PRE- AND POST-PROCESSING FOURIER TRANSFORM ALGORITHMS FOR REAL SYMMETRIC DATA USING A COMPLEX FFT

by

Agnes Anne O’Gallagher

B. A., University of Colorado, 1984

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Mathematics

1991
This thesis for the Master of Science

degree by

Agnes Anne O'Gallagher

has been approved for the

Department of

Mathematics

by

Roland A. Sweet

William L. Briggs

Thomas A. Manteuffel

Date 12/3/91
O’Gallagher, Agnes Anne (M.S., Mathematics)

Pre- and Post-Processing Fourier Transform Algorithms for Real Symmetric Data Using a Complex FFT

Thesis directed by Professor Roland A. Sweet

In this paper we explore FFT algorithms for symmetric data that use pre- and post-processing based on a complex FFT routine. The implementation of these routines in the new package VCFFTP is then discussed. Finally, the performance of these routines is compared to that of routines in VFFT-PAK which use pre- and post-processing based on a real routine. An attempt is made to discover how much is gained by avoiding extra passes through the data.

The form and content of this abstract are approved. I recommend its publication.

Signed Roland A. Sweet

Roland A. Sweet
Contents

Acknowledgements

1 Introduction ..................................................... 1

2 Some Properties of Symmetric Sequences and Their Fourier Transforms ........................................ 4

2.1 Symmetries .................................................... 4

2.2 Symmetries Produced by the Transform ..................... 4

2.2.1 Note on Odd Symmetric Sequences ....................... 7

2.3 Definition of the Transform in Trigonometric Form .... 7

2.4 Splitting Equations .......................................... 8

2.4.1 Splitting Equations for Real Data ...................... 9

3 The Algorithms .................................................. 10

3.1 The Real Algorithm ......................................... 13

3.2 The Sine Algorithm .......................................... 15
List of Figures

1 The combined symmetric algorithms replace the real FFT of the original symmetric algorithms with the three main steps of the real algorithm. ......................................................... 12
# List of Tables

1. A Symmetric Vector and Its Transform ........................................ 3
2. Definition of Pertinent Symmetries ........................................... 4
3. Symmetries of Transforms of Symmetric Sequences ...................... 7
4. Operation Counts for Vectors of Length $N = 2^p$ ....................... 34
5. The R Routines: VRCFTF vs VRFFT ........................................... 36
6. The O Routines: VSINC vs VSIINT ............................................ 38
7. The E Routines: VCOSC vs VCOST ............................................ 40
8. The Inverse QE Routines: VCSQCB vs VCOSQF ............................. 42
9. The Forward QE Routines: VCSQCF vs VCOSQB ............................. 43
10. The Inverse QO Routines: VSNQCB vs VSNQF ............................. 43
11. The Forward QO Routines: VSNQCF vs VSNQB ............................. 44
ACKNOWLEDGEMENTS

I wish to thank Roland A. Sweet for his teaching and guidance, William L. Briggs for his assistance and the generous contribution of his time, Paul N. Swarztrauber for his help in working out the algorithms and John M. Gary for his understanding and encouragement. I also wish to thank The National Center for Atmospheric Research and The National Institute of Standards and Technology for the use of their computing facilities and Kenneth J. Sewell, without whom this project could not have been done.
1 Introduction

A computational problem of great and widespread interest is the calculation of the discrete Fourier transform (DFT) of a vector. Its many applications provide the motivation for the development of efficient algorithms.

The forward DFT, \( \{X_k\} \), of a vector \( \{x_j\} \) of length \( N \) \([1]\) is defined as

\[
X_k = \frac{1}{N} \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi j k}{N}} \quad k = 0, 1, 2, \ldots N - 1
\]  

(1)

where \( i = \sqrt{-1} \). The inverse DFT is given by

\[
x_j = \sum_{k=0}^{N-1} X_k e^{\frac{2\pi j k}{N}} \quad j = 0, 1, 2, \ldots N - 1.
\]  

(2)

The factor of \( 1/N \) in equation (1) will be neglected for the remainder of this paper.

If calculated as defined, using a matrix-vector product, the DFT requires \( O(N^2) \) operations. The 1965 introduction by Cooley and Tukey\([2]\) of the Fast Fourier Transform (FFT) showed how this operation count can be reduced to \( O(N \log N) \). In 1970, the first algorithms designed for symmetric sequences \([3]\) were developed. These algorithms use a general FFT with pre- and post-processing.
In 1984, Swarztrauber [1], presented a set of pre- and post-processing algorithms for real, symmetric data that were based on a real, periodic transform. He implemented these in FFTPAK. A vectorized version of this package, VFFT-PAK, provided increased efficiency by adding the capability of processing many vectors at once. In the same paper, Swarztrauber showed how to use a complex routine with pre- and post-processing to produce the transform of real data. Although not described explicitly, another set of algorithms is implied which combines these two approaches. In this combined algorithm, one would use a complex transform with pre- and post-processing in place of the real transform used in FFTPAK and VFFT-PAK. It was thought that the performance of these combined algorithms could exceed that of those in VFFT-PAK because the use of the complex routine halves the length of the sequence and in doing so reduces the number of passes through the data for some cases.

It is these combined algorithms that will be discussed in this paper. As part of this project they have been implemented, in vectorized form, in a new package called VCFFTPBK.

In order to make the characteristics of symmetric sequences concrete let us take a specific case. Table 1 shows a vector of length $N = 8$ and its transform. The input vector, $\{x_j\}$, is real and odd symmetric ($x_{N-j} = -x_j$). Notice that
Table 1: A Symmetric Vector and Its Transform

<table>
<thead>
<tr>
<th>INPUT VECTOR</th>
<th>TRANSFORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>imaginary</td>
<td>imaginary</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>0.087</td>
<td>0.000</td>
</tr>
<tr>
<td>0.950</td>
<td>0.000</td>
</tr>
<tr>
<td>0.472</td>
<td>0.000</td>
</tr>
<tr>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>-0.472</td>
<td>0.000</td>
</tr>
<tr>
<td>-0.950</td>
<td>0.000</td>
</tr>
<tr>
<td>-0.087</td>
<td>0.000</td>
</tr>
</tbody>
</table>

The transform, \( \{X_k\} \), is imaginary and conjugate symmetric (\(X_{N-k} = \overline{X_k}\)). As will be shown, the transform of a real and odd vector will always exhibit these symmetries. They therefore require only a quarter of the computation that would be required for a general complex vector. Since such vectors often arise in applications, algorithms that exploit these symmetries are worthwhile.

Before the algorithms are presented some familiar properties of discrete Fourier transforms will be defined in order to provide a basis for the discussion.
2 Some Properties of Symmetric Sequences and Their Fourier Transforms

2.1 Symmetries

Several different symmetries are pertinent to the work that follows. For convenience they will be defined in Table 2. Each definition refers to a sequence \( \{x_j\} \) for \( j = 0, 1, 2, ..N - 1 \).

<table>
<thead>
<tr>
<th>SEQUENCE</th>
<th>ABBREV</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>R</td>
<td>( x_j = \overline{x}_j )</td>
</tr>
<tr>
<td>Imaginary</td>
<td>I</td>
<td>( x_j = -\overline{x}_j )</td>
</tr>
<tr>
<td>Even</td>
<td>E</td>
<td>( x_j = \overline{x}_{N-j} )</td>
</tr>
<tr>
<td>Odd</td>
<td>O</td>
<td>( x_j = -\overline{x}_{N-j} )</td>
</tr>
<tr>
<td>Quarter-wave even</td>
<td>QE</td>
<td>( x_j = \overline{x}_{N-j-1} )</td>
</tr>
<tr>
<td>Quarter-wave odd</td>
<td>QO</td>
<td>( x_j = -\overline{x}_{N-j-1} )</td>
</tr>
<tr>
<td>Conjugate Symmetric</td>
<td>CS</td>
<td>( x_j = \overline{x}_{N-j} )</td>
</tr>
</tbody>
</table>

2.2 Symmetries Produced by the Transform

It can be shown that the transforms of symmetric vectors themselves exhibit different symmetries. For example the transform of a real sequence is conjugate symmetric. To show this, begin with the definition of the forward

4
transform from equation (1)

\[ X_k = \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi jk}{N}}. \]

Substituting \( N - k \) for \( k \) in the above sum gives

\[
X_{N-k} = \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi j(N-k)}{N}}
= \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi jN}{N} e^{\frac{i2\pi jk}{N}}}
= \sum_{j=0}^{N-1} x_j e^{-i2\pi j} e^{\frac{i2\pi jk}{N}}.
\]

Using the fact that \( e^{-i2\pi j} = 1 \), we have that

\[
\overline{X}_{N-k} = \sum_{j=0}^{N-1} x_j e^{\frac{i2\pi jk}{N}}.
\]

If \( z = re^{i\theta} \) is a complex number, then \( \overline{z} = re^{-i\theta} \). Since \( \overline{x_j} = x_j \), it follows that

\[
\overline{X}_{N-k} = \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi jk}{N}} = X_k.
\]

As a second example, it will be shown that the transform of a real and odd vector is imaginary. As in the above case, we start with the definition of the forward discrete Fourier transform (1):

\[ X_k = \sum_{j=0}^{N-1} x_j e^{-\frac{i2\pi jk}{N}}. \]
Taking the conjugate of both sides and noting that $x_j = \bar{x}_j$ yields

\[
\overline{X}_k = \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi i jk}{N}}
\]
\[
= \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi i jk}{N}}
\]
\[
= \sum_{j=0}^{N-1} x_j e^{-\frac{i\omega jk}{N}}.
\]

Reversing the order of the summation gives

\[
\overline{X}_k = \sum_{j=0}^{N-1} x_{N-j} e^{-\frac{i\omega j(N-j)k}{N}}
\]
\[
= \sum_{j=0}^{N-1} x_{N-j} e^{i2\pi k} e^{-\frac{i\omega jk}{N}}
\]
\[
= \sum_{j=0}^{N-1} x_{N-j} e^{-\frac{i\omega jk}{N}}.
\]

Now since $x_j$ is odd, $x_j = -x_{N-j}$. This results in

\[
\overline{X}_k = \sum_{j=0}^{N-1} -x_j e^{-\frac{i\omega jk}{N}}
\]
\[
= -\sum_{j=0}^{N-1} x_j e^{-\frac{i\omega jk}{N}} = -X_k.
\]

Table 3 shows the definition of each of the symmetries that we will investigate and the symmetry displayed by the corresponding transform[1].

6
Table 3: Symmetries of Transforms of Symmetric Sequences

<table>
<thead>
<tr>
<th>INPUT VECTOR</th>
<th>TRANSFORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real</td>
<td>$X_k = X_{N-k}$</td>
</tr>
<tr>
<td>Real Even</td>
<td></td>
</tr>
<tr>
<td>Real Odd</td>
<td>$X_k = -\overline{X}_k$</td>
</tr>
<tr>
<td>Real Quarter-wave even</td>
<td>$X_k = e^{i\frac{k2\pi}{N}}X_k$</td>
</tr>
<tr>
<td>Real Quarter-wave odd</td>
<td>$X_k = -e^{i\frac{k2\pi}{N}}\overline{X}_k$</td>
</tr>
</tbody>
</table>

2.2.1 Note on Odd Symmetric Sequences

For an odd symmetric sequence $x_{N/2} = -x_{N-N/2} = -x_{N/2}$. Therefore $x_{N/2} = 0$. Also $x_N = -x_{N-N} = -x_0$. But by periodicity $x_N = x_0$. This means $x_N = x_0 = -x_0 \Rightarrow x_0 = 0$. So for any odd sequence $x_0 = x_{N/2} = 0$.

2.3 Definition of the Transform in Trigonometric Form

Let $\{x_j\}$ be a real sequence of length $N$ where $N$ is even. Then equation (2) for the inverse DFT can be written as

$$x_j = \sum_{k=0}^{N-1} X_k e^{\frac{j2\pi k}{N}} + \sum_{k=1}^{N/2} X_{N-k} e^{\frac{j2\pi (N-k)j}{N}}. \quad (3)$$

As shown above, for $\{x_j\}$ real, $\{X_k\}$ is conjugate symmetric. Therefore

$$x_j = X_0 + 2 \sum_{k=1}^{N/2-1} \Re \left\{ X_k e^{\frac{j2\pi k}{N}} \right\} + (-1)^j X_{N/2}. \quad (4)$$
Defining

\[ a_k = 2\Re(X_k) \]  \hspace{1cm} (5)  
\[ b_k = -2\Im(X_k), \]  \hspace{1cm} (6)  

where \( \Re(Z) \) and \( \Im(Z) \) signify the real and imaginary parts of \( Z \) respectively, (4) becomes

\[ x_j = \frac{a_0}{2} + \sum_{k=1}^{N-1} \left\{ a_k \cos\left(\frac{kj2\pi}{N}\right) + b_k \sin\left(\frac{kj2\pi}{N}\right) \right\} + (-1)^j\frac{a_{N/2}}{2}. \]  \hspace{1cm} (7)  

Using (2), (5), and (6) it can be shown that

\[ a_k = 2 \sum_{j=0}^{N-1} x_j \cos\left(\frac{jk2\pi}{N}\right) \quad k = 0, 1, 2, ...N/2, \]  \hspace{1cm} (8)  
\[ b_k = 2 \sum_{j=0}^{N-1} x_j \sin\left(\frac{jk2\pi}{N}\right) \quad k = 1, 2, 3, ...N/2-1. \]  \hspace{1cm} (9)  

The equations (7), (8), and (9) comprise the trigonometric, or real, form of the transform for a real sequence, while equations (5) and (6) define the relationship between the real and complex forms.

### 2.4 Splitting Equations

The Cooley-Tukey algorithm, which is the original fast Fourier transform, is based on splitting the sum in equation (1) into two sums as follows:

\[ X_k = \sum_{j=0}^{N/2-1} x_{2j}e^{-\frac{2\pi j k}{N}} + \sum_{j=0}^{N/2-1} x_{2j+1}e^{-\frac{2\pi j (k+1)}{N}}, \]  \hspace{1cm} (10)
\[
\begin{align*}
&= \sum_{j=0}^{N/2-1} x_{2j} e^{-i2\pi jk/N} + \sum_{j=0}^{N/2-1} x_{2j+1} e^{-i2\pi jk/N} e^{-i\pi k/N}.
\end{align*}
\]

Defining

\[
Y_k = \sum_{j=0}^{N/2-1} x_{2j} e^{-i2\pi jk/N}, \quad (11)
\]

\[
Z_k = \sum_{j=0}^{N/2-1} x_{2j+1} e^{-i2\pi jk/N}, \quad (12)
\]

the transform can then be calculated by the general Splitting Equations:

\[
X_k = Y_k + e^{-i\pi k/N} Z_k \quad k = 0, 1, 2, ..., N/2 - 1,
\]

\[
X_{N/2 + k} = Y_k - e^{-i\pi k/N} Z_k \quad k = 0, 1, 2, ..., N/2 - 1.
\]

The second equation is deduced from the periodicity of \( \{Y_k\} \) and \( \{Z_k\} \).

2.4.1 Splitting Equations for Real Data

If the sequence \( \{x_j\} \) is real, then the Fourier coefficients \( \{X_k\} \) for \( k = N/2+1, N/2+2, N/2+3, ..., N \) can be computed from the conjugate symmetry of the transform. It is therefore necessary to compute \( \{X_k\} \) for \( k = 0, 1, 2, ..., N/2 \) only. For the first \( N/4 \) of these coefficients we use

\[
X_k = Y_k + e^{-i\pi k/2} Z_k. \quad (15)
\]
For \( k = N/4 + 1, N/4 + 2, N/4 + 3, \ldots, N/2 \), substituting \( N/2 - k \) into equation (15) results in
\[
X_{\frac{N}{2} - k} = Y_{\frac{N}{2} - k} + e^{-\frac{i(N/2-k)2\pi}{N}} Z_{\frac{N}{2} - k}.
\]

By conjugate symmetry of \( \{Y_k\} \) and \( \{Z_k\} \) we have
\[
X_{\frac{N}{2} - k} = Y_k - e^{-\frac{i\pi k}{N}} Z_k \quad k = 0, 1, 2, \ldots N/4 - 1.
\]

Using (15) and (16) we are able to compute \( \{X_k\} \) for \( k = 0, 1, 2, \ldots, N/2 \).

3 The Algorithms

In [1] Swarztrauber presents a set of algorithms for real sequences with even (E), odd (O), quarter-wave even (QE) and quarter-wave odd (QO) symmetry. Each of these algorithms uses pre- and post-processing around a real, periodic FFT. The input vector is pre-processed to give a new vector (which is also real). The real FFT is then used to find the transform of this new vector. In the post-processing step, the transform of the original vector is calculated from the output of the real FFT.

In the same paper, a pre- and post-processing algorithm based on a complex routine is also described. This algorithm is for a general, real sequence
(R). The pre-processing step produces a complex vector from the real input. A complex transform is then used to find the transform of this vector. Finally, the transform of the original, real vector is obtained in the postprocessing step.

Implicit in these descriptions is another set of algorithms that combines these two approaches. Figure 1 gives a graphic representation of the algorithms and their combination. The call to the real FFT in the original algorithm for real, symmetric data, is replaced by the entire algorithm for general, real data.

These combined algorithms have not, to our knowledge, been implemented and it was thought that their efficiency might exceed that of the original symmetric algorithms. Even if they did not, because the complex FFT routine is often the first one to be optimized in a new computing environment, it was thought this set of routines might still prove useful.

In the rest of this section, these algorithms will be described in more detail. First, what we will call “the real algorithm” will be described. It is the one for general, real data which uses a complex FFT and which is mentioned above. Then, the “original” and “combined” algorithms will be given for the sine transform. For the rest of the symmetries, only the combined algorithm will be given.
Figure 1: The combined symmetric algorithms replace the real FFT of the original symmetric algorithms with the three main steps of the real algorithm.
3.1 The Real Algorithm

1. Given the real sequence \( \{x_j\} \) for \( j = 0, 1, 2, \ldots, N - 1 \) where \( N \) is even, compute two real sequences, \( \{y_j\} \) and \( \{z_j\} \), of length \( N/2 \) as follows:

\[
y_j = x_{2j} \quad j = 0, 1, 2, \ldots, N/2 - 1,
\]
\[
z_j = x_{2j+1} \quad j = 0, 1, 2, \ldots, N/2 - 1.
\]

Then compute the complex sequence \( \{w_j\} \) of length \( N/2 \)

\[
w_j = y_j + iz_j \quad j = 0, 1, 2, \ldots, N/2 - 1.
\]

2. Calculate \( \{W_k\} \), the Fourier transform of \( \{w_j\} \).

3. Recover \( \{Y_k\} \) and \( \{Z_k\} \), the transforms of \( \{y_j\} \) and \( \{z_j\} \), with the following formulas:

\[
Y_k = \frac{(W_k + \overline{W_{\frac{N}{2} - k}})}{2} \quad \text{for } k = 0, 1, 2, \ldots, \lfloor N/4 \rfloor, \quad (17)
\]
\[
Z_k = \frac{(W_k - \overline{W_{\frac{N}{2} - k}})}{(2i)} \quad \text{for } k = 0, 1, 2, \ldots, \lfloor N/4 \rfloor \quad (18)
\]

where \( \lfloor N/4 \rfloor \) is the integer closest to and less than \( N/4 \).

4. Use the splitting equations (15) and (16) for real sequences to get \( \{X_k\} \), the transform of \( \{x_j\} \):

\[
X_k = Y_k + e^{-i\frac{4\pi}{N}}Z_k \quad \text{for } k = 0, 1, 2, \ldots, N/4, \quad (19)
\]
\[ X_{N/2-k} = Y_k - e^{-\frac{2\pi i k}{N}} Z_k \quad \text{for } k = 0, 1, 2, ..., N/4 - 1. \quad (20) \]

If \( N \) is not divisible by 4, then both of these formulas must run from \( k = 0, 1, 2, ..., \lfloor N/4 \rfloor \).

The other half of the transform is calculated using the conjugate symmetry of \( \{X_k\} \):
\[ X_{N-k} = \overline{X_k} \quad \text{for } k = 1, 2, 3, ..., N/2 - 1. \]

We now derive the formulas for \( \{Y_k\} \) and \( \{Z_k\} \) used in step 3. Because the Fourier transform is linear
\[ W_k = Y_k + iZ_k. \quad (21) \]

Therefore
\[ W_{N-k} = Y_{N-k} + iZ_{N-k}. \]

Since \( \{y_j\} \) and \( \{z_j\} \) are real, their transforms are conjugate symmetric and
\[ W_{N-k} = \overline{Y_k + iZ_k}. \]

Taking conjugates of both sides, we have
\[ \overline{W_{N-k}} = Y_k - iZ_k. \quad (22) \]

Adding and subtracting (21) and (22) gives (17) and (18).
3.2 The Sine Algorithm

In this section the algorithm for an odd symmetric sequence is given. To be more accurate, it is an algorithm for a real sequence of length \( n \) which treats the data as if they had been extended to become an \( O \) sequence of length \( N = 2(n+1) \). Because of this imposed symmetry, the output sequence is purely imaginary.

Recall from the discussion of the properties of an odd sequence that \( x_0 \) and \( x_{N/2} = x_{n+1} \) are zero. So the input sequence is \( \{x_j\} \) for \( j = 1, 2, 3, \ldots n \). First, the algorithm sketched out in [1] for such a vector will be presented followed by the combined algorithm that has been implemented in this project.

3.2.1 The Original Algorithm

1. Given a sequence \( \{x_j\} \) for \( j = 1, 2, 3, \ldots n \) where \( n \) is odd, construct the sequence \( \{e_j\} \) as follows:

\[
e_j = (x_j - x_{n+1-j}) + (x_j + x_{n+1-j}) \sin\left(\frac{j\pi}{n+1}\right) \quad j = 1, 2, \ldots, n \quad (23)
\]

\[
e_0 = 0.
\]

Note that the definition for \( e_0 \) is consistent with the general definition of \( \{e_j\} \) and with the fact, discussed in the last section, that for an \( O \)
sequence, \( x_0 = x_N = 0 \). The vector \( \{e_j\} \) is thus defined for \( j = 0, 1, 2, \ldots n \) giving \( n + 1 \) values, which is an even number.

2. Use the real periodic transform to compute the Fourier transform of \( \{e_j\} \) in trigonometric form. From (7)

\[
e_j = \frac{c_0}{2} + \sum_{k=1}^{\frac{n+1}{2}-1} \left\{ c_k \cos\left(\frac{2\pi k}{n+1}\right) + d_k \sin\left(\frac{2\pi k}{n+1}\right)\right\} + (-1)^j \frac{a_{n+1}/2}{2}.
\]

(24)

The transform gives us the \( c_k \) and the \( d_k \).

3. As shown in section 2.2, the transform of \( \{x_j\} \) will be imaginary. Looking at (7) again, but this time applying it to \( \{x_j\} \) instead of \( \{e_j\} \) it can be seen that

\[
x_j = \sum_{k=1}^{n} b_k \sin\left(\frac{k j \pi}{n+1}\right).
\]

(25)

Substituting (25) into (23) and remembering that \( n \) is odd so that \( (n+1)/2 \) is an integer, we get

\[
e_j = b_1 + \sum_{k=1}^{\frac{n+1}{2}-1} \left[ (b_{2k+1} - b_{2k-1}) \cos\left(\frac{2\pi k}{n+1}\right) + b_{2k} \sin\left(\frac{2\pi k}{n+1}\right) \right] - (-1)^j b_n.
\]

(26)

Finally, comparing (24) and (26) we see that we can produce the sine coeffi-
\[ w_j = y_j + iz_j \quad j = 0, 1, 2, ..., (n+1)/2 - 1. \]

Note that \( \{w_j\} \) is complex and half as long as \( \{e_j\} \).

3. Call a complex FFT to compute \( \{W_k\} \), the Fourier transform of \( \{w_j\} \).
   \( \{W_k\} \) will be a complex sequence of length \((n+1)/2\). This will be in complex, as opposed to trigonometric, form.

4. Calculate \( \{Y_k\} \) and \( \{Z_k\} \), the transforms of \( \{y_j\} \) and \( \{z_j\} \), as in step 3 of the real algorithm.
   \[
   Y_k = \frac{1}{2}(W_k + W_{\frac{n+1-k}{2}}), \\
   Z_k = \frac{1}{2i}(W_k - W_{\frac{n+1-k}{2}}) \quad k = 0, 1, 2, ..., (n+1)/4 - 1.
   \]

Note that \( W_k \) is not defined for \( k = (n+1)/2 \), so in order to use this formula for \( k = 0 \) we must use the periodicity of the transform \( W_{(n+1)/2} = W_0 \).

5. Calculate \( \{E_k\} \) as in step 4 of the real algorithm.
   \[
   E_k = Y_k + e^{-ik\frac{2\pi}{n+1}}Z_k \quad k = 0, 1, 2, ..., (n+1)/4, \\
   \overline{E_{\frac{n+1-k}{2}}} = Y_k - e^{-ik\frac{2\pi}{n+1}}Z_k \quad k = 0, 1, 2, ..., (n+1)/4 - 1.
   \]

Thus \( \{E_k\} \) is defined for \( k = 0, 1, 2, ..., (n+1)/2. \)
6. Translate this into trigonometric form by using (5) and (6).

\[ c_k = 2\Re(E_k) \quad k = 0, 1, 2, \ldots, (n + 1)/2, \]

\[ d_k = -2\Im(E_k) \quad k = 1, 2, 3, \ldots, (n + 1)/2 - 1. \]

7. Finally find the \( \{b_k\} \) for \( k = 1, 2, 3, \ldots, n \) as it is done in the last step of the original algorithm.

### 3.3 The Cosine Algorithm

In this section we give the algorithm for even symmetric (E) data. The algorithm accepts a real, random vector and processes it as if it had been extended in such a way as to become an even sequence of length \( N = 2(n - 1) \); that is, so that \( x_j = x_N - j \), where \( N \) is the length of the extended vector and \( n \) is the length of the actual input vector. If, for example, \( n = 5 \) the extended vector would be as follows:

\[ x_0, x_1, x_2, x_3, x_4, x_3, x_2, x_1 \]

where \( \{x_0, x_1, x_2, x_3, x_4\} \) are the original data. Because of this imposed symmetry, the output sequence is real. Also as in the sine case, \( n \) must be odd.

The Original E algorithm as described by Swarztrauber[1] will not be
described. Only the combined algorithm, which is the one that was implemented during this project, will be discussed.

1. Given a sequence, \( \{x_j\} \) of length \( n \), where \( n \) is odd, produce the new sequence \( \{e_j\} \) as follows:

\[
e_j = (x_j + x_{n-j}) - (x_j - x_{n-j}) \sin\left(\frac{j\pi}{n-1}\right) \quad \text{for} \quad j = 0, 1, 2, \ldots, n-2. \tag{28}
\]

Note that \( \{e_j\} \) has length \( n - 1 \) and that all the input data are used in calculating \( \{e_j\} \).

2. Construct the real vectors \( \{y_j\} \) and \( \{z_j\} \), and the complex vector \( \{w_j\} \) all of length \( (n - 1)/2 \) using the following:

\[
y_j = e_{2j},
\]

\[
z_j = e_{2j+1},
\]

\[
w_j = y_j + iz_j \quad \text{for} \quad j = 0, 1, 2, \ldots, (n-1)/2 - 1.
\]

3. Call a complex FFT to compute the Fourier transform of \( \{w_j\} \). The result will be a complex sequence, \( \{W_k\} \), where \( 0 \leq k \leq (n - 1)/2 - 1 \) and will be in complex, as opposed to trigonometric form. By periodicity, \( W_{(n-1)/2} = W_0 \), a fact that will be used in the next step.
4. Recover the transforms of \( \{y_j\} \) and \( \{z_j\} \) as in step 3 of the real algorithm.

\[
Y_k = \frac{1}{2} (W_k + W_{n-k}), \\
Z_k = \frac{1}{2i} (W_k - W_{n-k}) \quad \text{for } k = 0, 1, 2, ..., (n-1)/4 - 1.
\]

5. Now calculate \( \{E_k\} \) which is the transform of \( \{e_j\} \) using (15) and (16), the splitting equations for a real sequence.

\[
E_k = Y_k + e^{\frac{j\pi k}{n-1}} Z_k \quad \text{for } k = 0, 1, 2, ..., (n-1)/4, \\
\overline{E_{n-k}} = Y_k - e^{\frac{j\pi k}{n-1}} Z_k \quad \text{for } k = 0, 1, 2, ..., (n-1)/4 - 1.
\]

Thus \( \{E_k\} \) is defined for \( 0 \leq k \leq (n-1)/2 \)

6. Translate this into trigonometric form by using (5) and (6).

\[
c_k = 2\Re(E_k) \quad \text{for } k = 0, 1, 2, ..., (n-1)/2, \\
d_k = -2\Im(E_k) \quad \text{for } k = 0, 1, 2, ..., (n-1)/2 - 1.
\]

7. As in the Sine algorithm, it is necessary to obtain the transform of the original vector from \( \{c_k\} \) and \( \{d_k\} \). Referring to (7) we see that

\[
e_j = \frac{c_0}{2} + \sum_{k=1}^{n-1} \left\{ c_k \cos\left(\frac{kJ2\pi}{n-1}\right) + d_k \sin\left(\frac{kJ2\pi}{n-1}\right) \right\} \\
+ (-1)^j \frac{c_{(n-1)/2}}{2}.
\]
As we know from section 2.2, for an even symmetric sequence \( \{x_j\} \), the transform is real which means (7) reduces to

\[
x_j = \frac{a_0}{2} + \sum_{k=1}^{n-2} a_k \cos\left(\frac{kj\pi}{n-1}\right) + (-1)^j a_{n-1}.
\]  

(30)

Substituting (30) into (28) yields

\[
e_j = a_0 + \sum_{k=1}^{n-2} \left[ a_{2k} \cos\left(\frac{kj2\pi}{n-1}\right) + (a_{2k+1} - a_{2k-1}) \sin\left(\frac{kj2\pi}{n-1}\right) \right] + (-1)^j a_{n-1}.
\]

(31)

Comparing (29) and (31) yields the following set of identities for the \( \{a_k\} \) which are the cosine coefficients of the original input.

\[
a_0 = \frac{c_0}{2},
\]

\[
a_{n-1} = \frac{c(n-1)/2}{2},
\]

\[
a_{2k} = c_k \text{ for } k = 1, 2, 3, \ldots, (n-1)/2 - 1,
\]

\[
a_{2k+1} - a_{2k-1} = d_k \text{ for } k = 1, 2, 3, \ldots, (n-1)/2 - 1.
\]

The coefficient \( a_1 \) is needed to start the recurrence relation in the last of these identities and it must be calculated explicitly. Using (8) and the fact that \( \{x_j\} \) is real and even, we have that

\[
a_1 = 2x_0 + 4 \sum_{j=1}^{n-2} x_j \cos\left(\frac{j\pi}{n-1}\right) + 2(-1)^k x_{n-1}.
\]
3.4 The Quarter-Wave Even Algorithm

We now proceed to the description of the combined algorithm for real quarter-wave even data. As for the algorithms already described, the input data need not have any symmetry other than being real. It is processed as if it had been extended to be a quarter-wave even symmetric vector. For these algorithms, \( n \), the length of the input sequence, must be even, and \( N = 2n \) where \( N \) is the length of the extended vector. The extension of the data is such that \( x_j = x_{N-j-1} \).

Although the transform of a quarter-wave even vector is complex, Swartztrauber [1] has shown that it can be represented by a vector which is real. That is, if \( \{X_k\} \) is the transform of the QE vector \( \{x_j\} \), then \( X_k = e^{ik\pi/N}\tilde{X}_k \) where \( \tilde{X}_k \) is real.

Note that as defined above, \( N \) is always even. With that restriction, the trigonometric form of the transform is:

\[
\begin{align*}
a_k &= 4 \sum_{j=0}^{N/2-1} x_j \cos \left[ \frac{k(2j + 1)\pi}{N} \right], \\
x_j &= \frac{a_0}{2} + \sum_{k=1}^{N/2-1} a_k \cos \left[ \frac{k(2j + 1)\pi}{N} \right]
\end{align*}
\]
where
\[ a_k = 2\tilde{X}_k = 2e^{-\frac{i\pi}{N}}X_k. \]

While for E and O data, the transform is its own inverse (neglecting scaling), this is not true for quarter-wave data. Both QE and QO data require distinct forward and backward algorithms. For reasons that will become apparent, it is more convenient to discuss the inverse transform first.

3.4.1 The Inverse Transform

1. Given \( \{a_k\} \) for \( k = 0, 1, 2, \ldots, n - 1 \), where \( \{a_k\} \) is the non-redundant portion of the transform of a QE vector in trigonometric form (recall from Section 2.2 that the transform of a real vector will be conjugate symmetric), compute \( \{e_k\} \) as follows:

\[ e_k = (a_k + a_{n-k})\cos\left(\frac{k\pi}{2n}\right) - (a_k - a_{n-k})\sin\left(\frac{k\pi}{2n}\right) \text{ for } k = 0, 1, \ldots, n-1. \] (33)

This definition of \( \{e_k\} \) requires \( a_n \) which is not given. However, from (32), and remembering that \( 2n = N \), it can be shown that \( a_n = 0 \).

2. Construct the real vectors \( \{y_k\} \) and \( \{z_k\} \), and the complex vector \( \{w_k\} \)
all of length \( n/2 \) using the following:

\[
y_k = e_{2k},
\]

\[
z_k = e_{2k+1},
\]

\[
w_k = y_k + iz_k \quad \text{for } k = 0, 1, 2, \ldots, n/2 - 1.
\]

3. Now calculate the Fourier transform \( \{W_j\} \) of the complex vector \( \{w_k\} \) by calling the forward complex routine.

4. Recover \( \{Y_j\} \) and \( \{Z_j\} \), the transforms of \( \{y_k\} \) and \( \{z_k\} \) as is done in step 3 of the real algorithm.

\[
Y_j = \frac{1}{2} (W_j + W_{2-j}) \quad \text{for } j = 0, 1, 2, \ldots, n/4
\]

\[
Z_j = \frac{1}{2} i (W_j - W_{2-j}) \quad \text{for } j = 0, 1, 2, \ldots, n/4.
\]

5. Obtain \( \{E_j\} \), which is the transform of \( \{e_k\} \), by using the splitting equations for a real sequence.

\[
E_j = Y_j + e^{-i\frac{2\pi j}{n}} Z_j \quad \text{for } j = 0, 1, 2, \ldots, n/4,
\]

\[
\overline{E_{2-j}} = Y_j - e^{-i\frac{2\pi j}{n}} Z_j \quad \text{for } j = 0, 1, 2, \ldots, n/4 - 1.
\]

6. Translate this into trigonometric form by using (5) and (6).

\[
c_j = 2\Re(E_j) \quad \text{for } j = 0, 1, 2, \ldots, n/2,
\]

\[
d_j = -2\Im(E_j) \quad \text{for } j = 1, 2, 3, \ldots, n/2 - 1.
\]
7. As in the algorithms already discussed, the results can be obtained from the \( \{c_j\} \) and \( \{d_j\} \) with the following justification: By (7) we know that

\[
e_k = \frac{c_0}{2} + \sum_{j=1}^{\frac{n}{2} - 1} \left[ c_j \cos\left(\frac{k j 2 \pi}{n}\right) + d_j \sin\left(\frac{k j 2 \pi}{n}\right) \right] + (-1)^k \frac{c_{n/2}}{2}. \tag{34}
\]

Then substituting (32) into (33) we see that

\[
e_k = 4x_0 + 4 \sum_{j=1}^{\frac{n}{2} - 1} \left[ (x_{2j} + x_{2j-1}) \cos\left(\frac{k j 2 \pi}{n}\right) + (x_{2j} - x_{2j-1}) \sin\left(\frac{k j 2 \pi}{n}\right) \right] + 4(-1)^k x_{n-1}. \tag{35}
\]

Comparing (35) and (34) we get the following set of formulas relating \( \{x_j\} \) to \( \{c_j\} \) and \( \{d_j\} \).

\[
c_0 = 8x_0, \tag{36}
\]

\[
c_j = 4(x_{2j} + x_{2j-1}) \quad j = 1, 2, 3, ..., n/2 - 1, \tag{37}
\]

\[
d_j = 4(x_{2j} - x_{2j-1}) \quad j = 1, 2, 3, ..., n/2 - 1, \tag{38}
\]

\[
c_y = 8x_{n-1}. \tag{39}
\]

Inverting these we get:

\[
x_0 = \frac{1}{8} c_0, \]

\[
x_{2j} = \frac{1}{8} (c_j + d_j), \]

\[
x_{2j-1} = \frac{1}{8} (c_j - d_j), \]

\[
x_{n-1} = \frac{1}{8} c_y. \]

26
By these last formulas we can obtain \( \{x_j\} \), the non-redundant portion of the QE vector. Note that this inverse transform for quarter-wave even data closely follows the pattern of the cosine and sine transforms. The forward transform is the mirror image of this algorithm.

### 3.4.2 The Forward Transform

1. Given a vector \( \{x_j\} \) for \( j = 0, 1, 2, \ldots, n - 1 \), compute the two vectors \( \{c_j\} \) and \( \{d_j\} \) using (36) through (39).

2. In the original quarter-wave even algorithm, one would do an inverse transform at this point. One would treat \( \{c_j\} \) and \( \{d_j\} \) as the trigonometric form of the transform of \( \{e_k\} \) (from the inverse algorithm), and use an inverse transform to obtain \( \{e_k\} \).

The combined algorithm will also obtain \( \{e_k\} \) but because it calls the inverse complex routine instead of the real periodic, more pre- and post-processing steps are necessary. The first of these is to obtain \( \{E_j\} \), the complex form of the transform of \( \{e_k\} \), by using the inverse of the definition of the trigonometric form of the transform for a real vector.

\[
\Re(E_j) = \frac{c_j}{2} \quad \text{for } j = 0, 1, 2, \ldots, n/2,
\]
\[ \Re(E_j) = \frac{-d_j}{2} \text{ for } j = 1, 2, 3, ..., n/2 - 1. \]

3. Now use the inverse of the splitting equations for a real vector (in this case \( \{e_k\} \)) to get \( \{Y_j\} \) and \( \{Z_j\} \). This corresponds to step 5 of the inverse algorithm.

\[
Y_j = \frac{1}{2} (E_j + \overline{E_{\frac{n}{2}-j}}) \text{ for } j = 0, 1, 2, ..., n/4, \\
Z_j = \frac{1}{2} e^{\frac{j \pi}{n}} (E_j - \overline{E_{\frac{n}{2}-j}}) \text{ for } j = 0, 1, 2, ..., n/4.
\]

4. Calculate \( \{W_j\} \), the transform of \( \{w_k\} \), by inverting formulas in step 3 of the inverse algorithm to get

\[
W_j = Y_j + iZ_j \text{ for } j = 0, 1, 2, ..., n/4, \\
\overline{W_{\frac{n}{2}-j}} = Y_j - iZ_j \text{ for } j = 0, 1, 2, ..., n/4 - 1.
\]

5. Call the inverse, complex transform with \( \{W_j\} \), which is complex and \( n/2 \) in length, to obtain \( \{w_k\} \).

6. Compute \( \{y_k\} \) and \( \{z_k\} \) from \( \{w_k\} \):

\[
y_k = \Re(w_k), \\
z_k = \Im(w_k) \text{ for } k = 0, 1, 2, ..., n/2 - 1.
\]
Then compute \( \{e_k\} \), the vector defined in step 1 of the inverse algorithm, as follows:

\[
e_{2k} = y_k, \tag{40}
\]
\[
e_{2k+1} = z_k \text{ for } k = 0, \frac{n}{2} - 1. \tag{41}
\]

7. Finally, we are able to compute \( \{a_k\} \) using the inverse of the formula in step 1.

\[
a_k = (e_k + e_{n-k}) \cos \left( \frac{k\pi}{2n} \right) - (e_k - e_{n-k}) \sin \left( \frac{k\pi}{2n} \right) \text{ for } k = 0, 1, 2, ..., n - 1.
\]

### 3.5 The Quarter-Wave Odd Algorithm

The algorithm for real quarter-wave odd data is very similar to the one for quarter-wave even data. Therefore only a short description will be given. Again, \( n \), the length of the input sequence, must be even, and \( N = 2n \) where \( N \) is the length of the extended vector. But this time the extension of the data is such that \( x_j = -x_{N-j-1} \).

The transform of a quarter-wave odd vector is complex, but Swarztrauber [1] has shown that if \( \{X_k\} \) is the transform of the QO vector \( \{x_j\} \), then \( X_k = e^{ik\pi/N} \tilde{X}_k \) where \( \tilde{X}_k \) is strictly imaginary.
Because \( N \) is always even, the trigonometric form of the transform is:

\[
\begin{align*}
  b_k &= 4 \sum_{j=0}^{N/2-1} x_j \sin\frac{k\pi}{N}(2j + 1), \\
  x_j &= \sum_{k=1}^{N-1} \left[ b_k \sin\frac{k\pi}{N}(2j + 1) \right] + (-1)^j b_{N/2}
\end{align*}
\]  

(42)

where

\[
  b_k = 2i\tilde{X}_k = 2ie^{-i\pi k} X_k.
\]

### 3.5.1 The Inverse Transform

1. Given \( \{b_k\} \) for \( k = 1, 2, 3, ..., n \), where \( \{b_k\} \) is the non-redundant portion of the transform of a QO vector in trigonometric form (since the data are real the transform will be conjugate symmetric), compute \( \{e_k\} \) as follows:

\[
e_k = (b_k + b_{n-k}) \cos\left(\frac{k\pi}{2n}\right) + (b_k - b_{n-k}) \sin\left(\frac{k\pi}{2n}\right) \quad \text{for} \quad k = 0, 1, ..., n-1. \quad (43)
\]

The formula for \( \{e_k\} \) requires \( \{b_k\} \) to be defined for \( 0 \leq k \leq n \), and we are not provided with \( b_0 \). But notice that using the above definition for \( \{b_k\} \), \( b_0 = 0 \).

2. Follow steps 2 through 6 of the QE algorithm to produce the \( \{c_j\} \) defined for \( 0 \leq j \leq n/2 \) and the \( \{d_j\} \) defined for \( 0 \leq j \leq n/2 - 1 \).
3. To develop the formulas for the relationship between these coefficients and the \( \{x_j\} \), substitute (42) into (43) to obtain

\[
c_k = 4x_0 + 4 \sum_{j=1}^{n-1} \left[ (x_{2j} - x_{2j-1}) \cos \left( \frac{kj2\pi}{n} \right) + (x_{2j} + x_{2j-1}) \sin \left( \frac{kj2\pi}{n} \right) \right] -4(-1)^k x_{n-1}.
\]

(44)

As in the QE case, by comparing (44) and (34) we find a set of relationships between \( \{x_j\} \), \( \{c_j\} \) and \( \{d_j\} \):

\[
c_0 = 8x_0, \quad (45)
\]

\[
c_j = 4(x_{2j} - x_{2j-1}) \quad j = 1, 2, 3, ..., n/2 - 1, \quad (46)
\]

\[
d_j = 4(x_{2j} + x_{2j-1}) \quad j = 1, 2, 3, ..., n/2 - 1, \quad (47)
\]

\[
c_{n/2} = -8x_{n-1}. \quad (48)
\]

Inverting these expressions gives

\[
x_0 = \frac{1}{8} c_0, \quad (49)
\]

\[
x_{2j} = \frac{1}{8} (c_j + d_j), \quad (50)
\]

\[
x_{2j-1} = -\frac{1}{8} (c_j - d_j), \quad (51)
\]

\[
x_{n-1} = -\frac{1}{8} c_{n/2}. \quad (52)
\]

from which we can obtain \( \{x_j\} \).
3.5.2 The Forward Transform

1. Given a vector \( \{x_j\} \) for \( j = 0, 1, 2, ..., n - 1 \) compute the two vectors \( \{c_j\} \) and \( \{d_j\} \) using (45), (46), (47), and (48).

2. Follow steps 2 through 6 of the QE forward transform.

3. Finally, we are able to compute \( \{b_k\} \) using the inverse of the formula in step 1:

\[
b_k = (c_k - c_{n-k}) \cos \left( \frac{k\pi}{2n} \right) + (c_k + c_{n-k}) \sin \left( \frac{k\pi}{2n} \right) \quad \text{for } k = 0, 1, 2, ..., n - 1.
\]

4 Implementation, Operation Counts and Timings

We now address the topics of the implementation of these algorithms in VCFFTP and the performance of these new codes. As stated earlier, the impetus behind this project was the desire to ascertain whether reducing the number of passes through the data saved a sufficient amount of execution time to compensate for the added operations required by these algorithms. Therefore, in each case, the performance of the new routine is measured against that of the analogous routine out of VFFTPAK. The codes for symmetric data are each compared to the routine that performs the “original” version of the
same algorithm (the VCFFT PK routine does the "combined" version). The
"real algorithm," which is implemented in the VCFFT PK routine VCFFT F,
is compared to the VFFTPAK routine VRFFT F which does the transform of
general, real data. Although VRFFT F does not use pre- and post-processing,
it's comparison with VCFFT F makes sense, not only because it produces the
same transform, but because VRFFT F provides the base for the pre- and post-
processing routines in VFFTPAK just as VCFFT F provides the basis for the
routines in VCFFT PK.

In our description of each of the algorithms, it was noted that there is a
restriction on the length of the input vector. This restriction is due to the use
of the complex routine. In each case, the input vector is processed to become a
new vector whose length is a function of the length of the input vector. Because
the new vector is passed to the complex routine, it must have an even number of
elements. Therefore, the input is required, depending on the case, to be either
even or odd: odd in the cosine and sine cases, even in the real and quarter-wave
cases.

The operations counts for the VCFFT PK routines are presented in Table 4
along with those of the analogous routines from VFFTPAK. The counts for
VFFTPAK are taken from [1] and those for VCFFT PK are deduced from the
Table 4: Operation Counts for Vectors of Length $N = 2^p$

<table>
<thead>
<tr>
<th>DATA</th>
<th>VFFTPAK adds</th>
<th>mults</th>
<th>VCFFTPK adds</th>
<th>mults</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>$(3p - 5)\frac{N}{2} + 2$</td>
<td>$(p - 3)N + 4$</td>
<td>$3p\frac{N}{2} - 2$</td>
<td>$(p - 2)N - 4$</td>
</tr>
<tr>
<td>E</td>
<td>$(3p + 2)\frac{N}{2} + 5$</td>
<td>$(p - 2)\frac{N}{2} + 1$</td>
<td>$(3p + 7)\frac{N}{2} + 1$</td>
<td>$(p + 3)\frac{N}{2} - 7$</td>
</tr>
<tr>
<td>O</td>
<td>$(3p - 3)\frac{N}{4} + 1$</td>
<td>$(p - 3)\frac{N}{4} + 2$</td>
<td>$(3p + 2)\frac{N}{4} - 3$</td>
<td>$(p - 2)\frac{N}{4} - 6$</td>
</tr>
<tr>
<td>QE</td>
<td>$(3p - 2)\frac{N}{4} + 4$</td>
<td>$(p - 3)\frac{N}{4} + 2$</td>
<td>$(3p + 3)\frac{N}{4}$</td>
<td>$(p - 2)\frac{N}{4} - 6$</td>
</tr>
<tr>
<td>QO</td>
<td>$(3p - 2)\frac{N}{4} + 4$</td>
<td>$(p - 3)\frac{N}{4} + 2$</td>
<td>$(3p + 3)\frac{N}{4}$</td>
<td>$(p - 2)\frac{N}{4} - 6$</td>
</tr>
</tbody>
</table>

same source. They are, therefore, theoretical counts rather than actual tallies.

Although the names of the vectorized packages are given, these counts apply to the scalar algorithms. For the vectorized version, given $m$ vectors of length $N$, each of the counts above would be multiplied by $m$. They are valid for $N \geq 16$.

The implementation of each of the algorithms is discussed below. The conclusions reached as a result of this project are given after this discussion.

The results presented in Tables 5 through 11 are of timings done on a single processor of the Cray Y-MP at NIST in Gaithersburg, Maryland. Preliminary testing was done on the Cray Y-MP at NCAR in Boulder, Colorado.

4.1 VRCFTF: The Real Algorithm

The algorithm described in section 3.1 has been implemented under the name VRCFTF. It was first coded in Fortran exactly as described above. Then the complex arithmetic was converted to real. Since, in Fortran, the complex
vector \{w_j\} and the real vector \{x_j\} are stored in exactly the same way, step 1 was thereby eliminated. Instead of creating \{w_j\}, the routine merely passes the input vector to the complex routine. Next, steps 3 and 4 were combined into one post-processing loop. This loop was arranged so that the calculations would be done in place, avoiding a final pass through the data to copy the results back into the input vector. When this process was completed, the resulting routine performed only one pass through the data.

Finally, the routine was vectorized by adding another dimension to all the arrays and changing each DO loop to a double DO loop. The complex routine used is VCFFFT which is a vectorized version of CFFT from FFTPAK. The original version was written by Swarztrauber. It uses a Gentleman-Sande algorithm [4].

Table 5 shows the results of the timings for VRCFTF and how they compare to those of VRFFT. The times for VCFFFT are also shown. Each line shows a different case in which \(N\) is the length of the sequences to be transformed and \(m\) is the number of sequences. The error is the maximum difference between the output of VRCFTF and that of VRFFT. The numbers under the names of the routines are the times that routine took to execute in milliseconds. The times for VRCFTF include the time spent in VCFFFT.
Table 5: The R Routines: VRCTF vs VRFFT

<table>
<thead>
<tr>
<th>m</th>
<th>N</th>
<th>error</th>
<th>VRCTF</th>
<th>VCPFFT</th>
<th>VRFFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>16</td>
<td>0.62E-14</td>
<td>0.09</td>
<td>0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>0.28E-13</td>
<td>0.16</td>
<td>0.10</td>
<td>0.14</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>0.11E-13</td>
<td>0.41</td>
<td>0.29</td>
<td>0.30</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>0.57E-13</td>
<td>0.86</td>
<td>0.64</td>
<td>0.73</td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>0.14E-13</td>
<td>1.81</td>
<td>1.36</td>
<td>1.58</td>
</tr>
<tr>
<td>64</td>
<td>512</td>
<td>0.57E-13</td>
<td>3.93</td>
<td>3.04</td>
<td>3.61</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>0.20E-13</td>
<td>9.29</td>
<td>7.52</td>
<td>7.72</td>
</tr>
<tr>
<td>64</td>
<td>2048</td>
<td>0.11E-12</td>
<td>19.82</td>
<td>16.29</td>
<td>17.78</td>
</tr>
</tbody>
</table>

4.2 VSINC: The Sine Algorithm

The algorithm described in section 3.2.2 is coded under the name VSINC. In programming this algorithm, we hoped to improve the performance of the original sine algorithm described in section 3.2.1 That algorithm is implemented in VSINT and is part of VFFTPAK. The operation counts for the two algorithms are shown in Table 4.

Originally written as described in section 3.2.2 with a separate step, and a separate data access for each of the 7 steps in the algorithm, the routine was eventually compressed to include one pre-processing step, which combined steps 1 and 2, the call to the complex routine, and one post-processing step which combined steps 4 through 7. The idea behind combining these loops is to
minimize the number of data accesses and maximize the amount of computation per access thereby optimizing the speed of the routine.

Unlike the real case, for the sine routine and all of the rest of these cases, the awkwardness of coding in place is avoided. Since all of these algorithms have a pre-processing step before the call to the complex subroutine and a post-processing step after it, the data flow is more natural. The first loop reads from the input vector and writes into a work vector. The work vector is then passed to the complex routine as input. Intermediate results are passed back in the same vector. It is then possible to read from the work vector and write back into the input vector during the last pass.

After the steps were combined it was possible, in both the pre-processing loop and the post-processing loop, to cut the number of data accesses in half again. From (23) it can be seen that \( e_{n-j} \) uses the same data as \( e_j \). We therefore structured the pre-processing loop so that these were done in one iteration, thus preventing that data from being brought from memory twice. Similarly, during post-processing, \( b_k \) and \( b_{n-k} \) need the same data. To take advantage of this, we ran the recurrence relation in (27) both ways. Thus it was possible and advantageous for the work to be doubled up at each iteration and the number of iterations to be cut in half.
Table 6: The O Routines: VSINC vs VSINT

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>error</th>
<th>VSINC</th>
<th>VCFFTF</th>
<th>VSINT</th>
<th>VRFFTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>15</td>
<td>0.29E-13</td>
<td>0.13</td>
<td>0.06</td>
<td>0.14</td>
<td>0.08</td>
</tr>
<tr>
<td>64</td>
<td>31</td>
<td>0.50E-13</td>
<td>0.21</td>
<td>0.10</td>
<td>0.22</td>
<td>0.14</td>
</tr>
<tr>
<td>64</td>
<td>63</td>
<td>0.69E-13</td>
<td>0.60</td>
<td>0.35</td>
<td>0.54</td>
<td>0.36</td>
</tr>
<tr>
<td>64</td>
<td>127</td>
<td>0.12E-12</td>
<td>1.13</td>
<td>0.67</td>
<td>1.14</td>
<td>0.77</td>
</tr>
<tr>
<td>64</td>
<td>255</td>
<td>0.22E-12</td>
<td>2.20</td>
<td>1.35</td>
<td>2.16</td>
<td>1.52</td>
</tr>
<tr>
<td>64</td>
<td>511</td>
<td>0.28E-12</td>
<td>4.67</td>
<td>2.98</td>
<td>4.72</td>
<td>3.44</td>
</tr>
<tr>
<td>64</td>
<td>1023</td>
<td>0.51E-12</td>
<td>10.83</td>
<td>7.45</td>
<td>9.95</td>
<td>7.42</td>
</tr>
<tr>
<td>64</td>
<td>2047</td>
<td>0.83E-12</td>
<td>22.83</td>
<td>16.08</td>
<td>22.25</td>
<td>17.18</td>
</tr>
</tbody>
</table>

Table 6 shows the results of the timings of VSINC and VSINT. The times for VCFFTF and VRFFTF, the complex and real routines called by these routines respectively, are also given. Note that in this case n is the length of the input vector. The operation counts given in Table 4 are in terms of \( N = 2(n+1) \). The other definitions are the same here as they were for the real case.

4.3 VCOSC: The Cosine Algorithm

The implementation of the algorithm described in section 3.3 is the VCFFTPK routine VCOSC. Essentially, the intention and theory behind this process was identical to that of VSINC. It, too, was compressed into a single pre-processing step, the call to the complex routine, and a single post-processing step in order to minimize passes through the data. Here, too, the loops were doubled up to
reduce the number of data accesses.

However, this case presented a few more complications than the sine case did. In the original sine algorithm, at the last step, the starting values for the recurrence relation are available. At the same place in the original cosine algorithm, these values must be computed explicitly. In order to run the recurrence relation both directions, as we had done in the sine routine, $a_{n-2}$ had to be calculated as well as $a_1$. Since both of these calculations are summations, it was possible to do them, one addition per iteration, in the loops that accomplish the pre- and post-processing. Therefore, although they add to the number of calculations being done, they do not add significantly to the data motion. But, as coded, this routine increases the number of adds in Table 4 by $N/4$ and the number of multiplications by the same amount.

Table 7 shows the results of the timings of VCOSC and VCOST. The definitions for the labels in this table are the same as those given for the sine case except that $N = 2(n - 1)$.

4.4 The Quarter-Wave Algorithms

As noted above, the transform of quarter-wave symmetric data is not its own inverse. Forward and inverse routines are required. It is somewhat
Table 7: The E Routines: VCOSC vs VCOST

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>error</th>
<th>VCOSC</th>
<th>VCFFFT</th>
<th>VCOST</th>
<th>VRFFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>17</td>
<td>0.28E-13</td>
<td>0.12</td>
<td>0.05</td>
<td>0.12</td>
<td>0.07</td>
</tr>
<tr>
<td>64</td>
<td>33</td>
<td>0.46E-13</td>
<td>0.23</td>
<td>0.10</td>
<td>0.25</td>
<td>0.14</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>0.57E-13</td>
<td>0.54</td>
<td>0.29</td>
<td>0.49</td>
<td>0.29</td>
</tr>
<tr>
<td>64</td>
<td>129</td>
<td>0.91E-13</td>
<td>1.13</td>
<td>0.63</td>
<td>1.10</td>
<td>0.71</td>
</tr>
<tr>
<td>64</td>
<td>257</td>
<td>0.13E-12</td>
<td>2.35</td>
<td>1.35</td>
<td>2.30</td>
<td>1.53</td>
</tr>
<tr>
<td>64</td>
<td>513</td>
<td>0.22E-12</td>
<td>4.97</td>
<td>2.97</td>
<td>5.05</td>
<td>3.45</td>
</tr>
<tr>
<td>64</td>
<td>1025</td>
<td>0.35E-12</td>
<td>11.43</td>
<td>7.45</td>
<td>10.47</td>
<td>7.44</td>
</tr>
<tr>
<td>64</td>
<td>2049</td>
<td>0.48E-12</td>
<td>23.96</td>
<td>16.03</td>
<td>23.18</td>
<td>17.15</td>
</tr>
</tbody>
</table>

arbitrary which algorithm is called the forward and which is called the backward transform.

The QO and QE routines in VCFFTP K correspond to the definition given above in sections 3.4 and 3.5 and to the algorithms described in [1]. The forward routines are named VSNQCF and VCSQCF. The inverse routines are VSNQCB and VCSQCB. However, the corresponding routines in VFFTPAK have their names switched. VSINQF and VCOSQF do what we have been calling the inverse transform. VSINQB and VCOSQB do the forward transform. Rather than redefine the transforms, we have chosen to leave the VCFFTP K routines named as they are and note clearly in the documentation that VSNQCF produces the same results as VSINQB, while VSNQCB produces the same results as VSINQF. Similarly, for the QE routines, VCSQCF corresponds to VCOSQB...
and VCSQCB to VCOSQF. This is why, in the timing results that follow, VFFTPAK forward routines are compared to VCFFFT PK inverse ones and vice-versa. Notice, too, that the forward VCFFFT PK routines call VCFFTB and the inverse ones call VCFFTF, respectively the inverse and forward complex routines.

We went about coding these two sets of routines in different ways. For the QE routines, we took the corresponding vectorized routines from VFFTPAK and edited them to call VCFFTF and VCFFTB (the vectorized version of the inverse complex routine from FFTP AK) with additional post-processing. However, the QO routines in VFFTPAK use a different algorithm in which they reverse the order of the input vector, call the QE routines, and then change every other sign in the output vector. In other words, another layer of pre- and post-processing and a subroutine call have been added. For this reason, and in order to confirm our understanding of the algorithms, we wrote the QO routines in VCFFFT PK from scratch. The procedure was very similar to that used for the sine and cosine routines. The development of these codes went much more smoothly than that of the QE routines. However, the end results of the two processes are very similar.

For the QO codes, we wrote scalar versions of the forward and backward real algorithm discussed in 3.1 and called these from scalar QO routines that
Table 8: The Inverse QE Routines: VCSQCB vs VCOSQF

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>error</th>
<th>VCSQCB</th>
<th>VCFFTF</th>
<th>VCOSQF</th>
<th>VRFFTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>16</td>
<td>0.16E-13</td>
<td>0.11</td>
<td>0.05</td>
<td>0.13</td>
<td>0.07</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>0.25E-13</td>
<td>0.22</td>
<td>0.10</td>
<td>0.26</td>
<td>0.15</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>0.28E-13</td>
<td>0.52</td>
<td>0.29</td>
<td>0.52</td>
<td>0.30</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>0.50E-13</td>
<td>1.08</td>
<td>0.63</td>
<td>1.14</td>
<td>0.73</td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>0.85E-13</td>
<td>2.28</td>
<td>1.37</td>
<td>2.45</td>
<td>1.60</td>
</tr>
<tr>
<td>64</td>
<td>512</td>
<td>0.99E-13</td>
<td>4.81</td>
<td>3.01</td>
<td>5.28</td>
<td>3.58</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>0.13E-12</td>
<td>11.27</td>
<td>7.64</td>
<td>11.32</td>
<td>7.95</td>
</tr>
<tr>
<td>64</td>
<td>2048</td>
<td>0.23E-12</td>
<td>23.35</td>
<td>16.18</td>
<td>24.30</td>
<td>17.72</td>
</tr>
</tbody>
</table>

did just the first and the last step of the algorithm outlined in 3.5. Once that was producing the correct result, we pulled the subroutines up into the main routine. Then the compression and vectorization proceeded just as it did for the sine and cosine routines.

Notice that the operations counts in Table 4 are identical for the QE and QO routines. The results of the timings are given in Tables 8, 9, 10, and 11. For all of them, \( N = 2n \) and the other definitions remain as they are for the other timings.
Table 9: The Forward QE Routines: VCSQCF vs VCOSQB

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>error</th>
<th>VCSQCF</th>
<th>VCFFTB</th>
<th>VCOSQB</th>
<th>VRFFTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>16</td>
<td>0.32E-13</td>
<td>0.11</td>
<td>0.05</td>
<td>0.13</td>
<td>0.07</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>0.28E-13</td>
<td>0.22</td>
<td>0.10</td>
<td>0.27</td>
<td>0.16</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>0.39E-13</td>
<td>0.52</td>
<td>0.29</td>
<td>0.54</td>
<td>0.32</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>0.43E-13</td>
<td>1.09</td>
<td>0.63</td>
<td>1.18</td>
<td>0.76</td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>0.50E-13</td>
<td>2.30</td>
<td>1.37</td>
<td>2.54</td>
<td>1.69</td>
</tr>
<tr>
<td>64</td>
<td>512</td>
<td>0.46E-13</td>
<td>4.85</td>
<td>3.01</td>
<td>5.38</td>
<td>3.73</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>0.57E-13</td>
<td>11.17</td>
<td>7.51</td>
<td>11.51</td>
<td>8.16</td>
</tr>
<tr>
<td>64</td>
<td>2048</td>
<td>0.53E-13</td>
<td>23.53</td>
<td>16.21</td>
<td>24.96</td>
<td>18.36</td>
</tr>
</tbody>
</table>

Table 10: The Inverse QO Routines: VSNQCB vs VSINQF

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>error</th>
<th>VSNQCB</th>
<th>VCFFTF</th>
<th>VSINQF</th>
<th>VRFFTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>16</td>
<td>0.80E-14</td>
<td>0.11</td>
<td>0.05</td>
<td>0.14</td>
<td>0.07</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>0.28E-13</td>
<td>0.21</td>
<td>0.10</td>
<td>0.29</td>
<td>0.15</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>0.15E-13</td>
<td>0.51</td>
<td>0.29</td>
<td>0.58</td>
<td>0.30</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>0.43E-13</td>
<td>1.07</td>
<td>0.64</td>
<td>1.27</td>
<td>0.73</td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>0.36E-13</td>
<td>2.22</td>
<td>1.36</td>
<td>2.65</td>
<td>1.58</td>
</tr>
<tr>
<td>64</td>
<td>512</td>
<td>0.85E-13</td>
<td>4.74</td>
<td>3.03</td>
<td>5.71</td>
<td>3.57</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>0.85E-13</td>
<td>10.93</td>
<td>7.52</td>
<td>11.96</td>
<td>7.72</td>
</tr>
<tr>
<td>64</td>
<td>2048</td>
<td>0.17E-12</td>
<td>23.17</td>
<td>16.34</td>
<td>26.26</td>
<td>17.77</td>
</tr>
</tbody>
</table>
Table 11: The Forward QO Routines: VSNQCF vs VSINQB

<table>
<thead>
<tr>
<th>n</th>
<th>error</th>
<th>VSNQCF</th>
<th>VCFFTB</th>
<th>VSINQB</th>
<th>VRFFTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>16</td>
<td>0.20E-13</td>
<td>0.12</td>
<td>0.05</td>
<td>0.15</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>0.32E-13</td>
<td>0.25</td>
<td>0.10</td>
<td>0.30</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>0.21E-13</td>
<td>0.58</td>
<td>0.29</td>
<td>0.59</td>
</tr>
<tr>
<td>64</td>
<td>128</td>
<td>0.36E-13</td>
<td>1.22</td>
<td>0.63</td>
<td>1.29</td>
</tr>
<tr>
<td>64</td>
<td>256</td>
<td>0.31E-13</td>
<td>2.52</td>
<td>1.36</td>
<td>2.71</td>
</tr>
<tr>
<td>64</td>
<td>512</td>
<td>0.43E-13</td>
<td>5.35</td>
<td>3.02</td>
<td>5.84</td>
</tr>
<tr>
<td>64</td>
<td>1024</td>
<td>0.36E-13</td>
<td>12.14</td>
<td>7.50</td>
<td>12.23</td>
</tr>
<tr>
<td>64</td>
<td>2048</td>
<td>0.50E-13</td>
<td>25.53</td>
<td>16.25</td>
<td>26.73</td>
</tr>
</tbody>
</table>

5 Conclusion

Looking at the timings for the O routines we see that, for half of the cases shown, the VCFFTPK routines run slightly faster than the VFFTPAK routines. In general, the cases where this is true are the ones in which an extra pass through the data is avoided, that is, every other case.

To see this, it must be noted that the number of passes performed by either VRFFTF or VCFFTF depend on the number of factors $N$ has, where $N$ is the length of the vector passed to these subroutines (as distinguished from the length of the original input vector). The timings were all done on vectors for which $N$ is a power of two. Recall, also, that if VRFFTF is passed a vector of length $N$, VCFFTF is passed one of length $N/2$. Let $N = 2^p$ where $p$ is
even. Then $N$ has $p/2$ factors of 4. $N/2$ has $p/2 - 1$ factors of 4 plus a factor of 2. So both algorithms require the same number of passes. However, if $p$ is odd, $N$ has $(p - 1)/2$ factors of 4 plus an additional factor of 2 while $N/2$ has only the $(p - 1)/2$ factors of 4, giving the algorithm based on the complex routine the advantage of one fewer pass. Thus, only every other power of two provides VCFFT PK with an advantage over VFFT PAK. This alternation can be detected in many of the timing tables.

Even when these routines require the extra pass through the data, they perform better than would be predicted by the number of operations added to the algorithm by the extra pre- and post-processing. This phenomenon is also seen in Tables 8 and 9 where the QE routines in VCFFT PK are shown to be faster than those in VFFT PAK for all of the cases run. We theorize that this is due to the fact that the VFFT PAK routines call VRFFT F which scales the results using an extra loop, thereby adding a pass through the data. The VCFFT PK routines call VCFFT F which does no scaling. Instead the scaling is combined with the post-processing pass. With this approach, even though the scaling costs about $n = N/2$ more multiplies it doesn’t cost the extra memory accesses the other routine incurs. In other words, we suspect that any speedup the VCFFT PK routines display is due to coding details rather than to the
algorithms themselves. We hope to establish this more firmly by doing further research.

The cosine routine does not perform as well when compared to its analog in VFFTPAK. This can be explained by the fact that in order to run the recursion both ways in the post-processing loop, we calculate \( a_{n-2} \) explicitly thereby adding \( n/2 \) operations. If this was not done, and the last loop was only run one direction, more data accesses would be required. Either way, the complications make this algorithm slower than VCOST.

The excellent performance of the QO routines, VSNQCF and VSNQCB, when compared to the VFFTPAK routines VSINQB and VSINQF, can be explained by the fact that the QO routines from VFFTPAK call the QE ones from that package. This design seems to sacrifice efficiency in favor of reducing the number of distinct routines. It is not surprising that the explicitly coded QO routines in VCFFTPK outperform these.

The R routine does poorly when compared to VRFFT. But notice that even if a pass through the data is saved inside of VCFFTF, it is added on again by the post-processing loop. So this routine has no chance to make up for the extra operations it has to do.
In conclusion, the combined algorithms used in VCFFTPK perform about as well, but no better, than the ones used in VFFTPAK. An extra pass through the data is, indeed, costly; but the time saved by avoiding it is cancelled out by the extra operations required to do so.

In addition to the algorithm described in section 3.1, and used by this package, Swarztrauber [1] describes another algorithm for real data that uses a complex routine. That algorithm combines two real vectors rather than splitting one and, therefore, has no restriction on the length of the vector. It also requires less post-processing. It was thought to be less promising for implementation on a vector computer because it requires movement in both the rows and the columns of the vector. Nevertheless, a set of routines based on that algorithm, instead of the one we have used, would be an interesting candidate for further study.
References


