TEK SPS BASIC
VØ2/VØ2XM
Signal Processing Package

CP57001/CP57501

Tektronix®

COMMITTED TO EXCELLENCE

Tektronix®
COMMITTED TO EXCELLENCE

TEK SPS BASIC
VØ2/VØ2XM
Signal Processing Package
CP57001/CP57501

INSTRUCTION MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Serial Number _____

First Printing MAY 1980 PRODUCED BY SPS DOCUMENTATION GROUP 070-2743-00

SOFTWARE SUPPORT POLICY

Unless otherwise provided, Tektronix, Inc., agrees that during the one (1) year period following installation, if the customer encounters a problem with this software which the customer's diagnosis indicates is caused by a software defect, the customer may submit a Software Performance Report to Tektronix, Inc. For problems occurring in current, unaltered releases of software, Tektronix, Inc., will respond to Software Performance Reports via a software maintenance periodical. The software maintenance periodical will be provided at no cost to the customer for one year following installation and will contain information for correcting or bypassing verified problems where possible, and will give notice of availability of corrected software.

Any software updates released by Tektronix, Inc., to correct problems during the one (1) year period will be provided to the customer at no charge on the standard distribution media specified in the software documentation. If media other than the standard distribution media is requested, the customer will only be charged for the current cost of the optional media.

SOFTWARE LICENSE

This software product, including subsequent improvements or updates, is furnished under a license for use on a single controller. It may only be copied, in whole or in part (with the proper inclusion of the Tektronix, Inc., copyright notice on the software), for use on that specific controller.

Specification and price change privileges are reserved.

Although the material in this manual has been thoroughly edited and checked for accuracy, Tektronix, Inc., makes no guarantees against typographical or human errors. Also, Tektronix, Inc., assumes no responsibility or liability, consequential or otherwise, of any kind arising from misinterpretation or misuse of the material in this manual. The contents of this manual are subject to change without notice.

PREFACE

This manual describes the Signal Processing Commands Package for TEK SPS BASIC VØ2 and VØ2XM software. Any exception to an option or a capability of a command in this package being supported by a specific release of the software is noted where appropriate. Information that pertains only to extended memory (XM) systems is shaded.

The prerequisite software for executing the commands in this package is the corresponding version of the TEK SPS BASIC System Software. The VØ2 package (CP57001) requires the VØ2 System Software (CP57000); the VØ2XM package (CP57501) requires the VØ2XM System Software (CP57500).

TABLE OF CONTENTS

SECTION 1 -- SIGNAL PROCESSING COMMANDS **1-1**

Introduction 1-1

Command Descriptions 1-3

Guide to Syntax Notation 1-3
Memory Requirements 1-3
Array Sizes 1-3
CONVL (Nonresident) 1-4
CORR (Nonresident) 1-10
DIFF (Nonresident) 1-17
INT (Nonresident) 1-26
POLAR (Nonresident) 1-33
RFFT (Nonresident) 1-37
RFFT1 (Nonresident) 1-51

SECTION 2 -- GLOSSARY **2-1**

SECTION 1

SIGNAL PROCESSING COMMANDS

Introduction

This manual describes the Signal Processing Package commands for TEK SPS BASIC VØ2 and VØ2XM. The package consists of seven nonresident commands for processing waveforms or arrays. These commands are summarized in Table 1-1. The operations performed include convolution, correlation, differentiation, integration, fast Fourier transform, inverse Fourier transform, and rectangular-to-polar conversion.

All the Signal Processing Package commands are designed to operate on floating-point waveforms and arrays. When a waveform is the destination for the output of a command, the data sampling interval (DSI), the horizontal units, and the vertical units are all updated to reflect the results of the operation. This automatic waveform arithmetic inludes the processing of the units strings (by an operation like the CAN string function) which cancels matching characters from strings that have a slash (/).

The Signal Processing commands are used in the same manner as other nonresident TEK SPS BASIC commands. The command may be preceded by a line number and included in a program, or the command may be executed directly by entering it from the terminal in immediate mode. If the command is not in memory when it is called, it is auto-loaded if it is stored on the system peripheral device. If it is not in memory and not on the system device, the command must be brought into memory with the LOAD command before it can be executed. Auto-loaded commands are auto-released from memory when more free memory is needed; explicitly LOADed commands are removed from memory with the RELEASE command. Also, all auto-loaded commands can be removed from memory with a RELEASE AUTO statement.

TABLE 1-1

Summary of Signal Processing Commands

CONVL	Performs a non-cyclic, discrete convolution operation on two source arrays or waveforms (which are overwritten by intermediate results) and places the result in a third array or waveform.
CORR	Performs a non-cyclic, discrete auto- or cross-correlation operation on two source arrays or waveforms (which are overwritten by intermediate results) and places the result in a third array or waveform.
DIFF	Performs a differentiation of an array or waveform and places the result in a second array or waveform. The source and destination arguments may be the same, in which case the source is overwritten by the result.
INT	Performs an integration of an array or waveform and places the result in a second array or waveform. The source and destination arguments may be the same, in which case, the source is overwritten by the result.
RFFT	Performs a fast Fourier transform on a real-valued array or waveform via a power-of-two algorithm and places the frequency results in two arrays or waveforms: one for real and one for imaginary components. It also performs the inverse Fourier transform given the frequency component arrays or waveforms.
POLAR	Performs a rectangular-to-polar conversion of the real and imaginary component arrays or waveforms as returned by the RFFT command. The source arguments are overwritten by the magnitude and phase results.
RFFT1	Performs a fast Fourier transform on a real-valued array or waveform via a power-of-two algorithm and overwrites the source with the result. It can also perform the inverse Fourier transform on frequency domain data stored in the same format as the result from its direct transform.

Command Descriptions

The remainder of this section contains the command descriptions for the Signal Processing Package commands. The command descriptions are listed in alphabetical order, and include examples, syntax forms, and a discussion of how the syntax options are used.

Guide to Syntax Notation

The syntax forms describe how the commands may be typed on the terminal. Upper case characters and punctuation must be typed as shown, but any information in brackets ([]) is optional. Braces ({}) indicate that a choice must be made between one of the listed items. Items followed by an ellipsis (...) may be repeated one or more times.

Memory Requirements

The approximate size of each command is listed in Appendix H of the System Software manual. This size refers to the number of words of memory required to load that particular command. In some cases, the amount of memory needed to execute the command will be considerably more.

Array Sizes

The Signal Processing commands operate only on floating-point arrays and waveforms. In standard memory systems, the size of an array is limited by the amount of free memory. In extended memory (XM) systems, the size of a floating-point array is limited to 8192 (8K) elements.

CONVL (Nonresident)

Examples:

```
   CONVL X1,X,Z,TB
10 CONVL A(Ø:127),B(Ø:127),C(Ø:255)
20 CONVL X,Y,Z,T(Ø:63)
```

Syntax Form:

[line no.] **CONVL** { floating-point array / floating-point waveform } , { floating-point array / floating-point waveform } ,
{ floating-point array / floating-point waveform } [, { simple numeric variable / floating-point array }]

Descriptive Form:

[line no.] **CONVL** source data, source data, target for convolved result [,sine table]

Purpose:

The CONVL command performs a non-cyclic, discrete convolution operation on two source arrays and places the result in a third array.

Discussion:

The CONVL command performs a fast convolution operation on two input arrays, placing the result in a third array. The convolution operation can be thought of as successively shifting, multiplying, and integrating the two arrays (waveforms) to be convolved, except that one of the waveforms is reversed in time before performing the shifting-multiplying-integrating process. This is evident from examining the summation that mathematically describes convolution discussed later in **The Theory of Convolution.** The actual operation in TEK SPS BASIC, however, is done in the frequency domain. This requires only a multiplication of the fast fourier transforms of the two arrays. (See **The CONVL Algorithm** for details.)

An important application for convolution is determining or predicting the output of a linear, time-invariant system (such as a passive filter or network). Given the input signal x(t) and the system impulse response h(t), the output can be predicted simply by convolving x(t) with h(t).

The format of the CONVL command is illustrated by the following example:

```
CONVL X,Y,Z,TB
```

The first two arguments (X and Y in the example) are the input arrays to be convolved. They must be floating-point waveforms, arrays, or contiguous subarrays, of length $N = 2^m$ for some integer $m \geq 4$. The source arguments must not overlap. **The CONVL command overwrites both source arrays with intermediate results during its execution.** Because of this overwriting, you may want to save the original array contents elsewhere before executing CONVL.

The third argument (Z in the example) specifies the target array for the convolution result. It must be a floating-point waveform, array, or contiguous subarray of length 2N, where N is the length of each source array. **(Since a floating-point array in an extended memory (XM) system is limited to 8K elements, the largest N can be in an XM program is 4K = 2^{12} = 4Ø96.)** After the convolution is performed, this array contains the result in its first 2N-1 locations, elements Z(Ø) through Z(2N-2). The last element, Z(2N-1), is used to store intermediate results. Thus, after CONVL executes, the content of the last element is meaningless and should be ignored.

The fourth argument (TB in the example) is optional. If it is specified, the Fourier transform (used by CONVL) is table driven. If the argument is a simple numeric variable, that variable is autodimensioned to an array of length N/2 and filled by CONVL with sine terms needed by the Fourier transform, creating the table. Subsequent executions of CONVL do not require regeneration of the table if the source and target array lengths remain unchanged and the table is not altered. (The table can be generated by the CORR, RFFT, and RFFT1 commands also. For more information about the table, see the RFFT command.) If the argument is an array, it must be of length N/2. CONVL checks the first element of the array. If it is the correct value, the array is assumed to be the proper sine-values table. If the first element is an incorrect value, CONVL fills the array with the correct

sine values. If the argument is not present, the necessary sine values are recursively generated as needed. This method saves memory at the cost of longer execution times.

The Theory of Convolution

The convolution of two time domain functions, say x(t) and y(t), can be expressed by the following integral:

$$z(t) = \int_{-\infty}^{\infty} x(u)y(t-u)\ du$$

The function z(t) is the result of convolving waveforms x and y according to the integral expression. The u is simply the arbitrary variable of integration for the definite integral.

In signal processing applications, this operation is performed with a discrete approximation having a finite time window. For a time window of N samples, where each sample is Δt seconds apart, the general expression for the discrete approximation may be written as follows:

$$Z(n\Delta t) = \Delta t \sum_{K=\emptyset}^{N-1} X(k\Delta t)Y(n\Delta t - k\Delta t) \quad \text{for } n = \emptyset, 1, 2, \ldots, 2N-2$$

where X and Y terms with indices that are less than zero or greater than N-1 are taken to be zero.

If Δt is normalized to be 1, then the discrete approximation reduces to:

$$Z(n) = \sum_{k=\emptyset}^{N-1} X(k)Y(n-k) \quad \text{for } n = \emptyset, 1, 2, \ldots, 2N-2$$

In the above summation, N is the length of the input arrays. Also, X(k) is the kth element of the first input array and Y(n-k) is the (n-k)th element of the second input array; k is merely an index that defines the range of the summation. The summation is computed by summing X(k)Y(n-k)

as k ranges from Ø to N-1; this is done for each value of n, beginning with Ø and ending with 2N-2. Each value of n thus defines a corresponding element of the final output array Z. For some values of n and k, the index, n-k, of array Y may be equal to a number less than Ø. When this occurs, Ø is used for Y(n-k).

The CONVL Algorithm

The preceding discussion illustrates one method for computing convolutions. It is instructive because it describes the way in which convolution data is formatted. However, rather than evaluating the discrete convolution summation directly, the CONVL command uses a faster computational method that takes advantage of the fast Fourier transform (FFT) and the inverse Fourier transform (IFT) algorithms. This method is based on the fact that the convolution of two signals (time domain) is equivalent to the multiplication of their Fourier transforms (frequency domain). Stated in more mathematical terms:

$$F(Z) = F(X)F(Y)$$

where F(X) and F(Y) are the Fourier transforms of the two time-domain signals, X and Y; and F(Z) is the Fourier transform of Z, the convolution of X and Y.

Before the CONVL command actually performs the convolution on two arrays, it appends N zeroes to each of the input arrays, X and Y. This prevents cyclic convolution which can arise from the assumption of periodicity that the FFT makes. The zero-appended input arrays, X and Y, are then transformed to the frequency domain via the FFT and a complex multiplication is performed on the resulting arrays. Finally, an inverse Fourier transform (IFT) is performed on the product, resulting in a convolution of the original arrays. This procedure is equivalent to the direct evaluation of the non-cyclic convolution summation previously discussed, but the whole process is performed faster due to the smaller number of calculations required.

Units and Data Sampling Interval (DSI) Definitions

The following list describes how the vertical and horizontal units and data sampling interval are automatically assigned when a waveform is the target for the CONVL command. To simplify this discussion, it is assumed that A and B are arrays and that WA, WB, and WC are waveforms. Accordingly, the following conventions are used:

SA: the data sampling interval for waveform WA
SB: the data sampling interval for waveform WB
SC: the data sampling interval for waveform WC

HA$: the horizontal units for waveform WA
HB$: the horizontal units for waveform WB
HC$: the horizontal units for waveform WC

VA$: the vertical units for waveform WA
VB$: the vertical units for waveform WB
VC$: the vertical units for waveform WC

Using the above notation, these rules apply when the target argument (WC) is a waveform. An ampersand (&) indicates that the units strings are concatenated (joined together).

CASE 1: Both source arguments are waveforms. For example:

```
CONVL WA,WB,WC
```

Then:

```
SC = SA
HC$ = HA$
VC$ = VA$ & VB$ & HA$
```

NOTE: A warning error is issued if the horizontal units for waveform WA do not equal the horizontal units for waveform WB, or if the data sampling interval for waveform WA is not equal to the data sampling interval for waveform WB.

CASE 2: The first source argument is a waveform; the second source argument is an array. For example:

```
CONVL WA,B,WC
```

Then:

```
SC = SA
HC$ = HA$
VC$ = VA$ & VA$ & HA$
```

CASE 3: The first source argument is a array; the second source argument is a waveform. For example:

```
CONVL A,WB,WC
```

Then:

```
SC = SB
HC$ = HB$
VC$ = VB$ & VB$ & HB$
```

CASE 4: Both source arguments are arrays. For example:

```
CONVL A,B,WC
```

Then:

```
SC = 1
HC$ = null string
VC$ = null string
```

CORR (Nonresident)

Examples:

```
    CORR X1,X2,Z,TB
100 CORR A(Ø:127),B,C(Ø:255)
200 CORR X,X,Y,T(Ø:63)
```

Syntax Form:

[line no.] **CORR** { floating-point array / floating-point waveform } , { floating-point array / floating-point waveform } ,
{ floating-point array / floating-point waveform } [, { simple numeric variable / floating-point array }]

Descriptive Form:

[line no.] **CORR** source data, source data, target for correlated result [,sine table]

Purpose:

The CORR command performs a non-cyclic, discrete auto- or cross-correlation operation on two source arrays and places the result in a third array.

Discussion:

The CORR command performs a fast correlation operation on two input arrays, placing the result in a third array. If the two source arrays are the same, or contain the same data, the operation is called autocorrelation. If the two input arrays are different, the operation is called cross-correlation. Like the convolution operation, the correlation process can be thought of as successively shifting, multiplying, and integrating the two waveforms to be correlated. But unlike convolution, the correlation process does not reverse in time either waveform before performing the shifting, multiplying, and integrating. In TEK SPS BASIC, however, the

actual operation is done in the frequeny domain by multiplying the fast Fourier transform of one source array by the complex conjugate of the fast Fourier transform of the other source array. (See **The CORR Algorithm** for details.)

Autocorrelation is a useful method of detecting the presence of periodic signals buried in noise. The technique is used in biomedical studies, astronomy, and tone-control systems -- to name just a few applications. On the other hand, cross-correlation is a useful tool for detecting whether a known signal is present in a noisy environment. A common application for cross-correlation is the detection and ranging of radar, sonar, and other transmitted signals.

The format of the CORR command is illustrated by the following example:

```
CORR X,Y,Z,TB
```

The first two arguments (X and Y in the example) are the input arrays to be correlated. They must be floating-point waveforms, arrays, or contiguous subarrays of length $N = 2^m$ for some integer $m \geq 4$. The source arguments must not partially overlap; however, they may be the same array or waveform, which results in the autocorrelation operation. **The CORR command overwrites both source arrays with intermediate results during its execution.** (Because the input arrays are overwritten, you may want to save the original array contents elsewhere before executing CORR.)

The third argument (Z in the example) specifies the destination, where the correlated result is stored. It must be a floating-point waveform, array, or contiguous subarray of length 2N, where N is the length of each source array. **(Since a floating-point array in an extended memory (XM) system is limited to 8K elements, the largest N can be in an XM program is $4K = 2^{12} = 4096$.)** After the correlation is performed, this array contains the result in its last 2N-1 locations. The first element, Z(Ø), is used to store intermediate results. Thus, after CORR executes, the content of the first element is meaningless and should be ignored.

The fourth argument (TB in the example) is optional. If it is specified, the Fourier transform (used by CORR) is table driven. If this argument is a simple numeric variable, that variable is autodimensioned to an array of length N/2 and filled with sine terms needed by the Fourier transform, creating the table. Subsequent executions of CORR do not require regeneration of the table if the source and target array lengths remain unchanged and

the table is not altered. (The table can be generated by the CONVL, RFFT, and RFFT1 commands also. For more information about the table, see the RFFT command.) If the argument is an array, it must be of length N/2. CORR checks the first element of the array. If it is the correct value, the array is assumed to be the proper sine-values table. If the first element is an incorrect value, CORR fills the array with the correct sine values. If the argument is not present, the necessary sine values are recursively generated as needed. This method saves controller memory at the cost of longer execution times.

The Theory of Correlation

The correlation of two time-domain functions, say x(t) and y(t) defined over the interval $\emptyset \leq t \leq T$ can be expressed by the following integrals:

$$z_{12}(t) = (1/T) \int_{\emptyset}^{T-t} x(u)y(u+t)\ du \qquad \text{for } \emptyset \leq t \leq T$$

and

$$z_{21}(t) = (1/T) \int_{\emptyset}^{T-t} y(u)x(u+t)\ du$$

$z_{12}(t)$ applies when waveform x(t) **lags** waveform y(t), and z_{21} applies when waveform x(t) **leads** waveform y(t). Notice that if x(t) = y(t) -- as is the case with autocorrelation -- then $z_{12}(t) = z_{21}(t)$.

In presenting the discrete approximation for correlation, again consider two waveforms, X(k) and Y(k). If these two waveforms are sampled N times with a finite time window such that k = Ø, 1, ..., N-1 then the discrete approximation for **cross-correlation** with X(k) **lagging** Y(k) with time lag n is:

$$Z_{12}(n) = (1/N) \sum_{k=\emptyset}^{N-n-1} X(k)Y(k+n) \qquad \text{for } n = \emptyset, 1, \ldots, N-1$$

and with X(k) **leading** Y(k) is:

$$Z_{21}(n) = (1/N) \sum_{k=\emptyset}^{N-n-1} Y(k)X(k+n) \quad \text{for } n = \emptyset, 1, \ldots, N-1$$

With some algebraic manipulation, it can be shown that $Z_{21}(n)$ is equal to $Z_{12}(-n)$. Thus an alternative to presenting the equations for $Z_{12}(n)$ and $Z_{21}(n)$ as defined above is to present only Z_{12} with n = -N+1, ..., -1, Ø, 1, ..., N-1. This latter approach is used by the CORR command. In summary, the summation for the cross-correlation function Z(n), where the subscripts have been dropped for convenience, is:

$$Z(n) = (1/N) \sum_{k=\emptyset}^{N-n-1} X(k)Y(k+n) \quad \text{for } n = -N+1, \ldots, -1, \emptyset, 1, \ldots, N-1$$

Any terms of X or Y with indices that are less than zero or greater than N-1 are assumed to equal zero.

The autocorrelation function is expressed in the same manner as the cross-correlation function. In this case, Z(n) is the autocorrelation function when X(k) = Y(k) for all values of k. With X(k) = Y(k), then Z(n) = Z(-n); and the autocorrelation is an even function about the time lag of n = Ø.

The CORR Algorithm

The preceding discussion describes one method of computing correlations. It is instructive because it describes the way in which correlation data is formatted. However, rather than evaluating the discrete correlation summation directly, the CORR command uses a faster computational method that takes advantage of the fast Fourier transform (FFT) and the inverse Fourier transform (IFT) algorithms. This method is based on the fact that the correlation of two signals (time domain) is equivalent to the complex conjugate multiplication of their Fourier transforms (frequency domain). Stated in more mathematical terms:

$$F(Z) = F^{*}(X)F(Y)$$

where $F^*(X)$ is the complex conjugate of the Fourier transform of X, and F(Y) is the Fourier transform of Y -- with X and Y being the two time-domain signals. F(Z) is the Fourier transform of Z -- the correlation of X and Y.

Before the CORR command actually performs the correlation of two arrays, it appends N zeroes to each of the input arrays, X and Y. This prevents the cyclic correlation that would be implemented because of the assumed periodicity of the discrete Fourier transform. The zero-appended input arrays, X and Y, are then transformed to the frequency domain via the FFT, and a complex-conjugate multiplication is performed on the resulting arrays. (By complex-conjugate multiplication, it is meant that the imaginary part of one of the Fourier transform pairs is negated before performing the complex multiplication.) Finally, an inverse Fourier transform (IFT) is performed on the product, resulting in the correlation of the original arrays. This procedure is equivalent to the direct evaluation of the non-cyclic correlation summation previously discussed, but the whole process is performed more quickly due to the smaller number of calculations required.

Normalizing the Output

The correlation between two waveforms is often expressed within a normalized range of -1 to +1, with perfect positive or negative correlation having a value of +1 or -1, respectively. The output of the CORR command is not normalized, but normalization is a simple process. Just divide the resulting array by the product of the RMS (Root Mean Square) values of the two source arrays.

Since the source arrays of the CORR command are overwritten during execution, their RMS values must be obtained before the command is executed. The following short routine demonstrates obtaining RMS values, cross-correlation, and normalizing the result. X and Y are the source arrays, and Z is the target array.

```
10 R=RMS(X)*RMS(Y)
15 CORR X,Y,Z
20 Z = Z/R
```

Units and Data Sampling Interval (DSI) Definitions

The following list describes how the vertical and horizontal units and the data sampling interval (DSI) are automatically assigned when a waveform is the target for the CORR command. To simplify this discussion, it is assumed that A and B are arrays and that WA, WB, and WC are waveforms. Accordingly, the following conventions are used:

```
SA:   the data sampling interval for waveform WA
SB:   the data sampling interval for waveform WB
SC:   the data sampling interval for waveform WC

HA$:  the horizontal units for waveform WA
HB$:  the horizontal units for waveform WB
HC$:  the horizontal units for waveform WC

VA$:  the vertical units for waveform WA
VB$:  the vertical units for waveform WB
VC$:  the vertical units for waveform WC
```

Using the above notation,these rules apply when the destination argument (WC) is a waveform. An ampersand (&) indicates that the units strings are concatenated (joined together).

CASE 1: Both source arguments are waveforms. For example:

```
CORR WA,WB,WC
```

Then:

```
SC = SA
HC$ = HA$
VC$ = VA$ & VB$
```

Note: A warning error is issued if the horizontal units for waveform WA do not equal the horizontal units for waveform WB, or if the data sampling interval for waveform WA is not equal to the data sampling interval for waveform WB.

CASE 2: The first source argument is a waveform; the second source argument is an array. For example:

```
CORR WA,B,WC
```

Then:

```
SC = SA
HC$ = HA$
VC$ = VA$ & VA$
```

CASE 3: The first source argument is an array; the second argument is a waveform. For example:

```
CORR A,WB,WC
```

Then:

```
SC = SB
HC$ = HB$
VC$ = VB$ & VB$
```

CASE 4: Both source arguments are arrays. For example:

```
CORR A,B,WC
```

Then:

```
SC = 1
HC$ = null string
VC$ = null string
```

DIFF (Nonresident)

Examples:

```
   DIFF A,A
   DIFF A,B
1Ø DIFF X,Y,FOR
2Ø DIFF C(Ø:511),D(Ø:511)
```

Syntax Form:

[line no.] **DIFF** { floating-point array / floating-point waveform } , { floating-point array / floating-point waveform }
[, { **FOR** / string expression }]

Descriptive Form:

[line no.] **DIFF** source data, target for differentiated result [,forward difference switch]

Purpose:

The DIFF command performs a differentiation of an array or waveform.

Discussion:

The DIFF command performs a two-point or three-point differentiation operation on a source array or waveform, placing the result in the specified target array or waveform. **The source and destination arguments may be the same, in which case the source data is overwritten by the results of the differentiation.**

Differentiation is an important branch of calculus that has numerous applications in physics, chemistry, statistics, electronics, and various other scientific and engineering disciplines. As an example, an array containing distance data can be differentiated once to yield an array of velocity data. The resulting velocity data can again be differentiated to

yield an array of acceleration data. In short, differentiation is useful in any application where it is necessary to determine the instantaneous rate of change of a function (the slope of the curve at a given point).

The format of the DIFF command is illustrated by the following example:

```
DIFF X,Y,FOR
```

The first argument (X in the example) is the input array or waveform to be differentiated. The second argument (Y in the example) specifies the target for the result.

The specified source and target arguments may be the same array or waveform, but they may not partially overlap. If the target array is different from the source array, the data in the source array is left intact. The length of the source and target arrays must be the same and greater than or equal to 3.

The third argument is optional. If it is present and equal to the keyword FOR (or a string expression equaling either "FOR", or "FD"), then a two-point differentiation is performed. If the third argument is omitted, three-point differentiation is performed.

The Theory of Differentiation

According to elementary calculus, the derivative, f'(t), of the function f(t) is defined by the limit:

$$\lim_{\Delta t \to \emptyset} \frac{f(t + \Delta t) - f(t)}{\Delta t}$$

In the preceding definition, t remains fixed while Δt tends to $\emptyset$. When the limit does not exist for a particular value of t, the function has no derivative for that value.

Though the definition of the derivative (in terms of limits) may initially appear a bit abstract, it is possible to get a good intuitive feel for it by considering the idea of slope (see Fig. 1-1). In elementary

terms, the derivative of f(t) is simply the change in f(t) divided by the change in t. In other words, $f'(t) = \Delta f(t)/\Delta t$. If y = f(t), we can rewrite this as:

$$f'(t) = \Delta y/\Delta t \qquad \text{or} \qquad dy/dt$$

where Δ or "d" denotes a **very small** change in the value of t and y.

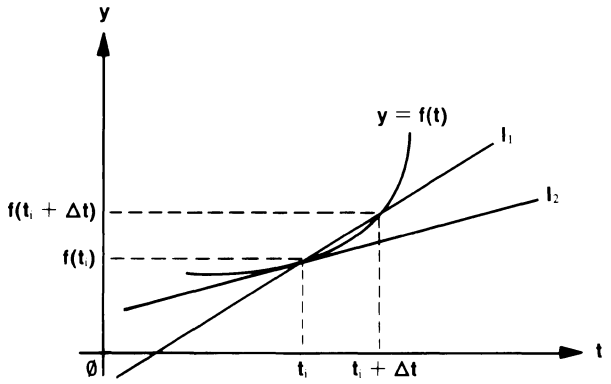


Fig. 1-1. The derivative of f(t) at t_i yields the slope (l_2) at t_i.

From examining Fig. 1-1, it is seen that the slope of the line l_1 over the interval Δt is:

$$[f(t_i + \Delta t) - f(t_i)] / \Delta t$$

However, as Δt approaches Ø, the slope ($\Delta f(t)/\Delta t$) more nearly approaches the slope of l_2, and thus the derivative of f(t) at the point t_i yields a slope that defines l_2.

The DIFF Algorithm

Two-point Method. In theory, it is possible to talk about the derivative of a function in terms of a limit, where the interval (Δt) approaches zero. However, when discussing numerical differentiation as implemented on a signal processing system, the interval (over which the slope is computed) cannot approach zero. Instead, the interval must be no smaller than the time segment between adjacent elements in the waveform array. This method

of numerical differentiation is known as the "two-point" derivative and can be demonstrated by a statement such as:

```
DIFF X,Y,FOR
```

where the optional keyword FOR designates a **for**ward-difference calculation for two-point differentiation. Executing the above command causes an element-by-element differentiation which is performed according to the following scheme:

$$Y(n) = \frac{X(n+1) - X(n)}{\Delta t} \qquad \text{for } n = 0, \ldots, N-2$$

$$Y(N-1) = Y(N-2)$$

where X and Y are the source and target arrays, respectively, both of length N. The data sampling interval (the time between the acquisition of adjacent array elements in a TEK SPS BASIC waveform) is Δt. It is normally equal to 10 times the horizontal scale factor divided by the array length N. In the case of an array, Δt is always equal to 1. The process of two-point differentiation is diagrammed in Fig. 1-2.

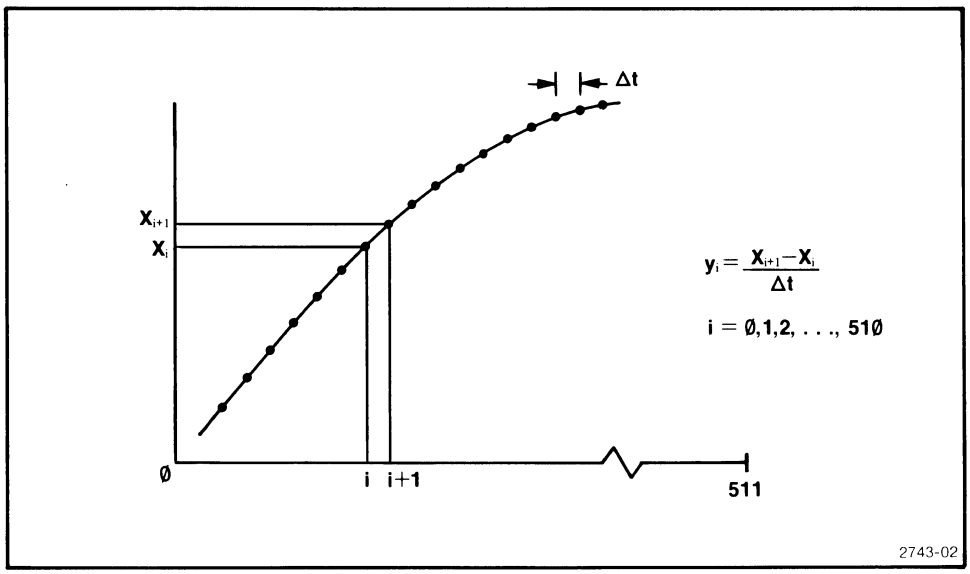


Fig. 1-2. Two-point derivative of X, a 512-element array, at element i.

Three-point Method. A more complicated method of differentiating an array of values on a signal processing system, is to compute the slope between the datum immediately preceding and immediately following the array element where the slope is to be computed. This method is known as "three-point differentiation" and can be demonstrated by the statement:

```
DIFF X,Y
```

The absence of the optional third argument specifies the three-point derivative. Executing the above command causes an element-by-element differentiation which is performed according to the following scheme:

Starting value: $$Y(0) = \frac{-3X(0) + 4X(1) - X(2)}{2\Delta t}$$

Intermediate values: $$Y(n) = \frac{X(n+1) - X(n-1)}{2\Delta t} \quad \text{for } n = 1 \text{ to } N-2$$

Ending value: $$Y(N-1) = \frac{X(N-3) - 4X(N-2) + 3X(N-1)}{2\Delta t}$$

where X and Y are the source and target arrays, respectively, both of length N. Again, Δt is the data sampling interval (the time between the acquisition of adjacent array elements in the waveform). The process of three-point differentiation is diagrammed in Fig. 1-3.

Two-point Versus Three-point Differentiation

For arrays where large transitions occur over intervals greater than three array elements, the three-point derivative exhibits the least analytic error in estimating the slope at a given point. Thus for smoothly varying functions (e.g., a sine wave), the three-point derivative is the most accurate means of differentiating the function.

When array values exhibit large transitions within intervals of three or fewer adjacent array elements, the two-point derivative may provide a better slope estimate within the interval of transition. Square waves,

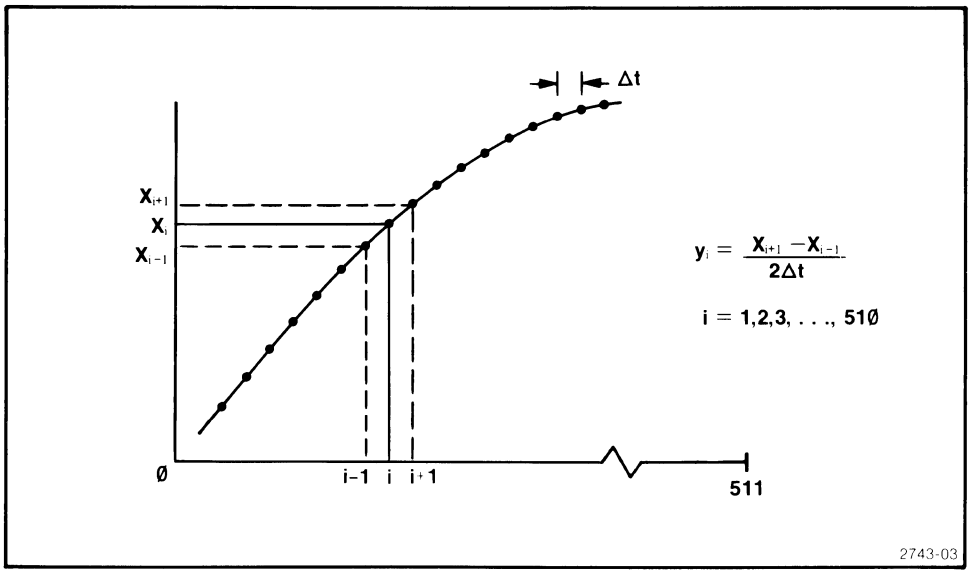


Fig. 1-3. Three-point derivative of X, a 512-element array, at element i.

steps, impulses, and other functions containing large transitions over limited intervals are best differentiated with the two-point algorithm.

Differentiation of a time-domain waveform is effectively the same as digitally filtering the waveform. Hence, the results of differentiation reflect the response function of the chosen scheme of differentiation. The response functions for two-point and three-point differentiation are shown in Fig. 1-4 and Fig. 1-5. In comparing these two figures, notice that the two-point function is maximum at the Nyquist frequency of $1/(2\Delta t)$. Also, notice in the equations which follow that the phase of the three-point function is zero at all frequencies.

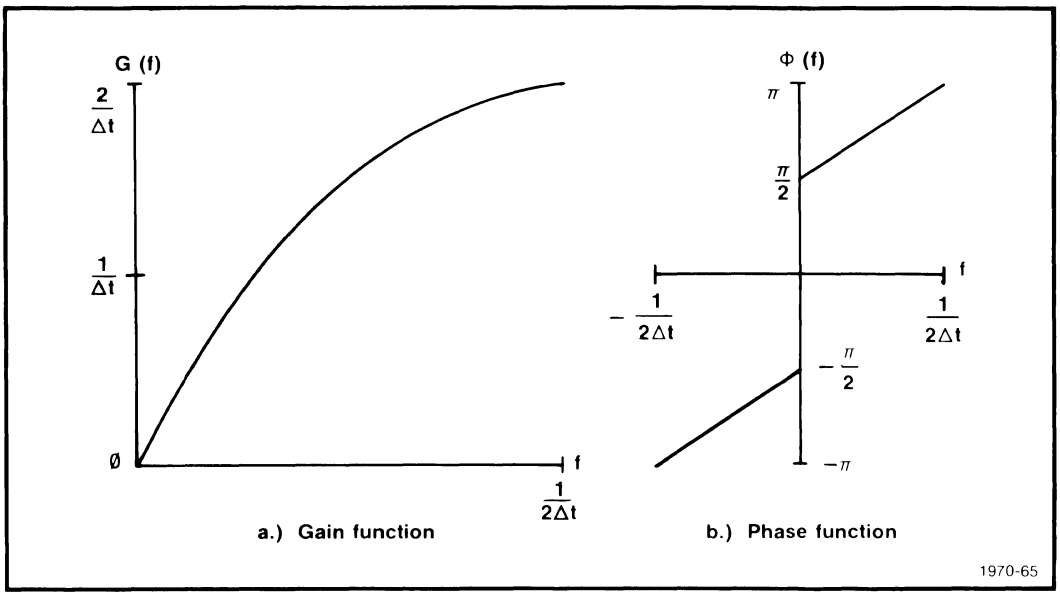


Fig. 1-4. Frequency response of the two-point derivative.

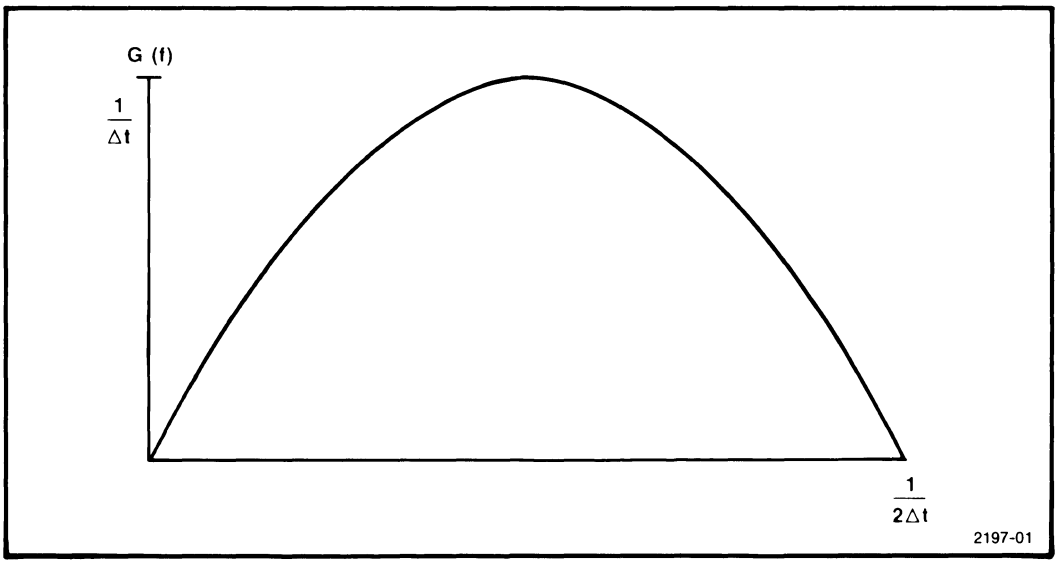


Fig. 1-5. Frequency response of the three-point derivative.

The formulas applicable to the two-point derivative and its frequency response function (Fig. 1-4) are as follows, where X is the function to be differentiated, X' is the derivative of X, and Δt is the waveform's data sampling interval:

Impulse response: $$X'(n) = \frac{X(n+1) - X(n)}{\Delta t}$$

Frequency response: $$H(f) = G(f)e^{j\Phi(f)}$$

Gain: $$G(f) = (2/\Delta t)|\sin(\pi f \Phi t)| \quad \text{for } -1/(2\Delta t) \leq f < 1/(2\Delta t)$$

Phase: $$\Phi(f) = \begin{cases} \pi\Delta t[f + 1/(2\Delta t)] & \text{for } -1/(2\Delta t) \leq f < \emptyset \\ \pi\Delta t[f - 1/(2\Delta t)] & \text{for } \emptyset \leq f < 1/(2\Delta t) \end{cases}$$

The formulas applicable to the three-point derivative at intermediate points and its frequency-response function (Fig. 1-5) are as follows:

Impulse response: $$X'(n) = \frac{X(n+1) - X(n-1)}{2\Delta t}$$

Frequency response: $$H(f) = G(f)\ e^{j\Phi(f)}$$

Gain: $$G(f) = (1/\Delta t)|\sin(2\pi f \Delta t)|$$

Phase: $$\Phi(f) = \emptyset \quad \text{for } -1/(2\Delta t) \leq f < 1/(2\Delta t)$$

Units and Data Sampling Interval (DSI) Definitions

The following list describes how the vertical and horizontal units and data sampling interval (DSI) are automatically assigned when a waveform is the target for the DIFF command. To simplify this discussion, it is assumed that A is an array, while WA and WB are waveforms. Accordingly, the following conventions are used:

SA: the data sampling interval for waveform WA
SB: the data sampling interval for waveform WB

HA$: the horizontal units for waveform WA
HB$: the horizontal units for waveform WB

VA$: the vertical units for waveform WA
VB$: the vertical units for waveform WB

Using the above notation,these rules apply when the target (WB) is a waveform. An ampersand (&) indicates that the units strings are concatenated (joined together).

CASE 1: The source argument is a waveform. For example:

```
DIFF WA,WB
```

Then:

```
SB = SA
HB$ = HA$
VB$ = VA$ & "/" & HA$
```

CASE 2: The source argument is an array. For example:

```
DIFF A,WB
```

Then:

```
SB = 1
HB$ = null string
VB$ = null string
```

INT (Nonresident)

Examples:

```
    INT A,A
100 INT C,D
200 INT X(0:511),Y(0:511)
```

Syntax Form:

[line no.] **INT** { floating-point array / floating-point waveform } , { floating-point array / floating-point waveform }

Descriptive Form:

[line no.] **INT** source data, target for integrated result

Purpose:

The INT command performs an integration of an array or waveform.

Discussion:

The INT command performs an integration operation on a source array or waveform, placing the results in the specified destination array or waveform. **The source and target arguments may be the same, in which case the source data is overwritten by the results of the integration.**

Integration is an important branch of calculus that has numerous applications in physics, chemistry, electronics, and various other scientific and engineering disciplines. As an example, an array representing acceleration data can be integrated to get an array of velocity data. Similarly, integrating an array of velocity data yields distance data. In short, integration is useful whenever you wish to determine the area under a curve, or determine the energy associated with a pulse.

The format of the INT command is illustrated by the following example:

```
INT X,Y
```

The first argument (X in the example) is the input array or waveform to be integrated. The second argument (Y in the example) specifies the target for the integrated result.

The specified source and target arguments may be the same array or waveform, but they may not partially overlap. If the destination array is different from the source array, the data in the source array is left intact. The length of the source and target arguments must be the same and greater than or equal to 3.

The Theory of Integration

Most elementary calculus texts define the definite integral of a function in terms of the "antiderivative" of that function. For example, since the derivative of $t^{n+1}/(n+1)$ is t^n, then the antiderivative of t^n is $t^{n+1}/(n+1)$. Thus the definite integral of t^n between two points, a and b, is

$$\int_a^b t^n dt = \left.\frac{t^{n+1}}{n+1}\right|_a^b = \frac{b^{n+1}}{n+1} - \frac{a^{n+1}}{n+1}$$

which represents the area underneath the curve t^n between a and b.

While the idea of the antiderivative is valid if an antiderivative can be found, there are still many functions for which an antiderivative cannot be found by any known method -- even though its definite integral has a specific value. This leads us to the concept of numerical integration, a technique that allows you to find an estimate of the definite integral for virtually any continuous function.

The idea of numerical integration is a simple concept, and in fact, is often used to introduce the fundamentals of integral calculus. One of the easiest methods of numerical integration is to simply estimate the area under the curve (the graph of the function) by dividing the curve into segments and then summing the areas of the rectangles under the curve. For example, in Fig. 1-6a, we see a graph of the function $y = t^2$, in which

five equal-width rectangles have been drawn under the curve. By computing the area of each rectangle (width times height) and summing all the computed areas, we can get a rough estimate of the area under the curve. Actually, the value described by this summation will be less than the area under the curve, but in many cases, the error is not significant if the width of the rectangles are kept small enough.

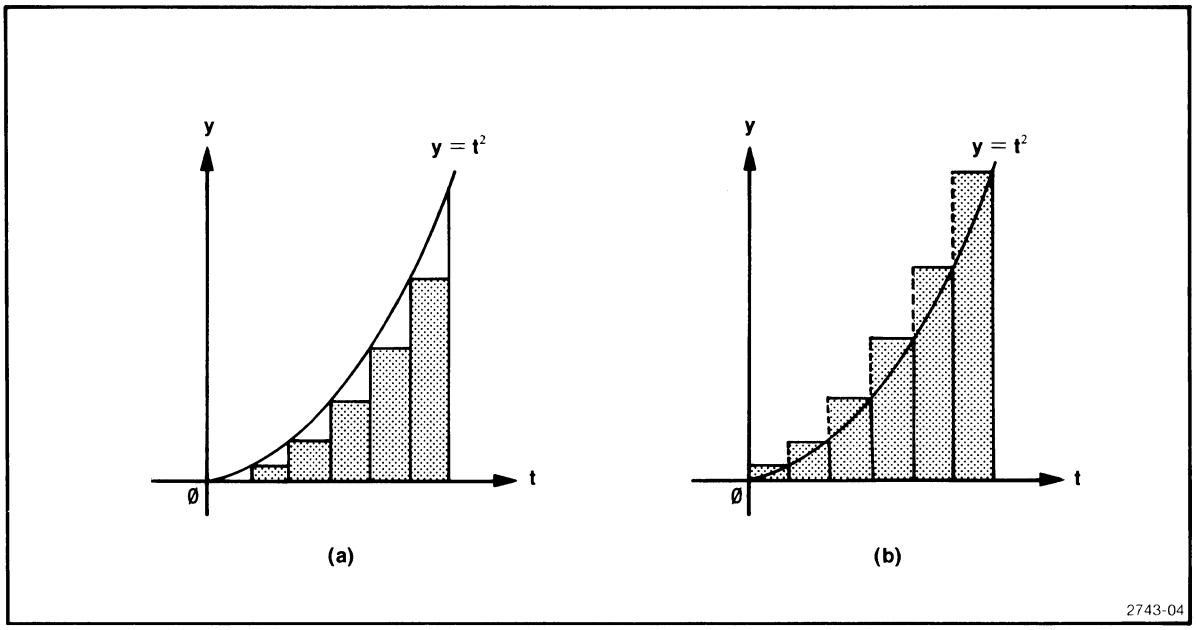


Fig. 1-6. Numerical integration of the function $y = t^2$ by estimating (a) the lower bound, and (b) the upper bound of the area under the curve.

Even more accuracy can be gained by summing the areas of the rectangles shown in Fig. 1-6b to get an upper bound on the total area under the curve. The true area is then very nearly equal to the mean of the lower-bound of the area (Fig. 1-6a) and the upper-bound of the area (Fig 2-6b).

The accuracy of the preceding method -- computing upper and lower bounds and taking the mean -- can also be achieved by a method known as the trapezoidal rule. Like the previously described method, the trapezoidal rule involves dividing the horizontal axis of the function into a number of equally spaced intervals. However, rather than drawing rectangles under and above the curve, the trapezoidal rule requires the construction of trapezoids whose upper vertices touch the curve at the endpoints of the horizontal intervals (see Fig. 1-7). The definite integral of the function

is then found by summing the area of each trapezoid under the curve between the limits of integration.

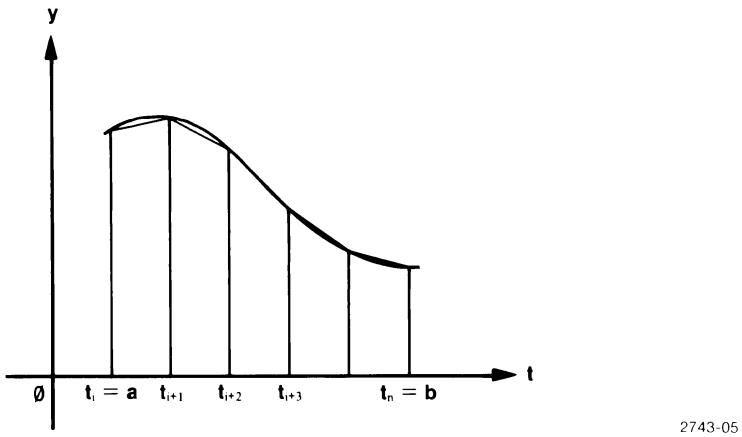


Fig. 1-7. Illustration of the trapezoidal rule for numerical integration.

There are other methods of numerical integration. (One common method is Simpson's rule, which requires the construction of rectangles under the curve to be integrated, such that the curve intersects each rectangle in the center of its upper edge.) In each of these methods, however, the answers are only approximate, and to achieve high accuracy, the width of each rectangle or trapezoid must be sufficiently small or the change in the value of the function must be small over the selected interval. Thus, as the width of each rectangle or trapezoid approaches zero, the approximate value obtained by numerical integration approaches the true value obtained by finding the antiderivative.

The INT Algorithm

In theory, it is possible to talk about the integral of a function in terms of rectangles or trapezoids whose width (Δt) approaches zero. However, when discussing numerical integration as implemented on a signal processing system, the interval (width of each rectangle or trapezoid) cannot approach zero. Instead, the width of the quadrilaterals can be no smaller than the time segment between adjacent elements in the waveform array.

The INTegrate command in TEK SPS BASIC uses the trapezoidal rule for integration. The format of INT is illustrated by the following example:

```
INT X,Y
```

Here, an array (or waveform) X of length N is integrated and the result is placed in array Y according to the following equations:

$$Y(\emptyset) = \emptyset$$

$$Y(n) = Y(n-1) + \frac{\Delta t[X(n-1) + X(n)]}{2} \quad \text{for } n = 1, 2, \ldots, N-1$$

Upon examining the second equation, it is seen that this is the familiar trapezoidal rule. Notice that Δt is the width of each trapezoid and [X(n-1) + X(n)]/2 is the average height of each trapezoid. The source waveform's data sampling interval is Δt (the time between adjacent array elements in a TEK SPS BASIC waveform); it is normally equal to 10 times the horizontal scale factor divided by the array length, N. In the case of an array, Δt is always equal to 1.

Integrating a waveform is the same as digitally filtering the waveform. The frequency-domain formulas that describe the gain and phase functions associated with integration are as follows:

Gain: $$G(f) = \frac{\Delta t\,|\sin(2\pi f \Delta t)|}{2[1 - \cos(2\pi f \Delta t)]} \quad \text{for} \quad \frac{-1}{2\Delta t} \leq f < \frac{1}{2\Delta t}$$

Phase: $$\Phi(f) = \begin{cases} \pi/2 & \text{for} \quad -1/(2\Delta t) \leq f < \emptyset \\ -\pi/2 & \text{for} \quad \emptyset \leq f < 1/(2\Delta t) \end{cases}$$

The gain and phase functions described by these formulas are plotted in Fig. 1-8.

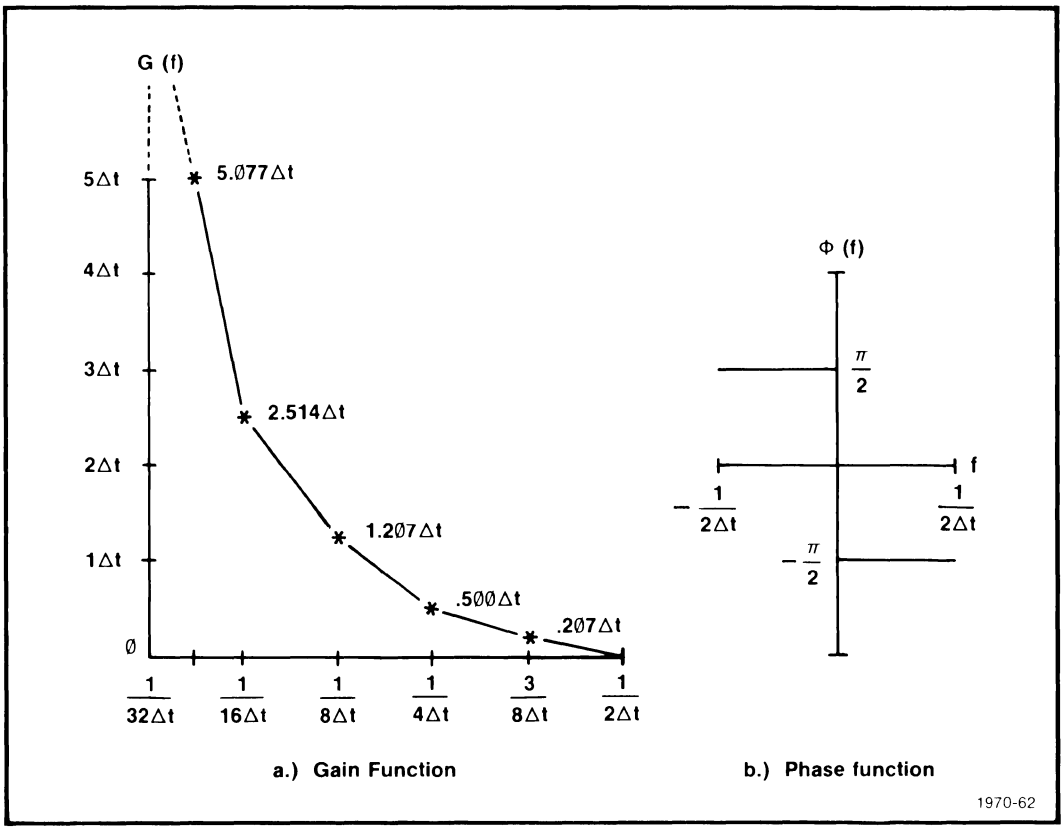


Fig. 1-8. Frequency response functions for integration.

Units and Data Sampling Interval (DSI) Definitions

The following list describes how the vertical and horizontal units and data sampling interval are automatically assigned when a waveform is the target for the INT command. To simplify this discussion, it is assumed that A is an array, while WA and WB are waveforms. Accordingly, the following conventions are used:

SA: the data sampling interval for waveform WA
SB: the data sampling interval for waveform WB

HA$: the horizontal units for waveform WA
HB$: the horizontal units for waveform WB

VA$: the vertical units for waveform WA
VB$: the vertical units for waveform WB

Using the above notation,these rules apply when the target (WB) is a waveform. An ampersand (&) indicates that the units strings are concatenated (joined together).

CASE 1: The source argument is a waveform. For example:

```
INT WA,WB
```

Then:

```
SB = SA
HB$ = HA$
VB$ = VA$ & HA$
```

CASE 2: The source argument is an array. For example:

```
INT A,WB
```

Then:

```
SB = 1
HB$ = null string
VB$ = null string
```

POLAR (Nonresident)

Examples:

```
     POLAR A,B
1ØØ  POLAR X,Y,DL
11Ø  POLAR A(1Ø:2Ø),B(7:17),N/8
```

Syntax Form:

[line no.] **POLAR** { floating-point variable / floating-point array / floating-point waveform } , { floating-point variable / floating-point array / floating-point waveform }

[,expression]

Descriptive Form:

[line no.] **POLAR** real source data and target for magnitude component,
imaginary source data and target for phase component [,delay estimate]

Purpose:

The POLAR command performs a rectangular-to-polar conversion.

Discussion:

Normally, the RFFT command returns its results in rectangular form (with the array components being real and imaginary numbers). Quite often though, the results can be interpreted more easily if they are in polar form (with the array components being magnitude and phase numbers). The POLAR command performs this conversion.

The format of the POLAR command is illustrated by the following example:

```
POLAR X,Y,DL
```

The first two arguments (X and Y in the example) correspond, respectively, to the real and imaginary components of the complex data (as returned by the RFFT command) to be converted. If these two arguments are arrays or waveforms, they must be of equal length.

After the POLAR command has executed, the two source arguments contain the results, with the magnitude component stored in the first argument (X) and the phase component stored in the second argument (Y). The first element of each array or waveform argument -- X(Ø) and Y(Ø) in the above example -- correspond to the DC (direct current) term. Succeeding elements of these arguments indicate the magnitude and phase numbers for increasing frequencies.

The optional third argument (DL in the example) represents phase delay and designates delay removal. If this argument is present, the phase is continuous; if it is omitted, the phase is discontinuous.

Theory of Operation

The operation of the POLAR command is best understood by considering the following example:

```
POLAR RX,IX
```

Assuming RX and IX are arrays of equal length N, the arrays are redefined as follows:

$$RX(n) = \sqrt{RX^2(n) + IX^2(n)} \quad \text{for } n = Ø, 1, \ldots, N-1$$

$$IX(n) = \arctan[IX(n)/RX(n)]$$

The phase information, returned in IX, is discontinuous and is expressed within a range of $-\pi$ to $+\pi$ radians.

Continuous phase can be provided for arrays or waveforms by specifying the optional third argument in a statement such as:

```
POLAR RX,IX,DL
```

assuming that RX and IX are waveforms of length N (where N is greater than 1), DL represents the phase delay expression which must evaluate to a value

less than N/2. Then, the phase information is "unwrapped" module 2π and the value:

$$2\pi*DL*DS*n \quad \text{for } \emptyset \leq n < N-1$$

is removed from the phase data. Here DS is the data sampling interval of RX if RX is a waveform. If RX is not a waveform and IX is a waveform, DS is the data sampling interval of IX. If neither RX nor IX is a waveform, DS equals 1.

If the value of the phase delay expression (DL in the example) is an adequate estimate of the true delay, then subtracting $2\pi*DL*DS*n$ from the phase has the effect of removing the linear component which has a slope of 2π times the phase delay. This makes the phase non-linearity or fluctuations much more apparent. If the value of the phase delay expression is zero, then nothing is subtracted, yet a continuous phase is still provided.

See the discussion of the RFFT command for examples of phase output of the POLAR command.

Units and Data Sampling Interval (DSI) definitions

The following list describes how the vertical and horizontal units and the data sampling interval (DSI) are automatically assigned when a waveform is the target for the POLAR command. To simplify this discussion, it is assumed that A and B are arrays and WA and WB are waveforms. Accordingly, the following conventions are used:

SA: the data sampling interval for waveform WA
SB: the data sampling interval for waveform WB

HA$: the horizontal units for waveform WA
HB$: the horizontal units for waveform WB

VA$: the vertical units for waveform WA
VB$: the vertical units for waveform WB

Using the above notation, these rules apply when the target for the magnitude is a waveform (WA) and/or when the target for the phase data is a waveform (WB).

CASE 1: Both source/target arguments are waveforms. For example:

```
POLAR WA,WB
```

Then:

SA, SB, HA$, HB$, and VA$ are unchanged.
VB$ is changed to "RAD"

CASE 2: The first source/target argument is a waveform; the second is an array. For example:

```
POLAR WA,B
```

Then:

SA, HA$, and VA$ are unchanged.

CASE 3: The first source/target argument is an array; the second is a waveform. For example:

```
POLAR A,WB
```

Then:

SB and HB$ are unchanged
VB$ is changed to "RAD"

RFFT (Nonresident)

Examples:

```
   RFFT A,B,C,INV
10 RFFT X(Ø:63),Y,Z(Ø:32),T,INV
20 RFFT X,R,I
30 RFFT M,N,Q,T,DIR
```

Syntax Form:

[line no.] **RFFT** { floating-point array / floating-point waveform } , { floating-point array / floating-point waveform } ,
{ floating-point array / floating-point waveform } [, { simple numeric variable / floating-point array }]
[, { **DIR** / **INV** / string expression }]

Descriptive Form:

[line no.] **RFFT** time domain data, real component of frequency domain data, imaginary component of frequency domain data [,sine table] [,direct or inverse transform switch]

Purpose:

The RFFT command performs a multi-argument fast Fourier transform on real-valued data via a power-of-two algorithm.

Discussion:

The RFFT command can perform either an FFT (fast Fourier transform) or an IFT (inverse Fourier transform) operation. The fast Fourier transform is an algorithm for quickly computing the discrete Fourier transform. By means of the FFT, a waveform (time domain) or other time-series data can

be converted to a corresponding spectrum (frequency domain). Normally, the FFT data is expressed in terms of real and imaginary data (rectangular format). However, this FFT data can be converted to a series of magnitude and phase data (polar format) via the POLAR command.

The IFT is just the inverse of the FFT operation. That is, the IFT converts an array of spectral data (frequency domain) into an array of waveform data (time domain). The FFT and IFT find numerous applications in such areas as frequency-response estimation, signature analysis, harmonic distortion measurements, and digital filtering.

The format of the RFFT command is illustrated by the following example, which performs an FFT operation:

```
RFFT X,RX,IX
```

The first argument (X in the example) corresponds to the real-valued, time-domain data. It must be a waveform, array, or contiguous subarray of length $N = 2^m$ for some integer $m \geq 4$. (Since a floating-point array in an extended memory (XM) system is limited to 8K elements, the largest N can be in an XM program is $8K = 2^{12} = 4096$.)

The second and third arguments (RX and IX in the example) correspond, respectively, to the real and imaginary components of the discrete Fourier coefficients (the frequency-domain data). These arguments will contain the output of the RFFT command. They must be waveforms, arrays, or contiguous subarrays of length N/2 + 1. (Note that the first elements of RX and IX correspond to the DC terms, while the last elements of RX and IX correspond to the Nyquist frequency.)

The presence of an optional fourth argument specifies that the RFFT computation will be table driven. In its absence, sine terms (required by the FFT algorithm) are recursively generated as needed, saving computer memory space but somewhat slowing the computations. If the fourth argument is a simple variable, it is auto-dimensioned to a floating-point array of length N/4, and it will be filled with the sine values needed by the transform (the values correspond to 1/4 of a cycle of a negated sine wave). If the fourth argument is otherwise specified, it must be a floating-point array or subarray of length N/4. (In this latter case, it is assumed that the array already contains the table of proper sine values. However, one element within the array is checked to see if it contains the proper sine value; if it does not, the entire table of sine values is regenerated.)

The presence of an optional keyword or string expression specifies whether a forward or inverse Fourier transform is to be performed. If the keyword DIR or a string expression equal to "DIR" is present (or if no keyword or string is present -- as in the preceding example), a direct FFT will be performed. In this case, the time-domain data in the first argument is the source, and the discrete Fourier coefficients are returned in the second and third arguments. If the keyword INV or a string expression equal to "INV" is present, the inverse Fourier transform (IFT) is performed. In this latter case, the second and third arguments are the source arrays and are assumed to contain the real and imaginary components of the discrete Fourier coefficients; the time-domain data is then returned in the first argument. **Data in the second and third arguments are overwritten by intermediate results when the INV function is performed.**

An example of an RFFT command that performs an IFT operation is:

```
RFFT X,RX,IX,TB,INV
```

The real and imaginary data (in RX and IX respectively) is inverse Fourier transformed and the resulting time-domain data is placed in array X. The optional argument INV specifies that the inverse Fourier transform is to be performed and the optional argument TB specifies that it is to be table driven with array TB.

The Theory of the Discrete Fourier Transform

The RFFT command performs a fast calculation of the DFT (discrete Fourier transform). The DFT can be expressed mathematically by the following summation:

$$X_d(n) = \Delta t \sum_{k=\emptyset}^{N-1} x(k)e^{-j2\pi nk/N} \quad \text{for } n = \emptyset, 1, \ldots, N/2$$

In the above equation, N refers to the length of the time-domain argument, Δt is the sampling interval between the elements of the time-domain data, and n is an index used in generating the various Fourier coefficients. Also, in accordance with usual math notation, e (2.718281828...) is the base of the natural logarithm, and j is the square root of -1. $X_d(n)$ is the nth Fourier coefficient, and x(k) refers to the kth element of the time-domain input array.

The summation is computed by summing $x(k)e^{-j2\pi nk/N}$ as k ranges from Ø to N-1; this is done for each value of n, beginning with Ø and ending with N/2. Each value of n thus defines a corresponding element of the final DFT result.

Theoretically, the discrete Fourier transform contains spectral components for negative frequencies as well as positive ones. In this case, the index n would range from -N/2+1 to N/2. However, by assuming that the time domain contains only real-valued data (as opposed to complex data), the spectral components for the negative frequencies can be determined from the components for the positive frequencies. Specifically, the spectral components at the negative frequencies are complex conjugates of the spectral components at the positive frequencies. This means that the components at the negative frequencies contain half of the total spectral energy. Therefore, the spectral output of the RFFT command has only half of the expected amplitude value since it shows only the information at the positive frequencies.

The X(n)'s are the N/2+1 positive complex frequency components from DC through the Nyquist frequency. These complex Fourier coefficients are stored in the two target arrays -- one for the real part and one for the imaginary part. If RX and IX are the destination arguments of the RFFT command, then the format of data storage is as follows:

RX(Ø) = DC term	IX(Ø) = Ø
RX(1) = Real part, 1st Fourier coeff.	IX(1) = Imag. part, 1st Fourier coeff.
RX(2) = Real part, 2nd Fourier coeff.	IX(2) = Imag. part, 2nd Fourier coeff.
RX(3) = Real part, 3rd Fourier coeff.	IX(3) = Imag. part, 3rd Fourier coeff.
RX(4) = Real part, 4th Fourier coeff.	IX(4) = Imag. part, 4th Fourier coeff.
.	.
.	.
.	.
RX(N/2) = Nyquist term	IX(N/2) = Ø

The inverse Fourier transform (IFT) is performed with the RFFT command by specifying the keyword INV, as illustrated in the following example:

```
RFFT X,RX,IX,INV
```

It is assumed that arrays RX and IX contain the real and imaginary components respectively of each of the Fourier coefficients. These Fourier components must be formatted as follows:

RX(Ø) = DC term	IX(Ø) = Ø
RX(1) = Real part, 1st Fourier coff	IX(1) = Imag. part, 1st Fourier coeff.
RX(2) = Real part, 2nd Fourier coeff.	IX(2) = Imag. part, 2nd Fourier coeff.
RX(3) = Real part, 3rd Fourier coeff.	IX(3) = Imag. part, 3rd Fourier coeff.
RX(4) = Real part, 4th Fourier coeff.	IX(4) = Imag. part, 4th Fourier coeff.
.	.
.	.
.	.
RX(N/2) = Nyquist term	IX(N/2) = Ø

Notice that this is the same format as output by the direct RFFT command.

The inverse Fourier transform can be expressed mathematically by the following summation:

$$x(k) = \Delta f \sum_{n=\emptyset}^{N-1} X_d(n) e^{j2\pi nk/N} \quad \text{for } k = \emptyset, 1, \ldots, N-1$$

In the above equation, N refers to the length of the time-domain array argument, Δf is the data sampling interval of the RX array (containing the real components), and $X_d(n)$ again refers to the nth positive complex Fourier coefficient. Here, $X_d(n)$ = RX(n) + jIX(n) where RX(n) is the nth element of the RX array (containing the real components) and IX(n) is the nth element of the IX array (containing the imaginary components). The $X_d(n)$'s for N/2 < n < N-1 are defined by $X_d(N-n) = X_d^*(n)$ with * denoting complex conjugation; x(k) refers to the kth element of the real time-domain data that results from the IFT operation.

The summation is computed by summing $X_d(n)e^{j2\pi nk/N}$ as n ranges from Ø to N-1. Each value of k thus defines a corresponding element of the final output array, x, containing the real time-domain data.

The FFT/IFT Algorithm

The preceding discussion describes one method for computing the discrete Fourier transform. It is instructive because it describes the way in which the FFT data is formatted. In the case of TEK SPS BASIC, however, there is a much faster method for computing the Fourier transform: it is the Sande-Tukey decimation-in-frequency algorithm. It uses a floating-point table of length N/4 (where N is the number of time-domain data points)

containing a one-quarter cycle of negative sine-wave data to generate the necessary complex exponentials.

The inverse Fourier transform uses the same algorithm as the direct transform except that some of the initialization parameters are changed. For more information on the Sande-Tukey FFT/IFT algorithm, refer to Section 7 in the Tektronix concept book entitled **The FFT: Fundamentals and Concepts** (Tektronix part number 070-1754-00).

An Example Program:

Applying the RFFT command to a waveform aids in understanding the basic concept of the Fourier transform. Figures 1-9 and 1-10 are graphs of a sine and cosine waveform, respectively. The sine wave (Fig. 1-9) has a period of 83.3 microseconds, and the cosine wave (Fig. 1-10) has a period of 62.5 microseconds. By calculating the reciprocal of the periods, we find that their frequencies are 12,000Hz and 16,000Hz. Notice also that the amplitude of the waveforms are 1 volt and .75 volt respectively.

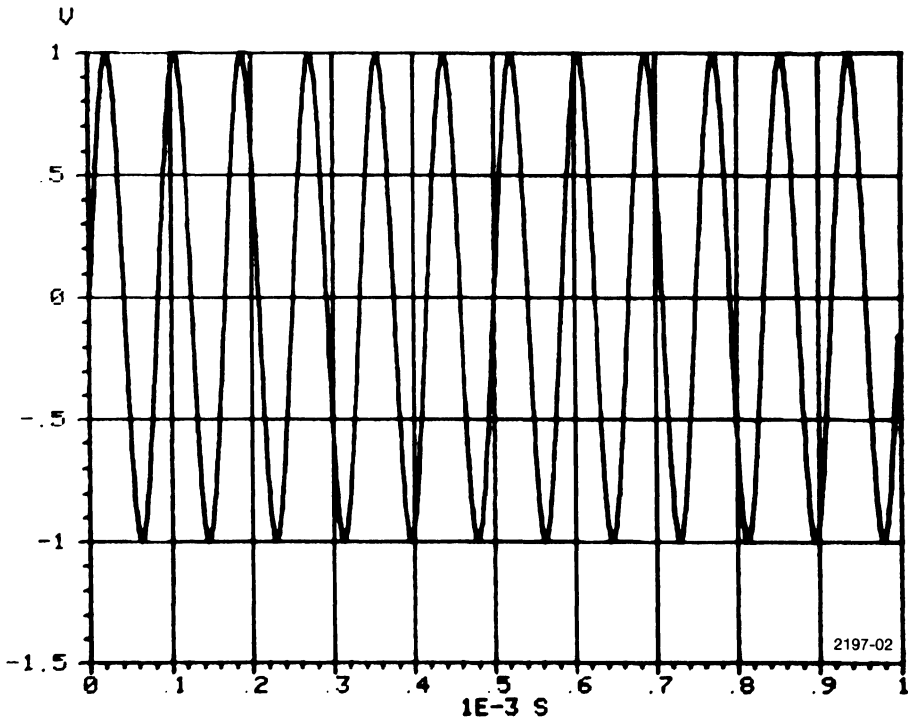


Fig. 1-9. Sine wave with frequency of 12,000 Hz.

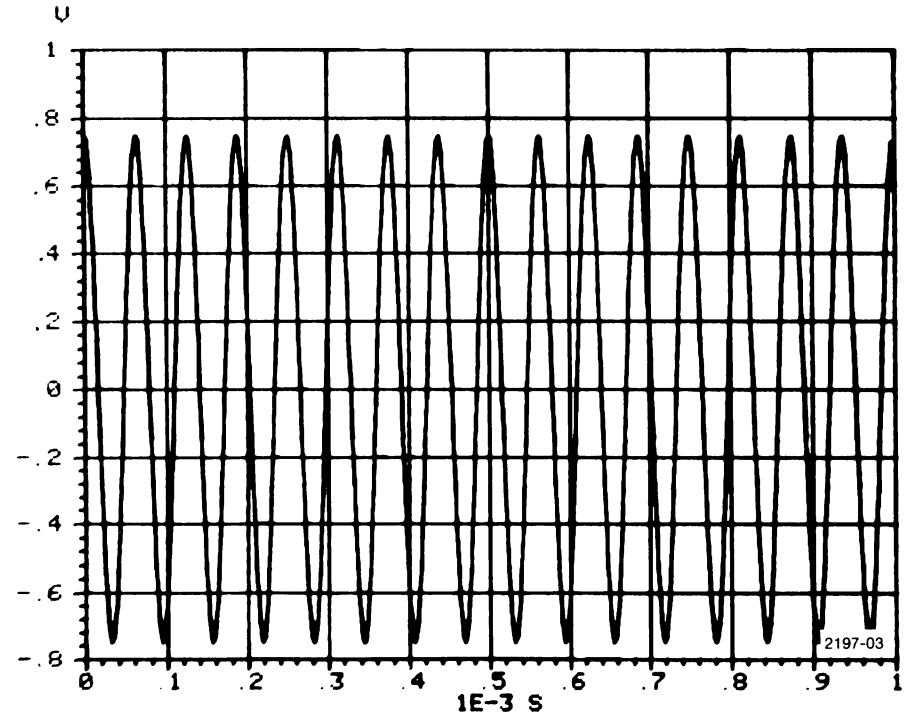


Fig. 1-10. Cosine wave with frequency of 16,000 Hz.

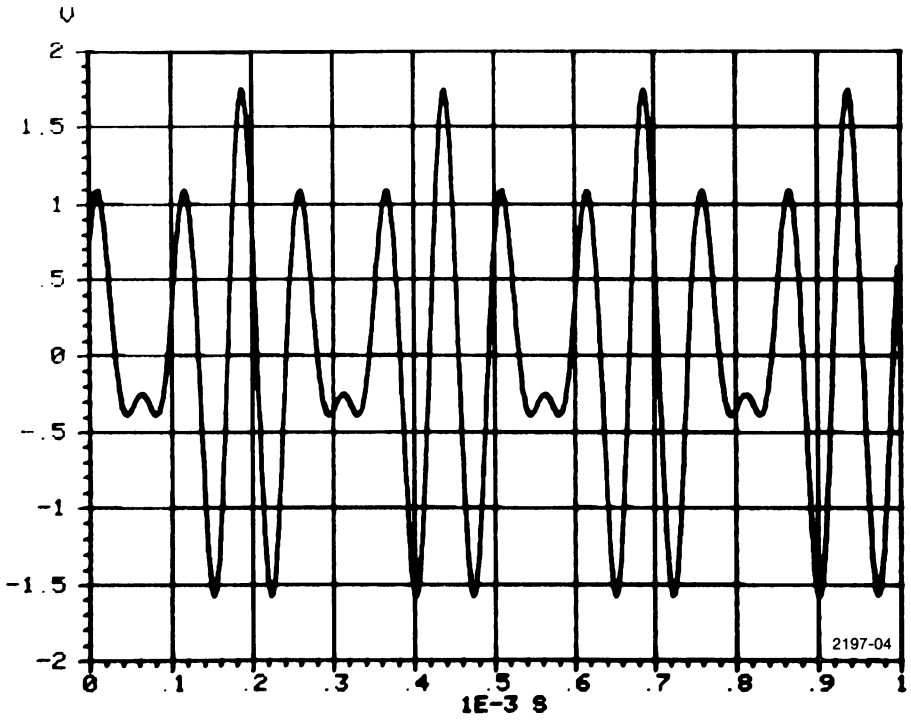


Fig. 1-11. Sum of waveforms in Figures 1-9 and 1-10.

Figure 1-11 is the sum of these two waveforms. This is the waveform that is transformed into the frequency domain by the program that is listed here. For simplicity, ideal waveforms are assumed. In real life, such things as aliasing, leakage, noise, etc. must be considered. For a complete description of the Fourier transform and its analysis, see the Tektronix publication **The FFT: Fundamentals and Concepts.**

```
10 REM *** DEFINE WAVEFORMS ***
20 WAVEFORM WA IS AA(511),SA,HA$,VA$
30 WAVEFORM W1 IS A1(256),S1,H1$,V1$
40 WAVEFORM W2 IS A2(256),S2,H2$,V2$
50 REM     CREATE SOURCE WAVEFORM WA
60 PI=3.14159
70 AA=1/512
80 INT AA,AA
90 AA=SIN(2*PI*AA*12)+.75*COS(2*PI*AA*16)
100 SA=1E-03/512
110 HA$="S"
120 VA$="V"
130 REM *** CONVERT TO FREQUENCY DOMAIN ***
140 RFFT WA,W1,W2
150 REM     CONVERT TO MAGNITUDE AND PHASE
160 POLAR W1,W2
170 REM *** COMPUTE MAGNITUDE, FREQUENCY, AND PHASE ***
180 REM     FIND SUBSCRIPT OF MAXIMUM FREQUENCY COMPONENT
190 REM     IN MAGNITUDE ARRAY
200 C=CRS(W1,MAX(W1))
210 GOSUB 540\REM SUBROUTINE TO CALCULATE AND PRINT
220 REM     FIND OTHER FREQUENCY COMPONENT
230 C=CRS(A1(C+1:256),MAX(A1(C+1:256)))
240 GOSUB 540\REM SUBROUTINE TO CALCULATE AND PRINT
250 END
500 REM     SUBROUTINE TO CALCULATE AND PRINT THE
510 REM     MAGNITUDE, FREQUENCY, AND PHASE OF THE COMPONENT
520 REM     GIVEN MAGNITUDE AND PHASE WAVEFORMS (W1 AND W2),
530 REM     LOCATION OF THE COMPONENT (C), AND PI=3.14159
540 M=W1(C)*S1*2
550 F=C*S1
560 PRINT "MAGNITUDE IS",M;" V"
570 PRINT "FREQUENCY IS",F;" ";H1$
```

```
580 PRINT "PHASE IS",W2(C);" ";V2$
590 PRINT "   OR",W2(C)*180/PI;" DEG"
600 PRINT\PRINT
610 RETURN
```

This program consists of three logical parts. These parts and their functions are:

1) Define the waveforms and create the composite waveform WA (lines 10 to 120).

2) Transform the time-domain waveform into the frequency domain and convert the real and imaginary output of the RFFT into magnitude and phase information (Lines 130 to 160).

3) Compute the magnitude, frequency, and phase of the source waveform's components and print the results (lines 170 to 240 and the subroutine in lines 500 to 610).

Each of these steps is discussed in detail below.

Part one. This section of the program creates the waveform WA and defines its DSI and vertical and horizontal units. The source array WA is created by summing two waveforms of different frequency, phase, and amplitude.

Part two. Here the time-domain information is transformed into magnitude and phase information. First, the RFFT command is executed to define waveforms W1 and W2 as the real and imaginary components. Next, the POLAR command is used to calculate the magnitude and phase information. These are the arrays used in the next part.

Part three. Refer to Fig. 1-12, a graph of the magnitude array created by the POLAR command. The two spikes represent the two original waveforms used to create the source (WA). The spike's horizontal position in the graph defines the frequency of the component, and its amplitude corresponds to the value from the formula for the discrete Fourier transform discussed earlier. This value closely approximates the integral Fourier transform and has vertical units of "VS" (volts seconds). When scaled by a factor of two times the data sampling interval of the magnitude waveform (line 540), it represents the total amplitude and has units of "V" (volts).

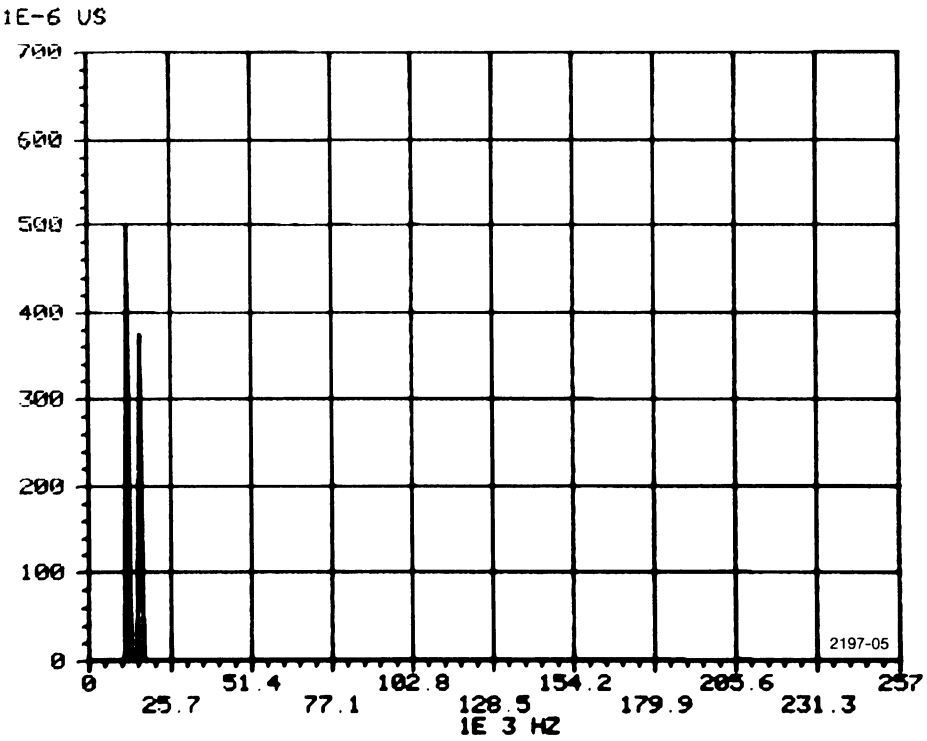


Fig. 1-12. Magnitude output of the POLAR command.

Next, the frequency of the component is determined by multiplying the subscript value by the data sampling interval. The frequency is expressed in units of Hertz. Then, phase is found in the waveform W2 created by POLAR (Fig. 1-13) and expressed in both radians and degrees. Phase is related to delay. A perfect cosine waveform has no delay. A perfect sine wave is 90 degrees delayed. Therefore, the phase information provided by the POLAR command can be used to find the phase shift of any waveform component. finally, the values are printed at the terminal, along with the associated units. The actual output is shown in Fig. 1-14.

Remember, while this program did find the amplitude, frequency, and phase of the original two waveforms, it is operating on ideal waveforms. It merely serves as an example of what the RFFT command can do.

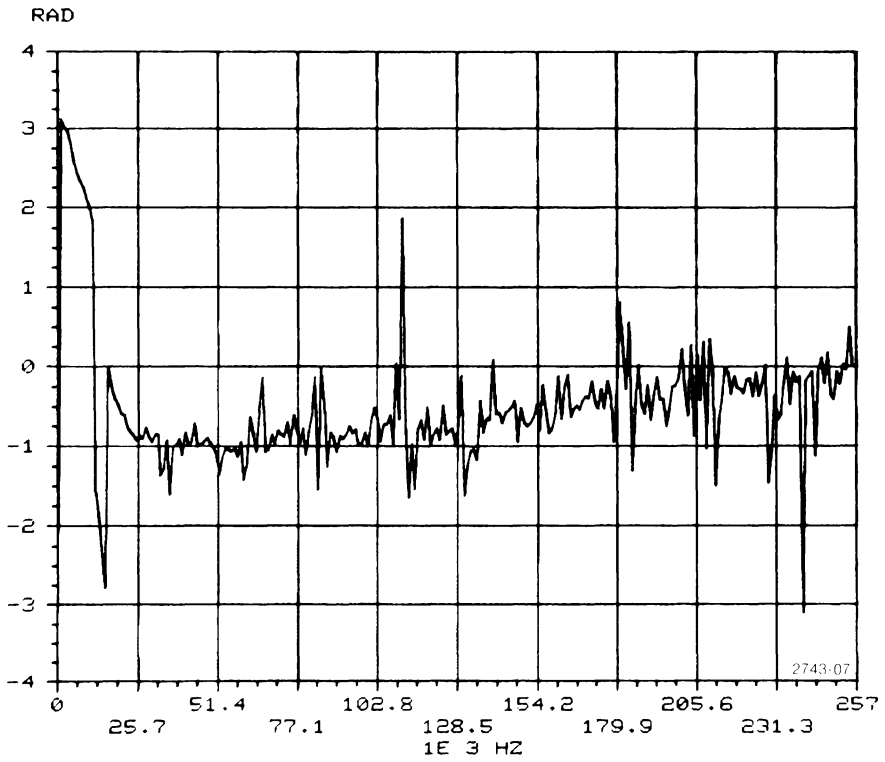


Fig. 1-13. Phase output of the POLAR command.

```
MAGNITUDE IS    1 V
FREQUENCY IS    12000 HZ
PHASE IS        -1.57083 RAD
   OR           -90.0021 DEG


MAGNITUDE IS    .750001 V
FREQUENCY IS    16000 HZ
PHASE IS        -4.84486E-05 RAD
   OR           -2.77590E-03 DEG
                                2743-06
```

Fig. 1-14. Printed results of the program that finds the components of a waveform.

Units and Sampling Interval (DSI) Definitions

The following lists describe how the vertical and horizontal units and data sampling interval are automatically assigned when a waveform is the target for the RFFT command. To simplify the discussion it is assumed that A, B, and C are arrays and that WA, WB, and WC are waveforms. Accordingly, the following conventions are used:

SA: the data sampling interval for waveform WA
SB: the data sampling interval for waveform WB
SC: the data sampling interval for waveform WC

HA$: the horizontal units for waveform WA
HB$: the horizontal units for waveform WB
HC$: the horizontal units for waveform WC

VA$: the vertical units for waveform WA
VB$: the vertical units for waveform WB
VC$: the vertical units for waveform WC

N: the length of the time-domain argument (WA or A)

Direct Transform. Using the above notation, these rules apply for a direct transform when the target for the real components is a waveform (WB) and/or the target for the imaginary components is a waveform (WC). An ampersand (&) indicates that the units strings are concatenated (joined together).

CASE 1: The source argument is a waveform; both target arguments are waveforms. For example:

```
RFFT WA,WB,WC
```

Then:

SB and SC = 1/(N * SA)

HB$ and HC$ = $\begin{cases} \text{"HZ"} & \text{if HA\$ = "S"} \\ \text{"/" \& HA\$} & \text{if HA\$} \neq \text{"S"} \end{cases}$

VB$ and VC$ = VA$ & HA$

CASE 2: The source argument is a waveform; the target for the real components is a waveform, but the target for the imaginary components is an array. For example:

```
RFFT WA,WB,C
```

Then:

SB, HB$, and VB$ are assigned values as in CASE 1.

CASE 3: The source argument is a waveform; the target for the real components is an array, while the target for the imaginary components is a waveform. For example:

```
RFFT WA,B,WC
```

Then:

SC, HC$, and VC$ are assigned values as in CASE 1.

CASE 4: The source argument is an array; both target arguments are waveforms. For example:

```
RFFT A,WB,WC
```

Then:

SB and SC = 1
HB$, HC$, VB$, and VC$ = null string

CASE 5: The source argument is an array; the target for the real components is a waveform, but the target for the imaginary components is an array. For example:

```
RFFT A,WB,C
```

Then:

SB, HB$, and VB$ are assigned values as in CASE 4.

CASE 6: The source argument is an array; the target for the real components is an array, while the target for the imaginary components is a waveform. For example:

```
RFFT A,B,WC
```

Then:

SC, HC$, and VC$ are assigned values as in CASE 4.

Inverse Transform. Using the same notation, these rules apply for an inverse transform when the target (WA) is a waveform. An ampersand (&) indicates that the units strings are concatenated (joined together).

CASE 1: The source of the real components is a waveform. (It does not matter if the source of the imaginary components is a waveform or an array.) For example:

```
RFFT WA,WB,WC,INV
```

or

```
RFFT WA,WB,C,INV
```

Then:

```
SA = 1/(N * SB)
HA$ = {"S"        if HB$ = "HZ"}
      {"/" & HB$  if HB$ ≠ "HZ"}
VA$ = VB$ & HB$
```

CASE 2: The source of the real components is an array. (It does not matter if the source of the imaginary components is a waveform or an array.) For example:

```
RFFT WA,B,WC,INV
```

or

```
RFFT WA,B,C,INV
```

Then:

```
SA =1
HA$ and VA$ = null string
```

RFFT1 (Nonresident)

Examples:

```
   RFFT1 A,INV
1Ø RFFT1 X(Ø:63),TB(Ø:15)
2Ø RFFT1 X,T,DIR
3Ø RFFT1 Y
```

Syntax Form

[line no.] **RFFT1** { floating-point array / floating-point waveform } [, { simple numeric variable / floating-point array }]

[, { **DIR** / **INV** / string expression }]

Descriptive Form:

[line no.] **RFFT1** time domain data or frequency domain data [,sine table] [,direct or inverse transform switch]

Purpose:

The RFFT1 command performs a single-argument fast Fourier transform on real-valued data via a power-of-two algorithm. The single argument format conserves data space. In standard memory systems, this allows longer arrays to be transformed than with RFFT.

Discussion:

Like the RFFT command, the RFFT1 command computes the fast Fourier transform of an input array or waveform. However, unlike the RFFT command, which stores the results of the transform in separate arrays, **the RFFT1 command overwrites the input array with the results of the transform.**

The format of the RFFT1 command is illustrated by the following example:

```
RFFT1 X,TB
```

The first argument (X in the example) initially contains the data to be transformed. It must be a waveform, array, or contiguous subarray of length $N = 2^m$ for some integer $m \geq 4$. (Since a floating-point array in an extended memory (XM) system is limited to 8K elements, the largest N can be in an XM program is $8K = 2^{12} = 4096$.) Data in this array is overwritten with the results of the command.

The second argument (TB in the example) is optional. If it is specified, the transform is table driven. If the argument is a simple numeric variable, that variable is autodimensioned to an array of length N/4 and filled by RFFT1 with sine terms needed by the transform, creating the table. Subsequent executions of RFFT1 do not require regeneration of the table if the source and target array lengths remain unchanged and the table is not altered. (The table can be generated by the CONVL, CORR, and RFFT commands also. For more information about the table, see the RFFT command.) If the argument is an array, it must be of length N/4. RFFT1 checks the first element of the array. If it is the correct value, the array is assumed to be the proper sine-values table. If the first element is an incorrect value, RFFT1 fills the array with the correct sine values. If the argument is not present, the necessary sine terms are generated as needed. This latter method saves memory at the cost of longer execution times.

The optional keyword or string expression specifies whether a direct transform or inverse transform is performed. If the keyword DIR or any string expression other than "INV" is present (or if no keyword or string expression is included as in the preceding example), a direct Fourier transform is performed. If the keyword INV or a string expression that equals "INV" is present, the inverse transform is performed. For example, for a statement such as:

```
RFFT1 X,TB,INV
```

the data in the input array (X) is assumed to contain FFT data and it is then replaced with the results of the IFT operation.

Theory of the FFT/IFT Operation

Like the RFFT command, the RFFT1 command transforms time-domain information into the frequency domain, or inversely, frequency-domain data into the time-domain.

The algorithms used for the direct and inverse transform are the same ones used in the RFFT command. However, since only one argument is used for both source and destination, some decoding of the output is required.

In the case of the direct transform, the time-domain information is transformed into the real and imaginary components of the discrete Fourier coefficients. These two data arrays overwrite the source data. Assuming X is the data array, the storage format following execution of RFFT1 (direct transform) will be:

X(Ø) = DC term
X(1) = Nyquist term
X(2) = Real part of 1st Fourier coefficient
X(3) = Imaginary part of 1st Fourier coefficient
X(4) = Real part of 2nd Fourier coefficient
X(5) = Imaginary part of 2nd Fourier coefficient
.
.
.
X(N-2) = Real part of $(N/2-1)^{th}$ Fourier coefficient
X(N-1) = Imaginary part of $(N/2-1)^{th}$ Fourier coefficient

When an inverse transform is to be performed, the frequency domain data should be stored in the X array in the above manner. This data is then replaced by the time-domain information when the command is executed.

See the RFFT command description for more information about the Fourier transform.

Units and Data Sampling Interval (DSI) Definitions

The following list describes how the vertical and horizontal units and data sampling interval are automatically assigned when a waveform is the source/target for the RFFT1 command. To simplify this discussion, it is assumed that WA is a waveform. Accordingly, the following conventions are used:

SA: the sampling interval for waveform WA

HA$: the horizontal units for waveform WA

VA$: the vertical units for waveform WA

Also, an ampersand (&) indicates that the units strings are concatenated (joined together).

Direct Transform. These rules apply when the source/target for the direct transform is a waveform. For example:

```
RFFT1 WA
```

Then:

SA is changed to: 1/(N * SA)
VA$ is changed to: VA$ & HA$
HA$ is changed to: {"HZ" if HA$ = "S"} {"/" & HA$ if HA$ ≠ "S"}

Inverse Transform. These rules apply when the source/target for the inverse transform is a waveform. For example:

```
RFFT1 WA,INV
```

Then:

SA is changed to: 1/(N * SA)
VA$ is changed to: VA$ & HA$
HA$ is changed to: {"S" if HA$ = "HZ"} {"/" & HA$ if HA$ ≠ "HZ"}

SECTION 2

GLOSSARY

aliasing. A phenomenon whereby high-frequency spectral components appear to be low-frequency components in an FFT spectrum. Aliasing occurs when the real-time input signal is sampled at too low a sampling rate. To avoid aliasing, the input signal must be sampled at a rate at least twice that of the highest frequency component of significance that is present in the input signal.

analog signal. A signal that is continuous in time (or any other appropriate independent variable) and that exhibits a continuous range of analog values.

analog-to-digital converter. A circuit or device that converts an analog signal into a corresponding digital representation of that signal.

autocorrelation. The process of correlating a signal with itself. (See "correlation" and "cross-correlation.")

complex conjugation. The process of negating the imaginary part of a complex number to obtain the complex conjugate. (The complex conjugate of a+jb is a-jb; the complex conjugate of 3-j5 is 3+j5.)

complex number. A number having the form a+jb, where a is the real part and jb is the imaginary part. ($j = \sqrt{-1}$.)

convolution. An operation mathematically similar to correlation. Like correlation, convolution can be thought of as successively shifting, multiplying, and integrating the two arrays (or waveforms) to be convolved. However, in the case of convolution, one of the waveforms is reversed in time before performing the shifting-multiplication-integration process. Convolution can be performed by computing the FFT of each signal to be convolved, multiplying these two FFT results, and then computing the IFT of the product.

correlation. A mathematical operation that indicates the similarity between two waveforms as a function of the delay (time-shift). Correlation can be thought of as successively shifting (by some horizontal increment), multiplying, and integrating the two signals to be correlated. From a mathematical standpoint, correlation can be achieved by computing the FFT

of each signal to be correlated, then forming a complex-conjugate product from the FFT results, and finally taking the IFT of the product.

cross-correlation. The process of correlating two different waveforms. (See "correlation" and "autocorrelation.")

data sampling interval (DSI). The time between acquisition of two successive data samples in a digitized waveform.

differentiation. An important mathematical operation that forms the basis of differential calculus. From an intuitive standpoint, the derivative at a given point of a function, array, or waveform corresponds to determining the slope of the curve at that point.

digitize. To perform an analog-to-digital conversion upon a signal, usually representing some physical measurement.

fast Fourier transform (FFT). A computer algorithm for converting a signal from the time domain to the frequency domain.

frequency domain. Refers to a way of representing a signal such that its amplitude is expressed as a function of frequency.

frequency response. The response of a circuit, device, or system when different frequencies are applied to it.

horizontal scale factor (HSF). The scale that applies to the time scale (horizontal axis) of the waveform acquired via an oscilloscope or signal processing system. On an oscilloscope, the HSF is usually expressed in terms of time per graticule division. Some HSFs are 1 sec/div, 50 ms/div, and 500 us/div.

imaginary number. A number having the form jb, where b is a real number and $j = \sqrt{-1}$.

impulse response. The response of a circuit, device, or system when an impulse is applied to it. (Theoretically, an impulse is a spike with zero width, infinite amplitude, and unity area. In a practical sense though, an impulse has finite amplitude -- great enough to elicit a response but not enough to damage the system -- and non-zero width. The width must be much less than the expected response time of the system.)

integration. An important mathematical operation that forms the basis of integral calculus. From an intuitive standpoint, the integral of a function, array, or waveform corresponds to determining the area under the curve or the energy contained in a pulse. Integration and differentiation are inverse processes.

inverse Fourier transform (IFT). A mathematical operation for converting a signal from the frequency domain to the time domain. The FFT and IFT are inverse operations. That is, the IFT of the FFT of a signal is equivalent to the signal itself.

Nyquist frequency. The highest frequency (f_n) that can be digitally represented for a given sampling rate (f_s) or sampling interval (Δt). $f_n = f_s/2 = 1/(2\Delta t)$

Nyquist Sampling Theorem. This theorem states that the sampling rate of a waveform digitizer must be twice that of the highest frequency component in the waveform being sampled. When this condition is not fulfilled, aliases (false frequency components) appear in the digitized waveform.

phase. 1) The angular relationship between current and voltage in alternating current circuits. 2) The angular displacement of a sinusoid from the phase Ø position. (See Fig. 2-1.) Phase is usually expressed in radians.

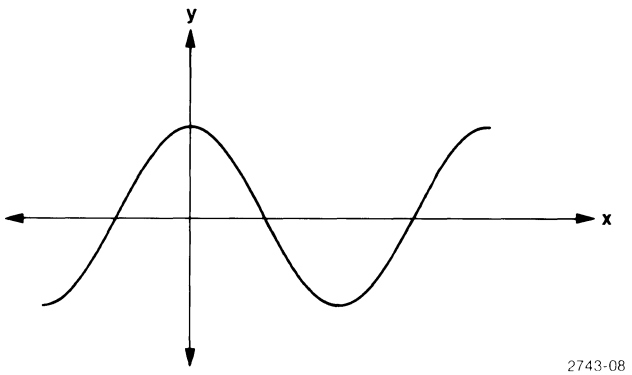


Fig. 2-1. A sinusoid with zero phase (symmetric about the y-axis).

polar form. An output format of the Fourier transform in which the spectral components are expressed in terms of magnitude and phase data. The polar form of the result of the FFT is derived from the rectangular form by

applying the formulas $M = \sqrt{R^2 + I^2}$ and $\theta = \arctan (I/R)$ where R and I are the reals and imaginaries, and M and θ are the magnitude and phase values, respectively. (See "rectangular form.")

real number. Any rational or irrational number. (A rational number can be expressed as the quotient of two integers.)

real-time process. A process in which, on the average, the computing associated with each sampling interval can be completed in a period less than or equal to the sampling interval.

rectangular form. An output format of the Fourier transform in which the spectral components are expressed as real and imaginary numbers.

root-mean-square (RMS) value. The effective value of a varying or alternating voltage. It is equivalent to that value which would produce the same power loss as if a continuous voltage of that value were applied to a pure resistance. (In sine-wave voltages, the RMS voltage is equal to 0.707107 times the peak voltage.)

signal averaging. The process of acquiring a given number of whole-waveform samples, summing them, and dividing by the number of acquired waveforms. Signal averaging improves the signal-to-noise ratio.

spectral components. Refers to significant amplitudes existing at certain frequencies within the spectrum.

spectrum. A graph of signal amplitude (or energy) versus frequency.

time domain. A way of representing a signal such that the signal amplitude is expressed as a function of time.

vertical scale factor (VSF). The scale that applies to the vertical axis of data acquired via an oscilloscope or signal processing system. On an oscilloscope, the VSF is usually expressed in terms of volts per graticule division. Some VSFs are 5 volts/div, 20 mV/div, and 10 mV/div.

window. Refers to the total period during which whole-waveform data is being acquired, or in which processed data is being displayed. Because the data acquisition process amounts to multiplying a waveform train by a rectangular window, data acquisition is sometimes referred to as "windowing a waveform."

YOUR COMMENTS COUNT

The Manual Writers at Tektronix, Inc. are interested in what you think about this manual, how you use it, and changes you might like to see in future manuals. Any queries regarding this manual will be answered personally.

What did you find that was:

interesting? _____

_____

frustrating? _____

_____

helpful? _____

_____

confusing? _____

_____

Is there anything you would like to see added to or deleted from this manual? _____

_____

_____

_____

What is your major application area for this product? _____

_____

_____

_____

Have you found any interesting applications, operating hints, or software routines which you would like to share with us? _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

* * * * *

Name: _____ Position: _____

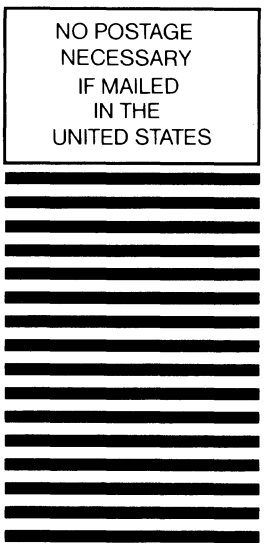
Company: _____ Department: _____

Street: _____

City: _____ State: _____ Zip: _____

Fold on dotted lines and tape.
Postage will be paid by Tektronix, Inc. if mailed in U.S.A.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

TEKTRONIX, INC.
P.O. Box 500
Beaverton, Oregon 97005
Attn: Del. Sta. 94-384