The Fractional Fourier Transform and Applications

David H. Bailey and Paul N. Swarztrauber

October 22, 1990

Abstract

This paper describes the "fractional Fourier transform", which admits computation by an algorithm that has complexity proportional to the fast Fourier transform algorithm. Whereas the discrete Fourier transform (DFT) is based on integral roots of unity $e^{-2\pi i/n}$, the fractional Fourier transform is based on fractional roots of unity $e^{-2\pi i\alpha}$, where α is arbitrary. The fractional Fourier transform and the corresponding fast algorithm are useful for such applications as computing DFTs of sequences with prime lengths, computing DFTs of sparse sequences, analyzing sequences with non-integer periodicities, performing high-resolution trigonometric interpolation, detecting lines in noisy images and detecting signals with linearly drifting frequencies. In many cases, the resulting algorithms are faster by arbitrarily large factors than conventional techniques.

Bailey is with the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center, Moffett Field, CA 94035. Swarztrauber is with the National Center for Atmospheric Research, Boulder, CO 80307, which is sponsored by National Science Foundation. This work was completed while Swarztrauber was visiting the Research Institute for Advanced Computer Science (RIACS) at NASA Ames. Swarztrauber's work was funded by the NAS Systems Division via Cooperative Agreement NCC 2-387 between NASA and the Universities Space Research Association.

1. Introduction

The conventional fast Fourier transform (FFT) algorithm is widely used to compute discrete Fourier transforms (DFTs) and discrete convolutions, and to perform trigonometric interpolation. However, in some applications of the FFT, either the input is only partially nonzero, or only part of the DFT result is required, or both. Nonetheless, the FFT algorithm is ordinarily used unless the desired results can be more efficiently computed directly from the definition of the DFT. We present here a technique that permits many of these applications to be computed more efficiently. This same technique can also be applied in other situations that do not admit efficient solution using standard FFTs.

The central concept here is a generalization of the DFT that is termed the fractional Fourier transform (FRFT). It is defined on the m-long complex sequence $\boldsymbol{x} = (x_j,\ 0 \le j < m)$ as

$$G_k(\boldsymbol{x}, \alpha) = \sum_{j=0}^{m-1} x_j e^{-2\pi i jk\alpha} \tag{1}$$

The parameter α will not be restricted to rational numbers and in fact may be any complex number. Although this transform is defined for all integers k, we will usually compute the first m nonnegative values, i.e. for $0 \le k < m$. Straightforward evaluation of these m values using (1) requires $8m^2$ floating point operations, assuming the exponential factors have been precomputed.

Note that the ordinary DFT and its inverse are special cases of the fractional Fourier transform:

$$\begin{aligned}
F_k(\boldsymbol{x}) &= \sum_{j=0}^{m-1} x_j e^{-2\pi i jk/m} && (2)\\
&= G_k(\boldsymbol{x}, 1/m) \qquad 0 \le k < m && (3)\\
F_k^{-1}(\boldsymbol{x}) &= \frac{1}{m}\sum_{j=0}^{m-1} x_j e^{2\pi i jk/m} && (4)\\
&= \frac{1}{m} G_k(\boldsymbol{x}, -1/m) \qquad 0 \le k < m && (5)
\end{aligned}$$

The discrete Laplace transform can also be written in terms of the fractional Fourier transform.

If α is a rational number, the FRFT can be reduced to a DFT and can thus be evaluated using conventional FFTs. Suppose that $\alpha = r/n$, where the integers r and n are relatively prime and where $n \ge m$. Let p be the integer such that $0 \le p < n$ and $pr \equiv 1 \pmod{n}$. Extend the input sequence $\boldsymbol{x}$ to length n by padding with zeroes. Then

$$\begin{aligned}
G_k(\boldsymbol{x}, \alpha) &= \sum_{j=0}^{m-1} x_j e^{-2\pi i jkr/n} && (6)\\
&= \sum_{j=0}^{n-1} x_{pj} e^{-2\pi i (pj)kr/n} && (7)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j=0}^{n-1} x_{pj} e^{-2\pi ijk/n} && (8)\\
&= F_k(\boldsymbol{y}) \qquad\qquad 0 \le k < n && (9)
\end{aligned}
$$

where $\boldsymbol{y}$ is the n-long sequence defined by $y_j = x_{pj}$ and where subscripts are interpreted modulo n. Thus the first n values of the FRFT can be computed by performing an n-point FFT on the sequence $\boldsymbol{y}$. We will take $5n \log_2 n$ as the cost of this operation, since that is the number of floating point operations in a radix-2 FFT of size n.

Since only the first m values of this DFT are required, we will see in section 3 that the computational cost can be reduced to about $5n \log_2 m$ floating point operations by employing a decimation scheme.

2. The Fast Fractional Fourier Transform Algorithm

An impetus for studying the fractional Fourier transform is the existence of an algorithm for computing it that is significantly more efficient than the schemes described in the previous section. The computational cost of this algorithm is only about $20m \log_2 m$ floating point operations, which is independent of the value of α. In particular, this cost does not depend on whether or not α is rational or even real. The algorithm is based on a technique originally due to Bluestein [6] and is related to what is known in the signal processing field as the "chirp z-transform" (see [11] and [12]).

This algorithm can be derived by noting that $2jk = j^2 + k^2 - (k-j)^2$. The expression for the FRFT then becomes

$$
\begin{aligned}
G_k(\boldsymbol{x}, \alpha) &= \sum_{j=0}^{m-1} x_j e^{-\pi i[j^2+k^2-(k-j)^2]\alpha} && (10)\\
&= e^{-\pi i k^2 \alpha} \sum_{j=0}^{m-1} x_j e^{-\pi i j^2 \alpha} e^{\pi i (k-j)^2 \alpha} && (11)\\
&= e^{-\pi i k^2 \alpha} \sum_{j=0}^{m-1} y_j z_{k-j} && (12)
\end{aligned}
$$

where the m-long sequences $\boldsymbol{y}$ and $\boldsymbol{z}$ are defined by

$$
\begin{aligned}
y_j &= x_j e^{-\pi i j^2 \alpha} && (13)\\
z_j &= e^{\pi i j^2 \alpha} && (14)
\end{aligned}
$$

Since the summation (12) is in the form of a discrete convolution, it suggests evaluation using the well known DFT-based procedure. However, the usual DFT method evaluates circular convolutions, wherein $z_{k-j} = z_{k-j+m}$. This condition is not satisfied here, but instead $z_{k-j} = z_{j-k}$ when $k - j < 0$. Fortunately, it is possible to convert this summation into a form that is a circular convolution. Suppose we wish to compute the the FRFT for $0 \le k < m$. First select an integer $p \ge m - 1$, and extend the sequences $\boldsymbol{y}$ and $\boldsymbol{z}$ to length

$2p$ as follows:

$$y_j = 0 \qquad m \le j < 2p \tag{15}$$

$$z_j = 0 \qquad m \le j < 2p - m \tag{16}$$

$$z_j = e^{\pi i (j-2p)^2} \qquad 2p - m \le j < 2p \tag{17}$$

It can be seen by inspection that the first m values of $G_k(\boldsymbol{x}, \alpha)$ satisfy

$$G_k(\boldsymbol{x}, \alpha) = e^{-\pi i k^2 \alpha} \sum_{j=0}^{2p-1} y_j z_{k-j} \qquad 0 \le k < m \tag{18}$$

It can also be seen by inspection that the sequence $\boldsymbol{z}$ now satisfies the required property for a $2p$-point circular convolution. Thus it follows that $2p$-point DFTs may be used to evaluate (18):

$$G_k(\boldsymbol{x}, \alpha) = e^{-\pi i k^2 \alpha} F_k^{-1}(\boldsymbol{w}) \qquad 0 \le k < m \tag{19}$$

where $\boldsymbol{w}$ is the $2p$-long sequence defined by $w_k = F_k(\boldsymbol{y}) F_k(\boldsymbol{z})$. It should be emphasized that this equality only holds for $0 \le k < m$. The remaining $2p - m$ results of the final inverse DFT are discarded. These three DFTs can of course be efficiently computed using $2p$-point FFTs (for discussions of computing FFTs, see [1], [4], [5], [7], [9], [11], [16] and [17]).

To compute a different m-long segment $G_{k+s}(\boldsymbol{x}, \alpha)$, $0 \le k < m$, it is necessary to slightly modify the above convolution procedure. In this case $\boldsymbol{z}$ is as follows:

$$z_j = e^{\pi i (j+s)^2 \alpha} \qquad 0 \le j < m \tag{20}$$

$$z_j = 0 \qquad m \le j < 2p - m \tag{21}$$

$$z_j = e^{\pi i (j+s-2p)^2 \alpha} \qquad 2p - m \le j < 2p \tag{22}$$

The remainder of the algorithm is unchanged. This complete procedure will be referred to as the fast fractional Fourier transform algorithm.

The technique of converting the summation (12) into a circular convolution can also be understood as the embedding of a Toeplitz matrix into a larger circulant matrix, which admits evaluation using an FFT. Readers who wish to study the matrix formulation of this algorithm are referred to [15].

We will assume that $5q \log_2 q$ floating point operations are required for a q-point FFT. This is the cost of the commonly used radix-2 complex FFT. If q is not a power of two, the cost is somewhat higher, depending on the factors of q. If m is a power of two, the obvious choice for p is $p = m$. Then the total computational cost of the above algorithm is $20m \log_2 m + 44m$ floating point operations, assuming that the exponential factors in (19) and the FFT of the $\boldsymbol{z}$ sequence have been precomputed. Note that this cost is approximately four times the cost of the usual m-point FFT.

The efficiency of the above algorithm depends on the efficiency of the underlying circular convolution algorithm used to evaluate (18). A variety of fast convolution algorithms exist in the literature (see [2] and [11]), and one of these may be more attractive than using FFTs, depending on the computer hardware and the application. Further, FFTs over fields other than the complex numbers (see [11]) may be profitably employed in some instances. However, for simplicity in assessing the computational costs of algorithms, we will assume in the following that ordinary complex FFTs are employed. We will also assume when assessing costs that the sizes of DFTs and FRFTs are powers of two, although the algorithms to be discussed are valid for any size. Hereafter only the dominant term of these computational costs will be listed, since lower-order terms are generally overshadowed by implementation details, especially for large sizes.

3. Computing Longer and Shorter Segments of Results

Formula (19) gives the first m values of the fractional Fourier transform. Multiple segments of m values can be computed even more efficiently. For example, suppose one wishes to compute the first r results of an FRFT, where $r = qm$. These can be efficiently computed by repeating formulas (19) through (22) for $s = 0,\ m,\ 2m,\ \cdots, (q-1)m$. In this case it is reasonable to assume that for each value of s, the DFT of the appropriate $\boldsymbol{z}$ sequence has been precomputed. Significantly, the DFT of the $\boldsymbol{y}$ sequence needs to be computed only once. Thus each segment of m results can be computed by multiplying the DFT of the $\boldsymbol{y}$ sequence by the precomputed DFT of the appropriate $\boldsymbol{z}$ sequence, performing an inverse $2p$-point FFT on the result, and multiplying by exponentials as in (19). The dominant cost of computing r results in this manner is $10r \log_2 m$ floating point operations (when r is large compared to m).

The discussion in the previous paragraph assumed that one is computing the first r values of the FRFT. However, any set of r results that is the union of m-long contiguous segments can be computed in this manner, and the computational cost is the same. It is not even necessary to assume that the starting indices s of the segments are multiples of m.

Segments of the FRFT shorter than m in length can be efficiently computed by using a decimation scheme similar to that used for computing a short segment of a DFT. Since the fast algorithm for a short segment of a DFT is not widely known, it will be presented first.

Suppose that one wishes to compute the first m values of the DFT of the n-long sequence $\boldsymbol{x}$, where $n = qm$. We can write

$$F_k(\boldsymbol{x}) = \sum_{l=0}^{m-1} \sum_{j=0}^{q-1} x_{j+lq} e^{-2\pi i(j+lq)k/n} \tag{23}$$

$$= \sum_{j=0}^{q-1} e^{-2\pi ijk/n} \sum_{l=0}^{m-1} x_{j+lq} e^{-2\pi ilk/m} \tag{24}$$

$$= \sum_{j=0}^{q-1} e^{-2\pi ijk/n} F_k(\boldsymbol{y}_j) \qquad 0 \le k < m \tag{25}$$

where $\boldsymbol{y}_j$ denotes the m-long sequence $(x_{j+lq},\ 0 \le l < m)$. This means that the first m

values of the n-point DFT of $\boldsymbol{x}$ can be computed by writing the input as a $q \times m$ matrix in column major order, performing m-point FFTs on each row, multiplying the results by certain exponentials and then summing each column. The dominant computational cost of this procedure is $5n \log_2 m$ floating point operations, which is less than that of computing a full-sized n-point FFT. Unfortunately, this savings is not significant in the important special case $m = n/2$, a case we have already encountered: recall that only the first half of the DFT results in (19) are required.

Formula (25) may be easily generalized to compute m values of the DFT starting at any index s. One then obtains

$$F_{k+s}(\boldsymbol{x}) = \sum_{j=0}^{q-1} e^{-2\pi i j(k+s)/n} F_k(\boldsymbol{z}_j) \qquad 0 \le k < m \tag{26}$$

where $\boldsymbol{z}_j = (x_{j+lq} e^{-2\pi i l s/m},\ 0 \le l < m)$.

The scheme described by equation (25) can be seen as an abbreviated form of an FFT algorithm sometimes called the "four step FFT" (see [1] p. 150, [4], [9] p. 569, and [17] p. 202 - 203). This FFT algorithm has received renewed interest lately due its suitability for some advanced computer architectures, particularly systems with hierarchical memories. Formula (25) is also related to what is known as the "discrete Zak transform" [3].

This same general method can be applied to compute an m-long segment of an n-point FRFT:

$$G_k(\boldsymbol{x}, \alpha) = \sum_{l=0}^{m-1} \sum_{j=0}^{q-1} x_{j+lq} e^{-2\pi i (j+lq)k\alpha} \tag{27}$$

$$= \sum_{j=0}^{q-1} e^{-2\pi i j k \alpha} \sum_{l=0}^{m-1} x_{j+lq} e^{-2\pi i l q k \alpha} \tag{28}$$

$$= \sum_{j=0}^{q-1} e^{-2\pi i j k \alpha} G_k(\boldsymbol{y}_j, q\alpha) \qquad 0 \le k < m \tag{29}$$

where $\boldsymbol{y}_j = (x_{j+lq},\ 0 \le l < m)$. As above, this means that the first m values of the n-point FRFT of $\boldsymbol{x}$ can be computed by writing the input as a $q \times m$ matrix in column major order, performing m-point FRFTs on each row, multiplying the results by certain exponentials and then summing each column. Note that the exponential factors in (29) may be incorporated into the exponential factors in formula (19) and thus do not need to be counted in the computational cost. Hence the dominant cost of this scheme is $20n \log_2 m$ floating operations, which is less than that of directly computing an n-point FRFT.

The corresponding formula for an m-long segment of an n-point FRFT beginning at some index s is

$$G_{k+s}(\boldsymbol{x}, \alpha) = \sum_{j=0}^{q-1} e^{-2\pi i j(k+s)\alpha} G_k(\boldsymbol{z}_j, q\alpha) \qquad 0 \le k < m \tag{30}$$

where $\boldsymbol{z}_j = (x_{j+lq} e^{-2\pi i l q s \alpha},\ 0 \le l < m)$. Note that the exponential factors in these $\boldsymbol{z}_j$ vectors can be incorporated into the factors in (20) and (22) and thus do not need to be counted. Hence the dominant cost of this scheme is $20n \log_2 m$ floating point operations, the same as above.

4. Computing DFTs of Sequences with Prime Lengths

It is popularly believed that DFTs can only be evaluated by a fast algorithm when the size m of the transform is a highly factorable integer, such as a power of two. But since the DFT is merely a special case of the FRFT, and since the size $2p$ of FFTs used in evaluating the FRFTs may be chosen to be any convenient value (such as a power of two) that is at least $2m - 2$, it follows that the fast fractional Fourier transform algorithm can be used to compute m-point DFTs even when m is a prime. The computational cost is only about four times the cost of a power of two FFT of comparable size. This algorithm was first presented in Bluestein's paper [6].

It should be emphasized that Bluestein's algorithm permits DFTs of any size (not just primes) to be computed using power of two FFTs, which are generally the most efficient FFTs available on computer system libraries. Thus depending on the computer system and implementation, Bluestein's algorithm may be more efficient for computing DFTs of certain sizes than conventional FFTs. On some computer systems, such as hypercubes, Bluestein's algorithm is preferable for computing DFTs of any size that is not a power of two [15].

Perhaps one reason that Bluestein's algorithm has not received more attention is that another FFT algorithm for prime m was discovered at about the same time by Rader [13]. This algorithm reduces a DFT of prime size m to a circular convolution of size $m - 1$, which in theory is only about half as costly as Bluestein's algorithm. Unlike Bluestein's algorithm, however, Rader's algorithm does not generalize to the case of arbitrary α, and there is little flexibility in the permissible sizes of FFTs that may be used for efficient evaluation.

There are a number of other important applications of the basic technique behind Bluestein's algorithm, which technique we have termed the FRFT. Several of these applications will be presented here, including some that were originally presented in [12].

5. Computing DFTs of Sparse Sequences

In this section we will describe efficient algorithms for computing DFTs of sequences with large numbers of zero elements. DFTs of such sequences arise in trigonometric interpolation, as we shall see in section 7. Another common application is in digital filtering, where high order harmonics are often zeroed before performing an inverse FFT. A third instance is in the computation of the "linear" or "aperiodic" convolution, which is defined on the n-long sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ as the $2n$-long sequence $\boldsymbol{z} = (\sum_{j=0}^{n-1} x_j y_{k-j},\ 0 \le k < 2n - 1)$, where the subscript $k - j$ is restricted to the range $0 \le k - j < n - 1$. Linear convolutions arise in applications as diverse as multiprecision arithmetic and the numerical solution of partial differential equations. This type of convolution is normally evaluated by padding the n-long input sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ with zeroes to length $2n$, and then computing their $2n$-point circular convolution using FFTs.

Three cases of sparse DFTs will be considered. First we will compute the complete DFT of an input sequence where only an initial segment is possibly nonzero. Second, we will compute segments of such a DFT. Third, we will compute segments of the DFT of an input sequence that has only a few nonzero segments.

Let $n = qm$. Suppose we wish to compute the DFT of an n-long sequence $\boldsymbol{x}$ that is known to be zero except for the first m entries. This can be done by employing a conventional FFT-based decimation scheme, as follows:

$$
\begin{aligned}
F_{k+lq}(\boldsymbol{x}) &= \sum_{j=0}^{m-1} x_j e^{-2\pi i j(k+lq)/n} && (31)\\
&= \sum_{j=0}^{m-1} x_j e^{-2\pi i jk/n} e^{-2\pi i jl/m} && (32)\\
&= F_l(\boldsymbol{y}_k) \qquad 0 \le l < m \quad 0 \le k < q && (33)
\end{aligned}
$$

where $\boldsymbol{y}_k = (x_j e^{-2\pi i jk/n},\ 0 \le j < m)$. Thus for each k, $0 \le k < q$, the m DFT values beginning at index k and increasing with stride q can be computed with a conventional m-point FFT on the input sequence multiplied by certain exponentials. The dominant cost of computing all n elements of the DFT by this procedure is $5n \log_2 m$ floating point operations, which is less than that of an n-point FFT. Unfortunately, this savings is not significant in the important special case $m = n/2$, a case we have already encountered: note that the sequence $\boldsymbol{y}$ defined in (13) and (15) is of this form.

If only a single contiguous segment of the n-point DFT is required, then this approach is unsatisfactory, since one can only obtain DFT results with a spacing of q from the above. Further, the conventional method for computing a segment of a DFT in section 3 is also unsatisfactory since each of the DFTs in the summation (26) must still be evaluated, with no savings in computation. However, the FRFT can be effectively applied to this situation.

The simplest case is when only m values of the n-point DFT are required. In this case we have

$$
\begin{aligned}
F_{k+s}(\boldsymbol{x}) &= \sum_{j=0}^{m-1} x_j e^{-2\pi i j(k+s)/n} && (34)\\
&= G_{k+s}(\boldsymbol{y}, 1/n) \qquad 0 \le k < m && (35)
\end{aligned}
$$

where $\boldsymbol{y} = (x_j,\ 0 \le j < m)$. The dominant cost here is the cost of an m-point FRFT, or $20m \log_2 m$ operations. This is less than the cost of conventional methods whenever $m < n/4$ (for sufficiently large n).

Multiple sections can be computed even more economically by merely repeating (35) for multiple values of s. As indicated in the first paragraph of section 3, savings result because the FFT of the sequence y in (18) needs to be computed only once. The dominant cost of computing r elements in this manner is $10r \log_2 m$ floating point operations (when

r is large compared to m). Thus up to $n/2$ elements of the DFT may be computed by this method, and the cost is still less than computing all or part of the n-point DFT by a conventional method (for large n).

We will now examine the case in which more than one m-long segment of the input sequence $\boldsymbol{x}$ is possibly nonzero. An m-long segment of the n-point DFT of $\boldsymbol{x}$ can be written

$$F_{k+s}(\boldsymbol{x}) = \sum_{l=0}^{q-1} \sum_{j=0}^{m-1} x_{j+lm} e^{-2\pi i(j+lm)(k+s)/n} \tag{36}$$

$$= \sum_{l=0}^{q-1} e^{-2\pi i(k+s)lm/n} \sum_{j=0}^{m-1} x_{j+lm} e^{-2\pi ij(k+s)/n} \tag{37}$$

$$= \sum_{l=0}^{q-1} e^{-2\pi i(k+s)l/q} G_{k+s}(\boldsymbol{y}_l, 1/n) \qquad 0 \le k < m \tag{38}$$

where $\boldsymbol{y}_l = (x_{j+lm},\ 0 \le j < m)$. With the assumption that only a few m-long input segments are zero, it follows that most of the terms of this summation are zero. Thus only a few of these FRFTs need to be actually computed.

We can assume that the exponential factors in (38) have been incorporated into the exponential factors of formula (19). Suppose that only t of the input segments $\boldsymbol{y}_l$ contain possibly nonzero elements. Then the total computational cost of computing an m-long segment of the DFT is on the order of $20tm\log_2 m$ operations. This is less than the cost of computing a segment of a DFT using the conventional scheme (33) whenever $t < q/4$ (for large n).

As indicated in section 3, multiple segments of results can be even more efficiently computed by repeating (38) for several values of s. The dominant cost of computing r values of the DFT in this way is $10tr\log_2 m$ floating point operations (when r is large compared to m). This is less than conventional methods whenever $tr < n/2$ (for large n). Thus this method is not appropriate unless only a few segments of the input are possibly nonzero and only a few segments of the result are desired.

6. Analyzing Sequences with Non-Integer Periodic Components

In many practical applications of DFTs and FFTs, notably in signal processing, it is not really true that the underlying time series being studied is exactly periodic with period m, where m is the size of the data sequence. Instead, the sampling interval and sample size are usually selected arbitrarily for the convenience of hardware and software. Thus in such applications it is the rule rather than the exception that a data sequence contains components with periods that are not exactly whole numbers.

One instance of such an application is in computing solar positions. Daily solar altitude and declination angles are readily available from almanacs. Such positions are not periodic with period 365 days, but they are very nearly periodic with a fractional period of approximately 365.2422 days. In other words, the apparent motion of the sun has a fairly simple Fourier representation based on a period of 365.2422 days, but the representation is not simple based on a period of 365 days. In this example we know from *a priori* information what the true period is. But it general we usually do not even know that. Thus we seek techniques that can determine an unknown period and compute the spectrum based on the true period once it is known.

Let $\boldsymbol{x}$ be the m-long sequence $(e^{2\pi ij\beta/m},\ 0 \le j < m)$ where β is unknown and not an integer. Then the DFT of $\boldsymbol{x}$ is

$$
\begin{aligned}
F_k(\boldsymbol{x}) &= \sum_{j=0}^{m-1} e^{2\pi ij\beta/m} e^{-2\pi ijk/m} && (39)\\
&= \sum_{j=0}^{m-1} e^{2\pi ij(\beta-k)/m} && (40)\\
&= \frac{1-e^{2\pi i(\beta-k)}}{1-e^{2\pi i(\beta-k)/m}} && (41)\\
&= e^{\pi i(\beta-k)(m-1)/m}\frac{\sin[\pi(\beta-k)]}{\sin[\pi(\beta-k)/m]} \qquad 0 \le k < m && (42)
\end{aligned}
$$

Thus the spectrum of such a sequence is not a "spike" at a single index but instead is spread over all indices, with magnitudes that decline rapidly with increasing distance from β. It is fairly easy to determine the unknown frequency β to within one unit by examining these magnitudes. Suppose that this has been done and that b is the greatest integer less than β.

One way to determine β more accurately is to apply the following formula, which was brought to the authors' attention by Oscar Buneman of Stanford University:

$$
\beta = b + \frac{m}{\pi}\tan^{-1}\frac{\sin(\pi/m)}{\cos(\pi/m)+|F_b(\boldsymbol{x})|/|F_{b+1}(\boldsymbol{x})|} \qquad (43)
$$

The derivation of this formula is straightforward from (42). It is exact for a pure sinusoid signal with frequency β in the range $b < \beta < b+1$. In the presence of noise or higher order harmonics, its accuracy degrades, but it is nonetheless quite usable for many applications.

An alternate method, which is somewhat more robust in the presence of noise, is based on the fractional Fourier transform. In particular, the FRFT can be used to efficiently compute a high resolution spectrum in the interval $[b, b+1]$. The m spectra values starting at index b and increasing with interval δ are given by

$$
\begin{aligned}
F_{b+k\delta}(\boldsymbol{x}) &= \sum_{j=0}^{m-1} x_j e^{-2\pi ij(b+k\delta)/m} && (44)\\
&= \sum_{j=0}^{m-1} x_j e^{-2\pi ijb/m} e^{-2\pi ijk\delta/m} && (45)\\
&= G_k(\boldsymbol{y}, \delta/m) \qquad 0 \le k < m && (46)
\end{aligned}
$$

where $\boldsymbol{y} = (x_j e^{-2\pi ijb/m},\ 0 \le j < m)$. δ is typically chosen to be $1/\sqrt{m}$, which means that only the first $\sqrt{m}$ values of the FRFT need to be computed to span $[b, b+1]$. Thus the cost of this calculation can be further reduced by applying the techniques of section 3. Suppose that $F_{b+k\delta}(\boldsymbol{x})$ is the largest element of this high resolution spectrum. Then an accurate estimate of the true frequency β is given by $b + k\delta$.

Once an accurate value of β has been obtained using either of these two methods, one might ask if the spectrum can be adjusted in some sense so that the integer index b corresponds to the fractional frequency β. Ideally the adjusted spectrum of a monochromatic signal would have a "spike" at b and zeroes at all other indices. To that end, let b to be the greatest integer less than β as above, and let $\alpha = b/\beta$, so that α is slightly less than one. Then consider the formula

$$\begin{aligned} X_k &= \sum_{j=0}^{r-1} x_j e^{-2\pi ijk/(m\alpha)} && (47) \\ &= G_k(\boldsymbol{z}, \frac{1}{m\alpha}) \qquad 0 \le k < r && (48) \end{aligned}$$

where r is the nearest integer to $m\alpha$ and where $\boldsymbol{z}$ is the $\boldsymbol{x}$ vector truncated to r elements. For computational purposes, it may be preferable to merely set $\boldsymbol{z}$ equal to $\boldsymbol{x}$, except that $z_j = 0$ for $r \le j < m$.

When $r = m\alpha$ is an integer, we have

$$\begin{aligned} X_k &= \sum_{j=0}^{r-1} x_j e^{-2\pi ijk/r} && (49) \\ &= \sum_{j=0}^{r-1} e^{2\pi ij\beta/m} e^{-2\pi ijk/r} && (50) \\ &= \sum_{j=0}^{r-1} e^{-2\pi ij(b-k)/r} && (51) \\ &= r\delta(k, b) && (52) \end{aligned}$$

where $\delta(k, b) = 1$ if $k = b$ but zero otherwise. Thus in this case we have precisely the desired result.

When $m\alpha$ is not an integer, it is still true that $X_b = r$, but the off-peak values are not zero. However, their magnitude is nonetheless much smaller than r. To make this precise, let $s = r - m\alpha$, and note that $|s| \le 1/2$. Then we can write

$$\begin{aligned} |X_k| &= \left| \sum_{j=0}^{r-1} x_j e^{-2\pi ijk/(m\alpha)} \right| && (53) \\ &= \left| \sum_{j=0}^{r-1} e^{-2\pi ij(b-k)/(m\alpha)} \right| && (54) \\ &= \left| \frac{1 - e^{2\pi ir(b-k)/(m\alpha)}}{1 - e^{2\pi i(b-k)/(m\alpha)}} \right| && (55) \end{aligned}$$

$$= \left| \frac{1 - e^{2\pi i s(b-k)/(m\alpha)}}{1 - e^{2\pi i (b-k)/(m\alpha)}} \right| \tag{56}$$

$$= \left| \frac{\sin[\pi s(b-k)/(m\alpha)]}{\sin[\pi(b-k)/(m\alpha)]} \right| \tag{57}$$

When k is close to b, we find that to a first-order approximation $|X_k|$ is equal to $|s|$, which is bounded by $1/2$ and therefore quite small compared with $X_b = r$. When k is large (near r) and b is small, or the reverse, the situation is not quite as favorable, since the denominator of (57) might become small. However, we still find that the magnitudes of the off-peak spectral values X_k are bounded by $m/(\pi\beta)$ or $m/[\pi(m-\beta)]$, which are small compared with r provided that β is neither a very small nor a very large fraction of m.

If the original signal is not monochromatic with frequency β, but instead consists of a component with frequency β plus several harmonics of this frequency, these higher order harmonics will be evident in the adjusted spectrum at $k = 2b,\ 3b$, etc. In this case an analysis of off-peak amplitudes gives results that are somewhat larger than above but still quite small compared with r for transforms of reasonably large sizes. The formula (48) also gives good results with the addition of moderate Gaussian noise.

7. Performing High-Resolution Trigonometric Interpolation

The FFT is frequently used to perform trigonometric interpolation. For example, given a sequence $\boldsymbol{x}$ in tabular form, we may wish to determine approximate midpoint values $x_{j+1/2}$. In the previous sections we have used the "aliased" form of the DFT:

$$F_k(\boldsymbol{x}) = \sum_{j=0}^{m-1} x_j e^{-2\pi i jk/m} \qquad 0 \le k < m \tag{58}$$

In this section we will use the "unaliased" form of the DFT, which is defined on $\hat{\boldsymbol{x}} = (x_j,\ -m/2 \le j < m/2)$ as

$$\hat{F}_k(\hat{\boldsymbol{x}}) = \sum_{j=-m/2}^{m/2-1} x_j e^{-2\pi i jk/m} \qquad -m/2 \le k < m/2 \tag{59}$$

The inverse unaliased DFT is defined analogously. For most purposes, the unaliased form is not as convenient as the aliased form, because two different formulas are required for m even and m odd (we will assume in the following that m is even). Further, most FFT software for computing DFTs assumes the aliased form.

In discussions of trigonometric interpolation, however, the unaliased form is preferred. The problem is that although $\hat{F}_k(\hat{\boldsymbol{x}}) = F_k(\boldsymbol{x})$ for each k, their interpolated values may be different (depending on how they are defined), so that $\hat{F}_{k+1/2}(\hat{\boldsymbol{x}}) \neq F_{k+1/2}(\boldsymbol{x})$. This disagreement stems from the fact that the function $f_k(t) = e^{ikt}$ has an infinite number of alternate characterizations (or "aliases") on the points $t_j = 2\pi j/m$, namely $f_{k+m}(t) = e^{i(k+m)t}$ for all integers m. These functions are indistinguishable from one another on the points t_j. However, at intermediate or interpolated values $f_k(t)$ and $f_{k+m}(t)$ are different.

Therefore a choice of basis functions for the trigonometric expansion of a function $f(t)$ must be made. The most common choice is the set $(f_k(t) = e^{ikt},\ -m/2 \leq k < m/2)$. This set has minimum variance on $0 \leq t \leq 2\pi$ and thus ensures that the interpolated values will be as smooth a possible.

For computational purposes, the unaliased form can be easily reduced to the aliased form. One approach is to define m-long sequences $\boldsymbol{y}$ and $\boldsymbol{Y}$ such that $y_j = (-1)^j x_{j-m/2}$ for $0 \leq j < m$ and $Y_k = (-1)^k \hat{F}_{k-m/2}(\hat{\boldsymbol{x}})$ for $0 \leq k < m$. Then

$$\begin{aligned}
Y_k &= (-1)^k \hat{F}_{k-m/2}(\hat{\boldsymbol{x}}) && (60)\\
&= e^{-\pi i k} \sum_{j=-m/2}^{m/2-1} x_j e^{-2\pi i j(k-m/2)/m} && (61)\\
&= \sum_{j=0}^{m-1} x_{j-m/2} e^{-2\pi i(j-m/2)k/m} && (62)\\
&= \sum_{j=0}^{m-1} y_j e^{-2\pi i jk/m} && (63)\\
&= F_k(\boldsymbol{y}) \qquad 0 \leq k < m && (64)
\end{aligned}$$

Thus $\boldsymbol{Y}$ is the ordinary m-point DFT of $\boldsymbol{y}$. Alternately, one can define $y_j = x_j$ for $0 \leq j < m/2$ and $y_j = x_{j-m}$ for $m/2 \leq j < m$, and similarly define Y_k in terms of $\hat{F}_k$. Then again $\boldsymbol{Y}$ is the ordinary DFT of $\boldsymbol{y}$. Either way, FFT software designed to compute the usual m-point aliased DFTs can be readily employed to compute m-point unaliased DFTs.

We will now develop the traditional approach to trigonometric interpolation. Suppose we wish to compute q interpolated values between each of the m values of $\hat{\boldsymbol{x}}$. Let $n = qm$. We first compute the inverse unaliased DFT of $\hat{\boldsymbol{x}}$ and then extend this spectra to the n-long sequence $\hat{\boldsymbol{Y}}$ as follows:

$$\begin{aligned}
Y_k &= \hat{F}_k^{-1}(\hat{\boldsymbol{x}}) && |k| < m/2 && (65)\\
Y_{-m/2} &= \frac{1}{2}\hat{F}_{-m/2}^{-1}(\hat{\boldsymbol{x}}) && && (66)\\
Y_{m/2} &= \frac{1}{2}\hat{F}_{-m/2}^{-1}(\hat{\boldsymbol{x}}) && && (67)\\
Y_k &= 0 && -n/2 \leq k < -m/2 && (68)\\
Y_k &= 0 && m/2 < k < n/2 && (69)
\end{aligned}$$

The splitting of $\hat{F}_{-m/2}^{-1}(\hat{\boldsymbol{x}})$ in the above is necessary to produce the most accurate interpolated values. For example, if this is not done then trigonometric interpolation on purely real data will yield complex values. This split is not necessary if m is odd.

The n-long vector $\hat{\boldsymbol{y}}$ of interpolated results $y_j = x_{j/q}$, where $n = qm$, can now be computed from the unaliased n-point DFT of $\hat{\boldsymbol{Y}}$:

$$y_j = \sum_{k=-m/2}^{m/2} Y_k e^{-2\pi i (j/q)k/m} \tag{70}$$

$$= \sum_{k=-n/2}^{n/2-1} Y_k e^{-2\pi ijk/n} \qquad -n/2 \le k < n/2 \tag{71}$$

This approach requires an m-point inverse FFT followed by an n-point forward FFT, so that the dominant cost is $5n \log_2 n$ floating point operations.

This cost can be reduced somewhat by applying a decimation technique similar to that employed in equations (31) through (33):

$$y_{j+lq} = \sum_{k=-m/2}^{m/2} Y_k e^{-2\pi ik(j+lq)/n} \tag{72}$$

$$= \sum_{k=-m/2}^{m/2} Y_k e^{-2\pi ikj/n} e^{-2\pi iklq/n} \tag{73}$$

$$= \sum_{k=-m/2}^{m/2-1} Y_k e^{-2\pi ikj/n} e^{-2\pi iklq/n} + Y_{m/2} e^{-\pi imj/n} e^{-\pi imlq/n} \tag{74}$$

$$= \hat{F}_l(\hat{\boldsymbol{Z}}_j) + (-1)^l Y_{m/2} e^{-\pi ij/q} \tag{75}$$

$$= \hat{F}_l(\hat{\boldsymbol{W}}_j) + (-1)^l 2 Y_{m/2} \cos(\pi j/q) \qquad -m/2 \le l < m/2 \quad 0 \le j < q \tag{76}$$

where $\hat{\boldsymbol{Z}}_j = (Y_k e^{-2\pi ikj/n},\ -m/2 \le k < m/2)$ and where $\hat{\boldsymbol{W}}_j = \hat{\boldsymbol{Z}}_j$ except that $(\hat{\boldsymbol{W}}_j)_{-m/2}$, the leftmost element of $\hat{\boldsymbol{W}}_j$, is zero. The formulas (75) and (76) differ only in that the second is more appropriate for working with real data, since a real-valued FFT may be used for computation. The dominant cost of computing all n interpolated values in this fashion is $5n \log_2 m$ floating point operations.

In many applications of trigonometric interpolation, only a segment of the n interpolated values are required. In this case the cost can be significantly reduced by applying the FRFT, using techniques similar to those in section 5. Suppose we wish to compute only the first m interpolated values in (71). This expression can be converted to a FRFT as follows:

$$y_j = \sum_{k=0}^{m} Y_{k-m/2} e^{-2\pi i(k-m/2)j/n} \tag{77}$$

$$= e^{\pi ij/q} \sum_{k=0}^{m} Y_{k-m/2} e^{-2\pi ikj/n} \tag{78}$$

$$= e^{\pi ij/q} G_j(\boldsymbol{Z}, 1/n) \qquad 0 \le j < m \tag{79}$$

where $\boldsymbol{Z}$ is the $(m+1)$-long sequence $(Y_{k-m/2},\ 0 \le k \le m)$.

Suppose for a minute that m is a power of two, and that the most efficient FFT sizes employed for evaluating the FRFT on a given computer are also powers of two. It might at first appear that increasing the size of an FRFT from m to $m+1$ would require that the size of FFTs used in computing the FRFT must be doubled to the next highest power of two, with a corresponding sharp increase in computational cost. However, recall that the parameter p of (18) can be any integer greater than one less than the size of the FRFT. Thus $2m$-point FFTs can still be used to evaluate FRFTs of size $m+1$, and the computational cost is virtually unchanged.

More generally, we can compute an m-long segment $\boldsymbol{z}$ of interpolated values beginning with $z_0 = x_s$ and increasing with interval δ as follows:

$$
\begin{aligned}
z_j &= \sum_{k=-m/2}^{m/2} Y_k e^{-2\pi i k(s+j\delta)/m} && (80)\\
&= \sum_{k=0}^{m} Y_{k-m/2} e^{-2\pi i (k-m/2)(s+j\delta)/m} && (81)\\
&= e^{\pi i (s+j\delta)} \sum_{k=0}^{m} Y_{k-m/2} e^{-2\pi i k s/m} e^{-2\pi i j k \delta/m} && (82)\\
&= e^{\pi i (s+j\delta)} G_j(\boldsymbol{Z}, \delta/m) \qquad 0 \le j < m && (83)
\end{aligned}
$$

where $\boldsymbol{Z}$ is the $(m+1)$-long sequence $(Y_{k-m/2} e^{-2\pi i k s/m},\ 0 \le k \le m)$.

Since the exponential factors in (83) can be incorporated into the precomputed exponentials in (19), we do not need to count the cost of these multiplications. The dominant cost of the algorithm is thus only $25 m \log_2 m$ floating point operations. This is less than the cost of the conventional procedure in equations (75) and (76) whenever $m < n/4$ (for large n). Larger sections of results can be efficiently obtained by repeating (83) for different starting indices s as indicated in section 3.

8. Detecting Lines in Images and Detecting Signals with Linearly Drifting Frequencies

The authors first learned of the problem of detecting signals with linearly drifting frequencies in discussions with scientists associated with the Search for Extraterrestrial Intelligence (SETI) project at NASA Ames. In fact, signals with drifting frequencies will arise in any situation where there is acceleration in the direction between the emitter of the signal and the receiver. It often can be assumed that over the period of observation the frequency drift is linear, so we will make this assumption in the following discussion.

It will first be shown that the problem of detecting a signal with a linearly drifting frequency can be reduced to the problem of detecting a straight line in a noisy two dimensional array. It will then be shown that the line detection problem can be efficiently solved using the FRFT.

Suppose that $\boldsymbol{y}$ is a time series of length $n = qm$ that is the sum of a signal with a linearly drifting frequency plus Gaussian noise. We will assume that the frequency is less than m. Thus the time series can be written

$$y_j = ae^{2\pi ij(\omega+j\tau)/m} + bg_j \qquad 0 \le j < n \tag{84}$$

where a and b are scaling factors, and g_j are complex Gaussian deviates with unit variance. We will be interested in applications where is a is much less than b, or in other words where the amplitude of the signal is much less than the amplitude of the noise.

The usual approach to detecting and analyzing such signals is to consider the n-long time series as a $q \times m$ array in row major order. We first compute the m-point DFT of each row in this array (i.e. the DFT of successive contiguous m-long segments of $\boldsymbol{y}$), and then compute squared magnitudes of each resulting array entry (see [8] and [10]):

$$X_{j,k} = |F_k(\boldsymbol{z}_j)|^2 \qquad 0 \le j < q, \quad 0 \le k < m \tag{85}$$

where $\boldsymbol{z}_j = (y_{j+lm},\ 0 \le l < m)$. When the frequency of the signal is constant (i.e. when τ is zero), detection of the signal can be achieved by merely summing the columns of X to form an m-long vector. The "bin" containing the signal will then be evident as a statistically large entry in this sum vector.

When the frequency of the signal to be detected is not constant (i.e. when τ is nonzero), this method fails since the "bin" containing the signal is different from segment to segment, and thus no entry of the sum vector will contain a statistically large value. Nonetheless, the signal is evident in this array as a "line" with a certain slope, and thus in theory it is possible to detect such signals by computing sums in the X array over lines with a range of slopes α:

$$S_k(\alpha) = \sum_{j=0}^{q-1} X_{j,\mathrm{nint}(k+\alpha j)} \qquad 0 \le k < m \tag{86}$$

where nint denotes nearest integer. When k is close to ω and α is close to $2m\tau$, then $S_k(\alpha)$ has a statistically large value and detection occurs. Unfortunately, the direct computation of all these sums over a range of possible slopes α is very expensive [8].

Consider what appears at first to be an even more expensive calculation, namely computing sums of values interpolated along the rows of X. In particular, consider the array of sums

$$S_{r,k} = \sum_{j=0}^{q-1} X_{j,k+(\alpha+\delta r)j} \qquad 0 \le r < q, \quad 0 \le k < m \tag{87}$$

where the subscript $k + (\alpha + \delta r)j$ is interpreted literally as its fractional value, namely as the result of trigonometric interpolation along the rows of the X array. Except for the fractional subscripts, this corresponds to computing the sums $S_k(\alpha_r)$, where $\alpha_r = \alpha + \delta r$. The location (r, k) of the maximum value in the S array yields the unknown slope α_r and starting position of the detected line. From these values the desired drift rate and the initial frequency of the signal can be easily determined.

This array of sums can be efficiently computed using the FRFT as follows. Define $X_{j,k}$ for negative k so that $X_{j,k} = X_{j,k+m}$. Then let $(\hat{x}_{j,l},\ -m/2 \le l < m/2)$ be the unaliased DFT of the j-th row of X, i.e. of the vector $(X_{j,k},\ -m/2 \le k < m/2)$. Now set $x_{j,l} = \hat{x}_{j,l}$ for $-m/2 < l < m/2$, but $x_{j,-m/2} = x_{j,m/2} = \frac{1}{2}\hat{x}_{j,-m/2}$. As explained in section 7, splitting the "Nyquist" value of the DFT in this manner is necessary to obtain purely real results when interpolating in real data.

Now we can write

$$X_{j,k} = \sum_{l=-m/2}^{m/2} x_{j,l} e^{-2\pi ikl/m} \tag{88}$$

$$X_{j,k+(\alpha+\delta r)j} = \sum_{l=-m/2}^{m/2} x_{j,l} e^{-2\pi i[k+(\alpha+\delta r)j]l/m} \tag{89}$$

Thus

$$\begin{aligned} S_{r,k} &= \sum_{j=0}^{q-1} \sum_{l=-m/2}^{m/2} x_{j,l} e^{-2\pi i[k+(\alpha+\delta r)j]l/m} && (90) \\ &= \sum_{l=-m/2}^{m/2} \left[\sum_{j=0}^{q-1} x_{j,l} e^{-2\pi i\alpha jl/m} e^{-2\pi i\delta rjl/m} \right] e^{-2\pi ikl/m} && (91) \\ &= \sum_{l=-m/2}^{m/2} s_{r,l} e^{-2\pi ikl/m} && (92) \end{aligned}$$

where

$$\begin{aligned} s_{r,l} &= G_l(\boldsymbol{z}_l, \delta r/m) && (93) \\ \boldsymbol{z}_l &= (x_{j,l} e^{-2\pi i\alpha jl/m},\ 0 \le j < q) && (94) \end{aligned}$$

Hence this summation algorithm consists of performing inverse FFTs on rows of the X array, followed by FRFTs on the resulting columns (with certain exponential multipliers), followed by forward FFTs on the resulting rows. The computational cost of entire detection algorithm is approximately $10n \log_2 n$ floating point operations, or in other words only about twice the cost of an n-point complex FFT. For complete details, see [14].

9. Conclusions

We have described a number of applications of the FRFT and its associated fast algorithm. The flexibility and efficiency of these methods permits new solutions to a variety of problems that involve discrete Fourier analysis. In many cases, the resulting algorithms are faster than the conventional methods of solution that utilize ordinary FFTs. In fact, for sufficiently large problem sizes, these methods are often faster than the conventional algorithms by arbitrarily large factors.

Acknowledgments

The authors wish to thank Kent Cullers of NASA Ames and Izidor Gertner of the CUNY Graduate Center for informative discussions on this subject.

References

1. Agarwal, R. C., and Cooley, J. W., "Fourier Transform and Convolution Subroutines for the IBM 3090 Vector Facility", *IBM Journal of Research and Development*, vol. 30 (1986), p. 145 - 162.

2. Agarwal, R. C., and Cooley, J. W., "New Algorithms for Digital Convolution", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-25, no. 5 (1977), p. 392 - 410.

3. Auslander, L., Gertner, I. and Tolimieri, R., "The Discrete Zak Transform: Applications to Time-Frequency Analysis and Synthesis of Non-Stationary Signals", *IEEE Transactions on Acoustics, Speech and Signal Processing*, to appear.

4. Bailey, D. H., "FFTs in External or Hierarchical Memory", *Journal of Supercomputing*, vol. 4 (1990), p. 23 - 35.

5. Bailey, D. H., "A High-Performance FFT Algorithm for Vector Supercomputers", *International Journal of Supercomputer Applications* vol. 2 (1988), p. 82 - 87.

6. Bluestein, L. I., "A Linear Filtering Approach to the Computation of the Discrete Fourier Transform", *IEEE Transactions on Audio and Electroacoustics*, vol. AU-18, no. 4 (1970), p. 451 - 455.

7. Cooley, J. W., and Tukey, J. W., "An Algorithm for Machine Computation of Complex Fourier Series", *Mathematics of Computation*, vol. 19 (1965), p. 297 - 301.

8. Cullers, D. K., Linscott, I. R., and Oliver, B. M., "Signal Processing in SETI", *Communications of the ACM*, vol. 28 (1985), p. 1151 - 1163.

9. Gentleman, W. M., and Sande, G., "Fast Fourier Transforms – For Fun and Profit", *AFIPS Proceedings*, vol. 29 (1966), p. 563 - 578.

10. Gertner, I., "Optimal Detection and Separation of Chirp Signals", *Proceedings of ICASSP-90*, 1990, to appear.

11. Nussbaumer, H. J., *Fast Fourier Transform and Convolution Algorithms*, Springer-Verlag, New York, 1981.

12. Rabiner, L. R., Schafer, R. W., and Rader, C. M., "The Chirp z-Transform Algorithm and Its Application", *Bell System Technical Journal*, vol. 48 (1969), p. 1249 - 1292.

13. Rader, C. M., "Discrete Fourier Transforms When the Number of Data Samples Is Prime", *Proceedings of the IEEE*, vol. 56 (1968), p. 1107 - 1108.

14. Swarztrauber, P. N., and Bailey, D. H., "Efficient Detection a Continuous Wave Signal with a Linear Frequency Drift", submitted for publication.

15. Swarztrauber, P. N., Sweet, R. A., Briggs, W. L., Henson, V. E., and Otto, J., "Bluestein's FFT for Arbitrary N on the Hypercube", submitted for publication.

16. Swarztrauber, P. N., "FFT Algorithms for Vector Computers", *Parallel Computing*, 1 (1984), p. 45 - 63.

17. Swarztrauber, P. N., "Multiprocessor FFTs", *Parallel Computing*, vol. 5 (1987), p. 197 - 210.