$\hat{a}+\hat{b}t$, and look at the residuals $X_1(t)=X(t)-\hat{a}-\hat{b}t$. This is a new function, which has a spectrum. Is this the spectrum we want? In other words, should one heed advice to pick out and throw away linear trends? The point is that there is no such thing as magically taking out linear trends. If one wants the spectrum of the trend, then one should be estimating that. If one wants the spectrum of the series that is hidden in, or superposed on, the trend, one should be estimating that. The spectrum should be interpreted with the idea that there are many spectra, each of which should be looked at.

Another important term is *coherence*. To my mind, one of the most important roles of spectra is in the fitting of models. In this role, two kinds of testing problems are very important in spectrum analysis: testing for the presence of white noise and testing for zero coherence.

To model a time series, we try to represent it as the output of a system whose input is white noise (a series of zero mean uncorrelated random variables). Now the question is: How does one know when one has the model that one has an input that is white? The idea of testing for white noise is a crucial problem in the statistical theory of spectral analysis. To interpret computed spectra, one must apply the many ways of testing for white noise, each powerful against different kinds of alternatives.

Similarly, to interpret computed spectra one must apply various tests of coherency. Unit coherence is indicative of linearity of a system, since we have unit coherence between input and output. Zero coherence is the measure of how one splits the signal into incoherent parts. The problem of estimating coherence happens to be a very tricky problem, since it is very easy to think one has zero coherence when it is not there, because of bad computational technique. So coherence is another aspect of spectral analysis that has to be kept in mind: its definition, how to compute it, and how to interpret it.

I would like to offer a final idea that may be useful. People are very often interested in classifying patterns or records (for example, cardiograms). That is, one may want to decide whether a cardiogram is from a "good" patient. Various techniques are being considered for examining the record and performing some kind of analysis on it. It seems to me that one ought to consider taking a Fourier transform of these records, and work with that in the same role. That is, whenever someone thinks of a time domain approach to a problem, one should consider taking the Fourier transform of the time record of the sample and use that. Similarly, when one talks about pattern recognition in the plane, people are interested in recognizing the various letters of the alphabet. I have always wondered why they do not take a two-dimensional Fourier transform of the data; this might avoid some positioning problems. These are some of the ideas that have come to mind as I listened to talks on pattern-recognition problems.

Historical Notes on the Fast Fourier Transform

JAMES W. COOLEY, PETER A. W. LEWIS, AND PETER D. WELCH

Abstract—**The fast Fourier transform algorithm has a long and interesting history that has only recently been appreciated. In this paper, the contributions of many investigators are described and placed in historical perspective.**

Historical Remarks

THE FAST FOURIER transform (FFT) algorithm is a method for computing the finite Fourier transform of a series of N (complex) data points in approximately $N \log_2 N$ operations. The algorithm has a fascinating history. When it was described by Cooley and Tukey [1] in 1965 it was regarded as new by many knowledgeable people who believed Fourier analysis to be a process requiring something proportional to N^2 operations with a proportionality factor which could be reduced by using the symmetries of the trigonometric functions. Computer programs using the N^2-operation methods were, in fact, using up hundreds of hours of machine time. However, in response to the Cooley–Tukey paper, Rudnick [5], of Scripps Institution of Oceanography, La Jolla, Calif., described his computer program which also takes a number of operations proportional to $N \log_2 N$ and is based on a method published by Danielson and Lanczos [2]. It is interesting that the Danielson-Lanczos paper described the use of the method in X-ray scattering problems, an area where, for many years after 1942, the calculations of

Manuscript received January 26, 1967; revised March 6, 1967.
The authors are with the IBM Research Center, Yorktown Heights, N. Y.

Fourier transforms presented a formidable bottleneck to researchers who were unaware of this efficient method. Danielson and Lanczos refer to Runge [6], [7] for the source of their method. These papers and the lecture notes of Runge and König [8] describe the procedure in terms of sine-cosine series. The greatest emphasis, however, was on the computational economy that could be derived from the *symmetries* of the sine and cosine functions. In a relatively short section of Runge and König [8] it was shown how one could use the *periodicity* of the sine-cosine functions to obtain a $2N$-point Fourier analysis from two N-point analyses with only slightly more than N operations. Going the other way, if the series to be transformed is of length N and N is a power of 2, the series can be split into $\log_2 N$ subseries and this doubling algorithm can be applied to compute the finite Fourier transform in $\log_2 N$ doublings. The number of computations in the resulting successive doubling algorithm is therefore proportional to $N \log_2 N$ rather than N^2. The use of symmetries only reduces the proportionality factor while the successive doubling algorithm replaces N^2 by $N \log N$. This distinction was not important for the values of N used in the days of Runge and König. However, when the advent of computing machinery made calculations with large N possible, and the $N \log N$ methods should have been thoroughly exploited, they were apparently overlooked, even though they had been published by well-read and well-referenced authors.

The fast Fourier transform algorithm of Cooley and Tukey [1] is more general in that it is applicable when N is composite and not necessarily a power of 2. Thus, if two factors of N are used, so that $N = r \times s$, the data is, in effect, put in an r-column, s-row rectangular array, and a two-dimensional transform is performed with a phase-shifting operation intervening between the transformations in the two dimensions. This results in $N(r+s)$ operations instead of N^2. By selecting N to be highly composite, substantial savings result. For the very favorable situation when N is equal to a power of 2, the Cooley–Tukey method is essentially the successive doubling algorithm mentioned above and takes $N \log_2 N$ operations.

The 23-year hiatus in the use of the algorithm seemed quite remarkable, and prompted us to inquire of Prof. L. H. Thomas at the IBM Watson Scientific Computing Laboratory, New York City, N. Y., as to whether he was familiar with the successive doubling algorithm for computing Fourier series, and knew of any occasions when it had been used. It turned out that Prof. Thomas had spent three months in 1948 doing calculations of Fourier series on a tabulating machine, using what he referred to as the "Stumpff method of subseries." The algorithm described by Thomas [10] was thought at first to be essentially the same as the fast Fourier transform algorithm of Cooley and Tukey since it also achieved its economy by performing one-dimensional Fourier analysis by doing multidimensional Fourier analysis. However, the algorithms are different for the following reasons: 1) in the Thomas algorithm the factors of N must be mutually prime; 2) in the Thomas algorithm the calculation is precisely multidimensional Fourier analysis with no intervening phase shifts or "twiddle factors" as they have been called; and 3) the correspondences between the one-dimensional index and the multidimensional indexes in the two algorithms are quite different. The Thomas or "prime factor" algorithm is described in detail and compared with the fast Fourier transform algorithm in the next section.[1] It can be extremely useful when used in combination with the fast Fourier transform algorithm.

Several other calculations have been reported in the literature and in private communications which use one or the other of the two algorithms.

Another line of development has since led to the Thomas algorithm in its full generality. This comes from work in the analysis and design of experiments. Let $A(k_0, k_1, \cdots, k_{m-1})$ be, for example, a crop yield when a level k_i of treatment i, which may be an amount of fertilizer, is used. Yates [11] considered the case where $k_i = 0$ or 1, meaning treatment i is or is not used. This yields $N = 2^m$ values of crop yields and, to get all possible differences between all possible averages, one would, in principle, have to compute N linear combinations of all of the A's. This would require N^2 operations. Yates devised a scheme whereby one computed a new array of N sums and differences of pairs of the A's. The process was repeated on the new array with pairs selected in a different order. This was done $m = \log_2 N$ times, meaning he did the calculation in $N \log_2 N$ operations instead of N^2.

Good [4] noted that the Yates method could be regarded as m-dimensional Fourier analysis with only two points in each direction and that the procedure could be generalized to one for an arbitrary number of points in each direction. Then Good showed that if N is composite, with mutually prime factors, i.e., $N = r_1 r_2, \cdots, r_m$, one could do a one-dimensional Fourier analysis of N points by doing m-dimensional Fourier analysis on an m-dimensional, $r_1 \times r_2 \times, \cdots, \times r_m$, array of points. With these ideas put together and developed, Good's paper contains the full generalization of the Thomas prime factor algorithm.

The Prime Factor Algorithm

As mentioned in the previous section, the algorithm used by Thomas and described later by Good has been mistakenly said to be equivalent to the fast Fourier transform algorithm of Cooley and Tukey. It is impor-

[1] Actually, Stumpff [9] gave only a doubling and a tripling algorithm and suggested (see Stumpff [9] p. 442, line 11) that the reader generalize to obtain the method for factors of N other than 2 or 3. Thomas made a further assumption (assuming that the index called r by Stumpff was equal to N/s where $s = 2$ or 3) which led to his algorithm. Without this assumption, Stumpff's description leads to the Cooley–Tukey algorithm.

TABLE I

CORRESPONDENCE BETWEEN ONE- AND TWO-DIMENSIONAL INDEXING IN THE ARBITRARY FACTOR ALGORITHM FOR THE CASE $r=8$, $s=3$, AND $N=24$

$n=sn_1+n_0=3n_1+n_0$

n_0 \ n_1	0	1	2	3	4	5	6	7
0	0	3	6	9	12	15	18	21
1	1	4	7	10	13	16	19	22
2	2	5	8	11	14	17	20	23

$j=rj_1+j_0=8j_1+j_0$

j_1 \ j_0	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23

TABLE II

CORRESPONDENCE BETWEEN ONE- AND TWO-DIMENSIONAL INDEXING IN THE PRIME FACTOR ALGORITHM FOR THE CASE $r=8$, $s=3$, AND $N=24$

$n\equiv rn_0+sn_1\equiv 8n_0+3n_1 \pmod{24},\ 0\leq n<N)$

n_0 \ n_1	0	1	2	3	4	5	6	7
0	0	3	6	9	12	15	18	21
1	8	11	14	17	20	23	2	5
2	16	19	22	1	4	7	10	13

$j\equiv s\cdot s_r j_0+r\cdot r_s j_1=9j_0+16j_1 \pmod{24},\ 0\leq j<N)$

j_1 \ j_0	0	1	2	3	4	5	6	7
0	0	9	18	3	12	21	6	15
1	16	1	10	19	4	13	22	7
2	8	17	2	11	20	5	14	23

tant to distinguish between these two algorithms since each has its particular advantages which can be exploited in appropriate circumstances.

The differences will be illustrated by considering the calculation of a Fourier series using two factors of N. The Fourier series is

$$X(j) = \sum_{n=0}^{N-1} A(n) W_N^{jn} \tag{1}$$

where $W_N = e^{2\pi i/N}$. Consider first the fast Fourier transform algorithm. We assume $N=r\cdot s$, and define a one-to-one mapping between the integers j, $0\leq j<N$, and the pairs of integers (j_1, j_0), $0\leq j_0<r$, $0\leq j_1<s$, by the relation

$$j = j_1 r + j_0. \tag{2}$$

Similarly, we let

$$n = n_1 s + n_0, \tag{3}$$

where

$$0 \leq n < N, \qquad 0 \leq n_0 < s, \qquad 0 \leq n_1 < r.$$

This enables us to refer to $A(n)$ and $X(j)$ as though they were two-dimensional arrays and permits us to do the Fourier analysis in two steps

$$A_1(j_0, n_0) = \sum_{n_1=0}^{r-1} A(n_1, n_0) W_r^{j_0 n_1} \tag{4}$$

$$X(j_1, j_0) = \sum_{n_0=0}^{s-1} A_1(j_0, n_0) W_s^{j_1 n_0} W_N^{j_0 n_0}. \tag{5}$$

Table I shows where $A(n)$ and $X(j)$ are placed in the two-dimensional arrays indexed by (n_1, n_0) and (j_1, j_0), respectively, for $r=8$ and $s=3$. For this case, (4) consists of three eight-term Fourier series, one for each row of the n table. Then, if j_0 is taken to be the column index of the results, $A_1(j_0, n_0)$, (5) describes eight Fourier series of three terms each on the columns of the array of $A_1(j_0, n_0) W_N^{j_0 n_0}$. The factor $W^{j_0 n_0}$, referred to as the "twiddle factor" by Gentleman and Sande [3], is usually combined with either the $W_r^{j_0 n_1}$ factor in (4) or the $W_s^{j_1 n_0}$ factor in (5).

For the Thomas prime factor algorithm, one must require that r and s be mutually prime. In this case, different mappings of the one-dimensional arrays into two-dimensional arrays are used. These are also one-to-one mappings and are defined as follows. Let

$$n \equiv rn_0 + sn_1 \pmod{N} \qquad (0 \leq n < N) \tag{6}$$

and

$$\begin{aligned} j_0 &\equiv j \pmod{r} \qquad (0 \leq j_0 < r) \\ j_1 &\equiv j \pmod{s} \qquad (0 \leq j_1 < s). \end{aligned} \tag{7}$$

Then the expression of j, in terms of j_0 and j_1, is a solution of the "Chinese remainder problem" and is given by

$$j \equiv s\cdot s_r j_0 + r\cdot r_s j_1 \pmod{N} \qquad (0 \leq j < N) \tag{8}$$

where s_r and r_s are solutions of

$$\begin{aligned} s\cdot s_r &\equiv 1 \pmod{r} \qquad s_r < r \\ r\cdot r_s &\equiv 1 \pmod{s} \qquad r_s < s, \end{aligned}$$

respectively. Substituting (6) and (8) and using (7) gives

$$W_N^{jn} = W_N^{jn_0 r} W_N^{jn_1 s} = W_s^{jn} W_r^{jn_1} = W_s^{j_1 n_0} W_r^{j_0 n_1}$$

which enables one to write the Fourier series (1) in the form

$$A_1(j_0, n_0) = \sum_{n_1=0}^{r-1} A(n_1, n_0) W_r^{j_0 n_1} \tag{9}$$

$$X(j_1, j_0) = \sum_{n_0=0}^{s-1} A_1(j_0, n_0) W_s^{j_1 n_0}. \tag{10}$$

As in the fast Fourier transform algorithm, this is a two-dimensional Fourier transform. The essential difference is that the "twiddle factor" $W_N^{j_0 n_0}$ does not appear in (10) and the correspondence between one- and two-dimensional indexing is different. The presence of the

"twiddle factor" does not introduce any more computation, but it does increase programming complexity slightly. To illustrate better how the indexing in the two algorithms differs, the mappings of n and j for the Thomas prime factor algorithm are given in Table II for comparison with the indexing described in Table I.

The prime factor algorithm can be programmed very easily in a source language like FORTRAN and, therefore, can be used efficiently with a subroutine designed for a number of terms equal to a power of two. For example, if r is a power of 2 and s is any odd number, the subseries (9) can be computed by the power of 2 subroutine.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. of Comput.*, vol. 19, pp. 297–301, April 1965.

[2] G. C. Danielson and C. Lanczos, "Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids," *J. Franklin Inst.*, vol. 233, pp. 365–380 and 435–452, April 1942.

[3] W. M. Gentleman and G. Sande, "Fast Fourier transforms for fun and profit," *1966 Fall Joint Computer Conf., AFIPS Proc.*, vol. 29. Washington, D. C.: Spartan, 1966, pp. 563–578.

[4] I. J. Good, "The interaction algorithm and practical Fourier analysis," *J. Roy. Statist. Soc.*, ser. B, vol. 20, pp. 361–372, 1958; Addendum, vol. 22, 1960, pp. 372–375. (MR 21 1674; MR 23 A4231.)

[5] P. Rudnick, "Note on the calculation of Fourier series," *Math. of Comput.*, vol. 20, pp. 429–430, July 1966.

[6] C. Runge, *Zeit. für Math. und Physik*, vol. 48, p. 443, 1903.

[7] C. Runge, *Zeit. für Math. und Physik*, vol. 53, p. 117, 1905.

[8] C. Runge and H. König, "Die Grundlehren der Mathematischen Wissenschaften," *Vorlesungen über Numerisches Rechnen*, vol. 11. Berlin: Julius Springer, 1924.

[9] K. Stumpff, *Tafeln und Aufgaben zur Harmonischen Analyse und Periodogrammrechnung*. Berlin: Julius Springer, 1939.

[10] L. H. Thomas, "Using a computer to solve problems in physics," *Application of Digital Computers*. Boston, Mass.: Ginn, 1963.

[11] F. Yates, *The Design and Analysis of Factorial Experiments*. Harpenden: Imperial Bureau of Soil Science.

Application of the Fast Fourier Transform to Computation of Fourier Integrals, Fourier Series, and Convolution Integrals

JAMES W. COOLEY, PETER A. W. LEWIS, AND PETER D. WELCH

Abstract—**The fast Fourier transform is a computational procedure for calculating the finite Fourier transform of a time series. In this paper, the properties of the finite Fourier transform are related to commonly used integral transforms including the Fourier transform and convolution integrals. The relationship between the finite Fourier transform and Fourier series is also discussed.**

Introduction

MANY PROBLEMS of current interest require the use of a variety of numerical methods for their solution. A technique that has found wide applicability is the integral transform method. However, numerical problems are generally solved with the aid of a digital computer which is not designed to handle the continuous waveforms that occur when integral transform methods are used. For this reason, it is necessary to convert continuous time series or other functions to a series of discrete data samples, and to perform numerical operations such as the finite Fourier transform on these samples. The fast Fourier transform (FFT) algorithm has reduced the time required to compute finite Fourier transforms by the fraction $\log_2 N/N$ where N is the number of discrete data samples. For large values of N, this reduction is important. It is therefore necessary to relate the properties of continuous integral transforms to the properties of the finite Fourier transform in order to take advantage of digital computers and the fast Fourier transform algorithm. In this paper, the correspondence between the finite Fourier transform and the Fourier integral is described, and a method that can be used to compute Fourier integrals is discussed in detail. Other problems covered are the computation of convolution integrals and integral equations of the convolution type. Finally, the relationship between the finite Fourier transform and the Fourier series is discussed in order to apply the FFT to problems that involve harmonic analysis and synthesis.

Manuscript received January 26, 1967; revised March 6, 1967.
The authors are with the IBM Research Center, Yorktown Heights, N. Y.

Relationship between the Finite Fourier Transform and the Fourier Transform: Use of the Algorithm to Calculate Fourier Integrals

Suppose we have a function $x(t)$ which has a Fourier transform