FAST FOURIER TRANSFORMS—FOR FUN AND PROFIT

W. M. Gentleman

Bell Telephone Laboratories
Murray Hill, New Jersey

and

G. Sande *

Princeton University
Princeton, New Jersey

IMPLEMENTING FAST FOURIER TRANSFORMS

Definition and Elementary Properties of Fourier Transforms

The "Fast Fourier Transform" has now been widely known for about a year. During that time it has had a major effect on several areas of computing, the most striking example being techniques of numerical convolution, which have been completely revolutionized. What exactly is the "Fast Fourier Transform"?

In fact, the Fast Fourier Transform is nothing more than an algorithm whereby, for appropriate length sequences, the finite discrete Fourier transform of the sequence may be computed much more rapidly than by other available algorithms. The properties and uses of the finite discrete Fourier transform, which become practical when used with the fast algorithm, make the technique important. Let us therefore first consider some properties of such transforms.

The usual infinite Fourier integral transform is well known and widely used—the physicist when solving a partial differential equation, the communication engineer looking at noise and the statistician studying distributions may all resort to Fourier transforms, not only because the mathematics may be simplified, but because nature itself is often easier to understand in terms of frequency. What is less well known is that many of the properties of the usual Fourier transform also hold, perhaps with slight modification, for the Fourier transform defined on finite, equispaced, discrete sequences.

We see in Table I that the most significant change required to modify the usual theorems so that they apply to the finite discrete case is that indexing must be considered modulo N. A useful heuristic interpretation of this is to think of the sequence $X(t)$ as a function defined at equispaced points on a circle. This interpretation is to be contrasted with what is

*This work made use of computer facilities supported in part by National Science Foundation grant NSF–GP579. Research was partially supported by the Office of Naval Research under contract Nonr 1858(05) and by the National Research Council of Canada.

TABLE I

A Comparison of Usual and Finite Discrete Fourier Transforms

Usual	Finite Discrete
Definition	
$\hat{X}(\hat{t}) = \int_{-\infty}^{\infty} X(t)e^{2\pi i t\hat{t}}dt$	$\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e^{\frac{2\pi i t\hat{t}}{N}}$
Linearity	
The Fourier transform is a linear operator.	The Fourier transform is a linear operator.
Orthogonality	
$\int_{-\infty}^{\infty} e^{2\pi i t(\hat{t}-\hat{t}')}dt = \delta(\hat{t}-\hat{t}')$ where $\delta(\hat{t}-\hat{t}')$ is the Dirac delta function. That is, $\int_b^a f(\hat{t})\delta(\hat{t})d\hat{t} = f(0)$ if 0 is in the interval (a,b), otherwise $\int_a^b f(\hat{t})\delta(\hat{t})d\hat{t} = 0$.	$\sum_{t=0}^{N-1} e^{\frac{2\pi i t(\hat{t}-\hat{t}')}{N}} = N\delta_N(\hat{t}-\hat{t}')$ where δ_N is the Kronecker delta function with its argument being considered modulo N. That is, $\delta_N(kN) = 1$, for integer k, otherwise $\delta_N = 0$.
Inverse Transform	
If $\hat{X}(\hat{t}) = \int_{-\infty}^{\infty} X(t)e^{2\pi i t\hat{t}}dt$ then $X(t) = \int_{-\infty}^{\infty} \hat{X}(\hat{t})e^{-2\pi i t\hat{t}}d\hat{t}$ which we observe can be considered as $X(t) = \left\{\int_{-\infty}^{\infty} \{\hat{X}(\hat{t})\}^* e^{2\pi i t\hat{t}}d\hat{t}\right\}^*$ or the complex conjugate of the Fourier transform of the complex conjugate of X.	If $\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e^{\frac{2\pi i t\hat{t}}{N}}$ then $X(t) = \frac{1}{N}\sum_{\hat{t}=0}^{N-1} \hat{X}(\hat{t})e^{\frac{-2\pi i t\hat{t}}{N}}$ which we observe can be considered as $X(t) = \frac{1}{N} \times \left\{\sum_{\hat{t}=0}^{N-1} \{\hat{X}(\hat{t})\}^* e^{\frac{2\pi i t\hat{t}}{N}}\right\}^*$ or the complex conjugate of the Fourier transform of the complex conjugate of X, divided by N.
Convolution Theorem	
$\int_{-\infty}^{\infty} X(\tau)Y(t-\tau)d\tau = \int_{-\infty}^{\infty} e^{-2\pi i t\hat{t}}\hat{X}(\hat{t})\hat{Y}(\hat{t})d\hat{t}$	$\sum_{\tau=0}^{N-1} X(\tau)Y(t-\tau) = \frac{1}{N} \times \sum_{\hat{t}=0}^{N-1} e^{\frac{-2\pi i t\hat{t}}{N}}\hat{X}(\hat{t})\hat{Y}(\hat{t})$

TABLE 1—(Continued)

Usual	Finite Discrete
that is, the inverse Fourier transform of the product of the Fourier transforms.	that is, the inverse Fourier transform of the product of the Fourier transforms. NOTE: The convolution here must be considered as cyclic, i.e., the indices of X and Y must be interpreted modulo N.
Operational Calculus	
An operational calculus can be defined, based on the property that $\int_{-\infty}^{\infty} \left(\frac{\partial}{\partial t} X(t)\right)e^{2\pi i t\hat{t}}dt = -2\pi i\hat{t}\hat{X}(\hat{t})$ i.e., the Fourier transform of the derivative of a function is the Fourier transform of the function, multiplied by $-2\pi i\hat{t}$.	An operational calculus can be defined, based on the property that $\sum_{t=0}^{N-1} (\triangle X(t))e^{\frac{2\pi i t\hat{t}}{N}} = \left(e^{\frac{-2\pi i\hat{t}}{N}} - 1\right)\hat{X}(\hat{t})$ i.e., the Fourier transform of the (forward) difference of a function is the Fourier transform of the function, multiplied by $\left(e^{\frac{-2\pi i\hat{t}}{N}} - 1\right)$. NOTE: The difference here must be considered cyclically, so that $\triangle X(t) = X(t+1) - X(t)$ becomes $\triangle X(N-1) = X(N) - X(N-1) = X(0) - X(N-1)$ for the case of $t = N - 1$.
Symmetries	
If X is real then $\hat{X}$ is hermitian symmetric, i.e., $\hat{X}(\hat{t}) = \{\hat{X}(-\hat{t})\}^*$; if X is hermitian symmetric then $\hat{X}$ is real. If Y is imaginary, then $\hat{Y}$ is hermitian antisymmetric, i.e., $\hat{Y}(\hat{t}) = -\{\hat{Y}(-\hat{t})\}^*$; if Y is hermitian antisymmetric then $\hat{Y}$ is imaginary.	If X is real then $\hat{X}$ is hermitian symmetric, i.e., $\hat{X}(\hat{t}) = \{\hat{X}(N-\hat{t})\}^*$; if X is hermitian symmetric, then $\hat{X}$ is real. If Y is imaginary, then $\hat{Y}$ is hermitian antisymmetric, i.e., $\hat{Y}(\hat{t}) = -\{\hat{Y}(N-\hat{t})\}^*$; if Y is hermitian antisymmetric then $\hat{Y}$ is imaginary. NOTE: The use of the terms hermitian symmetric and hermitian antisymmetric in the discrete case is consistent with that in the usual case if we interpret indices modulo N.
Shifting Theorems	
$\int_{-\infty}^{\infty} X(t+h)e^{2\pi i t\hat{t}}dt = e^{2\pi i\hat{t}h}\hat{X}(\hat{t})$	$\sum_{t=0}^{N-1} X(t+h)e^{\frac{2\pi i t\hat{t}}{N}} = e^{\frac{-2\pi i\hat{t}h}{N}}\hat{X}(\hat{t})$

(These results are all readily proved from the definitions.)

often the natural interpretation—as sampled data from a continuous infinite function.

Basic Algebra of Fast Fourier Transforms

At this point we will define the notation $e(X) = e^{2\pi iX}$ so that we may write expressions such as those of Table I in a simpler form. The two most important properties of $e(X)$ are that

$$e(X+Y) = e(X)e(Y)$$

and

$$e(X) = 1 \quad \text{if } X \text{ is an integer.}$$

Using this notation, the finite discrete Fourier transform of the sequence $X(t)$ is

$$\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e\left(\frac{t\hat{t}}{N}\right)$$

Suppose N has factors A and B so that $N = AB$. Then writing $\hat{t} = \hat{a} + \hat{b}A$ and $t = b + aB$ where $a,\hat{a} = 0,1,\ldots,A-1$ and $b,\hat{b} = 0,1,\ldots,B-1$; we have

$$\hat{X}(\hat{a}+\hat{b}A) =$$

$$= \sum_{b=0}^{B-1}\sum_{a=0}^{A-1} X(b+aB)e\left(\frac{\{\hat{a}+\hat{b}A\}\{b+aB\}}{AB}\right)$$

$$= \sum_{b=0}^{B-1}\sum_{a=0}^{A-1} X(b+aB)e\left(\frac{\hat{a}b}{AB}+\frac{\hat{a}a}{A}+\frac{\hat{b}b}{B}+\hat{ab}\right)$$

$$= \sum_{b=0}^{B-1}\sum_{a=0}^{A-1} X(b+aB)e\left(\frac{\hat{a}b}{AB}\right)e\left(\frac{a\hat{a}}{A}\right)e\left(\frac{b\hat{b}}{B}\right)$$

as $\hat{ab}$ is integral implies $e(\hat{ab}) = 1$

$$= \sum_{b=0}^{B-1} e\left(\frac{b\hat{b}}{B}\right)\left\{e\left(\frac{\hat{a}b}{AB}\right)\sum_{a=0}^{A-1} X(b+aB)e\left(\frac{a\hat{a}}{A}\right)\right\}$$

If we define the B different sequences

$$W_b(a) = X(b+aH) \quad a = 0,1,\ldots,A-1$$

and the A different sequences

$$Z_{\hat{a}}(b) = e\left(\frac{\hat{a}b}{AB}\right)\sum_{a=0}^{A-1} X(b+aB)e\left(\frac{a\hat{a}}{A}\right)$$

$$b = 0,1,\ldots,B-1$$

we can write the above equation as

$$\hat{X}(\hat{a}+\hat{b}B) = \sum_{b=0}^{B-1} e\left(\frac{b\hat{b}}{B}\right) Z_{\hat{a}}(b)$$

where

$$Z_{\hat{a}}(b) = e\left(\frac{\hat{a}b}{AB}\right)\left\{\sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right) W_b(a)\right\}$$

We recognize

$$\sum_{b=0}^{B-1} e\left(\frac{b\hat{b}}{B}\right) Z_{\hat{a}}(b) \text{ and } \sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right) W_b(a)$$

as Fourier transforms themselves, applied to shorter sequences. Obtaining a subsequence by starting at the bth element and taking every Bth element thereafter, in the manner $W_b(a)$ is obtained from $X(b+aB)$, is called decimating by B. Observe that the sequence $Z_{\hat{a}}(b)$ are not quite the sequence of frequency $\hat{a}$ values from the transforms of these decimated sequences, but these values multiplied by a "twiddle factor" $e\left(\frac{\hat{a}b}{AB}\right)$.

The Fourier transform of the complete AB point sequence may thus be accomplished by doing the B different A point Fourier transforms, multiplying through by the appropriate twiddle factors, then doing the A different B point Fourier transforms. This is a recursive formula which defines the larger Fourier transform in terms of smaller ones. The total number of operations is now proportional to $AB(A+B)$ rather than $(AB)^2$ as it would be for a direct implementation of the definition, hence the name "Fast Fourier Transform".

Associated observations:

1. Attention is drawn to the special advantage of factors 2 or 4, in that since a Fourier transform of a 2 or 4 point sequence may be done using only additions and subtractions, one stage of complex arithmetic may be avoided.

2. Although the recursive algorithm above may be implemented directly, it is much better to simulate the recursion as described in the following sections, since one may avoid the unnecessary calculation of some complex exponentials. With our programs, for example, the simulating the recursion takes only 2/5 as long for a 2^{10} point transform.*

* All programs discussed in this paper were implemented using ALCOR Algol 60 on the IBM 7094 at Princeton University.

3. The multi-dimensional finite discrete Fourier transform

$$\hat{X}(\hat{t}_1,\ldots,\hat{t}_k) = \sum_{t_1=0}^{N_1-1}\cdots\sum_{t_k=0}^{N_k-1} X(t_1,\ldots,t_k)\, e\left(\frac{t_1\hat{t}_1}{N_1}\right)\cdots e\left(\frac{t_k\hat{t}_k}{N_k}\right)$$

can be computed efficiently by factorizing in each dimension separately, as above.

4. The algebra presented here is not quite that appearing in "An Algorithm for the Machine Calculations of Complex Fourier Series," by J. W. Cooley and J. W. Tukey.[2] The difference can be seen if we consider N as having three factors, $N = ABC$.

The Cooley-Tukey algebra then is

$$\hat{X}(\hat{c}+\hat{b}C+\hat{a}BC) = \sum_{a=0}^{A-1}\sum_{b=0}^{B-1}\sum_{c=0}^{C-1} X(a+bA+cAB)\, e\left(\frac{(\hat{c}+\hat{b}C+\hat{a}BC)\cdot(a+bA+cAB)}{ABC}\right)$$

$$= \sum_{a=0}^{A-1}\sum_{b=0}^{B-1}\sum_{c=0}^{C-1} X(a+bA+cAB)\, e\left(\frac{a\cdot(\hat{c}+\hat{b}C+\hat{a}BC)}{ABC}\right) \cdot e\left(\frac{b\cdot(\hat{c}+\hat{b}C+\hat{a}BC)}{BC}\right) e\left(\frac{c(\hat{c}+\hat{b}C+\hat{a}BC)}{C}\right)$$

$$= \sum_{a=0}^{A-1} e\left(\frac{a\cdot(\hat{c}+\hat{b}C+\hat{a}BC)}{ABC}\right)\sum_{b=0}^{B-1} e\left(\frac{b\cdot(\hat{c}+\hat{b}C)}{BC}\right)\sum_{c=0}^{C-1} e\left(\frac{c\hat{c}}{C}\right) X(a+bA+cAB)$$

If, rather than collecting on the unhatted variables as above, we choose to collect on the hatted variables, we obtain

$$\hat{X}(\hat{c}+\hat{b}C+\hat{a}BC) = \sum_{a}^{A-1}\sum_{b}^{B-1}\sum_{c}^{C-1} X(a+bA+cAB)\, e\left(\frac{\hat{a}(a+bA+cAB)}{A}\right) \cdot e\left(\frac{\hat{b}\cdot(a+bA+cAB)}{AB}\right) e\left(\frac{\hat{c}\cdot(a+bA+cAB)}{ABC}\right)$$

$$= \sum_{a=0}^{A-1} e\left(\frac{\hat{a}a}{A}\right)\sum_{b=0}^{B-1} e\left(\frac{\hat{b}\cdot(a+bA)}{AB}\right)\sum_{c=0}^{C-1} e\left(\frac{\hat{c}\cdot(a+bA+cAB)}{ABC}\right) X(a+bA+cAB)$$

In both cases we may factor the twiddle factor outside the summation. If we do this we obtain in the first case

$$\hat{X}(\hat{c}+\hat{b}C+\hat{a}BC) = \sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right) e\left(\frac{a\cdot[\hat{c}+\hat{b}C]}{ABC}\right) \sum_{b=0}^{B-1} e\left(\frac{b\hat{b}}{B}\right) e\left(\frac{b\hat{c}}{BC}\right)\sum_{c=0}^{C-1} e\left(\frac{c\hat{c}}{C}\right) X(a+bA+cAB)$$

(Cooley version)

In the second case we obtain

$$\hat{X}(\hat{c}+\hat{b}C+\hat{a}BC) = \sum_{a=0}^{A-1} e\left(\frac{a\hat{a}}{A}\right) e\left(\frac{a\hat{b}}{AB}\right)\sum_{b=0}^{B-1} e\left(\frac{b\hat{b}}{B}\right) e\left(\frac{\hat{c}(a+bA)}{ABC}\right)\sum_{c=0}^{C-1} e\left(\frac{c\hat{c}}{C}\right) X(a+bA+cAB)$$

(Sande version)

The difference between these two versions becomes important when we come to the details of implementing the fast Fourier transform. We go on to this next.

Fast Fourier Transforms Using Scratch Storage

Let us consider the Sande version applied to an X sequence stored in serial order. The summation over c represents a C point Fourier transform of points spaced AB apart. There is one such transform for each of the AB values of $a + bA$. The result of one such transform is a frequency sequence indexed by $\hat{c}$. If we look at the twiddle factor $e\left(\frac{\hat{c}(a+bA)}{ABC}\right)$ we see that it depends upon $\hat{c}$ and $a + bA$ in a very convenient manner. To introduce some descriptive terminology we may call $\hat{c}$ the frequency for this analysis and $a + bA$ the displacement for this analysis. At this stage there are no free indices corresponding to replications. When we store the intermediate results in the scratch storage area, placing each of the C point Fourier transforms in contiguous blocks indexed by the displacement leads to elements stored at $\hat{c} + C(a+bA)$. The intermediate summation over b represents a B point Fourier transform with points spaced AC apart; one such transform for each of the AC values of $\hat{c} + aC$. The result of any

one of these transforms is a frequency sequence indexed by $\hat{b}$. The twiddle factor $e\left(\frac{a\hat{b}}{AB}\right)$ depends only upon a and $\hat{b}$, leaving $\hat{c}$ as a free index. Here we would call $\hat{b}$ the frequency for the analysis, a the displacement for the analysis and $\hat{c}$ the replication index. We would store the intermediate results at $\hat{c} + \hat{b}C + aBC$. In this case the contiguous blocks are spaced out by the replication factor C. The outer summation over a represents an A point Fourier transform with points spaced BC apart; one such transform for each of the BC values of $\hat{c} + \hat{b}C$. The result of any one of these transforms is a frequency sequence indexed by $\hat{a}$. There is no twiddle factor in this case. Here we would call $\hat{a}$ the frequency and $\hat{c} + \hat{b}C$ the replication index. There is no displacement at this stage. We would store the results at $\hat{c} + \hat{b}C + \hat{a}BC$. At this point we see that the results are stored in serial order of frequency.

When we compute one of the transforms we may wish to include the twiddle factor in the coefficients for the values of the replication index before computing new coefficients for a different displacement. If the Fourier transform is on two points there appears to be no advantage to either choice. For four points it is more economical to do the Fourier transform and then multiply by the twiddle factor. In the other cases it is more efficient if the twiddle factor is absorbed into the transform coefficients.

If we were to use the Cooley version on a sequence stored in serial order we would obtain an algorithm which differs only in minor details. The summation over c represents a C point Fourier transform of points spaced AB apart. The twiddle factor $e\left(\frac{b\hat{c}}{BC}\right)$ depends upon the frequency $\hat{c}$ and displacement b, leaving a free as a replication index. The elements are stored at $\hat{c} + aC + bAC$. The intermediate summation over b represents a B point Fourier transform of points spaced AC apart. The twiddle factor $e\left(\frac{a(\hat{c}+\hat{b}C)}{ABC}\right)$ depends upon the combined frequency $\hat{c} + \hat{b}C$ and the displacement a. There is no free index. The intermediate results are stored at $\hat{c} + \hat{b}C + aBC$. The outer summation represents an A point Fourier transform of results spaced BC apart. There is no twiddle factor in this case. The final results would be stored in serial order at $\hat{c} + \hat{b}C + \hat{a}BC$.

These two reductions are essentially identical from an algorithmic viewpoint when we use storage in this manner. We have only reversed the roles of the hatted and unhatted variables in the formation of the twiddle factors.

To obtain the general case of more than three factors we proceed to group our factors in three groups, for example $N = (P_1 \ldots P_{j-1})P_j(P_{j+1} \ldots P_n)$. If we identify

$$\begin{aligned} A &= P_1 \ldots P_{j-1} \\ B &= P_j \\ C &= P_{j+1} \ldots P_n \end{aligned}$$

and perform the above reductions we find that after two steps with this identification we arrive at exactly the same place as after only one step with the identification

$$\begin{aligned} A &= P_1 \ldots P_{j-2} \\ B &= P_{j-1} \\ C &= P_j \ldots P_n \end{aligned}$$

We can think of this as moving factors from the "A" part through the "B" part to the "C" part.

When we write a program to implement this, we set up a triply nested iteration. The outer loop selects the current factor being moved and sets up the limits and indexing parameters for the inner loops. The intermediate loop, in which we would compute the twiddle factor, corresponds to the displacement. The inner loop provides indexing over the replication variables.

Fast Fourier Transforms in Place

Let us reconsider the storage requirements for our algorithm. In particular, let us consider one of the small Fourier transforms on c for some value of $a + bA$. It is only during this transform that we will need this set of intermediate results. We have just vacated the C cells $a + bA + cAB$ (indexed by c) and are looking about for C cells in which to store our answers (indexed by $\hat{c}$). Why not use the cells that have just been vacated?

Having made this observation, let us reexamine the algorithm corresponding to the Sande factorization. The summation on c represents a C point Fourier transform of points spaced AB apart. The twiddle factor $e\left(\frac{\hat{c}(a+bA)}{ABC}\right)$ depends upon the frequency $\hat{c}$ and the displacement $a + bA$. There is no replication index. The intermediate results are stored at $a + bA + \hat{c}AB$. The summation on b represents a B point Fourier transform of points A apart. The

twiddle factor $e\left(\frac{a\hat{b}}{AB}\right)$ depends upon the frequency $\hat{b}$ and the displacement a. The replication index is $\hat{c}$. We treat each of the blocks of length AB indexed by $\hat{c}$ in an exactly equivalent manner. The summation over a represents contiguous A point Fourier transforms. The frequency is $\hat{a}$ and the replication index is $\hat{b} + c\hat{B}$. There is no displacement. The final answers are then stored at $\hat{a} + \hat{b}A + \hat{c}AB$. The use of "displacement" and "replication" is now much more suggestive and natural than when we used these while describing the use of scratch storage.

The more general case of more than three factors is again obtained by grouping to obtain "A", "B", and "C" and then moving factors from the "A" part to the "C" part; the program being written as a triply nested iteration loop.

When we reexamine the Cooley factorization for serially stored data, we will see that it has an unpleasant feature. The summation over c represents a C point Fourier transformation of points spaced AB apart. The twiddle factor $e\left(\frac{b\hat{c}}{AB}\right)$ depends only upon b and $\hat{c}$, leaving a as a free index. The intermediate results are stored at $a + bA + \hat{c}AB$. The summation over b represents a B point Fourier transform of points spaced A apart. The twiddle factor $e\left(\frac{a(\hat{c}+\hat{b}c)}{ABC}\right)$ depends upon the displacement a and the combined frequency $\hat{c} + \hat{b}C$. For data which was originally stored serially, this is not convenient to compute. The intermediate results are stored at $a + \hat{b}A + \hat{c}AB$. The summation over a represents contiguous A point Fourier transforms. The final results are stored at $\hat{a} + \hat{b}A + \hat{c}AB$. More than three factors can be handled in the same manner as before.

The common unpleasant feature of both of these algorithms is that the Fourier coefficient for frequency $\hat{c} + \hat{b}C + \hat{a}BC$ is stored at $\hat{a} + \hat{b}A + \hat{c}AB$. Such a storage scheme may be described as storage with "digit reversed subscripts." Before we may use our Fourier coefficients we must unscramble them.* Unscrambling has been found to require only a small proportion of the total time for the algorithm. A very elegant solution corresponds to running two counters, one with normal digits (easily obtained by simply incrementing by one) and one with reversed digits (easily obtained by nesting loops with the innermost stepping by the largest increments) and recopying from one array to another where one counter is used for each array. If all of the factors are the same, the required interchanges become pairwise and it is possible to do the unscrambling in place. One may unscramble the real and imaginary parts separately, so scratch space could be generated by backing one or the other onto auxiliary store. A slower method of unscrambling is to follow the permutation cycles so that no scratch storage is needed.

*This problem has nothing to do with the representation of numbers in any particular machine, and unless the machine has a "reverse digit order" instruction (e.g., "reverse bit order" for a binary machine), no advantage can be taken of such representation.

TABLE II

TIME FOR A 1024 POINT TRANSFORM BY VARIOUS METHODS

Method	*Time*
radix 2	2.0 seconds
radix 4 (also radix 4 + 2 and mixed radices)	1.1 seconds
radix 2 (recursively implemented)	5.2 seconds
Goertzel's method	59.1 seconds

Two implementations, both for serially stored sequences, have been described. Alternately, two implementations, both for data stored with digit reversed subscripts, may be developed: in this instance the Cooley factorization has more convenient twiddle factors. For the last two, the final results are correctly stored—the "unscrambling" having been done first. Digit reversed subscripts may arise because we have

1. scrambled serially stored data,
2. not unscrambled after a previous Fourier transform,
3. generated data in scrambled order.

We have written various Fourier transform programs in the manner described above. These include one- and multi-dimensional versions of radix 2 (all factors equal to 2), radix 4 (all factors equal to 4), radix 4 + 2 (all factors equal to 4 except perhaps the last, which may be 2), and mixed radices (N is factored into as many 4's as possible, a 2 if necessary, then any 3's, 5's or other primes). Times required to transform a 1024 point sequence are given in Table II, including the time required by a recursive radix 2 transform and by the pre-fast Fourier transform "efficient" Goertzel's method[3] (which still requires order N^2 operations) for comparison.

Our standard tool is the radix 4 + 2 Fourier transform which combines the speed of radix 4 with the

flexibility of radix 2. The mixed radices Fourier transform is used when other factors, for example 10, are desired. The mixed radix is used less as it is more bulky (requiring scratch storage for unscrambling as well as being a larger program) and is less thoroughly optimized than the radix 4 + 2 Fourier transform.

Fast Fourier Transforms Using Hierarchical Store

As we become interested in doing larger and larger Fourier transforms, we reach a point where we may not be able to have all of the data in core simultaneously, and some must be kept in slower store such as drum, disk, or tape. At the other end of the scale, small amounts of very fast memory will be available with some of the "new generation" computers—memory much faster than the rest of core. Automatic systems for providing "virtual core" obscure the distinctions within this hierarchy of memories, but at great loss of efficiency for Fourier transforms. On the other hand, we can apply the basic recursion formula of the section *Basic Algebra of Fast Fourier Transforms* to employ hierarchical store in such a way as to operate on sequences large enough to require the slower store, yet to run little slower than if all the memory had the speed of the faster store.

In particular, when we factor N into the two factors A and B, we choose A as the largest Fourier transform that can be done in the faster store. We then compute the transforms (including unscrambling) of the B subsequences obtained by decimating the original sequence by B. We next multiply through by the appropriate twiddle factors, and do the B point transforms. Thus, other than decimating and recopying, which amounts to a matrix transpose, all the computations are done in the faster store, the loading and unloading of which can be overlapped. As an example, consider using a $32K$ machine with disk to Fourier transform a quarter million (2^{18}) point sequence (point here means complex number) initially on tape, returning the transform to tape. We will assume that we are willing to do transforms as large as 4096 points (2^{12}) in core.

The first step is to create 64 (2^6) files on disk, each of length 4096. The tape is read into core and the various decimated sequences read out onto disk—the zeroth point going into the zeroth file, the first into the first, etc. up to the sixty-third into the sixty-third, then the sixty-fourth point starting back in the zeroth file again, etc. Appropriately buffered, this can run almost at disk transfer speed.

The 64 files are now brought into core one at a time. When a file comes in, it is Fourier transformed using a standard in-core transform, then the points are multiplied by the twiddle factor $e(\hat{a}b/2^{18})$, where b is the number of the file (0 through 63) and $\hat{a}$ is the frequency index for the particular point. The file is then written out again. By choosing to use transforms of only 4096 points, we can fully buffer these operations, one file being Fourier transformed while the previous file is being written out and the succeeding file read in.

We now start on the second set of transforms. Here we make use of the random access feature of disk to read simultaneously from each of the 64 files, since there are 4096 subsequences of length 64, obtained by taking one element from each file. Each subsequence is brought in, transformed, and then returned to its previous position on the disk, that is, as one number in each of the files 0 through 63, according to the frequency index. Again we may completely buffer this operation so that the disk time comes free.

Finally we output our Fourier transform onto tape, sequentially writing out each of the files, 0 through 63.

Rough calculations for the IBM 7094 Mod I show that the initial decimating is limited by disk speed, and takes about 3 minutes, that the first 64 (4096 point) transforms will take a total of just under 6 minutes and are compute-bound so that disk time is free, that the last 4096 (64 point) transforms will take a total of about 3 minutes, again compute-bound so that disk time is free, and the final output is completely limited by disk speed, and takes about 3 minutes. Thus our quarter-million point Fourier transform has cost us only about 15 minutes, 9 minutes of which is computing, the same 9 minutes which would be required if the machine had enough core to do the entire problem internally, and 6 minutes of which is strictly tape to disk or disk to tape transmission that could be avoided if the sequence was initially available, and the final answers acceptable, in appropriate form or could at least be overlapped with other computing.

Roundoff Considerations

So far we have discussed the fast Fourier transform as though it could be done with complete accuracy. What happens when the algorithm is carried out in a real fixed word-length computer using floating point arithmetic? How does the error in doing

this compare with the error from a direct application of the definition?

We may approach this question two ways—by strict bounds or by empirical values determined by experimentation. We shall derive bounds for the ratio of the Euclidean norm * of the error to the Euclidean norm of the data sequence, showing that whereas for direct application of the defining formula we can only bound this ratio by $1.06\sqrt{N}\,(2N)^{3/2}2^{-b}$, if we factor N into $n_1n_2\ldots n_k$ and use the fast Fourier transform we can bound the ratio by $1.06\sqrt{N}\left\{\sum_j (2n_j)^{3/2}\right\}2^{-b}$. In particular, if all the n_j are the same, so that $N = n^k$, then the ratio is less than $1.06\sqrt{N}\;k(2n)^{3/2}2^{-b}$, which is $k/n^{3/2(k-1)}$ times that for the direct calculation. (b here is the number of bits used in the mantissa of the floating-point representation). We shall then see from empirical results that the form of the bound is actually a fairly good description of the form of the observed errors, although the numerical constant is somewhat larger than is typical of actual trials.

Theorem: If we use the defining formula to Fourier transform a sequence $X(t)$ of length N in a machine using floating point arithmetic and a fixed word length with a b bit mantissa, then

$$||f\ell(\hat{X}) - \hat{X}||_E < 1.06\sqrt{N}\,(2N)^{3/2}2^{-b}||\hat{X}||_E$$

Proof: We require the following lemma from Wilkinson [9] (p. 83):

> Lemma: If A is a real p by q matrix and B is a real q by r matrix and if $f(.)$ denotes the result of floating point computation then
>
> $$||f\ell(AB) - AB||_E < 1.06q2^{-b}||A||_E||B||_E$$

*The Euclidean norm of a vector or sequence is the square root of the sum of the squares of the elements.

q is the extent of the summation in the matrix multiplication.

The definition of the Fourier transform of $X(t)$ is also the definition of the multiplication of the vector $X(t)$ by the matrix $\{e(\hat{it}/N)\}_{\hat{it}}$.

Expressing this in real arithmetic, we multiply the real sequence $\{Re(X(t)),\ Im(X(t))\}$ by the real matrix

$$\begin{pmatrix} C & -S \\ S & C \end{pmatrix}$$

where $C = \{cos\ 2\pi\hat{it}/N\}_{\hat{it}}$ and $S = \{sin\ 2\pi\hat{it}/N\}_{\hat{it}}$. The norm of this $2N\times 2N$ real matrix is

$$\sum_{\hat{it}=0}^{N-1}\ \sum_{\hat{it}=0}^{N-1} \{cos^2 2\pi\hat{it}/N + sin^2 2\pi\hat{it}/N + cos^2 2\pi\hat{it}/N + sin^2 2\pi\hat{it}/N\} = \sqrt{2N^2}$$

Since the extent of the summation is $2N$, we have by the lemma that $||f\ell(\hat{X}) - \hat{X}||_E < 1.06\times 2N\sqrt{2N^2}\,2^{-b}\,||X||_E$.

The proof is completed by observing that $||\hat{X}||_E = \sqrt{N}\,||X||_E$.

Theorem: If we use the fast Fourier transform, factoring N into $n_1n_2\ldots n_k$, to transform a sequence $X(t)$ in a machine using floating point arithmetic and a fixed word length with a b bit mantissa, then,

$$||f\ell(\hat{X}) - \hat{X}||_E < 1.06\sqrt{N}\sum_{j=1}^{k}(2n_j)^{3/2}2^{-b}||\hat{X}||_E$$

Proof: Here we represent the transform as a sequence of matrix multiplies (one for each factor), and employ the above lemma. Specifically, the fast Fourier transform is equivalent to evaluating $\sqrt{N}\,M_kM_{k-1}\ldots M_1X$ where M_j is a matrix consisting of N/n_j disjoint n_j point Fourier transforms (including the twiddle factors for the next stage), rescaled to be orthogonal. Thus

$$\begin{aligned}
||f\ell(\hat{X}) - \hat{X}||_E &= \sqrt{N}\,||f\ell M_k f\ell M_{k-1}\ldots f\ell M_1X - M_kM_{k-1}\ldots M_1X||_E \\
&= \sqrt{N}\,||f\ell M_k f\ell M_{k-1}\ldots f\ell M_1X - M_kf\ell M_{k-1}f\ell M_{k-2}\ldots f\ell M_1X \\
&\qquad + M_kf\ell M_{k-1}\ldots f\ell M_1X - M_kM_{k-1}f\ell M_{k-2}\ldots f\ell M_1X \\
&\qquad \ldots + M_kM_{k-1}\ldots f\ell M_1X - M_kM_{k-1}\ldots M_1X||_E \\
&\le \sum_{j=1}^{k}\sqrt{N}\,||M_k\ldots M_{j+1}(f\ell M_jf\ell M_{j-1}\ldots f\ell M_1X - M_jf\ell M_{j-1}\ldots f\ell M_1X)||_E
\end{aligned}$$

but since the M_j are orthogonal

$$= \sqrt{N}\sum_{j=1}^{k}||f\ell M_j(f\ell M_{j-1}\ldots f\ell M_1X) - M_j(f\ell M_{j-1}\ldots f\ell M_1X)||_E$$

When we compute the bound for $||f\ell(M_jY) - M_jY||_E$ we find that since M_j may be partitioned into N/n_j disjoint blocks, we can bound the error from each block separately, getting, since in real arithmetic each block is $2n_j$ square, and the norm of the block is $\sqrt{2n_j}$, a bound of $1.06(2n_j)^{3/2}2^{-b}||Y_S||_E$ where Y_S is the appropriate part of Y. By using Pythagoras' theorem this yields

$$||f\ell(M_jY) - M_jY||_E < 1.06(2n_j)^{3/2}2^{-b}||Y||_E$$

Finally we observe that except for error of order $N2^{-b}||X||_E$

$$||f\ell M_{j-1}f\ell M_{j-2}\ldots f\ell M_2X||_E = ||M_j\ldots M_1X||_E$$

Immediately we have $||M_j\ldots M_1X||_E = ||X||_E$ because of the orthogonality, so

$$||f\ell(\hat{X}) - \hat{X}||_E < 1.06\sqrt{N}\sum_{j=1}^{k}(2n_j)^{3/2}2^{-b}||\hat{X}||_E$$

Corollary: If a sequence is transformed and then the inverse transform applied to the result, the norm of the difference between the initial and final sequences can be bounded by $2\times1.06(2N)^{3/2}2^{-b}||X||_E$ if we use the definition, or $2\times1.06\sum_j(2n_j)^{3/2}2^{-b}||X||_E$ if we use the fast Fourier transform.

With this corollary in mind, an experiment we can readily do is to take a sequence, transform, inverse transform, and then compute the norm of the difference between the original and resultant sequences, dividing this by the norm of the original sequence. Table III gives these ratios for random Gaussian sequences of length 2^k, $k = 1,\ldots,12$, when the transforms were computed using: (1) a radix 2 transform; (2) a radix 4 + 2 transform; (3) a radix 4 + 2 transform with rounding instead of truncation; (4) a Goertzel's method transform; and (5) a transform which directly implemented the definition. The results of three replications of the experiment are given. Figure 1 is a plot of the average ratio divided by $\log_2N$ against $\log_2N$ to show the essentially linear dependence (for the factored transforms) of the ratio on $\log_2N$. Attention is drawn to the common scaling factor of 10^{-8} and the consistency of the tabled ratios for the larger transforms.

TABLE III

OBSERVED RATIOS OF ERROR NORM TO SEQUENCE NORM FOR THREE RANDOM SEQUENCES AT EACH LENGTH.
(in units of 10^{-8})

		Radix 4+2			
Log_2N	Radix 2	Radix 4+2	with rounding	Goertzel's method	Defining formula
1	1.03	(1.03)	.46	2.60	4.27
	.53	(.53)	.13	3.91	1.54
	.57	(.57)	.00	2.73	4.14
2	3.43	1.09	.92	9.73	5.15
	2.23	.92	1.05	12.6	3.31
	1.89	1.96	1.07	15.8	4.14
3	4.04	4.99	2.65	32.8	9.75
	5.57	5.58	2.69	16.6	13.0
	4.51	5.01	2.39	34.4	7.83
4	9.14	7.11	2.12	98.9	17.7
	8.64	5.92	2.91	91.5	18.4
	7.76	6.62	2.71	121.	17.1
5	10.7	11.0	4.58	202.	33.8
	12.7	12.2	4.44	258.	36.7
	11.9	11.4	5.40	198.	36.2
6	13.2	11.2	3.38	548.	69.1
	14.4	12.0	3.70	787.	75.2
	14.0	10.2	3.59	806.	63.8
7	17.6	17.0	6.24	1290.	143.
	16.9	16.3	6.74	1990.	141.
	17.3	16.8	6.78	1900.	135.
8	20.0	16.3	4.82	7050.	286.
	20.1	16.6	5.09	3490.	288.
	20.7	16.2	4.66	5090.	269.
9	22.8	21.6	7.61	13700.	579.
	22.8	21.4	7.81	10700.	578.
	22.9	21.7	7.91	11500.	561.
10	25.2	21.0	5.82	32100.	1170.
	25.6	21.0	5.44	27600.	1160.
	25.8	21.0	5.44	29900.	1140.
11	28.1	26.2	8.70		
	28.1	26.1	8.56	——	——
	28.5	26.5	8.53		
12	30.7	25.5	6.30		
	30.7	25.5	6.21	——	——
	31.1	25.7	6.21		

USING FAST FOURIER TRANSFORMS

Classical Usage with Partial Differential Equations

One of the classical motivations for the study of Fourier transforms has been the application of finding solutions of partial differential equations. Hockney [4] gives a discussion of numerical solutions of partial differential equations. He considers the problem

$$\frac{\partial^2\Phi(x,y)}{\partial x^2} + \frac{\partial^2\Phi(x,y)}{\partial y^2} = \rho(x,y) \qquad \begin{matrix} 0 \le x \le \ell \\ 0 \le y \le m \end{matrix}$$

with boundary conditions $\Phi(x,y) = 0$ if $x = 0,\ell$ or $y = 0,m$. The solution is obtained by discretizing

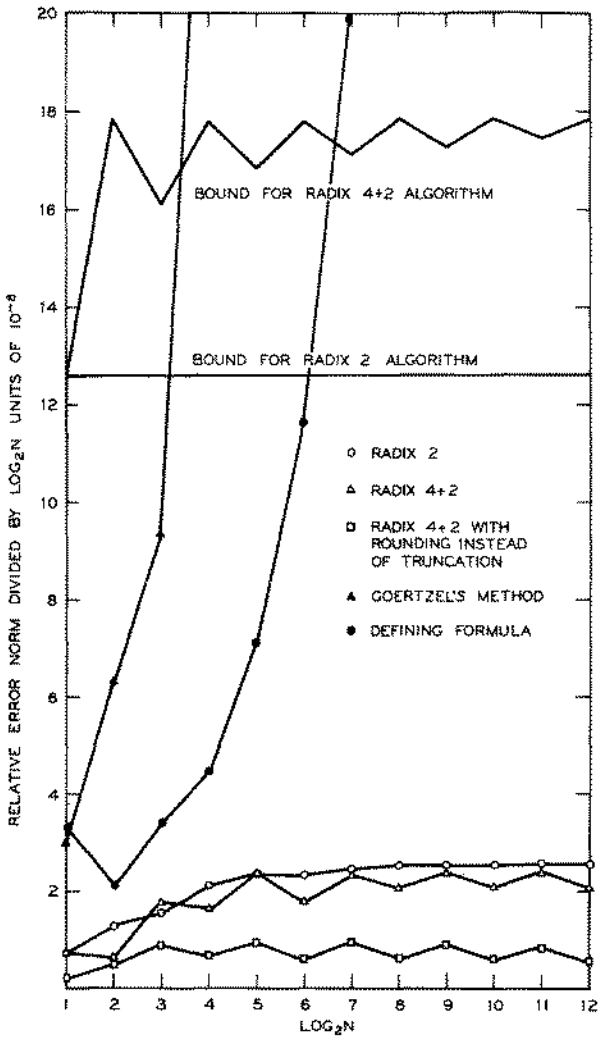

Figure 1. Observed values of relative error norm/Log_2N.

onto an $n \times n$ grid (Hockney considers $n = 48$, one case of $n = 12 \times 2^q$, $q = 0,1,2,\ldots$). A Fourier transform with respect to x reduces the problem to the solution of a system of equations for the y coordinate. By using a 12 point formula from Whittaker and Robinson,[8] Hockney is able to achieve a Fourier transform in $n^2/36$ operations.* If rather he were to use a fast Fourier transform for $n = 2^p$, $p = 0,1,2,\ldots$ he could achieve a Fourier transform in approximately $\frac{3}{2} n \log_2 n$ operations. This estimate of operations leads to a net count of $6n^2\log_2 n$ for Hockney's method. Under this accounting, Hockney's method is much superior to the other methods he considers. It must be cautioned that the factoring of 12×2^2 which Hockney uses for his Fourier transform will be equivalent to the fast Fourier transform. The apparent reduction only comes into play as the size, n, is increased past 48. As Hockney points out, the calculation of Fourier coefficients has long been recommended (and avoided because of cost) as a method of solving partial differential equations. Perhaps the fast Fourier transform will render the solution of partial differential equations both more economical and more straightforward.

*Operation in this section means a real operation of a multiplication and an addition.

Least Squares Approximation by Trigonometric Polynomials

Often one may want to approximate a given sequence by a band-limited sequence, that is, a sequence generated by a low order trigonometric polynomial. A possible example could be the determination of filter coefficients to approximate a desired transfer function.[5] The use of expansions in terms of orthogonal functions to obtain least square approximations is well known. The Fourier transform of a sequence gives its expansion in terms of the mutually orthogonal complex exponentials, hence to obtain the coefficients for the approximation, we simply retain the low order coefficients from the Fourier transform. We may compare the approximation to the original by looking at the inverse Fourier transform of the coefficients after replacing the unwanted coefficients by zeros.

If we had wanted a positive definite approximation, we could have approximated the square root of the original sequence. The square of the approximation to the root is then the positive approximation. Squaring a sequence like this has the effect of convolving the Fourier coefficients. Other variations in the approximating problem may be handled by similar specialized techniques.

A caution: Gibbs phenomenon is a persistent problem when using least squares, particularly in the case of trigonometric approximation. In some situations, the problem may be sufficiently bad to warrant the use of other approximation criteria.

Numerical Convolutions

To date, the most important uses of the fast Fourier transform have been in connection with the convolution theorem of Table I. Some uses of numerical convolutions are the following:

Auto- and Cross-Covariances: 1. METHOD AND TIMING CONSIDERATIONS. In the analysis of time

series by digital spectrum analysis, there are many computational advantages to the "indirect" method of spectral estimation by computing the autocovariance and then Fourier transforming rather than the "direct" method by obtaining the periodogram and then smoothing.[1] For example, it is more practical to consider examining a spectrum through several different spectral windows by the indirect method. Similarly, when analyzing a pair of series with co- and quadrature-spectra, we often find the cross-covariance of the two series computationally convenient.

The first major use of the convolution theorem, by Sande,[6] was to compute the auto-covariance

$$R_{XX}(\tau) = \frac{1}{N}\sum \{X(t)\}^* X(t+\tau) \qquad \tau = 0,\pm 1,\pm 2,\ldots,\pm L$$

and cross-covariance

$$R_{XY}(\tau) = \frac{1}{N}\sum \{X(t)\}^* Y(t+\tau) \qquad \tau = 0,\pm 1,\pm 2,\ldots \pm L.$$

where the summations are overall values of t for which the products are defined. (These are not in the form of Table I but can be readily put into that form.) The convolution theorem may lead to improvements by a factor of 20 in computing time as compared to the summing of lagged products. The two major problems are that these are not defined cyclically and that N may not be convenient for use of the fast Fourier transform. Appending zeros to one or both ends of the series solves both problems.

To see this, extend X to length $N' \geq N + L$ by adding zeros so that

$$X'(t) = X(t) \qquad t = 0,1,\ldots,N-1$$
$$X'(t) = 0 \qquad t = N,\ldots,N'-1$$

Extend the Y series similarly. Fourier transform both new sequences to obtain

$$\hat{X}'(\hat{t}) = \sum_{t=0}^{N'-1} X'(t) e\left(\frac{t\hat{t}}{N'}\right)$$

$$\hat{Y}'(\hat{t}) = \sum_{s=0}^{N'-1} Y'(s) e\left(\frac{s\hat{t}}{N'}\right)$$

Inverse transform the sequence formed by the product of the Y' transform and the complex conjugate of the X' transform.

$$C(\tau) = \frac{1}{N'}\sum_{\hat{t}=0}^{N'-1} \{\hat{X}'(\hat{t})\}^* \hat{Y}'(\hat{t}) e\left(-\frac{\hat{t}\tau}{N'}\right)$$

$$= \frac{1}{N'}\sum_{\hat{t}=0}^{N'-1}\sum_{t=0}^{N'-1}\sum_{s=0}^{N'-1} \{X'(t)\}^* Y'(s) e\left(-\frac{\hat{t}\tau}{N'}\right) e\left(-\frac{t\hat{t}}{N'}\right) e\left(\frac{s\hat{t}}{N'}\right)$$

$$= \frac{1}{N'}\sum_{t=0}^{N'-1}\sum_{s=0}^{N'-1} \{X'(t)\}^* Y'(s) \sum_{\hat{t}=0}^{N'-1} e\left(\frac{\hat{t}(s-t-\tau)}{N'}\right)$$

which by the orthogonality condition of Table I gives

$$= \sum_{t=0}^{N'-1}\sum_{s=0}^{N'-1} \{X'(t)\}^* Y'(s)\delta_{N'}(s-t-\tau)$$

$$= \sum_{t=0}^{N'-1} \{X'(t)\}^* Y'(t+\tau)$$

Recall that the indices must be interpreted modulo N'. For $\tau < N' - N$, this expression is just N times the cross-covariance R_{XY} because then the extraneous products are precisely zero.

Note that the auto-covariance is precisely the cross-covariance of a series with itself, so that the outline just given also describes how to compute the auto-covariance. On the other hand, we may take advantage of the fact that in most time series problems the series are real to compute two auto-covariances at one time. We use one series as the real part and the other series as the imaginary part in forming the sequence which we Fourier transform, using the symmetry properties from Table I to separate them later.

Exactly how much faster the convolution theorem approach is than the lagged products depends not only on the length of series and proportion of the available lags desired, but also on details of the programs used. There are $\frac{(2N-L)(L+1)}{2}$ multiplications and additions, plus associated indexing operations, in summing lagged products. On the other hand, the number of operations required for the Fourier transforms (approximately proportional to $N' \log N'$) depends on our choice of N' which can vary according to the availability of Fourier transform routines, a routine which can handle mixed factors being much more flexible than a radix 4 + 2 routine which restricts N' to be a power of two. Figure 2 gives a comparison of direct summing and

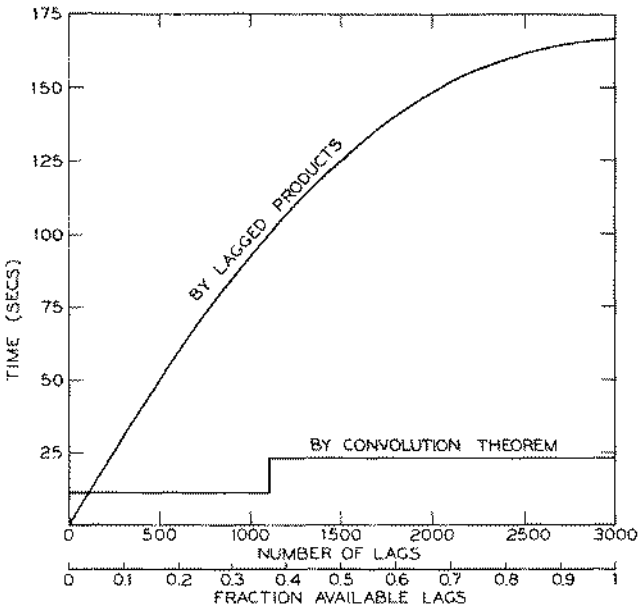

Figure 2. Computation times for auto-covariance of a 3000 point series, using convolution theorem or summing lagged products.

convolution theorem times using our radix 4 + 2 transform routine. Figure 3 shows (for this program) the regions in which either direct summing or the convolution theorem is more efficient as functions of N and L.

2. Extra-long Series. When we have to compute auto- or cross-covariances of extra-long series, we could just continue to use the method of the previous section, doing the oversize Fourier transforms by the method outlined in the section *Fast Fourier Transforms Using Hierarchical Store*. A simpler and occasionally more efficient method is also available. What we do instead is to partition the series X and Y into contiguous disjoint segments, and express the cross-covariance R_{XY} in terms of the cross-covariances between these various subseries.

A simple diagram will illustrate the method. In Fig. 4 we have plotted all the products of the elements of series Y against those of series X, the plotting position corresponding to the values of the respective indices. The cross-covariance $R_{XY}(\tau)$ then corresponds to $1/N$ times the sum over all elements on a line parallel to the main diagonal, but τ elements above it. Such a line is shown on the diagram.

As an example, we consider the problem of computing R_{XY} for $\tau = 0, \pm 1, \ldots \pm L$, when the largest cross-covariance we can compute by the method of the sub-section on Method and Timing Considerations is for series of length $N/5$. We are thus interested in those products in the hatched area of the diagram. The segments described above are indicated by the dotted lines; they are labeled $X_0, X_1, \ldots X_4$ and $Y_0, Y_1, \ldots Y_4$. We now see how $R_{XY}(\tau)$ may be calculated, by summing the appropriate cross-covariances of X_i with Y_j. For instance, for the particular value of τ illustrated,

$$R_{XY}(\tau) = \frac{1}{5}\left\{ R_{X_0Y_0}(\tau) + R_{X_1Y_0}\left(\tau - \frac{N}{5}\right) + R_{X_1Y_1}(\tau) + R_{X_2Y_1}\left(\tau - \frac{N}{5}\right) + R_{X_2Y_2}(\tau) + R_{X_3Y_2}\left(\tau - \frac{N}{5}\right) + R_{X_3Y_3}(\tau) + R_{X_4Y_3}\left(\tau - \frac{N}{5}\right) + R_{X_4Y_4}(\tau)\right\}$$

Filtering: Digital Filters. Often, when we have a set of data which is indexed by displacement or time, we wish to "smooth" our data to reduce the effect of unwanted "noise". If our original data were represented by $\{X(t)\}$ then forming

$$Y(t) = \frac{X(t)+X(t+1)+X(t+2)}{3}$$

would represent one method of smoothing our data. More generally, we could choose to use any arbitrary weights we pleased so that

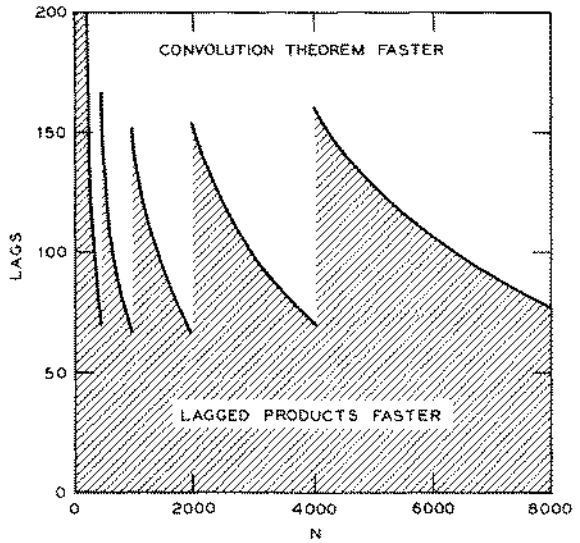

Figure 3. Region where using convolution theorem is faster than summing lagged products.

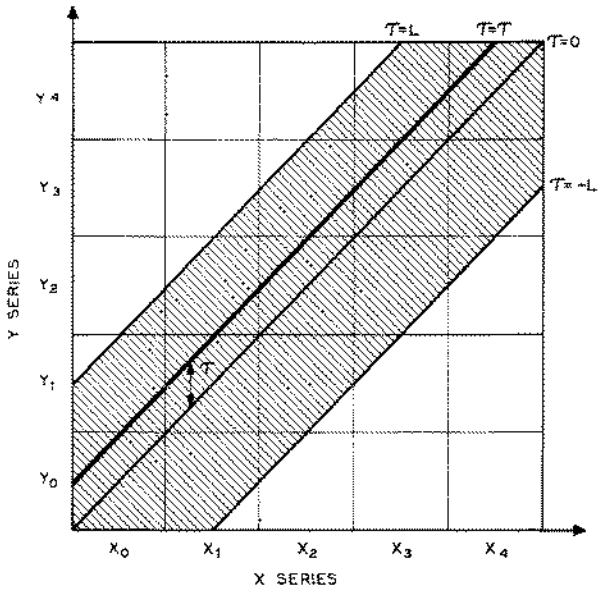

Figure 4. Diagram of cross products for computing cross-covariance.

$$Y(t) = c(k)X(t) + c(k-1)X(t+1) + \ldots + c(o)X(t+k) = \sum_{j=0}^{k} c(k-j)X(t+j).$$

If, rather than smoothing, we had desired to estimate moving short-term integrals (possibly with a kernel) we would again be able to express the result as a summation over the data points weighted by suitable coefficients. The estimation of derivatives also leads to a similar expression.

Let us consider the application of such a moving average to a complex exponential of a fixed frequency. Thus our data would be $X(t) = e^{i\omega t} = e(ft)$ and the result of the moving average would be

$$\begin{aligned} Y(t) &= c(k)X(t) + \ldots + c(o)X(t+k) \\ &= \sum_{j=0}^{k} c(k-j)X(t+j) \\ &= \sum_{j=0}^{k} c(k-j)e(f\times\{t+j\}) \\ &= e(tf) \sum_{j=0}^{k} c(k-j)e(fj). \end{aligned}$$

We can thus represent $Y(t)$ as the product of $e(ft)$ and $A(f) = \sum_{j=0}^{k} c(k-j)e(fj)$. $A(f)$ is called the (complex) gain of the moving average (or filter) determined by the coefficients $c(o),\ldots,c(k)$. Moving averages are linear in the data, for example, the moving average of $U(t) + V(t)$ is the same as the sum of the separate moving averages of $U(t)$ and $V(t)$. Thus if we consider our data as being composed of many Fourier components, we see that the moving average affects each of the Fourier components separately.

All of these considerations can be compactly stated using the terminology we have at hand. Taking the moving average is a convolution with a set of weights. The complex gain of the moving average is the Fourier transform of the coefficients. Filtering corresponds to multiplying by the complex gain in the frequency domain.

This has given but a small introduction to the analysis of linear filters. It does not tell us how to design a filter to perform a desired task. For this problem, the reader is referred to general texts such as Hamming,[3] or Blackman and Tukey,[1] or to more specialized works such as Kaiser.[5]

2. Sectioning. After we have chosen the weights with which we would like to filter, we still must choose how to implement the filter. A typical situation might be to have 15,000 data points which we wish to filter with 50 weights. We could do this by computing the convolution directly. We might also extend our data to 16K by adding zeros and our filter to 16K weights by adding zeros and then doing two Fourier transforms to get our convolution. For this choice of numbers the decision is not dramatically in favor of one or the other method.

Applying a 16K Fourier transform to only 50 weights seems rather extravagant. Perhaps we could do better if we were to filter only sections of the original data at any one time as first suggested by Stockham.[7] The problem is then to find a sensible size for the various sections. Let us say that we have D data points and F filter weights. We want to find N (the section length) so as to minimize the total time. This scheme will take about $D/(N-F)$ Fourier transforms (transforming two real sequences at one time). The total time will then be $t \quad \frac{D}{N-F}\, cN\ell n(N)$ where c may have a small dependency on N which we will ignore. By assuming that $F << D$ and that N is continuous we may find the minimum of t by differentiating with respect to N. Thus

$$\frac{\partial t}{\partial N} = Dc \frac{\partial}{\partial N}\left(\frac{N\ell n(N)}{N-F}\right)$$

$$= Dc\left\{\frac{\ell n(N)}{N-F} + \frac{N}{N-F}\frac{1}{N} - \frac{N\ell n(N)}{(N-F)^2}\right\}$$

$$= \frac{Dc}{(N-F)^2}\{(N-F)\ell n(N) + N - F - N\ell n(N)\}$$

$$= \frac{Dc}{(N-F)^2}\{-F\ell n(N) + N - F\}$$

$$\frac{\partial t}{\partial N} = 0 \quad \text{implies} \quad N - F(1 + \ell n(N)) = 0$$

which yields

$$F = \frac{N}{1 + \ell n(N)}.$$

For our case of 50 filter weights, this suggests using Fourier transforms of length about 300, for a reduction of the time required by a factor of two.

Interpolation. One of the standard operations in numerical computing is the interpolation in a table of function values to obtain approximate values on a grid at a finer spacing than the original.

1. Band-Limited Interpolation. The most obvious application of these ideas is to the case of band-limited interpolation, sometimes called trigonometric or cosine interpolation. This interpolation is done over the whole interval, by fitting a trigonometric polynomial (or complex exponential series) of sufficiently high order through all the given values, then evaluating it at the new points.[3] But the first step of this is exactly the finding of the Fourier transform of the original data. And the second step, as we shall see, can be accomplished by inverse transforming the sequence whose low order coefficients are the same as the coefficients of the transform of the original data, and whose high order coefficients are zero.

To justify this last statement, consider the case where the function is given at N equispaced grid points. We wish to interpolate in this sequence to obtain values at M times as many points. The original sequence of values, $X(t)$, has a Fourier transform $X(t)$ such that we have the representation

$$X(t) = \frac{1}{N}\sum_{\hat{t}=0}^{N-1} \hat{X}(\hat{t})e\left(-\frac{t\hat{t}}{N}\right)$$

Consider the augmented Fourier transform $\hat{Z}(\hat{s})$ $\hat{s} = 0,1,\ldots,N\cdot m-1$ such that

(a) $\hat{Z}(\hat{t}) = \hat{X}(\hat{t}) \left(-\frac{N}{2}\right) < \hat{t} < \frac{N}{2}$

(b) $\hat{Z}(-N/2) = \hat{Z}(N/2) = 1/2\hat{X}(N/2)$ if N even

(c) $\hat{Z}(\hat{s}) = 0$ for all other values of $\hat{s}$.

When we recall that $\hat{X}(\hat{t})$ has period N and that $\hat{Z}(\hat{s})$ has period NM we identify

$$\hat{X}(-\hat{t}) = \hat{X}(N-\hat{t})$$

and

$$\hat{Z}(-\hat{s}) = \hat{Z}(NM-\hat{s})$$

The construction of $\hat{Z}$ corresponds to taking the circle on which the Fourier transform coefficients of $\hat{X}$ are defined, breaking it at $\frac{N}{2}$, and inserting $M(N-1)$ zeros so that the low frequency Fourier coefficients of the two sequences match, and the high frequency coefficients of $\hat{Z}(\hat{s})$ are zero. What is the inverse transform of $\hat{Z}(\hat{s})$?

$$Z(s) = \frac{1}{MN}\sum_{\hat{s}=0}^{MN-1} \hat{Z}(\hat{s})e\left(-\frac{(s/M)\hat{s}}{N}\right)$$

which, we note, is just the band-limited interpolation of $X(t)$, except for a factor $\frac{1}{M}$. For example, when $s = Mt$,

$$Z(Mt) = \frac{1}{MN}\sum_{\hat{s}=0}^{MN-1} \hat{Z}(\hat{s})e\left(-\frac{t\hat{s}}{N}\right)$$

$$= \frac{1}{M}X(t) \text{ by the definition of } \hat{Z}(\hat{s})$$

Applying this in the case where the original values are not at equispaced points merely makes the finding of the first transform, $\hat{X}(\hat{t})$, more complicated. Furthermore, the extension of this scheme to multiple dimensions is immediate.

We remark that a fundamental fact of band-limited interpolation is that it is periodic, so that the first points of a sequence influence the values interpolated between the last points. This property, and the long range effects of single values, may mean we should avoid band-limited interpolations for certain uses. It may, however, be extremely useful in other circumstances.

2. General Interpolation Rules as Convolutions. Knowing the limitations of band-limited interpolation, we often prefer to use a local formula such as Lagrangian polynomial interpolation. Using any interpolation rule such that the interpolated

values are linear combinations of the adjacent known values to subtabulate a table of equispaced data may very conveniently be done by convolutions. If we denote the value to be interpolated at $t + p$, $0 \leq p < 1$, by $Z_p(t)$, then the series $Z_p(t)$ is the convolution of the original series $X(t)$ with an appropriate series of weights:

$$Z_p(t) = \sum_{s=\alpha}^{\beta} W_p(s)X(t-s)$$

It is possible to arrange that the convolutions for all values of p be done simultaneously, but the computational effort may be increased over doing them separately. Since, typically, the series of weights will be short compared to the length of the series $X(t)$, it may be profitable to employ the sectioning described earlier. When applied to problems in more than one dimension, additional profit may be made if we use interpolation rules which are direct products or direct sums of lower dimensional rules, for example, a two dimensional interpolation rule such that the weights $W_{pq}(r,s)$ are either a product $W'_p(r)W''_q(s)$ or a sum $W'_p(r) + W''_q(s)$. The advantage of such rules is that the higher dimensional convolutions may then be done as a succession of lower dimensional convolutions or equivalently that the Fourier transforms of such sequences are the products or sums, respectively, of the Fourier transforms of the constituent parts.

Cauchy Products for Symbolic Polynomial Manipulation. The Cauchy product of two infinite series is a well-known result of college algebra:

$$\sum_{i=0}^{\infty} a_i x^i \cdot \sum_{j=0}^{\infty} b_j x^j = \sum_{k=0}^{\infty} c_k x^k \text{ where}$$

$$c_k = \sum_{n=0}^{k} a_k b_{n-k}$$

subject to some convergence criteria. We readily recognize that the $\{c_k\}$ are convolutions of the $\{a_i\}$ and $\{b_j\}$ and that we could use our techniques to evaluate these discrete convolutions.

We may arrive at this by considering an alternative viewpoint. Let us write $z = e(\theta)$ and recognize that $z^n = e(n\theta)$. With this notation the Fourier transform becomes

$$\hat{X}(\hat{t}) = \sum_{t=0}^{N-1} X(t)e(\hat{t}t/N) = \sum_{t=0}^{N-1} X(t)z^t \text{ where}$$

$$z = e(\hat{t}/N)$$

This is a truncated power series evaluated at points on the unit circle in the complex plane. Our convolution theorem is now a statement about multiplying polynomials and using a Cauchy product representation. The inverse Fourier transform gives us a way of evaluating coefficients of polynomials given as values equispaced on the unit circle.

Of the applications so far suggested, this most deserves the epithet "fun." It is doubtful if this technique will supplant conventional symbol manipulation techniques as it makes no use of special properties of the coefficient sequences. It does, however, admirably illustrate that maintaining an open mind may well bring forth new, surprising and occasionally beneficial applications of finite discrete Fourier analysis.

Considering Problems in Transform Space

It has long been a fact that some problems of physics are more tractable in transform space than they are in the regular coordinates. Our refound ability to switch from one to the other may mean that some previously intractable problems will now be solvable. In electromagnetic problems, the return signal can often be represented as a convolution with a kernel. Taking the Fourier transform can change the form of the non-linearity from convolution to multiplication and may make the problems more manageable. In pattern recognition problems, one may seek measures which are free from the effects of linear transformations. Taking the Fourier transform may greatly reduce the problem of finding such measures. It has been suggested that quantizing the Fourier transform of a signal rather than quantizing the signal itself may result in a higher quality transmission for communication.

There are certainly a wealth of problems in which Fourier analysis arises as a most natural tool to be employed. Unfortunately it has often been rejected because of its high computational cost. Perhaps the application of the fast Fourier transform could swing the economic balance to make this time-honored technique again competitive.

ACKNOWLEDGMENT

The authors would like to express their appreciation to those from whose work we have borrowed, and whose fruitful conversations have helped in the preparation of this paper. In particular we should

like to thank C. Bingham, A. B. Langdon, C. L. Mallows, T. G. Stockham, and J. W. Tukey.

REFERENCES

1. R. B. Blackman and J. W. Tukey, *The Measurement of Power Spectra,* Dover, New York, 1958.

2. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation,* vol. 19, no. 90, (1965) pp. 297–301.

3. R. W. Hamming, *Numerical Analysis for Scientists and Engineers,* McGraw-Hill, New York, 1962.

4. R. W. Hockney, "A Fast Direct Solution of Poisson's Equation Using Fourier Analysis," *Journal of the Association of Computing Machinery,* vol. 12, no. 1, (1965) pp. 95–113.

5. J. F. Kaiser, "Some Practical Considerations in the Realization of Linear Digital Filters," Proceedings of the Third Allerton Conference on Circuit and System Theory, Monticello, Illinois, October 1965.

6. G. Sande, "On an Alternative Method of Calculating Covariance Functions," unpublished, Princeton University, 1965.

7. T. G. Stockham, "High Speed Convolution and Correlation," *AFIPS, volume 28, 1966 Spring Joint Computer Conference,* Spartan Books, Washington, 1966.

8. Whittaker and Robinson, *Calculus of Observations,* Blackie & Son, London, 1944.

9. J. H. Wilkinson, *Rounding Errors in Algebraic Processes,* Prentice-Hall, Englewood Cliffs, New Jersey, 1963.