) no.2
pp. 70-73.

(16) Blackman, R.B., and Tukey, J.W.
The measurement of power spectra.
New York, 1959.

(17) Papoulis, A.
The Fourier integral and its applications.
New York, 1962.

(18) Katznelson, Y.
An introduction to harmonic analysis.
New York, 1968.

(19) Harris, B. (Ed.)
Spectral analysis of time series.
New York, 1967.

(20) Grenander, U.,and Rosenblatt, M.
Statistical analysis of stationary time series.
New York, 1957.

(21) Doob, J.L.
Stochastic processes.
New York, 1963.

(22) Van Galen, J.
Spectraalanalyse van tijdreeksen.
Verslagen Kon. Nederlands Meteorologisch Instituut. V-212 (1968).

(23) Parzen, E.
Mathematical consideration in the estimation of spectra.
Technometrics, vol. 3 (1961), no. 2.

(24) Levert, C., and Van Galen, J.
Mathematical and statistical details on the simulation of
Markoff-type stochastic processes on an electronic computer.
Computing, vol. 3 (1968), pp. 65-75.

10. Appendix: teksten van de Algol-60-procedures

Voor de teksten van de procedures FFT, FFT2, FFT8, REVFFT2, REVFFT8, REORDER wordt verwezen naar de Collected Algorithms from C.A.C.M. (Singleton (14)).

teksten van:

FFT4
REVFFT4
UNSCRAMBEL
VUL
OPSLAG
WINDOW
COVSPEK
GETCOV
GETSPEK

FOUSPEK
SMOOTH

REALTWIN
COEFTWIN
REALTRAN
REVERSEBINARY

```
procedure FFT4(A,B,n,m,ks); value n,m,ks; integer n,m,ks; array A,B;
begin   integer k0,k1,k2,k3,k,span;
        real    A0,A1,A2,A3,B0,B1,B2,B3,re,im,
                rad,dc,ds,c1,c2,c3,s1,s2,s3;
        span:=ks; ks:=2↑m; rad:= 4.0×arctan(1.0)/ks;
        ks:=span÷ks; n:=n-1; k:=m;
        for m:=m-2 while m≥0 do
        begin   c1:=1.0; s1:=0; k0:=0; k:=ks; dc:=2.0×sin(rad)↑2;
                rad:=rad+rad; ds:=sin(rad); rad:=rad+rad; span:=span÷4;
        La:     k1:=k0+span; k2:=k1+span; k3:=k2+span;
                A0:=A[k0]; B0:=B[k0];
                A1:=A[k1]; B1:=B[k1];
                A2:=A[k2]; B2:=B[k2];
                A3:=A[k3]; B3:=B[k3];
                A[k0]:=A0+A1+A2+A3;
                B[k0]:=B0+B1+B2+B3;
                if s1=0 then
                begin   A[k1]:=A0+A2-A1-A3; B[k1]:=B0+B2-B1-B3;
                        A[k2]:=A0-A2-B1+B3; B[k2]:=B0-B2+A1-A3;
                        A[k3]:=A0-A2+B1-B3; B[k3]:=B0-B2-A1+A3;
                end
                else
                begin   re:=A0+A2-A1-A3; im:=B0+B2-B1-B3;
                        A[k1]:=re×c2-im×s2; B[k1]:=re×s2+im×c2;
                        re:=A0-A2-B1+B3; im:=B0-B2+A1-A3;
                        A[k2]:=re×c1-im×s1; B[k2]:=re×s1+im×c1;
                        re:=A0-A2+B1-B3; im:=B0-B2-A1+A3;
                        A[k3]:=re×c3-im×s3; B[k3]:=re×s3+im×c3;
                end;
                k0:=k3+span; if k0<n then goto La;
                k0:=k0-n;    if k0≠k then goto La;
                if k0≠span then
                begin   c2:=c1-(dc×c1+ds×s1);
                        s1:=(ds×c1-dc×s1)+s1; c1:=c2;
                        c2:=c1↑2-s1↑2; s2:=2.0×c1×s1;
                        c3:=c2×c1-s2×s1; s3:=c2×s1+s2×c1;
                        k:=k+ks; goto La
                end;
```

```
            k:=m;
        end;
        if k≠0 then
        begin   span:=span:2; k0:=0;
        Lb:     k2:=k0+span; A0:=A[k2]; B0:=B[k2];
                A[k2]:=A[k0]-A0; A[k0]:=A[k0]+A0;
                B[k2]:=B[k0]-B0; B[k0]:=B[k0]+B0;
                k0:=k2+span; if k0<n then goto Lb;
                k0:=k0-n; if k0≠span then goto Lb
        end
end     FFT4;


procedure REVFFT4(A,B,n,m,ks); value n,m,ks; integer n,m,ks; array A,B;
begin   integer k0,k1,k2,k3,k,span;
        real    A0,A1,A2,A3,B0,B1,B2,B3,
                rad,dc,ds,c1,c2,c3,s1,s2,s3;
        rad:=4.0×arctan(1.0); n:=n-1; k0:=0; span:=ks;
        if (m:2)×2≠m then
        begin
        La:     k2:=k0+span; A0:=A[k2]; B0:=B[k2];
                A[k2]:=A[k0]-A0; A[k0]:=A[k0]+A0;
                B[k2]:=B[k0]-B0; B[k0]:=B[k0]+B0;
                k0:=k2+span; if k0<n then goto La;
                k0:=k0-n; if k0≠span then goto La;
                span:=span+span; rad:=0.5×rad
        end;
        for m:=m-2 while m≥0 do
        begin   c1:=1.0; s1:=0; k0:=0; rad:=0.25×rad;
                dc:=2.0×sin(rad)↑2; ds:=sin(rad+rad); k:=ks;
        Lb:     k1:=k0+span; k2:=k1+span; k3:=k2+span;
                A0:=A[k0]; B0:=B[k0];
                if s1=0 then
                begin   A2:=A[k1]; B2:=B[k1];
                        A3:=A[k3]; B3:=B[k3];
                        A1:=A[k2]; B1:=B[k2];
                end
```

```
                else
                begin   A2:=A[k1]×c2-B[k1]×s2; B2:=A[k1]×s2+B[k1]×c2;
                        A1:=A[k2]×c1-B[k2]×s1; B1:=A[k2]×s1+B[k2]×c1;
                        A3:=A[k3]×c3-B[k3]×s3; B3:=A[k3]×s3+B[k3]×c3;
                end;
                A[k0]:=A0+A2+A1+A3; B[k0]:=B0+B2+B1+B3;
                A[k1]:=A0-A2-B1+B3; B[k1]:=B0-B2+A1-A3;
                A[k2]:=A0+A2-A1-A3; B[k2]:=B0+B2-B1-B3;
                A[k3]:=A0-A2+B1-B3; B[k3]:=B0-B2-A1+A3;
                k0:=k3+span; if k0<n then goto Lb;
                k0:=k0-n;    if k0≠k then goto Lb;
                if k0≠span then
                begin   c2:=c1-(dc×c1+ds×s1);
                        s1:=(ds×c1-dc×s1)+s1; c1:=c2;
                        c2:=c1↑2-s1↑2; s2:=2.0×c1×s1;
                        c3:=c2×c1-s2×s1; s3:=c2×s1+s2×c1;
                        k:=k+ks; goto Lb
                end;
                span:=4×span;
        end
end     REVFFT4;


comment deze procedure is een generalisatie van REALTRAN;
procedure UNSCRAMBEL (R,I,n,nf,evaluate);
value   n,nf,evaluate; integer n,nf; boolean evaluate; array R,I;
begin   integer nh,j,k1,k2;
        real    aa,ab,ba,bb,re,im,ck,sk,dc,ds;
        nh:=n:2; ds:=2.0×nf×arctan(1.0)/n; ck:=1.0;
        dc:=2.0×sin(ds)↑2; ds:=sin(ds+ds); sk:=0;
        if evaluate then begin ck:=-ck; ds:=-ds end
        else begin for j:=0,j+1 while j<nf do
                    begin R[n+j]:=R[j]; I[n+j]:=I[j]
                    end
             end;
        k1:=j:=0; k2:=n;
L:      aa:=R[k1]+R[k2]; ab:=R[k1]-R[k2];
        ba:=I[k1]+I[k2]; bb:=I[k1]-I[k2];
```

```
        re:=ck×ba+sk×ab; im:=sk×ba-ck×ab;
        I[k2]:=im-bb; I[k1]:=im+bb;
        R[k2]:=aa-re; R[k1]:=aa+re;
        k2:=k2+1; k1:=k1+1; j:=j+1;
        if j<nf then goto L; j:=0;
        aa:=ck-(dc×ck+ds×sk);
        sk:=(ds×ck-dc×sk)+sk;
        ck:=aa; k2:=n-k1;
        if k1≤nh then goto L
end     UNSCRAMBEL;


procedure VUL(R,I,bitrev,lag,nf,index);
integer lag,nf,index; integer array bitrev; array R,I;
begin   integer i1,i2,i21,j,j1,lagnf,nf2; real t;
        lagnf:=lag×nf; nf2:=nf+nf; i2:=0;
        inarray(drum,index,R); hold(R);
rep:    i1:=bitrev[i2:nf2]×nf2; i21:=i2+nf;
        for j:=0,j+1 while j<nf do I[i1+j]:=R[i21+j];
        if i1≤i2 then goto goon;
        for j:=0,j+1 while j<nf do
        begin   t:=R[i1+j]; R[i1+j]:=R[i2+j]; R[i2+j]:=t end;
goon:   i2:=i2+nf2;
        if i2<lagnf then goto rep;
        for j1:=nf,j1+nf2 while j1<lagnf do
        for  j:= 0,j+1 while j<nf do R[j1+j]:=I[j1+j]:=0
end     VUL;


procedure OPSLAG(R1,I1,R2,I2,OR,OI,lag,nf);
value lag,nf; integer lag,nf; array R1,R2,I1,I2,OR,OI;
begin   integer k,knf0,knf1,knf2,knf3,L,k1;
        real    r10,r11,r12,r13,r20,r21,r22,r23,i10,i11,i12,i13,i20,i21,i22,i23;
        boolean         bol;
        switch LA:=L0,L1,L2,L3;         switch  M:=M0,M1,M2,M3;
        switch  N:=N0,N1,N2,N3;         switch  O:=O0,O1,O2,O3;
```

```
bol:=false;
for k:=0,k+1 while k≤ lag do
begin   knf0:=k×nf; k1:=knf0×nf; bol:=¬bol; goto LA[nf];
L3:     knf3:=knf0+3; r13:=R1[knf3]; i13:=I1[knf3];
L2:     knf2:=knf0+2; r12:=R1[knf2]; i12:=I1[knf2];
L1:     knf1:=knf0+1; r11:=R1[knf1]; i11:=I1[knf1];
L0:                   r10:=R1[knf0]; i10:=I1[knf0];
        if bol then goto M[nf] else goto N[nf];
M3:     r23:=R2[knf3]+r13; i23:=I2[knf3]+i13;
M2:     r22:=R2[knf2]+r12; i22:=I2[knf2]+i12;
M1:     r21:=R2[knf1]+r11; i21:=I2[knf1]+i11;
M0:     r20:=R2[knf0]+r10; i20:=I2[knf0]+i10; goto O[nf];
N3:     r23:=r13-R2[knf3]; i23:=i13-I2[knf3];
N2:     r22:=r12-R2[knf2]; i22:=i12-I2[knf2];
N1:     r21:=r11-R2[knf1]; i21:=i11-I2[knf1];
N0:     r20:=r10-R2[knf0]; i20:=i10-I2[knf0]; goto O[nf];
O3:     L:=k1+3; OR[L]:=r10×r23+i10×i23+OR[L];
                 OI[L]:=r10×i23-i10×r23+OI[L];
        L:=L+nf; OR[L]:=r11×r23+i11×i23+OR[L];
                 OI[L]:=r11×i23-i11×r23+OI[L];
        L:=L+nf; OR[L]:=r12×r23+i12×i23+OR[L];
                 OI[L]:=r12×i23-i12×r23+OI[L];
        L:=L+nf; OR[L]:=r13×r23+i13×i23+OR[L];
                 OI[L]:=r13×i23-i13×r23+OI[L];
        L:=L-1 ; OR[L]:=r13×r22+i13×i22+OR[L];
                 OI[L]:=r13×i22-i13×r22+OI[L];
        L:=L-1 ; OR[L]:=r13×r21+i13×i21+OR[L];
                 OI[L]:=r13×i21-i13×r21+OI[L];
        L:=L-1 ; OR[L]:=r13×r20+i13×i20+OR[L];
                 OI[L]:=r13×i20-i13×r20+OI[L];
O2:     L:=k1+2; OR[L]:=r10×r22+i10×i22+OR[L];
                 OI[L]:=r10×i22-i10×r22+OI[L];
        L:=L+nf; OR[L]:=r11×r22+i11×i22+OR[L];
                 OI[L]:=r11×i22-i11×r22+OI[L];
        L:=L+nf; OR[L]:=r12×r22+i12×i22+OR[L];
                 OI[L]:=r12×i22-i12×r22+OI[L];
        L:=L-1 ; OR[L]:=r12×r21+i12×i21+OR[L];
                 OI[L]:=r12×i21-i12×r21+OI[L];
```

```
                L:=L-1 ; OR[L]:=r12×r20+i12×i20+OR[L];
                         OI[L]:=r12×i20-i12×r20+OI[L];
        01:     L:=k1+1; OR[L]:=r10×r21+i10×i21+OR[L];
                         OI[L]:=r10×i21-i10×r21+OI[L];
                L:=L+nf; OR[L]:=r11×r21+i11×i21+OR[L];
                         OI[L]:=r11×i21-i11×r21+OI[L];
                L:=L-1 ; OR[L]:=r11×r20+i11×i20+OR[L];
                         OI[L]:=r11×i20-i11×r20+OI[L];
        00:     L:=k1  ; OR[L]:=r10×r20+i10×i20+OR[L];
                         OI[L]:=r10×i20-i10×r20+OI[L];
        end
end     OPSLAG;


procedure WINDOW(A1,A2,lag,nf,bitrev,window);
value lag,nf; integer lag,nf,window; array A1,A2; integer array bitrev;
begin   integer lagh,i,j,i1,i11,i2,i12;
        real    fake,fako,pilag,qe,qo;
        switch  F:=F1,F2,F3;
        if window<0 ∨ window>3 then goto S;
        lagh:=lag:2; pilag:=arctan(1)/lagh;
        for i:=0,i+1 while i<lagh do
        begin   i1:=i×nf; i2:=i1+i1; i11:=bitrev[i]; i12:=i11+i1;
                goto F[window];
        F1:     fake:=cos(i12×pilag)↑2; fako:=cos((i12+1)×pilag)↑2;
                goto R;
        F2:     fake:=1-i12/lag; fako:=1-(i12+1)/lag;
                goto R;
        F3:     qe:=i12/lag; qo:=(i12+1)/lag;
                fake:=if i12<lagh then (qe-1)×qe×qe×6+1 else 2×(1-qe)↑3;
                fako:=if i12<lagh then (qo-1)×qo×qo×6+1 else 2×(1-qo)↑3;
        R:      for j:=0,j+1 while j<nf do
                begin   A1[i2+j]:=fake×A1[i2+j];
                        A2[i2+j]:=fako×A2[i2+j]
                end
        end;
S:
end     WINDOW;
```

```
procedure COVSPEK(nf,lag,adres,weg,nfl,window);
value nf,lag,adres,nfl; integer nf,lag,adres,window,weg,nfl;
begin   integer m,n,f,f1,f2,i1,i1,i2,i,j,i11,i21,nn,st,lim,
                nf2,lag2,lagh,lagnf,lagnf2,sp,nf22,nop,max,ln2,k;
        boolean start;
        real t;
        array   AR,AI,BR,BI[0:(lag+1)×nf-1],
                OR,OI[0:(lag+1)×nf×nf-1];
        integer array bitrev[0:lag:2-1];
        n:=(entier(nfl/lag)+2)×lag;
        max:=adres+n×nf×2; lag2:=lag+lag; lagh:=lag:2;
        nf2:=nf×nf; lagnf:=lag×nf; ln2:=lagnf+lagnf;
        nop:=(lag+1)×nf2; nf22:=nf2+nf2;
        lagnf2:=lagnf×nf; sp:=lagnf2:2;
        for i:=0,i+1 while i<lagnf do AR[i]:=0;
        outarray(drum,adres+nfl×nf×2,AR);
        outarray(drum,adres+(nfl+lag)×nf×2,AR);
        m:=0; for k:=lag,k:2 while k>1 do m:=m+1;
        for i:=0, i+1 while i<lagh do bitrev[i]:=0;
        k:=0; nn:=1; st:=lagh:2;
L0:     lim:=k+st+st;
        for i:=k+st,i+1 while i<lim do bitrev[i]:=bitrev[i]+nn;
        k:=lim; if k<lagh then goto L0;
        k:=0; nn:=nn+nn; st:=st:2; if st>1 then goto L0;
        for i:=0, i+1 while i<nop do OR[i]:=OI[i]:=0;
        start:=false;
L1:     VUL(AR,AI,bitrev,lag,nf,adres); adres:=adres+ln2;
        REVFFT4(AR,AI,lagnf,m,nf); UNSCRAMBEL(AR,AI,lagnf,nf,false);
        if start then goto L3 else start:=true;
L2:     VUL(BR,BI,bitrev,lag,nf,adres); adres:=adres+ln2;
        REVFFT4(BR,BI,lagnf,m,nf); UNSCRAMBEL(BR,BI,lagnf,nf,false);
        OPSLAG(AR,AI,BR,BI,OR,OI,lag,nf);
        if adres<max then goto L1 else goto L4;
L3:     OPSLAG(BR,BI,AR,AI,OR,OI,lag,nf);
        if adres<max then goto L2;
L4:     UNSCRAMBEL(OR,OI,lag×nf2,nf2,true);
        for i:=0, i+1 while i<nop do OI[i]:=-OI[i];
        FFT4(OR,OI,lag×nf2,m,lag×nf2);
```

```
f:=8×lag×nf1;
for i:=0, i+1 while i<nop do
begin   OR[i]:= OR[i]/f;
        OI[i]:=-OI[i]/f
end;
outarray(drum,weg,OR); hold(OR); weg:=weg+nop+nop;
outarray(drum,weg,OI); hold(OI); weg:=weg+nop+nop;
for i:=0,i+1 while i<nf2 do OR[i]:=0.5×OR[i];
WINDOW(OR,OI,lag,nf2,bitrev,window);
for i:=1, i+2 while i<lag do
begin   ii:=i × nf2;
        for j:=0,j+1 while j<nf2 do OR[ii+j]:=OI[ii+j]:=0;
end;
REVFFT4(OR,OI,lag×nf2,m,nf2);
UNSCRAMBEL(OR,OI,lag×nf2,nf2,false);
for i:=0, i+1 while i≤lag do
begin   ii:=i × nf;
        for f1:=0, f1+1 while f1<nf do
        for f2:=f1,f2+1 while f2<nf do
        begin   i1:=(ii+f1)×nf+f2;
                i2:=(ii+f2)×nf+f1;
                OR[i1]:=(OR[i1]+OR[i2])/2;
                if i1≠i2 then
                OR[i2]:=(OI[i1]-OI[i2])/2
        end
end;
outarray(drum,weg,OR); hold(OR);
inarray(drum,weg-4×nop,OR); hold(OR);
inarray(drum,weg-2×nop,OI); hold(OI);
for j:=0,j+1 while j<nf2 do OR[lagnf2+j]:=OR[nf2+j];
for i:=0,i+1 while i<lagh do
begin   i1:=bitrev[i]×nf2×2; i11:=i1+nf2;
        i2:=i×nf2×2; i21:=i2+nf2;
        for j:=0,j+1 while j<nf2 do
        begin   if i1<i2 then
                begin   t:=OR[i1+j]; OR[i1+j]:=OR[i2+j];
                        OR[i2+j]:=t
                end;
```

```
                        OR[111+j]:=OI[12+j]; OR[121+j]:=OI[11+j]
                end
        end;
        outarray(drum,weg-2×nop,OR); hold(OR)
end     COVSPEK;


procedure GETCOV(AR,weg,nf,lag); integer weg,nf,lag; array AR;
begin   inarray(drum,weg-(lag+1)×nf×nf×2,AR); hold(AR)
end     GETCOV;


procedure GETSPEK(AR,weg,nf,lag); integer weg,nf,lag; array AR;
begin   inarray(drum,weg,AR); hold(AR)
end     GETSPEK;



procedure FOUSPEK(nf,lag,adres,weg,nfl,window,stap);
value nf,lag,adres,nfl; integer nf,lag,window,adres,weg,nfl,stap;
begin   integer lag2,sp,i1,i2,aantal,lag2nf,i,j,k,l,m,nf2,lagnf,
                lagnf2,lagh,kp,km,k1,ii,L,nop,st,lim,nn;
        real    ap,am,bp,bm,fak,r10,r11,r12,r13,r20,r21,t,pih,wf,
                r22,r23,i10,i11,i12,i13,i20,i21,i22,i23,f1,f2;
        array   A,B[0:(lag+lag+1)×nf-1],OS[0:(lag+1)×nf×nf-1],WD[0:lag-1];
        integer array  bitrev[0:lag+lag-1];
        switch  M:=M0,M1,M2,M3; switch  O:=O0,O1,O2,O3;
        lag2:=lag+lag; lagnf:=lag×nf; lagh:=lag:2; lag2nf:=lag2×nf;
        nf2:=nf×nf; lagnf2:=lag×nf2; nop:=lagnf2+nf2;
        m:=0; for k:=lag,k:2 while k>1 do m:=m+1;
        sp:=(stap+stap)×nf; aantal:=entier((nfl-lag2)/stap)+1;
        for i:=0,i+1 while i<nop do OS[i]:=0;
        pih:=2×arctan(1.0); wf:=0;
        for i:=0,i+1 while i<lag do
        begin   t:=(lag-i-0.5)/lag;
                WD[i]:= if window=0 then 1.0 else
                        if window=1 then 1-t↑2 else
                        if window=3 then 1-t else 1.0;
```

```
            wf:=wf+WD[i]↑2;
        end;
        for i:=0,i+1 while i<lag2 do bitrev[i]:=0;
        k:=0; nn:=1; st:=lag;
LO:     lim:=k+st+st;
        for i:=k+st,i+1 while i<lim do bitrev[i]:=bitrev[i]+nn;
        k:=lim; if k<lag2 then goto LO;
        k:=0; nn:=nn+nn; st:=st:2; if st≥1 then goto LO;
        l:=0;
rep:    inarray(drum,adres,A); hold(A); adres:=adres+sp;
        inarray(drum,adres,B); hold(B); adres:=adres+sp;
        l:=l+1; if l=aantal then
        begin for i:=0,i+1 while i<lag2nf do B[i]:=0 end;
        for i:=0,i+1 while i<lag2 do
        begin   i2:=i×nf; i1:=bitrev[i]; i1:=i1×nf;
                if i<i1 then goto out;
                f1:=WD[if  i<lag then  i else lag2- i-1];
                f2:=WD[if i1<lag then i1 else lag2-i1-1];
                for j:=0,j+1 while j<nf do
                begin   t:=A[i1]×f2; A[i1]:=A[i2]×f1; A[i2]:=t;
                        t:=B[i1]×f2; B[i1]:=B[i2]×f1; B[i2]:=t;
                        i1:=i1+1; i2:=i2+1
                end;
        out:
        end;
        REVFFT4(A,B,lag2nf,m,nf);
        for j:=0,j+1 while j<nf do
        begin   A[lag2nf+j]:=A[j];
                B[lag2nf+j]:=B[j]
        end;
        for kp:=0,kp+nf while kp≤lagnf do
        begin   km:=lag2nf-kp; k1:=kp×nf; goto M[nf];
        M3:     ap:=A[kp+3]; am:=A[km+3]; bp:=B[kp+3]; bm:=B[km+3];
                r13:=ap+am;  i13:=bp-bm;  r23:=bp+bm;  i23:=am-ap;
        M2:     ap:=A[kp+2]; am:=A[km+2]; bp:=B[kp+2]; bm:=B[km+2];
                r12:=ap+am;  i12:=bp-bm;  r22:=bp+bm;  i22:=am-ap;
        M1:     ap:=A[kp+1]; am:=A[km+1]; bp:=B[kp+1]; bm:=B[km+1];
                r11:=ap+am;  i11:=bp-bm;  r21:=bp+bm;  i21:=am-ap;
```

```
        M0:     ap:=A[kp] ; am:=A[km] ; bp:=B[kp] ; bm:=B[km] ;
                r10:=ap+am; i10:=bp-bm; r20:=bp+bm; i20:=am-ap;
                goto O[nf];
        O3:     L:=k1+3; OS[L]:=r10×r13+i10×i13+r20×r23+i20×i23+OS[L];
                L:=L+nf; OS[L]:=r11×r13+i11×i13+r21×r23+i21×i23+OS[L];
                L:=L+nf; OS[L]:=r12×r13+i12×i13+r22×r23+i22×i23+OS[L];
                L:=L+nf; OS[L]:=r13×r13+i13×i13+r23×r23+i23×i23+OS[L];
                L:=L-1 ; OS[L]:=r12×i13-i12×r13+r22×i23-i22×r23+OS[L];
                L:=L-1 ; OS[L]:=r11×i13-i11×r13+r21×i23-i21×r23+OS[L];
                L:=L-1 ; OS[L]:=r10×i13-i10×r13+r20×i23-i20×r23+OS[L];
        O2:     L:=k1+2; OS[L]:=r10×r12+i10×i12+r20×r22+i20×i22+OS[L];
                L:=L+nf; OS[L]:=r11×r12+i11×i12+r21×r22+i21×i22+OS[L];
                L:=L+nf; OS[L]:=r12×r12+i12×i12+r22×r22+i22×i22+OS[L];
                L:=L-1 ; OS[L]:=r11×i12-i11×r12+r21×i22-i21×r22+OS[L];
                L:=L-1 ; OS[L]:=r10×i12-i10×r12+r20×i22-i21×r22+OS[L];
        O1:     L:=k1+1; OS[L]:=r10×r11+i10×i11+r20×r21+i20×i21+OS[L];
                L:=L+nf; OS[L]:=r11×r11+i11×i11+r21×r21+i21×i21+OS[L];
                L:=L-1 ; OS[L]:=r10×i11-i10×r11+r20×i21-i20×r21+OS[L];
        O0:     L:=k1  ; OS[L]:=r10×r10+i10×i10+r20×r20+i20×i20+OS[L];
        end;
        l:=l+1; if l<aantal then goto rep;
        fak:=1/(wf×aantal×8);
        for i:=0,i+1 while i<nop do OS[i]:=OS[i]×fak;
        outarray(drum,weg,OS); hold(OS)
end     FOUSPEK;


procedure SMOOTH(S,lag,nf,a1,a2,a3);
value nf; integer lag,nf; array S; real a1,a2,a3;
begin   integer i,j,lagnf2,nf2; real x1,x2,x3; boolean cospek;
        nf2:=nf×nf; lagnf2:=lag×nf2;
        for j:=0,j+1 while j<nf2 do
        begin   cospek:=j:nf>j-(j:nf)×nf;
                x2:=S[nf2+j]; x3:=S[j]; if cospek then x2:=-x2;
                for i:=j,i+nf2 while i<lagnf2 do
                begin   x1:=x2; x2:=x3; x3:=S[i+nf2];
                        S[i]:=a1×x1+a2×x2+a3×x3
                end;
```

```
                if ¬cospek then S[lagnf2+j]:=a1×x2+a2×x3+a3×x2
        end
end     SMOOTH;
```

```
procedure REALTWIN(F1,F2,n);
value n; integer n; array F1,F2;
begin   integer k,nk;
        real    r1,r2,i1,i2,p;
        p:=2/n;
        F1[0]:=F1[0]×p; F2[0]:=F2[0]×p;
        p:=1/n; k:=1; nk:=n-k;
L:      r1:=F1[k]; r2:=F1[nk];
        i1:=F2[k]; i2:=F2[nk];
        F1[nk]:=(i1-i2)×p;
        F2[nk]:=(r2-r1)×p;
        F1[k] :=(r1+r2)×p;
        F2[k] :=(i1+i2)×p;
        k:=k+1; nk:=nk-1;
        if k≤nk then goto L
end     REALTWIN;
```

```
procedure COEFTWIN(F1,F2,n);
value n; integer n; array F1,F2;
begin   integer nk,k;
        real    r1,r2,i1,i2;
        F1[0]:=F1[0]×0.5; F2[0]:=F2[0]×0.5;
        k:=1; nk:=n-k;
L:      r1:=F1[k] ; r2:=F2[k];
        i1:=F1[nk]; i2:=F2[nk];
        F1[nk]:=(r1-i2)×0.5;
        F2[k] :=(r2-i1)×0.5;
        F1[k] :=(r1+i2)×0.5;
        F2[nk]:=(r2+i1)×0.5;
        k:=k+1; nk:=nk-1;
        if k<nk then goto L;
```

```
        if k=nk then
        begin   F1[k]:=F1[k]×0.5;
                F2[k]:=F2[k]×0.5
        end
end     COEFTWIN;


procedure REALTRAN(A,B,n,evaluate);
value n,evaluate; integer n; boolean evaluate; array A,B;

begin   integer k,nk,nh; real aa,ab,ba,bb,re,im,ck,sk,dc,ds,r;
        nh:=n:2; r:=4×arctan(1.0)/n;
        ds:= sin(r); r:= -(2×sin(0.5×r))↑2; dc:= -0.5×r; ck:=1.0; sk:=0;
        if evaluate then begin ck:= -1.0; dc:= -dc end
                    else begin A[n]:= A[0]; B[n]:= B[0] end;
        for k:= 0 step 1 until nh do
        begin   nk:= n-k;
                aa:= A[k]+ A[nk]; ab:= A[k]- A[nk];
                ba:= B[k]+ B[nk]; bb:= B[k]- B[nk];
                re:=ck×ba+sk×ab; im:=sk×ba-ck×ab;
                B[nk]:= im-bb; B[k]:= im+bb; A[nk]:= aa-re; A[k]:= aa+re;
                dc:= r×ck+dc; ck:= ck+dc; ds:= r×sk+ds;  sk:= sk+ds
        end
end     REALTRAN;


procedure REVERSEBINARY(A,B,m); value m; integer m; array A,B;
begin   integer j,jj,k,lim,jk,n2,n4,n8,nn;
        real    t;
        integer array  C[0:m];
        C[0]:=nn:=1; jj:=0;
        for j:=1 step 1 until m do C[j]:=nn:=nn+nn;
        if m>1 then n4:=C[m-2]; if m>2 then n8:=C[m-3]; n2:=C[m-1];
        lim:=n2-1; nn:=nn-1; m:=m-4;
        for j:=1 step 1 until lim do
        begin   jk:=jj+n2;
                t:=A[j]; A[j]:=A[jk]; A[jk]:=t;
                t:=B[j]; B[j]:=B[jk]; B[jk]:=t;
```

```
                j:=j+1;
                if jj≥n4 then
                begin   jj:=jj-n4; if jj≥n8 then
                        begin   jj:=jj-n8; k:=m;
                        L:      if C[k]≤jj then
                                begin   jj:=jj-C[k]; k:=k-1;
                                        goto L
                                end;
                                jj:=C[k]+jj
                        end
                        else    jj:=jj+n8
                end
                else    jj:=jj+n4;
                if jj>j then
                begin   k:=nn-j; jk:=nn-jj;
                        t:=A[j]; A[j]:=A[jj]; A[jj]:=t;
                        t:=B[j]; B[j]:=B[jj]; B[jj]:=t;
                        t:=A[k]; A[k]:=A[jk]; A[jk]:=t;
                        t:=B[k]; B[k]:=B[jk]; B[jk]:=t
                end
        end
end REVERSEBINARY;
```