stichting
mathematisch
centrum

MC

AFDELING TOEGEPASTE WISKUNDE TN 89/77 DECEMBER

G.J.M. LAAN

EEN ALGOL 68-OPERATOR fft

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1970): 42-04,42A16

Een ALGOL 68-operator fft

door

G.J.M. Laan

SAMENVATTING

Dit geschrift bevat teksten in ALGOL 68 van de Fast Fourier Transform met als basis het getal 2, voor zowel reële als complexe input. Een toelichting wordt gegeven alsook suggesties voor uitbreidingen en verfijningen van de gegeven teksten.

TREFWOORDEN: *discrete fourier transformatie, fast fourier transform* ALGOL 68

INHOUDSOPGAVE

1. INLEIDING

Sinds de publicatie van "An algorithm for the machine calculation of complex Fourier series" in 1965 [1], is een enorme hoeveelheid literatuur over de Fast Fourier Transform verschenen. Hoogtepunt daarin is wellicht het "standaardwerk" over FFT van E.O. BRIGHAM [2], waarin opgenomen een bibliografie van ruim 200 titels. Het is niet mijn bedoeling daar veel nieuws aan toe te voegen. Wel wil ik verslag doen van mijn bevindingen tijdens omzwervingen in de literatuur over FFT en een programma in ALGOL 68 aanbieden.

De aantrekkingskracht van de algoritme FFT is m.i. gelegen in de toepasbaarheid op vele problemen van fysische oorsprong, waarvan o.a. Brigham getuige is, in de betrekkelijke eenvoud en efficiëntie, in de mogelijkheid tot verfijningen, uitbreidingen en generalisaties, in de relatie met andere wiskundige problemen en algoritmen, kortom in het feit dat er zowel voor wiskundigen als niet-wiskundigen "iets aan te beleven valt".

Wat betreft mijn eigen motivatie ten aanzien van het onderwerp FFT kan ik kort zijn: ik had een operator fft nodig voor een onderzoek naar het gebruik van verzwakkingsfactoren bij de berekening van Fouriercoëfficiënten, een onderzoek waarvan ik in [3] kond doe.

2. OVER DE EFFICIËNTIE VAN DE ALGORITME

De algoritme FFT wordt efficiënt genoemd. COOLEY en TUKEY zelf claimen in [1] een bovengrens van 2nlogn operaties voor de transformatie van een n-vector (in tegenstelling tot n^2 operaties voor een rechttoe-rechtaan berekening), waarbij onder "operatie" een complexe vermenigvuldiging gevolgd door een complexe optelling wordt verstaan en "log" de $^2\log$ is.

Anderszijds heeft J. MORGENSTERN in [4] bewezen dat het aantal benodigde operaties, mits men bij de berekening gebruik maakt van complexe getallen z met $|z| \leq 1$, groter is dan $\frac{n}{2} \log n$.

Een ondergrens voor het algemene geval is, voorzover mij bekend, niet bekend.

3. DE ALGORITME FFT

Ik zal hier volstaan met te zeggen wat FFT doet en hoe het werkt, zonder de achtergrond te beschrijven. Wie hierover toch meer wil weten kan deze in tal van publicaties vinden, zie BRIGHAM [2] en de literatuurlijst aldaar.

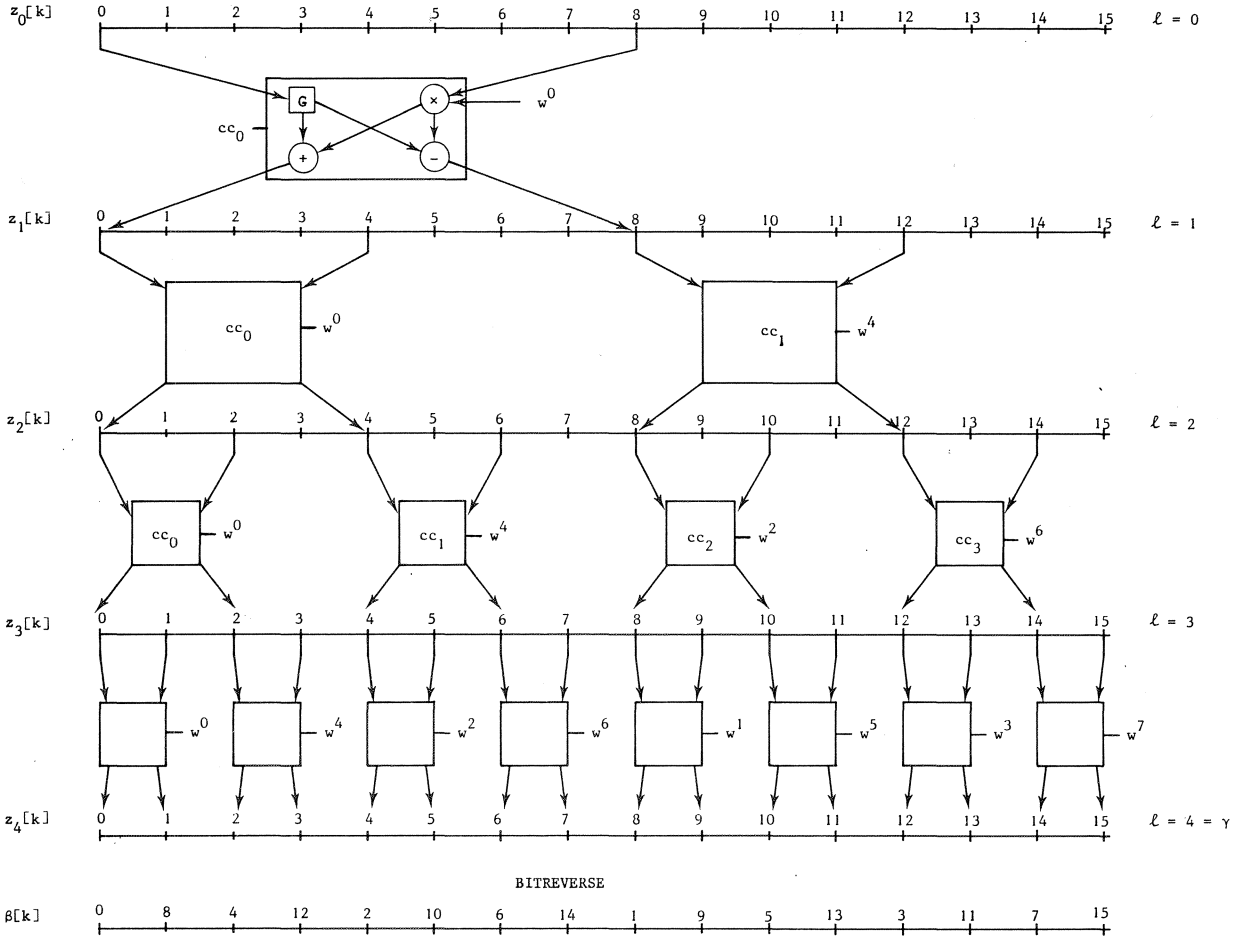
FFT berekent de discrete Fouriergetransformeerde van een (complexe) kolomvector van lengte n $f = (f_0, f_1, \ldots, f_{n-1})^T$. De Discrete Fourier Transformatie is als volgt gedefinieerd.
Zij A een n × n-matrix met $A[j,k] = e^{\frac{-2\pi ijk}{n}}$, $0 \leq j, k < n-1$, dan heet de vector Af, waarvan de j-de component gegeven wordt door

$$\beta_j = \sum_{k=0}^{n-1} f_k \, e^{\frac{-2\pi ijk}{n}}, \qquad 0 \leq j \leq n-1,$$

de discrete Fouriergetransformeerde van f.

Indien nu $n = 2^\gamma$ (γ geheel) dan kan de matrix A op slimme wijze in γ matrices gefactoriseerd worden, zodanig dat het aantal operaties dat nodig is voor het uitvoeren van DFT geminimaliseerd wordt.

De werking van de algoritme FFT kan waarschijnlijk het best aan de hand van onderstaande figuur geïllustreerd worden.
(Deze versie is de zogenaamde Cooley-Tukey variant van de algoritme FFT; de voorstellingswijze is afkomstig van G.D. BERGLAND [5]).



Figuur 1.

FFT schema voor n = 16 (γ=4); $w = e^{\frac{-2\pi i}{n}}$.

Toelichting bij figuur 1:

1. Op de bovenste lijn (ℓ=0) staat het array $z_0[0:15]$ met $z_0[k] = f_k$ voor $k = 0,1,\ldots,15$ als $f = (f_0,f_1,\ldots,f_{15})^T$ de te transformeren vector is.

2. De doos aangeduid met cc (complex calculation) berekent uit de waarden van z_0 het array z_1, als volgt:

$$\begin{cases} z_1[k] = z_0[k] + w^0 \, z_0[k+8] \\ z_1[k+8] = z_0[k] - w^0 \, z_0[k+8] \end{cases} \qquad \text{voor } 0 \le k \le 7$$

Deze bewerkingen zijn in de doos cc symbolisch aangegeven, waarin G een symbool voor tijdelijke opslag is.

3. Het array z_2 komt op zijn beurt als volgt tot stand:

$$\begin{cases} z_2[k] = z_1[k] + w^0 \, z_1[k+4] \\ z_2[k+4] = z_1[k] - w^0 \, z_1[k+4] \end{cases} \qquad \text{voor } 0 \le k \le 3$$

$$\begin{cases} z_2[k] = z_1[k] + w^4 \, z_1[k+4] \\ z_2[k+4] = z_1[k] - w^4 \, z_1[k+4] \end{cases} \qquad \text{voor } 8 \le k \le 11$$

4. In het algemeen wordt het array z_ℓ bepaald door het paar vergelijkingen:

$$\begin{cases} z_\ell[k] = z_{\ell-1}[k] + w^p \, z_{\ell-1}[k+n/2^\ell] \\ z_\ell[k+n/2^\ell] = z_{\ell-1}[k] - w^p \, z_{\ell-1}[k+n/2^\ell] \end{cases}$$

voor geschikte waarden van p en k (zie de volgende opmerkingen).

5. Het aantal dozen cc, en dus het aantal verschillende waarden van p, dat nodig is voor de berekening van de waarden van het array z_ℓ is gelijk aan $2^{\ell-1}$.
Indien we de dozen cc van links naar rechts nummeren van 0 tot $2^{\ell-1}-1$ en deze aangegeven met cc_j, dan blijkt dat de exponent p in de term w^p bepaald wordt door:
(a) schrijf de index j in binaire vorm met $\gamma-1$ bits
(b) draai de volgorde van de bits om
(c) dit "bit-reversed" getal is p.

In figuur 1 kan men de exponent p voor de vierde doos cc_3 voor de berekening van z_3 of z_4 als volgt bepalen:
(a) j=3; in binaire vorm met $\gamma-1 = 3$ bits: 011
(b) volgorde omdraaien levert: 110
(c) dit binaire getal is gelijk aan 6=p.

6. De doos cc_j, voor $0 \leq j \leq 2^{\ell-1}-1$, berekent de waarden $z_\ell[k]$ en $z_\ell[k+n/2^\ell]$ voor die waarden van k, waarvoor $2j\ n/(2^\ell) \leq k \leq (2j+1)\ n/2^\ell-1$. (Vb.: ℓ=4; j=3; n=16; doos cc_3 berekent $z_3[k]$ en $z_3[k+1]$ voor k=6).

Op grond van figuur 1 en de toelichting daarbij kom ik tot het volgende stukje programma in ALGOL 68:

- de arrays $z_1, z_2, \ldots, z_\gamma$ worden in die volgorde berekend in de loop:
  for ℓ to γ do ... od
- voor iedere waarde ℓ is het aantal verschillende dozen cc_j gelijk aan $2^{\ell-1}$; dit levert:

```
for ℓ to γ
do for j from 0 to 2^(ℓ-1)-1 do ... od
od
```

- voor iedere waarde van j doorloopt k in totaal $n/2^\ell$ verschillende waarden en daarna slaat k in totaal $n/2^\ell$ waarden over. Naar aanleiding hiervan krijgen we:

```
for ℓ to γ
do k:=0;
   for j from 0 to 2^(ℓ-1)-1
   do    to   n/2^ℓ
         do   ...

              k+:=1

         od;
         k+:= n/2^ℓ
   od
od
```

- op de plaats van de stippeltjes komt de eigenlijke berekening van het array z_ℓ door de doos cc_j. Daar een eenmaal gebruikte waarde $z_{\ell-1}[k]$ verder nutteloos is geworden kunnen we de waarde van $z_{\ell-1}[k]$ vervangen door die van $z_\ell[k]$, dwz. we kunnen volstaan met één array z, dat geïnitialiseerd is met de waarden van de componenten van de te transformeren vector f.

Iedere doos cc_j maakt gebruik van een andere factor $w^p = e^{-2\pi ip/n}$, waarin p = bitrev (j,γ-1) (zie opmerking 5 van deze paragraaf).

Dit geeft aanleiding tot:

```
for ℓ to γ
do   k:=p:=0;
     for j from 0 to 2^(ℓ-1)-1
     do p:= bitrev (j,γ-1); compl wp = exp(-2πip/n);
        to  n/2^ℓ
        do  compl g = wp*z[k+n/2^ℓ];
            z[k+n/2^ℓ] := z[k] -g;
            z[k] +:= g;
            k+:=1
        od
        k+:= n/2^ℓ
     od
od
```

- Het enige probleem dat nu nog om een oplossing vraagt is de procedure bitrev. Deze heeft een tweeledig doel:

  1. Het bepalen van de exponent p.
  2. Het blijkt dat de elementen van het laatst berekende array in "bitreversed order" staan. Om de DFT van f te krijgen moeten de elementen via de procedure bitrev in normale volgorde worden geplaatst.

  Dit probleem wordt in de volgende paragraaf behandeld.

4. OVER DE BITREVERSED VOLGORDE

Ik zal beginnen met het invoeren van enige notaties, die het afleiden van een aantal regels vergemakkelijken. Met behulp van deze regels komt dan een procedure tot stand die het gestelde probleem oplost.

(a) $n = 2^\gamma$, γ geheel en positief.

(b) Voor een getal d, $0 \le d \le 2^\gamma - 1$, stelt $[d_{\gamma-1}d_{\gamma-2}\ldots d_1 d_0]$ de binaire representatie van het getal d voor.

(Dus $d = d_{\gamma-1}2^{\gamma-1} + d_{\gamma-2}2^{\gamma-2} + \ldots + d_1 2 + d_0$).

(c) In verkorte vorm wordt de binaire representatie van d aangegeven met $\underline{bin}_\gamma d$.

(d) Onder $\underline{rev}_\gamma d$ wordt verstaan het getal
$d_0 2^{\gamma-1} + d_1 2^{\gamma-2} + \ldots + d_{\gamma-2}2^2 + d_{\gamma-1}$.

(e) In overeenstemming met (b) is dan
$\underline{bin}_\gamma \underline{rev}_\gamma d = [d_0 d_1 \ldots d_{\gamma-2} d_{\gamma-1}]$.

(f) In een binaire representatie stelt $\bar{d}_i$ de negatie van d_i voor, $0 \leq i \leq \gamma-1$. Anders gezegd:
$\bar{d}_i = \underline{if}\ d_i = 0\ \underline{then}\ 1\ \underline{else}\ 0\ \underline{fi}$ voor alle i, $0 \leq i \leq \gamma-1$.

(g) $[\bar{d}_{\gamma-1}\bar{d}_{\gamma-2}\ldots\bar{d}_1\bar{d}_0]$ wordt genoteerd als $[\overline{d_{\gamma-1}d_{\gamma-2}\ldots d_1 d_0}]$.

Met behulp van deze notaties laten de volgende regels zich gemakkelijk verifiëren:

REGEL 1: Voor d *oneven* geldt: $\underline{rev}_\gamma d = \underline{rev}_\gamma(d-1) + 2^{\gamma-1}$.

REGEL 2: $\underline{rev}_\gamma(2^\gamma-1-d) = 2^\gamma-1-\underline{rev}_\gamma d$.

REGEL 3: Voor d *even* geldt: $\underline{rev}_\gamma d = \underline{rev}_{\gamma-1}(d/2)$.

Ad 1. Als d oneven is dan geldt: $\underline{bin}_\gamma d = [d_{\gamma-1}\ldots d_1 1]$
Voor d-1 geldt dan: $\underline{bin}_\gamma(d-1) = [d_{\gamma-1}\ldots d_1 0]$
en dus
$\underline{bin}\ \underline{rev}_\gamma d = [1 d_1 \ldots d_{\gamma-1}]$ en
$\underline{bin}_\gamma \underline{rev}_\gamma(d-1) = [0 d_1 \ldots d_{\gamma-1}]$.
Hieruit volgt:
$\underline{bin}_\gamma \underline{rev}_\gamma d = \underline{bin}_\gamma \underline{rev}_\gamma(d-1) + \underline{bin}_\gamma(2^{\gamma-1})$ en dus regel 1.

Ad 2. $\underline{bin}_\gamma(2^\gamma-1) = [1\ 1\ \ldots 1\ 1]$
$\underline{bin}_\gamma d = [d_{\gamma-1}d_{\gamma-2}\ldots d_1 d_0]$ en dus is
$\underline{bin}_\gamma(2^\gamma-1-d) = [\overline{d_{\gamma-1}d_{\gamma-2}\ldots d_1 d_0}]$
d.w.z.
$\underline{bin}_\gamma \underline{rev}_\gamma(2^\gamma-1-d) = [\overline{d_0 d_1 \ldots d_{\gamma-2}d_{\gamma-1}}]$. (i)
Anderszijds is
$\underline{bin}_\gamma(2^\gamma-1) = [1\ 1\ \ldots 1\ 1]$
$\underline{bin}_\gamma \underline{rev}_\gamma d = [d_0 d_1 \ldots d_{\gamma-2}d_{\gamma-1}]$

en dus

$$\underline{\text{bin}}_\gamma(2^\gamma-1-\underline{\text{rev}}_\gamma d) = [\overline{d_0 d_1 \ldots d_{\gamma-2} d_{\gamma-1}}]. \qquad \text{(ii)}$$

Uit (i) en (ii) volgt regel 2.

Ad 3. Als d even is dan geldt: $\underline{\text{bin}}_\gamma d = [d_{\gamma-1}\ldots d_2 d_1 0]$
en $\underline{\text{bin}}_{\gamma-1}(\frac{d}{2}) = [d_{\gamma-1}\ldots d_2 d_1]$.
Dus: $\underline{\text{bin}}_\gamma \underline{\text{rev}}_\gamma d = [0 d_1 \ldots d_{\gamma-2} d_{\gamma-1}]$
en: $\text{bin}_{\gamma-1}\text{rev}_{\gamma-1}(\frac{d}{2}) = [d_1 \ldots d_{\gamma-2} d_{\gamma-1}]$,
waaruit regel 3 volgt.

Ik verlaat nu even bovenstaande drie regels en beschouw het volgende probleem:

Stel dat voor een of andere d, $0 \leq d < 2^\gamma-1$, het getal $\underline{\text{rev}}_\gamma d$ gegeven is. De vraag is hoe op grond hiervan het getal $\underline{\text{rev}}_\gamma(d+1)$ berekend kan worden. Welnu, als $\underline{\text{bin}}_\gamma d = [d_{\gamma-1} d_{\gamma-2}\ldots d_1 d_0]$ dan wordt $\underline{\text{bin}}_\gamma(d+1)$ verkregen door binair bij $\underline{\text{bin}}_\gamma d$ 1 op te tellen.
Op overeenkomstige wijze krijgen we $\underline{\text{rev}}_\gamma(d+1)$ door bij $\underline{\text{bin}}_\gamma \underline{\text{rev}}_\gamma d$ binair 1 op te tellen, *maar dan vanaf de linkerkant.*
De laatste bewerking wordt uitgevoerd door de volgende procedure:

```
proc increv = (ref int p, int γ) void:
     (int b := 2^(γ-1);
      if b > 1 then while b ≤ p do p -:= b; b overab 2 od fi;
      p +:= b
     );
```

Toelichting:

- p = $\underline{\text{rev}}_\gamma d$ bij aanroep; p = $\underline{\text{rev}}_\gamma(d+1)$ na afloop.
- de combinatie b := $2^{\gamma-1}$, b overab 2 zorgt er voor dat b (een beginstuk van) de waarden 2^k (k = γ-1, γ-2,...,1,0) doorloopt.
- zij $\underline{\text{bin}}_\gamma p = (p_{\gamma-1} p_{\gamma-2}\ldots p_k \ldots p_1 p_0)$.
  Als b ≤ p dan is kennelijk $p_k = 1$. p -:= b zorgt er voor dat p_k gelijk aan nul wordt. Zodra b > p dan is blijkbaar $p_k = 0$. Tenslotte zorgt p +:= b ervoor dat p_k dan gelijk aan 1 wordt, waarmee de binaire optelling

van links af voltooid is.

- de controle b > 1 maakt het mogelijk dat de procedure met $\gamma = 1$ aangeroepen wordt.

De procedure increv kan gebruikt worden om de elementen van een array z[0:2^{γ}-1] in bit-reversed volgorde te zetten, of omgekeerd - wegens p = $\underline{rev}_{\gamma}$ d desda $\underline{rev}_{\gamma}$ p = d - elementen die in bit reversed volgorde staan in normale volgorde te plaatsen.

De volgende operator ontwar voert deze bewerking uit, daarbij gebruik makend van de procedure increv en de regels 1, 2 en 3 uit deze paragraaf.

```
op ontwar = (ref [ ] compl z) ref [ ] compl:
      (k := p := 0;
       to n over 4 - 1
       do k +:= 1; z[k] ich z[p+2^(γ-1)];
          k +:= 1; increv (p,γ-1);
          if k < p then z[k] ich z[p]
                   elif p < k then z[n-1-k] ich z[n-1-p]
          fi
       od;
       k+:= 1; z[k] ich z[p+2^(γ-1)];
       z
      );
```

Toelichting:

- k is de lopende index; de waarde van p is het resultaat van een aanroep van increv. In het begin is k = 0 en p = $\underline{rev}_{\gamma}(0) = 0$.
- de dyadische operator ich verwisselt de waarden van de operanden:

```
op ich = (ref compl a,b) void:
      (compl ab := a; a := b; b := ab);
```

- als k *oneven* is dan worden z[k] en z[p+$2^{\gamma-1}$] verwisseld, waar p = $\underline{rev}_{\gamma}(k-1)$, in overeenstemming met regel 1.
- als k *even* is dan wordt increv aangeroepen met $(p,\gamma-1)$ als parameters,

waardoor p de waarde $\underline{rev}_\gamma k$ krijgt, immers:
bij aanroep van increv is $p = \underline{rev}_\gamma(k-2) = \underline{rev}_{\gamma-1}(\frac{k}{2}-1)$, vlg. regel 3, na afloop is $p = \underline{rev}_{\gamma-1}(k/2) = \underline{rev}_\gamma k$, vlg. regel 3.

- z[k] en z[p] worden alleen dan verwisseld als k < p om dubbele verwisselingen (effect: geen verwisseling) te voorkomen.
  Als k = p hoeft er uiteraard niet verwisseld te worden en als k > p (dan is n-1-k<n-1-p) dan worden z[n-1-k] en z[n-1-p] verwisseld, in overstemming met regel 2.
- deze laatste regel impliceert ook dat, om alle elementen van een array z[0:n-1] in normale volgorde te plaatsen, k slechts de waarden 0 tot $\frac{n}{2} - 1$ hoeft te doorlopen.
- omdat k tweemaal per cyclus wordt opgehoogd moet de loop $\frac{n}{4} - 1$ maal worden uitgevoerd, aan het eind heeft k de waarde $\frac{n}{2} - 2$, tot slot volgt nog eenmaal de regel voor $k = \frac{n}{2} - 1$ (= oneven).

In de volgende paragraaf geef ik, zonder commentaar, de volledige tekst van de operator fft.

5. DE OPERATOR FFT VOOR COMPLEXE INPUT

```
op fft = (ref [ ] compl z) ref [ ] compl:
    (int n = upb z + 1; int hn := n, k := 0, p;
     while hn > 1 do hn overab 2; k +:= 1 od; int gamma = k;
     proc increv = (ref int p, int m) void:
           (int b := 2**(m-1);
            if b > 1 then while b < = p do p -:= b; b overab 2 od fi;
            p +:= b
           );
     op ich = (ref compl a,b) void:
           (compl ab := a; a := b; b := ab);
```

```
op ontwar = (ref [ ] compl z) ref [ ] compl:
     (k := p := 0; int no2 = n over 2;
      to n over 4 - 1
      do k +:= 1; z[k] ich z[p+no2];
         k +:= 1; increv (p, gamma - 1);
         if k < p then z[k] ich z[p]
                  elif p < k then z[n-1-k] ich z[n-1-p]
         fi
      od;
      k +:= 1; z[k] ich z[p+no2];
      z
     );

   real tweepion = 2*pi/n;
   proc exp = (real arg) compl: cos(arg) i sin(arg);
   prio ich = 1;

   for ℓ to gamma
   do int twee1  = 2**ℓ;
      int no2ℓ   = n over twee1; k := p := 0;
      for j to twee1 over 2
      do if j > 0 then increv (p, gamma - 1) fi;
         compl wp = exp (-tweepi*p/n);
         to no2ℓ
         do compl g = wp*z[k+no2ℓ];
            z[k+no2ℓ] := z[k] - g;
            z[k] +:= g;
            k +:= 1
         od
         k =:= no2ℓ
      od
   od;
   ontwar z
  );
```

Een voorbeeld van het gebruik van de operator fft wordt gegeven door het

volgende programma:

```
([0:3] compl z;
 for k from 0 to 3 do z[k] := k od;
 fft z;
 for k from 0 to 3 do print ((z[k], newline)) od
)
```

Het resultaat is:

```
  6
- 2 + 2i
- 2
- 2 - 2i
```

6. OPMERKINGEN

(a) Er zijn varianten van de algoritme FFT bekend, sommige met de input-data in bit-reversed en output-data in gewone volgorde, of omgekeerd. In sommige algoritmen komen de machten p van w in normale volgorde aan bod.

(b) Behalve als basis het getal 2, komen als basis voor FFT de getallen 4, 8, 16 in de literatuur voor, en ook combinaties hiervan, '2+4' bijvoorbeeld, alsook zijn er algoritmen bekend voor $n = r_1 r_2 \ldots r_m$, r_i geheel, of zelfs voor willekeurige n. Als referentie hiervoor moge BRIGHAM [2] dienen en de verwijzingen aldaar.

(c) Efficiëntere algoritmen kan men krijgen indien de te transformeren vector f bepaalde symmetrie-eigenschappen bezit, of uitsluitend reële componenten heeft. Voor deze speciale gevallen komen binnenkort aan de RUG operatoren in ALGOL 68 beschikbaar (zie VAN DER LAAN [6]). Zie voor het reële geval de volgende paragrafen.

(d) Een efficiëntere algoritme verkrijgt men door gebruik te maken van symmetrieën van de factoren $w^p = e^{-2\pi ip/n}$. Deze truc staat bekend onder de naam "twiddle factors" en wordt o.a. in BRIGHAM [2] genoemd.

(e) OLIVER [7] heeft methodes aangegeven om de factoren w^p stabiel, vrij nauwkeurig en efficiënter dan door de aanroep van de standaardprocedures sin en cos te berekenen.

(f) De inverse van de discrete Fouriertransformatie wordt gegeven door

$$f_k = \frac{1}{n} \sum_{j=0}^{n-1} \beta_j \, e^{+2\pi ijk/n} , \qquad 0 \le k \le n-1.$$

Dit is, op het + teken in de e-macht en de factor $\frac{1}{n}$ na, dezelfde transformatie als DFT zelf. Met behulp van een enigszins aangepaste algoritme FFT kan men dus zowel DFT als zijn inverse berekenen.

7. FFT VOOR REËLE INPUT

Zij $r_k \in \mathbb{R}$, $k = 0,1,\ldots,n-1$
en
$$\beta_j = \sum_{k=0}^{n-1} r_k \, e^{-2\pi ijk/n}, \quad j = 0,1,\ldots,n-1.$$

Het is natuurlijk mogelijk de β_j direct te berekenen met behulp van de operator fft uit de vorige paragraaf. Het kan echter efficiënter op de volgende wijze:

Schrijf

$$\beta_j = \sum_{k=0}^{\frac{n}{2}-1} r_{2k} e^{-2\pi ijk/(n/2)} + \sum_{k=0}^{\frac{n}{2}-1} r_{2k+1} \, e^{-2\pi ijk/(n/2)} \, e^{-2\pi ij/n}$$

$$= \delta_j + e^{-2\pi ij/n} \, \varepsilon_j .$$

Merk op dat δ en ε discrete fouriertransformaties zijn van twee rijtjes reële getallen van lengte $n/2$.
Stel

$$\alpha_j = \sum_{k=0}^{\frac{n}{2}-1} (r_{2k} + i r_{2k+1}) e^{-2\pi ijk/(n/2)}$$

d.w.z. α is de DFT van een complexe vector van lengte $\frac{n}{2}$, die is opgebouwd

uit de gegeven n reële getallen r_k. Het is mogelijk δ en ε, en daarmee β, uit te drukken in termen van α als volgt:

$$\alpha_j = \delta_j + i\varepsilon_j \quad \text{dus} \quad \overline{\alpha_j} = \overline{\delta_j} - i\overline{\varepsilon_j}.$$

Omdat δ en ε de DFT van reële rijtjes zijn hebben de δ_j en ε_j de eigenschap:

$$\overline{\delta_{n/2-j}} = \delta_j \quad \text{en} \quad \overline{\varepsilon_{n/2-j}} = \varepsilon_j \quad \text{(zie bijv. LAAN [3], §3).}$$

We hebben nu:

$$\alpha_j = \delta_j + i\varepsilon_j$$

$$\overline{\alpha_{n/2-j}} = \delta_j - i\varepsilon_j.$$

Hieruit volgt:

$$\delta_j = \frac{1}{2}(\alpha_j + \overline{\alpha_{n/2-j}})$$

$$\varepsilon_j = -\frac{i}{2}(\alpha_j - \overline{\alpha_{n/2-j}})$$

en aldus volgt uit $\beta_j = \delta_j + e^{-2\pi ij/n}\,\varepsilon_j$ dat

$$\beta_j = \frac{1}{2}[\alpha_j + \overline{\alpha_{n/2-j}} - ie^{-2\pi ij/n}\,(\alpha_j - \overline{\alpha_{n/2-j}})]$$

voor $j = 0,1,\ldots,n-1$.

Met behulp van $\beta_{n-j} = \overline{\beta_j}$ en $\alpha_{n/2} = \alpha_0$ kunnen we ook schrijven:

$$\beta_j = \frac{1}{2}[\alpha_j + \overline{\alpha_{n/2-j}} - ie^{-2\pi ij/n}(\alpha_j - \overline{\alpha_{n/2-j}})]$$

en

$$\beta_{n-j} = \overline{\beta_j}, \qquad \text{voor } j = 1,2,\ldots,\frac{n}{2}-1,$$

$$\beta_0 = \underline{re}\ \alpha_0 + \underline{im}\ \alpha_0,$$

$$\beta_{n/2} = \underline{re}\ \alpha_0 - \underline{im}\ \alpha_0.$$

Op deze regels is de operator fft uit de volgende paragraaf gebaseerd.

8. DE OPERATOR FFT VOOR REËLE INPUT

```
op fft = (ref [ ] real r) ref [ ] compl:
   (int n = upb r + 1; int hn := n, k := 0, p;
    while hn > 1 do hn overab 2; k +:= 1 od; int gamma = k;
    [0:n-1] compl z;
    int n2 = n over 2;
    for k from 0 to n2-1 do z[k] := r[2*k] i r[2*k+1] od;

    ref [ ] compl alfa := z[0:n2-1 at 0];
    fft alfa;

    real rea = re alfa [0], ima = im alfa [0];
    (z[0] := rea + ima, z[n2]:= rea - ima);

    real tweepion = 2*pi/n;
    int n4 = n over 4;

    for k to n4 - 1
    do real arg = k * tweepion; int n2k = n2 - k;
       compl wk = sin(arg) i cos(arg), ak = alfa [k],
                  an2k = alfa [n2k];
       (z[k] := (ak + conj an2k + wk * (conj an2k - ak))/2,
        z[n2k] := (an2k + conj ak + conj wk * (conj ak - an2k))/2
       );
       (z[n-k] := conj z[k], z[n-n2k] := conj z[n2k]
       )
    od;

    z[n-n4] := alfa [n4];
    z[n4] := conj alfa [n4];
    z
    );
```

Een voorbeeld van het gebruik van deze operator fft wordt gegeven door het volgende programma:

```
([0:3] real r, ref [] compl z;
 for k from 0 to 3 do r[k] : = k od;
 z: = fft r;
 for k from 0 to 3 do print ((z[k], newline)) od
)
```

Het resultaat is hetzelfde als in het voorbeeld op bladzijde 12, maar wòrdt in minder rekentijd bereikt.

LITERATUUR

[1] COOLEY, J.W. & J.W. TUKEY, [1965], *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19, p. 297-301.

[2] BRIGHAM, E.O., [1974], *The Fast Fourir Transform*, Prentice-Hall, New Jersey.

[3] LAAN, G.J.M., [1977], *Fouriercoëfficiënten en verzwakkingsfactoren*, Rapport TN 88 (1977), Mathematisch Centrum, Amsterdam.

[4] MORGENSTERN, J., [1973], *Note on a lower bound of the linear complexity of the FFT*, JACM, 20, 2, p. 305-306.

[5] BERGLAND, G.D., [1968], *A Fast Fourier Transform algorithm for the real-valued series*, CACM, 11, 10, p. 703-710.

[6] LAAN, C.G. van der, [1976], *Fouriertransformatie*, in: Colloquium Numerieke Programmatuur, J.C.P. BUS (red.), MC Syllabus 29.1b, Mathematisch Centrum, Amsterdam.

[7] OLIVER, J., [1975], *Stable methods for evaluating the points* cos $1\pi/n$, J. Inst. Math. Applics., 16, p. 247-257.