SIAM J. SCI. COMPUT.
Vol. 17, No. 5, pp. 1150–1166, September 1996

008

ACCURACY OF THE DISCRETE FOURIER TRANSFORM AND THE FAST FOURIER TRANSFORM*

JAMES C. SCHATZMAN†

Abstract. Fast Fourier transform (FFT)-based computations can be far more accurate than the slow transforms suggest. Discrete Fourier transforms computed through the FFT are far more accurate than slow transforms, and convolutions computed via FFT are far more accurate than the direct results. However, these results depend critically on the accuracy of the FFT software employed, which should generally be considered suspect. Popular recursions for fast computation of the sine/cosine table (or twiddle factors) are inaccurate due to inherent instability. Some analyses of these recursions that have appeared heretofore in print, suggesting stability, are incorrect. Even in higher dimensions, the FFT is remarkably stable.

Key words. fast Fourier transform (FFT), discrete Fourier transform (DFT)

AMS subject classifications. 65T20, 65Y25, 68Q25, 42A65, 42C10

1. Introduction. The Fourier transform is one of the most important fundamental mathematical tools in existence today. While the discrete Fourier transform (DFT) and fast Fourier transform (FFT) are generally considered to be stable algorithms, reported quantifications of the stability have been inconsistent and confusing. Some analyses neglect the effect of errors in the coefficients (also called the sine/cosine table or twiddle factors), which turns out to be potentially the largest source of error.

Gentleman and Sande [4] report an analysis of FFT errors (for the Cooley–Tukey algorithm) in which the root mean square (RMS) relative error is

$$E_{RMS} = 1.06 \sum_{j=1}^{K} (2p_j)^{\frac{3}{2}} \epsilon, \tag{1}$$

where ϵ is the machine epsilon and $N = p_1 p_2 \ldots p_K$. For $N = 2^K$, and using the radix-2 algorithm, this becomes

$$E_{RMS} = 8.48 \log_2 N \epsilon. \tag{2}$$

Their corresponding formula for the slow DFT is

$$E_{RMS} = 1.06(2N)^{\frac{3}{2}} \epsilon. \tag{3}$$

The results of Gentleman and Sande are upper bounds. Our own *typical* results are quite different, being asymptotically of far better (smaller) order when the twiddle factors are accurate.

Kaneko and Liu [7] give an analysis for the Cooley–Tukey FFT with several types of input data sequences. Their results are rather complex, but their conclusion that errors in the twiddle factors have virtually no effect on the results (compare their Figures 4 and 7) is misleading.

Calvetti [2] gives an involved analysis for the Cooley–Tukey algorithm and the slow transform. She separates the effects of roundoff errors in addition and multiplication. The results are

*Received by the editors April 12, 1993; accepted for publication (in revised form) May 12, 1995.
†Department of Mathematics, University of Wyoming, P.O. Box 3036, Laramie, WY 82071 (jcs@uwyo.edu).
This research was supported partly by a University of Wyoming Faculty Grant-in-Aid.

$$
\begin{aligned}
E_{RMS} &= \sqrt{\log_2 N}\,\sigma_a, && \text{addition errors, FFT;}\\
E_{RMS} &= \frac{1}{2}\sqrt{\log_2 N}\,\sigma_a, && \text{multiplication errors, FFT;}\\
E_{RMS} &= \frac{\sqrt{n-1}}{n}\,\sigma_m, && \text{addition errors, slow DFT;}\\
E_{RMS} &= \frac{1}{n}\,\sigma_m, && \text{multiplication errors, slow DFT;}
\end{aligned}
\tag{4}
$$

where σ_a, σ_m are the standard deviations of the assumed independent random errors in addition and multiplication. These errors are relative to the maximum norm of the input data. Calvetti claims, "For very small expected value of the relative error for addition and multiplication, the traditional [slow] algorithm will produce more accurate results." She further suggests, "The FFT can be considered more accurate than the TFT [slow DFT] only if the expected value of the relative error for addition is of the same size or larger than the expected value of the relative error for multiplication." I do not believe that the analysis for the slow algorithm is correct, nor is the general sense of Calvetti's conclusion (that the slow DFT is accurate compared to the FFT) correct. I do agree with Calvetti's formulas for FFT errors, in the case where the twiddle factors are accurate. Calvetti's paper gives a useful bibliography of earlier work.

According to Gottlieb and Orszag [5], "Transform methods normally give no appreciable amplification of roundoff errors. In fact, the evaluation of convolution-like sums using FFTs often gives results with much smaller roundoff error than would be obtained if the convolution sums were evaluated directly." See Van Loan [12] for a more recent publication on this topic.

Our own conclusions concerning accuracy are that the FFT is *remarkably* stable, when implemented properly. The FFT is vastly more accurate than the slow DFT. However, the FFT is very sensitive to accuracy of the internal sine/cosine table (twiddle factors). A popular technique for calculating the sine/cosine table by recursion is ill-advised for this reason.

2. The DFT.

2.1. Properties of the standard DFT. The square DFT might be written $\vec{c} = G\vec{x}$ or

$$
c_k = \sum_{n=0}^{N-1} e^{-\imath\omega_k t_n} x_n, \tag{5}
$$

where most commonly

$$
\omega_k = k\Delta\omega, \qquad 0 \le k \le N\text{–}1, \tag{6}
$$

$$
t_n = n\Delta t, \qquad 0 \le n \le N\text{–}1, \tag{7}
$$

$$
\Delta\omega\Delta t = \frac{2\pi}{N}. \tag{8}
$$

2.1.1. Numerical errors. Errors in the computation of the DFT are limited to errors in the approximations of the sine/cosine phase factors and roundoff errors in multiplication and addition. To aid in characterizing these errors, we define the machine ϵ in the usual way as the smallest positive number such that $1 + \epsilon$ is distinguishable from unity in the floating point representation employed. For tests with 32- and 64-bit IEEE floating point, we use the corresponding $\epsilon_{32} = 6 \cdot 10^{-8}$ and $\epsilon_{64} = 1.1 \cdot 10^{-16}$.

A useful exact characterization of expected error with floating point calculations is difficult or impossible, particularly in the case of addition, because of the nature of the floating point

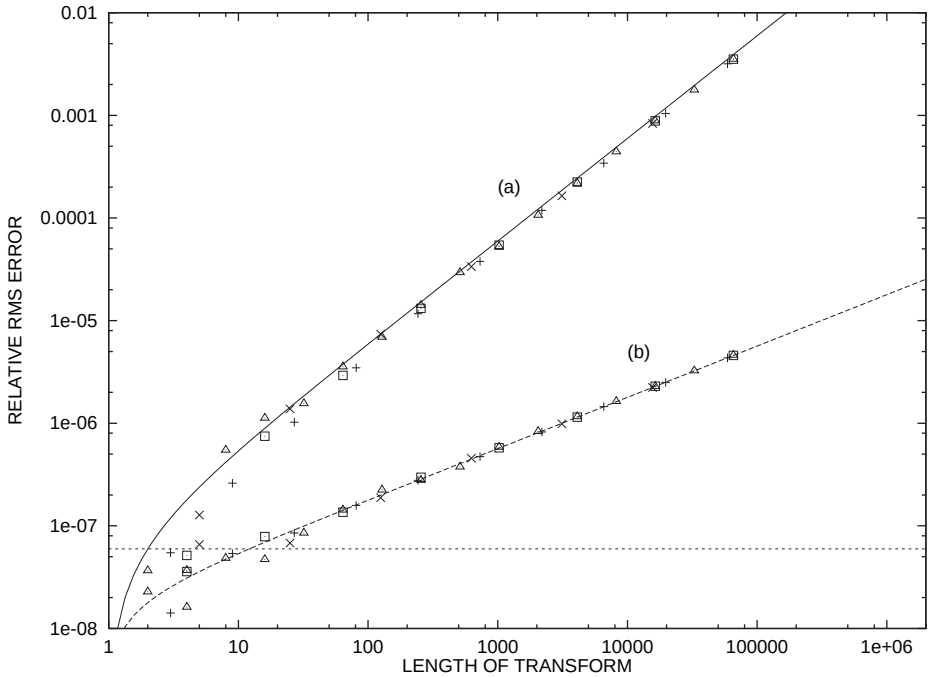

FIG. 1. *Relative RMS errors in the computed DFT for the* 32*-bit IEEE floating point slow Fourier transform using* (a) *the IBM* RS6000 *library* 32*-bit sine and cosine functions and the NCAR FFTPAK table calculation code, and* (b) *an accurate sine/cosine table.* $\triangle$, $\square$, $+$, *and* $\times$ *denote lengths that are powers of* 2, 3, 4, *and* 5, *respectively. The interpolating error functions (smooth curves) are* $e_1(N) = \epsilon_{32}(N-1)$ *and* $e_2(N) = 0.3\epsilon_{32}\sqrt{N-1}$. *The horizontal dashed line represents the constant* $\epsilon_{32} = 5.96 \cdot 10^{-8}$.

representation. Error bounds tend to lead to excessively pessimistic error estimates. Roughly, for data vectors exhibiting a reasonable degree of stationarity, if the errors associated with each coefficient, multiplication, and addition are uncorrelated (not a very reasonable assumption) the RMS error of the DFT would be expected to be approximately $\sqrt{N-1}$ times the error standard deviation associated with each term. If the errors are correlated, an RMS error proportional to $N-1$ would be expected. Assuming that relative multiplication errors are uniformly distributed on $[-\epsilon/2, \epsilon/2]$ (a reasonable approximation if full rounding is used), the corresponding error standard deviation is $\epsilon/\sqrt{12}$. Assuming that relative addition errors are uniformly distributed on $[-\epsilon, \epsilon]$ (a poor approximation), the corresponding error standard deviation is $\epsilon/\sqrt{3}$.

Putting these ideas together, we obtain a rough estimate of the overall RMS error in the case of uncorrelated individual errors:

$$E_{RMS} = \sqrt{(N-1)\left(\frac{1}{12} + \frac{1}{3}\right)\epsilon^2 + (N-1){\sigma_C}^2}, \tag{9}$$

where σ_C is the error standard deviation of the coefficients. If σ_C is $\epsilon/\sqrt{12}$, (9) reduces to $E_{RMS} = (\epsilon/\sqrt{2})\sqrt{N-1}$.

Figures 1 and 2 show a comparison of 128-bit floating point DFT results (taken to be truth) with

(1a) IEEE 32-bit DFT computations using the manufacturer's 32-bit sine/cosine functions,

(1b) IEEE 32-bit DFT computations modified to use an accurate table of phase factors,

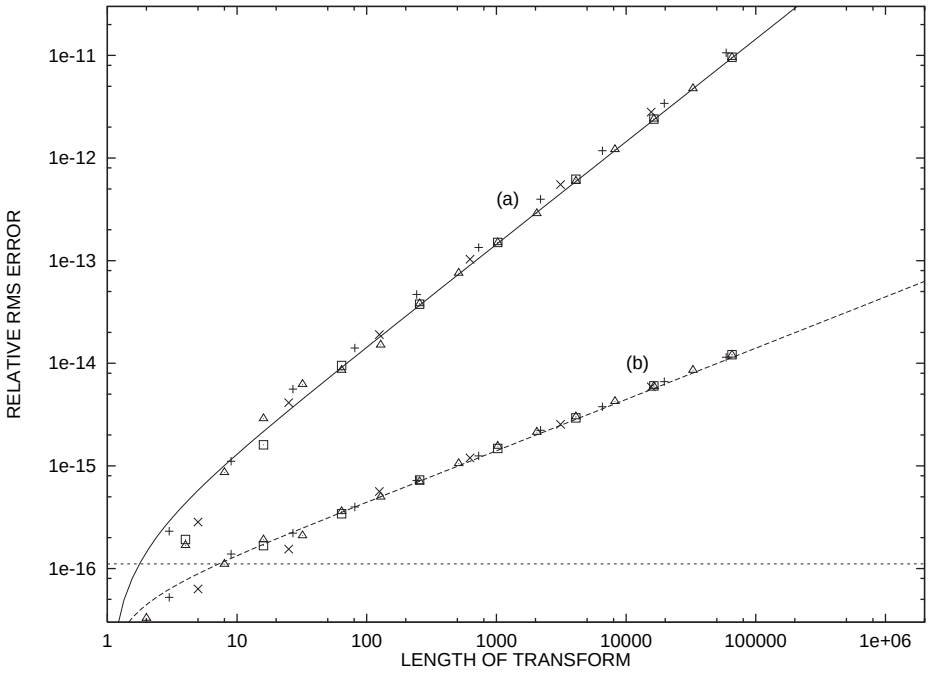

FIG. 2. *Relative RMS errors in the computed DFT for the* 64*-bit IEEE floating point slow Fourier transform using* (a) *the IBM RS6000 library* 64*-bit sine and cosine functions and the NCAR FFTPAK table calculation code, and* (b) *an accurate sine/cosine table.* $\triangle$, $+$, $\square$, *and* $\times$ *denote lengths that are powers of* 2, 3, 4, *and* 5, *respectively. The interpolating error functions (smooth curves) are* $e_1(N) = 1.3\epsilon_{64}(N-1)$ *and* $e_2(N) = 0.4\epsilon_{64}\sqrt{N-1}$. *The horizontal dashed line represents the constant* $\epsilon_{64} = 1.11 \cdot 10^{-16}$.

(2a) IEEE 64-bit DFT computations using the manufacturer's 64-bit sine/cosine functions,

(2b) IEEE 64-bit DFT computations modified to use an accurate table of phase factors.

The tests were performed on an IBM RS/6000 computer. The input data was a series of independent, identically distributed Gaussian random sequences (for the real and imaginary components). The RMS difference between the two computations, scaled by the RMS value of the "true" result, is displayed as a function of transform length.

As shown in the figures, fits to the measured RMS errors were obtained for the following error functions:

(1a) 32-bit: $\epsilon_{32}(N-1)$,

(1b) improved 32-bit: $0.3\epsilon_{32}\sqrt{N-1}$,

(2a) 64-bit: $1.3\epsilon_{64}(N-1)$,

(2b) improved 64-bit: $0.4\epsilon_{64}\sqrt{N-1}$.

When accurate phase factors are used (Figures 1b and 2b) square root error growth is observed; results are obtained that are more accurate than predicted by (9) by about a factor of two. When inaccurate phase factors are used (Figures 1a and 2a), linear growth is observed, suggesting that individual errors are correlated.

From these results we conclude the following:

(1) Accuracy of the sine/cosine table is critical to overall performance of the numerical DFT. When the sine/cosine table of the FFT is inaccurate, errors in the DFT may be correlated and the overall RMS DFT errors are then roughly proportional to the length of the transform.

(2) The IBM RS/6000 library sine/cosine functions are not very accurate; we do not recommend their use except for noncritical applications. A similar observation has been made for the Cray CFT77 library routines. The computation of long slow DFTs with IEEE 32-bit floating point may be expected to be of questionable accuracy unless more accurate sine/cosine approximations are used. Using the RS/6000 library routines, only about five significant figures should be expected for about 100 points ranging down to two significant figures for about 100k points.

(3) When the sine/cosine table is accurate, performance is dramatically superior for both 32- and 64-bit computations, and the RMS DFT errors grow proportionally to the square root of the length of the transform. In this case, the DFT may be regarded as a stable process. However, there still may be significant error for very long transforms; five to six significant figures of accuracy may be expected for transforms in the 100k+ point range using 32-bit floating point.

There is obviously some sensitivity of the errors to the data. An extreme example is the case of data consisting of mostly zeros. The above results should be interpreted as typical rather than definitive.

3. The FFT.

3.1. Sources of error. It is helpful to list the theoretical sources of error:

(1) instability of the FFT computations associated with the factorization,

(2) instability of the underlying DFT blocks,

(3) errors in the sine/cosine table (twiddle factors),

(4) roundoff error in all of the computations, compounded randomly.

While all errors could be considered roundoff errors, the above breakdown is reasonable. In particular, the idea of approximating roundoff by an instability effect and a random effect is a powerful tool. The validity of this model for general numerical computations remains to be demonstrated; there is good agreement between theoretical and empirical results in this analysis.

We will show that sources (1) and (2) may be discounted (the associated computations are extremely stable). Sources (3) and (4) are the principal sources of error; in a properly designed code, source (4) will dominate.

3.2. FFT formulation. We formulate the conventional version of the FFT to concretize the differences between it and the slow DFT. Suppose N is even. We observe

$$c_k = \sum_{j=0}^{N-1} w^{kn} x_j = \sum_{j=0}^{\frac{N}{2}-1} w^{2kj} x_{2j} + w^k \sum_{j=0}^{\frac{N}{2}-1} w^{2kj} x_{2j+1}$$

$$= \begin{cases} c_k^{(0)} + w^k c_k^{(1)}, & 0 \le k \le \frac{N}{2} - 1, \\ c_{k'}^{(0)} - w^{k'} c_{k'}^{(1)}, & \frac{N}{2} \le k \le N-1, \end{cases} \tag{10}$$

where $k' = k - \frac{N}{2}$, $w = e^{-\iota \frac{2\pi}{N}}$, $\vec{c}^{\,(0)}$ is the length $\frac{N}{2}$ DFT of the even terms $x_0, x_2, \ldots$, and $\vec{c}^{\,(1)}$ is the length $\frac{N}{2}$ DFT of the odd terms $x_1, x_3, \ldots$. This is the well-known result that a DFT of even length N can be computed as the combination of two DFTs of length $\frac{N}{2}$. The coefficients w^i and $w^{i'}$ are called the twiddle factors. In matrix notation (10) may be written

$$\vec{c} = \begin{pmatrix} I_{\frac{N}{2}} & \Omega^{\frac{1}{2}}_{\frac{N}{2}} \\ I_{\frac{N}{2}} & -\Omega^{\frac{1}{2}}_{\frac{N}{2}} \end{pmatrix} \begin{pmatrix} \vec{c}^{\,(0)} \\ \vec{c}^{\,(1)} \end{pmatrix}, \tag{11}$$

where

$$\Omega_q = \mathrm{diag}(z^0, z^1, z^2, \ldots, z^{q-1}), \qquad z = e^{-i\frac{2\pi}{q}}, \tag{12}$$

and $I_{\frac{N}{2}}$ is the $\frac{N}{2}$-by-$\frac{N}{2}$ identity matrix. Repeating this process for N a power of 2, we obtain

$$F_N = S_N^{(2)} \begin{pmatrix} S_{\frac{N}{2}}^{(2)} & \bigcirc \\ \bigcirc & S_{\frac{N}{2}}^{(2)} \end{pmatrix} \begin{pmatrix} S_{\frac{N}{4}}^{(2)} & \bigcirc & \bigcirc & \bigcirc \\ \bigcirc & S_{\frac{N}{4}}^{(2)} & \bigcirc & \bigcirc \\ \bigcirc & \bigcirc & S_{\frac{N}{4}}^{(2)} & \bigcirc \\ \bigcirc & \bigcirc & \bigcirc & S_{\frac{N}{4}}^{(2)} \end{pmatrix}$$

$$\cdots \begin{pmatrix} S_2^{(2)} & \cdots & \bigcirc \\ \vdots & \ddots & \vdots \\ \bigcirc & \cdots & S_2^{(2)} \end{pmatrix} P_N^{(2,2,2,\ldots,2)}, \tag{13}$$

where

$$\begin{cases} S_m^{(2)} = \begin{pmatrix} I_{\frac{m}{2}} & \Omega_{\frac{m}{2}}^{\frac{1}{2}} \\ I_{\frac{m}{2}} & -\Omega_{\frac{m}{2}}^{\frac{1}{2}} \end{pmatrix}, \\ P_N^{(2,2,2,\ldots,2)} = \text{bit-reversal ordering permutation matrix.} \end{cases} \tag{14}$$

For more general N, if $N = p_1 p_2 \cdots p_K$, where p_j are natural numbers, the FFT formula may be written

$$F_N = S_N^{(p_1)} \begin{pmatrix} S_{\frac{N}{p_1}}^{(p_2)} & & \bigcirc \\ & \ddots & \\ \bigcirc & & S_{\frac{N}{p_1}}^{(p_2)} \end{pmatrix} \begin{pmatrix} S_{\frac{N}{(p_1 p_2)}}^{(p_3)} & & \bigcirc \\ & \ddots & \\ \bigcirc & & S_{\frac{N}{(p_1 p_2)}}^{(p_3)} \end{pmatrix}$$

$$\cdots \begin{pmatrix} S_{p_K}^{(p_K)} & & \bigcirc \\ & \ddots & \\ \bigcirc & & S_{p_K}^{(p_K)} \end{pmatrix} P_N^{(p_1, p_2, \ldots, p_K)}, \tag{15}$$

where

$$S_m^{(p)} = \begin{pmatrix} (F_p)_{1,1} I_{\frac{m}{p}} & (F_p)_{1,2} \Omega_{\frac{m}{p}}^{\frac{1}{p}} & \cdots & (F_p)_{1,p} \Omega_{\frac{m}{p}}^{\frac{p-1}{p}} \\ (F_p)_{2,1} I_{\frac{m}{p}} & (F_p)_{2,2} \Omega_{\frac{m}{p}}^{\frac{1}{p}} & \cdots & (F_p)_{2,p} \Omega_{\frac{m}{p}}^{\frac{p-1}{p}} \\ \vdots & \vdots & & \vdots \\ (F_p)_{p,1} I_{\frac{m}{p}} & (F_p)_{p,2} \Omega_{\frac{m}{p}}^{\frac{1}{p}} & \cdots & (F_p)_{p,p} \Omega_{\frac{m}{p}}^{\frac{p-1}{p}} \end{pmatrix} \tag{16}$$

$$= F_m^{(p)} \, Q_m^{(p)}.$$

Here $F_m^{(p)}$ is the m-by-m matrix constructed from the p-by-p DFT matrix F_p by replacing each element of F_p by the product of the element and the m/p-by-m/p identity matrix. Also,

$$Q_m^{(p)} = \operatorname{diag}\left(I_{\frac{m}{p}}, \Omega_{\frac{m}{p}}^{\frac{1}{p}}, \Omega_{\frac{m}{p}}^{\frac{2}{p}}, \ldots, \Omega_{\frac{m}{p}}^{\frac{p-1}{p}} \right) \tag{17}$$

is the diagonal matrix of twiddle factors, and $P_N^{(p_1,p_2,\ldots,p_K)}$ is the mixed-radix digit-reversal ordering permutation matrix, defined as

$$\begin{cases} P_N^{(p_1,p_2,\ldots,p_K)}{}_{j,k} = \delta_{j,k'}, \qquad 0 \le j,k \le N\text{–}1, \\ \qquad k' = k'(k) \\ \qquad\quad = d_K + p_K(d_{K-1} + p_{K-1}(d_{K-2} + \cdots + p_2 d_1)), \\ \qquad k = d_1 + p_1(d_2 + p_2(d_3 + p_3(d_4 + \cdots + p_{K-1}d_K))), \end{cases} \tag{18}$$

where the mixed-radix digits d_l satisfy $0 \le d_l \le p_l - 1$. The formula (15) represents the so-called mixed-radix time-decimation FFT algorithm.

A mathematically equivalent formula that leads to the frequency-decimation algorithm follows from the observation that

$$\begin{aligned} F_N &= \left(\tfrac{1}{N} F_N^*\right)^{-1} \\ &= N \left(P_N^{(p_1,p_2,\ldots,p_K)}\right)^{-1} \begin{pmatrix} S_{p_K}^{(p_K)*} & & \bigcirc \\ & \ddots & \\ \bigcirc & & S_{p_K}^{(p_K)*} \end{pmatrix}^{-1} \\ &\quad \cdots \begin{pmatrix} S_{\frac{N}{(p_1p_2)}}^{(p_3)\ *} & & \bigcirc \\ & \ddots & \\ \bigcirc & & S_{\frac{N}{(p_1p_2)}}^{(p_3)\ *} \end{pmatrix}^{-1} \\ &\quad \begin{pmatrix} S_{\frac{N}{p_1}}^{(p_2)*} & & \bigcirc \\ & \ddots & \\ \bigcirc & & S_{\frac{N}{p_1}}^{(p_2)*} \end{pmatrix}^{-1} \left(S_N^{(p_1)*}\right)^{-1}. \end{aligned} \tag{19}$$

Then

$$\left(P_N^{(p_1,p_2,\ldots,p_K)}\right)^{-1} = P_N^{(p_K,p_{K-1},\ldots,p_1)} \tag{20}$$

and

$$\left(S_m^{(p)*}\right)^{-1} = \frac{1}{p} R_m^{(p)}, \tag{21}$$

where

$$R_m^{(p)} = \begin{pmatrix} (F_p)_{1,1} I_{\frac{m}{p}} & (F_p)_{1,2} I_{\frac{m}{p}} & \cdots & (F_p)_{1,p} I_{\frac{m}{p}} \\ (F_p)_{2,1} \Omega_{\frac{m}{p}}^{\frac{1}{p}} & (F_p)_{2,2} \Omega_{\frac{m}{p}}^{\frac{1}{p}} & \cdots & (F_p)_{2,p} \Omega_{\frac{m}{p}}^{\frac{1}{p}} \\ \vdots & \vdots & \vdots & \vdots \\ (F_p)_{p,1} \Omega_{\frac{m}{p}}^{\frac{p-1}{p}} & (F_p)_{p,2} \Omega_{\frac{m}{p}}^{\frac{p-1}{p}} & \cdots & (F_p)_{p,p} \Omega_{\frac{m}{p}}^{\frac{p-1}{p}} \end{pmatrix} \tag{22}$$

$$= Q_m^{(p)} \; F_m^{(p)}$$

so that

$$F_N = P_N^{(p_K, p_{K-1}, \ldots, p_1)} \begin{pmatrix} R_{p_K}^{(p_K)} & & \bigcirc \\ & \ddots & \\ \bigcirc & & R_{p_K}^{(p_K)} \end{pmatrix} \tag{23}$$

$$\cdots \begin{pmatrix} R_{\frac{N}{(p_1 p_2)}}^{(p_3)} & & \bigcirc \\ & \ddots & \\ \bigcirc & & R_{\frac{N}{(p_1 p_2)}}^{(p_3)} \end{pmatrix} \begin{pmatrix} R_{\frac{N}{p_1}}^{(p_2)} & & \bigcirc \\ & \ddots & \\ \bigcirc & & R_{\frac{N}{p_1}}^{(p_2)} \end{pmatrix} R_N^{(p_1)}.$$

These formulas may be simplified by use of the Kronecker matrix product $\otimes$ which is defined so that $A \otimes B$ is the LI-by-MJ matrix

$$A \otimes B = \begin{pmatrix} A_{1,1} B & A_{1,2} B & \cdots & A_{1,M} B \\ A_{2,1} B & A_{2,2} B & \cdots & A_{2,M} B \\ \vdots & \vdots & & \vdots \\ A_{L,1} B & A_{L,2} B & \cdots & A_{L,M} B \end{pmatrix} \tag{24}$$

for an L-by-M matrix A and an I-by-J matrix B. Then

$$\begin{aligned} F_m^{(p)} &= F_p \otimes I_{\frac{m}{p}}, \\ S_m^{(p)} &= F_m^{(p)} \; Q_m^{(p)} \\ &= \left(F_p \otimes I_{\frac{m}{p}} \right) \; Q_m^{(p)}, \\ R_m^{(p)} &= Q_m^{(p)} \; F_m^{(p)} \\ &= Q_m^{(p)} \; \left(F_p \otimes I_{\frac{m}{p}} \right), \end{aligned} \tag{25}$$

and

$$
\begin{aligned}
F_N &= S_N^{(p_1)} \left(I_{p_1} \otimes S_{\frac{N}{p_1}}^{(p_2)}\right) \left(I_{p_1 p_2} \otimes S_{\frac{N}{(p_1 p_2)}}^{(p_3)}\right) \\
&\quad \cdots \left(I_{p_1 p_2 \ldots p_{K-1}} \otimes S_{p_K}^{(p_K)}\right) P_N^{(p_1, p_2, \ldots, p_K)} \\
&= \left(F_{p_1} \otimes I_{\frac{N}{p_1}}\right) Q_N^{(p_1)} \left\{I_{p_1} \otimes \left[\left(F_{p_2} \otimes I_{\frac{N}{p_1 p_2}}\right) Q_{\frac{N}{p_1}}^{(p_2)}\right]\right\} \\
&\quad \left\{I_{p_1 p_2} \otimes \left[\left(F_{p_3} \otimes I_{\frac{N}{p_1 p_2 p_3}}\right) Q_{\frac{N}{(p_1 p_2)}}^{(p_3)}\right]\right\} \\
&\quad \cdots \left\{I_{p_1 p_2 \ldots p_{K-1}} \otimes \left[F_{p_K} Q_{p_K}^{(p_K)}\right]\right\} P_N^{(p_1, p_2, \ldots, p_K)} \\
&= \left\{I_{q_1} \otimes \left[\left(F_{p_1} \otimes I_{m_2}\right) Q_{m_1}^{(p_1)}\right]\right\} \\
&\quad \left\{I_{q_2} \otimes \left[\left(F_{p_2} \otimes I_{m_3}\right) Q_{m_2}^{(p_2)}\right]\right\} \\
&\quad \left\{I_{q_3} \otimes \left[\left(F_{p_3} \otimes I_{m_4}\right) Q_{m_3}^{(p_3)}\right]\right\} \\
&\quad \cdots \left\{I_{q_{K-1}} \otimes \left[\left(F_{p_{K-1}} \otimes I_{m_K}\right) Q_{m_{K-1}}^{(p_{K-1})}\right]\right\} \\
&\quad \left\{I_{q_K} \otimes \left[\left(F_{p_K} \otimes I_{m_{K+1}}\right) Q_{m_K}^{(p_K)}\right]\right\} P_N^{(p_1, p_2, \ldots, p_K)} \\
&= \prod_{j=1}^{K} \left\{I_{q_j} \otimes \left[\left(F_{p_j} \otimes I_{m_{j+1}}\right) Q_{m_j}^{(p_j)}\right]\right\} P_N^{(p_1, p_2, \ldots, p_K)} \qquad (26)
\end{aligned}
$$

for the time-decimation algorithm, and

$$
F_N = P_N^{(p_K, p_{K-1}, \ldots, p_1)} \prod_{j=K}^{1} \left\{I_{q_j} \otimes \left[Q_{m_j}^{(p_j)} \left(F_{p_j} \otimes I_{m_{j+1}}\right)\right]\right\} \qquad (27)
$$

for the frequency-decimation algorithm, where $m_j \equiv N/(p_1 p_2 p_3 \ldots p_{j-1})$ and $q_j \equiv p_1 p_2 \cdots p_{j-1}$.

A modification of (26) and (27) is possible where the mixed-radix permutation is distributed through the calculation as a series of two-factor permutations. The result is

$$
F_N = \prod_{j=1}^{K} A_j \left(P_{q_{j+1}}^{(q_j, p_j)} \otimes I_{m_{j+1}}\right) = \prod_{j=K}^{1} \left(P_{q_{j+1}}^{(p_j, q_j)} \otimes I_{m_{j+1}}\right) B_j, \qquad (28)
$$

where

$$
A_j = \left\{I_{q_j} \otimes \left[\left(F_{p_j} \otimes I_{m_{j+1}}\right) Q_{m_j}^{(p_j)}\right]\right\}, \qquad (29)
$$

$$
B_j = \left\{I_{q_j} \otimes \left[Q_{m_j}^{(p_j)} \left(F_{p_j} \otimes I_{m_{j+1}}\right)\right]\right\}. \qquad (30)
$$

More details are given in Schatzman [9]. We note that the permutations in (28) can be accomplished without an explicit permutation, but by appropriate indexing in each block

diagonal matrix-vector multiplication step

$$\sum_{l,m=0}^{N-1} (A_j)_{k,l} \left(P_{q_{j+1}}^{(q_j,p_j)} \otimes I_{m_{j+1}} \right)_{l,m} z_m = \sum_{m=0}^{N-1} (A_j)_{k,m'} z_m, \tag{31}$$

where the index mapping $m'(m)$ is computed according to

$$\begin{aligned} m' &= d_0 + d_2 m_{j+1} + d_1 m_{j+1} p_j, \\ m &= d_0 + d_1 m_{j+1} + d_2 m_{j+1} q_j, \end{aligned} \tag{32}$$

where the three-radix digits d_0, d_1, and d_2 satisfy $0 \le d_0 \le m_{j+1} - 1$, $0 \le d_1 \le q_j - 1$, and $0 \le d_2 \le p_j - 1$. This constitutes a block two radix reversed-digit mapping with a block size of m_{j+1}. These results are similar to those of Temperton [11].

Finally, the prime factor algorithm version of the FFT is nearly identical to the above, except that inputs and outputs are reordered, generally to effect the elimination of the twiddle factors (Burrus [1]; Schatzman [10]). Although the computations are reordered, and in some versions the underlying FFT blocks are modified by raising each element to a fixed integer power, the arguments above about stability of the stages of the algorithm still apply. The absence of twiddle factors could result in some improvement in accuracy; to this author's knowledge no results on this question have been published.

3.3. Stability of the FFT decomposition. If we examine (26)–(28) we see that the DFT is accomplished through a series of matrix-vector products. The permutation matrices have eigenvalues on the unit circle, with phases equal to multiples of $\frac{2\pi}{L}$, where L is the length of a permutation cycle associated with the matrix.

The matrices A_j and B_j are more complex. The matrix $F_{p_j} \otimes I_{m_{j+1}}$ has the same spectrum as F_{p_j}, namely eigenvalues $\pm\sqrt{p_j}$ and $\pm\imath\sqrt{p_j}$, but multiplied in multiplicity by the factor m_{j+1}. Likewise, the premultiplication $I_{q_j} \otimes C$ only multiplies the multiplicity of the eigenvalues of an arbitrary matrix C. Thus, the eigenvalues of A_j and B_j are modified only by the effect of postmultiplying and premultiplying $F_{p_j} \otimes I_{m_{j+1}}$ by $Q_{m_j}^{(p_j)}$, respectively. Because the matrices are unitary or scaled unitary, the eigenvalues of A_j and B_j all have magnitude $\sqrt{p_j}$; however, the phases of the eigenvalues are not the same as F_{p_j}. For example, the eigenvalues of $S_m^{(2)} = \left(F_2 \otimes I_{m/2}\right) Q_m^{(2)}$ are

$$\lambda_j = \frac{1}{2}\left(1 - w^j \pm \sqrt{(w^j - 1)^2 + 8w^j}\right), \qquad 0 \le j \le \frac{m}{2} - 1, \tag{33}$$

where $w = e^{-\imath 2\pi/m}$, which are irregularly distributed around the circle of radius $\sqrt{2}$ in the complex plane. From this elementary analysis, we see that no complications, such as increased sensitivity to numerical error due to increased condition number of the coefficient matrices, are to be expected from the FFT in any form. In fact, as we will see below, the FFT is very stable in most respects.

3.4. Numerical errors. Like the slow DFT, errors in the computation of the FFT consist of errors in the approximations of the sine/cosine phase factors and roundoff errors in multiplication and addition.

To test these predictions, slow DFTs were computed with single and double precision IEEE floating point. Figures 3 and 4 show a comparison of 128-bit floating point FFT results (taken to be truth) with

(3a) IEEE 32-bit FFT computations using the manufacturer's 32-bit sine/cosine functions and the FFTPAK recursion (see comments below),

(3b) IEEE 32-bit FFT computations modified to use accurate sine/cosine phase factors,

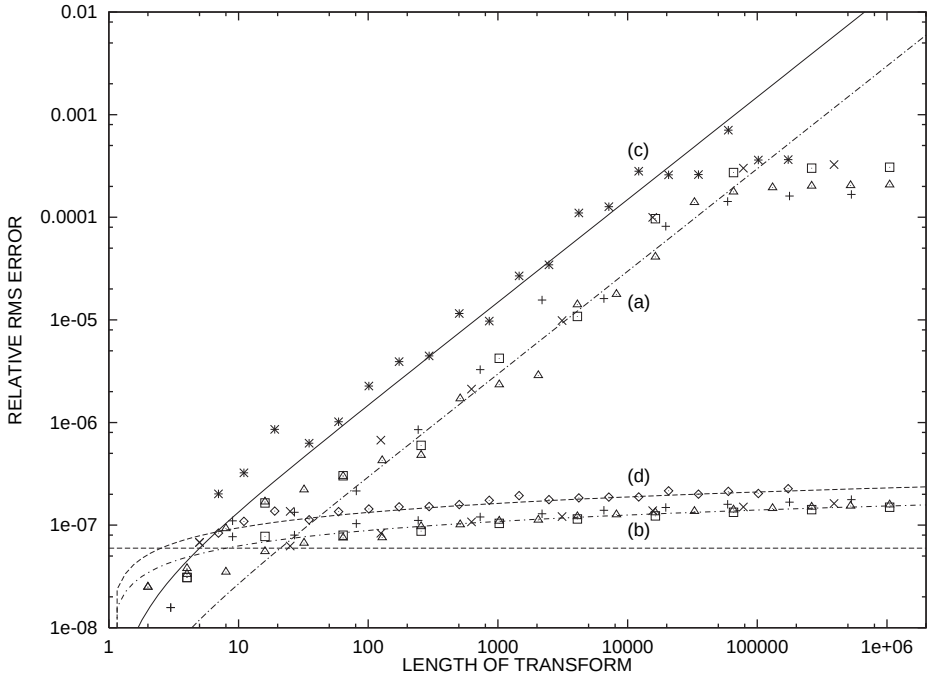

FIG. 3. *Relative RMS errors in the computed DFT for the* 32*-bit IEEE floating point FFT using* (a) *the IBM RS6000 library* 32*-bit sine and cosine functions and the NCAR FFTPAK table calculation code,* (b) *an accurate sine/cosine table,* (c) *the Schatzman algorithm* [9] *with inaccurate tables, and* (d) *the Schatzman algorithm* [9] *with accurate tables.* $\triangle, +, \square$, *and* $\times$ *denote lengths that are powers of* 2, 3, 4, *and* 5, *respectively;* $*$ *and* $\diamond$ *denote the results for the Schatzman algorithm* [9] *with inaccurate and accurate tables, respectively. The interpolating error functions (smooth curves) are* $e_1(N) = \frac{1}{10}\epsilon_{32}(N-1)$ *for the conventional FFT with inaccurate tables,* $e_2(N) = 0.6\epsilon_{32}\sqrt{\log_2 N}$ *with accurate tables,* $e_3(N) = \frac{1}{2}\epsilon_{32}(N-1)$ *for the Schatzman algorithm* [9] *with inaccurate tables, and* $e_4(N) = 0.9\epsilon_{32}\sqrt{\log_2 N}$ *with accurate tables. The horizontal dashed line represents the constant* $\epsilon_{32} = 5.96 \cdot 10^{-8}$.

(4a) IEEE 64-bit FFT computations using the manufacturer's 32-bit sine/cosine functions and the FFTPAK recursion (see comments below),

(4b) IEEE 64-bit FFT computations modified to use accurate sine/cosine phase factors.

The NCAR FFTPAK (Version 2, February 1978) FFT code was used as the basis for these results. However, the package was modified in several ways:

(1) Minor modifications to the code were made to bring it into conformance with the ANSI Fortran 77 standard.

(2) Power-of-two transforms were evaluated using $p = 2$, not a mix of $p = 2$ and $p = 4$ as in the original code. This change was made so that the performance of the $p = 2$ code could be measured independently of the $p = 4$ code.

(3) For the accurate table tests, the 32- and 64-bit sine/cosine table computations were replaced with 64- or 128-bit precision computations, respectively.

Input data and analysis of the results were as above for the slow DFT.

Fits to the measured RMS errors were obtained for the following error functions:

(3a) $\frac{1}{10}\epsilon_{32}(N-1)$,

(3b) $0.6\epsilon_{32}\sqrt{\log_2 N}$,

(4a) $\frac{1}{20}\epsilon_{64}(N-1)$,

(4b) $0.6\epsilon_{64}\sqrt{\log_2 N}$.

Also shown as Figures 3c, 3d, 4c, and 4d are results for the new algorithm of Schatzman [9] for prime lengths. Conclusions from these results follow:

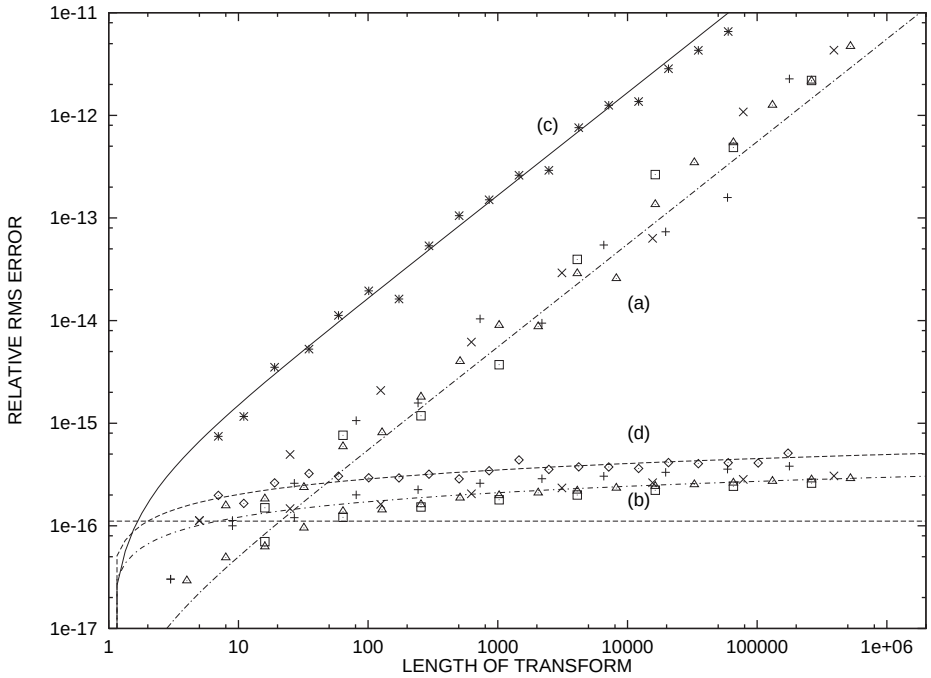

FIG. 4. *Relative RMS errors in the computed DFT for the* 64*-bit IEEE floating point FFT using* (a) *the IBM RS6000 library* 64*-bit sine and cosine functions and the NCAR FFTPAK table calculation code,* (b) *an accurate sine/cosine table,* (c) *the Schatzman algorithm* [9] *with inaccurate tables, and* (d) *the Schatzman algorithm* [9] *with accurate tables.* $\triangle$, $+$, $\square$, *and* $\times$ *denote lengths that are powers of* 2, 3, 4, *and* 5, *respectively;* $*$ *and* $\diamond$ *denote the results for the Schatzman algorithm* [9] *with inaccurate and accurate tables, respectively. The interpolating error functions (smooth curves) are* $e_1(N) = \frac{1}{20}\epsilon_{64}(N{-}1)$ *for the conventional FFT with inaccurate tables,* $e_2(N) = 0.6\epsilon_{64}\sqrt{\log_2 N}$ *with accurate tables, and* $e_3(N) = 1.5\epsilon_{64}(N{-}1)$ *for the Schatzman algorithm* [9] *with inaccurate tables. The horizontal dashed line represents the constant* $\epsilon_{64} = 1.11 \cdot 10^{-16}$.

(1) Comparing Figures 1a and 3a, 1b and 2b, etc., we conclude that *FFTs typically produce approximate DFTs that are more accurate than the slow algorithm by a factor of at least* 10.

(2) Accuracy of the sine/cosine phase factors are crucial to the overall accuracy of the DFT. However, in the NCAR FFTPAK code, only part of the table is computed by explicit sine and cosine evaluations. The remainder of the table is generated recursively. Replacing the explicit sine/cosine evaluations with extremely accurate calculations (128-bit) made very little difference. The resulting plots are nearly indistinguishable from Figures 3a and 4a. *The recursive evaluation of entries of the sine/cosine table, as implemented in the NCAR FFTPAK code, is by far the largest single source of error.* The particular recursion used in the NCAR FFTPAK accentuates the errors of the initial sine/cosine approximations.

(3) When the sine/cosine table of the FFT is inaccurate (as in Figures 3a and 4a) the RMS DFT errors are roughly proportional to the length of the transform.

(4) When the sine/cosine table is accurate (Figures 3b and 4b) the RMS DFT errors grow slowly, apparently more slowly than the logarithm of the length of the transform.

(5) The computation of long DFTs with IEEE 32-bit floating point is a process about which one should be cautious. If accuracy of better than 0.1% is required (three digits) for transforms in the 10,000+ point range, 64-bit floating point or a very carefully implemented 32-bit FFT should be used.

(6) The Schatzman [9] algorithm is less accurate than the conventional FFT, but only modestly so.

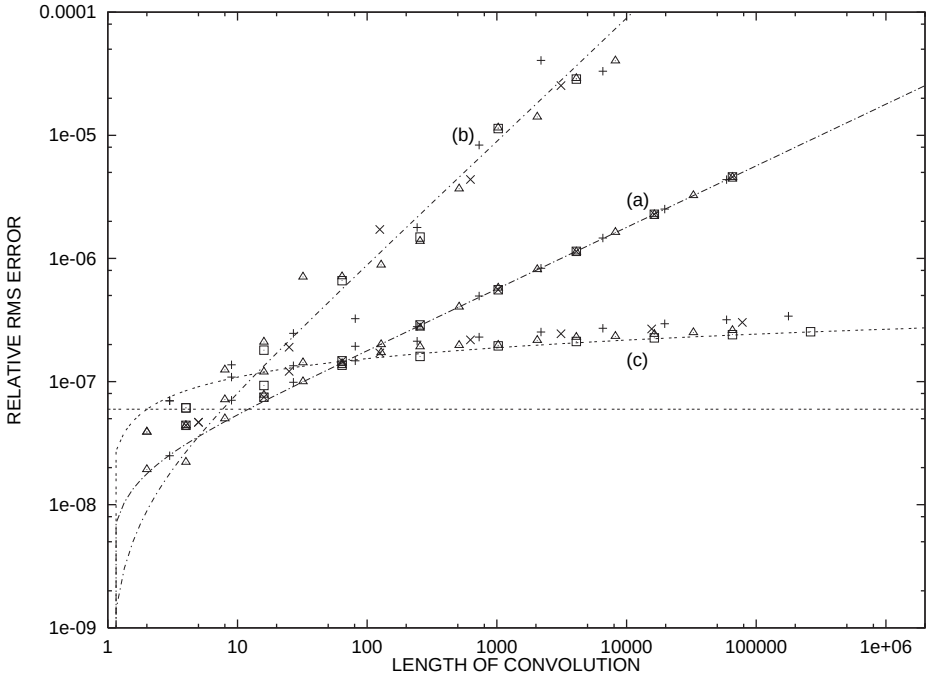

FIG. 5. *Relative RMS errors in the computed convolution for* (a) *the* 32*-bit IEEE floating point direct computation,* (b) *the FFT computation with inaccurate sine/cosine tables, and* (c) *with accurate tables.* $\triangle$, $+$, $\square$, *and* $\times$ *denote lengths that are powers of* 2, 3, 4, *and* 5, *respectively. The interpolating error functions (smooth curves) are* (a) $e_1(N) = 0.3\epsilon_{32}\sqrt{N-1}$, (b) $e_2(N) = 0.15\epsilon_{32}(N-1)$, *and* (c) $e_3(N) = \epsilon_{32}\sqrt{\log_2 N}$. *The horizontal dashed line represents the constant* $\epsilon_{32} = 5.96 \cdot 10^{-8}$.

4. Numerical errors for convolutions. We now study the computation of a circular convolution or cross-correlation via Fourier transform. Figures 5 and 6 show results for cross-correlation of white Gaussian complex sequences using direct computation and the comparable results using FFTs. We obtain fits to the data with the following functions:

(5a) Direct computation, 32-bit: $0.3\epsilon_{32}\sqrt{N-1}$;

(5b) FFT computation, 32-bit: $0.15\epsilon_{32}(N-1)$;

(5c) FFT computation, 32-bit with accurate sine/cosine table: $\epsilon_{32}\sqrt{\log_2 N}$;

(6a) Direct computation, 64-bit: $0.3\epsilon_{64}\sqrt{N-1}$;

(6b) FFT computation, 64-bit: $0.2\epsilon_{64}(N-1)$;

(6c) FFT computation, 64-bit with accurate sine/cosine table: $\epsilon_{64}\sqrt{\log_2 N}$.

These errors are very similar to the errors for the slow and fast DFT themselves, as reported above.

We recall the claim of Gottlieb and Orszag [5]: "Transform methods normally give no appreciable amplification of roundoff errors. In fact, the evaluation of convolution-like sums using FFTs often gives results with much smaller roundoff error than would be obtained if the convolution sums were evaluated directly." We have quantitatively confirmed these observations; the FFT method gives more accurate results for vector lengths of approximately 100 and greater with IEEE 32- and 64-bit floating point. However, this conclusion depends on having accurate FFT software. For example, the stock NCAR FFTPAK software gives less accurate results for the convolution than direct computation by arithmetic of the same precision. Asymptotically for large vector lengths, the RMS error is proportional to the square root of the logarithm of the vector length for the (accurate) FFT method and proportional to

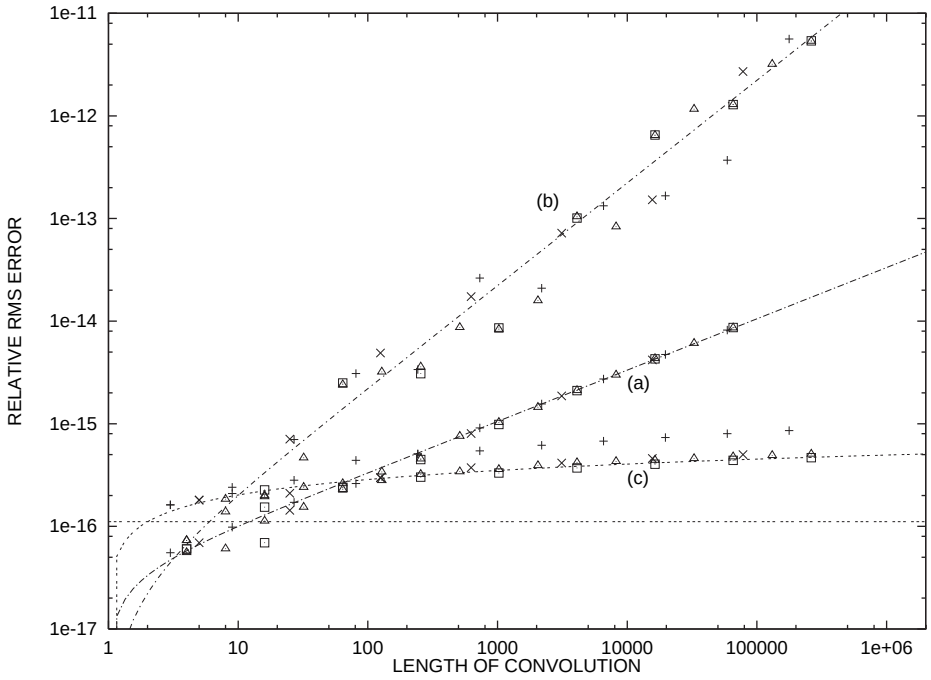

FIG. 6. *Relative RMS errors in the computed convolution for* (a) *the* 64*-bit IEEE floating point direct computation,* (b) *the FFT computation with inaccurate sine/cosine tables, and* (c) *with accurate tables. The interpolating error functions (smooth curves) are* (a) $e_1(N) = 0.3\epsilon_{64}\sqrt{N-1}$, (b) $e_2(N) = 0.2\epsilon_{64}(N-1)$, *and* (c) $e_3(N) = \epsilon_{64}\sqrt{\log_2 N}$. *The horizontal dashed line represents the constant* $\epsilon_{64} = 1.11 \cdot 10^{-16}$.

the square root of the vector length for the direct method. The explanation is that the FFT involves $\log(N)$ additions of uncorrelated errors for each component of the result, whereas the direct computation involves $N-1$ additions of uncorrelated errors.

5. Error in the twiddle factors. As observed above, the twiddle factors can be the dominant source of error. In some cases the library software for the sine/cosine is inaccurate. For example, on the Cray Y/MP, it appears that the decision was made to provide speed instead of accuracy. Accuracy of standard library routines should be tested; high-accuracy routines should be substituted if the library routines are inaccurate, keeping in mind that speed is generally irrelevant for this application (initiation of the FFT).

How to design algorithms that produce high-accuracy (one-half machine epsilon) sine/consine functions given floating point arithmetic of only machine epsilon accuracy is beyond the scope of this paper. I have found that it is difficult to obtain highly accurate sine and cosine functions using standard Taylor series or continued fraction expansions with floating hardware that does not provide double precision intermediary products. For example, the IBM RS/6000 processors provide high-accuracy (rounded) products whereas popular Intel and MIPS processors do not. After considerable experimentation, I offer the following conclusions and advice:

(1) Empirically, the error in accurate rounded 32- and 64-bit IEEE twiddle factor values as required by the FFT is essentially uniformly distributed on $[-\epsilon/2, \epsilon/2]$ for all n between 16 and 131,072. In other words, there is nothing peculiar about the twiddle factors that would upset this standard assumption. Therefore, the standard deviation of error in optimal twiddle factors is $\epsilon/\sqrt{12}$.

(2) The error obtained by direct library call to sine/cosine functions with the best libraries I tested is somewhat greater than the optimum—nearly 1.3–1.5 times the optimum. With other libraries the error can be much greater. Direct implementation of Taylor series leads to errors (with full-rounding floating point) of nearly 0.7ϵ. Kahan's summation method (Higham [6]) added to the Taylor series computation reduces the error to the theoretically optimal $\epsilon/\sqrt{12}$ or below.

(3) Errors resulting from the use of the recursion (34) are typically all of one sign (more about this below). The point is that the errors in the sine/consine table when computed this way are definitely not random (that is, not uniformly distributed over $[-\epsilon/2, \epsilon/2]$). A consequence is that overall FFT errors can be much larger than anticipated.

(4) If higher-precision arithmetic is available, by all means I recommend using that higher precision to compute the twiddle factors, which should then be rounded to the desired precision. Of course, arbitrary precision software is available (public domain) but the slowness may be objectionable.

(5) The sine/cosine tables could be pretabulated to high accuracy and recorded in binary form. Once this is done for a given floating point format, the code is portable to machines using that format. If implemented intelligently, the amount of disk space required for a complete tabulation for modestly large N is quite reasonable.

Some standard FFT packages use the following recursion for the sine and cosine functions:

$$\begin{cases} C_n = \cos\theta\ C_{n-1} - \sin\theta\ S_{n-1}, \\ S_n = \cos\theta\ S_{n-1} + \sin\theta\ C_{n-1} \end{cases} \tag{34}$$

for $n = 1, 2, \ldots, C_0 = \cos\theta_0$, and $S_0 = \sin\theta_0$. Replacing $\cos\theta$ by the approximate value c and $\sin\theta$ by the approximate value s, we obtain

$$\begin{pmatrix} C \\ S \end{pmatrix}_n = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} C \\ S \end{pmatrix}_{n-1}. \tag{35}$$

The eigenvalues of the coefficient matrix are $\lambda = c \pm \imath s$, and

$$\lambda^N = (c^2 + s^2)^{\frac{N}{2}} e^{\imath N \tan^{-1}\frac{s}{c}}. \tag{36}$$

Applying asymptotic analysis to (36), we see that errors in C_n and S_n grow at first linearly and later exponentially with the length of the recursion. This is highly undesirable; what is needed is a self-correcting recursion in which errors do not grow. While this analysis ignores roundoff error, actual calculations verify this exponential growth.

Chu [3] analyses the recursion (34), including the effects of roundoff, and concludes that errors grow linearly. This analysis is erroneous, resulting from ignoring errors in sines in the analysis of the cosines and vice versa. Oliver [8] gives an excellent review of recursions known at that time, but does not include (34). Chu [3] describes (34) and many other algorithms; some of the results of this paper are in error, however. Also, (34) can easily be improved as follows:

$$\begin{cases} A_n = \cos\theta\ C_{n-1} - \sin\theta\ S_{n-1} + \sin\theta\ \alpha, \\ B_n = \cos\theta\ S_{n-1} + \sin\theta\ C_{n-1}, \\ C_n = \frac{A_n}{\sqrt{A_n^2 + B_n^2}}, \\ S_n = \frac{B_n}{\sqrt{A_n^2 + B_n^2}}, \end{cases} \tag{37}$$

where α depends on N and the type of floating point hardware. Thus, we correct both the phase and amplitude of the sine/cosine pair at each step. This improved method is based on the empirical observations that the errors in (34) are patterned. However, the results of the best recursions are still significantly worse than accurate direct calculations.

My recommendation is that the twiddle factors be computed to high accuracy, either with high-precision arithmetic or Kahan's summation method (if Kahan's method works effectively on the given hardware). Truncated Taylor's series work very well for this purpose. If speed in initialization is required, depending on the machine architecture, either twiddle factors should be precomputed and stored on disk or faster converging approximations such as rational or continued fraction approximations should be used. Finally, if a recursive computation must be used for some reason, (34) should never be used but a more stable recursion (Oliver [8] and Chu [3]) should be employed.

6. Two and higher dimensions. I have not undertaken a thorough examination of the two- and higher-dimensional problem to verify that the one-dimensional results extend in a natural way. However, cursory examination of the problem suggests that higher-dimensional FFTs are even more remarkably stable, at least if the size of the dimensions is large and the number of dimensions is small.

For example, 1024-by-1024 transforms were executed with white random data using IEEE 64-bit floating point arithmetic. Using the manufacturer's sine/cosine functions and the usual recursions, RMS relative errors of $200\epsilon_{64}$ were observed. Using accurate sine/cosine tables, RMS relative errors of less than $3\epsilon_{64}$ were observed. These two-dimensional transforms were done in the usual way through repeated application of one-dimensional transforms along the rows and columns. Interestingly, the differences between the rows first/columns last and columns first/rows last results were observed to be approximately $4\epsilon_{64}$ regardless of the accuracy of the sine/cosine table. Finally, averaging the row/column and column/row results for the accurate sine/cosine table gave RMS relative errors of approximately $2\epsilon_{64}$. This is very little larger than the error for a one-dimensional transform of length 1024 ($1.9\epsilon_{64}$). It is remarkable that the transforms along the second dimension do not increase the error more than this. To lose only one bit of accuracy after thousands of floating point operations dramatizes the great stability of the FFT, when implemented carefully!

7. Conclusions. When an accurate sine/cosine table is used, the FFT is remarkably stable. However, the accuracy is greatly impaired by inaccurate sine/cosine tables. Inaccuracy in the sine/cosine table is common and results from inaccurate underlying software library functions and ill-advised recursions for table construction.

Acknowledgments. The anonymous reviewers were very helpful in drawing the author's attention to relevant literature.

REFERENCES

[1] C. S. BURRUS, *Index mappings for multidimensional formulation of the DFT convolution*, IEEE Trans. Acoustics, Speech, Signal Processing, 25 (1977), pp. 239–42.

[2] D. CALVETTI, *A stochastic roundoff error analysis for the fast Fourier transform*, Math. Comp., 56 (1991), pp. 755–774.

[3] C. Y. CHU, *The Fast Fourier Transform on Hypercube Parallel Computers*, Technical Report 87-882, Dept. of Computer Science, Cornell University, Ithaca, NY, 1987.

[4] W. M. GENTLEMAN AND G. SANDE, *Fast Fourier transforms–For fun and profit*, in 1966 Fall Joint Computer Conference, AFIPS Conf. Proceedings #29, Spartan, Washington, D.C., pp. 563–578.

[5] D. GOTTLIEB AND S. A. ORSZAG, *Numerical Analysis of Spectral Analysis: Theory and Applications*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[6] N. J. HIGHAM, *The accuracy of floating-point summation*, SIAM J. Sci. Comput., 14 (1993), pp. 783–799.

[7] T. KANEKO AND B. LIU, *Accumulation of round-off error in fast Fourier transforms*, J. Assoc. Comput. Mach., 17 (1970), pp. 637–654.
[8] J. OLIVER, *Stable methods for evaluating the points* $\cos(i\pi/n)$, J. Inst. Math. Applic., 16 (1975), pp. 247–257.
[9] J. SCHATZMAN, *Fast Fourier transform algorithms and complexity of the discrete Fourier transform*, Math. Comp., submitted.
[10] ———, *Index mappings for the fast Fourier transform*, IEEE Trans. Signal Processing, 44 (1996).
[11] C. TEMPERTON, *Self-sorting mixed-radix fast Fourier transforms*, J. Comput. Phys., 52 (1983), pp. 1–23.
[12] C. VAN LOAN, *Computational Frameworks for the Fast Fourier Transform*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.