INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONE: 224031
CABLE: CENTRATOM - TELEX 460392-1

SECOND SCHOOL ON ADVANCED TECHNIQUES
IN COMPUTATIONAL PHYSICS
(18 January - 12 February 1988)

SMR.282/29

FAST FOURIER TRANSFORM

A.Nobile

SISSA, Trieste, Italy

Fourier Transforms & Discrete Fourier Transform
(Bingham, Prentice-Hall, 1974)

a) Fourier series

$$f(x) = \sum_{k=-\infty}^{\infty} F_k \, e^{i k \frac{2\pi}{L} x}$$

$$F_k = \frac{1}{L} \int_0^L f(x) \, e^{-i k \frac{2\pi}{L} x} \, dx$$

b) Fourier Transform

If $$F(k) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x) \, e^{-ikx} \, dx$$

then $$f(x) = \int_{-\infty}^{\infty} F(k) \, e^{ikx} \, dk$$

c) Discrete Fourier Transform

If $f_l \in C$, $l \in N$, $0 \le l < N$

Then $$f_l = \sum_{k=0}^{N-1} F_k \, e^{i k l \cdot \frac{2\pi}{N}}$$

$$F_k = \frac{1}{N} \sum_{k=0}^{N-1} f_l \, e^{i k l \frac{2\pi}{N}}$$

Properties of FT's

	FT	FS
$f(x)+g(x)$	$F(k)+G(k)$	F_k+G_k
$f(-x)$	$F(-k)$	F_{-k}
$f(ax)$	$\frac{1}{\lvert a\rvert}F(\frac{k}{a})$	...
$f(x-x_0)$	$e^{-ikx_0}F(k)$	$e^{-ikx_0}F_k$
$\frac{df}{dx}$	$ikF(k)$	$i\frac{2\pi}{L}kF_k$
$f*g=\int f(y)g(x-y)dy$	$2\pi F(k)G(k)$	LF_kG_k
fg	$F*G$	$F*G=\sum_{n'}F_{n'}G_{n-n'}$

DFT

$f \to f \bmod N$

f_e+g_e	F_k+G_k
f_{-e}	F_{-k}
f_{e-e_0}	$e^{ike_0\frac{2\pi}{N}}F_k$
$f*g=\sum_{l=0}^{N-1}f_l g_{e-l}$	NF_kG_k

All follow trivially from definitions + orthogonality relations

$$\sum_{k=0}^{N-1} e^{ik\ell\frac{2\pi}{N}} = N\delta_{\ell,0} \quad ; \quad \int_0^L dx\, e^{ik\frac{2\pi}{L}x} = L\delta_{k0},\ k\in\mathbb{N}$$

$$\int_{-\infty}^{\infty} e^{ikx}\,dx = 2\pi\delta(k)$$

Other properties

	FT	FS
$f(x)$ real	$F(k)=-F^*(-k)$	$F_k=F^*_{-k}$
$f(x)$ real even	$F(k)$ real even	F_k real even
$f(x)$ real odd	$F(k)$ imaginary odd	F_k imaginary odd

For DFT

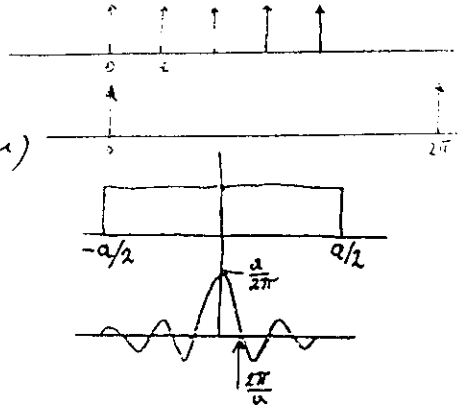
f_l real	$F_k=F^*_{N-k}$
f_l real, $f_{N-l}=f_l$	F_k real, $F_{N-k}=F_k$
f_l real, $f_{N-l}=-f_l$	F_k imaginary, $F_{N-k}=-F_k$

some other useful relations

- $$\sum_{n=-\infty}^{\infty} e^{-ikna} = \frac{2\pi}{a}\sum_{n=-\infty}^{\infty}\delta(k-\frac{2\pi}{a}n)$$

from which, if

- $$f(x)=\sum_{n=-\infty}^{\infty}\delta(x-na)$$
$$F(k)=\frac{1}{a}\sum_{n=-\infty}^{\infty}\delta(k-\frac{2\pi}{a}n)$$



- $$f(x)=\theta(x+a/2)-\theta(x-a/2)$$
$$F(k)=\frac{a}{2\pi}\,\frac{\sin\frac{ka}{2}}{\frac{ka}{2}}$$

Let $f(x)$ be periodic with period L, and

$$C^N(x) = \sum_{j=0}^{N-1} \delta\left(x - \frac{L}{N} j\right)$$

The F.S. coefficients of $f(x) C^N(x)$ are

$$\tilde{F}_k = \frac{1}{L} \int_0^L dx\, e^{-ik\frac{2\pi}{L}x} f(x)\, C^N(x) =$$

$$\frac{1}{L} \sum_{n=0}^{N-1} f\left(\frac{L}{N} n\right) e^{-ik\frac{2\pi}{N}n} =$$

$$\boxed{\frac{N}{L} F_k^D}$$

where

$$F_n^D = DFT\left(f_\ell \,|\, \ell = 0..N-1;\ f_\ell = f\left(\frac{L}{N}\ell\right)\right)$$

But

$$\tilde{F}_k = F * C^N = \sum_{k'} F_{k'}\, C^N_{k-k'} =$$

$$\sum_{k'} F_{k'} \left(\frac{N}{L} \sum_{\ell=-\infty}^{\infty} \delta_{k-k',\ \ell N}\right) =$$

$$\boxed{\frac{N}{L} \sum_{\ell=-\infty}^{\infty} F_{k+\ell N}}$$

Thus

$$\boxed{F_k^D = \sum_{\ell=-\infty}^{\infty} F_{k+\ell N}}$$

The DFT as an approximation to FT

$f(x)$ — $F(k)$

$$f_{(1)}(x) = f(x)\left[\theta\left(x+\tfrac{L}{2}\right) - \theta\left(x-\tfrac{L}{2}\right)\right] \qquad F_{(1)} = F * \frac{L}{2\pi} \frac{\sin\frac{kL}{2}}{\frac{kL}{2}}$$

$$f_{(2)}(x) = f_1 * \sum_{-\infty}^{\infty} \delta(x - \ell L) \qquad F_{(2)} = 2\pi F_{(1)} \cdot \frac{1}{L} \sum_{-\infty}^{\infty} \delta\left(x - \frac{2\pi}{L}\ell\right) =$$

$$= F(k) * \frac{\sin\frac{kL}{2}}{\frac{kL}{2}} \sum_{-\infty}^{\infty} \delta\left(x - \frac{2\pi}{L}\ell\right)$$

$f_{(2)}(x)$ is periodic, and its F.S. is

$$F_{(2)\ell} = \left(F * \frac{\sin\frac{kL}{2}}{\frac{kL}{2}} \right)\Bigg|_{k=\frac{2\pi}{L}\ell}$$

from previous results

$$f_{(3)}(x) = f_2 \cdot \sum_{0}^{N-1} \delta\left(x - \frac{L}{N} l\right) \qquad F_{(3)k} = \sum_{\ell=-\infty}^{\infty} F_{(2)k+\ell N}$$

» both a long range mixing ($\Delta k \approx \frac{N}{L}$) and a short range mixing ($\Delta k \approx \frac{1}{L}$)

Aliasing

» Attempts to replace the F.S coefficients by the DFT coefficients

$$f(x) \overset{?}{\simeq} \sum_{k=0}^{N-1} F_k^D\, e^{-ik\frac{2\pi}{L}x}$$

True at $x = \frac{L}{N} j$ (equality!);

Elsewhere problems:

Trivial case:

$f(x) = \cos(x)$ $\qquad F_k = \frac{1}{2}\left(\delta_{k,1} + \delta_{k,-1}\right)$

$$F_1^D = \sum_{l=-\infty}^{\infty} F_{1+lN} = \frac{1}{2}$$

$$F_{N-1}^D = \sum_{l=-\infty}^{\infty} F_{N-1+lN} = \frac{1}{2}$$

$$F_d^D = 0 \quad d \neq 1, N-1$$

$$\cos(x) = \frac{1}{2}\left(e^{ikx} + e^{ik(N-1)x}\right) \quad ?$$

- In general: unaliased inverse Transform

$$f(x) = \sum_{-N/2+1}^{N/2} F_k^D e^{ik\frac{2\pi}{L}x} \qquad N \text{ even}$$

$$f(x) = \sum_{-(N-1)/2}^{(N-1)/2} F_k^D e^{ik\frac{2\pi}{L}x} \qquad N \text{ odd}$$

Serious case

$$1) = f(x) \cdot C^N(x) \longrightarrow F_{(1)} = \sum_{l=-\infty}^{\infty} F_{k+lN}$$

If N sufficiently high, only one term (at most) in the sum → periodic repetition of F_k

If N too small, F_k and F_{k+N} important for some k → distorted F_k

⇓

Discrete approximation is not a good one for the continuous problem

1 The Fast Fourier Transform

The FFT is a fast algorithm for computing DFT.

Let's consider

$$F_k = \sum_{l=0}^{N-1} f_l e^{\imath kl\frac{2\pi}{N}} \tag{1}$$

In FFT literature it is customary to neglect the normalization factor

From now on we will omit the ^{D} superscript

Equation 1 requires N^2 complex products and $N(N-1)$ complex sums. Let's now split the sequence f_l (assuming N even)

$$x_l = f_{2l} \ , \ y_l = f_{2l+1} \ , \ l = 0 \ldots \frac{N}{2} - 1$$

$$F_k = \sum_{l=0}^{N/2-1} x_l e^{\imath k\, 2l\frac{2\pi}{N}} + y_l e^{\imath k\, 2l\frac{2\pi}{N}} e^{\imath k\frac{2\pi}{N}}$$

If $k < \frac{N}{2}$

$$F_k = \sum x_l e^{\imath kl\frac{2\pi}{N/2}} + e^{\imath kl\frac{2\pi}{N/2}} \sum y_l e^{\imath kl\frac{2\pi}{N/2}}$$

if $k > \frac{N}{2}$,

$$F_k = \sum x_l e^{\imath(k-N/2)l\frac{2\pi}{N/2}} + (-1)e^{\imath(k-N/2)l\frac{2\pi}{N/2}} \sum y_l e^{\imath(k-N/2)l\frac{2\pi}{N/2}}$$

Thus

$$F_k = X_k + e^{\imath kl\frac{2\pi}{N/2}} Y_k \quad k < N/2$$

$$F_k = X_{k-N/2} - e^{\imath(k-N/2)l\frac{2\pi}{N/2}} Y_{k-N/2} \quad k \geq N/2$$

Operation Count:

X_k	$\left(\frac{N}{2}\right)^2$ *products*	$\frac{N}{2}\left(\frac{N}{2}-1\right)$ *sums*
Y_k	$\left(\frac{N}{2}\right)^2$ *products*	$\frac{N}{2}\left(\frac{N}{2}-1\right)$ *sums*
F_k	N *products*	N *sums*

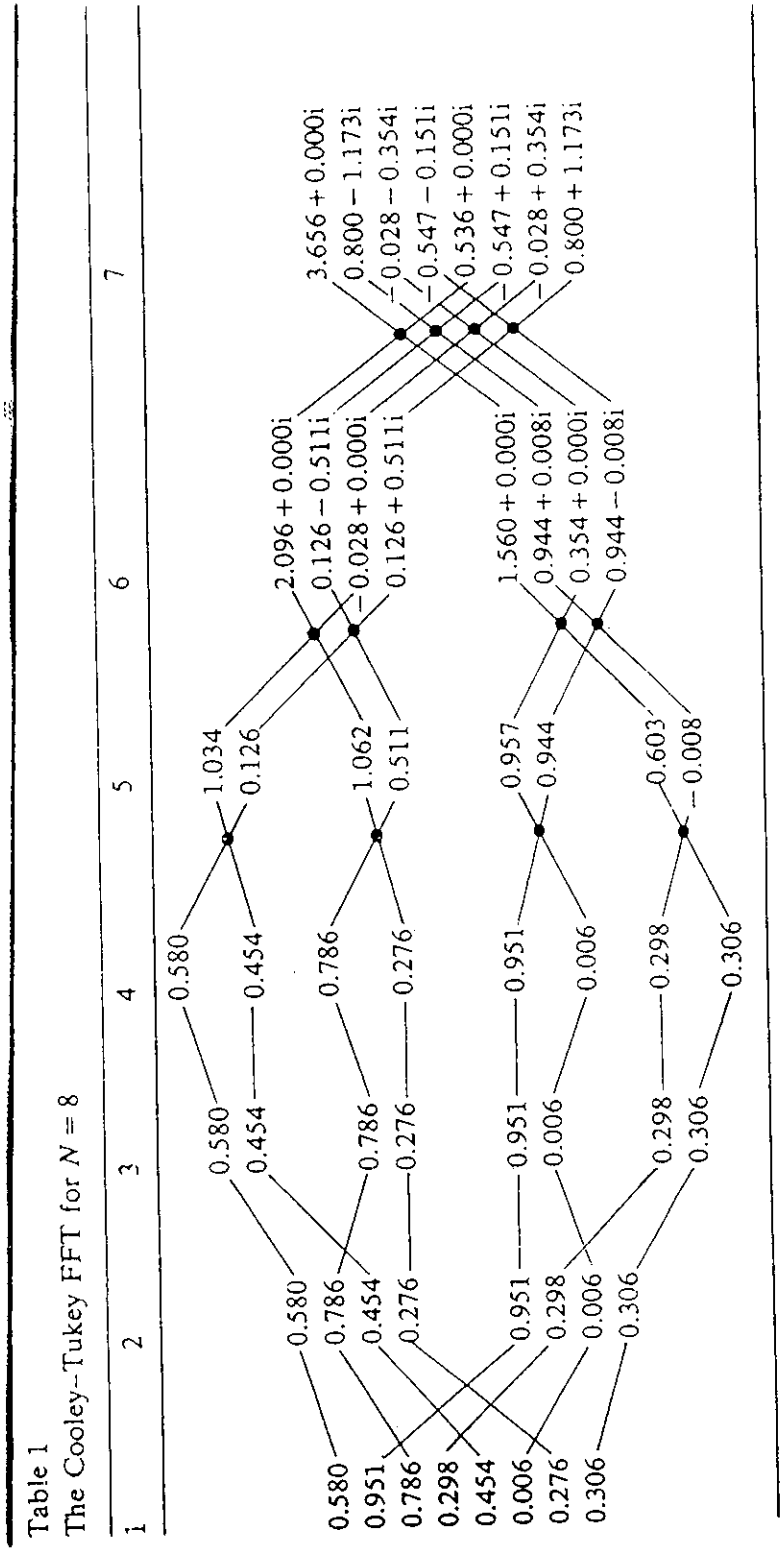
TOTAL$\sim \frac{N^2}{2} + N$ for N large, almost halved

If $N = 2^M$, the same method can be applied to the evaluation of X's and Y's.

DFT's	N^2	$N^2/2$	$N^2/4$	$N^2/8$	...	$N/2^M = N$
Combination	0	N	$2N$	$3N$	...	$MN = N \ln_2 N$

An example in the following picture:

Table 1
The Cooley-Tukey FFT for $N = 8$



The following programme describes the process

```
      SUBROUTINE CTSRT(C,M,WORK)
* Sorts the complex array C of size 2**M
* for Cooley-Tukey FFT
* WORK : complex array of the same size as C, scratch
       COMPLEX C(*),WORK(*)
       N=2**M
       DO 100 L=1,M
          MS=2**(L-1)
          LS=N/(2*NS)
*  MS*LS*2=N
          CALL CTSRT1(LS,MS,C,WORK)
          CALL VCOPY(C,WORK,N)
* VCOPY  copies WORK into C
100    CONTINUE
       END
       SUBROUTINE CTSRT1(LS,MS,C,C1)
       COMPLEX C(0:1,0:LS-1,0:MS-1),
     $         C1(0:LS-1,0:1,0:MS-1)
       DO 200 J=0,MS-1
       DO 200 I=0,LS-1
          C1(I,0,J)=C(0,I,J)
          C1(I,1,J)=C(1,I,J)
200    CONTINUE
       END
```

The key loop is loop 200 : a sequence of length $2 * LS$ ((0,I)(1,I)(0,I+1)...) is split in 2 –consecutive– sequences of length LS ((I,0)(I+1,0)... (I,1)(I+1,1)..).

The combination step is described by the following programme:

```
       SUBROUTINE CTCMBN(M,C)
* combine phase for a Cooley-Tukey FFT
* of the complex array C of size 2**M.
* C must have been passed through CTSRT.
       COMPLEX C(*)
       N=2**M
       DO 100 L=M,1,-1
         MS=2**(L-1)
         LS=N/(2*NS)
*  MS*LS*2=N
         CALL CTCMB1(LS,MS,C)
100    CONTINUE
       END
       SUBROUTINE CTCMB1(LS,MS,C)
       COMPLEX C(0:LS-1,0:1,0:MS-1)
       PARAMETER(PI=3.14...)
       COMPLEX WYK

       ANGLE=PI/REAL(LS)
       DO 200 J=0,MS-1
       DO 200 I=0,LS-1
         WYK=EXP(CMPLX(0.,-ANGLE*REAL(I)))*C(I,1,J)
         C(I,0,J)=C(I,0,J)+WYK
         C(I,1,J)=C(I,0,J)-WYK
200    CONTINUE
       END
```

2 Where do we go from here

- Variants of FFT algorithms;
- Mixed-radix algorithms;
- Special cases:
  - Real, even, odd transforms;
  - multiple transforms;
  - multidimensional transforms;
- If N is prime?
- Alternatives to FFT.

3 Variants of the FFT

3.1 The Gentleman-Sande algorithm

There are two ways of performing an unnormalized fast inverse DFT:

1. Use an algorithm for direct DFT, but with all the phase factors changed of sign(replace ANGLE by -ANGLE in loop 200);
2. Perform the algorithm for direct DFT backwards

If you apply both of the above, you obtain a new direct FFT.

Let's take the Cooley-Tukey algorithm, and reverse it: this imply to apply first the –inverted– combine phase, then the inverted sort phase.


```
      SUBROUTINE GSCMBN(M,C)
* combine phase for a Gentleman-Sande
* (reversed Cooley-Tukey) FFT of the
* complex array C of size 2**M.
* C must then be passed through GSSRT
      COMPLEX C(*)
      N=2**M
      DO 100 L=1,M
        MS=2**(L-1)
        LS=N/(2*NS)
*  MS*LS*2=N
        CALL GSCMB1(LS,MS,C)
100   CONTINUE
      END
      SUBROUTINE GSCMB1(LS,MS,C)
      COMPLEX C(0:LS-1,0:1,0:MS-1)
      PARAMETER(PI=3.14...)
      COMPLEX WYK

      ANGLE=PI/REAL(LS)
      DO 200 J=0,MS-1
      DO 200 I=0,LS-1
        WYK=EXP(CMPLX(0.,ANGLE*REAL(I)))*
     $        (C(I,0,J)-C(I,1,J))
        C(I,0,J)=C(I,0,J)+C(I,1,J)
        C(I,1,J)=WYK
200   CONTINUE
      END
```

Where the transformation performed by loop 200 is the inverse of the one performed by the equivalent loop in CTCMB1.

The inversion of the sort part is trivial, amounting simply to reversing the direction of loop 100 in CTSRT.

This new transform is known as Gentleman-Sande, and it shares with the Cooley-Tukey one the property of being "in place".

3.2 Avoiding the sort phase: the Stockham algorithm.

Let's consider the Cooley-Tukey diagram. The splitting phase is composed of two actions:

- split the original sequence in two, separating even and odd elements;
- put the 2 subsequences one after the other in a single array.

The first part is at the core of the FFT algorithm. *The second is arbitrary.*

PROBLEM 1: modify the second part in such a way that column 4 is identical to column 1, that is the sort part becomes a do-nothing.

PROBLEM 2: if problem 1 has a solution, what happens to the combination phase?

Problem 2 is simple: any stage of the combine phase must be the "inverse" of the correspondig stage of the sort

phase: the sequences in column 5 must be the Fourier transforms of those in column 3, those in column 6 of those in column 2, etc.

The solution to problem 1 is illustrated in the figure. It is equivalent to replacing routine CTSRT1 by the following:

```
      SUBROUTINE STSRT1(LS,MS,C,CH)
      COMPLEX C(0:1,0:LS-1,0:MS-1),
     *        CH(0:LS-1,0:1,0:MS-1)
      DO 200 J=0,MS-1
      DO 200 I=0,LS-1
        CH(I,0,J)=C(0,I,J)
        CH(I,1,J)=C(1,I,J)
200   CONTINUE
      END
```

Of course, since the result of the sort phase is identical to the original sequence, the sort phase does not need to be performed.

A major difference comes in the combine phase. As you can see from the picture, the result of a combine step does not occupy the same place as the original elements that entered the combination. This implies that the combine phase requires a work array, differently from Cooley-Tukey and Gentleman-Sande algorithms. The algorithm thus is

Table 2
The Stockham Variant FFT for $N = 8$


```
      SUBROUTINE STCMBN(C,M,WORK)
* performs the DFT of the complex array C
* of size 2**M using a variant of the Stockham
* autosort algorithm
* WORK is a complex array of the same size of C
      COMPLEX C(*),WORK(*)
      N=2**M
      DO 100 L=M,1,-1
        MS=2**(L-1)
        LS=N/(2*NS)
*  MS*LS*2=N
        CALL STCMB1(LS,MS,C,WORK)
        CALL VCOPY(C,WORK,N)
100   CONTINUE
      END
      SUBROUTINE STCMB1(LS,MS,C,CH)
      COMPLEX C(0:LS-1,0:MS-1,0:1),
     $        CH(0:LS-1,0:1,0:MS-1)
      PARAMETER(PI=3.14...)
      COMPLEX WYK

      ANGLE=PI/REAL(LS)
      DO 200 J=0,MS-1
      DO 200 I=0,LS-1
        WYK=EXP(CMPLX(0.,-ANGLE*REAL(I)))*C(I,1,J)
        C(I,0,J)=C(I,0,J)+WYK
        C(I,1,J)=C(I,0,J)-WYK
200   CONTINUE
      END
```

Like for the Cooley-Tukey algorithm, a variant of this one —in fact the original Stockham algorithm— can be obtained by inverting the algorithms and changing the sign of phase factors. The two variants have similar properties.

4 Mixed radix algorithms

If N is not a power of 2, an approach similar to the above can still be followed. For instance, if $N = 3N'$, we can split the sequence f_l in three as follows

$$x_l = f_{3l} \quad y_l = f_{3l+1} \quad z_l = f_{3l+2} \quad 0 \le l < N'$$

The resulting reduction in the operation count is slightly inferior to the one obtained in the radix-2 case.

General formulation for mixed-radix FFT

Let $N = r_1 r_2 \cdots r_m$

Let's define

$$\mu_i = \prod_{k=1}^{i} r_k \qquad \lambda_i = \prod_{k=i+1}^{m} r_k$$

Any integer $< N$ can be written in 2 different ways in a mixed-basis representa

$$k = k_{m-1} r_1 r_2 \cdots r_{m-1} + k_{m-2} r_1 r_2 \cdots r_{m-2} + \cdots k_1 r_1 + k_0 =$$

$$\sum_{j=0}^{m-1} k_j \mu_j \qquad 0 \le k_j < r_{j+1}$$

or

$$n = n_{m-1} r_2 \cdots r_m + n_{m-2} r_3 \cdots r_m + \cdots n_1 r_m + n_0$$

$$= \sum_{j=0}^{m-1} n_j \lambda_{m-j} \qquad 0 \le n_j < r_{m-j}$$

Let's now rewrite the basic DFT equation

$$X(k) = \sum_{n=0}^{N-1} x_0(n) W^{nk} \qquad W = e^{-i\frac{2\pi}{N}}$$

as

$$X(k_{m-1}, \cdots k_0) = \sum_{n_0} \sum_{n_1} \cdots \sum_{n_{m-1}} x_0(n_{m-1}, \cdots, n_0) W^{nk}$$

$$W^{kn} = W^{k\sum n_j \lambda_{m-j}} = W^{k\, n_{m-1}\lambda_1} W^{k\sum_{j=0}^{m-2} n_j \lambda_{m-j}}$$

the first factor becomes, expanding k

$$W^{(\sum_{j=0}^{m-1} k_j \mu_j)\, n_{m-1}\lambda_1} = W^{k_0\, n_{m-1}\lambda_1}$$

because $\mu_j \lambda_1 = r_1 r_2 \cdots r_m \cdot$ something, and $W^N = 1$. We singled out one term of the development of n and one of k

Hence

$$X(k_{m-1},\dots k_0) = \sum_{n_0}\cdots\left[\sum_{n_{m-1}} x_0(n_{m-1},\dots n_0)\, W^{k_0 n_{m-1}\lambda_1}\right]\cdot W^{k\sum_{j=0}^{m-2} n_j \lambda_{m-j}}$$

! we define

$$x_1(k_0, n_{m-2},\dots n_0) = \sum_{n_{m-1}} x_0(n_{m-1},\dots n_0)\cdot W^{k_0 n_{m-1}\underbrace{\lambda_1}_{N/r_1}}$$

our equation becomes

$$X(k_{m-1}\cdots k_0) = \sum_{n_0}'\cdots\left[\sum_{n_{m-2}} x_1(k_0, n_{m-2}\cdots, n_0)\, W^{k_0 n_{m-2}\lambda_2} W^{k_1 \underbrace{\mu_1 \lambda_2}_{=\frac{N}{r_2}} n_{m-2}}\right] W^{k\sum_{j=0} n_j \lambda_{m-j}}$$

from which it is natural to define x_2 analogous to x_1.

$$x_2(k_0 k_1 n_{m-3}\cdots n_0) = \sum_{n_{m-2}}\left(\left(x_1(k_0 n_{m-2}\cdots n_0) W^{k_0 n_{m-2}\lambda_2}\right)\cdot W^{k_1 n_{m-2}\underbrace{\mu_1\lambda_2}_{N/r_2}}\right)$$

In general

$$x_i(k_0\cdots k_{i-1} n_{m-i-1}\cdots n_0) = \left[\sum_{n_{m-i}}\left(x_{i-1}(k_0\cdots k_{i-2} n_{m-i}\cdots n_0)\, W^{\sum_{j=0}^{i-2} k_j \mu_j n_{m-i}\lambda_i}\right)\cdot W^{k_{i-1} n_{m-i}\left(\frac{N}{r_i}\right)}\right]$$

and $X(k_{m-1}\cdots k_0) = x_m(k_0\cdots k_{m-1})$

- $W_N^{\frac{N}{r_i}} = W_{r_i}$: The expression in square brackets is a DFT of order r_i
- Relation with routine CTCMB

$$X(\underbrace{k_0\cdots k_{i-2}}_{I}\ \underbrace{n_{m-i}}_{\phi a 1}\ \underbrace{n_{m-i-1}\cdots n_0}_{J})$$

$\sum_{j=0}^{i-2} k_j\mu_j \to I \qquad \lambda_i \to MS$

- The mapping

$$k = (k_{m-1},\dots k_0) \longrightarrow k' = (k_0,\dots k_{m-1})$$

is called "bit reversal" (proper if $r_1 = r_2 \cdots = r_m = 2$)

References:

Bingham

Nussbaumer

→ Singleton, J. comput. Phy. 52, 1983 (3 papers)

This makes it straightforward to sketch a mixed-radix algorithm:

```
        SUBROUTINE CTFFT(C,IFAC,M,WORK)
* mixed-radix Cooley-Tukey FFT of the complex array
C of size
* N=IFAC(1)*IFAC(2)*...IFAC(M)
* WORK is a complex work array of size N

        N=1
        DO 10 I=1,M
          N=N*IFAC(I)
10      CONTINUE
* sort phase
        MS=N
        DO 100 L=1,M
          MS=MS/IFAC(L)
          LS=N/(IFAC(L)*MS)
          CALL CTSRT1(LS,IFAC(L),MS,C,WORK)
          CALL VCOPY(C,WORK,N)
100     CONTINUE
* combine phase
        LS=N
        DO 1000 L=1,M
          LS=LS/IFAC(L)
          MS=N/(IFAC(L)*LS)
          CALL CTCMB1(LS,IFAC(L),MS,C)
1000    CONTINUE
        END
        SUBROUTINE CTSRT1(LS,NS,MS,C,CH)
```



```
        COMPLEX C(0:NS-1,0:LS-1,0:MS-1),
     $          C1(0:LS-1,0:NS-1,0:MS-1)
        DO 200 J=0,MS-1
        DO 200 I=0,LS-1
        DO 200 K=0,NS-1
          C1(I,K,J)=C(K,I,J)
200     CONTINUE
        END
        SUBROUTINE CTCMB1(LS,NS,MS,C)
        COMPLEX C(0:LS-1,0:NS-1,0:MS-1)
        PARAMETER (MAXRAD=...)
*       largest radix to be treated
        COMPLEX C1(MAXRAD)
        PARAMETER(PI=3.14...)
        COMPLEX WYK

        ANGLE=PI/REAL(LS)
        DO 200 J=0,MS-1
        DO 200 I=0,LS-1
          DO 110 K=0,NS-1
          C1(K)=EXP(CMPLX(0.,-ANGLE*REAL(I)))*C(I,K,J)
110     CONTINUE
        GOTO(9999,120,130,...)NS
9999      CONTINUE
*           illegal radix
            STOP
120       CONTINUE
*           radix 2
            C(I,0,J)=C1(0)+C1(1)
            C(I,1,J)=C1(0)-C1(1)
```

```
      GOTO 200
130   CONTINUE
*       radix 3
        C(I,0,J)=C1(0)+C1(1)+C(2)
        C(I,1,J)=C1(0)+
     $             C1(1)*(-0.5,SQRT(3.)/2.)+
     $             C1(2)*(-0.5,-SQRT(3.)/2.)
        C(I,2,J)=C1(0)+
     $             C1(1)*(-0.5,-SQRT(3.)/2.)+
     $             C1(2)*(-0.5,SQRT(3.)/2.)
      GOTO 200
140   CONTINUE
            .....
200   CONTINUE
      END
```

The above sketched program is totally inefficient: but since the algorithm is the right one, optimizing is only a problem of program transformation.

4.1 Implementation issues related to mixed radix

Choice of the factorization: Is it better to factorize 8 as $2 \times 2 \times 2$, or as 4×2?

- A program written to perform a DFT with a composed radix (e.g. 4, 8, 6) in a single step can exploit some symmetries in the coefficients to reduce the number of operations. A monolithic radix-4 transform has 25% less operations than the FFT done factorizing 4 to 2×2;

- The advantage is rapidly decreasing with larger radixes;
- The complexity of the program increases with the square of the radix(compare above radix 2 and radix 3);
- in some cases, the machine architecture can favour low-radix routines (memory traffic can be lower, because intermediate results must be stored back in memory,etc.)

5 Special cases

5.1 FFT of real data

If the original data are real, $f_l = f_l^*$, the transformed data have the known symmetry $F_k = F_{N-k}^*$.

Treating them as complex wastes 50% of the computations. There are three approaches to this problem:

- If one has to transform simultaneously an even number of sequences $f_{i,l}$, $1 \leq i \leq 2L$, let's define $z_{i,l} = f_{2i,l} + \iota f_{2i+1,l}$; then

$$F_{2i,k} = \frac{Z_k + Z_{N-k}}{2}$$

$$F_{2i+1,k} = \frac{Z_k - Z_{N-k}}{2}$$

- if there is one sequence, but N is even, let's define $z_l = f_{2l} + \iota f_{2l+1}$; then directly

$$F_k = (Z_k(1-\iota) + Z_{N-k}^*(1+\iota))e^{-\iota\frac{\pi k}{N}}$$

(see for instance Brigham,The Fast Fourier Transform, Prentice- Hall,1974, or Swarztrauber, in Parallel Computations, G.Rodrigue ed., Academic Press, 1982)

The post-processing requires $\sim N$ operations: in theory it is negligible, in practice it costs you some 10-15% of the total computer time.

- Smart way (Bergland, Comm. ACM, 1968): From flow diagram for CT : *the result of each combine step, columns 5-7, is composed of many subsequences* each presenting the simmetry $F_k = F^*_{LS-k}$ *because each one is the DFT of the corresponding* real *sequence in columns 1-3.*

  Thus: Do not compute, nor store, the elements of each subsequence with $I > LS/2$. When they are needed by further steps, use $C(I-L/2, K, J)^*$ instead.

  Very efficient algorithm!

5.2 FFT of real even or odd sequences

If f_l is real and even (real and odd) with period 2N, $F_k = F_{N-k}$ and F_k is real ($F_k = -F_{N-k}$ and F_k is imaginary).

A general real transform would waste 50% of the computations.

Algorithm: Transform f in a sequence with period N and no symmetries:

$$d_l = \frac{1}{2}(f_l - f_{N-l}) + \sin(l\frac{\pi}{N})(f_l + f_{N-l}) \qquad (2)$$

If, as customary, we consider the *sine*-DFT

$$f_l = \sum_{k=1}^{N-1} c_k \sin(kn\frac{\pi}{N}) \qquad (3)$$

replacing eq.(3) in eq.(2)

$$d_n = c_1 + \sum_{k=1}^{(N-1)/2} [(c_{2k+1} - c_{2k-1})\cos(kn\frac{2\pi}{N}) + c_{2k}\sin(kn\frac{2\pi}{N})] \qquad (4)$$

Thanks to the symmetries, we can write:

$$d_l = \sum_{0}^{N-1} 2\Re D_k \cos(\frac{2\pi}{N}kn) - 2\Im D_k \sin(\frac{2\pi}{N}kn)$$

Finally:

$$\begin{aligned} c_1 &= 2D_0 \\ c_{2k} &= 2\Im D_k \\ c_{2k+1} - c_{2k-1} &= 2\Re D_k \end{aligned}$$

Analogous formulas can be obtained for the even case, see Swarztrauber, in Parallel Computations, G Rodrigue ed. Academic Press 1982.

5.3 Multidimensional transforms

Conside 2 dimensions (trivially generalized):

- Simple approach: first transform along first dimension (N2 transforms of order N1), then along second one (N1 transforms of order N2). FFT routines often allow non-contiguous input sequences to allow this operation without actually transposing the matrix;

– True multidimensional elementary transforms: example for 2 dimension, square matrix, radix 2 only.

```
      SUBROUTINE CTCMB1(LS,MS,C)
      COMPLEX C(0:LS-1,0:1,0:MS-1,0:LS-1,0:1,0:MS-1)
      PARAMETER(PI=3.14...)
      COMPLEX CO,WYKO1,WYK10,WYK11

      ANGLE=PI/REAL(LS)
      DO 200 J1=0,MS-1
      DO 200 J2=0,MS-1
      DO 200 I1=0,LS-1
      DO 200 I2=0,LS-1
        WYK10=EXP(CMPLX(0.,-ANGLE*REAL(I1)))*
     $        C(I1,1,J1,I2,0,J2)
        WYKO1=EXP(CMPLX(0.,-ANGLE*REAL(I2)))*
     $        C(I1,0,J1,I2,1,J2)
        WYK11=EXP(CMPLX(0.,-ANGLE*REAL(I1+I2)))*
     $        C(I1,1,J1,I2,1,J2)
        CO=C(I1,0,J1,I2,0,J2)
        C(I1,0,J1,I2,0,J2)=CO+WYKO1+WYK10+WYK11
        C(I1,0,J1,I2,1,J2)=CO-WYKO1+WYK10-WYK11
        C(I1,1,J1,I2,0,J2)=CO+WYKO1-WYK10-WYK11
        C(I1,1,J1,I2,1,J2)=CO-WYKO1-WYK10+WYK11
200   CONTINUE
      END
```

Advantages:

* 3 complex products per step (doing it separately along the 2 axis requires 4);

Disadvantages:

* difficult to implement in multiradix case ($(n.radixes+1)^{n.dim.}$ different blocks!)
* very irregular memory access
* difficult to write if $n.dimensions > 2$(up to 7 subscripts in FORTRAN77!).

6 If N is prime?

If N is prime, the FFT algorithms do not apply. In practice, the same situation arises when N has any large factor. In these cases, let's rewrite eq(1) as

$$X_k = \sum_{n=0}^{N-1} x_n W^{nk+(k^2-k^2+n^2-n^2)/2}$$

that is

$$X_k = W^{k^2/2} \sum_{n=0}^{N-1} [W^{n^2/2} x_n] W^{-(k-n)^2/2} \qquad (5)$$

Let $y(n) = W^{n^2/2} x_n$ and $h(k-n) = W^{-(k-n)^2/2}$. Then 5 becomes

$$X_k = W^{k^2/2} \sum_{n=0}^{N-1} y(n) h(k-n)$$

that is a discrete (periodic) convolution. This can easily be shown to be equivalent to the discrete *aperiodic* convolution of y'_n : $y' = y$ if $0 \le n < N$, $y' = 0$ otherwise, with h'_n : $h' = h$ if $0 \le n < 2N$, $h' = 0$ otherwise. This aperiodic convolution can in turn be converted to a periodic one, having an arbitrary period of $N' > 2N$, simply replicating h' and y' with period N'. If N' is chosen highly factorizable, this convolution can be evaluated,

according to the general properties of DFT, by a direct FFT, a product and an inverse FFT. X_k is then evaluated with an additional product.

7 Alternatives to FFT

An important family of algorithms perform a DFT in a way that is radically different from the one used in FFT. They originate from the work of Winograd, and they are usually known as Winograd Fourier Transform (WFT). (see Winograd in Proc.Nat.Ac.Sci.US., 1976; Silverman, IEEE Trans. on Acoustics, 1977; Nussbaumer, Fast Fourier Transform and Convolution Algorithms, Springer, 1982). They are based on 2 fundamental concepts:

1. Let's write eq(1) as

$$X = W^{(N,N)}x$$

where $W^{(N,N)}$ is a $N \times N$ matrix. Then, for many small N's, Winograd demonstrated that there is a J, $J \geq N$, $J \sim N$, such that there is a decomposition of $W^{(N,N)}$ of the form

$$W^{(N,N)} = S^{(N,J)}C^{(J,J)}T^{(J,N)} \qquad (6)$$

where $S_{i,j}, T_{i,j} = 0, \pm 1$ and $C_{i,j} = \delta_{i,j}r_i$, r purely real or purely imaginary. This means that for these N's the DFT can be performed with $J \sim N$ *real* products, much less than the normal way, but at the price of a slightly increased number of sums. **These algorithms could be used as basic blocks in a standard FFT!**

2. If N is composed, but its factors are among those for which representation 6 is known, *and they are mutually prime*, Winograd has shown a way of combining 6's in a way that *does not require the "twiddle factors" used in normal FFT.*

This method of combination is based on the observation that,if $N_1, N_2, \ldots, N_L$ **are the mutually prime factors of** N, the *Kronecker product* $W_p^{(N,N)} = W^{(N_1,N_1)} \times W^{(N_2,N_2)} \times \ldots$ is simply a permutation of the rows and the colums of $W^{(N,N)}$.

Note: A Kronecker product of two matrices is defined by the following FORTRAN fragment:

```
      DIMENSION A(N1,N2,N1,N2),B(N1,N1),C(N2,N2)
      REAL KRONP(N1*N2,N1*N2)
      EQUIVALENCE (A,KRONP)
      DO 100 I=1,N1
      DO 100 J=1,N2
      DO 100 K=1,N1
      DO 100 L=1,N2
         A(I,J,K,L)=B(I,K)*C(J,L)
100   CONTINUE
```

A property of the Kronecker product is that $(AB) \times (CD)$ is equal to $(A \times C)(B \times D)$. Thus, from

$$W_p = W_{N_1} \times W_{N_2} \times \ldots$$

and from eq(6), one gets

$$W_p = (S_{N_1} \times S_{S_2} \times \ldots)(C_{N_1} \times C_{N_2} \ldots)(T_{N_1} \times T_{N_2} \times \ldots)$$

and it is easy to see that the only elements different from $0, \pm 1$ are contained in the Kronecker product

of the C's. The DFT is then reduced to a permutation, followed by a number of sums, followed by the product by the C's, followed by another set of sums and another permutation. Due to the structure of the C's, **the WFT decreases dramatically the number of multiplies (still of order $N \log_2 N$, but with a much lower coefficient)**, while it increases slightly the number of additions.

The implementation of the Winograd's FT is a piece of difficult programming. This because the nesting algorithm (item 2 above) is complex, and the various "small N" routines -item 1- are very different one from the other. Moreover, the saving offered by the Winograd's algorithm is maximal with machines with a slow multiply (a problem that's no longer as bad as it was 15 years ago), while it can be inefficient on machines with a separate multiplier and adder, where in any case the speed depends from the number of additions. For this reason, I do not know of any Winograd routine used as a general purpose tool, while it is widely used *for fixed N*, as parts of special-purpose machines (waveform analysis etc.)